aws

개발자 가이드

Amazon DocumentDB

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon DocumentDB: 개발자 가이드

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

Amazon DocumentDB란 무엇인가 1
개요 1
클러스터 3
인스턴스 4
리전 및 가용 영역 7
리전 7
가용 영역 7
가격 책정 10
무료 평가판 10
모니터링 10
인터페이스 11
AWS Management Console 11
AWS CLI 11
MongoDB 드라이버 11
다음 단계 12
작동 방식 12
Amazon DocumentDB 버전 13
Amazon DocumentDB 엔드포인트 15
TLS 지원 18
Amazon DocumentDB 스토리지 18
Amazon DocumentDB 복제 19
Amazon DocumentDB 신뢰성 19
읽기 기본 설정 옵션 20
TTL 삭제 24
청구 가능 리소스 25
문서 데이터베이스란? 28
사용 사례 28
문서 이해 29
문서 작업 35
시작 안내서 47
사전 조건 47
1단계: 클러스터 생성 49
2단계: 클러스터에 연결 51
3단계: 데이터 삽입 및 쿼리 52

4단계: 탐색 54
사용 빠른 시작 CloudFormation 55
사전 조건 55
Amazon DocumentDB CloudFormation 스택 시작 56
Amazon DocumentDB 클러스터 액세스 61
종료 방지 및 삭제 방지 61
마이그레이션 및 업그레이드 63
빠른 시작 설명서 63
DMS 소스 준비 64
DMS 설정 64
DocumentDB 압축 활성화 64
복제 작업 생성 64
진행 상황 모니터링 65
추가 정보 65
마이그레이션 런북 66
호환성 66
워크로드 검색 69
새 대상 클러스터 생성 72
인덱스 마이그레이션 72
사용자 마이그레이션 73
데이터 마이그레이션 75
모니터링 85
검증 87
Couchbase Server에서 마이그레이션 93
소개 93
Amazon DocumentDB와 비교 93
Discovery 96
계획 100
마이그레이션 104
검증 113
엔진 버전 지원 날짜 116
Amazon DocumentDB 메이저 버전 릴리스 일정 116
Amazon DocumentDB 마이너 버전 릴리스 일정 117
메이저 엔진 버전 업그레이드 117
지원되는 업그레이드 경로 118
사전 조건 119

모범 사례 121
Amazon DocumentDB 8.0으로 업그레이드한 후 변경되는 사항 124
3.6 또는 4.0에서 업그레이드된 클러스터에 대한 업그레이드 후 고려 사항 125
업그레이드 수행 125
문제 해결 128
마이너 엔진 버전 업그레이드 128
마이너 버전 업그레이드를 수행하기 전 128
마이너 버전 업그레이드 수행 129
마이너 버전 업그레이드 확인 130
를 사용하여 업그레이드 AWS DMS 130
1단계: 변경 스트림 활성화 131
2단계: 변경 스트림 보존 기간 수정 131
3단계: 인덱스 마이그레이션 131
4단계: AWS DMS 복제 인스턴스 생성 133
5단계: AWS DMS 소스 엔드포인트 생성 134
6단계: AWS DMS 대상 엔드포인트 생성 137
7단계: 마이그레이션 작업 생성 및 실행 139
8단계: 애플리케이션 엔드포인트를 대상 Amazon DocumentDB 클러스터로 변경 141
LTS(장기 지원) 릴리스 사용 141
현재 LTS 릴리스 141
LTS를 사용해야 하는 사용자 141
새 LTS 클러스터를 생성하는 방법 142
엔진 버전 3.6.0 또는 4.0.0에서 5.0.0 LTS 클러스터로 업그레이드하는 방법 142
기존 엔진 버전 5.0.0 클러스터가 LTS에 있는지 확인하는 방법 143
추가 지원 143
추가 지원 개요 144
추가 지원 요금 145
책임 145
기능 및 구성 146
엔진 상태 146
클러스터 유형 146
특성 147
인스턴스 유형 148
AWS 통합 149
Choosing Regions and Availability Zones 150
리전 가용성 151

OpenSearch 통합 153
Amazon OpenSearch Service의 대상입니다. 153
제한 사항 156
개발 158
DocumentDB에 연결 158
프로그래밍 방식으로 연결 159
Amazon VCP 외부에서 연결 188
복제본 세트로 연결 189
Studio 3T를 사용하여 연결 194
DataGrip을 사용하여 연결 203
Amazon EC2를 사용하여 연결 208
JDBC 드라이버를 사용하여 연결 242
ODBC 드라이버를 사용하여 연결 257
Amazon DocumentDB를 사용한 프로그램 271
DocumentDB Java 프로그래밍 설명서 271
JSON 스키마 검증 사용 301
쿼리 311
문서 쿼리 312
쿼리 계획 314
결과의 설명 317
쿼리 플래너 v2 322
쿼리 플래너 v3 336
지리 공간 데이터 341
부분 인덱스 347
텍스트 검색 354
인덱스 362
인덱스 관리 362
인덱스 유형 372
인덱스 속성 381
변경 스트림 사용 399
지원되는 연산자 400
확장된 이벤트 400
결제 401
제한 사항 401
변경 스트림 활성화 402
Python에서 변경 스트림 사용 404

전체 문서 조회 ... 407
$changeStreamSplitLargeEvent를 사용하여 대규모 이벤트를 조각으로 분할 ... 407
변경 스트림 재개 ... 408
startAtOperationTime로 재개 ... 410
postBatchResumeToken로 재개 ... 411
트랜잭션 ... 413
로그 보존 기간 수정 ... 413
보조 인스턴스의 스트림 변경 ... 416
트랜잭션 ... 417
요구 사항 ... 418
모범 사례 ... 418
제한 사항 ... 419
모니터링 및 진단 ... 419
트랜잭션 격리 수준 ... 420
사용 사례 ... 420
지원되는 명령 ... 458
지원되지 않는 기능 ... 458
세션 ... 459
트랜잭션 오류 ... 460
재시도 가능한 쓰기 ... 461
요구 사항 ... 461
지원되는 연산자 ... 462
제한 사항 ... 462
활성화 ... 463
작동 방식 ... 463
오류 처리 ... 463
마이그레이션 ... 464
데이터 정렬 사용 ... 464
제한 사항 ... 465
뷰 사용 ... 465
모범 사례 ... 466
집계자 연산자 호환성 ... 466
인덱스 및 뷰 활용 ... 467
생성형 AI ... 467
SageMaker Canvas ... 467
벡터 검색 ... 470

변경 스트림과 AWS Lambda 함께 사용 481
제한 사항 481
MongoDB 호환성 482
MongoDB 8.0 호환성 482
Amazon DocumentDB 8.0의 새로운 기능 482
Amazon DocumentDB로 시작하기 483
Amazon DocumentDB 5.0 또는 8.0으로 업그레이드 또는 마이그레이션 483
기능적 차이 484
MongoDB 5.0 호환성 484
Amazon DocumentDB 5.0의 새로운 기능 485
Amazon DocumentDB 5.0.1의 새로운 기능 485
Amazon DocumentDB 5.0 시작하기 483
Amazon DocumentDB 5.0으로 업그레이드 또는 마이그레이션 483
기능적 차이 484
MongoDB 4.0 호환성 488
Amazon DocumentDB 4.0 기능 488
Amazon DocumentDB 4.0 시작하기 490
Amazon DocumentDB 4.0으로 업그레이드 또는 마이그레이션 490
기능적 차이 490
MongoDB와 기능적 차이 491
Amazon DocumentDB의 기능적 이점 491
업데이트된 기능적 차이 493
MongoDB와 기능적 차이 496
지원되는 MongoDB API, 작업 및 데이터 형식 504
데이터베이스 명령 505
쿼리 및 프로젝션 연산자 515
업데이트 연산자 519
지리 공간 521
커서 메서드 522
집계 파이프라인 연산자 524
데이터 형식 542
인덱스 및 인덱스 속성 543
집계 파이프라인 연산자 545
지리 공간 1163
쿼리 및 프로젝션 연산자 1187
업데이트 연산자 1254

클러스터 및 인스턴스 관리 1307
운영 작업 개요 1307
Amazon DocumentDB 클러스터에 복제본 추가 1308
클러스터 및 인스턴스 설명 1308
클러스터 스냅샷 생성 1311
스냅샷에서 복원 1312
클러스터에서 인스턴스 제거 1313
클러스터 삭제 1313
글로벌 클러스터 1314
글로벌 클러스터란 무엇입니까? 1314
글로벌 클러스터는 어떻게 유용할까요? 1314
글로벌 클러스터의 현재 한계는 무엇입니까? 1315
빠른 시작 설명서 1315
글로벌 클러스터 관리 1327
글로벌 클러스터 연결 1333
글로벌 클러스터 모니터링 1334
재해 복구 1335
클러스터 관리 1358
클러스터 이해 1359
클러스터 설정 1361
클러스터 스토리지 구성 1364
클러스터 상태 확인 1367
클러스터 수명 주기 1368
클러스터 조정 1409
클러스터에 대한 볼륨 복제 1412
클러스터의 내결함성 이해 1424
인스턴스 관리 1425
인스턴스 상태 확인 1426
인스턴스 수명 주기 1426
인스턴스 클래스 관리 1449
NVMe 지원 인스턴스 1461
서브넷 그룹 관리 1463
서브넷 그룹 생성 1464
서브넷 그룹 설명 1470
서브넷 그룹 수정 1472
서브넷 그룹 삭제 1475

고가용성 및 복제 1477
읽기 확장 1477
높은 가용성 1477
복제본 추가 1479
장애 조치 1479
복제 지연 1484
이벤트 관리 1485
이벤트 범주 보기 1485
Amazon DocumentDB 이벤트 보기 1488
클러스터 파라미터 그룹 관리 1490
클러스터 파라미터 그룹 설명 1491
클러스터 파라미터 그룹 생성 1498
클러스터 파라미터 그룹 수정 1501
사용자 지정된 클러스터 파라미터 그룹을 사용하도록 클러스터 수정 1505
클러스터 파라미터 그룹 복사 1507
클러스터 파라미터 그룹 재설정 1509
클러스터 파라미터 그룹 삭제 1512
클러스터 파라미터 참조 1515
엔드포인트 이해 1535
클러스터 엔드포인트 찾기 1535
인스턴스의 엔드포인트 찾기 1538
엔드포인트에 연결 1541
Amazon DocumentDB ARN 이해 1542
ARN 생성 1543
ARN 찾기 1546
리소스에 태그 지정 1549
리소스 태그 개요 1549
태그 제약 조건 1550
태그 추가 또는 업데이트 1550
태그 나열 1552
태그 제거 1553
Amazon DocumentDB 유지 관리 1555
Amazon DocumentDB에 대한 유지 관리 작업 1556
버전 번호 지정 1556
유지 관리 기간 관리 1558
엔진 패치 알림 1560

보류 중인 유지 관리 보기 1562
엔진 업데이트 1565
마이너 버전 1568
운영 체제 업데이트 1568
사용자 시작 업데이트 1572
글로벌 클러스터 1573
서비스 연결 역할 이해 1574
서비스 연결 역할 권한 1574
서비스 연결 역할 만들기 1576
서비스 연결 역할 수정 1576
서비스 연결 역할 삭제 1576
Amazon DocumentDB 서비스 연결 역할에 지원되는 리전 1577
성능 최적화 1578
일반적인 성능 문제 1578
고성능 CPU 1578
연결 문제 1580
쿼리 실행 속도가 느림 1587
안티 패턴 1591
3개를 초과하는 복합 인덱스 안티 패턴 속성 1591
장기 실행 쿼리 1593
낮은 사용 및 중복 인덱스 1595
배열이 큰 다중 키 인덱스 1598
성능 개선 팁 1599
1. 집계 파이프라인에서 $match를 첫 번째 단계로 사용 1599
2. 집계 파이프라인에서 $project를 사용하여 파이프라인 데이터 크기 최소화 1599
3. 스토리지 비용, I/O 비용 및 쿼리 성능 향상을 위한 문서 압축 활성화 1600
4. 최적의 쿼리 성능을 위한 인덱스 활용 1600
5. 쿼리 패턴을 기반으로 데이터 모델 최적화 1601
쿼리 계획 분석 1603
전체 타이밍 1605
실행 단계 1606
계획 실행 통계 설명의 docsExamined 이해 1607
인덱스 전용 스캔 1615
Amazon DocumentDB가 인덱스 전용 스캔을 사용하는 경우 1615
적용 대상 쿼리에 대한 요구 사항 1618
제한 사항 1618

인덱스 전용 스캔 확인 1619
모범 사례 1620
캐시 필터 계획 1620
지원되는 명령 1621
지원되는 쿼리 셰이프 1623
필터 설정, 나열 및 지우기 1623
필터가 적용되었는지 확인 1625
예제 1626
힌트 및 인덱스 변경 사항이 있는 동작 1628
모범 사례 1629
기본 운영 지침 1630
인스턴스 크기 조정 1631
인덱스 작업 1632
보안 모범 사례 1633
비용 최적화 1633
지표를 통해 성능 문제 식별 1634
TTL 및 시계열 워크로드 1638
마이그레이션 1639
클러스터 파라미터 그룹 작업 1639
집계 파이프라인 쿼리 1639
`batchInsert` 및 `batchUpdate` 1639
인덱스 관리 1639
Amazon DocumentDB 인덱스 생성 1640
Amazon DocumentDB 인덱스 유지 관리 1645
문서 압축 관리 1647
문서 압축 관리 1648
문서 압축 모니터링 1650
사전 기반 압축 관리 1651
성능 고려 사항 1651
사전 기반 압축 활성화 1652
시작하기 1653
모니터링 1653
보안 1655
를 사용한 암호 관리 AWS Secrets Manager 1656
Secrets Manager와 Amazon DocumentDB 통합에 대한 제한 사항 1656
를 사용한 기본 사용자 암호 관리 개요 AWS Secrets Manager 1657

에서 기본 사용자 암호의 Amazon DocumentDB 관리 적용 AWS Secrets Manager 1657
Secrets Manager를 사용하여 클러스터의 기본 사용자 암호 관리 1658
데이터 보호 1658
클라이언트 측 필드 레벨 암호화 1659
저장 데이터 암호화 1666
전송 중 데이터 암호화 1674
키 관리 1684
ID 및 액세스 관리 1685
대상 1685
ID를 통한 인증 1685
정책을 사용하여 액세스 관리 1687
Amazon DocumentDB에서 IAM을 사용하는 방법 1688
ID 기반 정책 예시 1694
문제 해결 1697
Amazon DocumentDB 리소스에 대한 액세스 권한 관리 1699
ID 기반 정책(IAM 정책) 사용 1704
AWS Amazon DocumentDB에 대한 관리형 정책 1708
Amazon DocumentDB API 권한 참조 1726
교차 서비스 혼동된 대리인 방지 1735
IAM 자격 증명을 사용한 인증 1736
IAM 사용자 및 역할 시작하기 1737
컴퓨팅 유형 구성 1740
인증 요청 모니터링 1740
IAM 인증 사용 1741
IAM을 지원하는 드라이버 1741
IAM 자격 증명 인증 FAQ 1741
SCRAM-SHA-256을 사용한 인증 1742
사용자 생성 1743
사용자 업데이트 1744
인증 1744
Amazon DocumentDB 사용자 관리 1745
기본 및 `serviceadmin` 사용자 1745
추가 사용자 생성 1746
자동으로 암호 교체 1748
역할 기반 액세스 제어 1748
RBAC 개념 1749

RBAC 내장된 역할 시작하기 1751
RBAC 사용자 정의 역할 시작하기 1754
Amazon DocumentDB에 사용자로 연결 1759
공통 명령 1760
기능적 차이 1765
한도 1766
역할 기반 액세스 제어를 사용한 데이터베이스 액세스 1766
인증서 업데이트 1775
애플리케이션 및 Amazon DocumentDB 클러스터 업데이트 1775
자동 서버 인증서 교체 1779
문제 해결 1780
FAQ 1781
규정 준수 확인 1787
복원력 1788
인프라 보안 1789
VPC 엔드포인트(AWS PrivateLink) 1790
VPC 엔드포인트에 대한 고려 사항 1791
리전 가용성 1791
Amazon DocumentDB API에 대한 인터페이스 VPC 엔드포인트 생성 1792
Amazon DocumentDB API에 대한 VPC 엔드포인트 정책 생성 1792
보안 모범 사례 1794
이벤트 감사 1795
지원되는 이벤트 1796
감사 활성화 1800
감사 비활성화 1806
이벤트에 액세스 1809
DML 이벤트 필터링 1809
Amazon VPC 및 Amazon DocumentDB 1814
VPC에서의 DocumentDB 클러스터 1815
VPC의 클러스터에 액세스 1826
클러스터에 대한 IPv4 전용 VPC 생성 1828
클러스터에 대한 듀얼 스택 VPC 생성 1835
백업 및 복원 1845
백업 및 복원: 개념 1846
백업 스토리지 사용량 파악 1847
데이터 덤프, 복원, 가져오기 및 내보내기 1849

mongodump ... 1849
mongorestore ... 1850
mongoexport ... 1850
mongoimport ... 1851
자습서 ... 1852
클러스터 스냅샷 고려 사항 ... 1854
백업 스토리지 ... 1855
백업 기간 ... 1855
백업 보관 기간 ... 1857
클러스터 스냅샷 암호화 복사 ... 1857
자동 스냅샷과 수동 스냅샷 비교 ... 1858
수동 클러스터 스냅샷 생성 ... 1860
클러스터 스냅샷 복사 ... 1863
공유 스냅샷 복사 ... 1863
에서 스냅샷 복사 AWS 리전 ... 1864
제한 사항 ... 1864
암호화 처리 ... 1864
파라미터 그룹 고려 사항 ... 1865
클러스터 스냅샷 복사 ... 1865
클러스터 스냅샷 공유 ... 1874
암호화된 스냅샷 공유 ... 1875
스냅샷 공유 ... 1878
클러스터 스냅샷에서 복원 ... 1881
특정 시점으로 복원 ... 1889
클러스터 스냅샷 삭제 ... 1895
모니터링 및 로깅 ... 1897
클러스터 상태 모니터링 ... 1898
클러스터 상태 값 ... 1898
클러스터 상태 모니터링 ... 1900
인스턴스 상태 모니터링 ... 1902
인스턴스 상태 값 ... 1903
AWS Management Console 또는를 사용하여 인스턴스 상태 모니터링 AWS CLI ... 1905
인스턴스 상태 값 ... 1907
를 사용하여 인스턴스 상태 모니터링 AWS Management Console ... 1907
Amazon DocumentDB 권장 사항 보기 ... 1909
이벤트 구독 ... 1911

이벤트 구독 1912
구독 관리 1914
범주 및 메시지 1917
CloudWatch를 사용하여 Amazon DocumentDB 모니터링 1920
Amazon DocumentDB 지표 1920
CloudWatch 데이터 보기 1935
Amazon DocumentDB 차원 1941
opcounter 지표 모니터링 1941
데이터베이스 연결 모니터링 1941
CloudTrail로 Amazon DocumentDB API 호출 로깅 1941
CloudTrail의 Amazon DocumentDB 정보 1942
작업 프로파일링 1943
지원되는 연산자 1944
제한 사항 1944
프로파일러 활성화 1945
프로파일러 비활성화 1949
프로파일러 로그 내보내기 비활성화 1950
프로파일러 로그 액세스 1952
공통 쿼리 1953
성능 개선 도우미로 모니터링 1953
성능 개선 도우미 개념 1954
성능 개선 도우미 활성화 및 비활성화 1958
성능 개선 도우미에 대한 액세스 정책 구성 1960
성능 개선 도우미 대시보드를 사용한 지표 분석 1964
성능 개선 도우미 API를 사용하여 지표 검색 1983
성능 개선 도우미를 위한 Amazon CloudWatch 지표 1997
카운터 지표에 대한 성능 개선 도우미 1999
서버리스 2001
서버리스 사용 사례 2001
기존 프로비저닝된 워크로드에 Amazon DocumentDB Serverless 사용 2003
Amazon DocumentDB Serverless의 장점 2003
Serverless의 작동 방식 2004
개요 2004
Amazon DocumentDB 클러스터에 대한 구성 2006
Amazon DocumentDB Serverless 규모 조정 용량 2006
Amazon DocumentDB Serverless 규모 조정 2008

유휴 상태(0.5DCU) 2009
Serverless 요구 사항 및 제한 사항 2010
기능 요구 사항 2010
Serverless를 사용하는 클러스터 생성 2011
Amazon DocumentDB Serverless 클러스터 생성 2011
Amazon DocumentDB Serverless 인스턴스 추가 2014
서버리스로 마이그레이션 2015
기존 DocumentDB 클러스터를 서버리스로 마이그레이션 2015
MongoDB에서 DocumentDB Serverless로 마이그레이션 2020
Serverless 관리 2020
클러스터의 규모 조정 용량 범위 구성 보기 및 수정 2020
서버리스 리더의 승격 계층 보기 및 수정 2023
서버리스 인스턴스 제한 2025
서버리스 규모 조정 구성 2028
DocumentDB Serverless 클러스터의 규모 조정 용량 범위 선택 2028
DocumentDB Serverless 클러스터에 대한 MinCapacity 설정 선택 2028
DocumentDB Serverless 클러스터에 대한 MaxCapacity 설정 선택 2030
메모리 부족 오류 방지 2032
서버리스 인스턴스가 스케일 다운되지 않는 이유는 무엇인가요? 2033
Serverless 모니터링 2034
메모리 부족: 호환되지 않는 파라미터 상태 2034
DocumentDB Serverless에 대한 Amazon CloudWatch 지표 2034
성능 개선 도우미로 DocumentDB Serverless 성능 모니터링 2038
엘라스틱 클러스터 2040
탄력적 클러스터 사용 사례 2041
사용자 프로필 2041
콘텐츠 관리 및 이력 기록 2041
탄력적 클러스터의 장점 2041
AWS 서비스 통합 2041
리전 및 버전 사용 가능 여부 2042
탄력적 클러스터의 리전 가용성 2042
버전 가용성 2043
제한 사항 2043
탄력적 클러스터 관리 2043
쿼리 및 쓰기 작업 2044
컬렉션 및 인덱스 관리 2044

관리 및 진단 2044
옵트인 기능 2045
작동 방식 2045
Amazon DocumentDB 탄력적 클러스터 샤딩 2045
탄력적 클러스터 마이그레이션 2049
탄력적 클러스터 규모 조정 2049
탄력적 클러스터 안정성 2049
탄력적 클러스터 스토리지 및 가용성 2049
Amazon DocumentDB 4.0과 탄력적 클러스터 간의 기능적 차이 2049
시작하기 2051
사전 조건 2052
1단계: 엘라스틱 클러스터 생성 2053
2단계: 탄력적 클러스터에 연결 2059
3단계: 컬렉션 샤드, 데이터 삽입 및 쿼리 2060
4단계: 탐색 2063
모범 사례 2063
샤드 키 선택 2064
연결 관리 2064
샤딩되지 않은 컬렉션 2064
탄력적 클러스터 크기 조정 2065
탄력적 클러스터 모니터링 2065
탄력적 클러스터 관리 2066
엘라스틱 클러스터 구성 수정 2066
엘라스틱 클러스터 모니터링 2070
엘라스틱 클러스터 삭제 2073
엘라스틱 클러스터 스냅샷 관리 2076
탄력적 클러스터 중지 및 시작 2089
탄력적 클러스터 유지 관리 2094
유휴 시(저장된) 데이터 암호화 2101
Amazon DocumentDB 탄력적 클러스터가에서 권한 부여를 사용하는 방법 AWS KMS 2103
고객 관리형 키 생성 2104
Amazon DocumentDB 탄력적 클러스터에 대한 암호화 키 모니터링 2105
자세히 알아보기 2111
서비스 연결 역할 2111
탄력적 클러스터에 대한 서비스 연결 역할 권한 2111
할당량 2115

지원되는 인스턴스 유형 2115
지원되는 리전: 2117
리전별 할당량 2119
집계 할당량 2121
클러스터 할당량 2121
인스턴스 할당량 2123
명명 제약 조건 2130
TTL 제약 조건 2132
탄력적 클러스터 할당량 2132
탄력적 클러스터 샤드 할당량 2133
샤드당 탄력적 클러스터 CPU, 메모리, 연결 및 커서 할당량 2134
문제 해결 2135
연결 문제 2135
Amazon DocumentDB 엔드포인트에 연결할 수 없습니다. 2135
Amazon DocumentDB 인스턴스 연결 테스트 2140
유효하지 않은 엔드포인트에 연결 2141
연결 수에 영향을 미치는 드라이버 구성 2142
인덱스 2142
인덱스 빌드 실패 2142
백그라운드 인덱스 빌드 지연 문제 및 실패 2143
인덱스 팽창 2143
성능 및 리소스 사용률 2144
통계 보기, 삽입, 업데이트, 삭제 2145
캐시 성능 분석 2146
장기 실행 또는 차단된 쿼리 찾기 및 종료 2147
쿼리 계획 보기 및 쿼리 최적화 2149
엘라스틱 클러스터에서 쿼리 계획을 확인하려면 어떻게 해야 하나요? 2151
인스턴스에서 실행 중인 모든 작업 나열 2153
쿼리가 진행 중인 시기 파악 2156
시스템이 갑자기 느리게 실행되는 이유 확인 2158
CPU 사용률이 높은 원인 알아보기 2160
인스턴스에서 열린 커서를 찾으세요. 2161
현재 Amazon DocumentDB 엔진 버전 보기 2161
인덱스 사용량을 분석하고 사용하지 않는 인덱스를 식별 2162
누락된 인덱스를 식별 2164
데이터베이스 수집 팽창을 확인하려면 어떻게 해야 하나요? 2165

유용한 쿼리 요약 2166
가비지 수집 2168
Amazon DocumentDB의 가비지 수집 이해 2168
폐영역 회수 프로세스 2169
스토리지 아키텍처 및 확장 스토리지 2169
폐영역 회수 모니터링 2170
collStats 출력 예제 2172
자주 묻는 질문(FAQ) 2174
오류 코드 2177
릴리스 노트 2208
Amazon DocumentDB 엔진 버전 릴리스 정보 2208
일반 Amazon DocumentDB 릴리스 정보 2265
Amazon DocumentDB 탄력적 클러스터 릴리스 정보 2271
문서 기록 2275
..... mmcclxxxvi

Amazon DocumentDB(MongoDB 호환)란 무엇인가

Amazon DocumentDB(MongoDB와 호환됨)는 빠르고 안정적이며 완전 관리형 데이터베이스 서비스입니다. Amazon DocumentDB를 사용하면 클라우드에서 MongoDB 호환 데이터베이스를 쉽게 설치, 운영 및 규모를 조정할 수 있습니다. Amazon DocumentDB를 사용하면 MongoDB에서 사용하는 것과 동일한 애플리케이션 코드를 실행하고 동일한 드라이버와 도구를 사용할 수 있습니다.

Amazon DocumentDB를 사용하기 전에 작동 방식에 설명된 개념과 기능을 검토해야 합니다. 그 이후에는 시작 안내서의 단계를 완료합니다.

주제

- Amazon DocumentDB 개요
- 클러스터
- 인스턴스
- 리전 및 가용 영역
- Amazon DocumentDB 요금
- 모니터링
- 인터페이스
- 다음 단계
- Amazon DocumentDB: 작동 방식
- 문서 데이터베이스란?

Amazon DocumentDB 개요

다음은 Amazon DocumentDB의 일부 고급 기능입니다.

- Amazon DocumentDB는 인스턴스 기반 클러스터와 탄력적 클러스터라는 두 가지 유형의 클러스터를 지원합니다. 탄력적 클러스터는 초당 수백만 읽기/쓰기 및 페타바이트의 스토리지 용량을 갖춘 워크로드를 지원합니다. 탄력적 클러스터에 대한 자세한 내용은 Amazon DocumentDB 탄력적 클러스터 사용을 참조하세요. 아래 내용은 Amazon DocumentDB 인스턴스 기반 클러스터를 참조합니다.
- Amazon DocumentDB는 데이터베이스 스토리지가 증가함에 따라 스토리지 볼륨의 크기를 자동으로 증가시킵니다. 스토리지 볼륨은 10GB 단위로 증가하며 엔진 버전 8.0 이상의 경우 최대 256TiB(이전 엔진 버전의 경우 128TiB)까지 증가합니다. 향후 증가를 처리하기 위해 클러스터에 추가 스토리지를 프로비저닝할 필요가 없습니다.

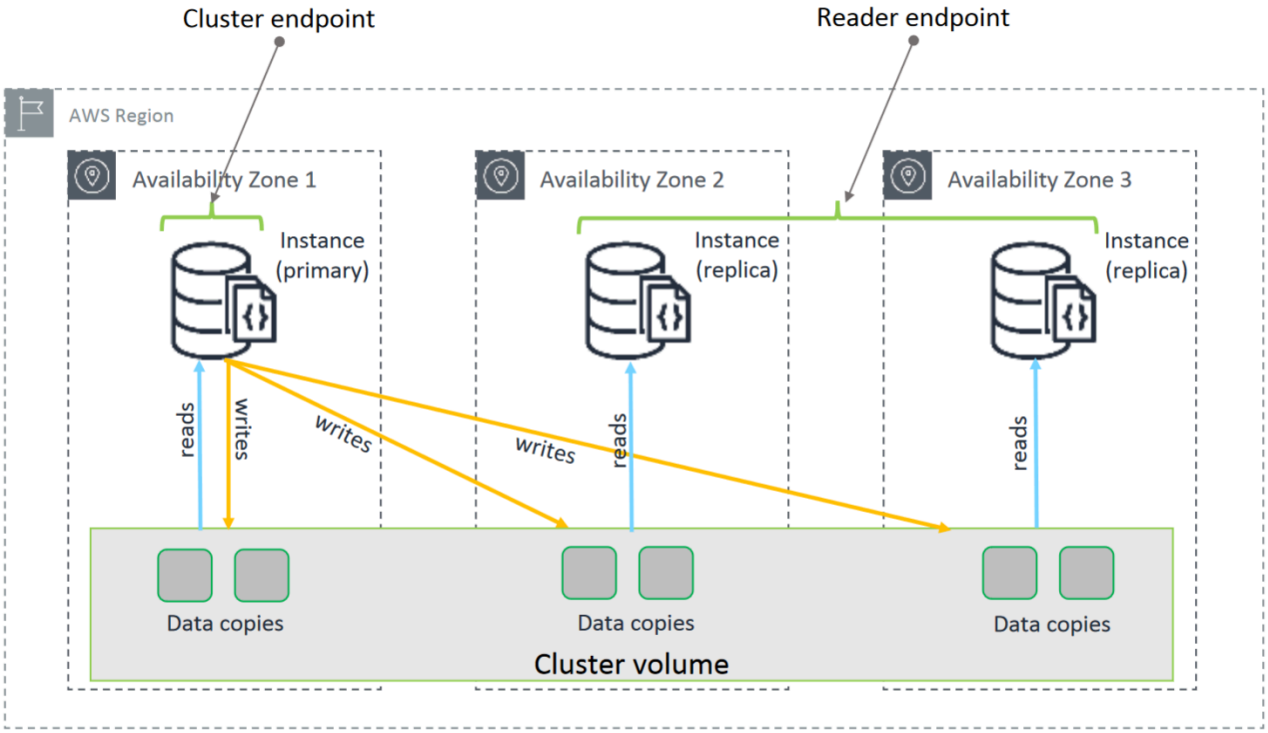
- Amazon DocumentDB를 사용하면 최대 15개의 복제본 인스턴스를 생성하여 읽기 처리량을 높여 높은 볼륨의 애플리케이션 요청을 지원할 수 있습니다. Amazon DocumentDB 복제본은 동일한 기본 스토리지를 공유하므로 비용이 절감되고 복제본 노드에서 쓰기를 수행할 필요가 없습니다. 이 기능을 사용하면 읽기 요청을 처리할 수 있는 처리 능력이 향상되고 복제본 지연 시간이 최대 한 자릿수 밀리초로 단축됩니다. 스토리지 볼륨 크기에 관계없이 몇 분 안에 복제본을 추가할 수 있습니다. 또한 Amazon DocumentDB는 리더 엔드포인트를 제공하므로 애플리케이션이 복제본이 추가 및 제거될 때 추적할 필요 없이 연결할 수 있습니다.
- Amazon DocumentDB를 사용하면 각 인스턴스의 계산 및 메모리 리소스를 위 또는 아래로 확장할 수 있습니다. 컴퓨팅 조정 작업은 일반적으로 몇 분이면 완료됩니다.
- Amazon DocumentDB는 Amazon Virtual Private Cloud(Amazon VPC)에서 실행되므로 데이터베이스를 자체 가상 네트워크에서 격리할 수 있습니다. 또한 클러스터에 대한 네트워크 액세스를 제어하도록 방화벽 설정을 구성할 수 있습니다.
- Amazon DocumentDB는 지속적으로 클러스터 상태를 모니터링합니다. 인스턴스 장애 발생 시 Amazon DocumentDB는 인스턴스 및 관련 프로세스를 자동으로 재시작합니다. Amazon DocumentDB는 데이터베이스 다시 실행 로그의 충돌 복구 재생이 필요하지 않으므로 재시작 시간이 크게 단축됩니다. 또한 Amazon DocumentDB는 데이터베이스 프로세스에서 데이터베이스 캐시를 분리하여 인스턴스를 다시 시작해도 캐시가 계속 유지되도록 합니다.
- 인스턴스 장애 시에서는 Amazon DocumentDB는 다른 가용 영역에서 작성하는 최대 15개의 Amazon DocumentDB 복제본 중 하나로 장애 조치를 자동화합니다. 복제본이 프로비저닝되지 않은 상태에서 오류가 발생하면 Amazon DocumentDB는 자동으로 새 Amazon DocumentDB 인스턴스를 작성하려고 합니다.
- Amazon DocumentDB의 백업 기능은 클러스터의 시점 복구를 가능하게 합니다. 이 기능을 통해 클러스터를 보존 기간 중 어느 시점(초)으로나 복원할 수 있습니다(마지막 5분까지 가능). 자동 백업 보존 기간은 최대 35일까지 구성할 수 있습니다. 자동 백업은 99.999999999% 의 내구성을 제공하도록 설계된 Amazon Simple Simple S3 Service(Amazon Simple S3)에 저장됩니다. Amazon DocumentDB 백업은 자동, 증분 및 지속적이며 클러스터 성능에 영향을 주지 않습니다.
- Amazon DocumentDB를 사용하면 AWS Key Management Service ()를 통해 생성하고 제어하는 키를 사용하여 데이터베이스를 암호화할 수 있습니다AWS KMS. Amazon DocumentDB 암호화와 함께 실행되는 데이터베이스 클러스터에서는 기본 저장소에 저장된 정지 상태의 데이터가 암호화됩니다. 동일한 클러스터에 있는 자동화된 백업, 스냅샷 및 복제본도 암호화됩니다.
- Amazon DocumentDB는 연방정 위험 및 인증 관리 프로그램(FedRAMP)에 따라 권한이 부여됩니다. 여기에는 AWS GovCloud(미국) 리전에 대한 FedRAMP High 권한 부여와 AWS 미국 동부/서부 리전에 대한 FedRAMP Moderate 권한이 있습니다. AWS 및 규정 준수 노력에 대한 자세한 내용은 AWS 규정 준수 프로그램 제공 범위 내 서비스를 참조하세요.

AWS 서비스를 처음 사용하는 경우 다음 리소스를 사용하여 자세히 알아보세요.

- AWS 는 컴퓨팅, 데이터베이스, 스토리지, 분석 및 기타 기능을 위한 서비스를 제공합니다. 모든 AWS 서비스에 대한 개요는 Amazon Web Services를 사용한 클라우드 컴퓨팅을 참조하세요.
- AWS 는 여러 데이터베이스 서비스를 제공합니다. 환경에 가장 적합한 서비스에 대한 지침은 AWS의 데이터베이스를 참조하십시오.

클러스터

클러스터는 0~16개의 인스턴스와 해당 인스턴스의 데이터를 관리하는 클러스터 스토리지 볼륨으로 구성됩니다. 모든 쓰기는 기본 인스턴스를 통해 수행됩니다. 모든 인스턴스(기본 및 복제본)는 읽기를 지원합니다. 클러스터의 데이터는 클러스터 볼륨에 저장되며 복사본은 세 개의 다른 가용 영역에 있습니다.



Amazon DocumentDB 5.0 인스턴스 기반 클러스터는 데이터베이스 클러스터에 대해 Amazon DocumentDB 표준 및 Amazon DocumentDB I/O 최적화라는 두 가지 스토리지 구성을 지원합니다. 자세한 내용은 Amazon DocumentDB 클러스터 스토리지 구성을 참조하세요.

인스턴스

Amazon DocumentDB 인스턴스는 클라우드에 있는 격리된 데이터베이스 환경입니다. 인스턴스에는 사용자가 만든 여러 개의 데이터베이스가 포함될 수 있습니다. AWS Management Console 또는를 사용하여 인스턴스를 생성하고 수정할 수 있습니다 AWS CLI.

인스턴스의 계산 및 메모리 용량은 인스턴스 클래스에 따라 결정됩니다. 사용자의 요구 사항에 가장 잘 맞는 인스턴스를 선택할 수 있습니다. 시간이 지나면서 요구 사항이 바뀌면 다른 인스턴스 클래스를 선택할 수 있습니다. 인스턴스 클래스 사양은 인스턴스 클래스 사양을 참조하십시오.

Amazon DocumentDB 인스턴스는 Amazon VPC 환경에서만 실행됩니다. Amazon VPC를 사용하면 가상 네트워킹 환경을 제어할 수 있습니다. 자신의 IP 주소 범위를 선택하고 서브넷을 생성하며 라우팅 및 액세스 제어 목록(ACL)을 구성할 수 있습니다.

Amazon DocumentDB 인스턴스를 작성하려면 먼저 인스턴스를 포함할 클러스터를 작성해야 합니다.

모든 인스턴스 클래스가 모든에서 지원되는 것은 아닙니다 AWS 리전. 다음 표에는 각 리전에서 지원되는 인스턴스 클래스가 나와 있습니다.

> **Note**
>
> 각 인스턴스 클래스에서 Amazon DocumentDB가 지원하는 인스턴스 유형의 전체 목록은 인스턴스 클래스 사양 섹션을 참조하세요.

리전별 지원되는 인스턴스 클래스

리전	R8G	R6GD	R6G	R5	R4	T4G	T3	서버리스
미국 동부(오하이오)	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨
미국 동부(버지니아 북부)	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨
미국 서부(오리건)	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨

리전	R8G	R6GD	R6G	R5	R4	T4G	T3	서버리스
아프리카(케이프타운)			지원됨	지원됨		지원됨	지원됨	지원됨
남아메리카(상파울루)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(홍콩)	지원됨		지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(하이데라바드)			지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(말레이시아)			지원됨			지원됨	지원됨	지원됨
아시아 태평양(뭄바이)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(오사카)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(서울)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(시드니)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(자카르타)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(멜버른)			지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(싱가포르)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨

리전	R8G	R6GD	R6G	R5	R4	T4G	T3	서버리스
아시아 태평양(태국)			지원됨			지원됨	지원됨	지원됨
아시아 태평양(도쿄)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
캐나다(중부)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
캐나다 서부(캘거리)			지원됨	지원됨		지원됨	지원됨	지원됨
유럽(프랑크푸르트)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
유럽(취리히)		지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
유럽(아일랜드)	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨
유럽(런던)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
유럽(밀라노)			지원됨	지원됨		지원됨	지원됨	지원됨
유럽(파리)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
유럽(스페인)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
유럽(스톡홀름)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
멕시코(중부)			지원됨			지원됨	지원됨	지원됨
중동(UAE)			지원됨	지원됨		지원됨	지원됨	지원됨
중국(베이징)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
중국(닝샤)	지원됨		지원됨	지원됨		지원됨	지원됨	지원됨
이스라엘(텔아비브)			지원됨	지원됨		지원됨	지원됨	지원됨

리전	R8G	R6GD	R6G	R5	R4	T4G	T3	서버리스
AWS GovCloud(미국 서부)	지원됨	지원됨	지원됨	지원됨			지원됨	지원됨
AWS GovCloud(미국 동부)		지원됨	지원됨	지원됨		지원됨	지원됨	지원됨

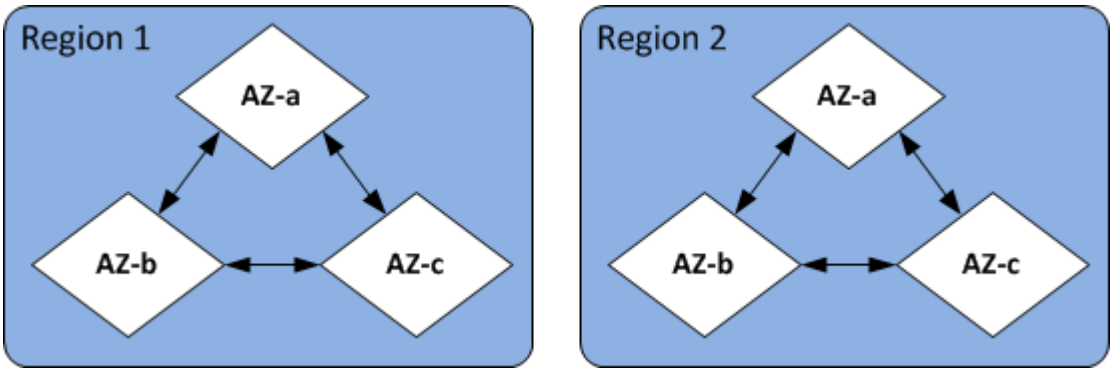
리전 및 가용 영역

AWS 리전 및 가용 영역은 클러스터 및 인스턴스의 물리적 위치를 정의합니다.

리전

AWS 클라우드 컴퓨팅 리소스는 전 세계 여러 지역(예: 북미, 유럽 또는 아시아)의 고가용성 데이터 센터 시설에 있습니다. 각 데이터 센터 위치를 AWS 리전이라고 합니다.

각 AWS 리전 는 서로 완전히 격리되도록 설계되었습니다 AWS 리전. 각 리전 내에는 가용 영역이 여러 개 있습니다. 서로 다른 가용 영역에서 노드를 시작하면 가능한 최고 수준의 내결함성을 갖출 수 있습니다. 다음 다이어그램은 AWS 리전 및 가용 영역의 작동 방식을 개괄적으로 보여줍니다.



가용 영역

각 AWS 리전 에는 가용 영역이라는 고유한 위치가 여러 개 포함되어 있습니다. 각 가용 영역은 다른 가용 영역의 장애로부터 격리되고 같은 리전의 다른 가용 영역에 경제적이고 지연 시간이 낮은 네트워크 연결을 제공하도록 엔지니어링됩니다. 여러 가용 영역에서 특정 클러스터에 대한 인스턴스를 시작하면 가용 영역에 드물게라도 장애가 발생할 경우 애플리케이션을 보호할 수 있습니다.

Amazon DocumentDB 아키텍처는 스토리지와 컴퓨팅을 분리합니다. 스토리지 계층의 경우 Amazon DocumentDB는 3개의 AWS 가용 영역에 걸쳐 6개의 데이터 사본을 복제합니다. 예를 들어, 두 개의 가용 영역만 지원하는 영역에서 Amazon DocumentDB 클러스터를 시작하는 경우 데이터 스토리지는 세 개의 가용 영역에 걸쳐 6가지 방식으로 복제되지만 컴퓨팅 인스턴스는 두 개의 가용 영역에서만 사용할 수 있습니다.

다음 표에는 지정된에서 클러스터의 컴퓨팅 인스턴스를 프로비저닝하는 AWS 리전 데 사용할 수 있는 가용 영역 수가 나열되어 있습니다.

리전 이름	리전	가용 영역(컴퓨팅)
미국 동부(오하이오)	us-east-2	3
미국 동부(버지니아 북부)	us-east-1	6
미국 서부(오리건)	us-west-2	4
아프리카(케이프타운)	af-south-1	3
남아메리카(상파울루)	sa-east-1	3
아시아 태평양(홍콩)	ap-east-1	3
아시아 태평양(하이데라바드)	ap-south-2	3
아시아 태평양(말레이시아)	ap-southeast-5	3
아시아 태평양(뭄바이)	ap-south-1	3
아시아 태평양(오사카)	ap-northeast-3	3
아시아 태평양(서울)	ap-northeast-2	4
아시아 태평양(싱가포르)	ap-southeast-1	3
아시아 태평양(시드니)	ap-southeast-2	3

리전 이름	리전	가용 영역(컴퓨팅)
아시아 태평양(자카르타)	ap-southeast-3	3
아시아 태평양(멜버른)	ap-southeast-4	3
아시아 태평양(태국)	ap-southeast-7	3
아시아 태평양(도쿄)	ap-northeast-1	3
캐나다(중부)	ca-central-1	3
캐나다 서부(캘거리)	ca-west-1	3
중국(베이징) 리전	cn-north-1	3
중국(닝샤)	cn-northwest-1	3
유럽(프랑크푸르트)	eu-central-1	3
유럽(취리히)	eu-central-2	3
유럽(아일랜드)	eu-west-1	3
유럽(런던)	eu-west-2	3
유럽(밀라노)	eu-south-1	3
유럽(파리)	eu-west-3	3
유럽(스페인)	eu-south-2	3
유럽(스톡홀름)	eu-north-1	3
멕시코(중부)	mx-central-1	3
중동(UAE)	me-central-1	3
이스라엘(텔아비브)	il-central-1	3

리전 이름	리전	가용 영역(컴퓨팅)
AWS GovCloud(미국 서부)	us-gov-west-1	3
AWS GovCloud(미국 동부)	us-gov-east-1	3

Amazon DocumentDB 요금

Amazon DocumentDB 클러스터는 다음 구성 요소를 기준으로 청구됩니다.

- 인스턴스 시간(시간당)—인스턴스의 인스턴스 클래스(예: db.r5.xlarge)를 기준으로 합니다. 요금은 시간 단위로 고시되지만, 청구서는 초 단위로 계산되고 시간을 10진수 형식으로 표시합니다. Amazon DocumentDB 사용량은 최소 10분을 기준으로 1초 단위로 청구됩니다. 자세한 내용은 인스턴스 클래스 관리 단원을 참조하십시오.
- I/O 요청(매월 100만 건당 요청) — 청구 주기 내에 수행하는 총 스토리지 I/O 요청 수.
- 백업 저장소(배월 단위 GiB) - 백업 저장소는 자동화된 데이터베이스 백업 및 사용 중인 데이터베이스 스냅샷과 관련된 저장소입니다. 백업 보존 기간을 연장하거나 추가 데이터베이스 스냅샷을 찍으면 데이터베이스가 사용하는 백업 스토리지가 증가합니다. 백업 스토리지는 GB-월 단위로 측정되며 초당으로 적용되지 않습니다. 자세한 내용은 Amazon DocumentDB에서 백업 및 복원 단원을 참조하십시오.
- 데이터 전송(GB당) - 인스턴스 안팎으로 인터넷 또는 다른 곳으로 데이터를 전송합니다 AWS 리전.

자세한 내용은 Amazon DocumentDB 요금을 참조하십시오.

무료 평가판

1개월 무료 평가판을 사용하여 Amazon DocumentDB를 무료로 사용해 볼 수 있습니다. 자세한 내용은 Amazon DocumentDB 요금 사이트의 무료 평가판 또는 Amazon DocumentDB 무료 평가판 FAQ를 참조하세요.

모니터링

인스턴스의 성능과 상태를 추적할 수 있는 여러 가지 방법이 있습니다. 무료 Amazon CloudWatch 서비스를 사용하여 인스턴스의 성능과 상태를 모니터링할 수 있습니다. 성능 차트는 Amazon DocumentDB

콘솔에서 찾을 수 있습니다. 인스턴스, 스냅샷, 파라미터 그룹 또는 보안 그룹에서 변경 사항이 발생할 때 알려줄 Amazon DocumentDB 이벤트에 가입할 수 있습니다.

자세한 내용은 다음을 참조하세요.

- CloudWatch를 사용하여 Amazon DocumentDB 모니터링
- 를 사용하여 Amazon DocumentDB API 호출 로깅 AWS CloudTrail

인터페이스

AWS Management Console 및를 포함하여 Amazon DocumentDB와 상호 작용하는 방법에는 여러 가지가 있습니다 AWS CLI.

AWS Management Console

AWS Management Console 는 간단한 웹 기반 사용자 인터페이스입니다. 콘솔에서 프로그래밍 없이 클러스터 및 인스턴스를 관리할 수 있습니다. Amazon DocumentDB 콘솔에 액세스하려면에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.

AWS CLI

AWS Command Line Interface (AWS CLI)를 사용하여 Amazon DocumentDB 클러스터 및 인스턴스를 관리할 수 있습니다. 최소한의 구성으로 원하는 터미널 프로그램에서 Amazon DocumentDB 콘솔에서 제공하는 모든 기능을 사용할 수 있습니다.

- 를 설치하려면 명령줄 인터페이스 설치를 AWS CLI참조하세요. AWS
- Amazon DocumentDB AWS CLI 용 사용을 시작하려면 AWS Amazon DocumentDB용 명령줄 인터페이스 참조를 참조하세요.

MongoDB 드라이버

Amazon DocumentDB 클러스터에서 애플리케이션을 개발하고 작성하는 경우, MongoDB 드라이버를 Amazon DocumentDB와 함께 사용할 수도 있습니다. 자세한 내용은 TLS가 활성화된 상태에서 연결 또는 TLS가 비활성화된 상태에서 연결의 MongoDB 쉘 탭을 참조하세요.

다음 단계

앞 섹션에서는 Amazon DocumentDB가 제공하는 기본 인프라 구성 요소에 대해 소개했습니다. 다음으로 무엇을 해야 할까요? 환경에 따라 시작하려면 다음 주제 중 하나를 참조하십시오.

- 를 사용하여 클러스터와 인스턴스를 생성하여 Amazon DocumentDB를 시작합니다 CloudFormation Amazon DocumentDB 빠른 시작 사용 CloudFormation.
- 시작 안내서의 지침에 따라 클러스터 및 인스턴스를 생성하여 Amazon DocumentDB를 시작하십시오.
- Amazon DocumentDB 엘라스틱 클러스터 시작하기의 지침에 따라 탄력적 클러스터를 생성하여 Amazon DocumentDB를 시작하십시오.
- Amazon DocumentDB 마이그레이션 및 업그레이드의 지침을 사용하여 MongoDB 구현을 Amazon DocumentDB로 마이그레이션하기

Amazon DocumentDB: 작동 방식

Amazon DocumentDB(MongoDB와 호환됨)는 완전 관리형의 MongoDB와 호환됨 데이터베이스 서비스입니다. Amazon DocumentDB를 사용하면 MongoDB에서 사용하는 것과 동일한 애플리케이션 코드를 실행하고 동일한 드라이버와 도구를 사용할 수 있습니다. Amazon DocumentDB는 MongoDB 3.6, 4.0, 5.0 및 8.0과 호환됩니다.

주제

- Amazon DocumentDB 버전
- Amazon DocumentDB 엔드포인트
- TLS 지원
- Amazon DocumentDB 스토리지
- Amazon DocumentDB 복제
- Amazon DocumentDB 신뢰성
- 읽기 기본 설정 옵션
- TTL 삭제
- 청구 가능 리소스

Amazon DocumentDB 버전

> **Note**
>
> 마이너 버전 번호 지정은 Amazon DocumentDB 5.0 이상에 적용됩니다. Amazon DocumentDB 3.6 및 4.0은 엔진 패치가 있는 메이저 버전만 사용합니다.

Amazon DocumentDB 5.0부터 엔진 버전은 major.major.minor.patch라는 네 가지 수준의 번호 지정 체계를 사용합니다.

- 메이저 버전 - 버전의 처음 두 숫자(예: 5.0)로 식별됩니다. 메이저 버전은 MongoDB 호환성 수준을 나타냅니다. 메이저 버전에는 새로운 기능, 성능 개선 및 MongoDB 호환성 업데이트가 도입되었습니다. 메이저 버전(예: 4.0에서 5.0으로) 간에 업그레이드하려면 메이저 버전 업그레이드(MVU)가 필요합니다. 자세한 내용은 Amazon DocumentDB 인플레이스 주요 버전 업그레이드 단원을 참조하십시오.
- 마이너 버전 - 버전의 세 번째 번호로 식별됩니다(예: 5.0.1의 1). 마이너 버전에는 동일한 메이저 버전 내의 새 기능, 보안 수정 및 버그 수정과 같은 개선 사항이 포함되어 있습니다. 새 마이너 버전으로 수동으로 업그레이드할 수 있습니다. 자세한 내용은 Amazon DocumentDB 마이너 버전 업그레이드 단원을 참조하십시오.
- 패치 버전 - 엔진 패치는 마이너 버전 내에 적용되며 중요한 보안 수정 및 버그 수정을 포함합니다. 패치 버전은를 실행하여 확인할 수 있는 엔진 패치 버전(예: `3.0.17983`)에 의해 추적됩니다`db.runCommand({getEngineVersion: 1})`. 패치는 클러스터의 유지 관리 기간 동안 적용됩니다. 자세한 내용은 Amazon DocumentDB 유지 관리 단원을 참조하십시오.

새 클러스터를 생성할 때 사용 가능한 엔진 버전을 지정할 수 있습니다. 사용 가능한 버전 및 지원 날짜에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB 엔진 버전 지원 날짜.

Amazon DocumentDB를 사용할 때는 클러스터를 생성하는 것부터 시작합니다. 클러스터는 0개 이상의 데이터베이스 인스턴스 및 이 인스턴스의 데이터를 관리하는 클러스터 볼륨으로 구성됩니다. Amazon DocumentDB 클러스터 볼륨은 여러 가용 영역에 걸쳐 있는 가상 데이터베이스 스토리지 볼륨입니다. 각 가용 영역마다 클러스터 데이터 복사본이 있습니다.

Amazon DocumentDB 클러스터는 다음 두 가지 구성 요소로 구성됩니다.

- 클러스터 볼륨—클라우드 네이티브 스토리지 서비스를 사용하여 3개의 가용 영역에 걸쳐 6가지 방식으로 데이터를 복제하여 내구성이 뛰어나고 가용성이 뛰어난 스토리지를 제공합니다. Amazon

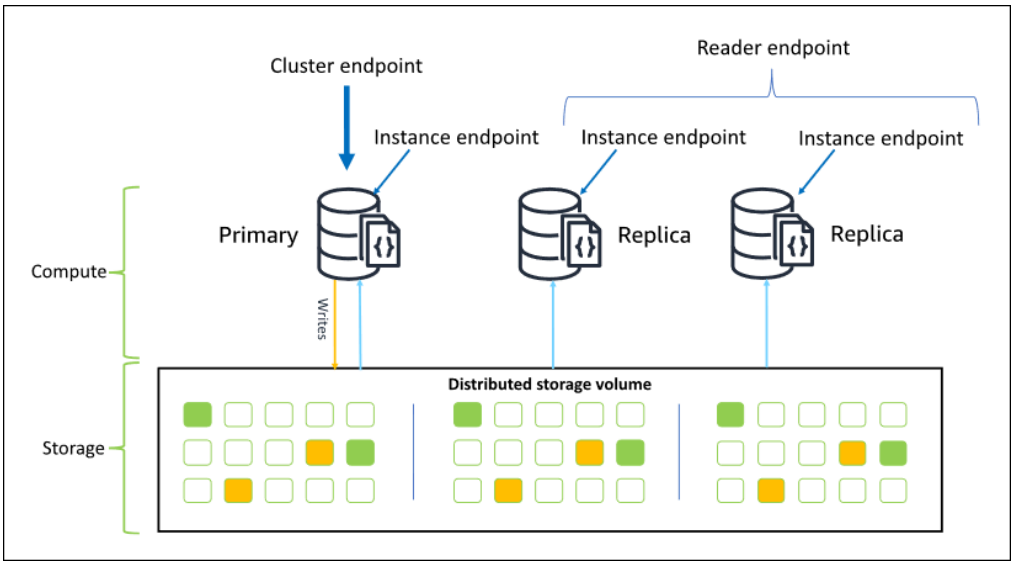
DocumentDB 클러스터에는 정확히 하나의 클러스터 볼륨이 있으며, 엔진 버전 8.0 이상(이전 엔진 버전의 경우 128TiB)에 대해 최대 256TiB의 데이터를 저장할 수 있습니다.

- 인스턴스—데이터베이스에 대한 처리 능력을 제공하고, 데이터를 클러스터 스토리지 볼륨에 쓰고, 데이터를 읽어냅니다. Amazon DocumentDB 클러스터에는 0~16개의 인스턴스가 있을 수 있습니다.

인스턴스는 두 가지 역할 중 하나를 수행합니다.

- 기본 인스턴스—읽기 및 쓰기 작업을 지원하고 클러스터 볼륨에 대한 모든 데이터 수정을 수행합니다. 각 Amazon DocumentDB 클러스터에는 기본 인스턴스가 하나씩 있습니다.
- 복제본 인스턴스 - 읽기 작업만 지원합니다. Amazon DocumentDB 클러스터는 기본 인스턴스 외에 최대 15개의 복제본을 가질 수 있습니다. 복제본이 여러 개 있으면 읽기 워크로드를 분산시킬 수 있습니다. 또한 복제본을 별도의 가용 영역에 두어 클러스터 가용성을 높일 수 있습니다.

다음 다이어그램은 Amazon DocumentDB 클러스터의 클러스터 볼륨, 기본 인스턴스 및 복제본 간의 관계를 보여줍니다:



클러스터 인스턴스의 인스턴스 클래스가 같을 필요는 없으며 이 인스턴스를 원하는 대로 프로비저닝하고 종료할 수 있습니다. 이 아키텍처를 사용하여 클러스터 컴퓨팅 파워를 스토리지와 별개로 확장할 수 있습니다.

애플리케이션이 기본 인스턴스에 데이터를 쓸 때 기본 인스턴스는 클러스터 볼륨에 내구성 있는 쓰기를 실행합니다. 그런 다음 각 활성 복제본에 해당 쓰기 상태(데이터 제외)를 복제합니다. Amazon DocumentDB 복제본은 쓰기 처리에 관여하지 않으므로 Amazon DocumentDB 복제본은 읽기 확장에 유리합니다. Amazon DocumentDB 복제본에서의 읽기 작업은 최종적으로 복제본 지연 시간을 최소화하면서 일관되게 이루어지는데, 이는 기본 인스턴스가 데이터를 쓴 후 보통 100밀리초 미만입니다. 복

제본에서 읽기는 기본 인스턴스에 쓰여진 순서대로 읽힙니다. 복제본 지연 시간은 데이터 변경 비율에 따라 달라지며 쓰기 작업 기간 집중적으로 일어나는 기간에는 복제본 지연 시간이 늘어날 수 있습니다. 자세한 내용은 Amazon DocumentDB 지표의 `ReplicationLag` 지표를 참조하십시오.

Amazon DocumentDB 엔드포인트

Amazon DocumentDB는 다양한 사용 사례를 제공하기 위해 여러 연결 옵션을 제공합니다. Amazon DocumentDB 클러스터의 인스턴스에 연결하려면 인스턴스의 엔드포인트를 지정합니다. 엔드포인트란 콜론으로 구분된 호스트 주소와 포트 번호입니다.

복제본 세트 모드에서 클러스터 엔드포인트를 사용하여 클러스터에 연결합니다(참조Amazon DocumentDB에 복제본 세트로 연결). 리더 엔드포인트 또는 인스턴스 엔드포인트에 연결해야 하는 경우가 아니면이 접근 방식을 사용합니다. 요청을 복제본으로 라우팅하려면 애플리케이션의 읽기 일관성 요구 사항을 충족하면서 읽기 확장성을 최대화하는 드라이버 읽기 기본 설정을 선택합니다. `secondaryPreferred` 읽기 기본 설정은 복제본 읽기를 활성화하고, 더 많은 작업을 수행할 수 있도록 기본 인스턴스를 비웁니다.

Amazon DocumentDB 클러스터에서 사용할 수 있는 엔드포인트는 다음과 같습니다.

클러스터 엔드포인트

클러스터 엔드포인트는 클러스터의 현재 기본 인스턴스에 연결됩니다. 읽기 및 쓰기 작업에 클러스터 엔드포인트를 사용할 수 있습니다. Amazon DocumentDB 클러스터에는 클러스터 엔드포인트가 하나만 있습니다.

클러스터 엔드포인트는 클러스터에 대한 읽기 및 쓰기 연결에 장애 조치 지원을 제공합니다. 클러스터의 현재 기본 인스턴스가 실패하고 클러스터에 적어도 하나의 활성 읽기 전용 복제본이 있으면 클러스터 엔드포인트가 연결 요청을 새 기본 인스턴스에 자동으로 리디렉션합니다. Amazon DocumentDB 클러스터에 연결할 때 복제본 세트 모드에서 클러스터 엔드포인트를 사용합니다(참조Amazon DocumentDB에 복제본 세트로 연결).

다음은 Amazon DocumentDB 클러스터 엔드포인트의 예제입니다.

```
sample-cluster.cluster-123456789012.us-east-1.docdb.amazonaws.com:27017
```

다음은 이 클러스터 엔드포인트를 사용하는 연결 문자열 예제입니다.

```
mongodb://username:password@sample-cluster.cluster-123456789012.us-
east-1.docdb.amazonaws.com:27017
```

클러스터 엔드포인트 찾기에 대한 자세한 내용은 클러스터 엔드포인트 찾기 단원을 참조하십시오.

리더 엔드포인트

리더 엔드포인트는 클러스터에 있는 사용 가능한 모든 복제본 간에 읽기 전용 연결을 로드 밸런싱합니다. `replicaSet` 모드를 통해 연결하는 경우 클러스터 리더 엔드포인트가 클러스터 엔드포인트로 작동합니다. 즉, 연결 문자열에서 복제본 세트 파라미터는 `&replicaSet=rs0`입니다. 이 경우 기본에서 쓰기 작업을 수행할 수 있습니다. 그러나 `directConnection=true`를 지정하는 클러스터에 연결하는 경우 리더 엔드포인트에 대한 연결을 통해 쓰기 작업을 수행하려고 하면 오류가 발생합니다. Amazon DocumentDB 클러스터에는 정확히 하나의 리더 엔트포인트가 있습니다.

클러스터에 기본 인스턴스 하나만 있는 경우 리더 엔드포인트가 기본 인스턴스에 연결합니다. 복제본 인스턴스를 Amazon DocumentDB 클러스터에 추가할 때, 리더 엔드포인트는 복제본이 활성화된 후 새 복제본에 대한 읽기 전용 연결을 엽니다.

다음은 Amazon DocumentDB 클러스터에 대한 리더 엔드포인트의 예제입니다.

```
sample-cluster.cluster-ro-123456789012.us-east-1.docdb.amazonaws.com:27017
```

다음은 리더 엔드포인트를 사용하는 연결 문자열 예제입니다.

```
mongodb://username:password@sample-cluster.cluster-ro-123456789012.us-
east-1.docdb.amazonaws.com:27017
```

리더 엔드포인트는 읽기 요청이 아닌 읽기 전용 연결을 로드 밸런싱합니다. 일부 리더 엔드포인트 연결이 다른 연결보다 매우 많이 사용되는 경우 읽기 요청이 클러스터의 인스턴스 간에 균등하게 밸런싱되지 않을 수 있습니다. 클러스터 엔드포인트에 복제본 세트로 연결하고 secondaryPreferred 읽기 기본 설정 옵션을 사용하여 요청을 분산하는 것이 좋습니다.

클러스터 엔드포인트 찾기에 대한 자세한 내용은 클러스터 엔드포인트 찾기 단원을 참조하십시오.

인스턴스 엔드포인트

인스턴스 엔드포인트는 클러스터에 있는 특정 인스턴스에 연결됩니다. 읽기 및 쓰기 작업에 현재 기본 인스턴스의 인스턴스 엔드포인트를 사용할 수 있습니다.니다. 하지만 읽기 전용 복제본의 인스턴스 엔드포인트에 대한 쓰기 작업을 수행하려는 시도로 인해 오류가 발생합니다. Amazon DocumentDB 클러스터에는 활성 인스턴스당 하나의 인스턴스 엔드포인트가 있습니다.

클러스터 엔드포인트 또는 리더 엔드포인트가 부적합한 시나리오에서는 인스턴스 엔드포인트가 특정 인스턴스에 대한 연결을 직접 제어합니다. 예제 사용 사례에서는 주기적인 읽기 전용 분석 워크로드를 프로비저닝합니다. 일반 복제본 인스턴스보다 큰 인스턴스를 프로비저닝하고, 인스턴스 엔드포인트를

사용하여 더 큰 새 인스턴스에 직접 연결하며, 분석 쿼리를 실행한 후 인스턴스를 종료할 수 있습니다. 인스턴스 엔드포인트를 사용하면 분석 트래픽이 다른 클러스터 인스턴스에 영향을 주지 않습니다.

다음은 Amazon DocumentDB 클러스터의 단일 인스턴스에 대한 인스턴스 엔드포인트의 예입니다.

```
sample-instance.123456789012.us-east-1.docdb.amazonaws.com:27017
```

다음은 이 인스턴스 엔드포인트를 사용하는 연결 문자열 예제입니다.

```
mongodb://username:password@sample-instance.123456789012.us-
east-1.docdb.amazonaws.com:27017
```

> **Note**
>
> 장애 조치 이벤트로 인해 기본 또는 복제본으로서 인스턴스의 역할이 변경될 수 있습니다. 애플리케이션에서 특정 인스턴스 엔드포인트가 기본 인스턴스라고 가정해서는 안 됩니다. 프로덕션 애플리케이션의 인스턴스 엔드포인트에 연결하지 마십시오. 대신 복제본 세트 모드에서 클러스터 엔드포인트를 사용합니다(참조Amazon DocumentDB에 복제본 세트로 연결). 인스턴스 장애 조치 우선 순위의 고급 제어에 대한 자세한 내용은 Amazon DocumentDB 클러스터 내결함성에 대한 이해 단원을 참조하십시오.

클러스터 엔드포인트 찾기에 대한 자세한 내용은 인스턴스의 엔드포인트 찾기 단원을 참조하십시오.

복제본 세트 모드

복제본 세트 이름 `rs0`을 지정하여 복제본 세트 모드에서 Amazon DocumentDB 클러스터 엔드포인트에 연결할 수 있습니다. 복제본 세트 모드에서 연결하면 Read Concern(읽기 문제), Write Concern(쓰기 문제) 및 Read Preference(읽기 기본 설정) 옵션을 지정할 수 있습니다. 자세한 내용은 읽기 정합성 단원을 참조하십시오.

다음은 복제본 세트 모드에서 연결하는 예제 연결 문자열입니다.

```
mongodb://username:password@sample-cluster.cluster-123456789012.us-
east-1.docdb.amazonaws.com:27017/?replicaSet=rs0
```

복제본 세트 모드로 연결하면 Amazon DocumentDB 클러스터가 복제본 세트로 드라이버와 클라이언트에 나타납니다. Amazon DocumentDB 클러스터에서 추가 및 제거된 인스턴스는 복제본 집합 구성에 자동으로 반영됩니다.

각 Amazon DocumentDB 클러스터는 기본 이름이 `rs0`인 단일 복제본 세트로 구성됩니다. 복제본 세트 이름은 수정할 수 없습니다.

일반적인 용도에서는 복제본 세트 모드에서 클러스터 엔드포인트에 연결하는 것이 좋은 방법입니다.

> **Note**
>
> Amazon DocumentDB 클러스터의 모든 인스턴스는 동일한 TCP 포트에서 연결을 수신합니다.

TLS 지원

전송 계층 보안(TLS)을 사용하여 Amazon DocumentDB에 연결하는 방법에 대한 자세한 내용은 전송 중 데이터 암호화을 참조하십시오.

Amazon DocumentDB 스토리지

Amazon DocumentDB 데이터는 솔리드 스테이트 드라이브(SSD)를 사용하는 단일 가상 볼륨인 클러스터 볼륨에 저장됩니다. 클러스터 볼륨은 데이터 복사본 6개로 구성되며, 데이터 복사본은 단일 AWS 리전에서 여러 가용 영역에 걸쳐 자동으로 복제됩니다. 이 복제를 통해 데이터의 내구성을 높이고 데이터 손실 가능성을 줄일 수 있습니다. 또한 다른 가용 영역에 데이터 복사본이 이미 있어 장애 조치가 이루어지는 동안 클러스터 가용성이 높아집니다. 이러한 복사본은 계속해서 Amazon DocumentDB 클러스터의 인스턴스에 데이터 요청을 제공할 수 있습니다.

데이터 스토리지 요금이 청구되는 방법

Amazon DocumentDB는 데이터의 양이 증가함에 따라 클러스터 볼륨의 크기가 자동으로 증가합니다. Amazon DocumentDB 클러스터 볼륨은 엔진 버전 8.0 이상의 경우 최대 256TiB(이전 엔진 버전의 경우 128TiB)까지 확장할 수 있습니다. 그러나 Amazon DocumentDB 클러스터 볼륨에서 사용하는 공간에 대해서만 요금이 부과됩니다. Amazon DocumentDB 4.0부터는 컬렉션이나 인덱스를 삭제하는 등의 방법으로 데이터가 제거되면 비슷한 양만큼 할당된 전체 공간이 감소합니다. 따라서 더 이상 필요하지 않은 컬렉션, 인덱스 및 데이터베이스를 삭제하여 스토리지 요금을 절감할 수 있습니다. 위 목록에 있는 Amazon DocumentDB 버전 3.6에서는 데이터를 제거할 때 복원된 공간을 클러스터 볼륨이 다시 사용할 수 있지만 볼륨 자체의 크기는 줄어들지 않습니다. 따라서 버전 3.6에서는 여유 공간이 재사용되더라도 컬렉션 또는 인덱스를 삭제할 때 스토리지가 변경되지 않을 수 있습니다.

> **Note**
>
> Amazon DocumentDB 3.6의 스토리지 비용은 스토리지 "하이 워터 마크"(Amazon DocumentDB 클러스터에 할당된 최대 양)를 기반으로 합니다. 대량의 임시 정보를 생성하거나

불필요한 오래된 데이터를 제거하기 전에 대량의 새 데이터를 로드하는 ETL 관행을 피함으로써 비용을 관리할 수 있습니다. Amazon DocumentDB 클러스터에서 데이터를 제거하면 많은 양의 공간이 할당되지만 사용되지 않는 경우 하이 워터 마크를 재설정하려면 논리적 데이터 덤프를 수행하고 mongodump 또는 mongorestore와 같은 도구를 사용하여 새 클러스터로 복원해야 합니다. 스냅샷을 생성하고 복원하면 기본 스토리지의 물리적 레이아웃이 복원된 스냅샷에서 동일하게 유지되기 때문에 할당된 스토리지가 감소하지 않습니다.

> **Note**
>
> mongodump와 mongorestore 같은 유틸리티를 사용하면 스토리지 볼륨에 읽고 쓰는 데이터의 크기에 따라 I/O 요금이 발생합니다.

Amazon DocumentDB 데이터 스토리지 및 I/O 요금에 대한 자세한 내용은 Amazon DocumentDB(MongoDB와 호환 가능) 요금 및 요금 FAQ를 참조하십시오.

Amazon DocumentDB 복제

Amazon DocumentDB 클러스터에서 각 복제본 인스턴스는 독립적 엔트포인트를 보여줍니다. 이 복제본 엔드포인트는 클러스터 볼륨의 데이터에 대한 읽기 전용 액세스를 제공하며 복제된 여러 인스턴스에서 데이터의 읽기 워크로드를 확장하도록 해줍니다. 또한 Amazon DocumentDB 클러스터의 데이터 읽기 성능을 개선하고 데이터 가용성을 높이는 데도 도움이 됩니다. Amazon DocumentDB 복제본은 장애 조치 대상이기도 하며 Amazon DocumentDB 클러스터의 기본 인스턴스에 장애가 발생할 경우 신속하게 승격됩니다.

Amazon DocumentDB 신뢰성

Amazon DocumentDB는 안정성, 내구성 및 내결함성을 고려하여 설계되었습니다. (가용성을 높이려면 Amazon DocumentDB 클러스터가 서로 다른 가용 영역에 여러 개의 복제본 인스턴스가 있도록 구성해야 합니다.) Amazon DocumentDB에는 신뢰할 수 있는 데이터베이스 솔루션으로 만들어 주는 몇 가지 자동 기능이 포함되어 있습니다.

스토리지 자동 복구

Amazon DocumentDB는 데이터의 여러 복사본을 3개의 가용 영역에 유지하므로 스토리지 장애로 인해 데이터가 손실될 가능성을 크게 줄입니다. Amazon DocumentDB는 클러스터 볼륨에서 장애를 자동으로 감지합니다. 클러스터 볼륨의 세그먼트에 장애가 발생하면 Amazon DocumentDB는 즉시 세그먼

트를 복구합니다. 이때 클러스터 볼륨을 구성하는 나머지 볼륨의 데이터를 사용하여 복구된 세그먼트의 데이터도 이용 가능합니다. 따라서 Amazon DocumentDB는 데이터 손실을 방지하고 인스턴스 장애를 복구하기 위해 시점 복원을 수행할 필요성을 줄입니다.

유지 가능한 캐시 워밍

Amazon DocumentDB는 페이지 캐시가 데이터베이스와 독립적으로 생존할 수 있도록 데이터베이스와 별도의 프로세스로 자신의 페이지 캐시를 관리합니다. 데이터베이스 장애가 발생한 경우에는 페이지 캐시가 메모리에 남습니다. 이렇게 하면 데이터베이스를 다시 시작할 때 버퍼 풀이 가장 최근의 상태로 워밍업됩니다.

충돌 복구

Amazon DocumentDB는 거의 순간적으로 충돌을 복구하고 애플리케이션 데이터를 계속해서 제공하도록 설계되었습니다. Amazon DocumentDB는 병렬 스레드에서 비동기식으로 충돌 복구를 수행하므로, 충돌 후 거의 즉시 데이터베이스가 열리고 사용할 수 있습니다.

리소스 거버넌스

Amazon DocumentDB는 상태 확인과 같은 중요한 프로세스에 대한 리소스를 보호합니다. 인스턴스에 높은 메모리 압력이 발생하면 Amazon DocumentDB는 요청을 제한합니다. 메모리 압력이 가라앉을 때까지 일부 작업이 대기 중일 수 있습니다. 메모리 압력이 계속되면 대기 중인 작업이 시간 초과될 수 있습니다.

CloudWatch 지표 `LowMemThrottleQueueDepth`, , ,를 사용하여 메모리 제한이 낮은지 모니터링합니
다`LowMemThrottleMaxQueueDepthLowMemNumOperationsThrottledLowMemNumOperationsTimed`
자세한 내용은 CloudWatch를 사용하여 Amazon DocumentDB 모니터링 단원을 참조하십시오.

메모리 압력이 지속되면 인스턴스를 확장하여 워크로드에 추가 메모리를 제공합니다.

읽기 기본 설정 옵션

Amazon DocumentDB는 클라우드 네이티브 공유 스토리지 서비스를 사용하여 세 개의 가용 영역에 걸쳐 데이터를 6회 복제하여 높은 수준의 내구성을 제공합니다. Amazon DocumentDB는 내구성을 확보하기 위해 데이터를 여러 인스턴스에 복제하지 않습니다. 인스턴스가 1개이건 15개이건 클러스터 데이터의 내구성이 유지됩니다.

주제

- 쓰기 내구성
- 읽기 격리
- 읽기 정합성
- 높은 가용성
- 읽기 확장

쓰기 내구성

Amazon DocumentDB는 고유한, 배포된, 내결함성, 자가 복구 스토리지 시스템을 사용합니다. 이 시스템은 3개의 AWS 가용 영역에 6개의 데이터 복사본(V=6)을 복제하여 고가용성과 내구성을 제공합니다. Amazon DocumentDB는 데이터를 작성할 때 클라이언트에게 쓰기를 승인하기 전에 모든 쓰기가 다수의 노드에 지속적으로 기록되도록 보장합니다. 3노드 MongoDB 복제본 세트를 실행하는 경우 쓰기 문제 `{w:3, j:true}`을 사용하면 Amazon DocumentDB와 비교할 때 최상의 구성을 얻을 수 있습니다.

Amazon DocumentDB 클러스터에 대한 쓰기는 클러스터의 라이터 인스턴스에서 처리해야 합니다. 리더에 쓰기를 시도하면 오류가 발생합니다. Amazon DocumentDB 기본 인스턴스에서 승인된 쓰기는 내구성이 강하며 롤백할 수 없습니다. Amazon DocumentDB는 기본적으로 내구성이 뛰어나며 내구성이 없는 쓰기 옵션을 지원하지 않습니다. 사용자가 내구성 수준(즉, 쓰기 문제)을 수정할 수 없습니다. Amazon DocumentDB는 w=아무거나를 무시하며 사실상 w: 3 및 j: true입니다. 줄일 수는 없습니다.

Amazon DocumentDB 아키텍처에서 스토리지와 컴퓨팅이 분리되어 있기 때문에 단일 인스턴스를 가진 클러스터는 내구성이 매우 높습니다. 내구성은 스토리지 계층에서 처리됩니다. 따라서 단일 인스턴스와 3개의 인스턴스가 있는 하나의 Amazon DocumentDB 클러스터는 동일한 수준의 내구성을 달성합니다. 데이터의 뛰어난 내구성을 유지하면서 클러스터를 특정 사용 사례로 구성할 수 있습니다.

Amazon DocumentDB 클러스터에 대한 쓰기는 단일 문서 내에서 원자성을 갖습니다.

Amazon DocumentDB는 `wtimeout` 옵션을 지원하지 않으며 값이 지정된 경우 오류를 반환하지 않습니다. 기본 Amazon DocumentDB 인스턴스에 대한 쓰기는 무기한 차단되지 않도록 보장됩니다.

읽기 격리

Amazon DocumentDB 인스턴스에서 읽은 데이터는 쿼리가 시작되기 전에만 지속되는 데이터를 반환합니다. 읽기는 쿼리 실행이 시작된 후 수정한 데이터를 반환하지 않으며 어떠한 경우에도 더티 읽기는 불가능합니다.

읽기 정합성

Amazon DocumentDB 클러스터에서 읽은 데이터는 내구성이 뛰어나며 롤백되지 않습니다. 요청 또는 연결에 대한 읽기 환경설정을 지정하여 Amazon DocumentDB 읽기에 대한 읽기 일관성을 수정할 수 있습니다. Amazon DocumentDB는 내구성이 떨어지는 읽기 옵션을 지원하지 않습니다.

Amazon DocumentDB 클러스터의 기본 인스턴스로부터의 읽기는 정상적인 동작 조건에서 강력하게 일관되고 쓰기 후 읽기 일관성을 갖습니다. 쓰기와 그 다음 읽기 사이에서 장애 조치 이벤트가 발생할 경우 시스템이 일시적으로 강력히 일관되지 않은 읽기를 반환할 수 있습니다. 읽기 전용 복제본에서 모든 읽기는 최종적으로 일관되며 같은 순서로 데이터를 반환할 뿐 아니라 복제 지연 시간이 종종 100ms 미만입니다.

Amazon DocumentDB 읽기 기본 설정

Amazon DocumentDB는 복제본 설정 모드에서 클러스터 엔드포인트에서 데이터를 읽을 때에만 읽기 환경설정 옵션을 설정할 수 있도록 지원합니다. 읽기 환경설정 옵션을 설정하면 MongoDB 클라이언트 또는 드라이버가 읽기 요청을 Amazon DocumentDB 클러스터의 인스턴스로 라우트하는 방법에 영향을 미칩니다. 특정 쿼리에 대해 또는 MongoDB 드라이버의 일반 옵션으로 읽기 기본 설정 옵션을 설정할 수 있습니다. 읽기 기본 설정 옵션을 설정하는 방법은 클라이언트 또는 드라이버 설명서를 참조하십시오.

클라이언트 또는 드라이버가 복제본 설정 모드에서 Amazon DocumentDB 클러스터 엔드포인트에 연결하지 않으면 읽기 환경설정을 지정한 결과가 정의되지 않습니다.

Amazon DocumentDB는 태그 세트를 읽기 환경설정으로 설정하는 것을 지원하지 않습니다.

지원되는 읽기 기본 설정 옵션

- **primary**—primary 읽기 기본 설정을 지정하면 모든 읽기가 클러스터의 기본 인스턴스로 라우팅되도록 할 수 있습니다. 기본 인스턴스를 사용할 수 없으면 읽기 작업이 실패합니다. primary 읽기 기본 설정은 쓰기 후 읽기 일관성을 생성하며, 고가용성과 읽기 확장성보다 쓰기 후 읽기 일관성이 우선시되는 사용 사례에 적합합니다.

  다음 예제에서는 primary 읽기 기본 설정을 지정합니다.

  ```
  db.example.find().readPref('primary')
  ```

- **primaryPreferred**—일반 작동 시 기본 인스턴스에 대한 primaryPreferred 읽기 기본 설정 경로를 지정합니다. 기본 장애 조치가 있을 경우 클라이언트가 요청을 복제본으로 라우팅합니다.

primaryPreferred 읽기 기본 설정은 일반적인 작업 중 쓰기 후 읽기 일관성을 생성하고, 장애 조치 이벤트 중 최종적 일관된 읽기를 생성합니다. primaryPreferred 읽기 기본 설정은 읽기 확장성보다 쓰기 후 읽기 일관성이 우선시되지만 고가용성도 필요한 사용 사례에 적합합니다.

다음 예제에서는 primaryPreferred 읽기 기본 설정을 지정합니다.

```
db.example.find().readPref('primaryPreferred')
```

- **secondary**— secondary 읽기 기본 설정을 지정하면 읽기가 기본 인스턴스가 아닌 복제본으로만 라우팅됩니다. 클러스터에 복제본 인스턴스가 없으면 읽기 요청이 실패합니다. secondary 읽기 기본 설정은 최종적 일관된 읽기를 생성하며 고가용성과 쓰기 후 읽기 일관성보다 기본 인스턴스 쓰기 처리량이 우선시되는 사용 사례에 적합합니다.

  다음 예제에서는 secondary 읽기 기본 설정을 지정합니다.

```
db.example.find().readPref('secondary')
```

- **secondaryPreferred**—secondaryPreferred 읽기 환경설정을 지정하면 읽기는 하나 이상의 복제본이 활성화될 때 읽기 복제본으로 라우팅됩니다. 클러스터에 활성 복제본 인스턴스가 없으면 읽기 요청이 기본 인스턴스로 라우팅됩니다. secondaryPreferred 읽기 기본 설정은 읽기 전용 복제본에서 읽기를 처리할 때 최종적 일관된 읽기를 생성합니다. 장애 조치 이벤트를 제외하고 기본 인스턴스에서 읽기를 처리할 때 쓰기 후 읽기 일관성이 생깁니다. secondaryPreferred 읽기 기본 설정은 쓰기 후 읽기 일관성보다 읽기 확장성 및 고가용성이 우선시되는 사용 사례에 적합합니다.

  다음 예제에서는 secondaryPreferred 읽기 기본 설정을 지정합니다.

```
db.example.find().readPref('secondaryPreferred')
```

- **nearest**—클라이언트와 Amazon DocumentDB 클러스터의 모든 인스턴스 간에 측정된 지연 시간만을 기준으로 nearest 읽기 기본 설정 경로를 지정합니다. nearest 읽기 기본 설정은 읽기 전용 복제본에서 읽기를 처리할 때 최종적 일관된 읽기를 생성합니다. 장애 조치 이벤트를 제외하고 기본 인스턴스에서 읽기를 처리할 때 쓰기 후 읽기 일관성이 생깁니다. nearest 읽기 기본 설정은 쓰기 후 읽기 일관성 및 읽기 확장성보다 가장 낮은 읽기 지연 시간 및 고가용성 달성이 우선시되는 사용 사례에 적합합니다.

다음 예제에서는 nearest 읽기 기본 설정을 지정합니다.

```
db.example.find().readPref('nearest')
```

높은 가용성

Amazon DocumentDB는 복제본을 기본 인스턴스의 장애 조치 대상으로 사용하여 가용성이 높은 클러스터 구성을 지원합니다. 기본 인스턴스가 실패하면 Amazon DocumentDB 복제본이 새 기본 인스턴스로 승격되며, 기본 인스턴스에 대한 읽기 및 쓰기 요청이 예외로 실패하는 짧은 중단이 발생합니다.

Amazon DocumentDB 클러스터에 복제본이 없는 경우, 기본 인스턴스는 실패 중에 다시 작성됩니다. Amazon DocumentDB 복제본 승격은 기본 인스턴스를 다시 생성하는 것보다 훨씬 빠르므로 하나 이상의 Amazon DocumentDB 복제본을 장애 조치 대상으로 생성합니다.

장애 조치 대상으로 사용할 복제본은 기본 인스턴스와 동일한 인스턴스 클래스에 속해야 하며 기본 인스턴스와는 다른 가용 영역에 프로비저닝되어야 합니다. 장애 조치 대상으로 사용할 복제본을 제어할 수 있습니다.니다. 고가용성을 위해 Amazon DocumentDB를 구성하는 모범 사례는 Amazon DocumentDB 클러스터 내결함성에 대한 이해을 참조하십시오.

읽기 확장

Amazon DocumentDB 복제본은 읽기 규모 조정에 적합합니다. 클러스터 볼륨에서 이루어지는 읽기 작업 전용으로 사용됩니다. 즉 복제본이 쓰기는 처리하지 않습니다. 데이터 복제는 클러스터 볼륨 내에서 일어나며 인스턴스 간에는 일어나지 않습니다. 따라서 각 복제본의 리소스는 쿼리 처리에만 사용되지 데이터의 복제 및 쓰기에 사용되지 않습니다.

애플리케이션에 읽기 용량이 더 필요하면 클러스터에 신속하게 복제본을 추가할 수 있습니다(보통 10분 미만). 읽기 용량 요구 사항이 줄어들면 불필요한 복제본을 제거할 수 있습니다. Amazon DocumentDB 복제본에서는 필요한 읽기 용량에 대해서만 비용을 지불합니다.

Amazon DocumentDB는 읽기 환경설정 옵션을 사용하여 클라이언트 측 읽기 스케일링을 지원합니다. 자세한 내용은 Amazon DocumentDB 읽기 기본 설정을 참조하세요.

TTL 삭제

백그라운드 프로세스를 통해 생성된 TTL 인덱스 영역에서의 삭제는 특정 기간 내에 삭제된다고 보장할 수 없으며 가능한 한 빠른 시간 내에 수행됩니다. 인스턴스 크기, 인스턴스 리소스 사용률, 문서 크기 및 전체 처리량과 같은 요소가 TTL 삭제 소요 시간에 영향을 줄 수 있습니다.

TTL 모니터가 문서를 삭제하면 삭제할 때마다 IO 비용이 발생하므로 청구 금액이 증가합니다. 처리량 및 TTL 삭제율이 증가하면 IO 사용량 증가로 인해 청구 금액이 증가할 것으로 예상됩니다.

기존 컬렉션에 TTL 인덱스를 생성할 때는 인덱스를 생성하기 전에 만료된 문서를 모두 삭제해야 합니다. 현재 TTL 구현은 컬렉션에서 소량의 문서를 삭제하는 데 최적화되어 있는데, 이는 컬렉션에서 처음부터 TTL을 활성화한 경우 일반적으로 발생하며, 한 번에 많은 문서를 삭제해야 하는 경우 IOPS가 필요 이상으로 높아질 수 있습니다.

문서를 삭제하기 위해 TTL 인덱스를 작성하지 않으려면, 대신 시간을 기준으로 문서를 컬렉션으로 분할하고, 문서가 더 이상 필요하지 않을 때 해당 컬렉션을 삭제할 수 있습니다. 예를 들어, 일주일에 한 개의 컬렉션을 만들어 IO 비용을 들이지 않고도 삭제할 수 있습니다. 이 방법은 TTL 인덱스를 사용하는 것보다 훨씬 더 비용 효율적일 수 있습니다.

청구 가능 리소스

청구 가능한 Amazon DocumentDB 리소스 식별

Amazon DocumentDB는 완전 관리형 데이터베이스 서비스로서 인스턴스, 스토리지, I/O, 백업 및 데이터 전송 비용을 청구합니다. 자세한 내용은 Amazon DocumentDB(MongoDB와 호환됨) 요금을 참조하십시오.

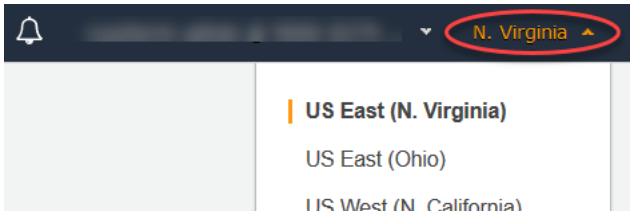
계정에서 청구 가능한 리소스를 검색하고 리소스를 잠재적으로 삭제하려면 AWS Management Console 또는를 사용할 수 있습니다 AWS CLI.

사용 AWS Management Console

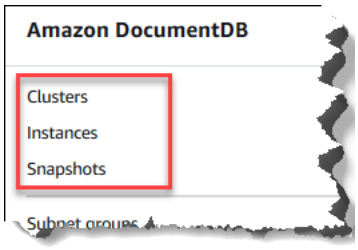
를 사용하면 지정된에 프로비저닝한 Amazon DocumentDB 클러스터, 인스턴스 및 스냅샷을 검색할 AWS Management Console수 있습니다 AWS 리전.

클러스터, 인스턴스 및 스냅샷을 검색하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 기본 리전이 아닌 다른 리전에서 청구 가능한 리소스를 검색하려면 화면의 오른쪽 상단 모서리에서 검색할 AWS 리전 를 선택합니다.



3. 탐색 창에서 검색하려는 청구 대상 리소스의 유형을 선택합니다. 클러스터, 인스턴스 또는 스냅샷.



4. 해당 리전에 프로비저닝된 모든 클러스터, 인스턴스 또는 스냅샷이 오른쪽 창에 나열됩니다. 클러스터, 인스턴스 및 스냅샷에 요금이 청구됩니다.

사용 AWS CLI

를 사용하면 지정된에 프로비저닝한 Amazon DocumentDB 클러스터, 인스턴스 및 스냅샷을 검색할 AWS CLI수 있습니다 AWS 리전.

클러스터 및 인스턴스를 검색하려면

다음 코드는 지정된 리전의 모든 클러스터 및 인스턴스를 나열합니다. 기본 리전에서 클러스터 및 인스턴스를 검색하려면 `--region` 파라미터를 생략할 수 있습니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-clusters \
    --region us-east-1 \
    --query 'DBClusters[?Engine==`docdb`]' | \
       grep -e "DBClusterIdentifier" -e "DBInstanceIdentifier"
```

Windows의 경우:

```
aws docdb describe-db-clusters ^
    --region us-east-1 ^
    --query 'DBClusters[?Engine==`docdb`]' | ^
       grep -e "DBClusterIdentifier" -e "DBInstanceIdentifier"
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
"DBClusterIdentifier": "docdb-2019-01-09-23-55-38",
        "DBInstanceIdentifier": "docdb-2019-01-09-23-55-38",
```

```
        "DBInstanceIdentifier": "docdb-2019-01-09-23-55-382",
"DBClusterIdentifier": "sample-cluster",
"DBClusterIdentifier": "sample-cluster2",
```

스냅샷을 검색하려면

다음 코드는 지정된 리전의 모든 스냅샷을 나열합니다. 기본 리전에서 스냅샷을 검색하려면 `--region` 파라미터를 생략할 수 있습니다.

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-cluster-snapshots \
  --region us-east-1 \
  --query 'DBClusterSnapshots[?Engine==`docdb`].
[DBClusterSnapshotIdentifier,SnapshotType]'
```

Windows의 경우:

```
aws docdb describe-db-cluster-snapshots ^
  --region us-east-1 ^
  --query 'DBClusterSnapshots[?Engine==`docdb`].
[DBClusterSnapshotIdentifier,SnapshotType]'
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
[
    [
        "rds:docdb-2019-01-09-23-55-38-2019-02-13-00-06",
        "automated"
    ],
    [
        "test-snap",
        "manual"
    ]
]
```

`manual` 스냅샷만 삭제하면 됩니다. `Automated` 스냅샷은 클러스터를 삭제할 때 함께 삭제됩니다.

불필요한 청구 대상 리소스 삭제

클러스터를 삭제하려면 먼저 해당 클러스터의 인스턴스를 모두 삭제해야 합니다.

- 인스턴스를 삭제하려면 Amazon DocumentDB 인스턴스 삭제 단원을 참조하십시오.

> **Important**
>
> 클러스터에서 인스턴스를 삭제하더라도 해당 클러스터와 연결된 스토리지 및 백업에 대해서는 계속 요금이 발생합니다. 모든 요금을 중단하려면 클러스터와 수동 스냅샷도 삭제해야 합니다.

- 클러스터를 삭제하려면 Amazon DocumentDB 클러스터 삭제 단원을 참조하십시오.
- 수동 스냅샷 삭제는 클러스터 스냅샷 삭제 단원을 참조하십시오.

문서 데이터베이스란?

일부 개발자는 정규화된 행과 열의 측면에서 데이터 모델을 생각하지 않습니다. 일반적으로, 애플리케이션 계층 내에서 데이터가 JSON 문서로서 나타나게 되는데, 그 이유는 개발자들이 자신의 데이터 모델을 문서로 생각하는 것이 보다 직관적이기 때문입니다.

애플리케이션 코드에서 사용하는 것과 동일한 문서 모델 형식을 이용함으로써 데이터베이스에서 데이터를 지속해서 사용할 수 있으므로 문서 데이터베이스의 대중성은 높아져 왔습니다. 문서 데이터베이스는 유연하고 신속한 개발을 위해 강력하고 직관적인 API를 제공합니다.

주제

- 문서 데이터베이스 사용 사례
- 문서 이해
- 문서 작업

문서 데이터베이스 사용 사례

사용 사례는 문서 데이터베이스 또는 데이터를 관리하기 위한 일부 다른 유형의 데이터베이스가 필요한지 여부를 결정합니다. 문서 데이터베이스는 빠르고 반복적인 개발을 위해 유연한 스키마가 필요한 워크로드에 유용합니다. 다음은 문서 데이터베이스가 중요한 이점을 제공하는 일부 사용 사례의 예입니다.

주제

- 사용자 프로필
- 실시간 빅 데이터

- 콘텐츠 관리

사용자 프로필

문서 데이터베이스에는 유연한 스키마가 있으므로 속성과 데이터 값이 다른 문서를 저장할 수 있습니다. 문서 데이터베이스는 서로 다른 사용자가 다양한 유형의 정보를 제공하는 온라인 프로필에 대한 실용적인 솔루션입니다. 문서 데이터베이스에서는 각 사용자에 고유한 속성만 저장할 수 있어 각 사용자의 프로필을 효율적으로 저장할 수 있습니다.

사용자가 프로필에서 정보를 추가하거나 제거하도록 선택했다고 가정합니다. 이 경우 해당 문서를 최근에 추가된 속성과 데이터가 포함되거나 새로 생략된 속성과 데이터가 생략된 업데이트된 버전으로 쉽게 바꿀 수 있습니다. 문서 데이터베이스는 이 수준의 특성 및 유동성을 쉽게 관리합니다.

실시간 빅 데이터

과거에는 운영 데이터베이스와 분석 데이터베이스가 각각 운영 환경과 업무/보고 환경 등 서로 다른 환경에서 유지 관리되어 운영 데이터에서 정보를 추출하는 데 어려움이 있었습니다. 실시간으로 운영 정보를 추출할 수 있다는 것은 경쟁이 치열한 비즈니스 환경에서 중요합니다. 문서 데이터베이스를 사용하면 비즈니스에서는 모든 소스에서 운영 데이터를 저장하고 관리할 수 있으며 동시에 분석을 위해 데이터를 선택한 BI 엔진에 전달할 수 있습니다. 두 개의 환경이 필요하지 않습니다.

콘텐츠 관리

콘텐츠를 효과적으로 관리하려면 다양한 소스에서 콘텐츠를 수집 및 집계한 다음 고객에게 제공할 수 있어야 합니다. 문서 데이터베이스는 유연한 스키마로 인해 모든 유형의 데이터 수집과 저장에 적합합니다. 이 데이터베이스를 사용하여 이미지, 설명, 비디오 등 사용자 생성 콘텐츠를 비롯한 새 유형의 콘텐츠를 생성하고 통합할 수 있습니다.

문서 이해

문서 데이터베이스는 관계형 데이터베이스에서처럼 각각 고유하고 고정된 구조를 가진 여러 테이블의 데이터를 정규화하는 대신 반정형 데이터를 문서로 저장하는 데 사용됩니다. 문서 데이터베이스에 저장된 문서는 중첩된 키-값 페어를 사용하여 문서의 구조 또는 스키마를 제공합니다. 그러나, 동일한 문서 데이터베이스에 다른 유형의 문서를 저장할 수 있으므로 다른 형식의 유사한 데이터를 처리하기 위한 요구 사항을 충족할 수 있습니다. 예를 들어, 각 문서는 자체적으로 설명되므로 문서 데이터베이스의 예제 문서 주제에 설명된 온라인 상점을 위한 JSON 인코딩 문서를 동일한 문서 데이터베이스에 저장할 수 있습니다.

주제

- SQL 대 비관계형 용어
- 단순 문서
- 내장 문서
- 문서 데이터베이스의 예제 문서
- 문서 데이터베이스에서 정규화 이해

SQL 대 비관계형 용어

다음 표에서는 문서 데이터베이스(MongoDB)에서 사용하는 용어를 SQL 데이터베이스에서 사용하는 용어와 비교합니다.

SQL	MongoDB
표	수집
열	문서
열	Field
프라이머리 키	ObjectId
인덱스	인덱스
보기	보기
중첩된 테이블 또는 객체	포함 문서
배열	배열

단순 문서

문서 데이터베이스의 모든 문서는 자체적으로 설명되어 있습니다. 이 설명서에서는 다른 인코딩 수단을 사용할 수 있는 경우에도 JSON과 유사한 형식의 문서를 사용합니다.

단순 문서에는 문서 내에서 모두 동일한 수준인 하나 이상의 필드가 있습니다. 다음 예에서 `SSN`, `LName`, `FName`, `DOB`, `Street`, `City`, `State-Province`, `PostalCode` 및 `Country` 필드는 문서 내에서 모두 형제 요소입니다.

```
{
   "SSN": "123-45-6789",
   "LName": "Rivera",
   "FName": "Martha",
   "DOB": "1992-11-16",
   "Street": "125 Main St.",
   "City": "Anytown",
   "State-Province": "WA",
   "PostalCode": "98117",
   "Country": "USA"
}
```

단순 문서에서 정보를 구성할 때 각 필드는 개별적으로 관리됩니다. 개인의 주소를 가져오려면 `Street`, `City`, `State-Province`, `PostalCode` 및 `Country`를 개별 데이터 항목으로 가져와야 합니다.

내장 문서

복잡한 문서는 문서 내에 내장 문서를 생성하여 데이터를 구성합니다. 내장 문서는 데이터 그룹화 및 개별 데이터 항목 중 제공된 사례에서 더 효율적인 방식으로 관리하는 데 도움이 됩니다. 이전 예제를 사용하면 기본 문서에 `Address` 문서를 포함할 수 있습니다. 이렇게 하면 다음과 같은 문서 구조가 발생합니다.

```
{
   "SSN": "123-45-6789",
   "LName": "Rivera",
   "FName": "Martha",
   "DOB": "1992-11-16",
   "Address":
   {
       "Street": "125 Main St.",
       "City": "Anytown",
       "State-Province": "WA",
       "PostalCode": "98117",
       "Country": "USA"
   }
}
```

이제 개별 필드(`"SSN":`), 내장 문서(`"Address":`) 또는 내장 문서의 멤버(`"Address": {"Street":}`)로 문서의 데이터에 액세스할 수 있습니다.

문서 데이터베이스의 예제 문서

이전에 설명한 대로 문서 데이터베이스의 각 문서는 자체적으로 설명되므로 문서 데이터베이스 내의 문서 구조는 서로 다를 수 있습니다. 다음의 두 문서(책 및 정기 간행물용)는 구조적으로 서로 다릅니다. 하지만 두 문서 모두 동일한 문서 데이터베이스에 있을 수 있습니다.

다음은 샘플 책 문서입니다.

```
{
    "_id" : "9876543210123",
    "Type": "book",
    "ISBN": "987-6-543-21012-3",
    "Author":
    {
        "LName":"Roe",
        "MI": "T",
        "FName": "Richard"
    },
    "Title": "Understanding Document Databases"
}
```

다음은 두 개의 기사가 있는 샘플 정기 간행물 문서입니다.

```
{
    "_id" : "0123456789012",
    "Publication": "Programming Today",
    "Issue":
    {
        "Volume": "14",
        "Number": "09"
    },
    "Articles" : [
        {
            "Title": "Is a Document Database Your Best Solution?",
            "Author":
            {
                "LName": "Major",
                "FName": "Mary"
            }
        },
        {
            "Title": "Databases for Online Solutions",
```

```
            "Author":
            {
                "LName": "Stiles",
                "FName": "John"
            }
        }
    ],
    "Type": "periodical"
}
```

이 두 문서의 구조를 비교합니다. 관계형 데이터베이스에서는 별도의 "정기 간행물" 및 "책" 테이블이나 "출판", "발행", "기사", "MI" 등 사용하지 않은 필드가 null 값인 단일 테이블이 필요합니다. 문서 데이터베이스는 각 문서가 자체 구조를 정의하는 반구조화이므로 이러한 두 문서는 null 필드가 없는 동일한 문서 데이터베이스에서 동시에 존재할 수 있습니다. 문서 데이터베이스는 희소 데이터를 처리하는 데 적합합니다.

문서 데이터베이스를 대상으로 개발하면 빠르고 반복적으로 개발할 수 있습니다. 이는 전체 모음에 대한 스키마를 변경하지 않고 문서의 데이터 구조를 동적으로 변경할 수 있기 때문입니다. 문서 데이터베이스는 신속한 개발 및 동적으로 변화하는 환경에 매우 적합합니다.

문서 데이터베이스에서 정규화 이해

문서 데이터베이스는 정규화되지 않습니다. 한 문서에서 발견된 데이터가 다른 문서에서 반복될 수 있습니다. 게다가 문서 간에 일부 데이터 차이가 있을 수 있습니다. 예를 들어, 온라인 상점에서 구매하고 구매에 대한 모든 세부 정보가 단일 문서에 저장되는 시나리오를 고려하십시오. 문서는 다음 JSON 문서와 같을 수 있습니다.

```
{
    "DateTime": "2018-08-15T12:13:10Z",
    "LName" : "Santos",
    "FName" : "Paul",
    "Cart" : [
        {
            "ItemId" : "9876543210123",
            "Description" : "Understanding Document Databases",
            "Price" : "29.95"
        },
        {
            "ItemId" : "0123456789012",
            "Description" : "Programming Today",
            "Issue": {
```

```
                "Volume": "14",
                "Number": "09"
            },
            "Price" : "8.95"
        },
        {
            "ItemId": "234567890-K",
            "Description": "Gel Pen (black)",
            "Price": "2.49"
        }
    ],
    "PaymentMethod" :
    {
        "Issuer" : "MasterCard",
        "Number" : "1234-5678-9012-3456"
    },
    "ShopperId" : "1234567890"
}
```

이러한 모든 정보는 거래 모음에서 문서로 저장됩니다. 나중에 품목 하나를 구매하지 않은 것을 알게 되었습니다. 따라서 다시 동일한 상점에 로그인하여 다시 구매했습니다. 이러한 구매도 거래 모음에서 다른 문서로 저장됩니다.

```
{
    "DateTime": "2018-08-15T14:49:00Z",
    "LName" : "Santos",
    "FName" : "Paul",
    "Cart" : [
        {
            "ItemId" : "2109876543210",
            "Description" : "Document Databases for Fun and Profit",
            "Price" : "45.95"
        }
    ],
    "PaymentMethod" :
    {
        "Issuer" : "Visa",
        "Number" : "0987-6543-2109-8765"
    },
    "ShopperId" : "1234567890"
}
```

이 두 문서, 즉 이름과 구매자 ID(같은 신용카드를 사용한 경우에는 신용카드 정보)가 중복된다는 점에 주목하세요. 그러나 스토리지가 저렴하고, 각 문서가 조인할 필요가 없는 단순 키-값 쿼리를 사용하여 빠르게 검색할 수 있는 단일 거래를 완전히 기록하기 때문에 괜찮습니다.

또한 신용 카드 정보라는 두 문서 간에도 명백한 차이가 있습니다. 각각의 구매에 대해 다른 신용카드를 사용한 것 같으므로 이러한 분명한 차이는 유일합니다. 각 문서는 문서화된 거래에 대해 정확합니다.

문서 작업

문서 데이터베이스로서 Amazon DocumentDB는 JSON 데이터를 쉽게 저장, 쿼리 및 인덱싱할 수 있습니다. Amazon DocumentDB에서 컬렉션은 모든 문서에 적용되는 단일 스키마가 없다는 점을 제외하면 관계형 데이터베이스의 테이블과 유사합니다. 모음을 통해 유사한 문서를 그룹화하여 모두 동일한 데이터베이스에 유지할 수 있으며, 구조가 동일하지 않아도 됩니다.

이전 단원의 예제 문서를 사용할 경우 `reading_material` 및 `office_supplies`에 대한 모음이 있을 수 있습니다. 소프트웨어에서는 문서가 속할 모음을 결정합니다.

다음 예제에서는 MongoDB API를 사용하여 문서를 추가, 쿼리, 업데이트 및 삭제하는 방법을 보여줍니다.

주제

- 문서 추가
- 문서 쿼리
- 문서 업데이트
- 문서 삭제

문서 추가

Amazon DocumentDB에서는 컬렉션에 문서를 처음 추가할 때 데이터베이스가 생성됩니다. 이 예제에서는 클러스터에 연결할 때 기본 데이터베이스인 `test` 데이터베이스에 `example`이라는 컬렉션을 만듭니다. 첫 번째 문서가 삽입될 때 컬렉션이 암시적으로 만들어지기 때문에 컬렉션 이름에 대한 오류 검사가 수행되지 않습니다. 따라서 컬렉션 이름에 오타(예: `example` 대신 `eexample`)가 생성되고 의도한 컬렉션이 아닌 `eexample` 컬렉션에 문서가 추가됩니다. 오류 확인은 애플리케이션을 통해 처리되어야 합니다.

다음 예제에서는 MongoDB API를 사용하여 문서를 추가합니다.

주제

- 단일 문서 추가
- 여러 문서 추가

단일 문서 추가

모음에 단일 문서를 추가하려면 모음에 추가할 문서에 `insertOne( {} )` 작업을 사용합니다.

```
db.example.insertOne(
    {
        "Item": "Ruler",
        "Colors": ["Red","Green","Blue","Clear","Yellow"],
        "Inventory": {
            "OnHand": 47,
            "MinOnHand": 40
        },
        "UnitPrice": 0.89
    }
)
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "acknowledged" : true,
    "insertedId" : ObjectId("5bedafbcf65ff161707de24f")
}
```

여러 문서 추가

모음에 여러 문서를 추가하려면 모음에 추가할 문서 목록에 `insertMany( [{},...,{}] )` 작업을 사용합니다. 이 특정 목록의 문서에 다른 스키마가 있는 경우에도 모두 동일한 모음에 추가할 수 있습니다.

```
db.example.insertMany(
    [
        {
            "Item": "Pen",
            "Colors": ["Red","Green","Blue","Black"],
            "Inventory": {
                "OnHand": 244,
```

```
                "MinOnHand": 72
            }
        },
        {
            "Item": "Poster Paint",
            "Colors": ["Red","Green","Blue","Black","White"],
            "Inventory": {
                "OnHand": 47,
                "MinOnHand": 50
            }
        },
        {
            "Item": "Spray Paint",
            "Colors": ["Black","Red","Green","Blue"],
            "Inventory": {
                "OnHand": 47,
                "MinOnHand": 50,
                "OrderQnty": 36
            }
        }
    ]
)
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "acknowledged" : true,
    "insertedIds" : [
            ObjectId("5bedb07941ca8d9198f5934c"),
            ObjectId("5bedb07941ca8d9198f5934d"),
            ObjectId("5bedb07941ca8d9198f5934e")
    ]
}
```

문서 쿼리

이따금 판매하는 물품을 고객이 보고 구매할 수 있도록 온라인 상점의 재고를 조회해야 할 수도 있습니다. 컬렉션을 쿼리하는 것은 컬렉션의 모든 문서에 대한 것이든 특정 기준을 충족하는 문서에만 대한 것이든 관계없이 상대적으로 간단합니다.

문서를 쿼리하려면 `find()` 작업을 사용합니다. `find()` 명령에는 반환할 문서 선택 시 사용할 기준을 정의하는 단일 문서 파라미터가 있습니다. `find()`의 출력은 줄 바꿈이 없이 한 줄로 된 텍스트 형식의

문서입니다. 쉽게 읽기 위해 출력 문서의 형식을 지정하려면 `find().pretty()`를 사용하십시오. 이 주제의 모든 예제는 `.pretty()`를 사용하여 출력 형식을 지정합니다.

앞선 두 가지 연습 `insertOne()` 및 `insertMany()`에서 `example` 컬렉션에 삽입한 4개의 문서를 사용합니다.

주제

- 컬렉션의 모든 문서 검색
- 피드 값과 일치하는 문서 검색
- 내장 문서와 일치하는 문서 검색
- 내장 문서의 필드 값과 일치하는 문서 검색
- 배열이 일치하는 문서 검색
- 배열의 값과 일치하는 문서 검색
- 연산자를 사용하여 문서 검색

컬렉션의 모든 문서 검색

모음의 모든 문서를 검색하려면 빈 쿼리 문서로 `find()` 작업을 사용하십시오.

다음 쿼리는 `example` 모음의 모든 문서를 반환합니다.

```
db.example.find( {} ).pretty()
```

피드 값과 일치하는 문서 검색

필드 및 값과 일치하는 모든 문서를 검색하려면 일치하는 필드 및 값을 식별하는 쿼리 문서로 `find()` 작업을 사용합니다.

이전 문서를 사용하면 이 쿼리는 "Item" 필드가 "Pen"과 동일한 모든 문서를 반환합니다.

```
db.example.find( { "Item": "Pen" } ).pretty()
```

내장 문서와 일치하는 문서 검색

내장 문서와 일치하는 모든 문서를 찾으려면 내장 문서 이름과 내장 문서의 모든 필드 및 값을 지정하는 쿼리 문서와 함께 `find()` 작업을 사용합니다.

내장 문서를 일치시킬 때 문서의 내장 문서는 쿼리에 있는 것과 동일한 이름을 가져야 합니다. 또한, 내장 문서의 필드 및 값은 쿼리와 일치해야 합니다.

다음 쿼리는 "Poster Paint" 문서만 반환합니다. "Pen"에는 "OnHand" 및 "MinOnHand"에 대한 다른 값이 있으며, "Spray Paint"에는 쿼리 문서보다 필드 하나(OrderQnty)가 더 있기 때문입니다.

```
db.example.find({"Inventory": {
    "OnHand": 47,
    "MinOnHand": 50 } } ).pretty()
```

내장 문서의 필드 값과 일치하는 문서 검색

내장 문서와 일치하는 모든 문서를 찾으려면 내장 문서 이름과 내장 문서의 모든 필드 및 값을 지정하는 쿼리 문서와 함께 `find()` 작업을 사용합니다.

이전 문서에서 다음 쿼리는 "점 표기법"을 사용하여 내장 문서와 관심 있는 필드를 지정합니다. 다른 필드가 내장 문서에 표시될 수 있는지 여부와 무관하게 이와 일치하는 모든 문서가 반환됩니다. "Poster Paint" 및 "Spray Paint"가 지정된 필드 및 값과 일치하므로 쿼리가 "Poster Paint" 및 "Spray Paint"를 반환합니다.

```
db.example.find({"Inventory.OnHand": 47, "Inventory.MinOnHand": 50 }).pretty()
```

배열이 일치하는 문서 검색

배열이 일치하는 모든 문서를 찾으려면 관심 있는 배열 이름과 해당 배열의 모든 값을 포함하여 `find()` 작업을 사용합니다. 쿼리가 배열 값이 동일하면서 쿼리와 동일한 순서인 해당 이름을 가진 배열을 포함한 모든 문서를 반환합니다.

"Poster Paint"에는 추가 색상(White)이 있고 "Spray Paint"에는 색상이 다른 순서로 있으므로 다음 쿼리는 "Pen"만을 반환합니다.

```
db.example.find( { "Colors": ["Red","Green","Blue","Black"] } ).pretty()
```

배열의 값과 일치하는 문서 검색

특정 배열 값을 보유한 모든 문서를 찾으려면 관심 있는 배열 이름과 값을 포함하여 `find()` 작업을 사용합니다.

```
db.example.find( { "Colors": "Red" } ).pretty()
```

각각 Colors라는 배열과 배열 내에 "Red" 값이 있으므로 이전 작업은 세 문서 모두를 반환합니다. "White" 값을 지정한 경우 쿼리는 "Poster Paint"만 반환합니다.

연산자를 사용하여 문서 검색

다음 쿼리는 "Inventory.OnHand" 값이 50 미만인 모든 문서를 반환합니다.

```
db.example.find(
        { "Inventory.OnHand": { $lt: 50 } } )
```

지원되는 쿼리 연산자 목록은 쿼리 및 프로젝션 연산자 섹션을 참조하십시오.

문서 업데이트

일반적으로 문서는 정적이 아니며 애플리케이션 워크플로의 일부로 업데이트됩니다. 다음은 문서를 업데이트할 수 있는 몇 가지 방법을 보여주는 예입니다.

기존 문서를 업데이트하려면 update() 작업을 사용합니다. update() 작업에는 두 개의 문서 파라미터가 있습니다. 첫 번째 문서는 업데이트할 문서를 식별합니다. 두 번째 문서는 업데이트를 지정합니다.

기존 필드를 업데이트할 때 (해당 필드가 단순 필드이든, 배열이든, 포함된 문서이든) 필드 이름과 해당 값을 지정합니다. 작업 종료 시 이전 문서의 필드가 새 필드와 값으로 교체된 것과 같습니다.

주제

- 기존 필드의 값 업데이트
- 새 필드 추가
- 내장 문서 교체
- 내장 문서에 새 필드 삽입
- 문서에서 필드 제거
- 여러 문서에서 필드 제거

기존 필드의 값 업데이트

다음 업데이트 작업에 대해 추가한 4개의 문서를 사용합니다.

```
{
    "Item": "Ruler",
    "Colors": ["Red","Green","Blue","Clear","Yellow"],
```

```
    "Inventory": {
        "OnHand": 47,
        "MinOnHand": 40
    },
    "UnitPrice": 0.89
},
{
    "Item": "Pen",
    "Colors": ["Red","Green","Blue","Black"],
    "Inventory": {
        "OnHand": 244,
        "MinOnHand": 72
    }
},
{
    "Item": "Poster Paint",
    "Colors": ["Red","Green","Blue","Black","White"],
    "Inventory": {
        "OnHand": 47,
        "MinOnHand": 50
    }
},
{
    "Item": "Spray Paint",
    "Colors": ["Black","Red","Green","Blue"],
    "Inventory": {
        "OnHand": 47,
        "MinOnHand": 50,
        "OrderQnty": 36
    }
}
```

단순 필드 업데이트

단순 필드를 업데이트하려면 $set와 함께 update()를 사용하여 필드 이름과 새 값을 지정합니다. 다음 예제에서는 Item을 "Pen"에서 "Gel Pen"으로 바꿉니다.

```
db.example.update(
    { "Item" : "Pen" },
    { $set: { "Item": "Gel Pen" } }
)
```

이 작업의 결과는 다음과 같습니다.

```
{
    "Item": "Gel Pen",
    "Colors": ["Red","Green","Blue","Black"],
    "Inventory": {
        "OnHand": 244,
        "MinOnHand": 72
    }
}
```

배열 업데이트

다음 예제에서는 색상 목록에서 Orange를 포함하고 White를 제외한 새 배열로 기존 배열을 교체합니다. 새 색상 목록은 update() 작업에서 지정된 순서입니다.

```
db.example.update(
    { "Item" : "Poster Paint" },
    { $set: { "Colors": ["Red","Green","Blue","Orange","Black"] } }
)
```

이 작업의 결과는 다음과 같습니다.

```
{
    "Item": "Poster Paint",
    "Colors": ["Red","Green","Blue","Orange","Black"],
    "Inventory": {
        "OnHand": 47,
        "MinOnHand": 50
    }
}
```

새 필드 추가

하나 이상의 새 필드를 추가하여 문서를 수정하려면 삽입할 문서 및 $set 연산자를 사용하여 삽입할 새 필드와 값을 식별하는 쿼리 문서에 update() 작업을 사용합니다.

다음 예제에서는 Spray Paint 문서에 UnitPrice 필드와 3.99 값을 추가합니다. 3.99라는 값은 숫자이지 문자열이 아닙니다.

```
db.example.update(
    { "Item": "Spray Paint" },
    { $set: { "UnitPrice": 3.99 } }
```

```
)
```

이 작업의 결과는 다음과 같습니다(JSON 형식).

```
{
    "Item": "Spray Paint",
    "Colors": ["Black","Red","Green","Blue"],
    "Inventory": {
        "OnHand": 47,
        "MinOnHand": 50,
        "OrderQnty": 36
    },
    "UnitPrice": 3.99
}
```

내장 문서 교체

내장 문서를 교체함으로써 문서를 수정하려면 내장 문서 및 `$set` 연산자를 사용한 새 필드와 값을 식별하는 문서에 `update()` 작업을 사용합니다.

다음과 같은 문서가 있다고 가정합니다.

```
db.example.insert({
    "DocName": "Document 1",
    "Date": {
        "Year": 1987,
        "Month": 4,
        "Day": 18
    }
})
```

내장 문서 교체

다음 예제에서는 현재 Date 문서를 새로운 문서로 교체합니다. 새 문서에는 `Month` 및 `Day` 필드만 있고, `Year` 필드는 제거되었습니다.

```
db.example.update(
    { "DocName" : "Document 1" },
    { $set: { "Date": { "Month": 4, "Day": 18 } } }
)
```

이 작업의 결과는 다음과 같습니다.

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18
    }
}
```

내장 문서에 새 필드 삽입

내장 문서에 필드를 추가하려면

내장 문서에 하나 이상의 새 필드를 추가하여 문서를 수정하려면 내장 문서와 내장 문서 지정을 위한 "점 표기법" 및 `$set` 연산자를 사용하여 삽입할 새 필드와 값을 식별하는 문서에 `update()` 작업을 사용합니다.

다음과 같은 문서가 있을 때 다음 코드는 "점 표기법"을 사용하여 `Year` 및 `DoW` 필드를 내장 `Date` 문서에 삽입하고, `Words`를 상위 문서에 삽입합니다.

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18
    }
}
```

```
db.example.update(
    { "DocName" : "Document 1" },
    { $set: { "Date.Year": 1987,
              "Date.DoW": "Saturday",
              "Words": 2482 } }
)
```

이 작업의 결과는 다음과 같습니다.

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
```

```
        "Year": 1987,
        "DoW": "Saturday"
    },
    "Words": 2482
}
```

문서에서 필드 제거

문서에서 필드를 제거하여 문서를 수정하려면 쿼리 문서에 필드를 제거할 문서를 식별하는 update() 작업을 사용하고, 제거할 필드를 지정하는 $unset 연산자를 사용합니다.

다음 예제에서는 이전 문서에서 Words 필드를 제거합니다.

```
db.example.update(
    { "DocName" : "Document 1" },
    { $unset: { Words:1 } }
)
```

이 작업의 결과는 다음과 같습니다.

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DoW": "Saturday"
    }
}
```

여러 문서에서 필드 제거

여러 문서에서 필드를 제거하여 문서를 수정하려면 $unset 연산자와 multi 옵션 세트를 true로 설정하여 update() 작업을 사용합니다.

다음 예제에서는 예제 컬렉션의 모든 문서에서 Inventory 필드를 제거합니다. 문서에 Inventory 필드가 없으면 해당 문서에 어떤 작업도 수행되지 않습니다. multi: true가 생략되면 기준을 충족하는 첫 번째 문서에만 작업이 수행됩니다.

```
db.example.update(
```

```
    {},
    { $unset: { Inventory:1 } },
    { multi: true }
)
```

문서 삭제

데이터베이스에서 문서를 제거하려면 `remove()` 작업을 사용하여 제거할 문서를 지정합니다. 다음 코드는 `example` 모음에서 "Gel Pen"을 제거합니다.

```
db.example.remove( { "Item": "Gel Pen" } )
```

데이터베이스에서 모든 문서를 제거하려면 `remove()` 작업과 빈 쿼리를 사용합니다.

```
db.example.remove( { } )
```

Amazon DocumentDB로 시작하기

Amazon DocumentDB에 연결하고 이를 통해 시작하는 방법에는 여러 가지가 있습니다. 이 안내서는 사용자가 강력한 도큐먼트 데이터베이스를 사용하여 시작할 수 있는 가장 빠르고 간단하며 쉬운 방법입니다. 이 가이드에서는 AWS CloudShell을 사용하여 AWS Management Console에서 직접 Amazon DocumentDB 클러스터를 연결하고 쿼리합니다. AWS 프리 티어 자격이 있는 신규 고객은 Amazon DocumentDB 및 CloudShell을 무료로 사용할 수 있습니다. AWS CloudShell 환경 또는 Amazon DocumentDB 클러스터가 프리 티어 이외의 리소스를 사용하는 경우 해당 리소스에 대한 일반 AWS 요금이 부과됩니다. 이 안내서를 통해 5분 이내에 Amazon DocumentDB를 시작할 수 있습니다.

> **Note**
>
> 이 가이드의 지침은 특히 Amazon DocumentDB 및를 사용할 AWS CloudShell 수 있는 Amazon DocumentDB 인스턴스 기반 클러스터를 생성하고 연결하는 데 사용됩니다.
>
> - Amazon DocumentDB 탄력성 클러스터를 생성하고 연결하려면 Amazon DocumentDB 엘라스틱 클러스터 시작하기을 참조하십시오.
> - AWS 중국 리전에 있는 경우 섹션을 참조하세요Amazon EC2 자동 연결.

주제

- 사전 조건
- 1단계: 클러스터 생성
- 2단계: 클러스터에 연결
- 3단계: 데이터 삽입 및 쿼리
- 4단계: 탐색
- Amazon DocumentDB 빠른 시작 사용 CloudFormation

사전 조건

첫 번째 Amazon DocumentDB 클러스터를 작성하기 전에 다음 작업을 수행해야 합니다:

Amazon Web Services(AWS) 계정 만들기

Amazon DocumentDB 사용을 시작하기 전에 Amazon Web Services(AWS) 계정이 있어야 합니다. AWS 계정은 무료입니다. 사용하는 서비스 및 리소스에 대해서만 비용을 지불하는 것입니다.

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://signin.aws.amazon.com/signup?request_type=register를 엽니다.
2. 온라인 지시 사항을 따르세요.

   등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

필요한 AWS Identity and Access Management (IAM) 권한을 설정합니다.

클러스터, 인스턴스 및 클러스터 파라미터 그룹과 같은 Amazon DocumentDB 리소스를 관리하기 위한 액세스에는가 요청을 인증하는 데 사용할 AWS 수 있는 자격 증명이 필요합니다. 자세한 내용은 Amazon DocumentDB의 ID 및 액세스 관리 섹션을 참조하세요.

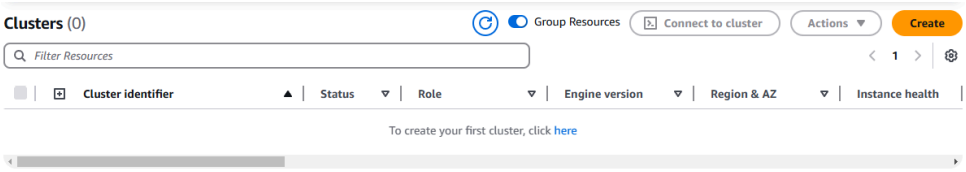
1. 의 검색 창에 나타나는 드롭다운 메뉴에서 IAM을 AWS Management Console**IAM** 입력하고 선택합니다.
2. IAM 콘솔에 들어가면 탐색 창에서 사용자를 선택합니다.
3. 사용자 이름을 선택합니다.
4. 권한 추가를 선택합니다.
5. 정책 직접 연결을 선택합니다.
6. 검색 창에 `AmazonDocDBFullAccess`을 입력하고 검색 결과에 표시되면 선택합니다.
7. 다음(Next)을 선택합니다.
8. 권한 추가를 선택합니다.

> **Note**
>
> AWS 계정에는 각 리전의 기본 VPC가 포함됩니다. Amazon VPC를 사용하기로 선택한 경우 Amazon VPC 사용 설명서에서 Amazon VPC 생성 주제의 단계를 완료합니다.

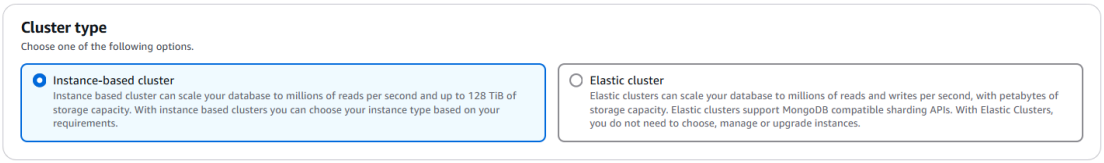
1단계: 클러스터 생성

이 단계에서 Amazon DocumentDB 클러스터를 생성합니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. Amazon DocumentDB 관리 콘솔의 클러스터에서 생성을 선택합니다.

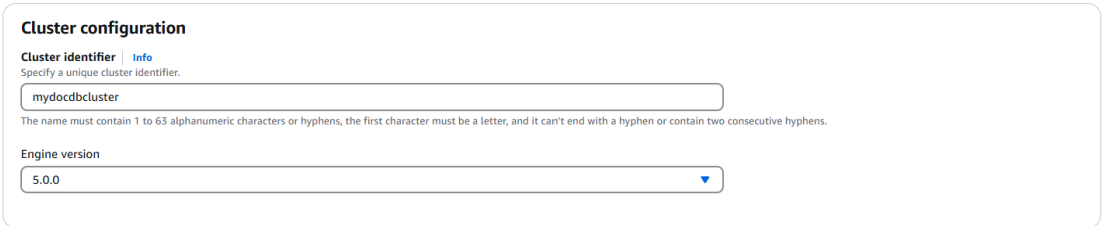


3. Amazon DocumentDB 클러스터 생성 페이지의 클러스터 유형 섹션에서 인스턴스 기반 클러스터(기본 옵션)를 선택합니다.

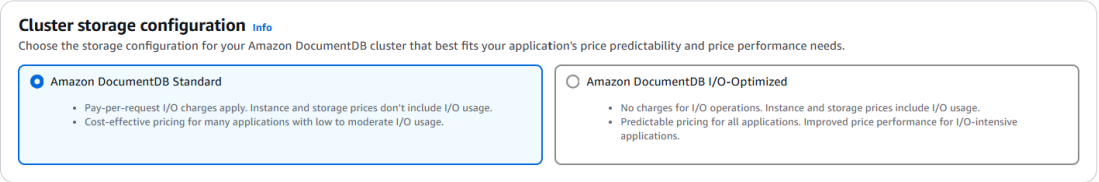


> **Note**
>
> 이 범주의 다른 옵션은 탄력적 클러스터입니다. Amazon DocumentDB 탄력적 클러스터에 대한 자세한 내용은 Amazon DocumentDB 탄력적 클러스터 사용 섹션을 참조하세요.

4. 클러스터 구성 섹션에서:
   a. 클러스터 식별자 에 **`mydocdbcluster`**와 같은 고유한 이름을 입력합니다. 콘솔은 모든 클러스터 이름을 입력 방식에 관계없이 소문자로 변경합니다.
   b. 엔진 버전의 경우 5.0.0을 선택합니다.



5. 클러스터 스토리지 구성 섹션에서 Amazon DocumentDB 표준(기본 옵션입니다)을 선택합니다.



> **Note**
>
> 이 범주의 다른 옵션은 Amazon DocumentDB I/O 최적화입니다. 옵션에 대해 자세히 알아보려면 Amazon DocumentDB 클러스터 스토리지 구성 섹션을 참조하세요.

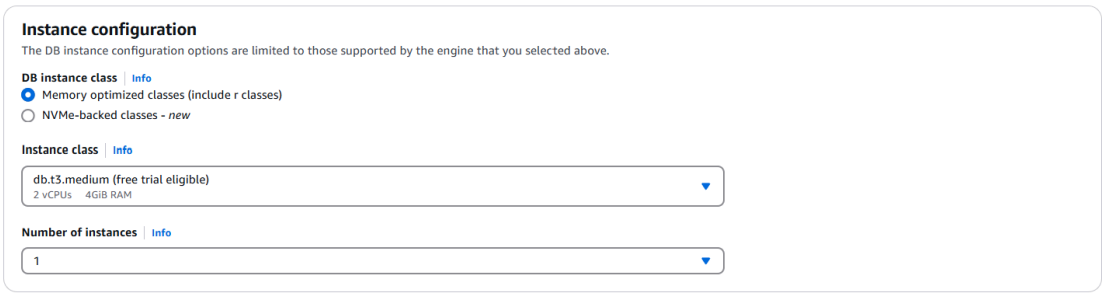
6. 인스턴스 구성 섹션에서:

   a. DB 인스턴스 클래스에서 메모리 최적화 클래스(r 클래스 포함)(기본값)를 선택합니다.

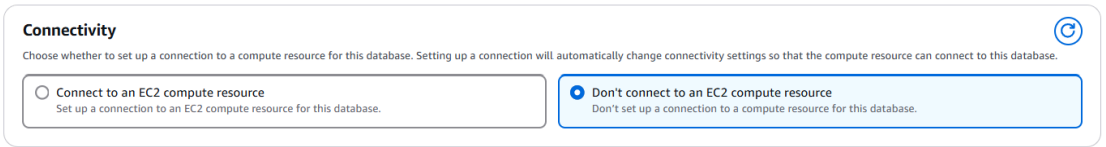
      다른 인스턴스 옵션은 NVMe 지원 클래스입니다. 자세한 내용은 NVMe 지원 인스턴스 섹션을 참조하세요.

   b. 인스턴스 클래스에서 db.t3.medium을 선택합니다. 무료 AWS 평가판을 사용할 수 있습니다.

   c. 인스턴스 수에 1 인스턴스를 선택합니다. 인스턴스 하나를 선택하면 비용을 최소화하는 데 도움이 됩니다. 프로덕션 시스템인 경우 고가용성을 위해 세 개의 인스턴스를 프로비저닝하는 것이 좋습니다.

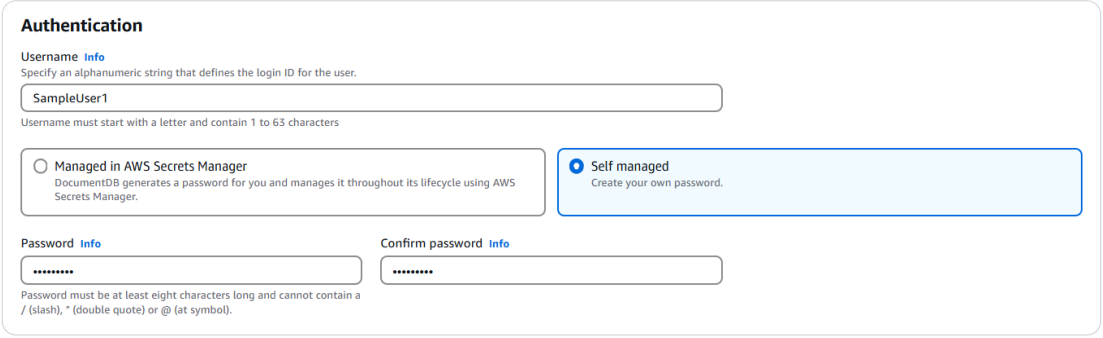


7. 연결 섹션에서 EC2 컴퓨팅 리소스에 연결 안 함을 기본 설정으로 둡니다.



8. 인증 섹션에 기본 사용자의 사용자 이름을 입력한 후 자체 관리형을 선택합니다. 암호를 입력한 다음 확인합니다.

대신 Managed in AWS Secrets Manager을 선택한 경우 자세한 내용은 Amazon DocumentDB 및 를 사용한 암호 관리 AWS Secrets Manager 섹션을 참조하세요.



9. 다른 모든 옵션을 기본값으로 두고 클러스터 생성을 선택합니다.

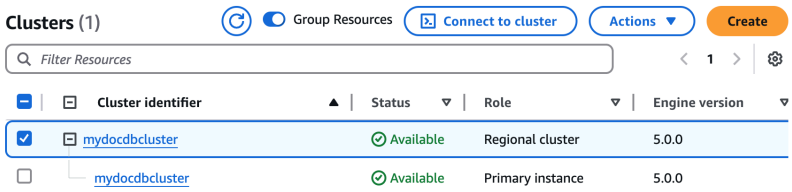
Amazon DocumentDB는 이제 클러스터를 프로비저닝하고 있으며, 완료하는 데 몇 분 정도 걸릴 수 있습니다.

> **Note**
>
> 클러스터 상태 값에 대한 자세한 내용은 Amazon DocumentDB 모니터링의 클러스터 상태 값(을)를 참조하십시오.

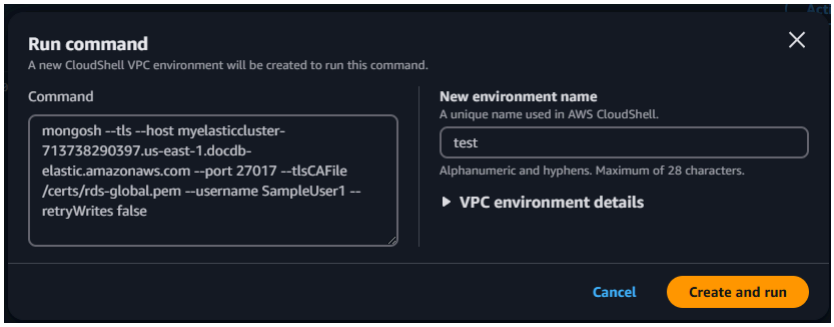
2단계: 클러스터에 연결

를 사용하여 Amazon DocumentDB 클러스터에 연결합니다 AWS CloudShell.

1. Amazon DocumentDB 관리 콘솔의 클러스터에서 생성한 클러스터를 찾습니다. 클러스터 옆의 확인란을 클릭하여 클러스터를 선택합니다.



2. 클러스터에 연결을 선택합니다(작업 드롭다운 메뉴 옆). 이 버튼은 클러스터 옆의 확인란을 클릭하고 리전 클러스터와 기본 인스턴스의 상태가 모두 사용 가능으로 표시된 후에만 활성화됩니다. CloudShell 명령 실행 화면이 나타납니다.
3. 새 환경 이름 필드에 '테스트'와 같은 고유한 이름을 입력하고 생성 및 실행을 선택합니다. VPC 환경 세부 정보는 Amazon DocumentDB 데이터베이스에 대해 자동으로 구성됩니다.



4. 메시지가 표시되면 1단계: Amazon DocumentDB 클러스터 생성(하위 7단계)에서 생성한 암호를 입력합니다.

   암호를 입력한 후 프롬프트에 `rs0 [direct: primary] <env-name>>`이 표시되면 Amazon DocumentDB 클러스터에 성공적으로 연결된 것입니다.

> **Note**
>
> 흐름 문제 해결에 대한 자세한 내용은 Amazon DocumentDB 문제 해결을 참조하십시오.

3단계: 데이터 삽입 및 쿼리

이제 클러스터에 연결되었으므로 몇 가지 쿼리를 실행하여 문서 데이터베이스 사용에 익숙해질 수 있습니다.

1. 단일 문서를 삽입하려면 다음을 입력합니다:

```
db.collection.insertOne({"hello":"DocumentDB"})
```

   출력은 다음과 같습니다:

```
{
```

```
  acknowledged: true,
  insertedId: ObjectId('673657216bdf6258466b128c')
}
```

2. findOne() 명령으로 작성한 문서를 읽을 수 있습니다 (단일 문서만 반환하기 때문). 다음 내용을 입력합니다.

```
db.collection.findOne()
```

다음과 같은 출력내용을 얻게 됩니다.

```
{ "_id" : ObjectId("5e401fe56056fda7321fbd67"), "hello" : "DocumentDB" }
```

3. 쿼리를 몇 개 더 수행하려면 게임 프로필 사용 사례를 고려해 보세요. 먼저 profiles이라는 제목의 컬렉션에 몇 개의 항목을 삽입합니다. 다음 내용을 입력합니다.

```
db.profiles.insertMany([{ _id: 1, name: 'Matt', status: 'active', level: 12, score:
 202 },
      { _id: 2, name: 'Frank', status: 'inactive', level: 2, score: 9 },
      { _id: 3, name: 'Karen', status: 'active', level: 7, score: 87 },
      { _id: 4, name: 'Katie', status: 'active', level: 3, score: 27 }
])
```

다음과 같은 출력내용을 얻게 됩니다.

```
{ acknowledged: true, insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4 } }
```

4. find() 명령을 사용하여 프로필 컬렉션의 모든 문서를 반환합니다. 다음 내용을 입력합니다.

```
db.profiles.find()
```

3단계에서 입력한 데이터와 일치하는 출력을 받게 됩니다.

5. 필터를 사용하여 단일 문서에 대한 쿼리를 사용하세요. 다음을 입력합니다:

```
db.profiles.find({name: "Katie"})
```

다음과 같은 출력내용을 얻게 됩니다.

```
{ "_id" : 4, "name" : "Katie", "status": "active", "level": 3, "score":27}
```

6. 이제 프로필을 찾아 `findAndModify` 명령을 사용하여 해당 프로필을 수정해 보겠습니다. 다음 코드를 사용하여 사용자 Matt에게 10점을 추가로 주겠습니다.

```
db.profiles.findAndModify({
        query: { name: "Matt", status: "active"},
        update: { $inc: { score: 10 } }
    })
```

다음과 같은 결과가 출력됩니다 (참고로 그의 점수는 아직 오르지 않았습니다):

```
{
    [{_id : 1, name : 'Matt', status: 'active', level: 12, score: 202}]
```

7. 다음 쿼리를 통해 그의 점수가 변경되었는지 확인할 수 있습니다:

```
db.profiles.find({name: "Matt"})
```

출력은 다음과 같습니다.

```
{ "_id" : 1, "name" : "Matt", "status" : "active", "level" : 12, "score" : 212 }
```

4단계: 탐색

축하합니다! Amazon DocumentDB 인스턴스 기반 클러스터에 대한 시작 안내서를 성공적으로 완료했습니다.

다음 단계? 몇 가지 인기 있는 기능을 통해 이 데이터베이스를 최대한 활용하는 방법을 알아보십시오:

- Amazon DocumentDB 관리
- 스케일링
- 백업 및 복원

> **Note**
>
> 이 시작 연습을 통해 생성한 클러스터는 해당 내용을 삭제하지 않는 한 계속 비용이 누적됩니다. 자세한 지침은 Amazon DocumentDB 클러스터 삭제를 참조하십시오.

Amazon DocumentDB 빠른 시작 사용 CloudFormation

이 섹션에는 CloudFormation을 사용하여 Amazon DocumentDB(MongoDB 호환)를 신속하게 시작하는 데 도움이 되는 단계 및 기타 정보가 포함되어 있습니다. Amazon DocumentDB 엔드포인트에 관한 일반 정보는 Amazon DocumentDB(MongoDB 호환)란 무엇인가를 참조하세요.

이 지침은 AWS CloudFormation 템플릿을 사용하여 기본 Amazon VPC에서 클러스터와 인스턴스를 생성합니다. 이러한 리소스를 사용자가 직접 생성하는 방법에 관한 지침은 Amazon DocumentDB로 시작하기(을)를 참조하십시오.

> **Important**
>
> 이 템플릿으로 생성된 CloudFormation 스택은 Amazon DocumentDB(예: 클러스터 및 인스턴스) 및 Amazon Elastic Compute Cloud(예: 서브넷 그룹)의 리소스를 포함하여 여러 리소스를 생성합니다.
> 이 리소스 중 일부는 프리 티어 리소스가 아닙니다. 가격 정보는 Amazon DocumentDB 가격 및 Amazon EC2 가격를 참조하십시오. 이 단원을 완료한 후에는 스택을 삭제하여 요금이 발생하지 않게 할 수 있습니다.

이 CloudFormation 스택은 자습서 목적으로만 사용됩니다. 프로덕션 환경에이 템플릿을 사용하는 경우 더 엄격한 IAM 정책 및 보안을 사용합니다. 리소스 보안에 대한 자세한 내용은 Amazon VPC 시큐리티 및 Amazon EC2 네트워크 및 시큐리티를 참조하십시오.

주제

- 사전 조건
- Amazon DocumentDB CloudFormation 스택 시작
- Amazon DocumentDB 클러스터 액세스
- 종료 방지 및 삭제 방지

사전 조건

Amazon DocumentDB 클러스터를 작성하기 전에 다음 항목이 있어야 합니다.

- 기본 Amazon VPC
- 필요한 IAM 권한

필요한 IAM 권한

다음 권한을 통해 CloudFormation 스택에 리소스를 생성할 수 있습니다:

AWS 관리형 정책

- `AWSCloudFormationReadOnlyAccess`
- `AmazonDocDBFullAccess`

추가 IAM 권한

다음 정책은이 CloudFormation 스택을 생성하고 삭제하는 데 필요한 추가 권한을 간략하게 설명합니다.

다음 예제에서는 각각의 `### ## ## ###`를 자신의 리소스 정보로 바꿉니다.

Amazon EC2 키 페어

CloudFormation 스택을 생성할 리전에서 키 페어(및 PEM 파일)를 사용할 수 있어야 합니다. 키 페어를 만들어야 할 경우, Amazon EC2 사용 설명서의 Amazon EC2를 사용하여 키 페어 만들기를 참조하세요.

Amazon DocumentDB CloudFormation 스택 시작

이 섹션에서는 Amazon DocumentDB CloudFormation 스택을 시작하고 구성하는 방법에 대해 설명합니다.

1. 에서에 로그인 AWS Management Console 합니다https://console.aws.amazon.com/.
2. 다음 표는 각 AWS 리전에 대한 Amazon DocumentDB 스택 템플릿을 나열합니다. 스택을 시작하려는 에 대해 스택 시작을 선택합니다. AWS 리전

리전	템플릿 보기	Designer에서 보기	시작
미국 동부(오하이오)	템플릿 보기	Designer에서 보기	Launch Stack
미국 동부(버지니아 북부)	템플릿 보기	Designer에서 보기	Launch Stack

리전	템플릿 보기	Designer에서 보기	시작
미국 서부(오레곤)	템플릿 보기	Designer에서 보기	Launch Stack
아시아 태평양(뭄바이)	템플릿 보기	Designer에서 보기	Launch Stack
아시아 태평양(서울)	템플릿 보기	Designer에서 보기	Launch Stack
아시아 태평양(싱가포르)	템플릿 보기	Designer에서 보기	Launch Stack
아시아 태평양(시드니)	템플릿 보기	Designer에서 보기	Launch Stack
아시아 태평양(도쿄)	템플릿 보기	Designer에서 보기	Launch Stack
캐나다(중부)	템플릿 보기	Designer에서 보기	Launch Stack
유럽(프랑크푸르트)	템플릿 보기	Designer에서 보기	Launch Stack
유럽(아일랜드)	템플릿 보기	Designer에서 보기	Launch Stack
유럽(런던)	템플릿 보기	Designer에서 보기	Launch Stack
유럽(파리)	템플릿 보기	Designer에서 보기	Launch Stack

3. 스택 생성 — 선택한 Amazon DocumentDB 템플릿을 설명합니다. 모든 스택은 스택에 포함하려는 AWS 리소스에 대한 구성이 포함된 JSON 또는 YAML 파일인 템플릿을 기반으로 합니다. 위의 제공된 템플릿에서 스택을 시작하도록 선택했으므로 선택한에 대한 Amazon DocumentDB 스택을 생성하도록 템플릿이 이미 구성되어 AWS 리전 있습니다.

   CloudFormation 스택을 시작하면 Amazon DocumentDB 클러스터에 대한 삭제 보호가 기본적으로 비활성화됩니다. 클러스터에 대해 삭제 방지를 활성화하려면 다음 단계를 완료하십시오. 그렇지 않으면 다음을 선택하여 다음 단계를 계속하십시오.

Amazon DocumentDB 클러스터에 대해 삭제 보호를 사용하려면:

1. 스택 생성 페이지의 오른쪽 하단에서 Designer에서 보기를 선택합니다.
2. 콘솔의 결과 CloudFormation Designer 페이지에서 통합 JSON 및 YAML 편집기를 사용하여 템플릿을 수정합니다. 다음과 같이 `Resources` 섹션으로 스크롤하여 `DeletionProtection`을 포함하도록 수정합니다. CloudFormation Designer 사용에 대한 자세한 내용은 CloudFormation Designer란 무엇입니까?를 참조하세요.

   JSON:

```
"Resources": {
    "DBCluster": {
        "Type": "AWS::DocDB::DBCluster",
        "DeletionPolicy": "Delete",
        "Properties": {
            "DBClusterIdentifier": {
                "Ref": "DBClusterName"
            },
            "MasterUsername": {
                "Ref": "MasterUser"
            },
            "MasterUserPassword": {
                "Ref": "MasterPassword"
            },
            "DeletionProtection": "true"
        }
    },
```

   YAML:

```
Resources:
  DBCluster:
    Type: 'AWS::DocDB::DBCluster'
    DeletionPolicy: Delete
    Properties:
      DBClusterIdentifier: !Ref DBClusterName
      MasterUsername: !Ref MasterUser
      MasterUserPassword: !Ref MasterPassword
      DeletionProtection: 'true'
```

3. 페이지 왼쪽 위에서 스택 만들기(

)

를 선택하여 변경 사항을 저장하고 이러한 변경 사항이 활성화된 스택을 만듭니다.

4. 변경 사항을 저장하면 스택 만들기 페이지로 리디렉션됩니다.
5. 다음을 선택하여 계속 진행합니다.

4. 스택 상세 내역 지정 — 템플릿의 스택 이름과 매개 변수를 입력합니다. 파라미터는 템플릿에서 정의되며, 이를 통해 스택을 생성하거나 업데이트 할 때 사용자 지정 값을 입력할 수 있습니다.

- 스택 이름에서 스택의 이름을 입력하거나 제공된 이름을 그대로 사용합니다. 스택 이름에는 문자(A—Z 및 a—z), 숫자(0-9) 및 대시(—)를 포함할 수 있습니다.
- 파라미터에서 다음 세부 정보를 입력합니다:
  - DBC클러스터 이름 — Amazon DocumentDB 클러스터의 이름을 입력하거나 제공된 이름을 사용합니다.

    클러스터 명명 제약 조건:

    - 길이는 [1-63]개의 글자, 숫자 또는 하이픈입니다.
    - 첫 번째 문자는 글자이어야 합니다.
    - 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.
    - 리전별로 Amazon RDS, Neptune 및 Amazon DocumentDB의 모든 클러스터 AWS 계정에 대해 고유해야 합니다.
  - DBInstanceClass — 드롭다운 목록에서 Amazon DocumentDB 클러스터의 인스턴스 클래스를 선택합니다.
  - DBInstanceName — Amazon DocumentDB 인스턴스의 이름을 입력하거나 제공된 이름을 사용합니다.

    인스턴스 명명 제약 조건:

    - 길이는 [1-63] 문자, 숫자 또는 하이픈입니다.
    - 첫 번째 문자는 글자이어야 합니다.
    - 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.
    - 리전별로 Amazon RDS, Neptune 및 Amazon DocumentDB의 모든 인스턴스 AWS 계정에 대해 고유해야 합니다.
  - MasterPassword — 데이터베이스 관리자 계정 암호입니다.
  - MasterUser — 데이터베이스 관리자 계정 사용자 이름입니다. MasterUser는 문자로 시작해야 하며 영숫자만 포함할 수 있습니다.

다음을 선택하여 변경 사항을 저장하고 계속합니다.

5. 스택옵션 구성 — 스택의 태그, 권한 및 추가 옵션을 구성합니다.

   - 태그 — 스택의 리소스에 적용할 태그(키-값) 쌍을 지정합니다. 각 스택에 대해 최대 50개의 고유 태그를 추가할 수 있습니다.
   - 권한 — 선택 사항. IAM 역할을 선택하여가 스택에서 리소스를 생성, 수정 또는 삭제 CloudFormation 하는 방법을 명시적으로 정의합니다. 역할을 선택하지 않으면는 사용자 자격 증명을 기반으로 권한을 CloudFormation 사용합니다. 서비스 역할을 지정하기 전에 역할을 전달할 수 있는 권한이 있는지 확인합니다(`iam:PassRole`). `iam:PassRole` 권한은 사용 가능한 역할을 지정합니다.

     > **Note**
     >
     > 서비스 역할을 지정하면는 CloudFormation 항상 해당 스택에서 수행되는 모든 작업에 해당 역할을 사용합니다. 이 스택에서 작업을 수행할 수 있는 권한이 있는 다른 사용자는 이 역할을 전달할 수 있는 권한이 없더라도 이 역할을 사용할 수 있습니다. 역할에 사용자에게 불필요한 권한이 포함되어 있는 경우 사용자의 권한을 실수로 에스컬레이션할 수 있습니다. 이 역할은 최소 권한을 부여해야 합니다.

   - 고급 옵션 — 다음 고급 옵션을 설정할 수 있습니다:

     - 스택 정책 — 선택 사항. 스택 업데이트 중 의도치 않게 업데이트되지 않도록 하려는 리소스를 정의합니다. 기본적으로 스택 업데이트 중에는 모든 리소스가 업데이트될 수 있습니다.

       스택 정책을 바로 JSON으로 입력하거나 스택 정책이 포함된 JSON 파일을 업로드할 수 있습니다. 자세한 내용은 스택 리소스에 대한 업데이트 방지를 참조하십시오.
     - 롤백 구성 — 선택 사항. 스택 CloudFormation 을 생성하고 업데이트할 때 모니터링할에 대한 CloudWatch Logs 경보를 지정합니다. 작업이 경보 임계값을 위반하면는 경보 임계값을 CloudFormation 롤백합니다.
     - 알림 옵션 — 선택 사항. Simple Notification System(SNS)에 대한 주제를 지정합니다.
     - 스택 생성 옵션 — 선택 사항. 다음과 같은 옵션을 지정할 수 있습니다:
       - 실패 시 롤백 — 스택 생성이 실패한 경우 스택을 롤백해야 하는지 여부입니다.
       - 제한시간 —스택 생성 시간이 초과되기 전까지의 시간(분)입니다.
       - 종료 보호 — 스택이 실수로 삭제되는 것을 방지합니다.

> **Note**
>
> CloudFormation 종료 방지는 삭제 방지라는 Amazon DocumentDB 개념과 다릅니다. 자세한 내용은 종료 방지 및 삭제 방지 섹션을 참조하세요.

다음을 선택하여 계속 진행합니다.

6. 리뷰 <stack-name> — 스택 템플릿, 세부 정보 및 구성 옵션을 검토합니다. 페이지 하단에서 빠른 생성 링크를 열어 이와 동일한 기본 구성을 가진 스택을 만들 수도 있습니다.

   - 생성을 선택하여 스택을 생성합니다.
   - 또는 변경 세트 만들기를 선택할 수 있습니다. 변경 세트는 스택을 만들기 전에 이 스택을 구성하는 방법에 대한 미리 보기입니다. 이렇게 하면 변경 세트를 실행하기 전에 다양한 구성을 검사할 수 있습니다.

Amazon DocumentDB 클러스터 액세스

CloudFormation 스택이 완료되면 Amazon EC2 인스턴스를 사용하여 Amazon DocumentDB 클러스터에 연결할 수 있습니다. SSH를 사용하여 Amazon EC2 인스턴스에 연결하는 방법에 대한 자세한 내용은 Amazon EC2 사용자 설명서의 Linux 인스턴스에 연결을 참조하십시오.

연결된 후에는 Amazon DocumentDB 사용에 대한 내용이 포함된 다음 섹션을 참조하십시오.

- 3단계: 데이터 삽입 및 쿼리
- Amazon DocumentDB 클러스터 및 인스턴스 관리
- Amazon DocumentDB에서 모니터링 및 로깅

종료 방지 및 삭제 방지

삭제 보호 및 종료 보호를 가능하게 하는 Amazon DocumentDB 모범 사례입니다. CloudFormation 종료 보호는 Amazon DocumentDB 삭제 보호 기능과는 확연히 다른 기능입니다.

- 종료 보호 — CloudFormation 스택에 대한 종료 보호 기능을 사용하여 스택이 실수로 삭제되는 것을 방지할 수 있습니다. 종료료 보호 기능이 활성화된 스택을 삭제하려고 시도하면 삭제가 실패하고 스택은 변함없이 그대로 유지됩니다. CloudFormation을 사용하여 스택을 만들 때 기본적으로 종료 보호 기능이 비활성화됩니다. 스택을 생성할 때 종료 보호 기능을 활성화할 수 있습니다. 자세한 내용은 CloudFormation 스택 옵션 설정을 참조하세요.

- 삭제 보호 — 또한 Amazon DocumentDB는 클러스터에 대해 삭제 보호를 실행하는 기능을 제공합니다. 사용자가 삭제 보호가 설정된 Amazon DocumentDB 클러스터를 삭제하려고 할 경우 삭제가 실패하고 클러스터는 변경되지 않습니다. 삭제 방지 기능을 활성화하면 Amazon DocumentDB AWS Management Console, AWS CLI및 CloudFormation에서 실수로 삭제되는 것을 방지할 수 있습니다. Amazon DocumentDB 클러스터에 대한 삭제 방지 사용 및 사용 안 함에 대한 자세한 내용은 삭제 방지을 참조하십시오.

Amazon DocumentDB 마이그레이션 및 업그레이드

Amazon DocumentDB는 MongoDB API와 호환되는 완전관리형 데이터베이스 서비스입니다. 이 섹션에서 설명하는 프로세스를 사용하여 온프레미스 또는 Amazon Elastic Compute Cloud(Amazon EC2)에서 실행 중인 MongoDB 데이터베이스에서 Amazon DocumentDB로 데이터를 마이그레이션할 수 있습니다.

주제

- AWS Database Migration Service(DMS)를 사용하여 Amazon DocumentDB로 마이그레이션: 빠른 시작 안내서
- Amazon DocumentDB 마이그레이션 런북
- Couchbase Server에서 마이그레이션
- Amazon DocumentDB 엔진 버전 지원 날짜
- Amazon DocumentDB 인플레이스 주요 버전 업그레이드
- Amazon DocumentDB 마이너 버전 업그레이드
- 를 사용하여 Amazon DocumentDB 클러스터 업그레이드 AWS Database Migration Service
- LTS(장기 지원) 릴리스 사용
- Amazon DocumentDB 추가 지원

AWS Database Migration Service(DMS)를 사용하여 Amazon DocumentDB로 마이그레이션: 빠른 시작 안내서

주제

- DMS 소스 준비
- DMS 설정
- DocumentDB 압축 활성화
- 복제 작업 생성
- 진행 상황 모니터링
- 추가 정보

DMS 소스 준비

DocumentDB 변경 스트림을 활성화하거나 MongoDB Oplog가 DMS 변경 데이터 캡처(CDC)를 지원하도록 하려면 변경 스트림 활성화 섹션을 참조하세요.

- DMS 소스는 포함된 모든 컬렉션에 대해 DMS 전체 로드가 완료될 때까지 진행 중인 모든 변경 사항을 유지해야 합니다.
- DocumentDB 변경 스트림은 시간 기반입니다. 전체 로드를 완료하는 데 걸리는 시간을 수용할 수 있을 만큼 `change_stream_log_retention_duration` 설정이 충분히 큰지 확인합니다.
- MongoDB Oplog는 고정된 크기입니다. 전체 로드 중에 모든 작업을 유지할 수 있는 크기인지 확인합니다.

DMS 설정

DMS 인스턴스, 소스 및 대상 엔드포인트를 생성하고 각 엔드포인트를 테스트합니다.

DocumentDB 압축 활성화

Amazon DocumentDB 5.0: 사용자 지정 파라미터 그룹을 Amazon DocumentDB 클러스터에 연결하고 `default_collection_compression` 파라미터를 로 설정하여 압축을 활성화합니다`enabled`. 자세한 정보는 컬렉션 수준 문서 압축 관리을 참조하세요.

Amazon DocumentDB 8.0: Zstd 압축이 기본적으로 활성화됩니다. 자세한 정보는 Amazon DocumentDB 8.0에서 사전 기반 압축 관리을 참조하세요.

복제 작업 생성

1. DMS 콘솔의 탐색 창에서 마이그레이션 또는 복제를 선택한 후 작업을 선택합니다.
2. 작업 생성을 선택합니다.
3. 작업 생성 페이지의 작업 구성 섹션에서 다음을 수행합니다.
    - 고유하고 의미 있는 작업 식별자(예: "mongodb-docdb-replication")를 입력합니다.
    - 소스 데이터베이스 엔드포인트 드롭다운 메뉴에서 이전에 생성한 소스 엔드포인트를 선택합니다.
    - 대상 데이터베이스 엔드포인트 드롭다운 메뉴에서 이전에 생성한 대상 엔드포인트를 선택합니다.

- 작업 유형에서 마이그레이션 및 복제를 선택합니다.

4. 설정 섹션에서 다음을 수행합니다.

   - 작업 로그에서 CloudWatch 켜기 로그 상자를 선택합니다.
   - 편집 모드(섹션 상단)에서 JSON 편집기를 선택하고 다음 속성을 설정합니다.
     - `ParallelApplyThreads`를 5로 설정합니다(`TargetMetadata` 아래). 그러면 CDC에서 초당 ~1,000개의 insert/update/delete 작업이 활성화됩니다.
     - `MaxFullLoadSubTasks`를 16으로 설정합니다(`FullLoadSettings` 아래). 인스턴스 크기에 따라 이 값을 늘리는 것이 좋습니다.
     - 대용량 컬렉션(100GB 이상)의 경우 자동 파티션을 활성화합니다(테이블 매핑 아래 및 `parallel-load` 속성 아래).
       - "type": "partitions-auto"
       - "number-of-partitions": 16

진행 상황 모니터링

AWS DMS 콘솔을 사용하거나 사용자 지정 대시보드(대시보드 도구)를 생성하여 마이그레이션을 추적합니다. 다음 지표에 중점을 둡니다.

- FullLoadThroughputBandwidthTarget - 마이그레이션의 전체 로드 단계에서 데이터를 대상 데이터베이스로 전송할 때 DMS에서 사용하는 네트워크 대역폭(KB/초)을 측정합니다.
- CDCLatencyTarget - 소스 데이터베이스에서 변경이 발생할 때와 해당 변경이 대상 데이터베이스에 적용될 때 발생하는 시간 지연(초)을 측정합니다.
- CDCThroughputRowsTarget - 마이그레이션의 지속적 복제 단계에서 DMS가 대상 데이터베이스에 적용하는 초당 행 수를 측정합니다.

추가 정보

Amazon DocumentDB 및에 대한 자세한 내용은 다음을 AWS DMS참조하세요.

- Amazon DocumentDB 마이그레이션 런북
- MongoDB에서 Amazon DocumentDB로 마이그레이션

Amazon DocumentDB 마이그레이션 런북

이 실행서는 AWS Database Migration Service (DMS)를 사용하여 MongoDB 데이터베이스를 Amazon DocumentDB로 마이그레이션하기 위한 포괄적인 가이드를 제공합니다. 초기 검색부터 마이그레이션 후 검증에 이르기까지 종단 간 마이그레이션 여정 전반에 걸쳐 데이터베이스 관리자, 클라우드 엔지니어 및 개발자를 지원하도록 설계되었습니다.

이 런북에서는 MongoDB와 Amazon DocumentDB 간의 구현 및 지원되는 기능의 차이를 고려하여 구조화되고 체계적인 접근 방식을 강조합니다. 필수 마이그레이션 전 평가를 간략하게 설명하고, 호환성 고려 사항을 강조하고, 중단을 최소화하면서 성공적인 마이그레이션을 보장하는 데 필요한 주요 작업을 자세히 설명합니다.

이 런북은 다음 주제로 구성됩니다.

- 호환성 - Amazon DocumentDB에서 지원되는 MongoDB 기능 및 데이터 유형을 이해하고 잠재적인 비호환성을 식별합니다.
- 워크로드 검색 - 읽기/쓰기 패턴, 데이터 볼륨 및 성능 기준을 포함한 기존 MongoDB 워크로드를 분석합니다.
- 인덱스 마이그레이션 - Amazon DocumentDB에서 최적의 성능을 위해 MongoDB 인덱스를 추출하고 변환하기 위한 전략을 분석합니다.
- 사용자 마이그레이션 - 데이터베이스 사용자, 역할 및 액세스 제어를 Amazon DocumentDB로 마이그레이션하는 접근 방식을 자세히 설명합니다.
- 데이터 마이그레이션 - 전체 로드 및 변경 데이터 캡처(CDC) AWS DMS를 포함하여를 사용한 데이터 마이그레이션을 위한 다양한 방법을 다룹니다.
- 모니터링 - DMS 또는 기본 도구를 사용하여 마이그레이션할 때 다양한 모니터링 접근 방식을 자세히 설명합니다.
- 검증 - 마이그레이션 후 데이터 무결성 검사, 기능 검증 및 성능 비교 절차를 제공합니다.

이 런북의 지침에 따라 팀은 애플리케이션 기능을 보존하고 위험을 최소화하면서 Amazon DocumentDB로 원활하고 안전하며 효율적으로 전환할 수 있습니다.

호환성

주제

- 핵심 기능 호환성

- Amazon DocumentDB 호환성 평가 도구

MongoDB에서 Amazon DocumentDB로 마이그레이션할 때는 성공적인 마이그레이션을 위해 철저한 초기 평가 및 기능 호환성 검사가 필수적입니다. 이 프로세스는 집계 파이프라인 연산자, 쿼리 패턴, 인덱스 및 데이터 모델을 포함한 MongoDB 기능의 포괄적인 인벤토리로 시작됩니다.

Amazon DocumentDB는 MongoDB 3.6, 4.0, 5.0 및 8.0 API와 호환되므로 최신 MongoDB 전용 기능을 사용하는 애플리케이션에는 리팩터링이 필요할 수 있습니다. 평가할 중요 영역에는 샤딩 메커니즘(Amazon DocumentDB는 다른 접근 방식을 사용), 트랜잭션 구현, 변경 스트림 기능 및 인덱스 유형(특히 희소 인덱스와 부분 인덱스)이 포함됩니다.

성능 특성도 다릅니다. Amazon DocumentDB는 예측 가능한 성능을 갖춘 엔터프라이즈 워크로드에 최적화되어 있습니다. 테스트에는 최적화가 필요할 수 있는 쿼리 패턴을 식별하기 위해 두 시스템에 대해 대표적인 워크로드를 실행하는 작업이 포함되어야 합니다.

평가 단계에서는 실행 계획을 모니터링하여 잠재적 성능 격차를 감지하는 것이 중요합니다. 이를 통해 명확한 마이그레이션 로드맵을 생성하고, 필요한 애플리케이션 변경 사항을 식별하고, 원활한 전환을 위한 현실적인 일정을 수립할 수 있습니다.

핵심 기능 호환성

포괄적인 기능 지원

- CRUD 작업 - 대량 및 쿼리 연산자를 포함한 모든 기본 생성, 읽기, 업데이트, 삭제 작업을 완벽하게 지원하여 원활한 애플리케이션 호환성을 제공합니다.
- 풍부한 인덱싱 기능 - 단일 필드, 복합, TTL, 부분, 희소, 2Dsphere 인덱스에 대한 포괄적인 지원을 활용하여 텍스트 기반 조회를 위한 쿼리 성능 및 텍스트 인덱스(버전 5)를 최적화합니다.
- 엔터프라이즈급 복제 - 운영 오버헤드 없이 뛰어난 고가용성을 위해 읽기 전용 복제본이 포함된 강력한 자동 장애 조치 메커니즘의 이점을 누릴 수 있습니다.
- 고급 백업 솔루션 - 데이터 보호를 위한 시점 복구(PITR) 및 온디맨드 수동 스냅샷을 갖춘 자동 백업 시스템으로 편하게 작업할 수 있습니다.

향상된 AWS통합 기능

- 간소화된 집계 - 엔터프라이즈 워크로드에 최적화된 성능으로 가장 일반적으로 사용되는 집계 단계(`$match`, `$group`, `$sort`, `$project` 등)를 활용합니다.

- 트랜잭션 지원 - 대부분의 비즈니스 애플리케이션 요구 사항에 적합한 다중 문서 및 다중 컬렉션 트랜잭션을 구현합니다.
- 실시간 데이터 추적 - 간단한 명령으로 변경 스트림을 활성화하고 실시간 데이터 변경 모니터링을 위한 간단한 파라미터 그룹 설정을 통해 변경 스트림 보존 기간을 늘립니다.
- 위치 기반 서비스 - `$geoNear` 연산자 및 2Dsphere 인덱스를 지원하는 지리 공간 애플리케이션을 구현합니다.
- 텍스트 검색 기능 - 콘텐츠 검색 요구 사항에 맞는 기본 제공 텍스트 검색 기능을 활용합니다.

최신 아키텍처의 장점

- 클라우드 네이티브 설계 - MapReduce와 같은 레거시 기능을 보다 효율적인 집계 파이프라인 작업으로 대체하는 최적화된 AWS아키텍처를 제공합니다.
- 보안 강화 - AWS Identity and Access Management (IAM), SCRAM-SHA-1, SCRAM-SHA-256, X.509 인증서 인증 및 암호 기반 인증의 이점.
- 예측 가능한 성능 - 엔터프라이즈 워크로드에 특별히 최적화된 일관된 성능을 경험할 수 있습니다.

Amazon DocumentDB의 기능에 대한 포괄적인 개요는 Amazon DocumentDB에서 지원되는 MongoDB API, 작업 및 데이터 형식 및 기능적 차이: Amazon DocumentDB 및 MongoDB 섹션을 참조하여 데이터베이스의 잠재력을 극대화하세요.

Amazon DocumentDB는 MongoDB에서 제공하는 모든 인덱스를 지원하지는 않습니다. 호환성을 확인하기 위해 GitHub에서 무료 인덱스 도구를 제공합니다. 인덱스 도구를 실행하여 비호환성을 평가하고 그에 따라 해결 방법을 계획합니다.

Amazon DocumentDB 호환성 평가 도구

GitHub의 Amazon DocumentDB 호환성 도구는 MongoDB 워크로드를 분석하고 지원되지 않거나 Amazon DocumentDB에서 변경이 필요한 연산자, 명령 및 기능을 보고합니다. 3.6~8.0.1(기본값: 8.0.1)의 대상 버전을 지원합니다.

권장: URI 모드(MongoDB 5.0 이상)

URI 모드는 MongoDB 인스턴스에 직접 연결하고, 작업을 실시간으로 샘플링하며, 프로파일링이나 로그 액세스가 필요하지 않습니다.

대체 방법

- 로그 기반 분석 - MongoDB 프로파일러 로그를 구문 분석합니다. 실제 런타임 동작을 캡처하지만 프로파일링을 활성화해야 하며 로깅 기간만 다릅니다.
- 소스 코드 분석 - 애플리케이션 소스 코드에서 MongoDB API 사용을 스캔합니다. 포괄적인 범위를 제공하지만 사용하지 않는 코드 경로에 플래그를 지정할 수 있으며 동적으로 구성된 쿼리를 감지할 수 없습니다.

사전 조건

- Python 3.7 이상
- 도구를 복제합니다. `git clone https://github.com/awslabs/amazon-documentdb-tools.git`
- 종속성 설치: `pip install -r compat-tool/requirements.txt`

전체 사용 세부 정보 및 사용 가능한 모든 옵션은 GitHub의 README를 참조하세요.

워크로드 검색

MongoDB에서 Amazon DocumentDB로 마이그레이션하려면 기존 데이터베이스 워크로드를 철저히 이해해야 합니다. 워크로드 검색은 데이터베이스 사용 패턴, 데이터 구조, 쿼리 성능 및 운영 종속성을 분석하여 중단을 최소화하면서 원활한 전환을 보장하는 프로세스입니다. 이 섹션에서는 MongoDB에서 Amazon DocumentDB로 효과적으로 마이그레이션할 수 있도록 워크로드 검색과 관련된 주요 단계를 간략하게 설명합니다.

주제

- 기존 MongoDB 배포 평가
- 데이터 모델 차이점 식별
- 쿼리 및 성능 분석
- 보안 및 액세스 제어 검토
- 운영 및 모니터링 고려 사항

기존 MongoDB 배포 평가

마이그레이션하기 전에 다음을 포함하여 현재 MongoDB 환경을 평가하는 것이 중요합니다.

- 클러스터 아키텍처 - 노드, 복제본 세트 및 샤딩 구성의 수를 식별합니다. MongoDB에서 Amazon DocumentDB로 마이그레이션할 때는 Amazon DocumentDB가 사용자 제어 샤딩을 지원하지 않으

므로 MongoDB 샤딩 구성을 이해하는 것이 중요합니다. Amazon DocumentDB는 스토리지 기반 아키텍처와 다른 규모 조정 접근 방식을 사용하므로 샤딩된 MongoDB 환경을 위해 설계된 애플리케이션에는 아키텍처 변경이 필요합니다. Amazon DocumentDB로 이동할 때 데이터 배포 전략을 조정하고 샤딩된 컬렉션을 통합해야 할 수 있습니다.

- 스토리지 및 데이터 볼륨 - 클러스터의 총 데이터 크기와 인덱스 크기를 측정합니다. Oplog 검토 도구를 통해 이를 보완하여 쓰기 패턴과 데이터 증가 속도를 파악합니다. 클러스터 크기 조정에 대한 자세한 내용은 인스턴스 크기 조정 섹션을 참조하세요.
- 워크로드 패턴 - 읽기 및 쓰기 처리량, 쿼리 실행 빈도 및 인덱싱 효율성을 분석합니다.
- 운영 종속성 - MongoDB에 의존하는 모든 애플리케이션, 서비스 및 통합을 문서화합니다.

데이터 모델 차이점 식별

Amazon DocumentDB는 MongoDB와 호환되지만 지원되는 기능에는 다음과 같은 차이점이 있습니다.

- 트랜잭션 - Amazon DocumentDB는 일부 제한 사항에서 ACID 트랜잭션을 지원합니다.
- 스키마 설계 - 문서 구조, 임베디드 문서 및 참조가 Amazon DocumentDB의 모범 사례와 일치하는지 확인합니다.

쿼리 및 성능 분석

쿼리 동작을 이해하면 마이그레이션 및 마이그레이션 후 성능을 최적화할 수 있습니다. 분석할 주요 영역은 다음과 같습니다.

- 느린 쿼리 - MongoDB의 프로파일링 도구를 사용하여 실행 시간이 긴 쿼리를 식별합니다.
- 쿼리 패턴 - CRUD 작업 및 집계를 포함한 일반적인 쿼리 유형을 분류합니다.
- 인덱스 사용량 - Amazon DocumentDB에서 인덱스가 효과적으로 활용되는지 또는 최적화가 필요한지 평가합니다. Amazon DocumentDB에서 인덱스 사용량을 평가하고 성능을 최적화하려면 중요한 쿼리에서 `explain()` 메서드와 결합된 `$indexStats` 집계 파이프라인 단계를 사용합니다. `db.collection.aggregate([{$indexStats{}}])`를 실행하여 사용 중인 인덱스를 식별합니다. `explainPlan`을 통해 가장 빈번한 쿼리를 실행하여 보다 세부적인 분석을 수행할 수 있습니다.
- 동시성 및 워크로드 배포 - 읽기 및 쓰기 비율, 연결 풀링 및 성능 병목 현상을 평가합니다.

보안 및 액세스 제어 검토

인증 및 권한 부여

- MongoDB RBAC를 Amazon DocumentDB IAM 및 RBAC로 - MongoDB의 역할 기반 액세스 제어 사용자 및 역할을 AWS Identity and Access Management (IAM) 정책 및 Amazon DocumentDB SCRAM 인증 사용자에게 매핑합니다.
- 사용자 마이그레이션 전략 - 데이터베이스 사용자, 사용자 지정 역할 및 권한을 Amazon DocumentDB의 지원되는 인증 메커니즘으로 마이그레이션하는 계획을 세웁니다.
- 권한 차이 - 직접적으로 Amazon DocumentDB에 상응하는 권한(예: 클러스터 관리 역할)이 없는 MongoDB 권한을 식별합니다.
- 애플리케이션 인증 - Amazon DocumentDB의 암호 정책에 대한 연결 문자열 및 자격 증명 관리를 업데이트합니다. 보안 암호 관리자를 사용하여 자격 증명을 저장하고 암호를 교체할 수 있습니다.
- 서비스 계정 관리 -에서 서비스 계정 자격 증명을 관리하기 위한 프로세스를 설정합니다 AWS Secrets Manager.
- 최소 권한 구현 - 액세스 제어를 검토하고 구체화하여 새 환경에서 최소 권한 원칙을 구현합니다.

암호화(Encryption)

저장 및 전송 중 암호화가 규정 준수 요구 사항에 부합하는지 확인합니다.

네트워크 구성

가상 프라이빗 클라우드(VPC) 설정 및 보안 그룹 규칙을 계획합니다.

운영 및 모니터링 고려 사항

시스템 신뢰성을 유지하기 위해 워크로드 검색에는 다음이 포함되어야 합니다.

- 백업 및 복원 전략 - 기존 백업 방법과 Amazon DocumentDB의 백업 기능을 평가합니다.
- AWS Backup 통합 - Amazon DocumentDB AWS Backup 를 포함한 AWS 여러 서비스에서 중앙 집중식 백업 관리를 활용합니다.
- CloudWatch 지표 - MongoDB 모니터링 지표를 CPU, 메모리, 연결 및 스토리지에 대한 Amazon DocumentDB CloudWatch 지표에 매핑합니다.
- 성능 인사이트 - Amazon DocumentDB 성능 인사이트를 구현하여 데이터베이스 로드를 시각화하고 자세한 쿼리 분석을 통해 성능 문제를 분석합니다.
- 프로파일러 - 느린 실행 작업을 캡처하도록 Amazon DocumentDB 프로파일러를 구성합니다 (MongoDB의 프로파일러와 비슷하지만 Amazon DocumentDB별 설정 사용).
  - 적절한 임계값이 있는 파라미터 그룹을 통해 활성화합니다.
  - 프로파일러 데이터를 분석하여 최적화 기회 식별

- CloudWatch Events - Amazon DocumentDB 클러스터 이벤트에 대한 이벤트 기반 모니터링을 설정합니다.
  - 백업 이벤트, 유지 관리 기간 및 장애 조치에 대한 알림을 구성합니다.
  - 알림 및 AWS Lambda 자동 응답을 위해 Amazon SNS와 통합합니다.
- 감사 로깅 - 사용자 활동 및 보안 관련 이벤트를 추적하기 위한 감사 로깅 구성을 계획합니다.
- 향상된 모니터링 - 1초 간격으로 세분화된 OS 수준 지표에 대한 향상된 모니터링을 활성화합니다.

새 대상 클러스터 생성

의 단계에 따라 Amazon DocumentDB 클러스터를 생성합니다Amazon DocumentDB 클러스터 생성. 마이그레이션을 위해 클러스터를 구성할 때 다음 권장 사항을 적용합니다.

- 인스턴스 클래스: 최상의 성능을 위해 R8G를 선택합니다. 대상 리전에서 R8G를 사용할 수 없는 경우 R6G를 사용합니다. 최상의 전체 로드 처리량을 위해 가능한 한 큰 인스턴스를 선택합니다. 마이그레이션이 완료된 후 스케일 다운합니다.
- 엔진 버전: 최신(8.0.1)
- 인스턴스 수: 인스턴스 1개를 선택하여 마이그레이션 중에 비용을 최소화합니다. 전체 로드 마이그레이션이 완료된 후 고가용성을 위해 인스턴스 3개로 확장합니다.
- 네트워크 설정: Amazon DocumentDB 보안 그룹이 DMS 복제 인스턴스의 보안 그룹(온라인 마이그레이션용) 또는 EC2 마이그레이션 호스트(mongorestore를 사용한 오프라인 마이그레이션용)로부터의 인바운드 연결을 허용하는지 확인합니다.

인덱스 마이그레이션

MongoDB에서 Amazon DocumentDB로 마이그레이션하려면 데이터뿐만 아니라 인덱스를 전송하여 쿼리 성능을 유지하고 데이터베이스 작업을 최적화해야 합니다. 이 섹션에서는 호환성과 효율성을 보장하면서 MongoDB에서 Amazon DocumentDB로 인덱스를 마이그레이션하는 단계별 프로세스에 대해 간략하게 설명합니다.

Amazon DocumentDB 인덱스 도구 사용

GitHub에서 인덱스 도구 복제

```
git clone https://github.com/awslabs/amazon-documentdb-tools.git
cd amazon-documentdb-tools/index-tool
```

```
pip install -r requirements.txt
```

MongoDB에서 인덱스 내보내기(MongoDB에서 마이그레이션하는 경우)

```
python3 migrationtools/documentdb_index_tool.py \
  --dump-indexes \
  --dir index_export \
  --uri 'mongodb://<username>:<password>@<source_endpoint>:27017'
```

인덱스 확인

```
python3 migrationtools/documentdb_index_tool.py \
  --show-issues \
  --dir index_export
```

인덱스 가져오기

```
python3 migrationtools/documentdb_index_tool.py \
  --restore-indexes \
  --skip-incompatible \
  --dir index_export \
  --uri 'mongodb://<username>:<password>@<target_endpoint>:27017/?
tls=true&tlsCAFile=global-bundle.pem&replicaSet=rs0&retryWrites=false'
```

사용자 마이그레이션

MongoDB에서 Amazon DocumentDB로 사용자를 마이그레이션하는 작업은 액세스 제어, 인증 및 데이터베이스 보안을 유지하는 데 필수적입니다. 이 섹션에서는 Amazon DocumentDB 사용자 내보내기 도구를 사용하여 역할과 권한을 유지하면서 MongoDB 사용자를 성공적으로 마이그레이션하는 자세한 단계에 대해 간략하게 설명합니다.

Amazon DocumentDB 사용자 내보내기 도구 사용

Export Users tool는 사용자와 역할을 MongoDB 또는 Amazon DocumentDB에서 JavaScript 파일로 내보낸 후 다른 클러스터에서 다시 생성하는 데 사용할 수 있습니다.

사전 조건

```
# Clone the repository
git clone https://github.com/awslabs/amazon-documentdb-tools.git
```

```
cd amazon-documentdb-tools/migration/export-users
```

```
# Install required dependencies
pip install pymongo
```

1단계: 사용자 및 역할 내보내기

```
# Export users and roles to JavaScript files
python3 docdbExportUsers.py \
  --users-file mongodb-users.js \
  --roles-file mongodb-roles.js \
  --uri "mongodb://<username>:<password>@<source_endpoint>:27017"
```

2단계: 사용자 파일 편집

보안상의 이유로 암호를 내보내지 않습니다. `mongodb-users.js` 파일을 열고 각 `createUser` 문의 `pwd` 값을 대체하여 각 사용자의 암호를 추가합니다.

```
db.getSiblingDB("admin").createUser({
  user: "appuser",
  pwd: "REPLACE_THIS_PASS",
  roles: [
    { role: "readWrite", db: "mydb" }
  ]
})
```

3단계: Amazon DocumentDB에 사용자 지정 역할 복원

```
# Import roles first
mongosh \
  --tls \
  --host <target_endpoint>:27017 \
  --tlsCAFile global-bundle.pem \
  --username <username> \
  --password <password> \
  mongodb-roles.js
```

4단계: Amazon DocumentDB에 사용자 복원

```
# Import users after roles are created
mongosh \
```

```
  --tls \
  --host <target_endpoint>:27017 \
  --tlsCAFile global-bundle.pem \
  --username <username> \
  --password <password> \
  mongodb-users.js
```

중요 정보

- 암호는 보안상의 이유로 내보내지 않으며 users.js 파일에 수동으로 추가해야 합니다.
- 적절한 역할 할당을 보장하려면 사용자보다 역할을 먼저 가져와야 합니다.
- 이 도구는 mongosh 셸로 직접 실행할 수 있는 JavaScript 파일을 생성합니다.
- 사용자 지정 역할과 해당 권한은 마이그레이션 중에 유지됩니다.
- 이 접근 방식을 사용하면 가져오기 전에 사용자 권한을 검토하고 수정할 수 있습니다.

이 방법은 마이그레이션 프로세스 중에 암호 재설정을 허용하면서 MongoDB에서 Amazon DocumentDB로 사용자 및 역할을 마이그레이션하는 안전하고 유연한 접근 방식을 제공합니다.

데이터 마이그레이션

주제

- 온라인 마이그레이션
- 오프라인 마이그레이션

온라인 마이그레이션

이 섹션에서는 가동 중지 시간을 최소화하고 연속 복제를 위해를 사용하여 MongoDB에서 Amazon DocumentDB로 온라인 마이그레이션 AWS DMS 을 수행하는 자세한 단계를 제공합니다. 시작하려면 Amazon DocumentDB 클러스터를 대상으로 설정하고 MongoDB 인스턴스가 소스로 올바르게 구성되어 있는지 확인합니다. 일반적으로 변경 데이터 캡처를 위해 복제본 세트 모드가 필요합니다. 그런 다음 DMS 복제 인스턴스를 생성하고 필요한 연결 세부 정보로 소스 및 대상 엔드포인트를 정의합니다. 엔드포인트를 검증한 후 전체 데이터 로드, 지속적 복제 또는 둘 다를 포함할 수 있는 마이그레이션 작업을 구성하고 시작합니다.

새 대상 클러스터 생성

새 대상 클러스터 생성을(를) 참조하세요.

소스를 구성합니다.

MongoDB와 Amazon DocumentDB는 모두 시나리오에 따라 마이그레이션 소스 역할을 할 수 있습니다.

- MongoDB 소스 - 온프레미스 또는 자체 관리형 MongoDB에서 Amazon DocumentDB 또는 기타 AWS 데이터베이스 서비스로 마이그레이션할 때 일반적입니다. 마이그레이션 중에 변경 데이터 캡처를 지원하려면 적절한 크기의 oplog를 사용하여 복제본 세트 모드에서 실행해야 합니다(전체 로드 중에 모든 작업을 보관할 수 있는 크기여야 함).
- Amazon DocumentDB 소스 - 일반적으로 리전 간 복제, 버전 업그레이드 또는 MongoDB Atlas와 같은 다른 데이터베이스 서비스로 마이그레이션하는 데 사용됩니다. 클러스터 파라미터 그룹에서 `change_stream_log_retention_duration` 파라미터를 설정하여 마이그레이션 중에 진행 중인 변경 사항을 캡처하려면 변경 스트림 활성화가 필요합니다. 전체 로드를 완료하는 데 필요한 시간을 수용할 수 있을 만큼 `change_stream_log_retention_duration` 설정이 충분히 큰지 확인합니다.

마이그레이션을 시작하기 전에 AWS DMS 액세스를 허용하도록 소스를 구성합니다.

적절한 권한이 있는 MongoDB 사용자를 생성합니다.

```
db.createUser({
  user: "dmsUser",
  pwd: "<password>",
  roles: [{ role: "readAnyDatabase", db: "admin" }]
})
```

네트워크 및 인증을 구성합니다.

MongoDB에서 DMS로의 마이그레이션을 위한 네트워크 연결을 구성하는 경우 다음을 수행합니다.

EC2 호스팅 MongoDB 소스

- DMS 복제 인스턴스 보안 그룹의 인바운드 트래픽을 허용하도록 EC2 보안 그룹을 수정합니다.
- TCP 포트 27017(또는 사용자 지정 MongoDB 포트)에 대한 규칙을 추가합니다.
- DMS 복제 인스턴스의 보안 그룹 ID를 정확한 액세스 제어를 위한 소스로 사용합니다.
- EC2 인스턴스의 서브넷에 DMS 복제 인스턴스의 서브넷에 대한 경로가 있는지 확인합니다.

온프레미스 MongoDB 소스

- DMS 복제 인스턴스의 퍼블릭 IP 주소로부터의 인바운드 연결을 허용하도록 방화벽을 구성합니다.
- Direct Connect 또는 VPN을 사용하는 경우 네트워크와 DMS 인스턴스가 포함된 VPC 간의 라우팅이 적절한지 확인합니다.
- telnet 또는 nc 명령을 사용하여 DMS 서브넷에서 MongoDB 서버로의 연결을 테스트합니다.

MongoDB Atlas 소스

- DMS 복제 인스턴스 IP 주소를 MongoDB Atlas IP 허용 목록에 추가합니다.
- Atlas가 실행 중인 경우 AWS VPC와 MongoDB Atlas VPC 간에 VPC 피어링을 구성합니다 AWS.
- 다른 클라우드 공급자에서 실행되는 경우 프라이빗 연결(엔터프라이즈 계층) AWS PrivateLink 을 설정합니다.
- 적절한 읽기/쓰기 권한이 있는 전용 사용자를 생성합니다.
- SSL 모드가 'verify-full'로 설정된 MongoDB Atlas 연결 문자열을 사용합니다.
- 마이그레이션 기간에 충분한 oplog 크기를 확보합니다.

Amazon DocumentDB 소스

DMS 복제 인스턴스 보안 그룹의 인바운드 트래픽을 허용하도록 소스 Amazon DocumentDB 보안 그룹을 구성합니다.

DMS 복제 인스턴스 생성

GitHub의 DMS Buddy를 사용하여 최적의 DMS 설정 및 인스턴스 크기로 최적의 마이그레이션 인프라를 생성합니다. 수동으로 구성하려면 다음 단계를 따르세요.

1. AWS DMS 콘솔의 탐색 창에서 마이그레이션 또는 복제를 선택한 다음 프로비저닝된 인스턴스를 선택합니다.
2. 복제 인스턴스 세부 정보를 입력합니다.

   - 인스턴스 이름: 고유한 이름을 선택합니다.
   - 인스턴스 클래스: 워크로드를 기반으로 선택합니다. 예: dms.r7i.large(작은 워크로드), dms.r7i.4xlarge(큰 워크로드).
   - 엔진 버전: 3.5.4
   - 할당된 스토리지: 기본값은 50GB입니다(필요한 경우 증가). 문서 크기, 업데이트/초 및 전체 로드 기간에 따라 달라집니다.

- Multi-AZ 배포: 필요한 경우 고가용성을 활성화합니다.
- Amazon DocumentDB와 동일한 VPC를 선택합니다.
- 보안 그룹이 소스 및 Amazon DocumentDB의 인바운드 트래픽을 허용하는지 확인합니다.

3. 복제 인스턴스 생성을 선택하고 상태를 사용할 수 있을 때까지 기다립니다.

DMS 엔드포인트 생성

소스 엔드포인트 생성

MongoDB 소스의 경우

1. DMS 콘솔의 탐색 창에서 마이그레이션 또는 복제를 선택한 다음 엔드포인트를 선택합니다.
2. 엔드포인트 생성을 선택합니다.
3. 엔드포인트 생성 페이지에서 소스 엔드포인트를 선택합니다.
4. 엔드포인트 구성 섹션에서 다음을 수행합니다.

   - 고유하고 의미 있는 엔드포인트 식별자(예: "mongodb-source")를 입력합니다.
   - MongoDB를 소스 엔진으로 선택합니다.
   - 엔드포인트 데이터베이스 액세스에서 수동으로 액세스 정보 제공을 선택합니다.
   - 서버 이름에 *MongoDB ## DNS ##/IP ##*를 입력합니다.
   - 포트에 27017(기본 MongoDB 포트)을 입력합니다.
   - 인증 모드에서 애플리케이션에 적합한 모드(암호/SSL)를 선택합니다(기본값은 보안 암호 관리자임).
   - 인증 모드가 암호인 경우 다음을 제공합니다.
     - 사용자 이름 및 암호: MongoDB 자격 증명을 입력합니다.
     - 데이터베이스 이름: 소스 데이터베이스의 이름입니다.
     - 인증 메커니즘: SCRAM-SHA-1(기본값) 또는 적절한 메커니즘
5. 메타데이터 모드의 경우 문서의 기본 설정을 그대로 유지합니다.
6. 추가 연결 속성:

   - authSource=admin(인증 데이터베이스가 다른 경우)
   - replicaSet=<your-replica-set-name>(CDC의 경우 필수)

Amazon DocumentDB 소스의 경우

1. DMS 콘솔의 탐색 창에서 마이그레이션 또는 복제를 선택한 다음 엔드포인트를 선택합니다.
2. 엔드포인트 생성을 선택합니다.
3. 엔드포인트 생성 페이지에서 소스 엔드포인트를 선택합니다.
4. 엔드포인트 구성 섹션에서 다음을 수행합니다.

   - 고유하고 의미 있는 엔드포인트 식별자(예: "docdb-source")를 입력합니다.
   - 소스 엔진으로 Amazon DocumentDB를 선택합니다.
   - 엔드포인트 데이터베이스 액세스에서 수동으로 액세스 정보 제공을 선택합니다.
   - 서버 이름에 *## Amazon DocumentDB #### #####*를 입력합니다.
   - 포트에 27017(기본 Amazon DocumentDB 포트)을 입력합니다.
   - SSL 모드에서 verify-full(Amazon DocumentDB에 권장)을 선택합니다.
   - CA 인증서에서 Amazon RDS 루트 CA 인증서를 선택합니다.
   - 인증 모드에서 애플리케이션에 적합한 모드(암호/SSL)를 선택합니다(기본값은 보안 암호 관리자임).
   - 인증 모드가 암호인 경우 다음을 제공합니다.
     - 사용자 이름 및 암호: Amazon DocumentDB 자격 증명을 입력합니다.
     - 데이터베이스 이름: 소스 데이터베이스의 이름입니다.
     - 인증 메커니즘: SCRAM-SHA-1(기본값) 또는 적절한 메커니즘
5. 메타데이터 모드의 경우 문서의 기본 설정을 그대로 유지합니다.

대상 엔드포인트 생성(Amazon DocumentDB)

1. DMS 콘솔의 탐색 창에서 마이그레이션 또는 복제를 선택한 다음 엔드포인트를 선택합니다.
2. 엔드포인트 생성을 선택합니다.
3. 엔드포인트 생성 페이지에서 대상 엔드포인트를 선택합니다.
4. 엔드포인트 구성 섹션에서 다음을 수행합니다.

   - 고유하고 의미 있는 엔드포인트 식별자(예: "docdb-target")를 입력합니다.
   - 대상 엔진으로 Amazon DocumentDB를 선택합니다.
   - 엔드포인트 데이터베이스에 대한 액세스에서 데이터베이스에 대한 액세스를 인증하는 데 사용할 방법을 선택합니다.
     - AWS Secrets Manager를 선택하는 경우 보안 암호 필드에서 Amazon DocumentDB 자격 증명을 저장하는 보안 암호를 선택합니다.

- 수동으로 액세스 정보 제공을 선택합니다.
  - 서버 이름에 *## Amazon DocumentDB #### #####*를 입력합니다.
  - 포트에 27017(기본 Amazon DocumentDB 포트)을 입력합니다.
  - SSL 모드에서 verify-full(Amazon DocumentDB에 권장)을 선택합니다.
  - CA 인증서의 경우 SSL 확인을 위해 CA 인증서 번들을 다운로드하고 지정합니다.
  - 인증 모드에서 애플리케이션에 적합한 모드(암호/SSL)를 선택합니다(기본값은 보안 암호 관리자임).
  - 인증 모드가 암호인 경우 다음을 제공합니다.
    - 사용자 이름 및 암호: Amazon DocumentDB 자격 증명을 입력합니다.
    - 데이터베이스 이름: 소스 데이터베이스의 이름입니다.
    - 인증 메커니즘: SCRAM-SHA-1(기본값) 또는 적절한 메커니즘

5. 메타데이터 모드의 경우 문서의 기본 설정을 그대로 유지합니다.

복제 작업 생성

1. DMS 콘솔의 탐색 창에서 마이그레이션 또는 복제를 선택한 후 작업을 선택합니다.
2. 작업 생성을 선택합니다.
3. 작업 생성 페이지의 작업 구성 섹션에서 다음을 수행합니다.
   - 고유하고 의미 있는 작업 식별자(예: "mongodb-docdb-replication")를 입력합니다.
   - 소스 데이터베이스 엔드포인트 드롭다운 메뉴에서 이전에 생성한 소스 엔드포인트를 선택합니다.
   - 대상 데이터베이스 엔드포인트 드롭다운 메뉴에서 이전에 생성한 대상 엔드포인트를 선택합니다.
   - 프로비저닝된 인스턴스 드롭다운 메뉴에서 이전에 생성한 복제 인스턴스를 선택합니다.
   - 작업 유형에서 마이그레이션 및 복제를 선택합니다.
   - 소스에서 대상으로 계속 복제하려는 기간은 얼마나 됩니까?에서 제한된 시간 동안을 선택합니다.
4. 설정 섹션에서 다음을 수행합니다.
   - 대상 테이블 준비 모드에서 아무 작업도 수행하지 않음을 선택합니다. 그렇지 않으면 이미 생성된 컬렉션과 인덱스가 삭제됩니다.
   - 작업 로그에서 CloudWatch 커거 로그 상자를 선택합니다.

- 다른 모든 설정의 기본값은 그대로 둡니다.

5. 설정 섹션 상단으로 돌아가 편집 모드에서 JSON 편집기를 선택하고 다음 속성을 설정합니다.

```
{
  "TargetMetadata": {
    "ParallelApplyThreads": 5
  },
  "FullLoadSettings": {
    "MaxFullLoadSubTasks": 16
  }
}
```

   컬렉션 수와 인스턴스 용량에 MaxFullLoadSubTasks 따라를 설정합니다. 더 적거나 작은 컬렉션에는 더 낮은 값(2~4)을 사용하고 많은 대규모 컬렉션에는 더 높은 값(8~16)을 사용합니다. 자동 크기 조정은 GitHub의 DMS Buddy를 참조하세요.

6. 테이블 매핑 섹션에서 새 선택 규칙을 추가합니다.

   - 스키마 이름에 마이그레이션할 소스 데이터베이스를 추가합니다. %를 사용하여 여러 데이터베이스를 지정합니다.
   - 스키마 테이블 이름에 마이그레이션할 소스 컬렉션을 추가합니다. %를 사용하여 여러 컬렉션을 지정합니다.
   - 작업에서 기본 설정인 포함을 그대로 유지합니다.

7. 대용량 컬렉션(100GB 이상)의 경우 테이블 설정 규칙을 추가합니다.

   - 스키마 이름에 마이그레이션할 소스 데이터베이스를 추가합니다. %를 사용하여 여러 데이터베이스를 지정합니다.
   - 스키마 테이블 이름에 마이그레이션할 소스 컬렉션을 추가합니다. %를 사용하여 여러 컬렉션을 지정합니다.
   - 파티션 수의 경우 워크로드에 따라를 설정합니다. 소수의 대규모 컬렉션에서 파티션을 ≥로 설정MaxFullLoadSubTasks하여 모든 작업자를 바쁘게 유지합니다. 병렬로 로드되는 많은 컬렉션의 경우 컬렉션 수 자체가 충분한 병렬 처리를 제공하므로 파티션을 ≤MaxFullLoadSubTasks로 설정합니다.

8. 마이그레이션 전 평가 섹션에서 해당 옵션이 꺼져 있는지 확인합니다. 사전 마이그레이션 평가는 Amazon DocumentDB에서 대상으로 지원되지 않으므로 비활성화해야 합니다.

오프라인 마이그레이션

이 섹션에서는 기본 MongoDB 도구인 `mongodump` 및 `mongorestore`를 사용하여 자체 관리형 MongoDB 인스턴스에서 Amazon DocumentDB로 오프라인 마이그레이션을 수행하는 프로세스에 대해 간략하게 설명합니다.

사전 조건

소스 MongoDB 요구 사항

- 적절한 권한이 있는 소스 MongoDB 인스턴스에 대한 액세스 권한입니다.
- 필요한 경우 `mongodump`를 설치합니다(MongoDB 설치 중에 설치됨).
- 덤프 파일을 위한 디스크 공간이 충분한지 확인합니다.

대상 Amazon DocumentDB 요구 사항

- Amazon DocumentDB 클러스터가 프로비저닝되어 있는지 확인합니다.
- 마이그레이션을 용이하게 하려면 Amazon DocumentDB와 동일한 VPC에 EC2 인스턴스가 있는지 확인합니다.
- 소스 환경과 Amazon DocumentDB 간에 네트워크 연결을 사용할 수 있어야 합니다.
- `mongorestore`는 마이그레이션 EC2 인스턴스에 설치해야 합니다.
- Amazon DocumentDB에 액세스하려면 적절한 IAM 권한을 구성해야 합니다.

일반 요구 사항

- AWS CLI 를 구성해야 합니다(중간 스토리지에 AWS 서비스를 사용하는 경우).
- 데이터 전송에 충분한 대역폭을 사용할 수 있어야 합니다.
- 가동 중지 기간이 승인되어야 합니다(실시간 마이그레이션을 수행하는 경우 다른 접근 방식 고려).

Amazon DocumentDB 클러스터 생성

새 대상 클러스터 생성을(를) 참조하세요.

데이터 덤프 수행(mongodump)

다음 옵션 중 하나를 선택하여 덤프 파일을 생성합니다.

- 옵션 1: 기본

```
mongodump \
  --uri="mongodb://<user>:<password>@<source_endpoint>:27017/<database>" \
  --out=/path/to/dump
```

- 옵션 2: 제어 및 성능 향상

  gzip을 사용하여 출력을 압축하고, 컬렉션을 병렬로 덤프하고, TLS를 활성화하고, 사용 가능한 경우 보조 복제본에서 읽습니다.

```
mongodump \
  --uri="mongodb://<user>:<password>@<source_endpoint>:27017" \
  --out=/path/to/dump \
  --gzip \
  --numParallelCollections=4 \
  --tls \
  --authenticationDatabase=admin \
  --readPreference=secondaryPreferred
```

- 옵션 3: 대용량 데이터베이스

  특정 데이터베이스 및 컬렉션을 덤프하고, 쿼리별로 문서를 필터링하고, 단일 압축 아카이브로 출력합니다.

```
mongodump \
  --host=<source_endpoint> \
  --port=27017 \
  --username=<user> \
  --password=<password> \
  --db=<specific_db> \
  --collection=<specific_collection> \
  --query='{ "date": { "$gt": "2020-01-01" } }' \
  --archive=/path/to/archive.gz \
  --gzip \
  --tls
```

복원 환경으로 덤프 파일 전송

덤프 크기에 따라 적절한 방법을 선택합니다.

- 소규모 - 마이그레이션 머신(이전에 생성한 EC2 인스턴스)에 직접 복사합니다.

```
scp -r /path/to/dump user@migration-machine:/path/to/restore
```

- 중간 규모 - Amazon S3를 중간 스토리지로 사용합니다.

```
aws s3 cp --recursive /path/to/dump s3://your-bucket/mongodb-dump/
```

- 대규모 - 매우 큰 데이터베이스의 경우 AWS DataSync 또는 물리적 전송을 고려합니다.

Amazon DocumentDB로 데이터 복원(mongorestore)

복원 프로세스를 시작하기 전에 Amazon DocumentDB에서 인덱스를 생성합니다. Amazon DocumentDB 인덱스 도구를 사용하여 인덱스를 내보내고 가져올 수 있습니다.

다음 옵션 중 하나를 선택하여 데이터를 복원합니다.

- 옵션 1: 기본 복원

```
mongorestore \
  --uri="mongodb://<user>:<password>@<target_endpoint>:27017" \
  --tls \
  --tlsCAFile=global-bundle.pem \
  /path/to/dump
```

- 옵션 2: 제어 및 성능 향상

  병렬 수집 및 문서 삽입으로 gzip 압축 덤프를 복원합니다. 인덱스가 미리 생성되므로 인덱스 복원을 건너뜁니다.

```
mongorestore \
  --uri="mongodb://<user>:<password>@<target_endpoint>:27017" \
  --tls \
  --tlsCAFile=global-bundle.pem \
  --gzip \
  --numParallelCollections=4 \
  --numInsertionWorkersPerCollection=4 \
  --noIndexRestore \
  /path/to/dump
```

- 옵션 3: 대용량 데이터베이스 또는 특정 제어

네임스페이스 필터링이 있는 압축된 아카이브에서 복원하여 특정 데이터베이스 및 컬렉션을 포함하거나 제외합니다. 인덱스가 미리 생성되므로 인덱스 복원을 건너뜁니다.

```
mongorestore \
  --host=<target_endpoint> \
  --port=27017 \
  --username=<user> \
  --password=<password> \
  --tls \
  --tlsCAFile=global-bundle.pem \
  --archive=/path/to/archive.gz \
  --gzip \
  --nsInclude="db1.*" \
  --nsExclude="db1.sensitive_data" \
  --noIndexRestore
```

모니터링

이 섹션에서는 진행 중인 마이그레이션의 진행 상황, 성능 및 상태를 추적하는 자세한 모니터링 프로세스를 제공합니다.

모니터링 단계는 마이그레이션 방법(AWS DMS, mongodump/mongorestore 또는 기타 도구)에 관계없이 적용됩니다.

AWS DMS 마이그레이션 모니터링(해당하는 경우)

다음 주요 CloudWatch 지표를 모니터링합니다.

전체 로드 단계 지표

- FullLoadThroughputBandwidthTarget - 전체 로드 중 네트워크 대역폭(KB/초)
- FullLoadThroughputRowsTarget - 초당 로드된 행/문서 수
- FullLoadThroughputTablesTarget - 분당 완료된 테이블/컬렉션 수
- FullLoadProgressPercent - 완료된 전체 로드의 백분율
- TablesLoaded - 성공적으로 로드된 테이블/컬렉션 수
- TablesLoading - 현재 로드 중인 테이블/컬렉션 수
- TablesQueued - 로드 대기 중인 테이블/컬렉션 수
- TablesErrored - 로드에 실패한 테이블/컬렉션 수

CDC 단계 지표

- CDCLatencyTarget - 소스 변경과 대상 애플리케이션 간의 시간 지연(초)
- CDCLatencySource - 소스 변경과 DMS 읽기 간의 시간 지연(초)
- CDCThroughputRowsTarget - 지속적 복제 중에 적용되는 초당 행
- CDCThroughputBandwidthTarget - CDC 중 네트워크 대역폭(KB/초)
- CDCIncomingChanges - 소스에서 수신된 변경 이벤트 수
- CDCChangesMemoryTarget - 대상 측에 변경 사항을 저장하는 데 사용되는 메모리(MB)

리소스 지표

- CPUUtilization - 복제 인스턴스의 CPU 사용량
- FreeableMemory - 복제 인스턴스에서 사용 가능한 메모리
- FreeStorageSpace - 복제 인스턴스에서 사용 가능한 스토리지
- NetworkTransmitThroughput - 복제 인스턴스의 네트워크 처리량
- NetworkReceiveThroughput - 복제 인스턴스의 네트워크 처리량

오류 지표

- ErrorsCount - 마이그레이션 중 총 오류 수
- TableErrorsCount - 테이블별 오류 수
- RecordsErrorsCount - 레코드별 오류 수

`CDCLatencyTarget` 및 `CPUUtilization` 등의 중요한 지표에 대한 CloudWatch 경보를 생성하여 마이그레이션 성능이 저하될 경우 알림을 수신합니다.

DMS 로그(CloudWatch 로그)

1. Amazon CloudWatch Logs 콘솔로 이동합니다.
2. 로그 그룹에서 찾아 선택합니다. "dms-tasks –"와 비슷합니다.
3. 오류 정보가 포함될 수 있는 로그 스트림을 찾습니다.
    - 이름에 "error"가 있는 스트림
    - 작업 ID 또는 엔드포인트 이름이 있는 스트림

- 마이그레이션 중 가장 최근의 로그 스트림

4. 이러한 스트림 내에서 다음과 같은 키워드를 검색합니다.

- "error"
- "exception"
- "failed"
- "warning"

DMS 작업 상태(사용 AWS CLI)

```
aws dms describe-replication-tasks \
  --filters Name=replication-task-id,Values=<task_id> \
  --query "ReplicationTasks[0].Status"
```

예상 상태 흐름:

생성 → 준비 → 실행 중 → 중지 중 → 중지됨(또는 실패)

를 사용하여 모니터링 **docdb-dashboarder**

docdb-dashboarder 도구는 필수 성능 지표가 포함된 CloudWatch 대시보드를 자동으로 생성하여 Amazon DocumentDB 클러스터에 대한 포괄적인 모니터링을 제공합니다. 이러한 대시보드에는 중요한 클러스터 수준 지표(복제 지연, 작업 카운터), 인스턴스 수준 지표(CPU, 메모리, 연결) 및 스토리지 지표(볼륨 사용량, 백업 스토리지)가 표시됩니다. 마이그레이션 시나리오의 경우 이 도구는 CDC 복제 지연 및 작업 속도와 같은 지표를 사용하여 마이그레이션 진행 상황을 추적하는 특수 대시보드를 제공합니다. 대시보드는 여러 클러스터를 동시에 모니터링하고 NVMe 지원 인스턴스에 대한 지원을 포함할 수 있습니다. 팀은 이러한 지표를 시각화하여 성능 병목 현상을 사전에 식별하고, 리소스 할당을 최적화하고, Amazon DocumentDB 배포의 원활한 운영을 보장할 수 있습니다. 이 도구를 사용하면 모든 환경에서 일관된 모니터링을 제공하면서 수동 대시보드를 생성할 필요가 없습니다. 설정 지침 및 고급 구성 옵션은 Amazon DocumentDB Dashboarder Tool GitHub 리포지토리를 참조하세요.

검증

주제

- 검증 체크리스트
- 스키마 및 인덱스 검증

- 데이터 샘플링 및 필드 수준 검증
- DataDiffer 도구를 사용한 검증

이 섹션에서는 마이그레이션 후 데이터 일관성, 무결성 및 애플리케이션 호환성을 보장하기 위한 자세한 검증 프로세스를 제공합니다.

검증 단계는 마이그레이션 방법(AWS DMS, mongodump/mongorestore 또는 기타 도구)에 관계없이 적용됩니다.

검증 체크리스트

각 컬렉션의 문서 수가 소스 및 대상 간에 일치하는지 확인합니다.

MongoDB 소스

```
mongosh \
  --host <source_endpoint> \
  --port 27017 \
  --username <user> \
  --password <password> \
  --eval "db.<collection>.countDocuments()"
```

Amazon DocumentDB 대상

```
mongosh \
  --host <target_endpoint> \
  --port 27017 \
  --username <user> \
  --password <password> \
  --tls \
  --tlsCAFile global-bundle.pem \
  --eval "db.<collection>.countDocuments()"
```

스키마 및 인덱스 검증

다음을 확인하세요.

- 모든 컬렉션이 대상에 존재합니다.
- 인덱스가 올바르게 복제됩니다.
- 스키마 정의가 동일합니다(적용된 경우).

컬렉션 확인(소스 대 대상)

```
mongosh --host <source_endpoint> \
  --port 27017 \
  --username <username> \
  --password <password> \
  --eval "db = db.getSiblingDB('<database>'); db.getCollectionNames();"

mongosh --host <target_endpoint> \
  --port 27017 \
  --username <username> \
  --password <password> \
  --tls --tlsCAFile global-bundle.pem \
  --eval "db = db.getSiblingDB('<database>'); db.getCollectionNames();"
```

인덱스 확인(소스 대 대상)

```
mongosh --host <source_endpoint> \
  --port 27017 \
  --username <username> \
  --password <password> \
  --eval "db.<collection>.getIndexes()"

mongosh --host <target_endpoint> \
  --port 27017 \
  --username <username> \
  --password <password> \
  --tls --tlsCAFile global-bundle.pem \
  --eval "db.<collection>.getIndexes()"
```

컬렉션 목록을 비교하여 누락되거나 추가되는 컬렉션이 없는지 확인합니다.

인덱스 이름, 키 정의, 고유 제약 조건 및 TTL 인덱스를 확인하여 인덱스를 검증합니다(있는 경우).

스키마 검증 규칙 확인(MongoDB에서 스키마 검증을 사용하는 경우)

```
mongosh --host <source_endpoint> \
  --port 27017 \
  --username <username> \
  --password <password> \
  --eval "db.getCollectionInfos({name: '<collection>'})[0].options.validator"

mongosh --host <target_endpoint> \
```

```
  --port 27017 \
  --username <username> \
  --password <password> \
  --tls --tlsCAFile global-bundle.pem \
  --eval "db.getCollectionInfos({name: '<collection>'})[0].options.validator"
```

데이터 샘플링 및 필드 수준 검증

무작위로 문서를 샘플링하고 소스와 대상 간의 필드를 비교할 수 있습니다.

수동 샘플링

5개의 무작위 문서 가져오기(소스):

```
mongosh --host <source_endpoint> \
  --port 27017 \
  --username <username> \
  --password <password> \
  --eval "db.<collection>.aggregate([{ \$sample: { size: 5 } }])"
```

동일한 문서 ID 가져오기(대상):

```
mongosh --host <target_endpoint> \
  --port 27017 \
  --username <username> \
  --password <password> \
  --tls --tlsCAFile global-bundle.pem \
  --eval "db.<collection>.find({ _id: { \$in: [<list_of_ids>] } })"
```

자동 샘플링

```
import pymongo

# Connect to source and target
source_client = pymongo.MongoClient("<source_uri>")
target_client = pymongo.MongoClient("<target_uri>",
                                    tls=True,
                                    tlsCAFile='global-bundle.pem')
source_db = source_client["<db_name>"]
target_db = target_client["<db_name>"]

# Compare 100 random documents
```

```
for doc in source_db["<collection>"].aggregate([{"$sample": {"size": 100}}]):
    target_doc = target_db["<collection>"].find_one({"_id": doc["_id"]})
    if target_doc != doc:
        print(f"Mismatch in _id: {doc['_id']}")
    else:
        print(f"Match: {doc['_id']}")
```

DataDiffer 도구를 사용한 검증

GitHub의 DataDiffer 도구는 소스 데이터베이스와 대상 데이터베이스 간의 데이터를 비교할 수 있는 신뢰할 수 있는 방법을 제공합니다.

사전 조건

DataDiffer 도구를 설치하기 전에 다음 사전 조건을 충족해야 합니다.

- Python 3.7 이상
- PyMongo 라이브러리
- 소스 MongoDB 클러스터와 대상 Amazon DocumentDB 클러스터 모두에 대한 네트워크 연결

설정 및 설치

리포지토리 복제 및 DataDiffer 디렉터리로 이동

```
git clone https://github.com/awslabs/amazon-documentdb-tools.git
cd amazon-documentdb-tools/migration/data-differ
```

필요한 종속 항목 설치

```
pip install -r requirements.txt
```

데이터 유효성 검사 실행

연결 세부 정보가 포함된 구성 파일(예: config.json) 생성

```
{
  "source": {
    "uri": "mongodb://<username>:<password>@<source_endpoint>:27017/?replicaSet=rs0",
    "db": "your_database",
    "collection": "your_collection"
  },
```

```
  "target": {
    "uri": "mongodb://<username>:<password>@<target_endpoint>:27017/?
tls=true&tlsCAFile=global-bundle.pem&replicaSet=rs0&retryWrites=false",
    "db": "your_database",
    "collection": "your_collection"
  },
  "options": {
    "batch_size": 1000,
    "threads": 4,
    "sample_size": 0,
    "verbose": true
  }
}
```

DataDiffer 도구 실행

```
python differ.py --config config.json
```

대규모 컬렉션의 경우 샘플링을 사용하여 데이터의 하위 집합 검증

```
python differ.py --config config.json --sample-size 10000
```

여러 컬렉션을 검증하는 경우 별도의 구성 파일을 생성하거나 배치 모드 사용

```
python differ.py --batch-config batch_config.json
```

결과 해석

도구는 다음을 출력합니다.

- 소스 및 대상의 총 문서
- 일치하는 문서 수
- 누락된 문서 수
- 차이가 있는 문서 수
- 차이에 대한 세부 보고서(있는 경우)

모범 사례

다음은 DataDiffer 도구를 사용할 때의 모범 사례입니다.

- 단계별 실행 - 먼저 문서 수를 검증한 다음 키 문서를 샘플링하고 마지막으로 필요한 경우 전체 비교를 실행합니다.
- 스키마 차이점 확인 - Amazon DocumentDB에는 MongoDB에 비해 몇 가지 제한 사항이 있습니다. 도구는 호환되지 않는 데이터 유형 또는 구조를 강조 표시합니다.
- 정적 기간 동안 검증 - 일관성을 보장하기 위해 쓰기 작업이 최소화될 때 검증을 실행합니다.
- 리소스 사용량 모니터링 - 비교 프로세스는 리소스 집약적일 수 있습니다. 이에 따라 배치 크기와 스레드 수를 조정합니다.
- 인덱스 검증 - 데이터 검증 후 대상 Amazon DocumentDB 클러스터에 필요한 모든 인덱스가 생성되었는지 확인합니다.
- 문서 검증 결과 - 마이그레이션 설명서의 일부로 각 컬렉션에 대한 검증 결과를 기록해 둡니다.

Couchbase Server에서 마이그레이션

주제

- 소개
- Amazon DocumentDB와 비교
- Discovery
- 계획
- 마이그레이션
- 검증

소개

이 가이드에서는 Couchbase Server에서 Amazon DocumentDB로 마이그레이션할 때 고려해야 할 주요 사항을 제공합니다. 마이그레이션의 검색, 계획, 실행 및 검증 단계에 대한 고려 사항을 설명합니다. 또한 오프라인 및 온라인 마이그레이션을 수행하는 방법도 설명합니다.

Amazon DocumentDB와 비교

	카우치베이스 서버	Amazon DocumentDB
데이터 조직	버전 7.0 이상에서는 데이터가 버킷, 범위 및 컬렉션으로 구성됩니다.	데이터는 데이터베이스와 컬렉션으로 구성됩니다.

	카우치베이스 서버	Amazon DocumentDB
	이전 버전에서는 데이터가 버킷으로 구성됩니다.	
호환성	각 서비스(예: 데이터, 인덱스, 검색 등)에는 별도의 APIs가 있습니다. 보조 조회는 ANSI 표준 SQL을 기반으로 하는 쿼리 언어인 SQL++(이전에는 N1QL이라고 함)를 사용하므로 많은 개발자에게 친숙합니다.	Amazon DocumentDB는 MongoDB API와 호환됩니다.
아키텍처	스토리지는 각 클러스터 인스턴스에 연결됩니다. 스토리지와 독립적으로 컴퓨팅을 확장할 수 없습니다.	Amazon DocumentDB는 클라우드용으로 설계되었으며 기존 데이터베이스 아키텍처의 제한을 피하도록 설계되었습니다. 컴퓨팅 계층과 스토리지 계층은 Amazon DocumentDB에서 분리되며 컴퓨팅 계층은 스토리지와 독립적으로 확장할 수 있습니다. Amazon DocumentDB
온디맨드 읽기 용량 추가	인스턴스를 추가하여 클러스터를 확장할 수 있습니다. 스토리지는 서비스가 실행 중인 인스턴스에 연결되므로 확장하는 데 걸리는 시간은 새 인스턴스로 이동하거나 재조정해야 하는 데이터의 양에 따라 달라집니다.	클러스터에 최대 15개의 Amazon DocumentDB 복제본을 생성하여 Amazon DocumentDB 클러스터에 대한 읽기 조정을 수행할 수 있습니다. Amazon DocumentDB 스토리지 계층에는 영향이 없습니다.
노드 장애로부터 빠르게 복구	클러스터에는 자동 장애 조치 기능이 있지만 클러스터를 최대 강도로 되돌리는 데 걸리는 시간은 새 인스턴스로 이동해야 하는 데이터의 양에 따라 달라집니다.	Amazon DocumentDB는 일반적으로 30초 이내에 기본를 장애 조치하고 클러스터의 데이터 양에 관계없이 8~10분 내에 클러스터를 최대 강도로 복원할 수 있습니다.

	카우치베이스 서버	Amazon DocumentDB
데이터 증가에 따라 스토리지 확장	자체 관리형 클러스터의 경우 스토리지 및 IOs 자동으로 확장되지 않습니다.	Amazon DocumentDB 스토리지 및 IOs 자동으로 확장됩니다.
성능에 영향을 주지 않고 데이터 백업	백업은 백업 서비스에서 수행되며 기본적으로 활성화되지 않습니다. 스토리지와 컴퓨팅은 분리되지 않으므로 성능에 영향을 미칠 수 있습니다.	Amazon DocumentDB 백업은 기본적으로 활성화되어 있으며 끌 수 없습니다. 백업은 스토리지 계층에서 처리되므로 컴퓨팅 계층에 영향을 미치지 않습니다. Amazon DocumentDB는 클러스터 스냅샷에서 복원하고 특정 시점으로 복원할 수 있도록 지원합니다.
데이터 내구성	클러스터에는 최대 3개의 데이터 복제본 복사본이 있을 수 있으며, 총 4개의 복사본이 있을 수 있습니다. 데이터 서비스가 실행 중인 각 인스턴스에는 활성 및 1, 2 또는 3개의 데이터 복제본 사본이 있습니다.	Amazon DocumentDB는 쓰기 쿼럼이 4인 컴퓨팅 인스턴스의 수에 관계없이 6개의 데이터 복사본을 유지하며 true를 유지합니다. 클라이언트는 스토리지 계층이 4개의 데이터 복사본을 유지한 후 승인을 받습니다.
일관성	K/V 작업에 대한 즉각적인 일관성이 지원됩니다. Couchbase SDK는 데이터의 활성 사본이 포함된 특정 인스턴스로 K/V 요청을 라우팅하므로 업데이트가 승인되면 클라이언트가 해당 업데이트를 읽도록 보장됩니다. 다른 서비스(인덱스, 검색, 분석, 이벤트)에 대한 업데이트 복제는 최종적으로 일관됩니다.	Amazon DocumentDB 복제본은 최종적으로 일관됩니다. 즉각적인 일관성 읽기가 필요한 경우 클라이언트는 기본 인스턴스에서 읽을 수 있습니다.
복제	Cross-Data Center Replication(XDCR)은 많은 토폴로지에서 필터링된 액티브-패시브/액티브-액티브 데이터 복제를 제공합니다.	Amazon DocumentDB 글로벌 클러스터는 1:many(최대 10개) 토폴로지에서 액티브-패시브 복제를 제공합니다.

Discovery

Amazon DocumentDB로 마이그레이션하려면 기존 데이터베이스 워크로드를 철저히 이해해야 합니다. 워크로드 검색은 Couchbase 클러스터 구성 및 운영 특성, 즉 데이터 세트, 인덱스 및 워크로드를 분석하여 중단을 최소화하면서 원활한 전환을 보장하는 프로세스입니다.

클러스터 구성

카우치베이스는 각 기능이 서비스에 해당하는 서비스 중심 아키텍처를 사용합니다. Couchbase 클러스터에 대해 다음 명령을 실행하여 사용 중인 서비스를 확인합니다(노드에 대한 정보 가져오기 참조).

```
curl -v -u <administrator>:<password> \
  http://<ip-address-or-hostname>:<port>/pools/nodes | \
  jq '[.nodes[].services[]] | unique'
```

샘플 출력:

```
[
  "backup",
  "cbas",
  "eventing",
  "fts",
  "index",
  "kv",
  "n1ql"
]
```

카우치베이스 서비스에는 다음이 포함됩니다.

데이터 서비스(kv)

데이터 서비스는 메모리 및 디스크의 데이터에 대한 읽기/쓰기 액세스를 제공합니다.

Amazon DocumentDB는 MongoDB API를 통해 JSON 데이터에 대한 K/V 작업을 지원합니다.

쿼리 서비스(n1ql)

쿼리 서비스는 SQL++를 통한 JSON 데이터 쿼리를 지원합니다.

Amazon DocumentDB는 MongoDB API를 통한 JSON 데이터 쿼리를 지원합니다.

인덱스 서비스(인덱스)

인덱스 서비스는 데이터에 대한 인덱스를 생성하고 유지 관리하므로 쿼리 속도가 빨라집니다.

Amazon DocumentDB는 MongoDB API를 통해 기본 기본 인덱스와 JSON 데이터에 대한 보조 인덱스 생성을 지원합니다.

검색 서비스(ft)

검색 서비스는 전체 텍스트 검색을 위한 인덱스 생성을 지원합니다.

Amazon DocumentDB의 기본 전체 텍스트 검색 기능을 사용하면 MongoDB API를 사용하여 특수 목적 텍스트 인덱스를 사용하여 대규모 텍스트 데이터 세트에 대한 기본 텍스트 쿼리를 수행할 수 있습니다. 퍼지 검색, 다국어 검색, 교차 컬렉션 검색과 같은 고급 기능의 경우 Amazon DocumentDB 제로 ETL과 Amazon OpenSearch Service의 통합을 사용합니다.

분석 서비스(cbas)

분석 서비스는 거의 실시간으로 JSON 데이터 분석을 지원합니다.

Amazon DocumentDB는 MongoDB API를 통해 JSON 데이터에 대한 임시 쿼리를 지원합니다. Amazon EMR에서 실행되는 Apache Spark를 사용하여 Amazon DocumentDB의 JSON 데이터에 대해 복잡한 쿼리를 실행할 수도 있습니다.

이벤트 서비스(이벤트)

이벤트 서비스는 데이터 변경에 대한 응답으로 사용자 정의 비즈니스 로직을 실행합니다.

Amazon DocumentDB는 Amazon DocumentDB 클러스터에서 데이터가 변경될 때마다 AWS Lambda 함수를 호출하여 이벤트 기반 워크로드를 자동화합니다.

백업 서비스(백업)

백업 서비스는 전체 및 증분 데이터 백업을 예약하고 이전 데이터 백업을 병합합니다.

Amazon DocumentDB는 보존 기간이 1~35일인 Amazon S3에 데이터를 지속적으로 백업하므로 백업 보존 기간 내의 어느 시점으로든 빠르게 복원할 수 있습니다. 또한 Amazon DocumentDB는 이러한 연속 백업 프로세스의 일환으로 데이터의 스냅샷을 자동으로 생성합니다. 를 사용하여 Amazon DocumentDB의 백업 및 복원을 관리할 수도 있습니다 AWS Backup.

운영 특성

카우치베이스용 검색 도구를 사용하여 데이터 세트, 인덱스 및 워크로드에 대한 다음 정보를 가져옵니다. 이 정보는 Amazon DocumentDB 클러스터의 크기를 조정하는 데 도움이 됩니다.

데이터 세트

도구는 다음 버킷, 범위 및 컬렉션 정보를 검색합니다.

1. 버킷 이름
2. 버킷 유형
3. 범위 이름
4. 컬렉션 이름
5. 총 크기(바이트)
6. 총 항목 수
7. 항목 크기(바이트)

인덱스

도구는 모든 버킷에 대해 다음과 같은 인덱스 통계와 모든 인덱스 정의를 검색합니다. Amazon DocumentDB는 각 컬렉션에 대한 기본 인덱스를 자동으로 생성하기 때문에 기본 인덱스는 제외됩니다.

1. 버킷 이름
2. 범위 이름
3. 컬렉션 이름
4. 인덱스 이름
5. 인덱스 크기(바이트)

워크로드

도구는 K/V 및 N1QL 쿼리 지표를 검색합니다. K/V 지표 값은 버킷 수준에서 수집되고 SQL++ 지표는 클러스터 수준에서 수집됩니다.

도구 명령줄 옵션은 다음과 같습니다.

```
python3 discovery.py \
  --username <source cluster username> \
  --password <source cluster password> \
  --data_node <data node IP address or DNS name> \
  --admin_port <administration http REST port> \
  --kv_zoom <get bucket statistics for specified interval> \
  --tools_path <full path to Couchbase tools> \
```

```
  --index_metrics <gather index definitions and SQL++ metrics> \
  --indexer_port <indexer service http REST port> \
  --n1ql_start <start time for sampling> \
  --n1ql_step <sample interval over the sample period>
```

다음은 명령 예제입니다.

```
python3 discovery.py \
  --username username \
  --password ******** \
  --data_node "http://10.0.0.1" \
  --admin_port 8091 \
  --kv_zoom week \
  --tools_path "/opt/couchbase/bin" \
  --index_metrics true \
  --indexer_port 9102 \
  --n1ql_start -60000 \
  --n1ql_step 1000
```

K/V 지표 값은 지난 주 동안 10분마다 샘플을 기반으로 합니다(HTTP 메서드 및 URI 참조). SQL++ 지표 값은 지난 60초 동안 1초마다 샘플을 기반으로 합니다(일반 레이블 참조). 명령의 출력은 다음 파일에 있습니다.

collection-stats.csv – 버킷, 범위 및 수집 정보

```
bucket,bucket_type,scope_name,collection_name,total_size,total_items,document_size
beer-sample,membase,_default,_default,2796956,7303,383
gamesim-sample,membase,_default,_default,114275,586,196
pillowfight,membase,_default,_default,1901907769,1000006,1902
travel-sample,membase,inventory,airport,547914,1968,279
travel-sample,membase,inventory,airline,117261,187,628
travel-sample,membase,inventory,route,13402503,24024,558
travel-sample,membase,inventory,landmark,3072746,4495,684
travel-sample,membase,inventory,hotel,4086989,917,4457
...
```

index-stats.csv – 인덱스 이름 및 크기

```
bucket,scope,collection,index-name,index-size
beer-sample,_default,_default,beer_primary,468144
gamesim-sample,_default,_default,gamesim_primary,87081
travel-sample,inventory,airline,def_inventory_airline_primary,198290
```

```
travel-sample,inventory,airport,def_inventory_airport_airportname,513805
travel-sample,inventory,airport,def_inventory_airport_city,487289
travel-sample,inventory,airport,def_inventory_airport_faa,526343
travel-sample,inventory,airport,def_inventory_airport_primary,287475
travel-sample,inventory,hotel,def_inventory_hotel_city,497125
...
```

kv-stats.csv - 모든 버킷에 대한 지표 가져오기, 설정 및 삭제

```
bucket,gets,sets,deletes
beer-sample,0,0,0
gamesim-sample,0,0,0
pillowfight,369,521,194
travel-sample,0,0,0
```

n1ql-stats.csv – SQL++ 클러스터에 대한 지표 선택, 삭제 및 삽입

```
selects,deletes,inserts
0,132,87
```

indexs-<bucket-name>.txt - 버킷에 있는 모든 인덱스의 인덱스 정의입니다. Amazon DocumentDB는 각 컬렉션에 대한 기본 인덱스를 자동으로 생성하기 때문에 기본 인덱스는 제외됩니다.

```
CREATE INDEX `def_airportname` ON `travel-sample`(`airportname`)
CREATE INDEX `def_city` ON `travel-sample`(`city`)
CREATE INDEX `def_faa` ON `travel-sample`(`faa`)
CREATE INDEX `def_icao` ON `travel-sample`(`icao`)
CREATE INDEX `def_inventory_airport_city` ON `travel-
sample`.`inventory`.`airport`(`city`)
CREATE INDEX `def_inventory_airport_faa` ON `travel-
sample`.`inventory`.`airport`(`faa`)
CREATE INDEX `def_inventory_hotel_city` ON `travel-sample`.`inventory`.`hotel`(`city`)
CREATE INDEX `def_inventory_landmark_city` ON `travel-
sample`.`inventory`.`landmark`(`city`)
CREATE INDEX `def_sourceairport` ON `travel-sample`(`sourceairport`)
...
```

계획

계획 단계에서는 Amazon DocumentDB 클러스터 요구 사항을 결정하고 Couchbase 버킷, 범위 및 컬렉션을 Amazon DocumentDB 데이터베이스 및 컬렉션에 매핑합니다.

Amazon DocumentDB 클러스터 요구 사항

검색 단계에서 수집된 데이터를 사용하여 Amazon DocumentDB 클러스터의 크기를 조정합니다. Amazon DocumentDB 클러스터 크기 조정에 대한 자세한 내용은 인스턴스 크기 조정을 참조하세요.

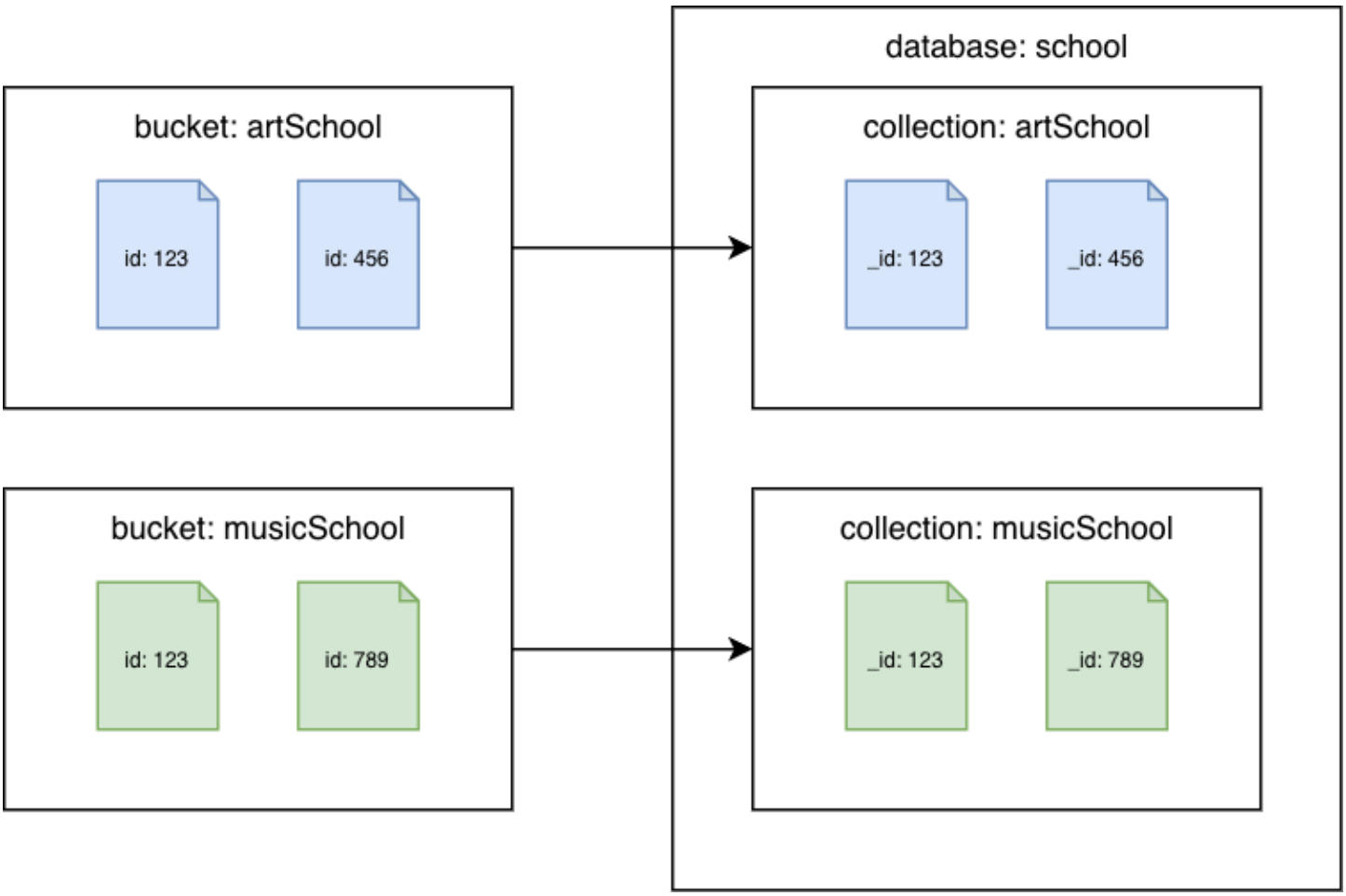
버킷, 범위 및 컬렉션을 데이터베이스 및 컬렉션에 매핑

Amazon DocumentDB 클러스터(들)에 존재할 데이터베이스와 컬렉션을 결정합니다. Couchbase 클러스터에서 데이터가 구성되는 방식에 따라 다음 옵션을 고려하세요. 이러한 옵션이 유일한 옵션은 아니지만 고려해야 할 시작점을 제공합니다.

Couchbase Server 6.x 이하

카우치베이스 버킷에서 Amazon DocumentDB 컬렉션으로

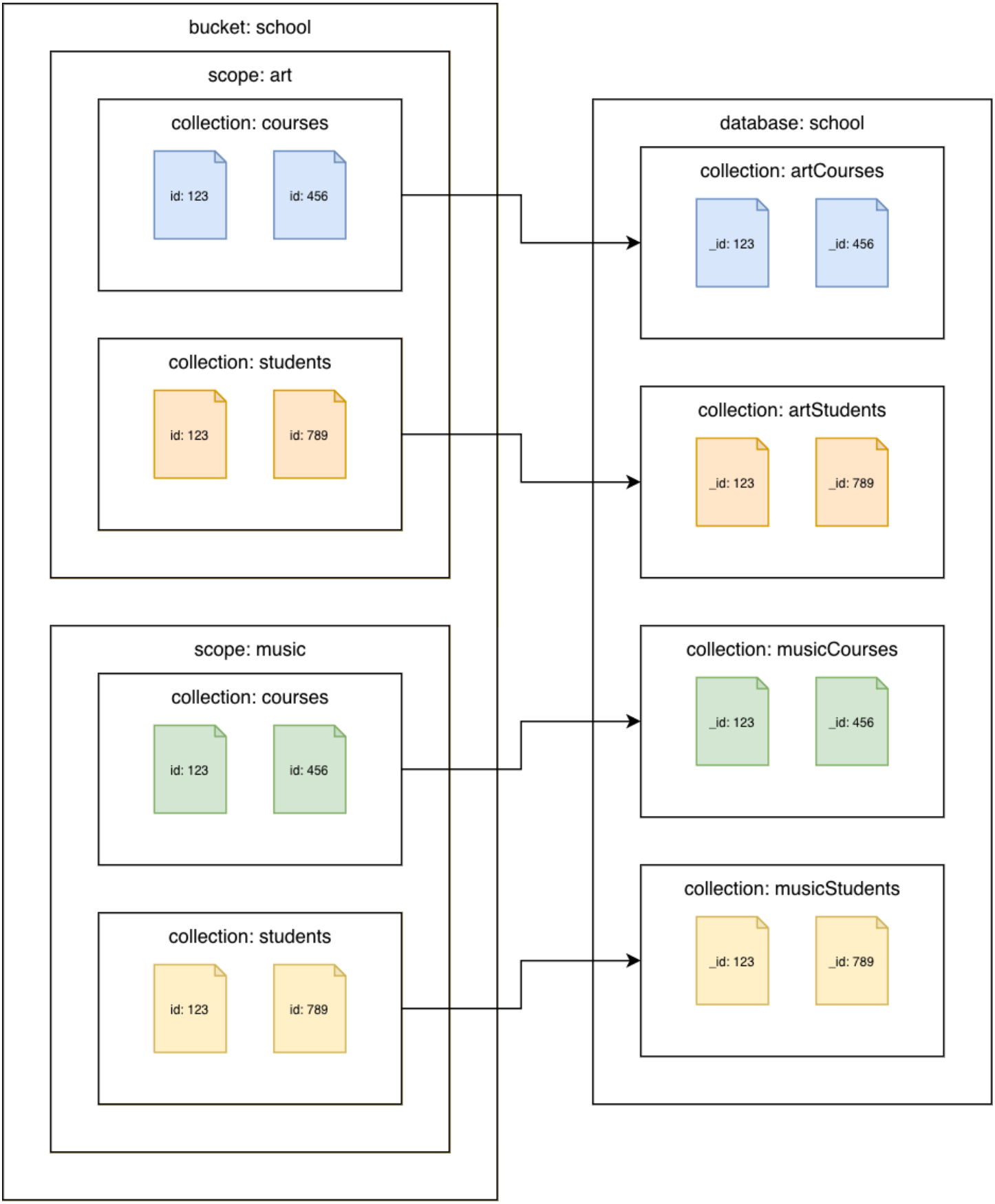
각 버킷을 다른 Amazon DocumentDB 컬렉션으로 마이그레이션합니다. 이 시나리오에서는 Couchbase 문서 `id` 값이 Amazon DocumentDB `_id` 값으로 사용됩니다.



Couchbase Server 7.0 이상

카우치베이스 컬렉션에서 Amazon DocumentDB 컬렉션으로

각 컬렉션을 다른 Amazon DocumentDB 컬렉션으로 마이그레이션합니다. 이 시나리오에서는 Couchbase 문서 `id` 값이 Amazon DocumentDB `_id` 값으로 사용됩니다.


bucket: school
scope: art
collection: courses
id: 123
id: 456
collection: students
id: 123
id: 789
scope: music
collection: courses
id: 123
id: 456
collection: students
id: 123
id: 789
database: school
collection: artCourses
_id: 123
_id: 456
collection: artStudents
_id: 123
_id: 789
collection: musicCourses
_id: 123
_id: 456
collection: musicStudents
_id: 123
_id: 789

마이그레이션

인덱스 마이그레이션

Amazon DocumentDB로 마이그레이션하려면 데이터뿐만 아니라 쿼리 성능을 유지하고 데이터베이스 작업을 최적화하기 위한 인덱스도 전송해야 합니다. 이 섹션에서는 호환성과 효율성을 보장하면서 Amazon DocumentDB로 인덱스를 마이그레이션하는 step-by-step 프로세스를 간략하게 설명합니다.

Amazon Q(AWS Management Console 또는 IDE에서 사용 가능)를 사용하여 SQL++ `CREATE INDEX` 문을 Amazon DocumentDB `createIndex()` 명령으로 변환합니다.

1. Couchbase용 검색 도구에서 생성한 indexes-<bucket name>.txt 파일(들)을 업로드합니다.
2. 다음 프롬프트를 입력합니다.

```
Convert the Couchbase CREATE INDEX statements to Amazon DocumentDB
createIndex commands
```

Amazon Q는 동등한 Amazon DocumentDB `createIndex()` 명령을 생성합니다. Couchbase 버킷, 범위 및 컬렉션을 Amazon DocumentDB 컬렉션에 매핑한 방법에 따라 컬렉션 이름을 업데이트해야 할 수 있습니다.

예제:

indexes-beer-sample.txt

```
CREATE INDEX `beerType` ON `beer-sample`(`type`)
CREATE INDEX `code` ON `beer-sample`(`code`) WHERE (`type` = "brewery")
```

Amazon Q 출력 예제(발췌):

```
db.beerSample.createIndex(
  { "type": 1 },
  {
    "name": "beerType",
    "background": true
  }
)

db.beerSample.createIndex(
  { "code": 1 },
  {
```

```
    "name": "code",
    "background": true,
    "partialFilterExpression": { "type": "brewery" }
  }
)
```

Amazon Q가 변환할 수 없는 인덱스에 대한 자세한 내용은 Amazon DocumentDB 인덱스 및 인덱스 및 인덱스 속성 관리를 참조하세요. mongo-apis.html#mongo-apis-index

MongoDB APIs를 사용하기 위한 코드 리팩터링

클라이언트는 Couchbase SDKs를 사용하여 Couchbase Server에 연결합니다. Amazon DocumentDB 클라이언트는 MongoDB 드라이버를 사용하여 Amazon DocumentDB에 연결합니다. Couchbase SDKs에서 지원하는 모든 언어는 MongoDB 드라이버에서도 지원됩니다. 해당 언어의 드라이버에 대한 자세한 내용은 MongoDB 드라이버를 참조하세요.

APIs는 Couchbase Server와 Amazon DocumentDB 간에 다르므로 적절한 MongoDB APIs. Amazon Q를 사용하여 K/V API 호출 및 SQL++ 쿼리를 동등한 MongoDB APIs.

1. 소스 코드 파일(들)을 업로드합니다.
2. 다음 프롬프트를 입력합니다.

```
Convert the Couchbase API code to Amazon DocumentDB API code
```

Hello Couchbase Python 코드 샘플을 사용하여 Amazon Q는 다음을 생성합니다(발췌).

```
from pymongo import MongoClient

# Connection parameters
database_name = "travel-sample"

# Connect to Amazon DocumentDB cluster
client = MongoClient('<Amazon DocumentDB connection string>',
                     tls=True,
                     tlsCAFile='global-bundle.pem')

# Get reference to database and collection
db = client['travel-sample']
airline_collection = db['airline']

# upsert document function
```

```
def upsert_document(doc):
    print("\nUpsert Result: ")
    try:
        # key will equal: "airline_8091"
        key = doc["type"] + "_" + str(doc["id"])
        doc['_id'] = key # Amazon DocumentDB uses _id as primary key
        result = airline_collection.update_one(
            {'_id': key},
            {'$set': doc},
            upsert=True
        )
        print(f"Modified count: {result.modified_count}")
    except Exception as e:
        print(e)

# get document function
def get_airline_by_key(key):
    print("\nGet Result: ")
    try:
        result = airline_collection.find_one({'_id': key})
        print(result)
    except Exception as e:
        print(e)

# query for document by callsign
def lookup_by_callsign(cs):
    print("\nLookup Result: ")
    try:
        result = airline_collection.find(
            {'callsign': cs},
            {'name': 1, '_id': 0}
        )
        for doc in result:
            print(doc['name'])
    except Exception as e:
        print(e)

# Test document
airline = {
    "type": "airline",
    "id": 8091,
    "callsign": "CBS",
    "iata": None,
    "icao": None,
```

```
    "name": "Couchbase Airways",
}

upsert_document(airline)
get_airline_by_key("airline_8091")
lookup_by_callsign("CBS")
```

Python, Node.js, PHP, Go, Java, C#/에서 Amazon DocumentDB에 연결하는 예제는 Amazon DocumentDB에 프로그래밍 방식으로 연결을 참조하세요. Amazon DocumentDB NET, R 및 Ruby.

마이그레이션 접근 방식 선택

Amazon DocumentDB로 데이터를 마이그레이션할 때 다음 두 가지 옵션이 있습니다.

1. 오프라인 마이그레이션
2. 온라인 마이그레이션

> **Note**
>
> Amazon DocumentDB는 `_id` 필드를 각 문서의 기본 키로 사용합니다. 이 가이드의 마이그레이션 접근 방식은 Couchbase 문서 키를 `_id` 필드에 저장합니다. 오프라인 마이그레이션의 경우의 `--include-key _id` 옵션을 통해 수행됩니다`cbexport`. 온라인 마이그레이션의 경우 Kafka 커넥터의를 통해 수행됩니다`ProvidedInKeyStrategy`.

오프라인 마이그레이션

다음과 같은 경우 오프라인 마이그레이션을 고려하세요.

- 가동 중지 허용: 오프라인 마이그레이션에는 소스 데이터베이스에 대한 쓰기 작업 중지, 데이터 내보내기, Amazon DocumentDB로 가져오기가 포함됩니다. 이 프로세스는 애플리케이션에 가동 중지 시간을 발생시킵니다. 애플리케이션 또는 워크로드가이 기간 동안 사용할 수 없는 경우 오프라인 마이그레이션이 실행 가능한 옵션입니다.
- 소규모 데이터 세트 마이그레이션 또는 개념 증명 수행: 소규모 데이터 세트의 경우 내보내기 및 가져오기 프로세스에 필요한 시간이 비교적 짧기 때문에 오프라인 마이그레이션이 빠르고 간단한 방법입니다. 가동 중지 시간이 덜 중요한 개발, 테스트 및 proof-of-concept 환경에도 적합합니다.
- 단순성이 우선 순위입니다. cbexport와 mongoimport를 사용하는 오프라인 방법은 일반적으로 데이터를 마이그레이션하는 가장 간단한 접근 방식입니다. 온라인 마이그레이션 방법과 관련된 변경 데이터 캡처(CDC)의 복잡성을 방지합니다.

- 지속적인 변경 사항을 복제할 필요가 없음: 마이그레이션 중에 소스 데이터베이스가 변경 사항을 적극적으로 수신하지 않거나 마이그레이션 프로세스 중에 이러한 변경 사항을 캡처하여 대상에 적용하는 것이 중요하지 않은 경우 오프라인 접근 방식이 적합합니다.

주제

- Couchbase Server 6.x 이하
- Couchbase Server 7.0 이상

Couchbase Server 6.x 이하

카우치베이스 버킷에서 Amazon DocumentDB 컬렉션으로

cbexport json을 사용하여 데이터를 내보내 버킷에 있는 모든 데이터의 JSON 덤프를 생성합니다. `--format` 옵션의 경우 `lines` 또는를 사용할 수 있습니다`list`.

```
cbexport json \
  --cluster <source cluster endpoint> \
  --bucket <bucket name> \
  --format <lines | list> \
  --username <username> \
  --password <password> \
  --output export.json \
  --include-key _id
```

적절한 옵션과 함께 mongoimport를 사용하여 Amazon DocumentDB 컬렉션으로 데이터를 가져와 줄 또는 목록을 가져옵니다.

줄:

```
mongoimport \
  --db <database> \
  --collection <collection> \
  --uri "<Amazon DocumentDB cluster connection string>" \
  --ssl \
  --sslCAFile global-bundle.pem \
  --file export.json
```

목록:

```
mongoimport \
```

```
  --db <database> \
  --collection <collection> \
  --uri "<Amazon DocumentDB cluster connection string>" \
  --ssl \
  --sslCAFile global-bundle.pem \
  --jsonArray \
  --file export.json
```

Couchbase Server 7.0 이상

오프라인 마이그레이션을 수행하려면 cbexport 및 mongoimport 도구를 사용합니다.

기본 범위 및 기본 컬렉션이 있는 카우치베이스 버킷

cbexport json을 사용하여 데이터를 내보내 버킷에 있는 모든 컬렉션의 JSON 덤프를 생성합니다. `--format` 옵션의 경우 `lines` 또는를 사용할 수 있습니다`list`.

```
cbexport json \
  --cluster <source cluster endpoint> \
  --bucket <bucket name> \
  --format <lines | list> \
  --username <username> \
  --password <password> \
  --output export.json \
  --include-key _id
```

적절한 옵션과 함께 mongoimport를 사용하여 Amazon DocumentDB 컬렉션으로 데이터를 가져와 줄 또는 목록을 가져옵니다.

줄:

```
mongoimport \
  --db <database> \
  --collection <collection> \
  --uri "<Amazon DocumentDB cluster connection string>" \
  --ssl \
  --sslCAFile global-bundle.pem \
  --file export.json
```

목록:

```
mongoimport \
```

```
  --db <database> \
  --collection <collection> \
  --uri "<Amazon DocumentDB cluster connection string>" \
  --ssl \
  --sslCAFile global-bundle.pem \
  --jsonArray \
  --file export.json
```

카우치베이스 컬렉션에서 Amazon DocumentDB 컬렉션으로

cbexport json을 사용하여 데이터를 내보내 각 컬렉션의 JSON 덤프를 생성합니다. `--include-data` 옵션을 사용하여 각 컬렉션을 내보냅니다. `--format` 옵션의 경우 `lines` 또는를 사용할 수 있습니다`list`. `--scope-field` 및 `--collection-field` 옵션을 사용하여 범위 및 모음의 이름을 각 JSON 문서의 지정된 필드에 저장합니다.

```
cbexport json \
  --cluster <source cluster endpoint> \
  --bucket <bucket name> \
  --include-data <scope name>.<collection name> \
  --format <lines | list> \
  --username <username> \
  --password <password> \
  --output export.json \
  --include-key _id \
  --scope-field "_scope" \
  --collection-field "_collection"
```

cbexport는 내보낸 모든 문서에 `_scope` 및 `_collection` 필드를 추가했으므로 검색 및 교체, `sed`또는 원하는 방법을 통해 내보내기 파일의 모든 문서에서 제거할 수 있습니다.

행 또는 목록을 가져오는 적절한 옵션과 함께 mongoimport를 사용하여 각 컬렉션의 데이터를 Amazon DocumentDB 컬렉션으로 가져옵니다.

줄:

```
mongoimport \
  --db <database> \
  --collection <collection> \
  --uri "<Amazon DocumentDB cluster connection string>" \
  --ssl \
  --sslCAFile global-bundle.pem \
```

```
  --file export.json
```

목록:

```
mongoimport \
  --db <database> \
  --collection <collection> \
  --uri "<Amazon DocumentDB cluster connection string>" \
  --ssl \
  --sslCAFile global-bundle.pem \
  --jsonArray \
  --file export.json
```

온라인 마이그레이션

가동 중지 시간을 최소화해야 하고 지속적인 변경 사항을 거의 실시간으로 Amazon DocumentDB에 복제해야 하는 경우 온라인 마이그레이션을 고려하세요.

Amazon DocumentDB로 라이브 마이그레이션을 수행하는 방법을 알아보려면 Couchbase에서 Amazon DocumentDB로 라이브 마이그레이션을 수행하는 방법을 참조하세요. 이 설명서에서는 솔루션을 배포하고 버킷을 Amazon DocumentDB 클러스터로 라이브 마이그레이션하는 방법을 안내합니다.

주제

- Couchbase Server 6.x 이하
- Couchbase Server 7.0 이상

Couchbase Server 6.x 이하

카우치베이스 버킷에서 Amazon DocumentDB 컬렉션으로

Couchbase용 마이그레이션 유틸리티는 Couchbase 버킷을 Amazon DocumentDB 컬렉션으로 온라인 마이그레이션하도록 사전 구성되어 있습니다. 싱크 커넥터 구성을 보면 `document.id.strategy` 파라미터는 메시지 키 값을 `_id` 필드 값으로 사용하도록 구성됩니다(싱크 커넥터 ID 전략 속성 참조).

```
ConnectorConfiguration:
  document.id.strategy:
 'com.mongodb.kafka.connect.sink.processor.id.strategy.ProvidedInKeyStrategy'
```

Couchbase Server 7.0 이상

기본 범위 및 기본 컬렉션이 있는 카우치베이스 버킷

Couchbase용 마이그레이션 유틸리티는 Couchbase 버킷을 Amazon DocumentDB 컬렉션으로 온라인 마이그레이션하도록 사전 구성되어 있습니다. 싱크 커넥터 구성을 보면 `document.id.strategy` 파라미터는 메시지 키 값을 `_id` 필드 값으로 사용하도록 구성됩니다(싱크 커넥터 ID 전략 속성 참조).

```
ConnectorConfiguration:
  document.id.strategy:
 'com.mongodb.kafka.connect.sink.processor.id.strategy.ProvidedInKeyStrategy'
```

카우치베이스 컬렉션에서 Amazon DocumentDB 컬렉션으로

각 범위의 각 Couchbase 컬렉션을 별도의 주제로 스트리밍하도록 소스 커넥터를 구성합니다(소스 구성 옵션 참조). 예제:

```
ConnectorConfiguration:
  # add couchbase.collections configuration
  couchbase.collections: '<scope 1>.<collection 1>, <scope 1>.<collection 2>, ...'
```

각 주제에서 별도의 Amazon DocumentDB 컬렉션으로 스트리밍하도록 싱크 커넥터를 구성합니다(싱크 커넥터 구성 속성 참조). 예제:

```
ConnectorConfiguration:
  # remove collection configuration
  #collection: 'test'

  # modify topics configuration
  topics: '<bucket>.<scope 1>.<collection 1>, <bucket>.<scope 1>.<collection 2>, ...'

  # add topic.override.%s.%s configurations for each topic
  topic.override.<bucket>.<scope 1>.<collection 1>.collection: '<collection>'
```

```
  topic.override.<bucket>.<scope 1>.<collection 2>.collection: '<collection>'
```

검증

이 섹션에서는 Amazon DocumentDB로 마이그레이션한 후 데이터 일관성과 무결성을 확인하는 자세한 검증 프로세스를 제공합니다. 검증 단계는 마이그레이션 방법에 관계없이 적용됩니다.

주제

- 모든 컬렉션이 대상에 존재하는지 확인
- 소스 클러스터와 대상 클러스터 간의 문서 수 확인
- 소스 클러스터와 대상 클러스터 간의 문서 비교

모든 컬렉션이 대상에 존재하는지 확인

카우치베이스 소스

옵션 1: 워크벤치 쿼리

```
SELECT RAW `path`
  FROM system:keyspaces
  WHERE `bucket` = '<bucket>'
```

옵션 2: cbq 도구

```
cbq \
  -e <source cluster endpoint> \
  -u <username> \
  -p <password> \
  -q "SELECT RAW `path`
       FROM system:keyspaces
       WHERE `bucket` = '<bucket>'"
```

Amazon DocumentDB 대상

mongosh(Amazon DocumentDB 클러스터에 연결 참조):

```
db.getSiblingDB('<database>')
```

```
db.getCollectionNames()
```

소스 클러스터와 대상 클러스터 간의 문서 수 확인

카우치베이스 소스

Couchbase Server 6.x 이하

옵션 1: 워크벤치 쿼리

```
SELECT COUNT(*)
FROM `<bucket>`
```

옵션 2: cbq

```
cbq \
  -e <source cluster endpoint> \
  -u <username> \
  -p <password> \
  -q "SELECT COUNT(*)
       FROM `<bucket>`"
```

Couchbase Server 7.0 이상

옵션 1: 워크벤치 쿼리

```
SELECT COUNT(*)
FROM `<bucket>`.`<scope>`.`<collection>`
```

옵션 2: cbq

```
cbq \
  -e <source cluster endpoint> \
  -u <username> \
  -p <password> \
  -q "SELECT COUNT(*)
       FROM `<bucket>`.`<scope>`.`<collection>`"
```

Amazon DocumentDB 대상

mongosh(Amazon DocumentDB 클러스터에 연결 참조):

```
db = db.getSiblingDB('<database>')
db.getCollection('<collection>').countDocuments()
```

소스 클러스터와 대상 클러스터 간의 문서 비교

카우치베이스 소스

Couchbase Server 6.x 이하

옵션 1: 워크벤치 쿼리

```
SELECT META().id as _id, *
FROM `<bucket>`
LIMIT 5
```

옵션 2: cbq

```
cbq \
  -e <source cluster endpoint> \
  -u <username> \
  -p <password> \
  -q "SELECT META().id as _id, *
       FROM `<bucket>`
       LIMIT 5"
```

Couchbase Server 7.0 이상

옵션 1: 워크벤치 쿼리

```
SELECT META().id as _id, *
FROM `<bucket>`.`<scope>`.`<collection>`
LIMIT 5
```

옵션 2: cbq

```
cbq \
  -e <source cluster endpoint> \
  -u <username> \
  -p <password> \
```

```
  -q "SELECT META().id as _id, *
       FROM `<bucket>`.`<scope>`.`<collection>`
       LIMIT 5"
```

Amazon DocumentDB 대상

mongosh(Amazon DocumentDB 클러스터에 연결 참조):

```
db = db.getSiblingDB('<database>')
db.getCollection('<collection>').find({
  _id: {
    $in: [
      <_id 1>, <_id 2>, <_id 3>, <_id 4>, <_id 5>
    ]
  }
})
```

Amazon DocumentDB 엔진 버전 지원 날짜

다음 날짜를 사용하여 테스트 및 업그레이드 주기를 계획할 수 있습니다.

Amazon DocumentDB 메이저 버전 릴리스 일정

Amazon DocumentDB는 현재 다음과 같은 메이저 버전을 지원합니다. 엔진 버전을 업그레이드하려면 메이저 엔진 버전 업그레이드 또는 섹션을 참조하세요를 사용하여 업그레이드 AWS DMS.

엔진 버전	릴리스 날짜	표준 지원 종료일	추가 지원 시작일(1년 차 요금)	추가 지원 시작일(3년 차 요금)	추가 지원 종료일
버전 3.6	2019년 1월 9일	2026년 3월 30일	2026년 3월 31일	2028년 3월 31일	2029년 3월 30일
버전 4.0	2020년 11월 9일	해당 사항 없음	해당 사항 없음	해당 사항 없음	해당 사항 없음
버전 5.0	2023년 3월 1일	해당 사항 없음	해당 사항 없음	해당 사항 없음	해당 사항 없음

엔진 버전	릴리스 날짜	표준 지원 종료일	추가 지원 시작일(1년 차 요금)	추가 지원 시작일(3년 차 요금)	추가 지원 종료일
버전 8.0	2025년 11월 14일	해당 사항 없음	해당 사항 없음	해당 사항 없음	해당 사항 없음

Amazon DocumentDB는 여러 엔진 버전을 표준 지원에 따라 지원합니다. 표준 지원 종료일이 지난 후에도 추가 지원 수수료를 지불하면 버전을 계속 실행할 수 있습니다. 자세한 내용은 Amazon DocumentDB 추가 지원 및 Amazon DocumentDB 요금을 참조하세요.

Amazon DocumentDB 마이너 버전 릴리스 일정

Amazon DocumentDB는 현재 다음과 같은 마이너 버전을 지원합니다. 릴리스 일정은 추가 기능 또는 수정 사항에 따라 달라집니다. 마이너 버전은 해당 메이저 버전보다 먼저 표준 지원이 종료될 수 있습니다. 엔진 버전을 업그레이드하려면 섹션을 참조하세요마이너 엔진 버전 업그레이드.

> **Note**
>
> 마이너 버전은 메이저 버전 5.0부터 사용할 수 있습니다.

엔진 버전	릴리스 날짜
버전 5.0.0(LTS)	2026년 2월 16일
버전 5.0.1	2026년 6월 8일

Amazon DocumentDB는 특정 버전을 장기 지원(LTS) 릴리스로 지정합니다. 자세한 내용은 LTS(장기 지원) 릴리스 사용 단원을 참조하십시오.

Amazon DocumentDB 인플레이스 주요 버전 업그레이드

동일한 엔드포인트, 스토리지 및 태그를 유지하면서 Amazon DocumentDB 클러스터의 인플레이스 메이저 버전 업그레이드(MVU)를 수행할 수 있습니다. 애플리케이션은 수정 없이 계속 작동합니다. 이 기

능은 Amazon DocumentDB를 사용할 수 AWS 리전 있는 모든에서 추가 비용 없이 사용할 수 있습니다.

> **Important**
>
> 업그레이드 중에는 클러스터를 사용할 수 없으며 여러 번 재부팅됩니다. 업그레이드를 시작한 후에는 클러스터에 연결하거나 클러스터에서 읽거나 클러스터에 쓰지 마십시오. 가동 중지 시간은 컬렉션, 인덱스, 데이터베이스 및 인스턴스 수에 따라 달라집니다. 유지 관리 기간 또는 사용률이 낮은 시간에 업그레이드를 수행합니다.
>
> 업그레이드한 후에는 이전 버전으로 다운그레이드할 수 없습니다. 필요한 경우 업그레이드 전 스냅샷을 새 클러스터로 복원할 수 있습니다.

주제

- 지원되는 업그레이드 경로
- 사전 조건
- 모범 사례
- Amazon DocumentDB 8.0으로 업그레이드한 후 변경되는 사항
- 3.6 또는 4.0에서 업그레이드된 클러스터에 대한 업그레이드 후 고려 사항
- 업그레이드 수행
- 문제 해결

지원되는 업그레이드 경로

다음 표에는 각 소스 버전에 대해 선택할 수 있는 대상 엔진 버전이 나열되어 있습니다. 먼저 `.0` 마이너 버전으로 업그레이드한 다음 마이너 버전 업그레이드를 적용할 필요는 없습니다.

소스 버전	지원되는 대상 버전
3.6	게시된 모든 5.0 또는 8.0 마이너 버전
4.0	게시된 모든 5.0 또는 8.0 마이너 버전
5.0.0	8.0.0, 8.0.1, 8.0.2
5.0.1	8.0.0, 8.0.1, 8.0.2

소스 버전	지원되는 대상 버전
5.0.2	8.0.2

업그레이드 후 고려 사항은 the section called “3.6 또는 4.0에서 업그레이드된 클러스터에 대한 업그레이드 후 고려 사항” 및 섹션을 참조하세요the section called “Amazon DocumentDB 8.0으로 업그레이드한 후 변경되는 사항”.

AWS 리전에서 사용 가능한 마이너 버전을 보려면 AWS CLI 명령을 사용합니다`aws docdb describe-db-engine-versions`. 릴리스된 마이너 버전 목록은 섹션을 참조하세요릴리스 노트.

> **Note**
>
> 현재 위치 MVU는 글로벌 클러스터 또는 탄력적 클러스터에서 지원되지 않습니다. 글로벌 클러스터를 업그레이드하려면 보조 클러스터를 제거하고, 기본 클러스터를 리전 클러스터로 변환하고, MVU를 수행한 다음 동일한 이름을 사용하여 보조 클러스터를 추가하여 글로벌 클러스터를 다시 생성하여 엔드포인트를 유지합니다. 업그레이드된 기본가 새 보조에 데이터를 복제하는 동안 I/O 요금이 발생합니다. 자세한 단계는 Amazon DocumentDB 글로벌 클러스터에서 클러스터 분리 섹션을 참조하세요.

사전 조건

> **Important**
>
> 업그레이드하기 전에 버스트 가능한 인스턴스를 확장합니다. 클러스터가 버스트 가능 인스턴스 유형(예: `db.t3.medium` 또는 `db.t4g.medium`)을 사용하는 경우 업그레이드를 시작하기 `db.r6g.large` 전에 기본 인스턴스를 최소한 `db.r5.large` 또는 로 확장합니다. 버스트 가능 인스턴스에는 업그레이드 프로세스를 완료하기에 충분한 CPU와 메모리가 없을 수 있으며, 이로 인해 업그레이드 실패 및 클러스터 가용성 확장이 발생할 수 있습니다. 업그레이드가 완료된 후 다시 축소할 수 있습니다.
>
> 업그레이드하기 전에 서버리스 라이터 인스턴스를 확장합니다. 클러스터에 서버리스 라이터 인스턴스가 있는 경우 업그레이드를 시작하기 전에의 최대 DCU를 3 이상으로 설정해야 `ServerlessV2ScalingConfiguration` 합니다. 최대 DCU가 3 미만인 클러스터는 업그레이드를 완료하기에 충분한 CPU 또는 메모리가 없기 때문에 업그레이드할 수 없습니다.
>
> 업그레이드하기 전에 부분 인덱스를 확인합니다(Amazon DocumentDB 5.0에서 8.0으로 업그레이드). 업그레이드를 시작하기 전에 연`$type`산`partialFilterExpression`자를 사

용하여 특정 BSON 유형, 특히 , binData, booldate, doubledecimal, 유형 array별칭 또는와 같은 숫자 유형 코드와 일치하는에 대한 부분 인덱스를 검토합니다{ "$type" : 1 }. 인덱스가 읽기 및 쓰기에 정상적으로 작동하더라도 이러한 부분 인덱스로 인해 업그레이드가 실패합니다. db.collection.getIndexes()를 사용하여 부분 인덱스partialFilterExpression의를 검토합니다. 업그레이드를 시작하기 전에 영향을 받는 부분 인덱스를 삭제하고 업그레이드가 완료된 후 다시 생성합니다. 인덱스가 영향을 받는지 확실하지 않은 경우 업그레이드하기 전에 AWS 지원팀에 문의하십시오.

- 인스턴스 유형 - Amazon DocumentDB 4.0+는 db.r4 인스턴스를 지원하지 않습니다. 업그레이드하기 전에 db.r4.* 인스턴스를 db.r5.* 인스턴스 이상으로 수정합니다. Amazon DocumentDB 인스턴스 수정 및 리전별 지원되는 인스턴스 클래스 섹션을 참조하세요.
- OS 패치 - 업그레이드하기 전에 모든 인스턴스에 보류 중인 OS 유지 관리 작업을 적용합니다. Amazon DocumentDB 운영 체제 업데이트 섹션을 참조하세요.

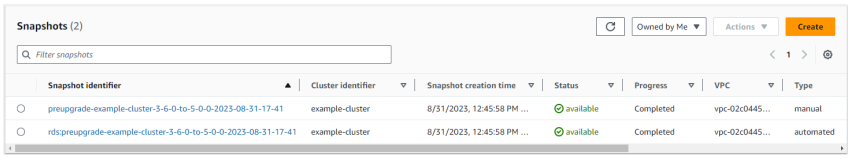
  > **Note**
  >
  > 보류 중인 클러스터 수준 엔진 패치는 인스턴스 OS 패치를 숨길 수 있습니다. 필요한 경우 먼저 엔진 패치를 적용합니다. 클러스터의 엔진 버전에 대한 패치 업데이트 수행 섹션을 참조하세요.

- 버스트 가능 인스턴스(t 패밀리 인스턴스)의 인덱스 제한 - 버스트 가능 인스턴스에 3,000개 이상의 인덱스가 있는 경우 업그레이드하기 전에 기본를 최소 db.r5.large로 확장합니다. 업그레이드가 완료된 후 다시 축소할 수 있습니다.

Instance	MVU의 최대 인덱스
db.t4g.medium	3천
db.t3.medium	1만

- 파라미터 그룹 - 업그레이드하기 전에 대상 버전에 대한 사용자 지정 클러스터 파라미터 그룹을 준비합니다. 지정하지 않으면 대상 버전의 기본 파라미터 그룹이 사용됩니다(예: default.docdb5.0 또는 default.docdb8.0).
- 수동 스냅샷 - 업그레이드하기 전에 수동 스냅샷을 생성합니다. 업그레이드 프로세스는 라는 자동 스냅샷을 생성preupgrade-<name>-<version>-<timestamp>하지만 항상 자체 백업을 생성합니다. 수동 클러스터 스냅샷 생성 섹션을 참조하세요.

> **Note**
>
> 업그레이드 프로세스에서 생성된 자동 스냅샷은 인플레이스 주요 버전 업그레이드가 완료된 후에도 자동으로 삭제되지 않습니다. 이 스냅샷은 보존 기간 내에 있는 한 요금이 부과되지 않습니다. 클러스터의 성공적인 업그레이드가 확인되면 이 스냅샷을 삭제하도록 선택할 수 있습니다.



모범 사례

업그레이드 전 - 복제본으로 테스트

1. Amazon DocumentDB 클러스터에 대한 볼륨 복제를 사용하여 클러스터의 복제본을 생성합니다. 복제본의 데이터를 수정하지 않는 한 스토리지 비용이 발생하지 않습니다.
2. 실제 시간 추정을 위해 복제본의 인스턴스 수를 대상 클러스터와 일치시킵니다.
3. 복제본에서 MVU를 수행하고 기능적 차이를 완전히 테스트합니다.
4. 를 실행`describe-db-clusters`하고를 찾아 업그레이드가 이미 예약되어 있는지 확인합니다`PendingModifiedValues.EngineVersion`. 클러스터를 수정하고 다음 유지 관리 기간에 적용하도록 선택한 경우 콘솔에 일정이 표시되지 않지만 AWS CLI다음에서 볼 수 있습니다.

```
aws docdb describe-db-clusters \
  --region us-east-1 \
  --db-cluster-identifier mydocdbcluster
```

```
"PendingModifiedValues": {
    "EngineVersion": "5.0.0"
},
```

5. 테스트에 성공하면 프로덕션 클러스터에서 업그레이드를 진행합니다.

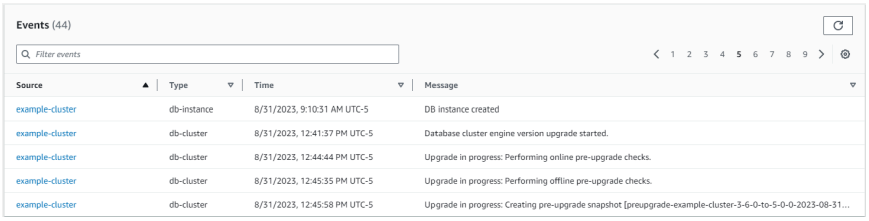
업그레이드 중

클러스터 유지 관리 이벤트를 구독하면 인플레이스 주요 버전 업그레이드 진행 상황을 모니터링할 수 있습니다. 업그레이드가 완료되면 "데이터베이스 클러스터 메이저 버전이 업그레이드됨" 이벤트를 받게 됩니다. 업그레이드 중에 발생하는이 이벤트 및 기타 이벤트는 Amazon DocumentDB 콘솔의 클러스터 세부 정보 페이지의 이벤트 및 태그 섹션에 표시됩니다. 그러면 클러스터 상태가에서 upgrading로 변경됩니다available.

업그레이드 중에 다음 이벤트가 생성됩니다.

1. 데이터베이스 클러스터 엔진 메이저 버전 업그레이드가 시작되었습니다. 클러스터는 온라인 상태로 유지됩니다.
2. 업그레이드 준비 진행 중: 온라인 업그레이드 사전 확인 시작.
3. 업그레이드 준비 진행 중: 온라인 업그레이드 사전 확인 완료.
4. 기본 인스턴스가 패치/업그레이드 프로세스를 완료하는 동안 데이터베이스 클러스터를 오프라인으로 전환합니다.
5. 업그레이드 준비 진행 중: 오프라인 업그레이드 사전 확인 시작.
6. 업그레이드 준비 진행 중: 오프라인 업그레이드 사전 확인을 완료했습니다.
7. 업그레이드 진행 중: 업그레이드 전 스냅샷 생성[preupgrade-<cluster-name>-<version-from>-to-<version-to>-<timestamp>].
8. 업그레이드 진행 중: 볼륨 복제.
9. 업그레이드 진행 중: 라이터 업그레이드.
10. 업그레이드 진행 중: 리더 업그레이드.
11. 데이터베이스 클러스터 엔진 메이저 버전이 업그레이드되었습니다.

이벤트는 콘솔의 이벤트 페이지에 표시됩니다.



에서 AWS CLI를 실행aws docdb describe-events하여 업그레이드 진행 상황을 모니터링할 수 있습니다. 알림을 자동으로 수신하려면 aws docdb create-event-subscription를 사용하여 이벤트를 구독하고 이메일, 푸시 메시지 또는 기타 전송 방법을 위해 Amazon SNS로 라우팅합니다. 자세한 내용은 Amazon DocumentDB 이벤트 구독 섹션을 참조하세요.

```
aws docdb describe-events
  --source-identifier mydocdbcluster
  --source-type db-cluster
```

이 명령은 다음과 비슷한 출력을 반환합니다.

```
{
    "Events": [
        {
            "SourceIdentifier": "mydocdbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster engine version upgrade started.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2023-07-11T23:20:32.444000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:xxxx:cluster:mycluster"
        }
    ]
}
```

업그레이드 후

Warning

인플레이스 메이저 버전 업그레이드 직후 Amazon DocumentDB 클러스터는 데이터베이스 엔진이 쿼리 실행 계획을 최적화하는 데 사용하는 인덱스 메타데이터를 다시 채웁니다. 이 프로세스가 완료되면 쿼리 성능이 예상 수준으로 돌아갑니다. 일반적으로 몇 분 내에 완료되지만 클러스터의 인덱스 수에 따라 최대 2시간이 걸릴 수 있습니다.
이 시간 동안 라이터 인스턴스를 재부팅, 장애 조치 또는 스케일 업/다운하지 마십시오. 인덱스 메타데이터 재계산이 중단될 수 있습니다. 이러한 변경을 수행하기 전에 예상 쿼리 성능이 관찰될 때까지 기다립니다.
다음 클러스터 이벤트를 통해 진행 상황을 추적합니다.

- 업그레이드 후 클러스터 상태: 인덱스 메타데이터 새로 고침 프로세스가 시작됨
- 업그레이드 후 클러스터 상태: X초 내에 완료된 인덱스 메타데이터 새로 고침 프로세스

인덱스 메타데이터 새로 고침 프로세스가 3시간 이내에 완료되지 않았거나 프로세스가 완료된 후에도 성능 문제가 계속 발생하는 경우 AWS 지원팀에 문의하세요.

1. 업그레이드 후 상태로 복원해야 하는 경우 업그레이드된 클러스터의 수동 스냅샷을 생성합니다. 인플레이스 메이저 버전 업그레이드가 완료되는 즉시 자동 스냅샷 프로세스가 재개됩니다.
2. 3.6에서 업그레이드된 태그 클러스터. 태그를 추가하여 3.6에서 업그레이드된 클러스터를 구분합니다(참조the section called “3.6 또는 4.0에서 업그레이드된 클러스터에 대한 업그레이드 후 고려 사항”).
3. 드라이버를 업데이트합니다. 새 기능(예: 8.0의 데이터 정렬, 뷰 또는 Zstd 압축)을 사용하려면 해당 MongoDB API 버전으로 업그레이드합니다. 자세한 내용은 Amazon DocumentDB 8.0의 새로운 기능 섹션을 참조하세요.
4. 철저히 테스트합니다. 업그레이드된 클러스터에 대해 애플리케이션을 검증합니다.

Amazon DocumentDB 8.0으로 업그레이드한 후 변경되는 사항

Amazon DocumentDB 8.0(Amazon DocumentDB 5.0에서 또는 3.6 또는 4.0에서 직접)으로 메이저 버전 업그레이드를 수행한 후 다음 기능이 활성화되거나 변경됩니다.

- 데이터 정렬. Amazon DocumentDB 8.0은 데이터 정렬을 지원합니다. 업그레이드 후 새 컬렉션과 해당 인덱스, 기존 컬렉션의 새 인덱스에는 기본적으로 데이터 정렬이 활성화되어 있습니다.
- 텍스트 인덱스. 새 텍스트 인덱스는 MongoDB 호환성 향상을 위해 업데이트된 텍스트 검색 구문 분석기를 사용하는 텍스트 인덱스 V2를 사용하여 생성됩니다. 기존 텍스트 인덱스는 영향을 받지 않습니다.
- 쿼리 플래너 버전. 사용자 지정 파라미터 그룹이 없는 경우 플래너 버전 3이 자동으로 선택된 Amazon DocumentDB 8.0에 대한 새 기본 파라미터 그룹이 생성됩니다. 쿼리 플래너 버전 3에서는 뷰도 사용할 수 있습니다.
- 압축. Amazon DocumentDB 8.0은 Zstd 알고리즘을 사용한 사전 기반 문서 압축을 지원합니다. 업그레이드 후 기본적으로 Zstd 압축이 활성화된 새 컬렉션이 생성됩니다. 5.0의 기존 컬렉션은 압축 설정을 유지합니다. 기존 컬렉션에서 Zstd 압축을 활용하려면 압축 설정을 수정할 수 있습니다. 자세한 내용은 Amazon DocumentDB 8.0에서 사전 기반 압축 관리 섹션을 참조하세요.
- 인덱스 재구축. Amazon DocumentDB 5.0에서 Amazon DocumentDB 8.0으로 업그레이드하는 경우 인덱스 재구축이 필요하지 않습니다. Amazon DocumentDB 3.6 또는 4.0에서 8.0으로 직접 업그레

이드하는 경우에 설명된 대로 인덱스를 다시 빌드합니다the section called “3.6 또는 4.0에서 업그레이드된 클러스터에 대한 업그레이드 후 고려 사항”.

> **Important**
>
> Amazon DocumentDB 8.0에는 TLS 1.2 이상이 필요합니다. TLS 1.0 및 TLS 1.1은 더 이상 지원되지 않습니다.

> **Note**
>
> 기능 차이의 전체 목록은 섹션을 참조하세요Amazon DocumentDB의 MongoDB 호환성.

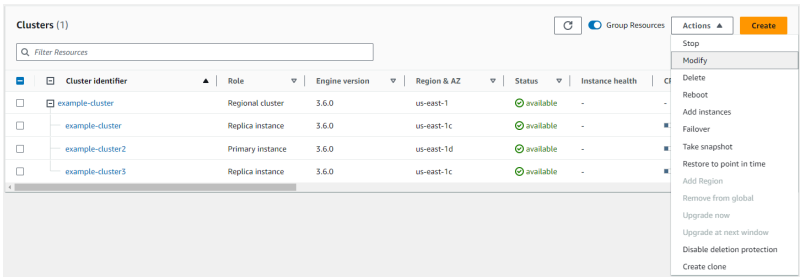
3.6 또는 4.0에서 업그레이드된 클러스터에 대한 업그레이드 후 고려 사항

- 인덱스 재구축. MVU는 원본 인덱스를 유지합니다. Amazon DocumentDB 5.0 이상에서는 특히 카디널리티가 낮은 인덱스의 경우 인덱스 유지 관리 및 가비지 수집이 개선되었습니다. 3.6 또는 4.0(5.0 또는 8.0)으로 업그레이드한 후 최적의 쿼리 성능을 보장하기 위해 인덱스를 다시 빌드합니다(선택 사항, 추가 I/O 포함). `reIndex`를 사용한 인덱스 유지 관리 섹션을 참조하세요.
- 하위 문서 숫자 비교(3.6만 해당). 3.6에서 업그레이드된 클러스터는 하위 문서의 숫자 유형이 유형 간에 비교되지 않는 3.6 동작을 상속합니다. 예를 들어 `{a: {b: NumberLong(1)}}`는 3.6`{a: {b: 1}}`에서는 같지 않지만 4.0 이상에서는 동일합니다. 이 동작은 3.6에서 업그레이드된 모든 클러스터에 영향을 줍니다.

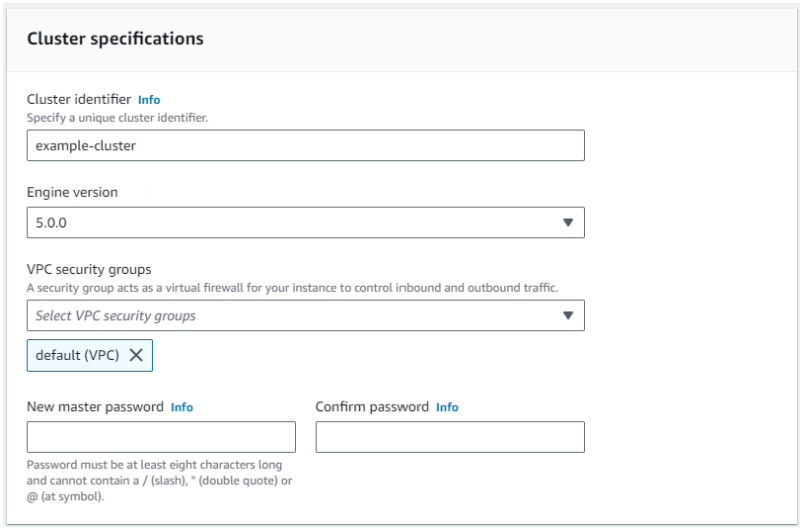
업그레이드 수행

Using the AWS Management Console

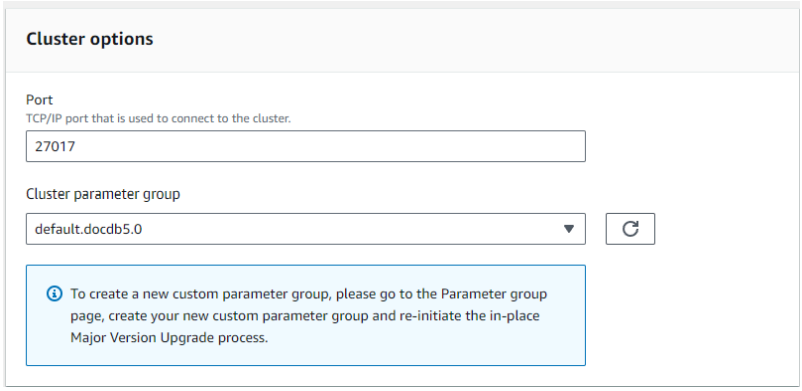
1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.
2. 클러스터 테이블에서 소스 클러스터를 선택하고 작업을 선택한 다음 수정을 선택합니다.



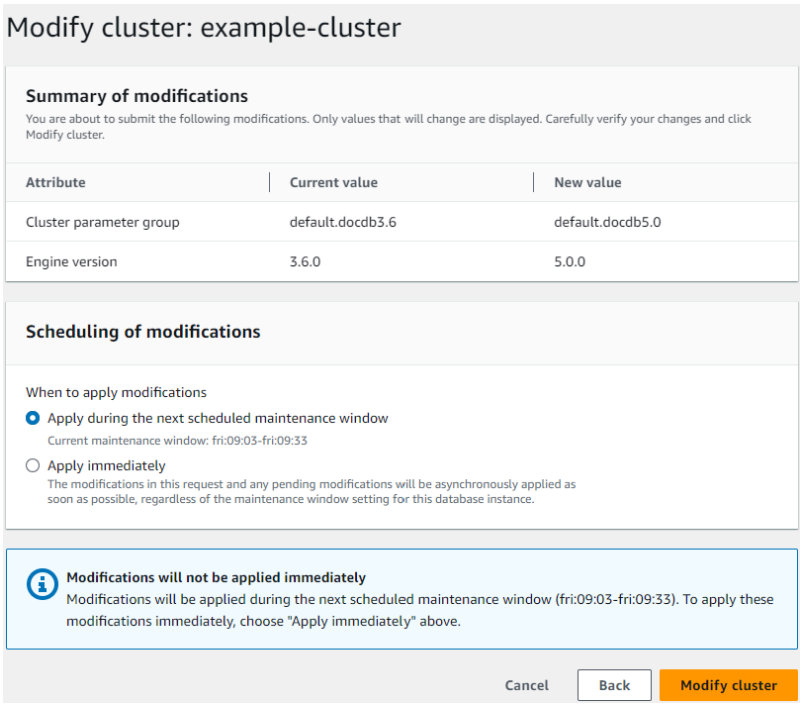
3. 클러스터 사양의 엔진 버전 드롭다운에서 대상 버전(예: 5.0.0 또는 8.0.0)을 선택합니다.



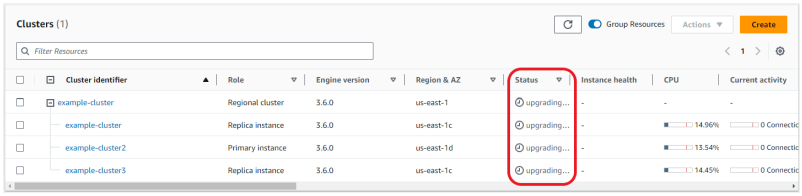
4. 클러스터 옵션에서 대상 엔진 버전의 클러스터 파라미터 그룹을 선택합니다. 기본값(예: default.docdb5.0 또는 default.docdb8.0) 또는 생성한 사용자 지정 파라미터 그룹을 사용할 수 있습니다.



5. 계속을 선택하고 예약 기본 설정(즉시 적용 또는 다음 유지 관리 기간)을 선택한 다음 클러스터 수정을 선택합니다.



6. 업그레이드로 변경될 때 클러스터 테이블의 클러스터 상태를 모니터링합니다.



Using the AWS CLI

`--allow-major-version-upgrade` 플래그와 [modify-db-cluster](#) 함께를 사용합니다.

```
aws docdb modify-db-cluster \
  --db-cluster-identifier mydocdbcluster \
  --allow-major-version-upgrade \
  --engine-version 8.0.0 \
  --apply-immediately \
  --cluster-parameter-group mydocdbparametergroup \
  --region us-east-1
```

각 *## ####* 클러스터 정보로 바꿉니다.

문제 해결

- 업그레이드 전 검사 실패. 업그레이드가 시작되기 전에 Amazon DocumentDB는 업그레이드 전 검증 검사를 실행합니다. 다음은 사전 확인 실패의 일반적인 원인입니다.
  - 컬렉션에 58자 이상의 이름이 있기 때문에 업그레이드를 진행할 수 없습니다. 업그레이드를 다시 시도하기 전에 영향을 받는 컬렉션 이름을 더 짧은 이름으로 바꿉니다.
  - 인덱스 수가 인스턴스 유형에 대한 제한을 초과하여 업그레이드를 진행할 수 없습니다. 업그레이드를 다시 시도하기 전에 더 큰 인스턴스 유형으로 업그레이드합니다. 인스턴스 유형별 인덱스 제한은 섹션을 참조하세요사전 조건.
- 업그레이드 실패 및 롤백. 업그레이드에 실패하면 자동으로 롤백을 시도합니다. 롤백에 성공하면 "데이터베이스 클러스터가 업그레이드할 수 없는 상태입니다."라는 이벤트가 생성됩니다. 클러스터가 업그레이드 전 상태로 돌아가고 계속 사용할 수 있습니다. 다시 시도하기 전에 문제를 해결하려면 AWS 지원팀에 문의하세요.
- 업그레이드 후 성능. 인덱스 메타데이터 새로 고침이 실행되는 동안 일시적인 성능 저하와 높은 CPU 사용률이 발생할 수 있습니다. 3시간 후에도 성능 저하가 지속되면 AWS 지원팀에 문의하십시오.

추가 지원은 AWS Support에 문의하세요.

Amazon DocumentDB 마이너 버전 업그레이드

> **Note**
>
> 이 주제에서는 동일한 메이저 버전 내의 마이너 버전 업그레이드를 다룹니다(예: 5.0.0에서 5.0.1로 업그레이드). 다른 메이저 버전으로 업그레이드하는 방법에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB 인플레이스 주요 버전 업그레이드.

클러스터를 수정하여 Amazon DocumentDB 클러스터의 마이너 엔진 버전(예: 5.0.0에서 5.0.1로)을 업그레이드할 수 있습니다. 마이너 버전 업그레이드는 즉시 또는 다음 유지 관리 기간에 적용할 수 있습니다.

마이너 버전 업그레이드를 수행하기 전

마이너 버전 업그레이드를 수행하기 전에 다음 작업이 유용할 수 있습니다.

- 트래픽이 적은 기간 동안 업그레이드를 수행하여 짧은 가동 중지 시간의 영향을 줄입니다.

- `describe-db-engine-versions` 명령을 사용하여 대상 마이너 버전이 현재 버전의 유효한 업그레이드 대상인지 확인합니다.
- 글로벌 클러스터의 경우 기본 클러스터보다 먼저 보조 클러스터를 업그레이드해야 합니다. 자세한 내용은 글로벌 클러스터 패치 적용 단원을 참조하십시오.

> **Important**
>
> - 업그레이드는 단방향 작업이므로 업그레이드 후에는 다운그레이드할 수 없습니다.
> - 업그레이드 프로세스 중에 클러스터가 잠시 가동 중지됩니다.

마이너 버전 업그레이드 수행

Using the AWS Management Console

를 사용하여 마이너 버전 업그레이드를 수행하려면 AWS Management Console:

1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.
2. 클러스터 테이블에서 업그레이드할 클러스터를 선택하고 작업을 선택한 다음 수정을 선택합니다.
3. 클러스터 수정 대화 상자의 클러스터 사양 섹션에서 엔진 버전 드롭다운 메뉴에서 대상 마이너 버전을 선택합니다.
4. 아래로 스크롤하여 계속을 선택합니다.
5. 수정 일정 섹션에서 변경 사항을 즉시 적용하거나 다음 유지 관리 기간에 적용하도록 선택합니다.

   그런 다음 클러스터 수정을 선택합니다.

Using the AWS CLI

`--engine-version` 파라미터와 함께 `modify-db-cluster` 명령을 사용하세요.

```
aws docdb modify-db-cluster \
    --db-cluster-identifier mydocdbcluster \
    --engine-version 5.0.1 \
    --apply-immediately \
```

```
    --region us-east-1
```

위의 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

> **Note**
>
> 즉시 대신 다음 유지 관리 기간에 업그레이드를 적용하려면 대신 `--no-apply-immediately`를 사용합니다`--apply-immediately`.

마이너 버전 업그레이드 확인

업그레이드가 완료되면 AWS CLI다음을 사용하여 엔진 버전을 확인합니다.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier mydocdbcluster \
    --query 'DBClusters[0].EngineVersion'
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
"5.0.1"
```

를 사용하여 Amazon DocumentDB 클러스터 업그레이드 AWS Database Migration Service

> **Important**
>
> Amazon DocumentDB는 MongoDB와 동일한 지원 수명 주기를 따르지 않으며 MongoDB의 수명 종료 일정은 Amazon DocumentDB에 적용되지 않습니다.

를 사용하여 가동 중지 시간을 최소화하면서 Amazon DocumentDB 클러스터를 상위 버전으로 업그레이드할 수 있습니다 AWS DMS. AWS DMS 는 이전 Amazon DocumentDB 버전, 관계형 데이터베이스 및 비관계형 데이터베이스에서 대상 Amazon DocumentDB 클러스터로 쉽게 마이그레이션할 수 있는 완전 관리형 서비스입니다.

주제

- 1단계: 변경 스트림 활성화

- 2단계: 변경 스트림 보존 기간 수정
- 3단계: 인덱스 마이그레이션
- 4단계: AWS DMS 복제 인스턴스 생성
- 5단계: AWS DMS 소스 엔드포인트 생성
- 6단계: AWS DMS 대상 엔드포인트 생성
- 7단계: 마이그레이션 작업 생성 및 실행
- 8단계: 애플리케이션 엔드포인트를 대상 Amazon DocumentDB 클러스터로 변경

1단계: 변경 스트림 활성화

가동 중지 시간 마이그레이션을 최소화하려면 클러스터의 변경 스트림에 대한 액세스 권한이 AWS DMS 필요합니다. Amazon DocumentDB 변경 스트림 기능은 클러스터의 컬렉션과 데이터베이스 내에서 시간순으로 발생하는 업데이트 이벤트 시퀀스를 제공합니다. 변경 스트림에서 읽으면가 변경 데이터 캡처(CDC) AWS DMS 를 수행하고 대상 Amazon DocumentDB 클러스터에 증분 업데이트를 적용할 수 있습니다.

특정 데이터베이스의 모든 컬렉션에 대해 변경 스트림을 활성화하려면 mongo 쉘을 사용하여 Amazon DocumentDB 클러스터를 인증하고 다음 명령을 실행합니다.

```
db.adminCommand({modifyChangeStreams: 1,
    database: "db_name",
    collection: "",
    enable: true});
```

2단계: 변경 스트림 보존 기간 수정

그런 다음 변경 스트림에서 변경 이벤트를 보존하려는 기간에 따라 변경 스트림 보존 기간을 수정합니다. 예를 들어를 사용한 Amazon DocumentDB 클러스터 마이그레이션 AWS DMS 에 12시간이 걸릴 것으로 예상되는 경우 변경 스트림 보존을 12시간보다 큰 값으로 설정해야 합니다. Amazon DocumentDB 클러스터의 기본 보존 기간은 3시간입니다. AWS Management Console 또는를 사용하여 Amazon DocumentDB 클러스터의 변경 스트림 로그 보존 기간을 1시간에서 7일로 수정할 수 있습니다 AWS CLI. 자세한 내용은 변경 스트림 로그 보존 기간 수정 단원을 참조하십시오.

3단계: 인덱스 마이그레이션

소스 Amazon DocumentDB 클러스터에 있는 것과 동일한 인덱스를 대상 Amazon DocumentDB 클러스터에 생성합니다. 는 데이터 마이그레이션을 AWS DMS 처리하지만 인덱스는 마이그레이션하

지 않습니다. 인덱스를 마이그레이션하려면 Amazon DocumentDB 인덱스 도구를 사용하여 소스 Amazon DocumentDB 클러스터에서 인덱스를 내보냅니다. Amazon DocumentDB 도구 GitHub 리포지토리의 복제본을 생성하고 README.md의 지침을 따르며 도구를 다운로드할 수 있습니다. Amazon DocumentDB 클러스터와 동일한 Amazon VPC에서 실행되는 환경 또는 Amazon EC2 인스턴스에서 도구를 실행할 수 있습니다. AWS Cloud9 Amazon DocumentDB

다음은 자신의 정보를 각각의 *### ## ## ###*로 변경하는 예제입니다.

다음 코드는 소스 Amazon DocumentDB 클러스터의 인덱스를 덤프합니다.

```
python migrationtools/documentdb_index_tool.py --dump-indexes
--uri mongodb://sample-user:user-password@sample-source-cluster.node.us-
east 1.docdb.amazonaws.com:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false'
--dir ~/index.js/

2020-02-11 21:51:23,245: Successfully authenticated to database: admin2020-02-11
 21:46:50,432: Successfully connected to instance docdb-40-xx.cluster-xxxxxxxx.us-
east-1.docdb.amazonaws.com:27017
2020-02-11 21:46:50,432: Retrieving indexes from server...2020-02-11 21:46:50,440:
 Completed writing index metadata to local folder: /home/ec2-user/index.js/
```

인덱스를 성공적으로 내보내기 하면 대상 Amazon DocumentDB 클러스터에서 해당 인덱스를 복원합니다. 이전 단계에서 내보낸 인덱스를 복원하려면 Amazon DocumentDB 인데스 도구를 사용하세요. 다음 명령은 지정된 디렉터리에서 대상 Amazon DocumentDB 클러스터의 인덱스를 복원합니다.

```
python migrationtools/documentdb_index_tool.py --restore-indexes
--uri mongodb://sample-user:user-password@sample-destination-
cluster.node.us-east 1.docdb.amazonaws.com:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false'
--dir ~/index.js/

2020-02-11 21:51:23,245: Successfully authenticated to database: admin2020-02-11
 21:51:23,245: Successfully connected to instance docdb-50-xx.cluster-xxxxxxxx.us-
east-1.docdb.amazonaws.com:27017
2020-02-11 21:51:23,264: testdb.coll: added index: _id
```

인덱스가 제대로 복원되었는지 확인하려면 mongo 쉘을 사용하여 대상 Amazon DocumentDB 클러스터에 연결하고 해당 컬렉션의 인덱스를 나열하세요. 다음 코드를 확인합니다.

```
mongo --ssl
```

```
--host docdb-xx-xx.cluster-xxxxxxxx.us-east-1.docdb.amazonaws.com:27017
--sslCAFile rds-ca-2019-root.pem --username documentdb --password documentdb

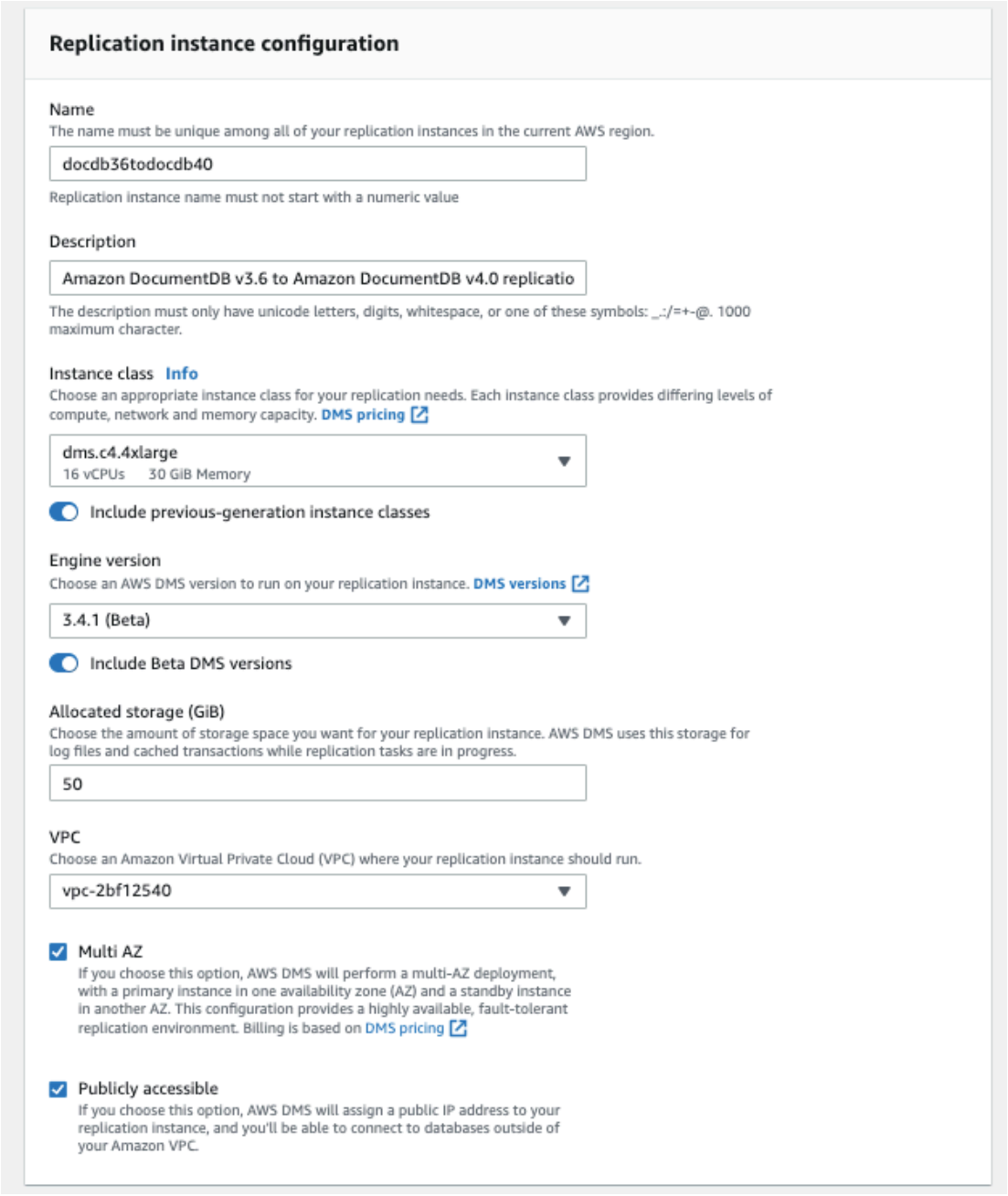
db.coll.getIndexes()
```

4단계: AWS DMS 복제 인스턴스 생성

AWS DMS 복제 인스턴스는 소스 Amazon DocumentDB 클러스터에서 데이터를 연결하고 읽은 다음 대상 Amazon DocumentDB 클러스터에 씁니다. AWS DMS 복제 인스턴스는 대량 로드 및 CDC 작업을 모두 수행할 수 있습니다. 이 절차 대다수는 메모리에서 진행됩니다. 그렇지만, 대규모 작업은 디스크에서 일부 버퍼링이 필요할 수 있습니다. 캐시된 트랜잭션과 로그 파일도 디스크에 기록됩니다. 데이터가 마이그레이션되면 복제 인스턴스는 소스와 대상이 동기화되어 있는지 확인하기 위해 변경 이벤트도 스트리밍합니다.

AWS DMS 복제 인스턴스를 생성하려면:

1. AWS DMS 콘솔을 엽니다.
2. 탐색 창에서 복제 인스턴스를 선택합니다.
3. 복제 인스턴스 생성을 선택하고 다음 정보를 입력합니다.
   - 이름에 원하는 이름을 입력합니다. 예를 들어 `docdb36todocdb40`입니다.
   - 설명에 선택에 대한 설명을 입력합니다. listitem의 경우 Amazon DocumentDB 3.6에서 Amazon DocumentDB 4.0 복제 인스턴스입니다.
   - 인스턴스 클래스에서 필요에 따라 크기를 선택합니다.
   - 엔진 버전에서 `3.4.1.` 항목을 선택합니다.
   - Amazon VPC에서 원본 및 대상 Amazon DocumentDB 클러스터를 포함하는 Amazon VPC를 선택합니다.
   - 할당된 스토리지(GiB)에서 기본값인 50GiB를 사용합니다. 쓰기 처리량 워크로드가 높은 경우 워크로드에 맞게 이 값을 높입니다.
   - 다중 AZ의 경우 고가용성 및 장애 조치 지원이 필요한 경우 예를 선택합니다.
   - 퍼블릭 액세스 가능에서 이 옵션을 활성화합니다.



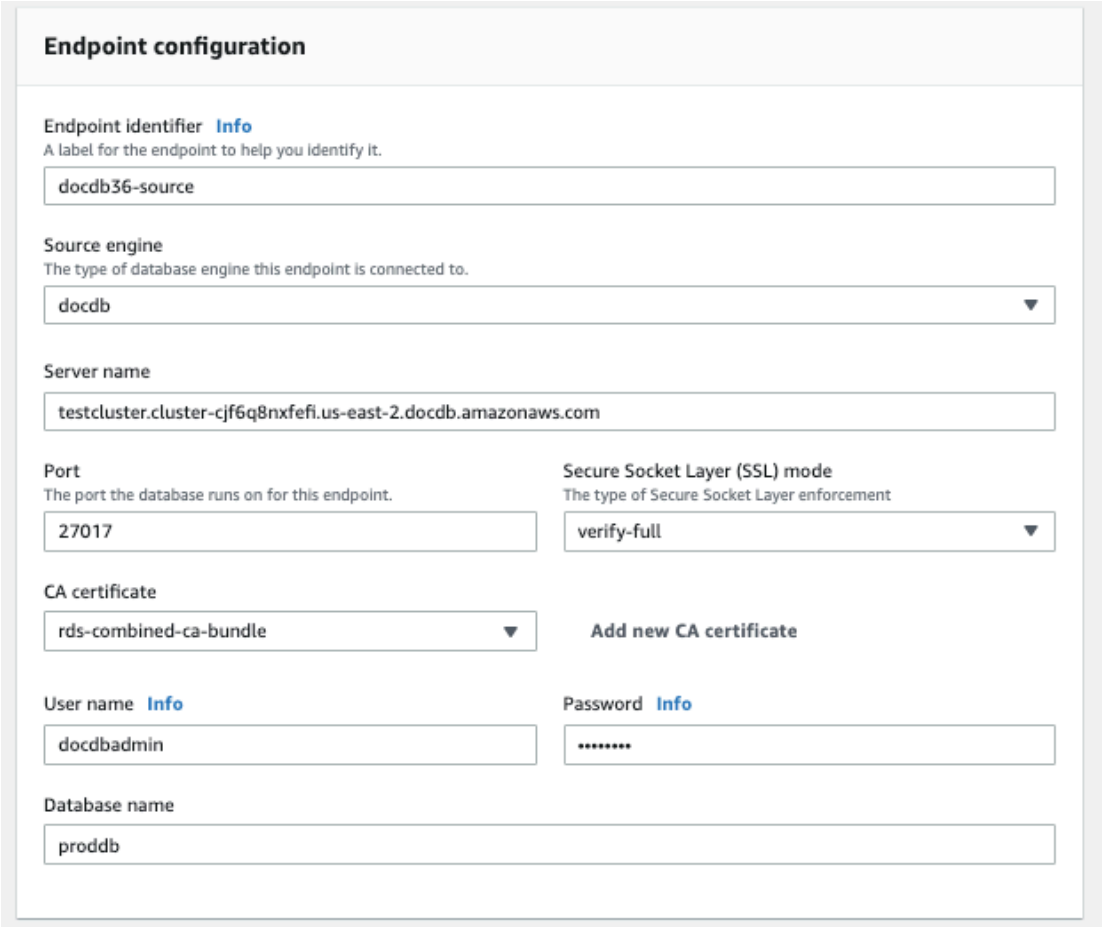
4. 복제 인스턴스 생성을 선택합니다.

5단계: AWS DMS 소스 엔드포인트 생성

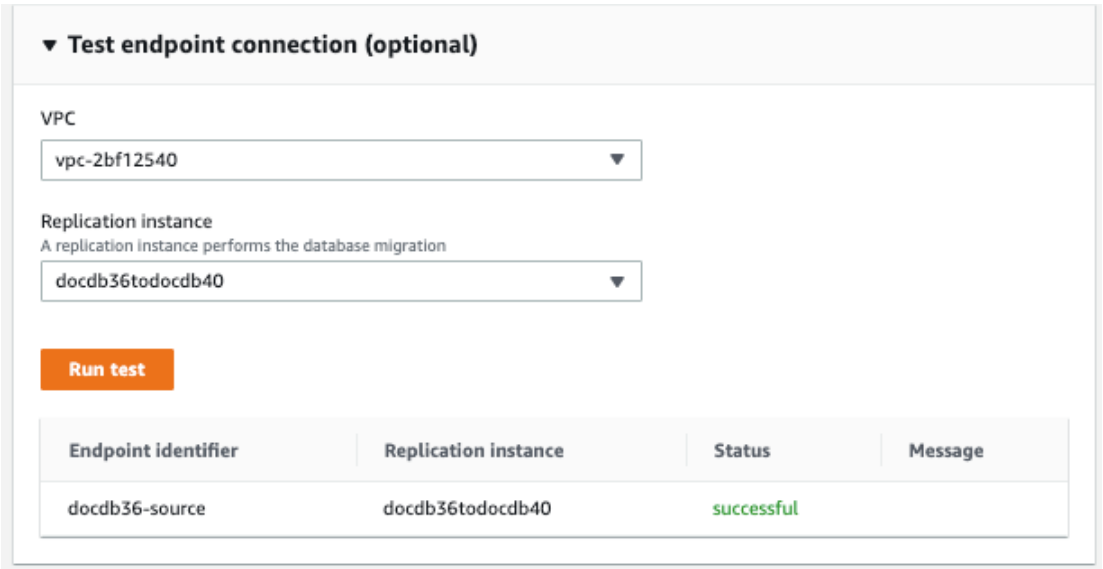
소스 엔드포인트는 소스 Amazon DocumentDB 클러스터에 사용됩니다.

소스 엔드포인트를 생성하려면

1. AWS DMS 콘솔을 엽니다.
2. 탐색 창에서 엔드포인트를 선택합니다.
3. `Create endpoint` 항목을 선택하고 다음 정보를 입력합니다.
   - 엔드포인트 유형에서 소스를 선택합니다.
   - >엔드포인트 식별자에 기억하기 쉬운 이름(예: `docdb-source`)을 입력합니다.
   - Source engine에서 `docdb` 항목을 선택합니다.
   - 서버 이름에 소스 Amazon DocumentDB 클러스터의 DNS 이름을 입력합니다.
   - 포트에 소스 Amazon DocumentDB 클러스터의 포트 번호를 입력합니다.
   - SSL 모드에서 `verify-full` 항목을 선택합니다.
   - CA 인증서의 경우 새 CA 인증서 추가를 선택합니다. 새 CA 인증서 를 다운로드하여 TLS 연결 번들을 생성합니다. 인증서 식별자에 `rds-combined-ca-bundle` 항목을 입력합니다. 인증서 파일 가져오기에서 파일 선택을 선택하고 이전에 다운로드한 `.pem` 파일로 이동합니다. 파일을 선택하고 엽니다. 인증서 가져오기를 선택한 다음 인증서 선택 드롭다운에서 `rds-combined-ca-bundle` 항목을 선택합니다.
   - 서버 이름에 원본 Amazon DocumentDB 클러스터의 기본 사용자 이름을 입력합니다.
   - 암호에 원본 Amazon DocumentDB 클러스터의 기본 사용자 암호를 입력합니다.
   - 데이터베이스 이름에 업그레이드하려는 데이터베이스 이름을 입력합니다.



4. 연결을 테스트하여 성공적으로 설정되었는지 확인합니다.



5. 엔드포인트 생성을 선택합니다.

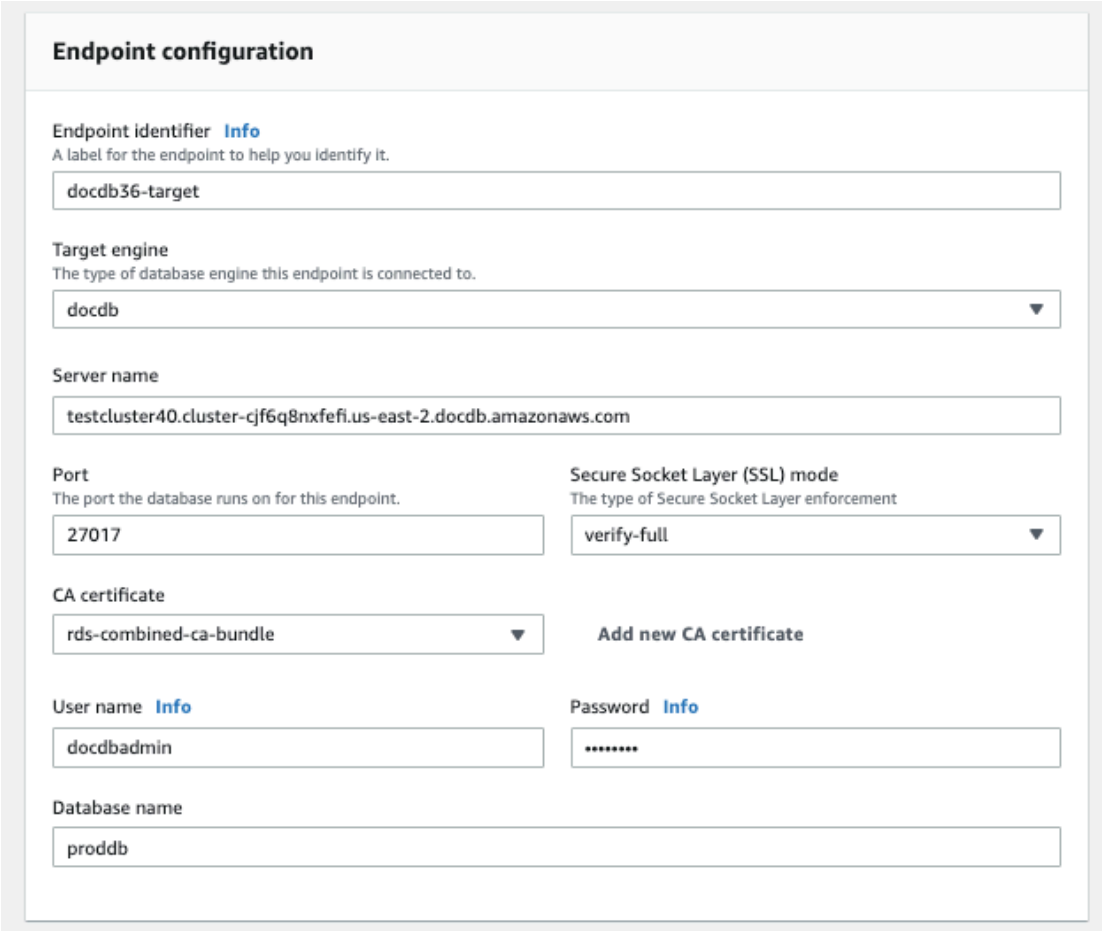
> **Note**
>
> AWS DMS 는 한 번에 하나의 데이터베이스만 마이그레이션할 수 있습니다.

6단계: AWS DMS 대상 엔드포인트 생성

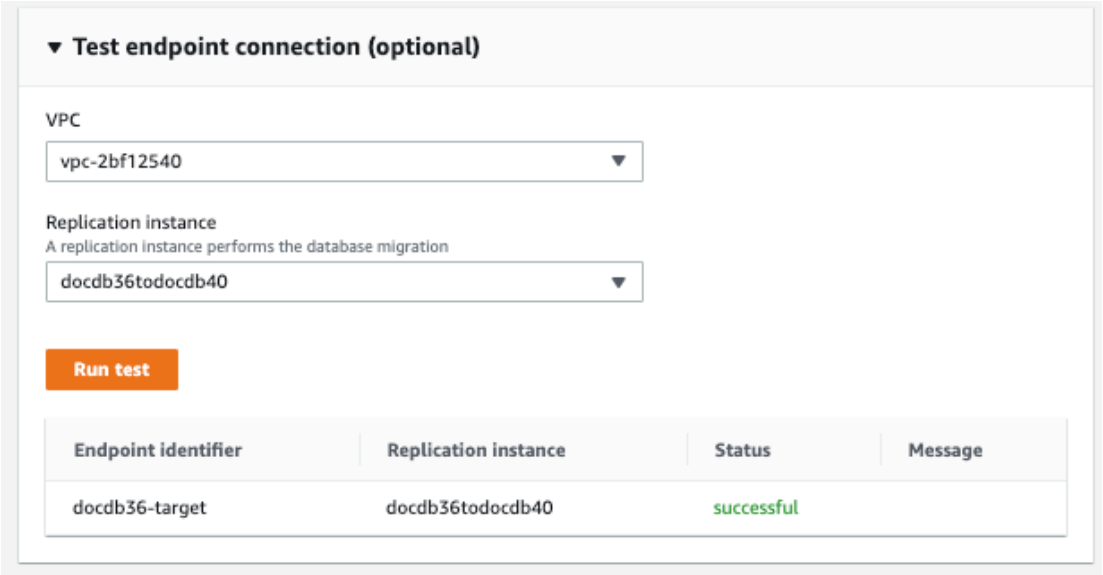
대상 엔드포인트는 대상 Amazon DocumentDB 클러스터에 사용됩니다.

대상 엔드포인트를 생성하려면:

1. AWS DMS 콘솔을 엽니다.
2. 탐색 창에서 엔드포인트를 선택합니다.
3. 엔드포인트 생성을 선택하고 다음 정보를 입력합니다.
   - 엔드포인트 유형에서 대상을 선택합니다.
   - 엔드포인트 식별자에 기억하기 쉬운 이름(예: `docdb-target`)을 입력합니다.
   - Source engine에서 `docdb` 항목을 선택합니다.
   - 서버 이름에 대상 Amazon DocumentDB 클러스터의 DNS 이름을 입력합니다.
   - 포트에 대상 Amazon DocumentDB 클러스터의 포트 번호를 입력합니다.
   - SSL 모드에서 `verify-full` 항목을 선택합니다.
   - CA 인증서의 경우, 인증서 선택 드롭다운에서 기존 `rds-combined-ca-bundle` 인증서를 선택합니다.
   - 사용자 이름에 대상 Amazon DocumentDB 클러스터의 기본 사용자 이름을 입력합니다.
   - 암호에 대상 Amazon DocumentDB 클러스터의 기본 암호를 입력합니다.
   - 데이터베이스 이름에 소스 엔드포인트를 설정하는 데 사용한 것과 동일한 데이터베이스 이름을 입력합니다.



4. 연결을 테스트하여 성공적으로 설정되었는지 확인합니다.



5. 엔드포인트 생성을 선택합니다.

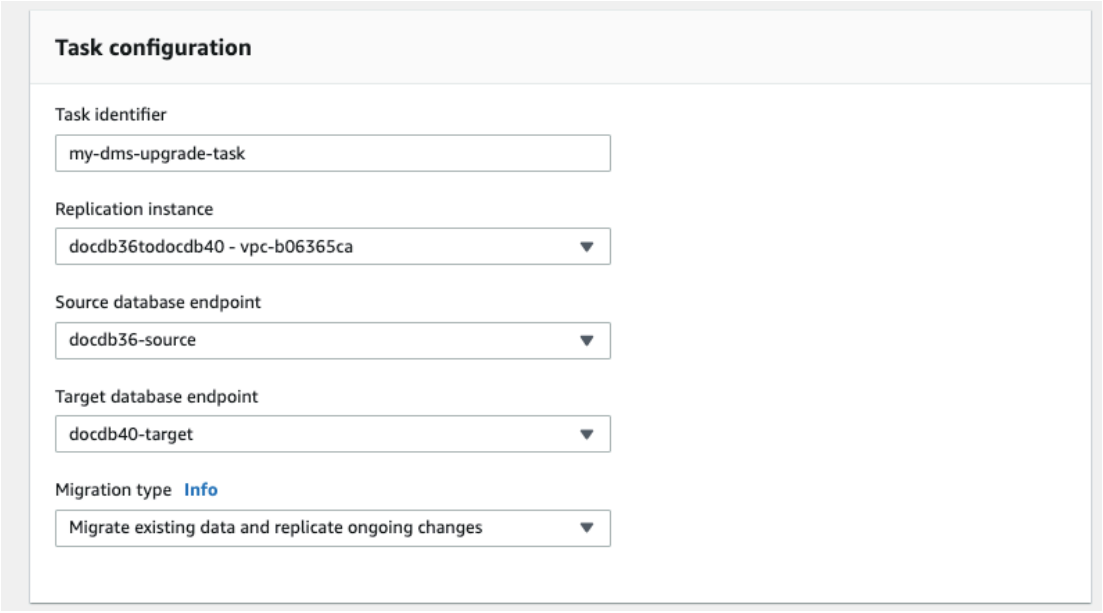
7단계: 마이그레이션 작업 생성 및 실행

AWS DMS 작업은 복제 인스턴스를 소스 및 대상 인스턴스와 바인딩합니다. 마이그레이션 작업을 생성할 때 소스 엔드포인트, 대상 엔드포인트, 복제 인스턴스 및 원하는 마이그레이션 설정을 지정합니다. AWS DMS 작업은 기존 데이터 마이그레이션, 기존 데이터 마이그레이션, 진행 중인 변경 사항 복제 또는 데이터 변경 사항만 복제라는 세 가지 마이그레이션 유형으로 생성할 수 있습니다. 이 연습의 목적은 가동 중지 시간을 최소화하면서 Amazon DocumentDB 클러스터를 업그레이드하는 것이므로 이 단계에서는 옵션을 사용하여 기존 데이터를 마이그레이션하고 지속적인 변경 사항을 복제합니다. 이 옵션을 사용하면 기존 데이터를 마이그레이션하는 동안 변경 사항을 AWS DMS 캡처합니다.는 대량 데이터가 로드된 후에도 변경 사항을 AWS DMS 계속 캡처하고 적용합니다. 결국 원본과 대상 데이터베이스는 동기화되어 가동 중지가 최소화된 마이그레이션이 가능합니다.

다음은 가동 중지 시간 최소화 마이그레이션을 위한 마이그레이션 작업을 생성하는 단계입니다.

1. AWS DMS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스 마이그레이션 작업을 선택합니다.
3. 데이터베이스 마이그레이션 작업 생성을 선택하고 작업 구성 섹션에 다음 정보를 입력합니다.

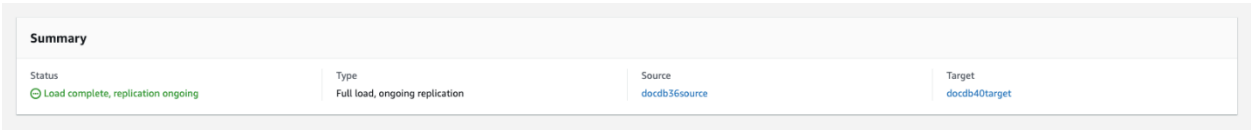
   - 태스크 식별자에 기억하기 쉬운 이름(예: `my-dms-upgrade-task`)을 입력합니다.
   - 설명이 포함된 Amazon 리소스 이름(ARN)에 사용자에게 친숙한 이름을 입력하여 기본 DMS ARN을 재정의합니다.
   - 복제 인스턴스에서 4단계: AWS DMS 복제 인스턴스 생성에서 생성한 복제 인스턴스를 선택합니다.
   - 소스 데이터베이스 엔드포인트에는 5단계: AWS DMS 소스 엔드포인트 생성에서 생성한 소스 엔드포인트를 선택합니다.
   - 대상 데이터베이스 엔드포인트에는 6단계: AWS DMS 대상 엔드포인트 생성에서 생성한 대상 엔드포인트를 선택합니다.
   - 마이그레이션 유형에는 마이그레이션 및 복제를 선택합니다.



4. 작업 설정 섹션에서 다음 정보를 입력합니다.

   - 대상 테이블 준비 모드에서 아무 작업 안 함을 선택합니다. 이렇게 하면 3단계에서 생성된 인덱스가 삭제되지 않습니다.
   - 작업 로그 하위 섹션에서 CloudWatch 로그 켜기를 선택합니다.
   - 마이그레이션 작업 스타트업 구성에는 생성 시 자동으로를 선택합니다. 이를 통해 마이그레이션 작업이 생성되면 자동으로 시작됩니다.
   - 데이터베이스 마이그레이션 작업 생성을 선택합니다.

AWS DMS 이제가 소스 Amazon DocumentDB 클러스터에서 대상 Amazon DocumentDB 클러스터로 데이터 마이그레이션을 시작합니다. 작업 상태가 시작 중에서 실행 중으로 바뀝니다. AWS DMS 콘솔에서 작업을 선택하여 진행 상황을 모니터링할 수 있습니다. 몇 분/몇 시간 후(마이그레이션 크기에 따라 다름) 상태가 로드 완료, 복제 진행 중으로 변경됩니다. 즉, AWS DMS 는 소스 Amazon DocumentDB 클러스터를 대상 Amazon DocumentDB 클러스터로 전체 로드 마이그레이션을 완료했으며 이제 변경 이벤트를 복제하고 있습니다.



결국 원본과 대상이 동기화됩니다. 컬렉션에서 `count()` 작업을 실행하여 모든 변경 이벤트가 마이그레이션되었는지 확인하여 동기화 여부를 확인할 수 있습니다.

8단계: 애플리케이션 엔드포인트를 대상 Amazon DocumentDB 클러스터로 변경

전체 로드가 완료되고 CDC 프로세스가 계속 복제되면 애플리케이션의 데이터베이스 연결 엔드포인트를 원본 Amazon DocumentDB 클러스터에서 대상 Amazon DocumentDB 클러스터로 변경할 준비가 된 것입니다.

LTS(장기 지원) 릴리스 사용

Amazon DocumentDB는 특정 버전을 장기 지원(LTS) 릴리스로 지정합니다. LTS를 사용하는 데이터베이스 클러스터는 비 LTS 릴리스를 사용하는 클러스터보다 동일한 버전을 더 오래 유지하고 업그레이드 주기를 줄일 수 있습니다. LTS 버전은 새로운 기능을 도입하지 않고 중요한 안정성 및 보안 패치가 있는 경우에만 패치됩니다. 중요한 장애를 디버깅하기 위한 지표와 같은 추가 선택적 패치가 릴리스될 수 있습니다. 이는 필수 패치와 함께 누적 적용됩니다.

현재 LTS 릴리스

Amazon DocumentDB의 현재 LTS 릴리스는 다음과 같습니다.

- 마이너 버전 5.0.0

클러스터가 켜져 있는 버전을 확인하려면 섹션을 참조하세요the section called “기존 엔진 버전 5.0.0 클러스터가 LTS에 있는지 확인하는 방법”.

LTS 버전의 지원 타임라인 및 릴리스 주기에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB 엔진 버전 지원 날짜. Amazon DocumentDB는 표준 통신 채널을 통해 다음 LTS 버전을 발표합니다.

> **Note**
>
> Amazon DocumentDB 8.0용 LTS 릴리스는 아직 지정되지 않았습니다. LTS가 필요한 경우 패치 버전 3.0.17983 이상에서 엔진 버전 5.0.0을 사용합니다.

LTS를 사용해야 하는 사용자

Amazon DocumentDB LTS 릴리스는 업그레이드 기간이 제한된 클러스터용으로 설계되었습니다. LTS는 데이터베이스 가용성이 비즈니스 운영에 직접적인 영향을 미치는 프로덕션 환경에 적합합니다.

다음과 같은 경우 LTS 릴리스를 고려합니다.

- 애플리케이션에 업그레이드 기회를 제한하는 엄격한 가동 시간 요구 사항이 있습니다.
- 애플리케이션에 필요한 모든 필수 데이터베이스 엔진 기능과 개선 사항이 있습니다.

다음과 같은 경우 표준 릴리스를 대신 고려하세요.

- 애플리케이션이 중단을 최소화할 수 있음
- 최신 기능 및 개선 사항이 릴리스되는 즉시 원하는 경우

> **Important**
>
> - 업그레이드는 단방향 작업입니다. 업그레이드 후에는 다운그레이드할 수 없습니다.
> - 업그레이드 프로세스 중에 클러스터에 짧은 가동 중지가 발생합니다.

새 LTS 클러스터를 생성하는 방법

LTS 클러스터를 생성하려면 다음 단계를 따릅니다.

1. Management Console에서 새 엔진 버전 5.0.0 클러스터를 생성합니다.
2. AWS CLI다음을 사용하여 클러스터 생성 후 엔진 버전 5.0.0을 사용하고 있는지 확인합니다.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier mydocdbcluster \
    --query 'DBClusters[0].EngineVersion'
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
"5.0.0"
```

엔진 버전 3.6.0 또는 4.0.0에서 5.0.0 LTS 클러스터로 업그레이드하는 방법

LTS 클러스터로 업그레이드하려면 다음 단계를 따릅니다.

1. Management Console에서 엔진 버전 3.6.0 또는 4.0.0 클러스터를 5.0.0으로 메이저 버전 업그레이드(MVU)합니다. MVU 수행에 대한 지침은 섹션을 참조하세요Amazon DocumentDB 인플레이스 주요 버전 업그레이드.
2. AWS CLI다음을 사용하여 업그레이드 후 엔진 버전 5.0.0을 사용하고 있는지 확인합니다.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier mydocdbcluster \
    --query 'DBClusters[0].EngineVersion'
```

기존 엔진 버전 5.0.0 클러스터가 LTS에 있는지 확인하는 방법

엔진 버전 5.0.0 클러스터가 LTS 버전을 실행 중인지 확인하려면:

1. AWS CLI다음을 사용하여 엔진 버전 5.0.0을 사용하고 있는지 확인합니다.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier mydocdbcluster \
    --query 'DBClusters[0].EngineVersion'
```

2. 필요한 경우 기존 엔진 버전 5.0.0 클러스터를 업그레이드합니다. 패치 업데이트 수행에 대한 지침은 섹션을 참조하세요클러스터의 엔진 버전에 대한 패치 업데이트 수행.

Amazon DocumentDB 추가 지원

Amazon DocumentDB 추가 지원을 사용하면 표준 지원 종료일이 지난 후 추가 비용을 지불하고 엔진 버전을 계속 실행할 수 있습니다. 표준 지원 종료일이 지나면 클러스터가 추가 지원에 등록됩니다. 엔진 버전 표준 지원 날짜가 종료될 때까지 업그레이드를 완료하지 않으면 클러스터에 추가 지원 요금이 청구됩니다.

주제

- 추가 지원 개요
- 추가 지원 요금
- 책임

Amazon DocumentDB 추가 지원은 다음과 같은 업데이트 및 기술 지원을 제공합니다.

- 인스턴스 또는 클러스터의 중요 및 높은 CVE에 대한 보안 업데이트

- 중요한 문제에 대한 버그 수정 및 패치
- 표준 Amazon DocumentDB 서비스 수준 계약 내에서 지원 사례를 열고 문제 해결 지원을 받을 수 있는 기능

이 유료 기능을 사용하면 지원되는 엔진 버전으로 업그레이드하는 데 더 많은 시간을 할애할 수 있습니다. 예를 들어 버전 3.6의 Amazon DocumentDB 표준 지원 종료일은 2026년 3월 30일이지만 해당 날짜 이전에 버전 5.0 또는 8.0으로 업그레이드할 준비가 되지 않았습니다. 이 경우 Amazon DocumentDB는 2026년 3월 31일에 추가 지원에 버전 3.6 클러스터를 등록하므로 Amazon DocumentDB 버전 3.6을 계속 사용할 수 있습니다. 2026년 7월 1일부터는 Amazon DocumentDB 버전 3.6 클러스터에 대한 추가 지원 프리미엄을 AWS 부과합니다.

Amazon DocumentDB 추가 지원은 엔진 버전 표준 지원 종료 날짜가 지난 후 최대 3년 동안 제공됩니다. 3년이 지난 후에도 지원되는 버전으로 엔진 버전을 업그레이드하지 않으면 Amazon RDS는 엔진 버전을 업그레이드합니다. 가능한 한 빨리 지원되는 엔진 버전으로 업그레이드합니다.

Amazon DocumentDB 버전 3.6 표준 지원 종료일 및 추가 지원 날짜:

- 버전 3.6 릴리스 - 2019년 1월 9일
- 버전 3.6 표준 지원 종료 - 2026년 3월 30일
- 버전 3.6 추가 지원 시작 1년차 요금 - 2026년 7월 1일
- 버전 3.6 추가 지원 시작 3년차 요금 - 2028년 3월 31일
- 버전 3.6 추가 지원 종료 - 2029년 3월 30일

추가 지원 개요

클러스터가 아직 해당 버전에 있지 않은 경우 Amazon DocumentDB는 클러스터를 마지막으로 릴리스된 엔진 버전(추가 지원 종료일 이전)으로 업그레이드합니다. 업그레이드 기간은 해당 엔진 버전의 표준 지원 날짜 이후까지 발생하지 않습니다.

해당 엔진 버전에 클러스터가 이미 있는 경우 표준 지원 종료일에 도달한 엔진 버전으로 새 클러스터를 생성할 수 있습니다. Amazon DocumentDB는 이러한 새 클러스터를 Amazon DocumentDB 추가 지원에 등록하고 이 제품에 대해 요금을 부과합니다. 버전 3.6의 표준 지원 종료일 이전에 아직 Amazon DocumentDB 표준 지원이 적용되는 엔진 버전으로 업그레이드하는 경우 추가 지원 프리미엄이 부과되지 않습니다.

Amazon DocumentDB 추가 지원 등록은 언제든지 종료할 수 있습니다. 등록을 종료하려면 등록된 각 클러스터를 아직 Amazon DocumentDB 표준 지원이 적용되는 최신 엔진 버전으로 업그레이드합니다.

RDS 추가 지원 등록 종료는 아직 Amazon DocumentDB 표준 지원이 적용되는 최신 엔진 버전으로 업그레이드를 완료하는 날부터 유효합니다.

Amazon DocumentDB 표준 지원 종료일 및 Amazon DocumentDB 추가 지원 종료일에 대한 자세한 내용은 Amazon DocumentDB 엔진 버전 지원 날짜 섹션을 참조하세요.

추가 지원 요금

2026년 7월 1일부터 추가 지원에서 실행되는 모든 클러스터에 대해 요금이 발생합니다. Amazon DocumentDB 버전 3.6 표준 지원 종료일은 Amazon DocumentDB 엔진 버전 지원 날짜 섹션을 참조하세요.

다음 작업 중 하나를 수행하는 경우 Amazon DocumentDB 추가 지원에 대한 추가 요금이 중지됩니다.

- 표준 지원이 적용되는 엔진 버전으로 업그레이드
- 표준 지원 종료일이 지난 후 버전 3.6을 실행 중인 클러스터 삭제

예를 들어 Amazon DocumentDB 버전 3.6은 2026년 3월 31일에 추가 지원을 시작하지만 요금은 2026년 7월 1일까지 시작되지 않습니다. 2026년 7월 31일에 Amazon DocumentDB 버전 3.6 클러스터를 Amazon DocumentDB 버전 5.0으로 업그레이드하는 경우 Amazon DocumentDB 버전 3.6에서 30일간의 추가 지원에 대해서만 요금이 부과됩니다. 자세한 내용은 Amazon DocumentDB 요금을 참조하세요.

책임

다음은 Amazon DocumentDB 책임과 Amazon DocumentDB 추가 지원에 대한 사용자의 책임입니다.

Amazon DocumentDB 책임

Amazon DocumentDB 버전 3.6 표준 지원 종료일 이후 Amazon DocumentDB는 추가 지원에 등록된 버전 3.6 클러스터에 패치, 버그 수정 및 업그레이드를 적용합니다.

사용자의 책임

또한 Amazon DocumentDB 버전 3.6의 추가 지원 종료일 이전에 엔진을 최신 엔진 버전으로 업그레이드하는 것은 사용자의 책임입니다. 클러스터를 업그레이드하지 않는 경우 Amazon DocumentDB 버전 3.6 추가 지원 종료일 이후 Amazon DocumentDB는 클러스터를 Amazon DocumentDB 표준 지원에 따라 지원되는 최신 엔진 버전으로 업그레이드하려고 시도합니다.

Amazon DocumentDB 기능 및 구성

다음 표에서는 Amazon DocumentDB 엔진 버전과 탄력적 클러스터의 기능, 기능 및 지원되는 인스턴스 유형을 비교합니다. 다음 표를 사용하여 워크로드 요구 사항에 가장 적합한 엔진 버전을 결정합니다.

엔진 상태

버전별 엔진 상태

기능/기능	v3.6	v4.0	v5.0	v8.0	
지원 상태	EOL: 2026년 3월	활성	활성(LTS 사용 가능)	활성(GA 2025년 11월)	
추가 지원 사용 가능	예(유료, 2029년 3월까지)	해당 사항 없음	해당 사항 없음	해당 사항 없음	
LTS 릴리스 사용 가능	아니요	아니요	예(v5.0 LTS)	아니요	
엘라스틱 클러스터	아니요	아니요	예	아니요	

클러스터 유형

버전별 클러스터 유형

클러스터 유형(Cluster Type)	v3.6	v4.0	v5.0	v8.0	엘라스틱 클러스터
인스턴스 기반 클러스터	예	예	예	예	예
서버리스 인스턴스	아니요	아니요	예	예	아니요

특성

엔진 버전별 기능 및 구성

기능	v3.6	v4.0	v5.0	v8.0	엘라스틱 클러스터
글로벌 클러스터	아니요	예	예	예	아니요
다중 AZ(AZ AZs)	예	예	예	예	예(샤드당)
읽기 전용 복제본 지원	15	15	15	15	샤드 당 15개
ACID 트랜잭션	아니요	예	예	예	아니요
재시도 가능한 쓰기	아니요	아니요	아니요	예(8.0.2 이상)	아니요
Change streams	예	예	예	예	아니요
벡터 검색	아니요	아니요	예	예	아니요
성능 개선 도우미	예	예	예	예	아니요
인플레이스 메이저 버전 업그레이드(MVU)	예(5.0 또는 8.0으로)	예(5.0 또는 8.0으로)	예(8.0으로)	아니요(대상만 해당)	아니요
클라이언트 측 필드 레벨 암호화(FLE)	아니요	아니요	예	예	아니요
I/O 최적화 스토리지	아니요	아니요	예	예	아니요
TLS 인증서 교체(재부팅 없음)	예(패치 1.0.208662 이상)	예(패치 2.0.10179 이상)	예(패치 3.0.4780 이상)	예	예
리전 간 스냅샷 복제	예	예	예	예	예
교차 계정 스냅샷 공유	예	예	예	예	아니요

기능	v3.6	v4.0	v5.0	v8.0	엘라스틱 클러스터
감사 및 느린 쿼리 로그	예	예	예	예	예
듀얼 스택 IPv4/IPv6	아니요	예(패치 2.0.11747 이상)	예(패치 3.0.15902 이상)	예	해당 사항 없음
콜레이션	아니요	아니요	아니요	예	아니요
뷰	아니요	아니요	아니요	예	아니요
압축 지원	아니요	아니요	LZ4	LZ4/Ztsd	아니요
텍스트 인덱스	아니요	아니요	V1	V2	아니요
FedRAMP(GovCloud)	아니요	아니요	예	예	아니요

인스턴스 유형

버전별 인스턴스 유형 지원

인스턴스 유형	v3.6	v4.0	v5.0	v8.0	
db.r4(이전 세대)	예	아니요	아니요	아니요	
db.r5	예	예	예	예	
db.r6g(Graviton2)	아니요	예	예	예	
db.r6gd(NVMe 지원 Graviton2)	아니요	아니요	예	예	
db.r8g(Graviton4)	아니요	아니요	예	예	
db.t3.medium	예	예	예	예	
db.t4g.medium(Graviton2)	아니요	예	예	예	

인스턴스 유형	v3.6	v4.0	v5.0	v8.0	
서버리스	아니요	아니요	예	예	

AWS 통합

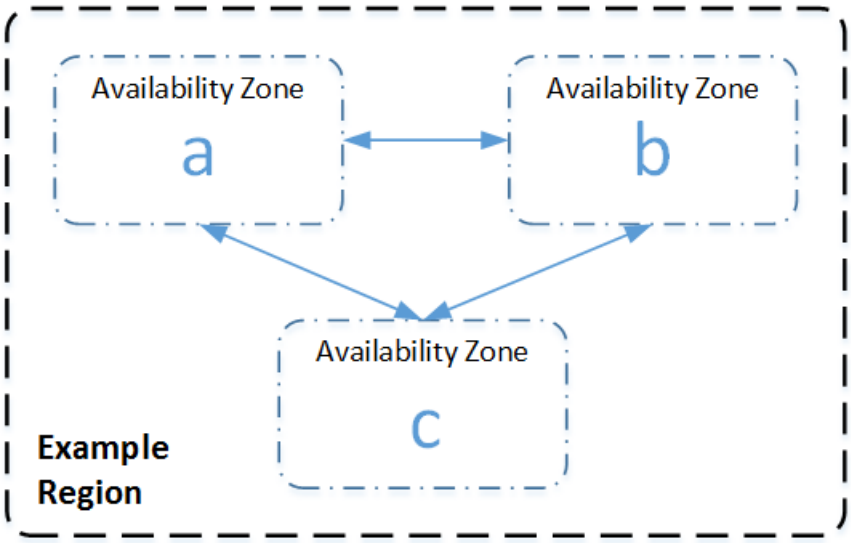
AWS 버전별 통합 지원

AWS 통합/기능	v3.6	v4.0	v5.0	v8.0	엘라스틱 클러스터
AWS Backup	예	예	예	예	아니요
AWS Secrets Manager	예	예	예	예	아니요
AWS Lambda (네이티브 ESM)	아니요	예	예	아니요	아니요
Amazon OpenSearch Service Ingestion(제로 ETL)	아니요	예	예	예	아니요
AWS DMS 마이그레이션 지원	예	예	예	예(5.0부터 경유 AWS DMS)	예
AWS Identity and Access Management authentication	아니요	아니요	예(5.0 인스턴스 기반)	예	아니요
AWS CloudShell	아니요	아니요	예	예	아니요
AWS Key Management Service	예	예	예	예	예
AWS CloudTrail	예	예	예	예	예

Choosing Regions and Availability Zones

Amazon 클라우드 컴퓨팅 리소스는 전 세계 여러 위치에서 호스팅됩니다. 이러한 위치는 AWS 리전 및 가용 영역으로 구성됩니다. 각 AWS 리전은 개별 지리 영역입니다. 각 리전은 가용 영역이라고 알려진 격리된 위치를 여러 개 가지고 있습니다. Amazon DocumentDB를 사용하면 인스턴스와 같은 리소스와 데이터를 여러 위치에 배치할 수 있습니다. 특별히 복제하지 않는 AWS 리전 한 리소스는 간에 복제되지 않습니다.

Amazon은 고급의 고가용성 데이터 센터를 운영하고 있습니다. 드물기는 하지만 동일한 위치에 있는 인스턴스의 가용성에 영향을 미치는 장애가 발생할 수도 있습니다. 그런 장애의 영향을 받는 단일한 위치에서 모든 인스턴스를 호스팅하는 경우에는 모든 인스턴스가 사용이 불가능해질 수 있습니다. 다음 다이어그램은 세 개의 가용 영역이 AWS 리전 있는를 보여줍니다.



각 리전은 서로 독립적이라는 점에 유의하십시오. 사용자의 모든 Amazon DocumentDB 활동(인스턴스 또는 사용할 수 있는 인스턴스 목록 생성 등)은 현재 기본 AWS 리전리전에서만 실행됩니다. `EC2_REGION` 환경 변수를 설정하여 콘솔의 기본 리전을 변경할 수 있습니다. 또는 AWS CLI에서 `--region` 파라미터를 사용하여 재정의할 수 있습니다. 자세한 내용은 구성 AWS Command Line Interface, 특히 환경 변수 및 명령줄 옵션에 대한 섹션을 참조하세요.

Amazon DocumentDB 콘솔을 사용하여 클러스터를 생성하고 다른 가용 영역에서 복제본을 생성하도록 선택할 경우 Amazon DocumentDB는 두 개의 인스턴스를 생성합니다. 가용 영역 하나에서 기본 인스턴스를 생성하고 다른 가용 영역에서 복제본 인스턴스를 생성합니다. 클러스터 볼륨은 항상 세 개의 가용 영역에 걸쳐 복제됩니다.

특정에서 Amazon DocumentDB 인스턴스를 생성하거나 사용하려면 해당 리전 서비스 엔드포인트를 AWS 리전사용합니다.

리전 가용성

Amazon DocumentDB는 다음 AWS 리전에서 사용할 수 있습니다.

Amazon DocumentDB에서 지원하는 지역

리전 이름	리전	가용 영역(컴퓨팅)
미국 동부(오하이오)	us-east-2	3
미국 동부(버지니아 북부)	us-east-1	6
미국 서부(오리건)	us-west-2	4
아프리카(케이프타운)	af-south-1	3
남아메리카(상파울루)	sa-east-1	3
아시아 태평양(홍콩)	ap-east-1	3
아시아 태평양(하이데라바드)	ap-south-2	3
아시아 태평양(말레이시아)	ap-southeast-5	3
아시아 태평양(뭄바이)	ap-south-1	3
아시아 태평양(오사카)	ap-northeast-3	3
아시아 태평양(서울)	ap-northeast-2	4
아시아 태평양(싱가포르)	ap-southeast-1	3
아시아 태평양(시드니)	ap-southeast-2	3
아시아 태평양(자카르타)	ap-southeast-3	3

리전 이름	리전	가용 영역(컴퓨팅)
아시아 태평양(멜버른)	ap-southeast-4	3
아시아 태평양(태국)	ap-southeast-7	3
아시아 태평양(도쿄)	ap-northeast-1	3
캐나다(중부)	ca-central-1	3
캐나다 서부(캘거리)	ca-west-1	3
중국(베이징) 리전	cn-north-1	3
중국(닝샤)	cn-northwest-1	3
유럽(프랑크푸르트)	eu-central-1	3
유럽(취리히)	eu-central-2	3
유럽(아일랜드)	eu-west-1	3
유럽(런던)	eu-west-2	3
유럽(밀라노)	eu-south-1	3
유럽(파리)	eu-west-3	3
유럽(스페인)	eu-south-2	3
유럽(스톡홀름)	eu-north-1	3
멕시코(중부)	mx-central-1	3
중동(UAE)	me-central-1	3
이스라엘(텔아비브)	il-central-1	3
AWS GovCloud(미국 서부)	us-gov-west-1	3

리전 이름	리전	가용 영역(컴퓨팅)
AWS GovCloud(미국 동부)	us-gov-east-1	3

기본적으로 Amazon DocumentDB 클러스터의 시간대는 협정 세계시(UTC)입니다.

특정 리전의 클러스터 및 인스턴스에 대한 연결 엔드포인트를 찾는 방법에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB 엔드포인트에 대한 이해.

Amazon OpenSearch Service와 제로 ETL 통합

주제

- Amazon OpenSearch Service의 대상입니다.
- 제한 사항

Amazon OpenSearch Service의 대상입니다.

Amazon DocumentDB와의 OpenSearch Service 통합을 통해 전체 로드를 스트리밍하고 데이터 이벤트를 OpenSearch 도메인으로 변경할 수 있습니다. 수집 인프라는 OpenSearch 수집 파이프라인으로 호스팅되며 Amazon DocumentDB 컬렉션에서 데이터를 지속적으로 스트리밍할 수 있는 대규모의 짧은 지연 시간 메커니즘을 제공합니다.

전체 로드 중에 제로 ETL 통합은 먼저 수집 파이프라인을 사용하여 과거 전체 로드 데이터를 OpenSearch로 추출합니다. 전체 로드 데이터가 수집되면 OpenSearch 수집 파이프라인은 Amazon DocumentDB 변경 스트림에서 데이터를 읽기 시작하고 결국 Amazon DocumentDB와 OpenSearch 간의 실시간에 가까운 데이터 일관성을 유지하는 것에 가까워집니다. OpenSearch는 문서를 인덱스에 저장합니다. Amazon DocumentDB 컬렉션에서 수신되는 데이터는 하나의 인덱스로 전송하거나 다른 인덱스로 분할할 수 있습니다. 수집 파이프라인은 Amazon DocumentDB 컬렉션의 모든 생성, 업데이트 및 삭제 이벤트를 OpenSearch 문서의 해당 생성, 업데이트 및 삭제로 동기화하여 두 데이터 시스템을 동기화된 상태로 유지합니다. 수집 파이프라인은 한 가지 컬렉션에서 데이터를 읽고 한 가지 인덱스에 쓰거나 한 가지 컬렉션에서 데이터를 읽고 여러 인덱스로 조건부로 라우팅하도록 구성할 수 있습니다.

수집 파이프라인은 다음을 사용하여 Amazon DocumentDB에서 Amazon OpenSearch Service로 데이터를 스트리밍하도록 구성할 수 있습니다.

- 전체 로드만

- 전체 로드 없이 Amazon DocumentDB에서 스트림 변경 이벤트 스트리밍
- 전체 로드 후 Amazon DocumentDB에서 스트림 변경

수집 파이프라인을 설정하려면 다음 단계를 수행합니다.

1단계: Amazon OpenSearch Service 도메인 또는 OpenSearch 서버리스 컬렉션 생성

데이터를 읽을 수 있는 적절한 권한이 있는 Amazon OpenSearch Service 컬렉션이 필요합니다. 모음을 생성하려면 Amazon OpenSearch Service 개발자 가이드의 Amazon OpenSearch Service 시작하기 또는 Amazon OpenSearch Serverless 시작하기를 참조하세요. Amazon OpenSearch Service 개발자 안내서의 Amazon OpenSearch Ingestion을 참조하여 컬렉션 또는 도메인에 쓰기 데이터에 액세스할 수 있는 올바른 권한이 있는 AIM 역할을 생성합니다.

2단계: Amazon DocumentDB 클러스터에서 변경 스트림 활성화

Amazon DocumentDB 클러스터의 필수 컬렉션에서 변경 스트림이 활성화되어 있는지 확인합니다. 자세한 내용은 Amazon DocumentDB에서 변경 스트림 사용 섹션을 참조하세요.

3단계: Amazon S3 버킷 및 대상 도메인 또는 컬렉션에 쓸 수 있는 권한을 가진 파이프라인 역할 설정

Amazon DocumentDB 컬렉션을 생성하고 변경 스트림을 활성화한 후 파이프라인 구성에 사용할 파이프라인 역할을 설정하고 역할에 다음 권한을 추가합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "allowReadAndWriteToS3ForExport",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket/export/*"
```

```
            ]
        }
    ]
}
```

OpenSearch 파이프라인이 OpenSearch 도메인에 데이터를 쓰려면 도메인에 sts_role_arn 파이프라인 역할이 도메인에 액세스할 수 있도록 허용하는 도메인 수준 액세스 정책이 있어야 합니다. 다음 샘플 도메인 액세스 정책은 이전 단계에서 생성한 `pipeline-role`이라는 파이프라인 역할을 사용하여 `ingestion-domain`이라는 도메인에 데이터를 쓸 수 있도록 허용합니다.

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::{your-account-id}:role/{pipeline-role}"
      },
      "Action": ["es:DescribeDomain", "es:ESHttp*"],
      "Resource": "arn:aws:es:{region}:{your-account-id}:domain/{domain-name}/*"
    }
  ]
}
```

4단계: 파이프라인 역할에 필요한 권한을 추가하여 X-ENI 생성

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AttachNetworkInterface",
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DetachNetworkInterface",
                "ec2:DescribeNetworkInterfaces"
```

```
            ],
            "Resource": [
                "arn:aws:ec2:*:420497401461:network-interface/*",
                "arn:aws:ec2:*:420497401461:subnet/*",
                "arn:aws:ec2:*:420497401461:security-group/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:Describe*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [ "ec2:CreateTags" ],
            "Resource": "arn:aws:ec2:*:*:network-interface/*",
            "Condition": {
               "StringEquals": { "aws:RequestTag/OSISManaged": "true" }
            }
        }
    ]
}
```

5단계: 파이프라인 생성

Amazon DocumentDB를 원본으로 지정하는 OpenSearch 수집 파이프라인을 구성합니다. 이 샘플 파이프라인 구성은 변경 스트림 가져오기 메커니즘을 사용하는 것으로 가정합니다. 자세한 내용은 Amazon OpenSearch Service 개발자 안내서의 Amazon DocumentDB에서 OpenSearch 수집 파이프라인 사용을 참조하세요.

제한 사항

Amazon DocumentDB OpenSearch 통합에는 다음 제한 사항이 적용됩니다.

- 파이프라인당 하나의 Amazon DocumentDB 컬렉션만 원본으로 지원됩니다.

- 교차 리전 데이터 수집은 지원되지 않습니다. Amazon DocumentDB 클러스터와 OpenSearch 도메인은 동일한에 있어야 합니다 AWS 리전.
- 교차 계정 데이터 수집은 지원되지 않습니다. Amazon DocumentDB 클러스터와 OpenSearch 수집 파이프라인은 동일한 AWS 계정에 있어야 합니다.
- Amazon DocumentDB 탄력적 클러스터는 지원되지 않습니다. Amazon DocumentDB 인스턴스 기반 클러스터만 지원됩니다.
- Amazon DocumentDB 클러스터에 AWS secrets. AWS secrets를 사용하여 인증이 활성화되어 있는지 확인합니다.
- 기존 파이프라인 구성은 다른 데이터베이스 및/또는 다른 컬렉션에서 데이터를 수집하도록 업데이트할 수 없습니다. 파이프라인의 데이터베이스 및/또는 컬렉션 이름을 업데이트하려면 새 파이프라인을 생성해야 합니다.

Amazon DocumentDB로 개발

이 섹션에서는 Amazon DocumentDB 클러스터에 연결, 데이터 쿼리 및 관리, 인덱스 작업, 생성형 AI 애플리케이션에 대한 변경 스트림, 트랜잭션 및 벡터 검색과 같은 기능 사용에 대해 다룹니다.

주제

- Amazon DocumentDB에 연결
- Amazon DocumentDB를 사용한 프로그램
- Amazon DocumentDB에서 쿼리
- 인덱스
- Amazon DocumentDB에서 변경 스트림 사용
- Amazon DocumentDB의 트랜잭션
- Amazon DocumentDB에서 재시도 가능한 쓰기
- Amazon DocumentDB 8.0에서 데이터 정렬 사용
- Amazon DocumentDB 8.0에서 뷰 사용
- Amazon DocumentDB 생성형 인공 지능
- 변경 스트림과 AWS Lambda 함께 사용

Amazon DocumentDB에 연결

Amazon DocumentDB는 MongoDB 드라이버를 사용한 프로그래밍 방식 연결, SSH 터널링 또는 AWS VPN을 통해 VPC 외부에서 연결, Studio 3T 및 DataGrip과 같은 인기 있는 도구를 비롯한 여러 연결 옵션을 제공합니다. 이 서비스는 고가용성을 위해 복제본 세트 연결을 지원합니다. 분석 및 보고를 위해 JDBC 또는 ODBC 드라이버를 사용하여 연결할 수 있습니다. 동일한 VPC의 EC2 인스턴스는 DocumentDB 클러스터에 직접 연결할 수 있습니다. 모든 연결에는 TLS와 적절한 AWS 자격 증명/인증이 필요합니다.

주제

- Amazon DocumentDB에 프로그래밍 방식으로 연결
- Amazon VPC 외부에서 Amazon DocumentDB 클러스터에 연결
- Amazon DocumentDB에 복제본 세트로 연결
- Studio 3T에서 Amazon DocumentDB 클러스터에 연결
- DataGrip을 사용하여 Amazon DocumentDB에 연결

- Amazon EC2를 사용하여 연결
- Amazon DocumentDB JDBC 드라이버를 사용하여 연결합니다.
- Amazon DocumentDB ODBC 드라이버를 사용하여 연결

Amazon DocumentDB에 프로그래밍 방식으로 연결

이 섹션에는 다양한 언어를 사용하여 Amazon DocumentDB(MongoDB 호환)에 연결하는 방법을 보여주는 코드 예제가 포함되어 있습니다. 이들 예제는 전송 계층 보안(TLS)이 활성화 또는 비활성화된 클러스터에 연결하는지에 따라 두 가지로 구분됩니다. 기본적으로 Amazon DocumentDB 클러스터에서는 TLS가 활성화됩니다. 하지만 원할 경우 TLS를 비활성화할 수 있습니다. 자세한 내용은 전송 중 데이터 암호화 섹션을 참조하세요.

클러스터가 있는 VPC 외부에서 Amazon DocumentDB에 연결하려는 경우 섹션을 참조하세요Amazon VPC 외부에서 Amazon DocumentDB 클러스터에 연결.

클러스터에 연결하기 전에 클러스터에서 TLS가 활성화되어 있는지 여부를 알아야 합니다. 다음 섹션에서는 AWS Management Console 또는 AWS CLI를 사용하여 클러스터의 `tls` 파라미터 값을 확인하는 방법을 설명합니다. 그런 다음 적절한 코드 예제를 찾아 적용할 수 있습니다.

주제

- tls 파라미터 값 확인
- TLS가 활성화된 상태에서 연결
- TLS가 비활성화된 상태에서 연결
- Amazon DocumentDB의 네트워크 압축

`tls` 파라미터 값 확인

클러스터에 TLS가 활성화되어 있는지 확인하는 것은 AWS Management Console 또는를 사용하여 수행할 수 있는 2단계 프로세스입니다 AWS CLI.

1. 어느 파라미터 그룹이 클러스터를 관리하는지 결정합니다.

   Using the AWS Management Console

   1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
   2. 좌측 탐색 창에서 클러스터를 선택합니다.

3. 클러스터 목록에서 클러스터 이름을 선택합니다.
4. 결과 페이지에는 선택한 클러스터의 세부 정보가 표시됩니다. [구성(Configuration)] 탭을 선택합니다. 구성 및 상태 섹션의 클러스터 파라미터 그룹 아래에서 파라미터 그룹의 이름을 찾습니다.

Using the AWS CLI

다음 AWS CLI 코드는 클러스터를 관리하는 파라미터를 결정합니다. *sample-cluster*를 클러스터의 이름으로 바꿔야 합니다.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterIdentifier,DBClusterParameterGroup]'
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
[
       [
           "sample-cluster",
           "sample-parameter-group"
       ]
]
```

2. 클러스터의 파라미터 그룹에서 **tls** 파라미터의 값을 확인합니다.

Using the AWS Management Console

1. 탐색 창에서 파라미터 그룹을 선택합니다.
2. 클러스터 파라미터 그룹 창에서 1단계의 클러스터 파라미터 그룹 이름을 선택합니다.
3. 결과 페이지에는 클러스터 파라미터 그룹의 파라미터가 표시됩니다. 여기서 `tls` 파라미터의 값을 볼 수 있습니다. 이 파라미터 수정에 대한 자세한 내용은 Amazon DocumentDB 클러스터 파라미터 그룹 수정 섹션을 참조하세요.

Using the AWS CLI

`describe-db-cluster-parameters` AWS CLI 명령을 사용하여 클러스터 파라미터 그룹의 파라미터 세부 정보를 볼 수 있습니다.

- **--describe-db-cluster-parameters** - 파라미터 그룹 내의 모든 파라미터와 해당 값을 나열합니다.
  - **--db-cluster-parameter-group name** - 필수입니다. 클러스터 파라미터 그룹의 이름입니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
        "Parameters": [
            {
                "ParameterName": "profiler_threshold_ms",
                "ParameterValue": "100",
                "Description": "Operations longer than profiler_threshold_ms
 will be logged",
                "Source": "system",
                "ApplyType": "dynamic",
                "DataType": "integer",
                "AllowedValues": "50-2147483646",
                "IsModifiable": true,
                "ApplyMethod": "pending-reboot"
            },
            {
                "ParameterName": "tls",
                "ParameterValue": "disabled",
                "Description": "Config to enable/disable TLS",
                "Source": "user",
                "ApplyType": "static",
                "DataType": "string",
                "AllowedValues": "disabled,enabled,fips-140-3",
                "IsModifiable": true,
                "ApplyMethod": "pending-reboot"
            }
        ]
}
```

> **Note**
>
> Amazon DocumentDB는 ca-central-1, us-west-2, us-east-1, us-east-2, us-gov-east-1, us-gov-west-1 리전에서 Amazon DocumentDB 5.0(엔진 버전 3.0.3727) 클러스터로 시작하는 FIPS 140-3 엔드포인트를 지원합니다.

`tls` 파라미터의 값을 확인했으면 이어서 다음 섹션에 수록된 코드 예제 중 하나를 사용하여 클러스터에 연결합니다.

- TLS가 활성화된 상태에서 연결
- TLS가 비활성화된 상태에서 연결

TLS가 활성화된 상태에서 연결

TLS 지원 Amazon DocumentDB 클러스터에 프로그램 방식으로 연결하는 코드 예제를 보려면 사용하려는 언어에 해당하는 탭을 선택합니다.

전송 중인 데이터를 암호화하려면 다음 작업을 사용하여 이름이 `global-bundle.pem`인 Amazon DocumentDB에 대한 퍼블릭 키를 다운로드합니다.

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

애플리케이션이 Microsoft Windows에서 실행되어 PKCS7 파일이 필요한 경우에는 PKCS7 인증서 번들을 다운로드할 수 있습니다. 이 번들에는 https://truststore.pki.rds.amazonaws.com/global/global-bundle.p7b의 중간 인증서와 루트 인증서가 모두 포함되어 있습니다.

Python

다음 코드는 TLS가 활성화된 상태에서 Python을 사용하여 Amazon DocumentDB에 연결하는 방법을 보여줍니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
import pymongo
import sys

##Create a MongoDB client, open a connection to Amazon DocumentDB as a replica set
 and specify the read preference as secondary preferred
```

```
client = pymongo.MongoClient('mongodb://sample-user:password@sample-
cluster.node.us-east-1.docdb.amazonaws.com:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

##Specify the database to be used
db = client.sample_database

##Specify the collection to be used
col = db.sample_collection

##Insert a single document
col.insert_one({'hello':'Amazon DocumentDB'})

##Find the document that was previously written
x = col.find_one({'hello':'Amazon DocumentDB'})

##Print the result to the screen
print(x)

##Close the connection
client.close()
```

Node.js

다음 코드는 TLS가 활성화된 상태에서 Node.js를 사용하여 Amazon DocumentDB에 연결하는 방법을 보여줍니다.

> **Important**
>
> Amazon DocumentDB에 대한 IAM ID 인증에서 지원되지 않는 Node.js 드라이버(6.13.1 이전 버전)에는 알려진 제한 사항이 있습니다. Node.js 드라이버를 사용하는 Node.js 드라이버 및 도구(예: mongosh)는 Node.js 드라이버 버전 6.13.1 이상을 사용하도록 업그레이드해야 합니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
var MongoClient = require('mongodb').MongoClient

//Create a MongoDB client, open a connection to DocDB; as a replica set,
//  and specify the read preference as secondary preferred
```

```
var client = MongoClient.connect(
'mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/sample-database?
tls=true&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false',
{
  tlsCAFile: `global-bundle.pem` //Specify the DocDB; cert
},
function(err, client) {
    if(err)
        throw err;

    //Specify the database to be used
    db = client.db('sample-database');

    //Specify the collection to be used
    col = db.collection('sample-collection');

    //Insert a single document
    col.insertOne({'hello':'Amazon DocumentDB'}, function(err, result){
      //Find the document that was previously written
      col.findOne({'hello':'Amazon DocumentDB'}, function(err, result){
        //Print the result to the screen
        console.log(result);

        //Close the connection
        client.close()
      });
   });
});
```

PHP

다음 코드는 TLS가 활성화된 상태에서 PHP를 사용하여 Amazon DocumentDB에 연결하는 방법을 보여줍니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
<?php
//Include Composer's autoloader
require 'vendor/autoload.php';

$TLS_DIR = "/home/ubuntu/global-bundle.pem";
```

```
//Create a MongoDB client and open connection to Amazon DocumentDB
$client = new MongoDB\Client("mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/?retryWrites=false", ["tls" => "true", "tlsCAFile"
 => $TLS_DIR ]);

//Specify the database and collection to be used
$col = $client->sampledatabase->samplecollection;

//Insert a single document
$result = $col->insertOne( [ 'hello' => 'Amazon DocumentDB'] );

//Find the document that was previously written
$result = $col->findOne(array('hello' => 'Amazon DocumentDB'));

//Print the result to the screen
print_r($result);
?>
```

Go

다음 코드는 TLS가 활성화된 상태에서 Go를 사용하여 Amazon DocumentDB에 연결하는 방법을 보여줍니다.

> **Note**
>
> 버전 1.2.1부터 MongoDB Go Driver는 `sslcertificateauthorityfile`에서 발견된 첫 번째 CA 서버 인증서만을 사용합니다. 아래 예제 코드는 `sslcertificateauthorityfile`에서 찾은 모든 서버 인증서를 클라이언트 생성 중에 사용되는 사용자 지정 TLS 구성에 수동으로 추가하여 이 제한을 해결합니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
package main

import (
 "context"
 "fmt"
 "log"
 "time"

 "go.mongodb.org/mongo-driver/bson"
```

```
 "go.mongodb.org/mongo-driver/mongo"
 "go.mongodb.org/mongo-driver/mongo/options"

 "io/ioutil"
 "crypto/tls"
 "crypto/x509"
 "errors"
)

const (
 // Path to the AWS CA file
 caFilePath = "global-bundle.pem"

 // Timeout operations after N seconds
 connectTimeout  = 5
 queryTimeout    = 30
 username        = "sample-user"
 password        = "password"
 clusterEndpoint = "sample-cluster.node.us-east-1.docdb.amazonaws.com:27017"

 // Which instances to read from
 readPreference = "secondaryPreferred"

 connectionStringTemplate = "mongodb://%s:%s@%s/sample-database?
tls=true&replicaSet=rs0&readpreference=%s"
)

func main() {

 connectionURI := fmt.Sprintf(connectionStringTemplate, username, password,
 clusterEndpoint, readPreference)

 tlsConfig, err := getCustomTLSConfig(caFilePath)
 if err != nil {
  log.Fatalf("Failed getting TLS configuration: %v", err)
 }

 client, err :=
 mongo.NewClient(options.Client().ApplyURI(connectionURI).SetTLSConfig(tlsConfig))
 if err != nil {
  log.Fatalf("Failed to create client: %v", err)
 }
```

```
ctx, cancel := context.WithTimeout(context.Background(),
connectTimeout*time.Second)
defer cancel()

err = client.Connect(ctx)
if err != nil {
 log.Fatalf("Failed to connect to cluster: %v", err)
}

// Force a connection to verify our connection string
err = client.Ping(ctx, nil)
if err != nil {
 log.Fatalf("Failed to ping cluster: %v", err)
}

fmt.Println("Connected to DocumentDB!")

collection := client.Database("sample-database").Collection("sample-collection")

ctx, cancel = context.WithTimeout(context.Background(), queryTimeout*time.Second)
defer cancel()

res, err := collection.InsertOne(ctx, bson.M{"name": "pi", "value": 3.14159})
if err != nil {
 log.Fatalf("Failed to insert document: %v", err)
}

id := res.InsertedID
log.Printf("Inserted document ID: %s", id)

ctx, cancel = context.WithTimeout(context.Background(), queryTimeout*time.Second)
defer cancel()

cur, err := collection.Find(ctx, bson.D{})

if err != nil {
 log.Fatalf("Failed to run find query: %v", err)
}
defer cur.Close(ctx)

for cur.Next(ctx) {
 var result bson.M
 err := cur.Decode(&result)
 log.Printf("Returned: %v", result)
```

```
  if err != nil {
   log.Fatal(err)
  }
 }

 if err := cur.Err(); err != nil {
  log.Fatal(err)
 }

}

func getCustomTLSConfig(caFile string) (*tls.Config, error) {
 tlsConfig := new(tls.Config)
 certs, err := ioutil.ReadFile(caFile)

 if err != nil {
  return tlsConfig, err
 }

 tlsConfig.RootCAs = x509.NewCertPool()
 ok := tlsConfig.RootCAs.AppendCertsFromPEM(certs)

 if !ok {
  return tlsConfig, errors.New("Failed parsing pem file")
 }

 return tlsConfig, nil
```

Java

Java 애플리케이션에서 TLS 지원 Amazon DocumentDB 클러스터에 연결할 때 프로그램은 AWS 제공 인증 기관(CA) 파일을 사용하여 연결을 검증해야 합니다. Amazon RDS CA 인증서를 사용하려면 다음을 수행합니다.

1. https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem 에서 Amazon RDS CA 파일을 다운로드하세요.
2. 다음 명령을 수행하여 파일에 포함된 CA 인증서로 트러스트 스토어를 생성합니다. *truststore-password*를 다른 값으로 변경해야 합니다. 이전 CA 인증서(rds-ca-2015-root.pem)와 새 CA 인증서(rds-ca-2019-root.pem)가 모두 포함된 트러스트 저장소에 액세스하는 경우 인증서 번들을 트러스트 저장소로 가져올 수 있습니다.

다음은 Linux 운영 체제에서 트러스트 스토어로 인증서 번들을 가져오는 샘플 쉘 스크립트입니다. 다음 예에서는 자신의 정보로 각각의 *### ## ## ###*를 바꿉니다. 특히 예제 디렉터리 "*mydir*"가 스크립트에 있는 경우 이 작업을 위해 생성한 디렉터리로 교체합니다.

```
mydir=/tmp/certs
truststore=${mydir}/rds-truststore.jks
storepassword=truststore-password

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem" >
 ${mydir}/global-bundle.pem
awk 'split_after == 1 {n++;split_after=0} /-----END CERTIFICATE-----/
 {split_after=1}{print > "rds-ca-" n ".pem"}' < ${mydir}/global-bundle.pem

for CERT in rds-ca-*; do
  alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /
Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep
 Alias | cut -d " " -f3- | while read alias
do
   expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword}
 -alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print
 "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

다음은 macOS에서 트러스트 스토어로 인증서 번들을 가져오는 샘플 쉘 스크립트입니다.

```
mydir=/tmp/certs
truststore=${mydir}/rds-truststore.jks
storepassword=truststore-password
```

```
curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem" >
 ${mydir}/global-bundle.pem
split -p "-----BEGIN CERTIFICATE-----" ${mydir}/global-bundle.pem rds-ca-

for CERT in rds-ca-*; do
  alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /
Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep
 Alias | cut -d " " -f3- | while read alias
do
   expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword}
 -alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print
 "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

3. Amazon DocumentDB 클러스터에 연결하기 전에 애플리케이션에서 다음 시스템 속성을 설정하여 프로그램에서 keystore를 사용합니다.

```
javax.net.ssl.trustStore: truststore
javax.net.ssl.trustStorePassword: truststore-password;
```

4. 다음 코드는 TLS가 활성화된 상태에서 Java를 사용하여 Amazon DocumentDB에 연결하는 방법을 보여줍니다.

   다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
package com.example.documentdb;

import com.mongodb.client.*;
import org.bson.Document;
```

```
public final class Test {
    private Test() {
    }
    public static void main(String[] args) {

        String template = "mongodb://%s:%s@%s/sample-database?
ssl=true&replicaSet=rs0&readpreference=%s";
        String username = "sample-user";
        String password = "password";
        String clusterEndpoint = "sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017";
        String readPreference = "secondaryPreferred";
        String connectionString = String.format(template, username, password,
 clusterEndpoint, readPreference);

        String truststore = "truststore";
        String truststorePassword = "truststore-password";

        System.setProperty("javax.net.ssl.trustStore", truststore);
        System.setProperty("javax.net.ssl.trustStorePassword",
 truststorePassword);

        MongoClient mongoClient = MongoClients.create(connectionString);

        MongoDatabase testDB = mongoClient.getDatabase("sample-database");
        MongoCollection<Document> numbersCollection =
 testDB.getCollection("sample-collection");

        Document doc = new Document("name", "pi").append("value", 3.14159);
        numbersCollection.insertOne(doc);

        MongoCursor<Document> cursor = numbersCollection.find().iterator();
        try {
            while (cursor.hasNext()) {
                System.out.println(cursor.next().toJson());
            }
        } finally {
            cursor.close();
        }

    }
}
```

C# / .NET

다음 코드는 TLS가 활성화된 상태에서 C# / .NET를 사용하여 Amazon DocumentDB에 연결하는 방법을 보여줍니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
using System;
using System.Text;
using System.Linq;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;
using System.Net.Security;
using MongoDB.Driver;
using MongoDB.Bson;

namespace DocDB
{
    class Program
    {
        static void Main(string[] args)
        {
            string template = "mongodb://{0}:{1}@{2}/sampledatabase?
tls=true&replicaSet=rs0&readpreference={3}";
            string username = "sample-user";
            string password = "password";
            string readPreference = "secondaryPreferred";
            string clusterEndpoint="sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017";
            string connectionString = String.Format(template, username, password,
 clusterEndpoint, readPreference);

            string pathToCAFile = "<PATH/global-bundle.p7b_file>";

            // ADD CA certificate to local trust store
            // DO this once - Maybe when your service starts
            X509Store localTrustStore = new X509Store(StoreName.Root);
            X509Certificate2Collection certificateCollection = new
 X509Certificate2Collection();
            certificateCollection.Import(pathToCAFile);
            try
            {
                localTrustStore.Open(OpenFlags.ReadWrite);
```

```
                localTrustStore.AddRange(certificateCollection);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Root certificate import failed: " + ex.Message);
                throw;
            }
            finally
            {
                localTrustStore.Close();
            }

            var settings = MongoClientSettings.FromUrl(new
 MongoUrl(connectionString));
            var client = new MongoClient(settings);

            var database = client.GetDatabase("sampledatabase");
            var collection =
 database.GetCollection<BsonDocument>("samplecollection");
            var docToInsert = new BsonDocument { { "pi", 3.14159 } };
            collection.InsertOne(docToInsert);
        }
    }
}
```

MongoDB Shell

다음 코드는 TLS가 활성화된 상태에서 새 버전, mongosh 또는 이전의 Mongo 쉘 버전을 사용하여 Amazon DocumentDB에 연결하고 쿼리하는 방법을 보여줍니다.

mongosh를 사용하여 Amazon DocumentDB에 연결

> **Important**
>
> Amazon DocumentDB에 대한 IAM ID 인증에서 지원되지 않는 Node.js 드라이버(6.13.1 이전 버전)에는 알려진 제한 사항이 있습니다. Node.js 드라이버를 사용하는 Node.js 드라이버 및 도구(예: mongosh)는 Node.js 드라이버 버전 6.13.1 이상을 사용하도록 업그레이드해야 합니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
mongosh --tls --host cluster-end-point:27017 --tlsCAFile  global-bundle.pem --
username sample-user --password password --retryWrites false
```

이전의 Mongo 쉘 버전을 사용하여 Amazon DocumentDB에 연결

IAM을 사용하는 경우 이전 버전의 Mongo 쉘을 사용해야 합니다. 다음 명령 옵션 중 하나를 입력합니다.

```
mongo --ssl --host cluster-end-point:27017 --sslCAFile global-bundle.pem --
username sample-user --password password
```

4.2 이상의 버전을 사용하는 경우 다음 코드를 사용하여 연결하세요. 엔진 버전 8.0.2부터 Amazon DocumentDB는 재시도 가능한 쓰기를 지원합니다. 이전 엔진 버전에서는 재시도 가능한 쓰기가 지원되지 않습니다. mongo 쉘(mongosh 제외)을 사용하는 경우 코드 문자열에 `retryWrites=false` 명령을 포함시키지 마십시오. 자세한 내용은 Amazon DocumentDB에서 재시도 가능한 쓰기 섹션을 참조하세요.

```
mongo --tls --host cluster-end-point:27017 --tlsCAFile global-bundle.pem --
username sample-user --password password
```

연결 테스트

1. 단일 문서를 삽입합니다.

   ```
   db.myTestCollection.insertOne({'hello':'Amazon DocumentDB'})
   ```

2. 이전에 삽입된 문서를 찾습니다.

   ```
   db.myTestCollection.find({'hello':'Amazon DocumentDB'})
   ```

R

다음 코드는 TLS가 활성화된 상태에서 mongolite(https://jeroen.github.io/mongolite/)를 사용하여 R을 Amazon DocumentDB에 연결하고 쿼리하는 방법을 보여줍니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
#Include the mongolite library.
library(mongolite)
```

```
mongourl <- paste("mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/test2?ssl=true&",
          "readPreference=secondaryPreferred&replicaSet=rs0", sep="")

#Create a MongoDB client, open a connection to Amazon DocumentDB as a replica
#   set and specify the read preference as secondary preferred
client <-  mongo(url = mongourl, options = ssl_options(weak_cert_validation = F, ca
 ="<PATH/global-bundle.pem>"))

#Insert a single document
str <- c('{"hello" : "Amazon DocumentDB"}')
client$insert(str)

#Find the document that was previously written
client$find()
```

Ruby

다음 코드는 TLS가 활성화된 상태에서 Java를 사용하여 Ruby를 Amazon DocumentDB에 연결하는 방법을 보여줍니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
require 'mongo'
require 'neatjson'
require 'json'
client_host = 'mongodb://sample-cluster.node.us-east-1.docdb.amazonaws.com:27017'
client_options = {
   database: 'test',
   replica_set: 'rs0',
   read: {:secondary_preferred => 1},
   user: 'sample-user',
   password: 'password',
   ssl: true,
   ssl_verify: true,
   ssl_ca_cert: 'PATH/global-bundle.pem',
   retry_writes: false
}


begin
   ##Create a MongoDB client, open a connection to Amazon DocumentDB as a
   ##   replica set and specify the read preference as secondary preferred
```

```
    client = Mongo::Client.new(client_host, client_options)

    ##Insert a single document
    x = client[:test].insert_one({"hello":"Amazon DocumentDB"})

    ##Find the document that was previously written
    result = client[:test].find()

    #Print the document
    result.each do |document|
       puts JSON.neat_generate(document)
    end
end

#Close the connection
client.close
```

TLS가 비활성화된 상태에서 연결

TLS가 비활성화된 Amazon DocumentDB 클러스터에 프로그래밍 방식으로 연결하기 위한 코드 예제를 보려면 사용할 언어에 해당하는 탭을 선택합니다.

Python

다음 코드는 TLS가 비활성화된 상태에서 Python을 사용하여 Amazon DocumentDB에 연결하는 방법을 보여줍니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
## Create a MongoDB client, open a connection to Amazon DocumentDB as a replica set
 and specify the read preference as secondary preferred

import pymongo
import sys

client = pymongo.MongoClient('mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/?
replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

##Specify the database to be used
db = client.sample_database
```

```
##Specify the collection to be used
col = db.sample_collection

##Insert a single document
col.insert_one({'hello':'Amazon DocumentDB'})

##Find the document that was previously written
x = col.find_one({'hello':'Amazon DocumentDB'})

##Print the result to the screen
print(x)

##Close the connection
client.close()
```

Node.js

다음 코드는 TLS가 비활성화된 상태에서 Node.js를 사용하여 Amazon DocumentDB에 연결하는 방법을 보여줍니다.

> **Important**
>
> Amazon DocumentDB에 대한 IAM ID 인증에서 지원되지 않는 Node.js 드라이버(6.13.1 이전 버전)에는 알려진 제한 사항이 있습니다. Node.js 드라이버를 사용하는 Node.js 드라이버 및 도구(예: mongosh)는 Node.js 드라이버 버전 6.13.1 이상을 사용하도록 업그레이드해야 합니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
var MongoClient = require('mongodb').MongoClient;

//Create a MongoDB client, open a connection to Amazon DocumentDB as a replica set,
//  and specify the read preference as secondary preferred
var client = MongoClient.connect(
'mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/sample-database?
replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false',
{
  useNewUrlParser: true
},
```

```
function(err, client) {
    if(err)
        throw err;
    //Specify the database to be used
    db = client.db('sample-database');

    //Specify the collection to be used
    col = db.collection('sample-collection');

    //Insert a single document
    col.insertOne({'hello':'Amazon DocumentDB'}, function(err, result){
      //Find the document that was previously written
      col.findOne({'hello':'Amazon DocumentDB'}, function(err, result){
        //Print the result to the screen
        console.log(result);

        //Close the connection
        client.close()
      });
   });
});
```

PHP

다음 코드는 TLS가 비활성화된 상태에서 PHP를 사용하여 Amazon DocumentDB에 연결하는 방법을 보여줍니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
<?php
//Include Composer's autoloader
require 'vendor/autoload.php';

//Create a MongoDB client and open connection to Amazon DocumentDB
$client = new MongoDB\Client("mongodb://sample-user:password@sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017/?retryWrites=false");

//Specify the database and collection to be used
$col = $client->sampledatabase->samplecollection;

//Insert a single document
$result = $col->insertOne( [ 'hello' => 'Amazon DocumentDB'] );

//Find the document that was previously written
```

```
$result = $col->findOne(array('hello' => 'Amazon DocumentDB'));

//Print the result to the screen
print_r($result);
?>
```

Go

다음 코드는 TLS가 비활성화된 상태에서 Go를 사용하여 Amazon DocumentDB에 연결하는 방법을 보여줍니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
package main

import (
 "context"
 "fmt"
 "log"
 "time"

 "go.mongodb.org/mongo-driver/bson"
 "go.mongodb.org/mongo-driver/mongo"
 "go.mongodb.org/mongo-driver/mongo/options"
)

const (
 // Timeout operations after N seconds
 connectTimeout  = 5
 queryTimeout    = 30
 username        = "sample-user"
 password        = "password"
 clusterEndpoint = "sample-cluster.node.us-east-1.docdb.amazonaws.com:27017"

 // Which instances to read from
 readPreference           = "secondaryPreferred"
 connectionStringTemplate = "mongodb://%s:%s@%s/sample-database?
replicaSet=rs0&readpreference=%s"
)

func main() {

 connectionURI := fmt.Sprintf(connectionStringTemplate, username, password,
 clusterEndpoint, readPreference)
```

```
client, err := mongo.NewClient(options.Client().ApplyURI(connectionURI))
if err != nil {
 log.Fatalf("Failed to create client: %v", err)
}

ctx, cancel := context.WithTimeout(context.Background(),
connectTimeout*time.Second)
defer cancel()

err = client.Connect(ctx)
if err != nil {
 log.Fatalf("Failed to connect to cluster: %v", err)
}

// Force a connection to verify our connection string
err = client.Ping(ctx, nil)
if err != nil {
 log.Fatalf("Failed to ping cluster: %v", err)
}

fmt.Println("Connected to DocumentDB!")

collection := client.Database("sample-database").Collection("sample-collection")

ctx, cancel = context.WithTimeout(context.Background(), queryTimeout*time.Second)
defer cancel()

res, err := collection.InsertOne(ctx, bson.M{"name": "pi", "value": 3.14159})
if err != nil {
 log.Fatalf("Failed to insert document: %v", err)
}

id := res.InsertedID
log.Printf("Inserted document ID: %s", id)

ctx, cancel = context.WithTimeout(context.Background(), queryTimeout*time.Second)
defer cancel()

cur, err := collection.Find(ctx, bson.D{})

if err != nil {
 log.Fatalf("Failed to run find query: %v", err)
}
```

```
 defer cur.Close(ctx)

 for cur.Next(ctx) {
  var result bson.M
  err := cur.Decode(&result)
  log.Printf("Returned: %v", result)

  if err != nil {
   log.Fatal(err)
  }
 }

 if err := cur.Err(); err != nil {
  log.Fatal(err)
 }

}
```

Java

다음 코드는 TLS가 비활성화된 상태에서 Java를 사용하여 Amazon DocumentDB에 연결하는 방법을 보여줍니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
package com.example.documentdb;

import com.mongodb.MongoClient;
import com.mongodb.MongoClientURI;
import com.mongodb.ServerAddress;
import com.mongodb.MongoException;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.MongoCollection;
import org.bson.Document;


public final class Main {
    private Main() {
    }
    public static void main(String[] args) {

        String template = "mongodb://%s:%s@%s/sample-database?
replicaSet=rs0&readpreference=%s";
```

```
        String username = "sample-user";
        String password = "password";
        String clusterEndpoint = "sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017";
        String readPreference = "secondaryPreferred";
        String connectionString = String.format(template, username, password,
 clusterEndpoint, readPreference);

        MongoClientURI clientURI = new MongoClientURI(connectionString);
        MongoClient mongoClient = new MongoClient(clientURI);

        MongoDatabase testDB = mongoClient.getDatabase("sample-database");
        MongoCollection<Document> numbersCollection = testDB.getCollection("sample-
collection");

        Document doc = new Document("name", "pi").append("value", 3.14159);
        numbersCollection.insertOne(doc);

        MongoCursor<Document> cursor = numbersCollection.find().iterator();
        try {
            while (cursor.hasNext()) {
                System.out.println(cursor.next().toJson());
            }
        } finally {
            cursor.close();
        }

    }
}
```

C# / .NET

다음 코드는 TLS가 비활성화된 상태에서 C# / .NET를 사용하여 Amazon DocumentDB에 연결하는 방법을 보여줍니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
using System;
using System.Text;
using System.Linq;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;
using System.Net.Security;
```

```
using MongoDB.Driver;
using MongoDB.Bson;

namespace CSharpSample
{
   class Program
    {
       static void Main(string[] args)
        {
           string template = "mongodb://{0}:{1}@{2}/sampledatabase?
replicaSet=rs0&readpreference={3}";
           string username = "sample-user";
           string password = "password";
           string clusterEndpoint = "sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017";
           string readPreference = "secondaryPreferred";
           string connectionString = String.Format(template, username, password,
 clusterEndpoint, readPreference);

           var settings = MongoClientSettings.FromUrl(new
 MongoUrl(connectionString));
           var client = new MongoClient(settings);

           var database = client.GetDatabase("sampledatabase");
           var collection =
 database.GetCollection<BsonDocument>("samplecollection");
           var docToInsert = new BsonDocument { { "pi", 3.14159 } };
            collection.InsertOne(docToInsert);
        }
    }
}
```

MongoDB Shell

다음 코드는 TLS가 비활성화된 상태에서 새 버전, mongosh 또는 이전의 Mongo 쉘 버전을 사용하여 Amazon DocumentDB에 연결하고 쿼리하는 방법을 보여줍니다.

mongosh를 사용하여 Amazon DocumentDB에 연결

> **Important**
>
> Amazon DocumentDB에 대한 IAM ID 인증에서 지원되지 않는 Node.js 드라이버(6.13.1 이전 버전)에는 알려진 제한 사항이 있습니다. Node.js 드라이버를 사용하는 Node.js 드라이

버 및 도구(예: mongosh)는 Node.js 드라이버 버전 6.13.1 이상을 사용하도록 업그레이드해야 합니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
mongosh --host cluster-end-point:27017 --username sample-user --password password --
retryWrites false
```

이전의 Mongo 쉘 버전을 사용하여 Amazon DocumentDB에 연결

IAM을 사용하는 경우 이전 버전의 Mongo 쉘을 사용해야 합니다. 다음 명령 옵션 중 하나를 입력합니다.

```
mongo --host cluster-end-point:27017 --username sample-user --password password
```

4.2 이상의 버전을 사용하는 경우 다음 코드를 사용하여 연결하세요. Amazon DocumentDB에서는 재시도 가능한 쓰기가 지원되지 않습니다. mongo 쉘(mongosh 제외)을 사용하는 경우 코드 문자열에 `retryWrites=false` 명령을 포함시키지 마십시오. 기본적으로 재시도 가능한 쓰기는 비활성화되어 있습니다. `retryWrites=false`를 포함하면 일반 읽기 명령에서 오류가 발생할 수 있습니다.

```
mongo --host cluster-end-point:27017 --username sample-user --password password
```

연결 테스트

1. 단일 문서를 삽입합니다.

   ```
   db.myTestCollection.insertOne({'hello':'Amazon DocumentDB'})
   ```

2. 이전에 삽입된 문서를 찾습니다.

   ```
   db.myTestCollection.find({'hello':'Amazon DocumentDB'})
   ```

R

다음 코드는 TLS가 비활성화된 상태에서 mongolite(https://jeroen.github.io/mongolite/)를 사용하여 R을 Amazon DocumentDB에 연결하는 방법을 보여줍니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
#Include the mongolite library.
library(mongolite)

#Create a MongoDB client, open a connection to Amazon DocumentDB as a replica
#   set and specify the read preference as secondary preferred
client <- mongo(url = "mongodb://sample-user:password@sample-
cluster.node.us-east-1.docdb.amazonaws.com:27017/sample-database?
readPreference=secondaryPreferred&replicaSet=rs0")

##Insert a single document
str <- c('{"hello" : "Amazon DocumentDB"}')
client$insert(str)

##Find the document that was previously written
client$find()
```

Ruby

다음 코드는 TLS가 활성화된 상태에서 Ruby를 Amazon DocumentDB에 연결하는 방법을 보여줍니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
require 'mongo'
require 'neatjson'
require 'json'
client_host = 'mongodb://sample-cluster.node.us-east-1.docdb.amazonaws.com:27017'
client_options = {
   database: 'test',
   replica_set: 'rs0',
   read: {:secondary_preferred => 1},
   user: 'sample-user',
   password: 'password',
   retry_writes: false
}

begin
   ##Create a MongoDB client, open a connection to Amazon DocumentDB as a
   ##   replica set and specify the read preference as secondary preferred
   client = Mongo::Client.new(client_host, client_options)
```

```
   ##Insert a single document
   x = client[:test].insert_one({"hello":"Amazon DocumentDB"})

   ##Find the document that was previously written
   result = client[:test].find()

   #Print the document
   result.each do |document|
      puts JSON.neat_generate(document)
   end
end

#Close the connection
client.close
```

Amazon DocumentDB의 네트워크 압축

유선 프로토콜 압축을 사용하면 애플리케이션이 네트워크를 통해 전송하는 데이터를 줄일 수 있습니다. Amazon DocumentDB 8.0.1 이상은 압축을 지원합니다. 대용량 문서 또는 대용량 결과 세트를 전송하는 워크로드의 처리량이 향상됩니다. 압축을 활성화하면 드라이버와 서버가 압축된 메시지를 교환합니다.

네트워크 압축을 활성화하면 CPU 리소스를 교환하여 네트워크 대역폭을 줄일 수 있습니다. 워크로드가 네트워크 바운드이고 대용량 페이로드를 전송할 때 압축을 활성화합니다. 작은 요청 및 응답의 경우 압축 오버헤드가 절감액을 초과할 수 있으므로 순 이점이 보이지 않을 수 있습니다.

지원되는 압축기

Amazon DocumentDB는 `snappy` 압축기를 지원합니다. Amazon DocumentDB는 `zlib` 및와 같이 MongoDB 드라이버가 제공하는 다른 압축기를 지원하지 않습니다`zstd`.

드라이버가 압축기 목록을 요청하는 경우 Amazon DocumentDB는 지원되는 압축기만 협상합니다. 예를 들어 드라이버가 `snappy` 및를 제공하는 경우 `zstd` Amazon DocumentDB는를 선택하고 `snappy` 무시합니다`zstd`. 드라이버가 지원되지 않는 압축기만 제공하는 경우 Amazon DocumentDB는 연결을 압축하지 않습니다. 이 경우 드라이버와 서버는 압축 없이 메시지를 교환합니다.

압축 활성화

압축을 활성화하려면 연결 문자열에 `compressors` 옵션을 추가하고 로 설정합니다`snappy`. 다음 연결 문자열에서 , *`password`*및 *`usernamedocdb-cluster-endpoint`*를 고유한 값으로 바꿉니다.

```
mongodb://username:password@docdb-cluster-endpoint:27017/?
tls=true&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false&compressors=snappy
```

MongoDB 드라이버를 사용하는 경우 드라이버의 연결 옵션을 통해 동일한 옵션을 설정합니다. 대부분의 드라이버는 스냅 압축을 수행하기 위해 추가 클라이언트 측 종속성이 필요합니다.

- Node.js 드라이버에는 `snappy` 패키지가 필요합니다.
- Python 드라이버(PyMongo)에는 `python-snappy` 패키지가 필요합니다. Linux에서는 시스템 스냅 라이브러리`python-snappy`도 필요합니다(예: `dnf` 또는를 `snappy-devel` 통해`yum`).

정확한 옵션 이름과 필요한 패키지는 드라이버 설명서를 참조하세요.

> **Note**
>
> 클라이언트 측 압축 종속성이 누락된 경우 드라이버가 성공적으로 연결됩니다. 그러나 압축 없이 메시지를 교환하고 오류를 반환하지 않습니다. 압축이 활성 상태인지 확인하려면에 설명된 대로 협상된 압축기를 확인합니다압축 확인.

다음 Python 코드 예제는 스냅 압축이 활성화된 클러스터에 연결하고 및 `try` `except` 블록을 사용하여 연결 오류를 처리합니다.

```
import pymongo

try:
    client = pymongo.MongoClient(
        'mongodb://username:password@docdb-cluster-endpoint:27017/'
        '?tls=true&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false'
        '&compressors=snappy',
        tlsCAFile='global-bundle.pem')

    db = client['sample-database']
    col = db['sample-collection']
    col.insert_one({'item': 'DocumentDB'})
    print(col.find_one())
except Exception as error:
    print(f'Connection failed: {error}')
```

압축 확인

드라이버와 Amazon DocumentDB가 압축을 협상했는지 확인하려면 `hello` MongoDB 유선 프로토콜 명령이 반환하는 `compression` 필드를 검사합니다. 드라이버와 Amazon DocumentDB가 압축을 협상할 때이 필드에는 합의된 압축기가 나열됩니다.

Amazon VPC 외부에서 Amazon DocumentDB 클러스터에 연결

Amazon DocumentDB(MongoDB 호환) 클러스터는 Amazon Virtual Private Cloud (Amazon VPC) 내에 배포됩니다. Amazon EC2 인스턴스 또는 동일한 Amazon VPC에 배포된 다른 AWS 서비스에서 직접 액세스할 수 있습니다. 또한 EC2 인스턴스 또는 VPCs 피어링을 통해 동일한 AWS 리전 리전 또는 다른 리전의 다른 VPC에 있는 다른 AWS 서비스에서 Amazon DocumentDB에 액세스할 수 있습니다.

그러나 사용 사례에서 사용자 또는 사용자의 애플리케이션이 클러스터의 VPC 외부에서 Amazon DocumentDB 리소스에 액세스해야 한다고 가정합니다. 그럴 경우 SSH 터널링(포트 포워딩이라고도 함)을 사용하여 Amazon DocumentDB 리소스에 액세스할 수 있습니다.

SSH 터널링에 대한 자세한 내용은 이 주제의 범위를 벗어납니다. SSH 터널링에 대한 자세한 내용은 다음을 참조하세요.

- SSH 터널
- SSH 포트 전송 예, 특히 로컬 전송 섹션

SSH 터널을 만들려면 Amazon DocumentDB 클러스터와 동일한 Amazon VPC에서 실행 중인 Amazon EC2 인스턴스가 필요합니다. 클러스터와 동일한 VPC에서 기존 EC2 인스턴스를 사용하거나 새 인스턴스를 생성할 수 있습니다. 자세한 내용은 사용 중인 운영 체제에 해당하는 주제를 참조하세요.

- Amazon EC2 Linux 인스턴스 시작하기
- Amazon EC2 Windows 인스턴스 시작하기

일반적으로 다음 명령을 사용하여 EC2 인스턴스에 연결할 수 있습니다.

```
ssh -i "ec2Access.pem" ubuntu@ec2-34-229-221-164.compute-1.amazonaws.com
```

그 경우 로컬 컴퓨터에서 다음 명령을 사용하여 SSH 터널을 Amazon DocumentDB 클러스터 `sample-cluster.node.us-east-1.docdb.amazonaws.com`으로 설정할 수 있습니다. `-L` 플래그는 로컬 포트를 전달하는 데 사용됩니다. SSH 터널을 사용하는 경우 클러스터 엔드포인트를 사용하

여 클러스터에 연결하고 복제본 세트 모드(예: 연결 문자열`replicaSet=rs0`에 지정)에서 연결을 시도하지 마십시오. 오류가 발생합니다.

```
ssh -i "ec2Access.pem" -L 27017:sample-cluster.node.us-east-1.docdb.amazonaws.com:27017
 ubuntu@ec2-34-229-221-164.compute-1.amazonaws.com -N
```

SSH 터널이 생성된 이후에 `localhost:27017`에 대해 실행한 명령은 Amazon VPC에서 실행 중인 Amazon DocumentDB 클러스터 `sample-cluster`에 전달됩니다. Amazon DocumentDB 클러스터에서 전송 계층 보안 (TLS) 이 활성화된 경우 Amazon DocumentDB의 퍼블릭 키를 에서 다운로드해야 합니다. https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem 다음 작업은 이 파일을 다운로드합니다:

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

> **Note**
>
> 새 Amazon DocumentDB 클러스터에는 기본적으로 TLS가 사용 설정되어 있습니다. 그러나 비활성화는 가능합니다. 자세한 내용은 Amazon DocumentDB 클러스터 TLS 설정 관리 단원을 참조하십시오.

Amazon VPC 외부에서 Amazon DocumentDB 클러스터에 연결하려면 다음 명령을 사용하세요.

```
mongo --sslAllowInvalidHostnames --ssl --sslCAFile global-bundle.pem --username
 <yourUsername> --password <yourPassword>
```

Amazon DocumentDB에 복제본 세트로 연결

Amazon DocumentDB를 기반으로 개발할 때는 클러스터에 복제본 세트로 연결하고 드라이버의 기본 제공 읽기 기본 설정 기능을 사용하여 복제본 인스턴스에 읽기를 배포합니다. 이 섹션에서는 그 의미에 대해 자세히 살펴보고 Python을 예로 사용하여 Amazon DocumentDB 클러스터를 복제본 세트로 연결하는 방법을 설명합니다.

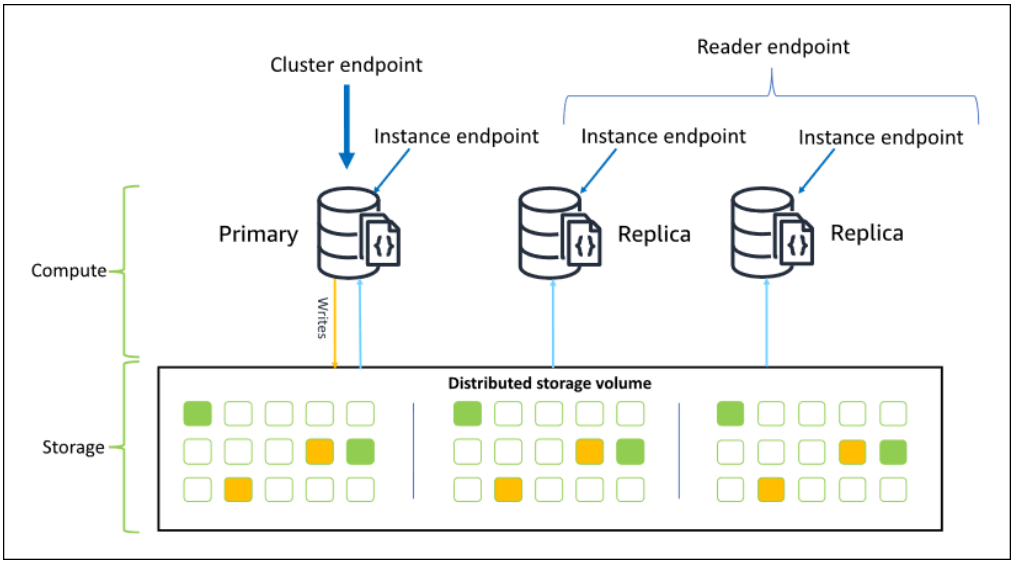
Amazon DocumentDB에는 클러스터에 연결하는 데 사용할 수 있는 세 가지 엔드포인트가 있습니다:

- 클러스터 엔드포인트
- 리더 엔드포인트
- 인스턴스 엔드포인트

Amazon DocumentDB에 연결할 때 클러스터 엔드포인트를 사용합니다. 이는 다음 다이어그램과 같이 클러스터의 기본 인스턴스를 가리키는 CNAME입니다.

SSH 터널을 사용하는 경우 클러스터 엔드포인트를 사용하여 클러스터에 연결합니다. 복제본 세트 모드(연결 문자열`replicaSet=rs0`에서 지정)로 연결하지 마십시오. 오류가 발생합니다.

> **Note**
>
> Amazon DocumentDB 엔드포인트에 대한 자세한 내용은 Amazon DocumentDB 엔드포인트 섹션을 참조하세요.



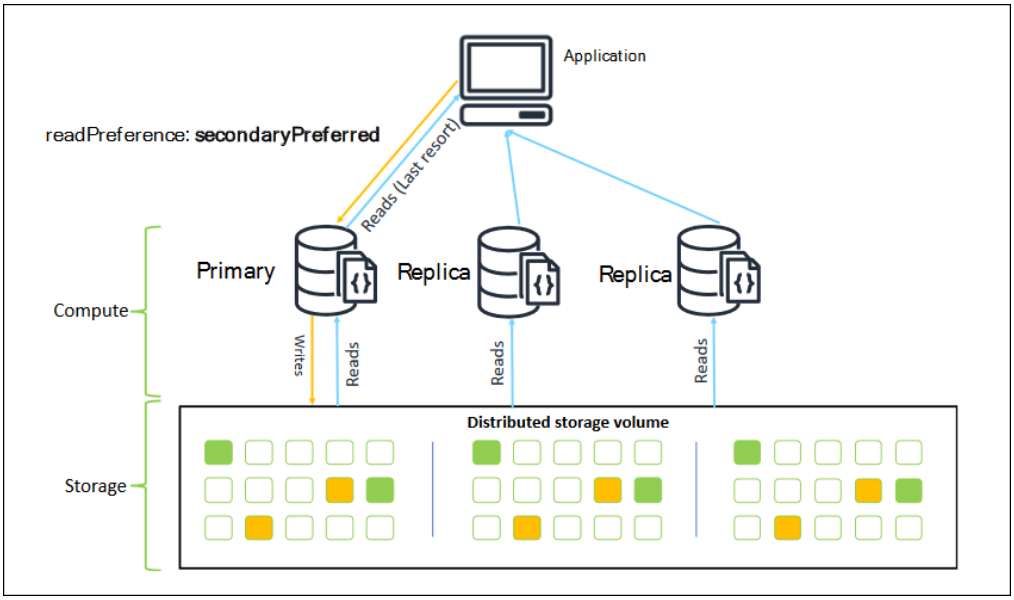
클러스터 엔드포인트를 사용하여 복제본 세트 모드에서 클러스터에 연결할 수 있습니다. 그런 다음 내장된 읽기 기본 설정 드라이버 기능을 사용할 수 있습니다. 다음 예제에서 `/?replicaSet=rs0`을 지정하면 복제본 세트로 연결하려는 SDK를 나타냅니다. `/?replicaSet=rs0'`을 생략하면 클라이언트는 모든 요청을 클러스터 엔드포인트, 즉 기본 인스턴스로 라우팅합니다.

```
## Create a MongoDB client, open a connection to Amazon DocumentDB as a
   ## replica set and specify the read preference as secondary preferred
   client = pymongo.MongoClient('mongodb://<user-name>:<password>@mycluster.node.us-
east-1.docdb.amazonaws.com:27017/?replicaSet=rs0')
```

복제본 세트로 연결하면 클러스터에서 인스턴스를 추가하거나 제거할 때를 포함하여 SDK가 클러스터 지형을 자동으로 검색할 수 있다는 장점이 있습니다. 그런 다음 읽기 요청을 복제본 인스턴스로 라우팅하여 클러스터를 보다 효율적으로 사용할 수 있습니다.

복제본 세트로 연결할 때 연결을 위해 `readPreference`를 지정할 수 있습니다. `secondaryPreferred`의 읽기 기본 설정을 지정하면 클라이언트는 읽기 쿼리를 복제본에 라우팅하고 쿼리를 기본 인스턴스에 씁니다(다음 다이어그램 참조). 이것은 클러스터 리소스를 더 잘 사용하는 것입니다. 자세한 내용은 읽기 기본 설정 옵션 섹션을 참조하십시오.

```
## Create a MongoDB client, open a connection to Amazon DocumentDB as a
   ##   replica set and specify the read preference as secondary preferred
client = pymongo.MongoClient('mongodb://<user-name>:<password>@mycluster.node.us-
east-1.docdb.amazonaws.com:27017/?replicaSet=rs0&readPreference=secondaryPreferred')
```



Amazon DocumentDB 복제본의 읽기는 결국 일치합니다. 기본 데이터에 기록된 순서와 동일한 순서로 데이터를 반환하며 50ms 미만의 복제 지연이 있는 경우가 많습니다. Amazon CloudWatch 지표 `DBInstanceReplicaLag`및 `DBClusterReplicaLagMaximum`를 사용하여 클러스터의 복제 지연을 모니터링할 수 있습니다. 자세한 내용은 CloudWatch를 사용하여 Amazon DocumentDB 모니터링 섹션을 참조하십시오.

기존의 단일 데이터베이스 아키텍처와 달리 Amazon DocumentDB는 스토리지와 컴퓨팅을 분리합니다. 이러한 최신 아키텍처를 고려할 때, 복제본 인스턴스에서 읽기 확장을 권장합니다. 복제본 인스턴스에 대한 읽기는 기본 인스턴스에서 복제되는 쓰기를 차단하지 않습니다. 클러스터에 최대 15개의 읽기 전용 복제본 인스턴스를 추가하고 초당 수백만 건의 읽기로 스케일 아웃할 수 있습니다.

복제본 세트로 연결하고 복제본에 읽기를 배포할 때 얻을 수 있는 주요 이점은 클러스터에서 애플리케이션에 사용할 수 있는 전체 리소스가 늘어난다는 것입니다. 복제본 세트로 연결하는 것이 가장 좋습니다. 이는 다음 시나리오에서 가장 유용합니다.

- 기본에서 거의 100% CPU를 사용하고 있습니다.
- 버퍼 캐시 적중률이 거의 0입니다.
- 개별 인스턴스의 연결 또는 커서 한계에 도달합니다.

클러스터 인스턴스 크기를 확장하는 것은 옵션이며 경우에 따라 클러스터를 확장하는 가장 좋은 방법일 수 있습니다. 그러나 이미 클러스터에 있는 복제본을 더 잘 사용하는 방법도 고려해야 합니다. 이를 통해 더 큰 인스턴스 유형을 사용하는 비용을 늘리지 않고도 규모를 조정할 수 있습니다. 또한 CloudWatch 알람을 사용하여 이러한 제한(즉, `CPUUtilization`, `DatabaseConnections`, 및 `BufferCacheHitRatio`)을 모니터링하고 경고하여 리소스가 많이 사용되는 시기를 파악하는 것이 좋습니다.

자세한 내용은 다음 주제를 참조하세요.

- Amazon DocumentDB 모범 사례
- Amazon DocumentDB 할당량

클러스터 연결 사용

클러스터에서 모든 연결을 사용하는 시나리오를 고려하세요. 예를 들어, `r5.2xlarge` 인스턴스에는 4,500개의 연결(및 450개의 열려 있는 커서) 제한이 있습니다. 3-인스턴스 Amazon DocumentDB 클러스터를 생성하고 클러스터 끝점을 사용하여 기본 인스턴스에만 연결하는 경우 열린 연결 및 커서에 대한 클러스터 제한은 각각 4,500 및 450입니다. 컨테이너에서 작성하는 많은 작업자를 사용하는 애플리케이션을 작성하는 경우 이러한 제한에 도달할 수 있습니다. 컨테이너는 한 번에 많은 연결을 열고 클러스터를 포화시킵니다.

대신 복제본 세트로 Amazon DocumentDB 클러스터에 연결하고 읽기를 복제본 인스턴스에 분배할 수 있습니다. 그런 다음 클러스터에서 사용 가능한 연결 및 커서 수를 각각 13,500 및 1,350으로 세 배로 늘릴 수 있습니다. 클러스터에 인스턴스를 더 추가하면 읽기 워크로드에 대한 연결 및 커서 수가 증가합니다. 클러스터에 쓰기 위해 연결 수를 늘려야 하는 경우 인스턴스 크기를 늘리는 것이 좋습니다.

> **Note**
>
> `large`, `xlarge` 및 `2xlarge` 인스턴스에 대한 연결 수는 인스턴스 크기가 4,500개가 될 때까지 증가합니다. `4xlarge` 인스턴스 이상의 경우 인스턴스당 최대 연결 수는 4,500개입니다. 인스턴스 유형별 제한에 대한 자세한 내용은 인스턴스 할당량 섹션을 참조하세요.

일반적으로 secondary 읽기 환경 설정을 사용하여 클러스터에 연결하지 않는 것이 좋습니다. 클러스터에 복제본 인스턴스가 없으면 읽기가 실패하기 때문입니다. 예를 들어, 기본 복제본과 복제본이 하나인 두 개의 인스턴스 Amazon DocumentDB 클러스터가 있다고 가정합니다. 복제본에 문제가 있으면 secondary로 설정된 연결 풀의 읽기 요청은 실패합니다. secondaryPreferred는 클라이언트가 연결하기에 적합한 복제본 인스턴스를 찾을 수 없는 경우 기본 인스턴스로 돌아가서 읽을 수 있다는 장점이 있습니다.

다중 연결 풀

일부 시나리오에서는 애플리케이션의 읽기가 쓰기 후 읽기 일관성을 가져야 하며, 이는 Amazon DocumentDB의 기본 인스턴스에서만 제공될 수 있습니다. 이러한 시나리오에서는 두 개의 클라이언트 연결 풀을 작성할 수 있습니다. 하나는 쓰기용이고 다른 하나는 쓰기 후 읽기 일관성이 필요한 읽기용입니다. 이를 위한 코드는 다음과 같을 것입니다.

```
## Create a MongoDB client,
##    open a connection to Amazon DocumentDB as a replica set and specify the
 readPreference as primary
clientPrimary = pymongo.MongoClient('mongodb://<user-
name>:<password>@mycluster.node.us-east-1.docdb.amazonaws.com:27017/?
replicaSet=rs0&readPreference=primary')

## Create a MongoDB client,
##    open a connection to Amazon DocumentDB as a replica set and specify the
 readPreference as secondaryPreferred
secondaryPreferred = pymongo.MongoClient('mongodb://<user-
name>:<password>@mycluster.node.us-east-1.docdb.amazonaws.com:27017/?
replicaSet=rs0&readPreference=secondaryPreferred')
```

다른 옵션은 단일 연결 풀을 작성하고 주어진 컬렉션에 대한 읽기 환경 설정을 덮어쓰는 것입니다.

```
##Specify the collection and set the read preference level for that collection
col = db.review.with_options(read_preference=ReadPreference.SECONDARY_PREFERRED)
```

요약

클러스터의 리소스를 더 잘 사용하려면 복제본 세트 모드를 사용하여 연결합니다. 애플리케이션에 적합한 경우 읽기를 복제본 인스턴스에 분배하여 애플리케이션 읽기를 확장할 수 있습니다.

Studio 3T에서 Amazon DocumentDB 클러스터에 연결

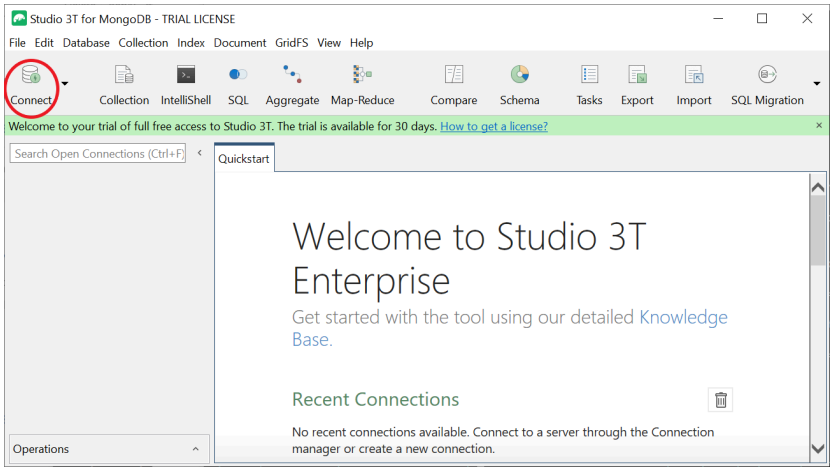
Studio 3T는 MongoDB를 사용하는 개발자와 데이터 엔지니어에게 널리 사용되는 GUI 및 IDE입니다. 데이터의 트리, 테이블 및 JSON 보기, CSV, JSON, SQL 및 BSON/Mongodump로 쉽게 가져오기/내보내기, 유연한 쿼리 옵션, 시각적 드래그 앤 드롭 UI, 자동 완성 기능이 있는 내장 몽고 쉘, 집계 파이프라인 편집기 및 SQL 쿼리 지원 등의 여러 가지 강력한 기능을 제공합니다.

사전 조건

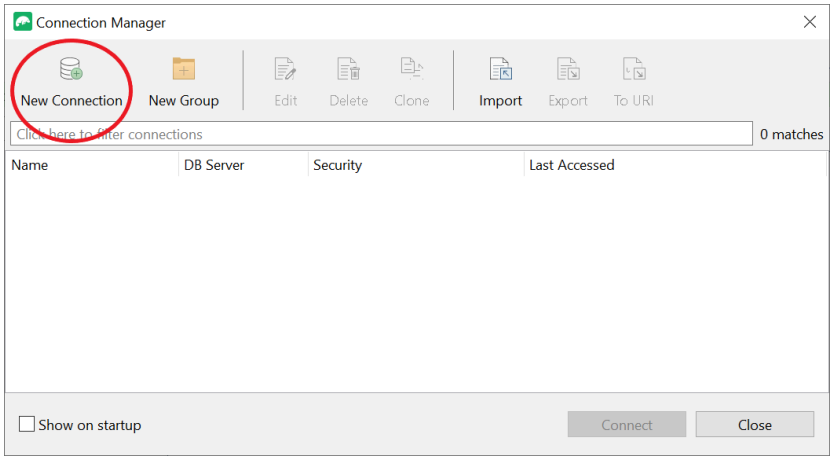
- Amazon EC2를 접속/점프 호스트로 사용하는 Amazon DocumentDB 클러스터가 아직 없는 경우 섹션을 참조하세요Amazon EC2를 사용하여 연결.
- Studio 3T가 없는 경우, 다운로드하여 설치합니다.

Studio 3T와 연결

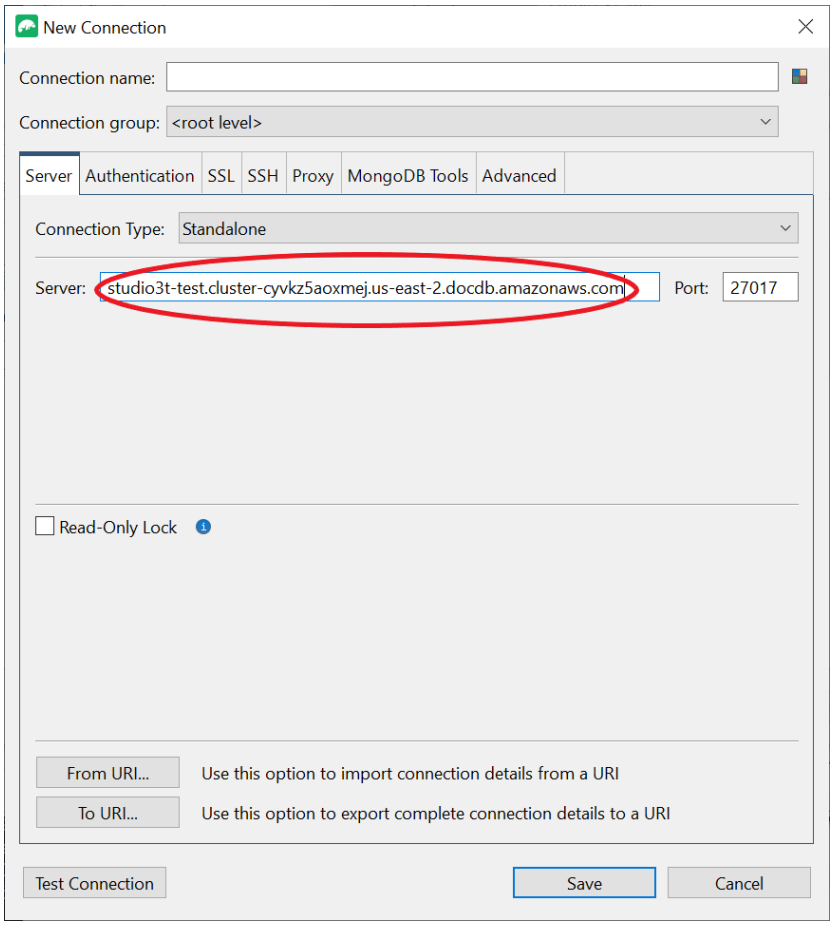
1. 도구 모음의 왼쪽 상단에서 연결을 선택합니다.



2. 도구 모음의 왼쪽 상단에서 새 연결을 선택합니다.

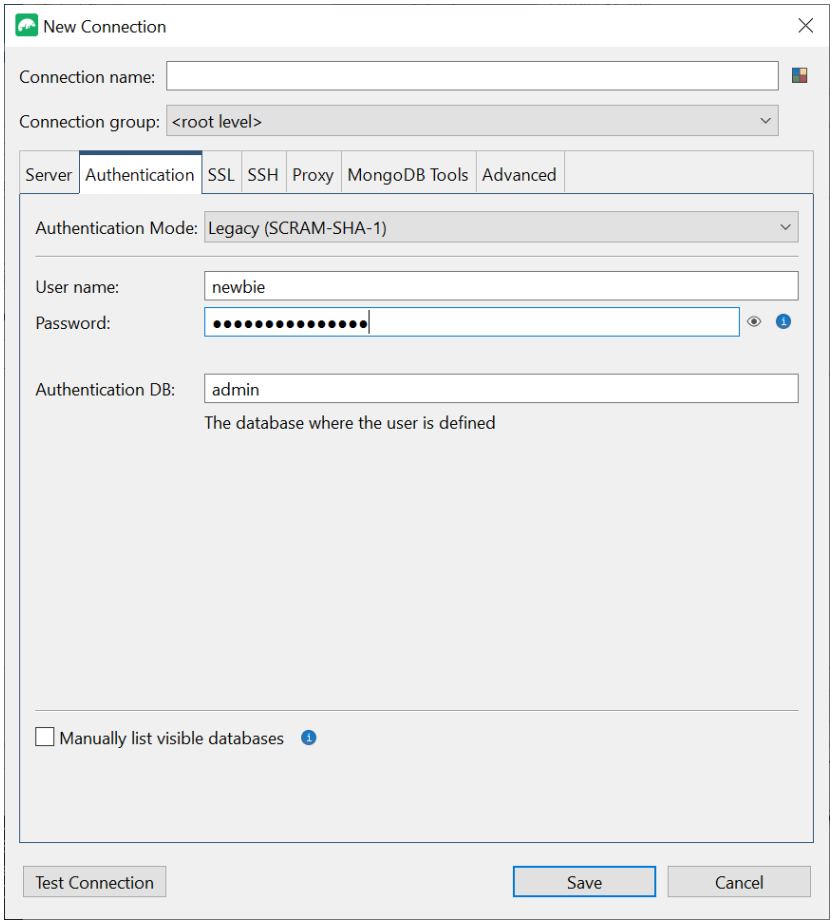


3. 서버 탭의 서버 필드에 클러스터 엔드포인트 정보를 입력합니다.

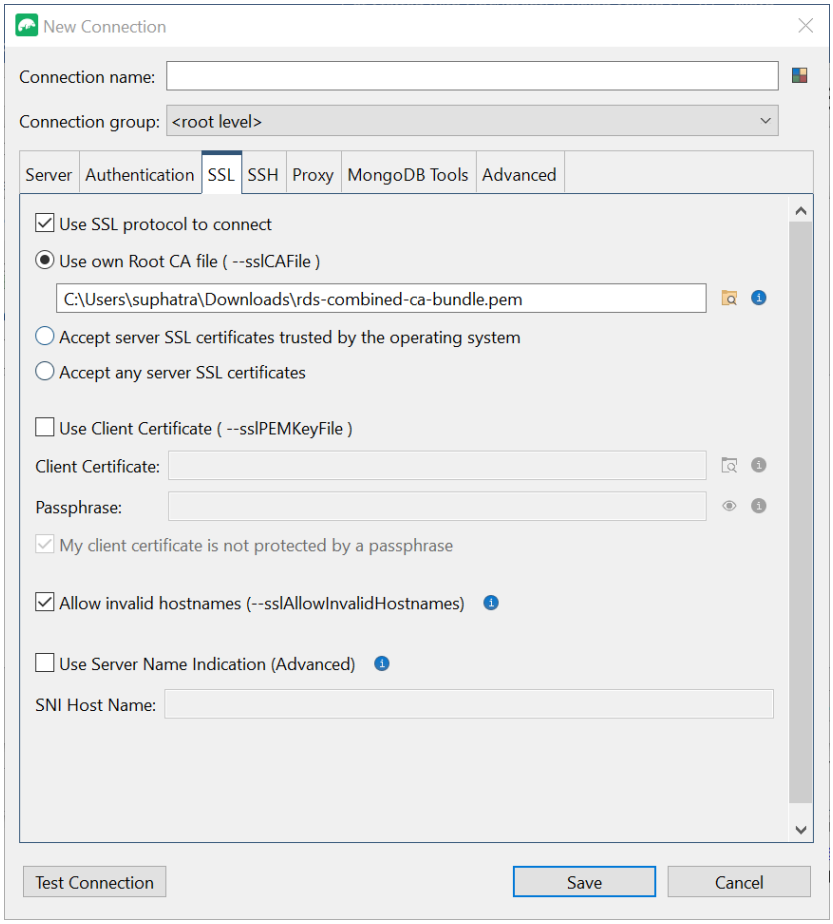


> **Note**
>
> 클러스터 엔드포인트를 찾으려면 섹션을 참조하세요인스턴스의 엔드포인트 찾기.

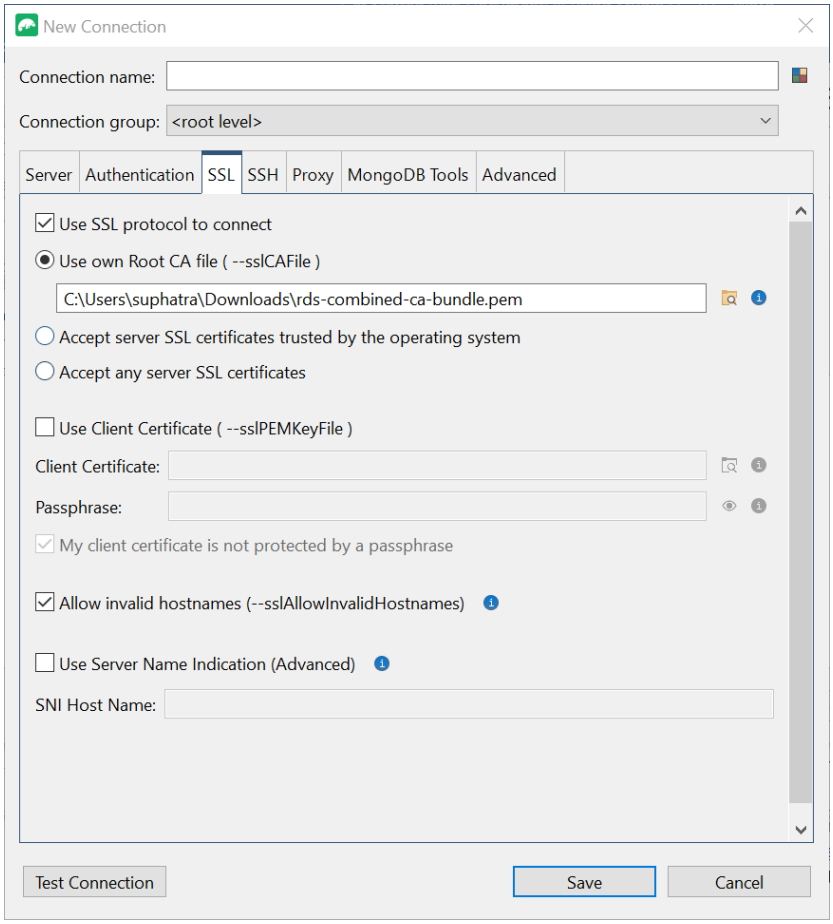
4. 인증 탭을 선택하고 인증 모드의 드롭다운 메뉴에서 레거시를 선택합니다.



5. 사용자 이름 및 암호 필드에 사용자 이름과 자격 증명을 입력합니다.
6. SSL 탭을 선택하고 SSL 프로토콜을 사용하여 연결 박스를 선택합니다.



7. 자체 루트 CA 파일 사용을 선택합니다. 그런 다음 Amazon DocumentDB 인증서를 추가합니다(DocumentDB 클러스터에서 SSL이 비활성화된 경우 이 단계를 건너뛸 수 있음). 잘못된 호스트 이름을 허용하려면 박스를 선택합니다.



> **Note**
>
> 인증서가 없으세요? 다음 명령을 사용하여 다운로드할 수 있습니다.
>
> ```
> wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
> ```

8. Amazon VPC 외부의 클라이언트 머신에서 연결하는 경우 SSH 터널을 생성해야 합니다. SSH 탭에서 이 작업을 수행합니다.

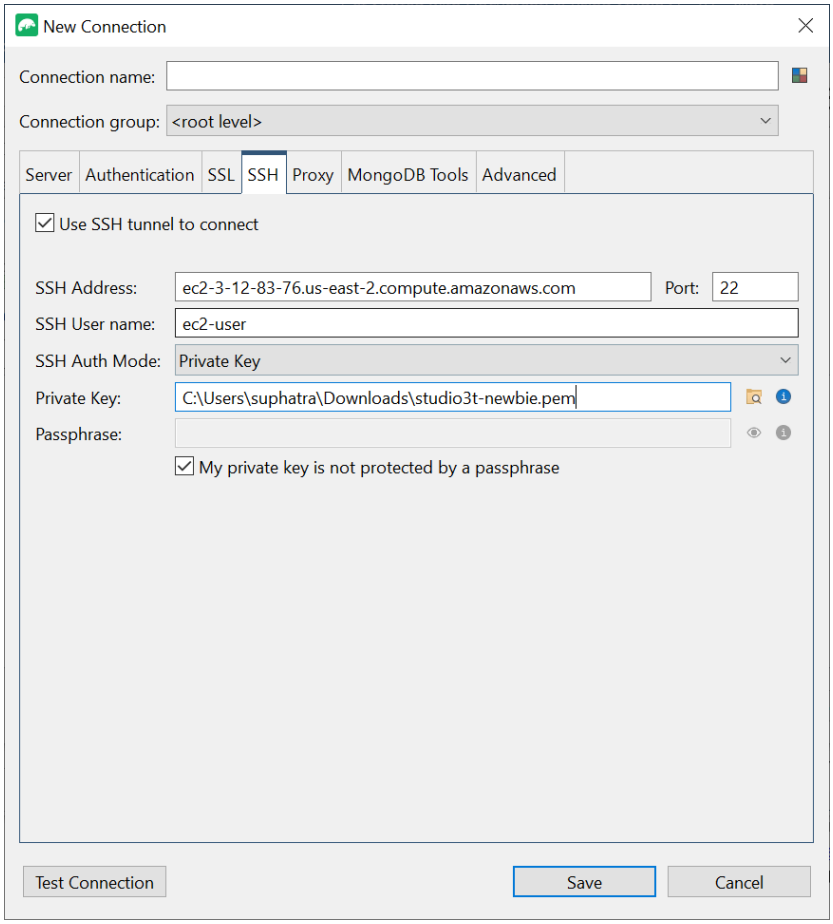
   a. SSH 터널 사용 박스를 선택하고 SSH 주소 필드에 SSH 주소를 입력합니다. 인스턴스 퍼블릭 DNS(IPV4)입니다. Amazon EC2 관리 콘솔에서 이 URL을 가져올 수 있습니다.

   b. 사용자 이름을 입력합니다. Amazon EC2 인스턴스의 사용자 이름입니다.

   c. SSH 인증 모드의 경우 프라이빗 키를 선택합니다. 프라이빗 키 필드에서 파일 찾기 아이콘을 선택하여 Amazon EC2 인스턴스의 프라이빗 키를 찾아 선택합니다. Amazon EC2 콘솔에서 인스턴스를 생성할 때 저장한 .pem 파일(키 페어)입니다.

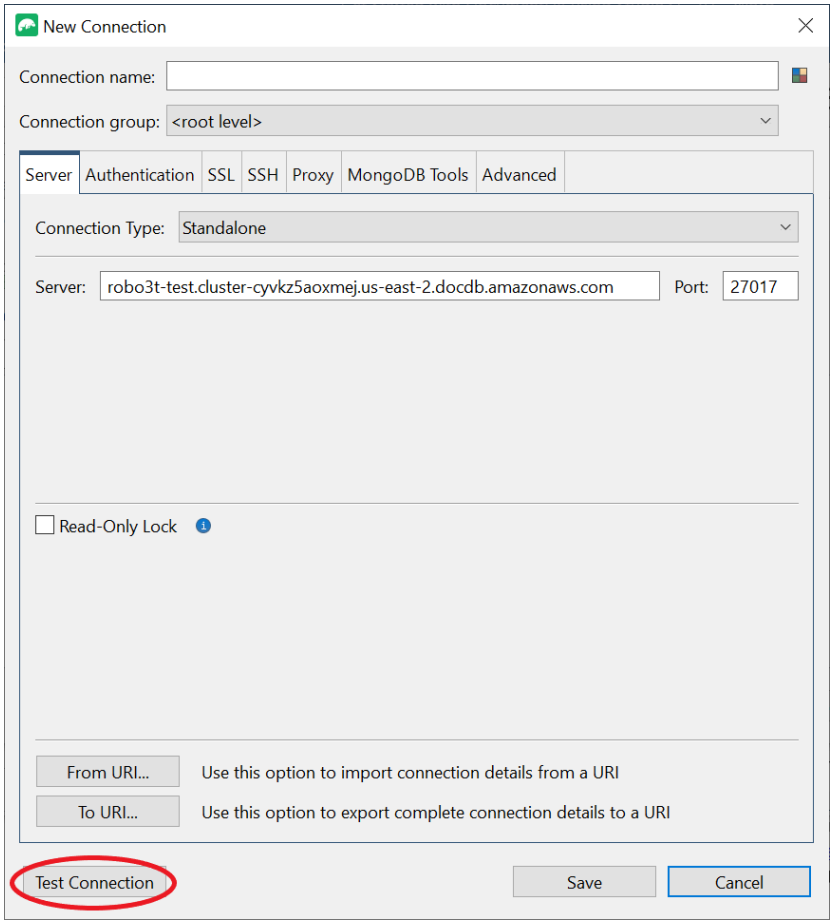
d. Linux/macOS 클라이언트 머신을 사용하는 경우 다음 명령을 사용하여 프라이빗 키의 권한을 변경해야 할 수 있습니다.

```
chmod 400 /fullPathToYourPemFile/<yourKey>.pem
```

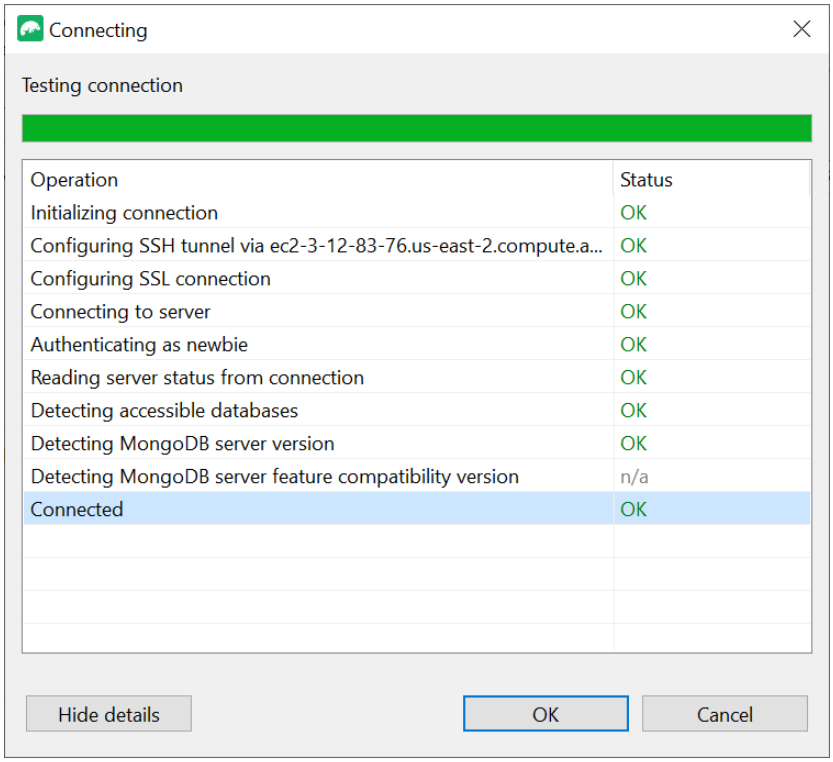


> **Note**
>
> Amazon EC2 인스턴스는 DocumentDB 클러스터와 동일한 Amazon VPC 및 보안 그룹에 있어야 합니다. Amazon EC2 관리 콘솔에서 SSH 주소, 사용자 이름 및 프라이빗 키를 가져올 수 있습니다.

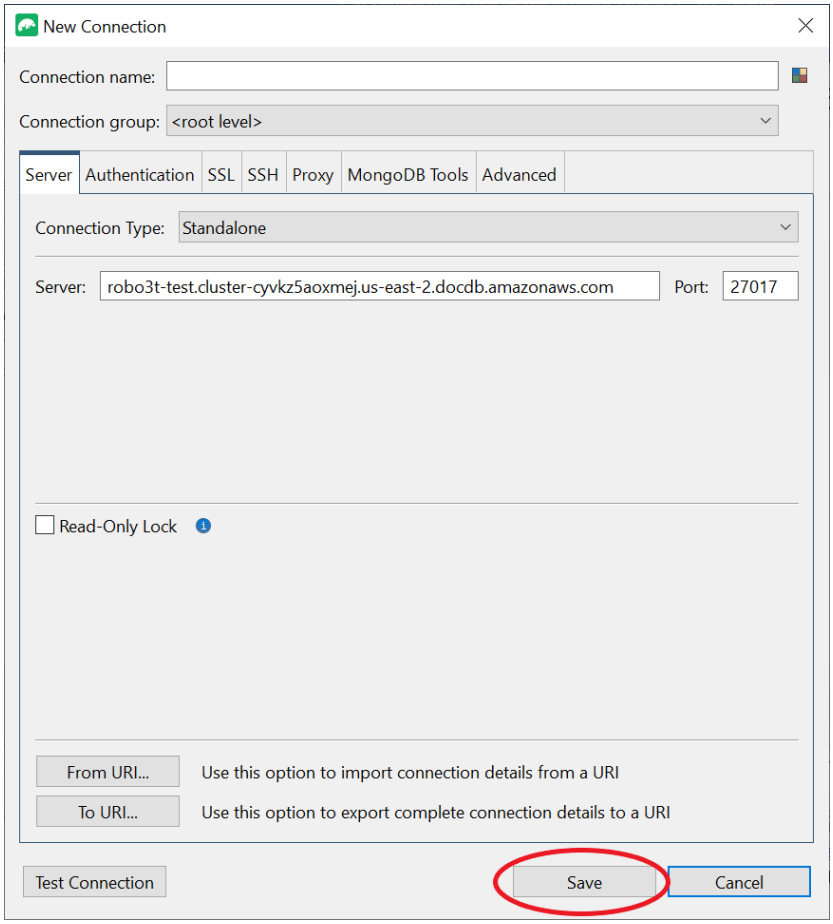
9. 이제 연결 테스트 버튼을 선택하여 구성을 테스트합니다.



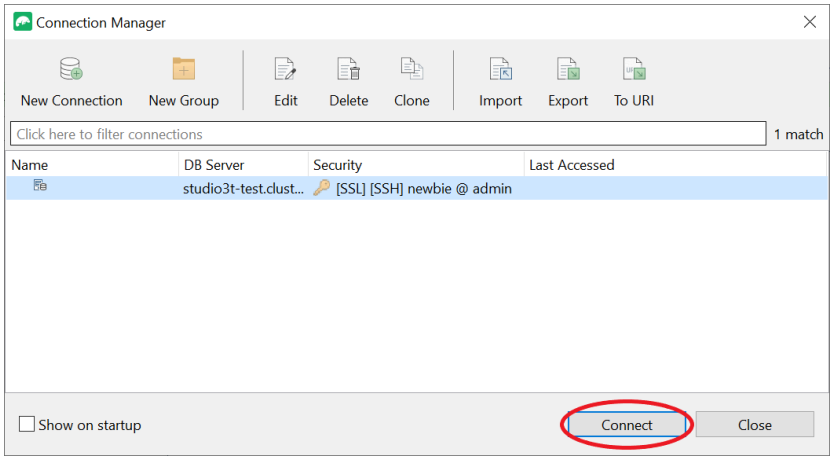
10. 진단 창에 녹색 막대가 로드되어 테스트가 성공했음을 나타냅니다. 이제 OK를 선택하여 진단 창을 닫습니다.



11. 나중에 사용할 수 있도록 연결을 저장하려면 저장을 선택합니다.



12. 이제 클러스터를 선택하고 연결을 선택합니다.



축하합니다! 이제 Studio 3T를 통해 Amazon DocumentDB 클러스터에 성공적으로 연결되었습니다.

DataGrip을 사용하여 Amazon DocumentDB에 연결

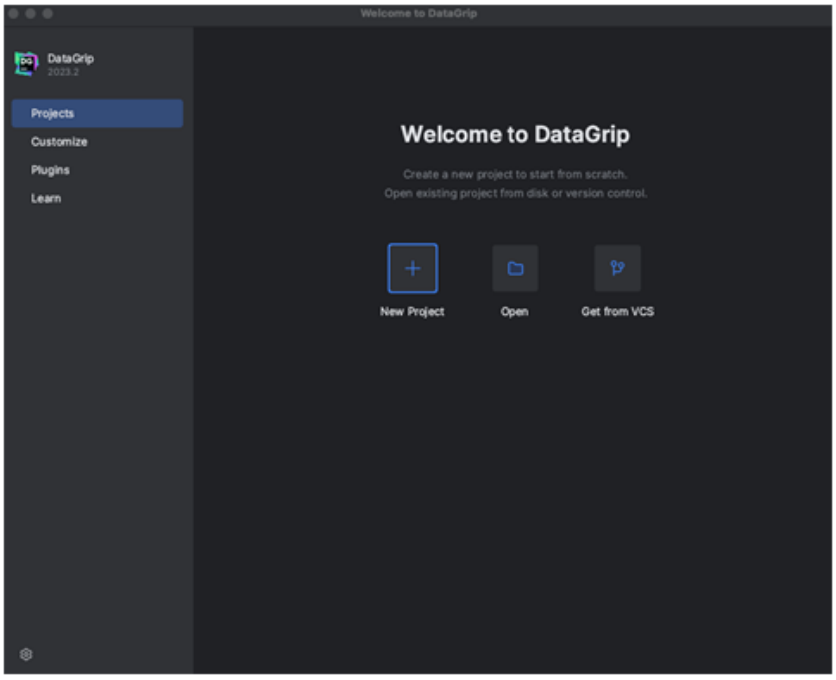
DataGrip은 Amazon DocumentDB를 비롯한 다양한 데이터베이스 시스템을 지원하는 강력한 통합 개발 환경(IDE)입니다. 이 섹션에서는 DataGrip을 사용하여 Amazon DocumentDB 클러스터에 연결하는 단계를 안내하며, 그래픽 인터페이스를 사용하여 데이터를 쉽게 관리하고 쿼리할 수 있습니다.

사전 조건

- 컴퓨터에 데이터그립 IDE가 설치되어 있습니다. JetBrains에서 다운로드할 수 있습니다.
- Amazon DocumentDB 클러스터와 동일한 VPC에서 실행 중인 Amazon EC2 인스턴스입니다. 이 인스턴스를 사용하여 로컬 시스템에서 Amazon DocumentDB 클러스터로 연결되는 보안 터널을 설정합니다. Amazon EC2를 사용하여 연결 방법을 알려면 다음 지침에 따르십시오.
- Amazon EC2 인스턴스, VPN 연결 또는 이미 보안 VPN을 사용하여 AWS 인프라에 액세스하는 경우의 대안입니다. 이 옵션을 선호하는 경우 지침에 따라를 사용하여 Amazon DocumentDB에 안전하게 액세스 AWS Client VPN합니다.

DataGrip을 사용한 연결

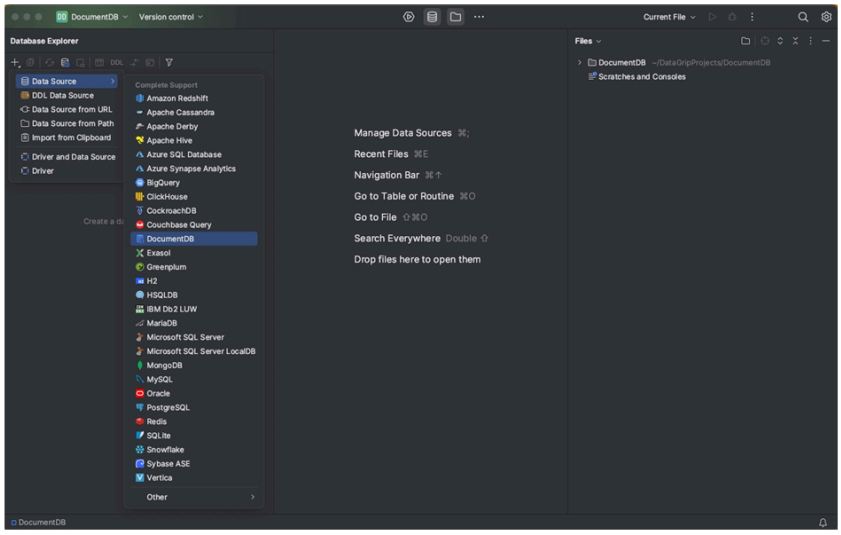
1. 컴퓨터에서 DataGrip을 시작하고 새 프로젝트를 생성합니다.



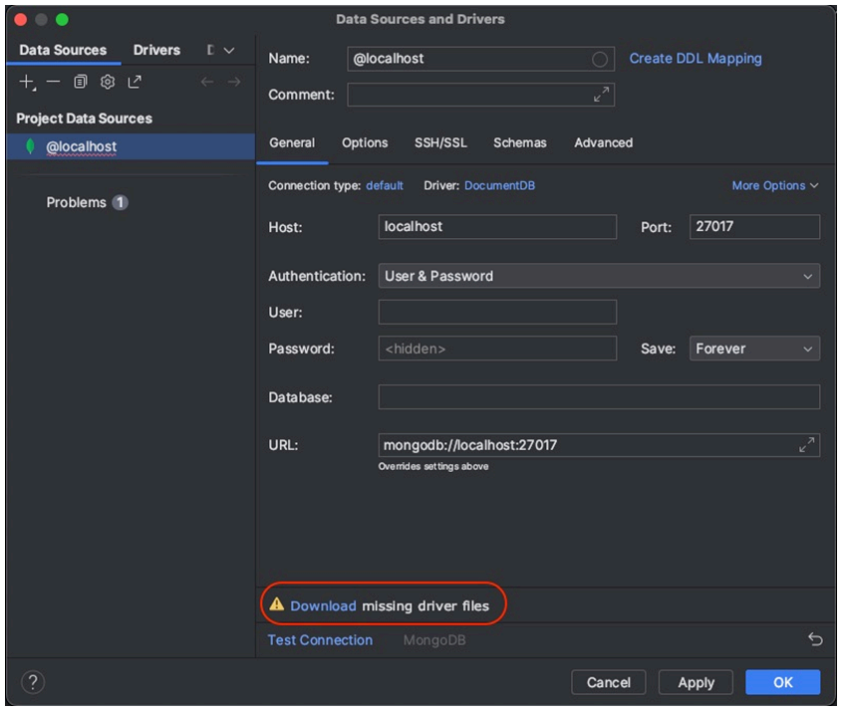
2. 다음 방법 중 하나로 새 데이터 원본을 추가합니다.

   a. 기본 메뉴에서 파일 — 새로 만들기 — 데이터 원본으로 이동한 다음 DocumentDB를 선택합니다.

b. Database Explorer의 도구 모음에서 새 아이콘(+)을 선택합니다. 데이터 소스로 이동하여 DocumentDB를 선택합니다.



3. 일반 탭의 데이터 소스 페이지에서 연결 설정 영역 하단에 누락된 드라이버 파일 다운로드 링크가 있는지 확인합니다. 데이터베이스와 상호 작용하는 데 필요한 드라이버를 다운로드하려면이 링크를 선택합니다. 직접 다운로드 링크는 JetBrains JDBC 드라이버를 참조하십시오.



4. 일반 탭에서 연결 세부 정보를 지정합니다.

a. 호스트 필드에 Amazon DocumentDB 클러스터 엔드포인트를 지정합니다.

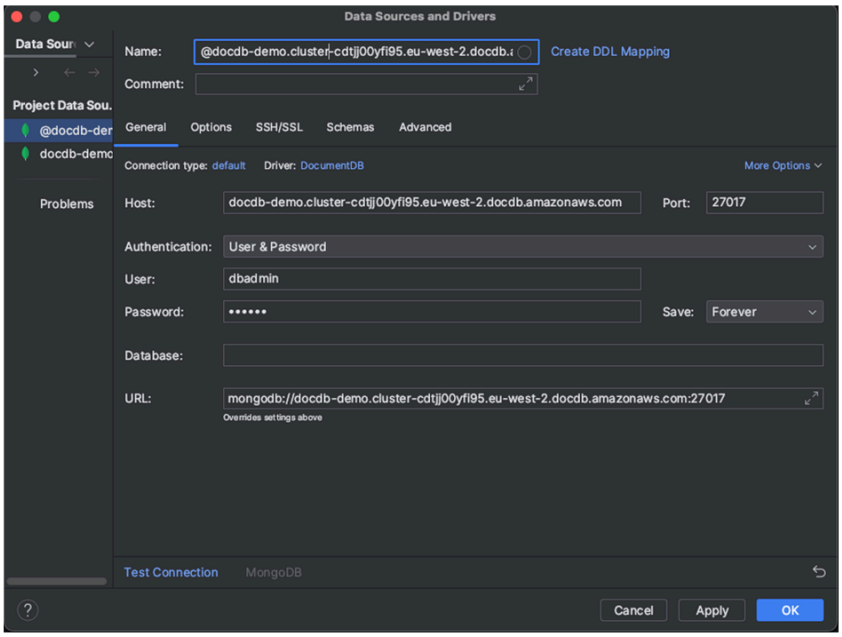
b. 포트는 이미 27017로 설정되어 있습니다. 클러스터가 다른 포트에 배포된 경우 변경하십시오.

c. 인증에서 사용자 및 암호를 선택합니다.

d. 사용자 이름 및 암호 정보를 입력하십시오.

e. 데이터베이스 필드는 선택 사항입니다. 연결할 데이터베이스를 지정할 수 있습니다.

f. 위의 세부 정보를 추가하면 URL 필드가 자동으로 완성됩니다.



5. SSH/SSL 탭에서 SSH 터널 사용을 활성화한 다음 아이콘을 선택하여 SSH 구성 대화 상자를 엽니다. 다음 정보를 입력합니다.

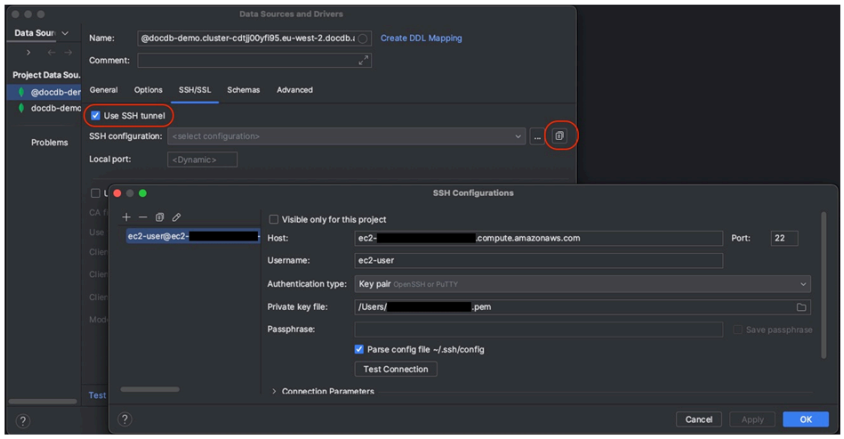
a. 호스트 필드에 Amazon EC2 인스턴스의 호스트 이름을 입력합니다.

b. Amazon EC2 인스턴스의 사용자 이름과 암호를 입력합니다.

c. 인증 유형에서 키 페어를 선택합니다.

d. 프라이빗 키 파일을 입력합니다.

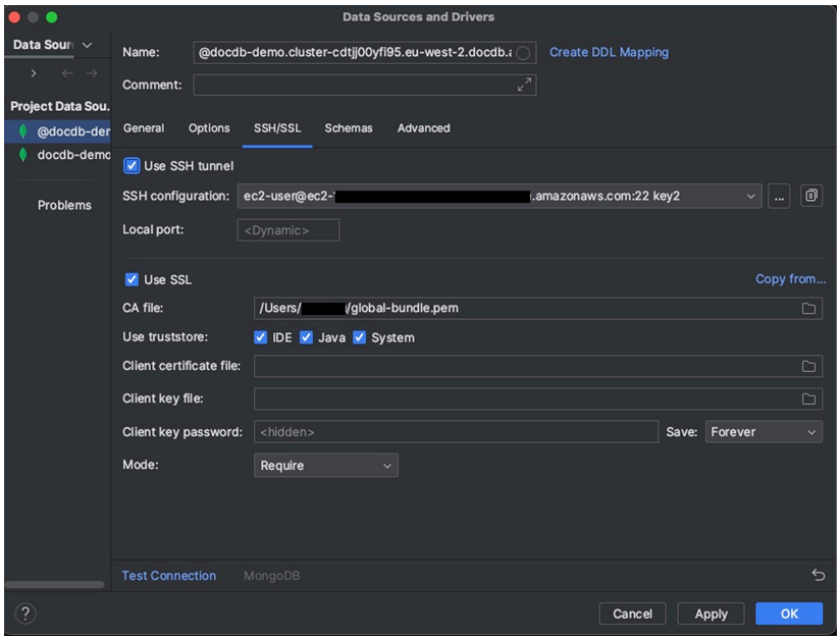
> **Note**
>
> VPN 옵션을 사용하는 경우 SSH 터널을 구성할 필요가 없습니다.



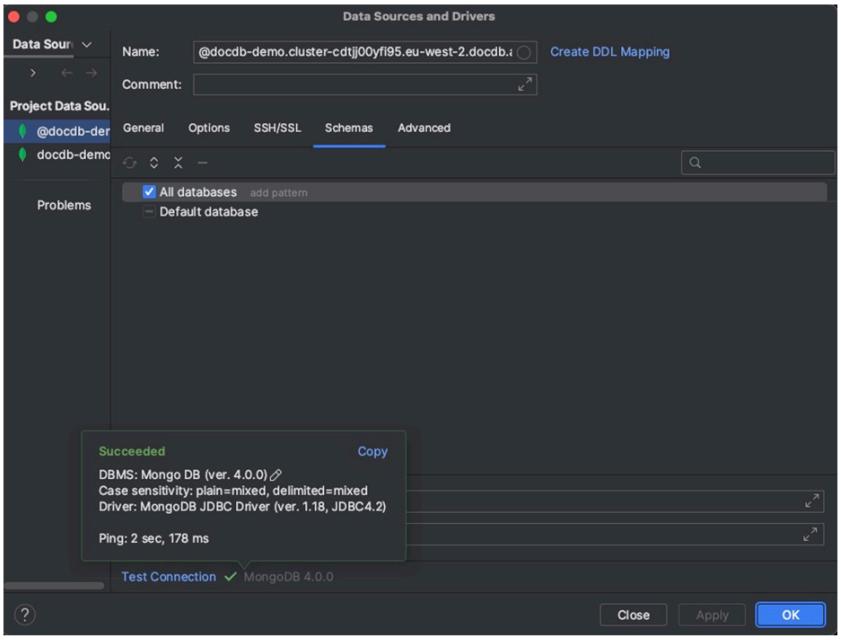
6. SSH/SSL 탭에서 SSL 사용을 활성화합니다. CA 파일 필드에 컴퓨터에 있는 `global-bundle.pem` 파일의 위치를 입력합니다. 모드의 경우 필수 옵션을 그대로 두십시오.

> **Note**
>
> 이 위치에서 또는 wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem 명령을 사용하여 인증서를 다운로드할 수 있습니다.

> **Note**
>
> Amazon DocumentDB 엘라스틱 클러스터에 연결하는 경우 CA 파일을 지정하지 않아도 됩니다. SSL 사용 옵션은 선택된 상태로 두고 다른 모든 옵션은 기본값으로 둡니다.



7. 스키마 탭에서 모든 데이터베이스를 선택하거나 스키마 패턴 필드에 “*: *” 필터를 입력합니다. 연결 테스트를 선택하여 연결을 테스트합니다.



8. 연결이 성공적으로 테스트되면 확인을 선택하여 데이터 소스 구성을 저장합니다.

데이터그립 기능

DataGrip은 Amazon DocumentDB를 효율적으로 사용하는 데 도움이 되는 다양한 기능을 제공합니다.

- SQL 편집기 — DataGrip의 SQL 편집기를 사용하여 DocumentDB 컬렉션에서 SQL과 유사한 쿼리를 작성하고 실행할 수 있습니다.
- 비주얼 쿼리 빌더 - 비주얼 쿼리 빌더를 사용하면 SQL 코드를 작성하지 않고도 그래픽 방식으로 쿼리를 만들 수 있습니다.
- 스키마 관리 — 컬렉션 생성, 변경, 삭제 등 데이터베이스 스키마를 쉽게 관리할 수 있습니다.
- 데이터 시각화 — DataGrip에서 제공되는 다양한 시각화 도구를 사용하여 데이터를 보고 분석할 수 있습니다.
- 데이터 내보내기 및 가져오기 — DataGrip의 내보내기 및 가져오기 기능을 사용하여 Amazon DocumentDB와 다른 데이터베이스 간에 데이터를 전송합니다.

Amazon DocumentDB 및 기타 데이터베이스 시스템 사용에 대한 고급 기능 및 팁은 공식 DataGrip 설명서를 참조하십시오.

Amazon EC2를 사용하여 연결

이 섹션에서는 Amazon DocumentDB 클러스터와 Amazon EC2 간의 연결을 설정하고 Amazon EC2 인스턴스에서 Amazon DocumentDB 클러스터에 액세스하는 방법을 설명합니다.

EC2 연결을 구성하기 위한 두 가지 옵션이 있습니다.

- EC2 인스턴스를 Amazon DocumentDB 데이터베이스에 자동 연결 - EC2 콘솔의 자동 연결 기능을 사용하여 EC2 인스턴스와 새 또는 기존 Amazon DocumentDB 데이터베이스 간의 연결을 자동으로 구성합니다. 이 연결을 통해 트래픽이 EC2 인스턴스와 Amazon DocumentDB 데이터베이스 간에 이동할 수 있습니다. 이 옵션은 일반적으로 새 보안 그룹을 테스트하고 생성하는 데 사용됩니다.
- EC2 인스턴스를 Amazon DocumentDB 데이터베이스에 수동으로 연결 - 자동 연결 기능으로 생성된 구성을 재현하도록 보안 그룹을 수동으로 구성하고 할당하여 EC2 인스턴스와 Amazon DocumentDB 데이터베이스 간의 연결을 구성합니다. 이 옵션은 일반적으로 고급 설정을 변경하고 기존 보안 그룹을 사용하는 데 사용됩니다.

사전 조건

옵션과 관계 없이 첫 번째 Amazon DocumentDB 클러스터를 작성하기 전에 다음 작업을 수행해야 합니다:

Amazon Web Services(AWS) 계정 만들기

Amazon DocumentDB 사용을 시작하기 전에 Amazon Web Services(AWS) 계정이 있어야 합니다. AWS 계정은 무료입니다. 사용하는 서비스 및 리소스에 대해서만 비용을 지불하는 것입니다.

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://signin.aws.amazon.com/signup?request_type=register를 엽니다.
2. 온라인 지시 사항을 따르세요.

   등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

선택적으로 필요한 AWS Identity and Access Management (IAM) 권한을 설정합니다.

클러스터, 인스턴스 및 클러스터 파라미터 그룹과 같은 Amazon DocumentDB 리소스를 관리하기 위한 액세스에는가 요청을 인증하는 데 사용할 AWS 수 있는 자격 증명이 필요합니다. 자세한 내용은 Amazon DocumentDB의 ID 및 액세스 관리 섹션을 참조하세요.

1. 의 검색 창에 나타나는 드롭다운 메뉴에서 IAM을 AWS Management Console**IAM** 입력하고 선택합니다.
2. IAM 콘솔에 들어가면 탐색 창에서 사용자를 선택합니다.
3. 사용자 이름을 선택합니다.
4. 권한 추가를 선택합니다.
5. 기존 정책 직접 연결을 선택합니다.
6. 검색 창에 `AmazonDocDBFullAccess`을 입력하고 검색 결과에 표시되면 선택합니다.
7. 다음: 검토를 선택합니다.
8. 권한 추가를 선택합니다.

Amazon Virtual Private Cloud (Amazon VPC) 생성

AWS 리전 현재 위치에 따라 기본 VPC가 이미 생성되어 있을 수도 있고 그렇지 않을 수도 있습니다. 기본 VPC가 없는 경우, Amazon VPC 사용 설명서의 Amazon VPC 시작하기에서 1단계를 완료하세요. 이 작업에는 5분 미만이 소요됩니다.

Amazon EC2 자동 연결

주제

- EC2 인스턴스를 새 Amazon DocumentDB 데이터베이스에 자동으로 연결
- EC2 인스턴스를 기존 Amazon DocumentDB 데이터베이스에 자동 연결
- EC2 인스턴스와의 자동 연결 개요
- 연결된 컴퓨팅 리소스 보기

EC2 인스턴스와 새 Amazon DocumentDB 데이터베이스 간의 연결을 설정하기 전에 EC2 인스턴스와의 자동 연결 개요에 설명된 요구 사항을 충족하는지 확인하세요. 연결을 구성한 후에 보안 그룹을 변경할 경우 변경 사항이 EC2 인스턴스와Amazon DocumentDB 데이터베이스 간의 연결에 영향을 미칠 수 있습니다.

> **Note**
>
> AWS Management Console을 사용해야만 EC2 인스턴스와 Amazon DocumentDB 데이터베이스 간의 연결을 자동으로 설정할 수 있습니다. AWS CLI 또는 Amazon DocumentDB API를 사용하여 연결을 자동으로 설정할 수 없습니다.

EC2 인스턴스를 새 Amazon DocumentDB 데이터베이스에 자동으로 연결

다음 프로세스에서는 사전 조건 주제의 단계를 완료했다고 가정합니다.

단계(Steps)

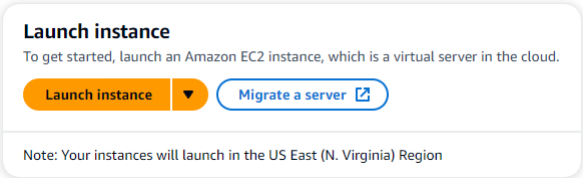
- 1단계: Amazon EC2 인스턴스 생성
- 2단계: Amazon DocumentDB 클러스터 생성
- 3단계: Amazon EC2 인스턴스 구성
- 4단계: MongoDB 쉘 설치
- 5단계: Amazon DocumentDB TLS 관리

- 6단계: Amazon DocumentDB 클러스터에 연결
- 7단계: 데이터 삽입 및 쿼리
- 8단계: 살펴보기

1단계: Amazon EC2 인스턴스 생성

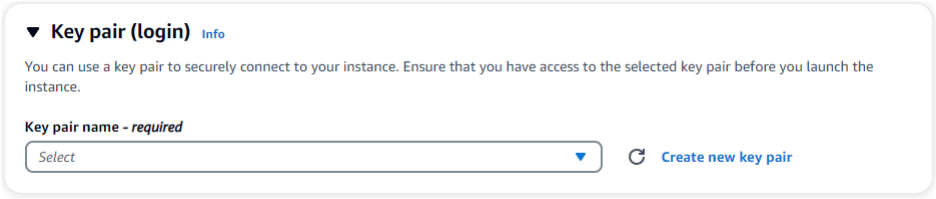
이 단계에서는 나중에 Amazon DocumentDB 클러스터를 프로비저닝하는 데 사용할 동일한 지역 및 Amazon VPC에 Amazon EC2 인스턴스를 생성합니다.

1. Amazon EC2 콘솔 대시보드에서 인스턴스 시작을 선택합니다.



2. 이름 및 태그 섹션에 있는 이름 필드에 이름 또는 식별자를 입력합니다.
3. Amazon Machine Image(AMI) 드롭다운 목록에서 Amazon Linux 2 AMI를 찾아 선택합니다.

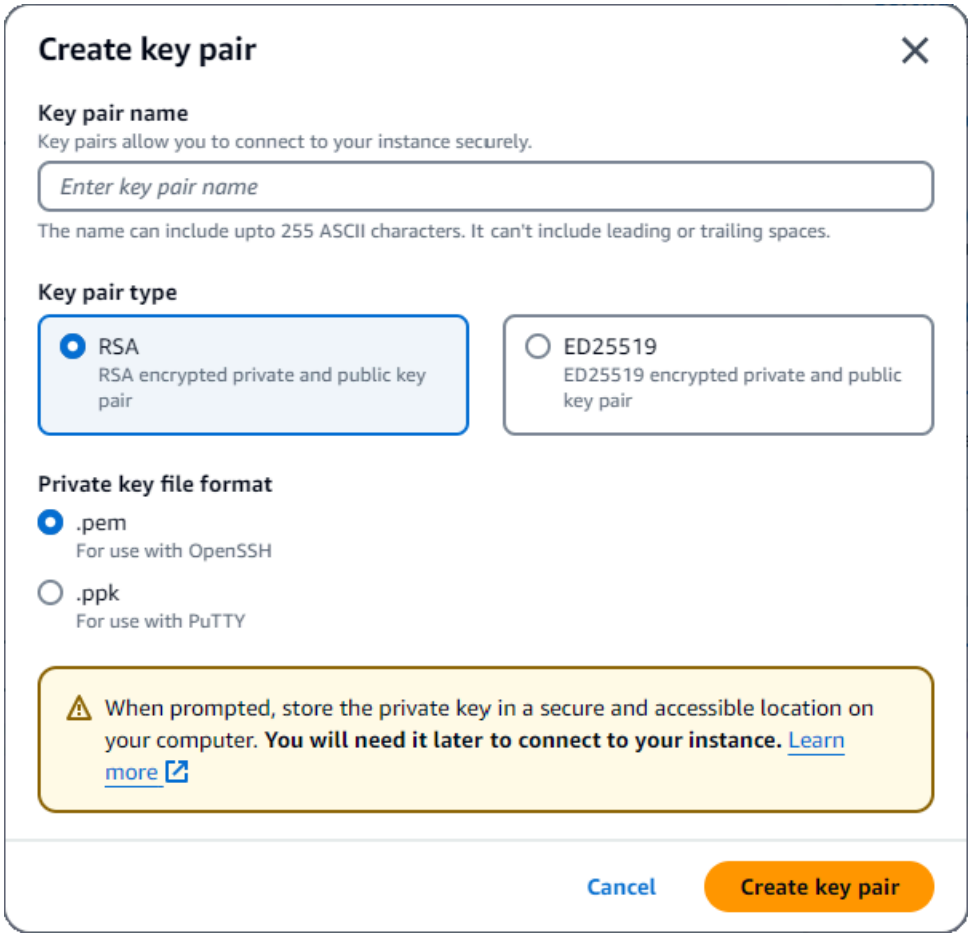
4. 인스턴스 유형 드롭다운 목록에서 t3.micro를 찾아 선택합니다.
5. 키 페어(로그인) 섹션에서 기존 키 페어의 식별자를 입력하거나 새 키 페어 생성을 선택합니다.



Amazon EC2 키 페어를 제공해야 합니다.

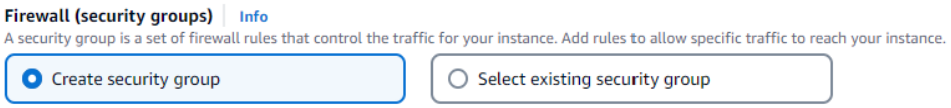
- Amazon EC2 키 페어가 있는 경우:
  1. 키 페어를 선택하고 목록에서 키 페어를 선택합니다.
  2. Amazon EC2 인스턴스에 로그인하려면 프라이빗 키 파일(.pem 또는 .ppk 파일)이 있어야 합니다.
- Amazon EC2 키 페어가 없는 경우:
  1. 새 키 페어 생성을 선택하면 키 페어 생성 대화 상자가 나타납니다.
  2. 키 페어 명칭 필드에 명칭을 입력합니다.

3. 키 페어 유형 및 프라이빗 키 파일 형식을 선택합니다.
4. 키 페어 생성(Create key pair)를 선택합니다.



> **Note**
>
> 보안을 위해 SSH 및 EC2 인스턴스에 대한 인터넷 연결 모두에 키 페어를 사용하는 것이 좋습니다.

6. 선택 사항: 네트워크 설정 섹션의 방화벽(보안 그룹)에서 보안 그룹 생성을 선택합니다.



보안 그룹 생성을 선택합니다(EC2 연결에 적용되는 모든 트래픽 허용 규칙 확인).

> **Note**
>
> 기존 보안 그룹을 사용하려면 Amazon EC2 수동 연결의 지침을 따르세요.

7. 요약 섹션에서 EC2 구성을 검토하고 올바른 경우 인스턴스 시작을 선택합니다.

2단계: Amazon DocumentDB 클러스터 생성

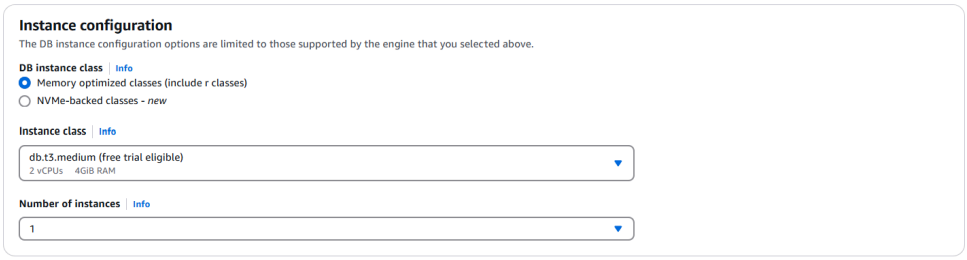
Amazon EC2 인스턴스가 프로비저닝되는 동안 Amazon DocumentDB 클러스터를 생성합니다.

1. Amazon DocumentDB 콘솔로 이동한 다음 탐색 창에서 클러스터를 선택합니다.
2. 생성을 선택합니다.
3. 클러스터 유형 설정을 인스턴스 기반 클러스터 의 기본값으로 둡니다.
4. 클러스터 구성에서 클러스터 식별자에 고유한 이름을 입력합니다. 콘솔은 모든 클러스터 이름을 입력 방식에 관계없이 소문자로 변경합니다.

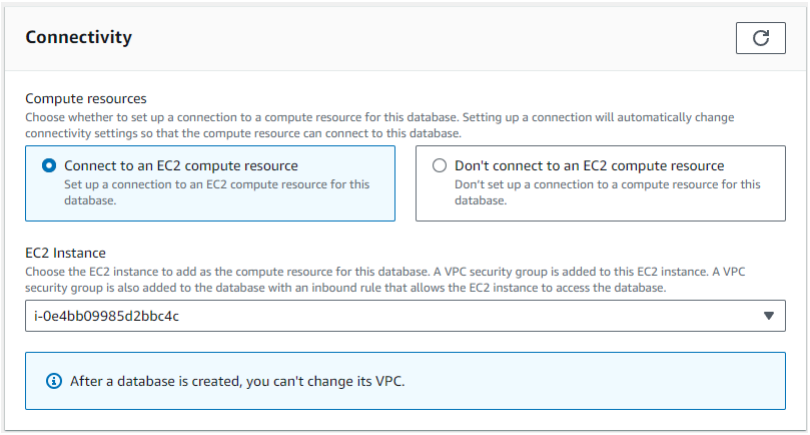
   엔진 버전을 기본값인 5.0.0으로 그대로 유지합니다.
5. 클러스터 스토리지 구성의 경우 Amazon DocumentDB Standard의 기본 설정을 그대로 유지합니다.
6. 인스턴스 구성에서 다음을 수행합니다.

   - DB 인스턴스 클래스에서 메모리 최적화 클래스(r 클래스 포함)(기본값)를 선택합니다.

     다른 인스턴스 옵션은 NVMe 지원 클래스입니다. 자세한 내용은 NVMe 지원 인스턴스 섹션을 참조하세요.
   - 인스턴스 클래스에서 요구 사항에 맞는 인스턴스 유형을 선택합니다. 인스턴스 클래스에 대한 자세한 설명은 인스턴스 클래스 사양 섹션을 참조하세요.
   - 인스턴스 수에서 요구 사항을 가장 잘 반영하는 숫자를 선택합니다. 숫자가 낮을수록 비용이 낮아지고 클러스터에서 관리할 수 있는 읽기/쓰기 볼륨이 낮아집니다.



7. 연결성에서 EC2 컴퓨팅 리소스에 연결을 선택합니다. 이것이 1단계에서 생성한 EC2 인스턴스입니다.



> **Note**
>
> EC2 컴퓨팅 리소스에 연결하면 Amazon DocumentDB 클러스터에 대한 EC2 컴퓨팅 리소스 연결을 위한 보안 그룹이 자동으로 생성됩니다. 클러스터 생성을 완료했고 새로 생성된 보안 그룹을 보려면 클러스터 목록으로 이동하여 클러스터의 식별자를 선택합니다. 연결 및 보안 탭에서 보안 그룹으로 이동하여 보안 그룹 이름(ID)에서 그룹을 찾습니다. 클러스터의 이름이 포함되며 `docdb-ec2-docdb-2023-12-11-21-33-41:i-0e4bb09985d2bbc4c (sg-0238e0b0bf0f73877)` 항목과 비슷합니다.

8. 인증 섹션에 기본 사용자의 사용자 이름을 입력한 후 자체 관리형을 선택합니다. 암호를 입력한 다음 확인합니다.

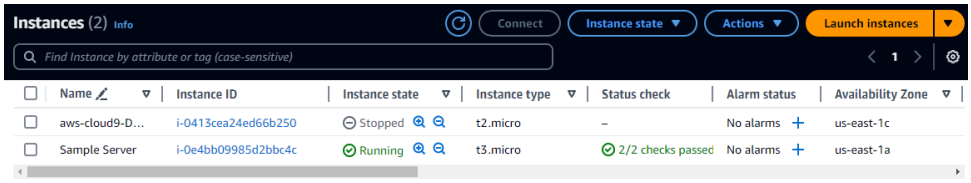
   대신 관리형을 AWS Secrets Manager 선택한 경우 자세한 내용은 Amazon DocumentDB 및를 사용한 암호 관리 AWS Secrets Manager 섹션을 참조하세요.

9. 클러스터 생성을 선택합니다.

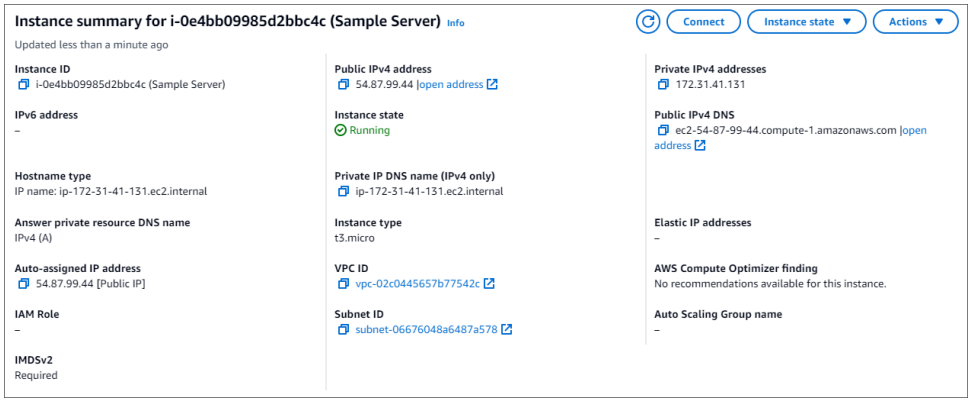
3단계: Amazon EC2 인스턴스 구성

mongo 쉘을 설치하려면 먼저 Amazon EC2 인스턴스에 연결해야 합니다. mongo 쉘을 설치하면 Amazon DocumentDB 클러스터에 연결하고 쿼리할 수 있습니다. 다음 단계를 완료합니다.

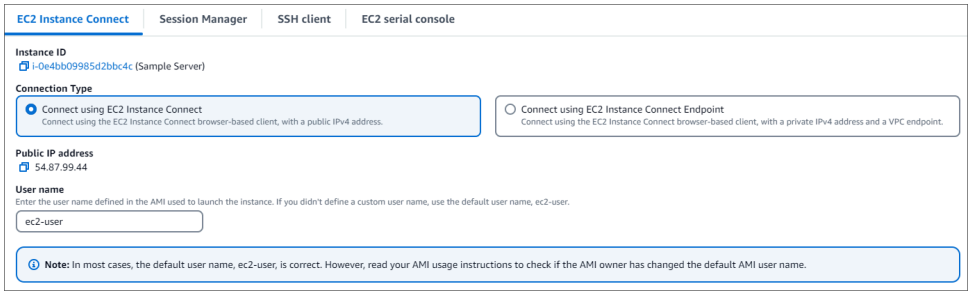
1. Amazon EC2 콘솔에서 인스턴스로 이동하여 방금 생성한 인스턴스가 실행 중인지 확인합니다. 그렇다면 인스턴스 ID를 클릭하여 인스턴스를 선택합니다.



2. 연결을 선택합니다.



3. 연결 방법에는 Amazon EC2 Instance Connect, 세션 관리자, SSH 클라이언트 또는 EC2 직렬 콘솔의 4가지 탭 옵션이 표시됩니다. 하나를 선택하고 지침을 따라야 합니다. 완료되면 연결을 선택합니다.



> **Note**
>
> 이 안내를 시작한 후 IP 주소가 변경되었거나 나중에 해당 환경으로 돌아오려는 경우 새 API 주소로부터의 인바운드 트래픽을 활성화하도록 demoEC2 보안 그룹 인바운드 규칙을 업데이트해야 합니다.

4단계: MongoDB 쉘 설치

이제 Amazon DocumentDB 클러스터에 연결하고 쿼리하는 데 사용하는 명령줄 유틸리티인 MongoDB 쉘을 설치할 수 있습니다. 현재 MongoDB 쉘에는 최신 버전인 mongosh와 이전 버전인 Mongo 쉘의 두 가지 버전이 있습니다.

> **Important**
>
> Amazon DocumentDB에 대한 IAM ID 인증에서 지원되지 않는 Node.js 드라이버(6.13.1 이전 버전)에는 알려진 제한 사항이 있습니다. Node.js 드라이버를 사용하는 Node.js 드라이버 및 도구(예: mongosh)는 Node.js 드라이버 버전 6.13.1 이상을 사용하도록 업그레이드해야 합니다.

아래 지침에 따라 사용 중인 운영 체제에 맞는 MongoDB 쉘을 설치합니다.

On Amazon Linux

Amazon Linux에 MongoDB 쉘 설치

IAM 인증을 사용하지 않고 최신 MongoDB 쉘(mongosh)을 사용하여 Amazon DocumentDB 클러스터에 연결하려면 다음 단계를 따르세요.

1. 리포지토리 파일을 생성합니다. 생성한 EC2 인스턴스의 명령줄에서 다음 명령을 실행합니다.

```
echo -e "[mongodb-org-5.0] \nname=MongoDB Repository\nbaseurl=https://
repo.mongodb.org/yum/amazon/2023/mongodb-org/5.0/x86_64/\ngpgcheck=1 \nenabled=1
 \ngpgkey=https://pgp.mongodb.com/server-5.0.asc" | sudo tee /etc/yum.repos.d/
mongodb-org-5.0.repo
```

2. 완료되면 명령 프롬프트에서 다음 두 명령 옵션 중 하나를 사용하여 mongosh를 설치합니다.

   옵션 1 - Amazon EC2 구성 중에 기본 Amazon Linux 2023을 선택한 경우 다음 명령을 입력합니다.

```
sudo yum install -y mongodb-mongosh-shared-openssl3
```

   옵션 2 - Amazon EC2 구성 중에 Amazon Linux 2를 선택한 경우 다음 명령을 입력합니다.

```
sudo yum install -y mongodb-mongosh
```

IAM 인증을 사용하는 경우 이전 버전의 MongoDB 쉘(5.0)을 사용하여 Amazon DocumentDB 클러스터에 연결해야 합니다. 다음 단계를 따르세요.

1. 리포지토리 파일을 생성합니다. 생성한 EC2 인스턴스의 명령줄에서 다음 명령을 실행합니다.

```
echo -e "[mongodb-org-5.0] \nname=MongoDB Repository\nbaseurl=https://
repo.mongodb.org/yum/amazon/2023/mongodb-org/5.0/x86_64/\ngpgcheck=1 \nenabled=1
 \ngpgkey=https://pgp.mongodb.com/server-5.0.asc" | sudo tee /etc/yum.repos.d/
mongodb-org-5.0.repo
```

2. 완료되면 명령 프롬프트에서 다음 명령 옵션을 사용하여 mongodb 5.0 쉘을 설치합니다.

```
sudo yum install -y mongodb-org-shell
```

On Ubuntu

Ubuntu에 mongosh 설치

1. 패키지 관리 시스템에서 사용할 퍼블릭 키를 가져옵니다.

```
curl -fsSL https://pgp.mongodb.com/server-5.0.asc | sudo gpg --dearmor -o /usr/
share/keyrings/mongodb-server-5.0.gpg
```

2. Ubuntu 버전에 맞는 명령을 사용하여 MongoDB의 목록 파일 `mongodb-org-5.0.list`를 생성합니다.

```
echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-
server-5.0.gpg ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/5.0
 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-5.0.list
```

3. 다음 명령을 사용하여 로컬 패키지 데이터베이스를 가져오고 업데이트합니다.

```
sudo apt-get update
```

4. mongosh를 설치합니다.

```
sudo apt-get install -y mongodb-mongosh
```

Ubuntu 시스템에서 이전 버전의 MongoDB를 설치하는 방법에 대한 자세한 내용은 Install MongoDB Community Edition on Ubuntu를 참조하세요.

On other operating systems

다른 운영 체제에서 mongo 쉘을 설치하려면 MongoDB 설명서의 MongoDB Community Edition 설치를 참조하세요.

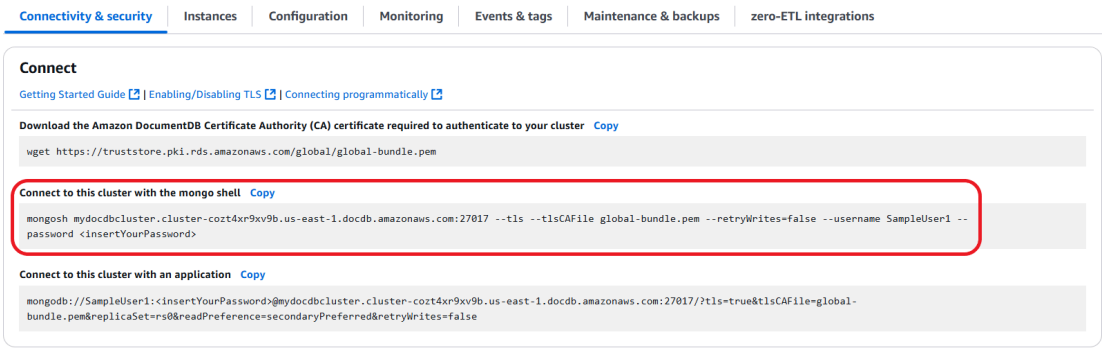
5단계: Amazon DocumentDB TLS 관리

다음 코드를 사용하여 Amazon DocumentDB용 CA 인증서를 다운로드하세요. `wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem`

> **Note**
>
> 전송 계층 보안 (TLS) 은 모든 새 Amazon DocumentDB 클러스터에 기본적으로 활성화됩니다. 자세한 내용은 전송 중 데이터 암호화 섹션을 참조하세요.

6단계: Amazon DocumentDB 클러스터에 연결

1. Amazon DocumentDB 콘솔의 클러스터에서 클러스터를 찾습니다. 해당 클러스터의 클러스터 식별자를 클릭하여 생성한 클러스터를 선택합니다.
2. 연결 및 보안 탭에서 연결 상자의 mongo 쉘을 사용하여 이 클러스터에 연결을 찾습니다.



제공된 연결 문자열을 복사하여 터미널에 붙여넣습니다.

해당 항목을 다음과 같이 변경합니다.

a. 문자열에 올바른 사용자 이름이 있는지 확인합니다.
b. 그런 다음 연결할 때 mongo 쉘에서 비밀번호를 입력하라는 메시지가 표시되도록 `<insertYourPassword>` 항목을 생략합니다.
c. 선택 사항: IAM 인증을 사용하거나 이전 버전의 MongoDB 쉘을 사용하는 경우 다음과 같이 연결 문자열을 수정합니다.

```
mongo --ssl --host mydocdbcluster.cluster-cozt4xr9xv9b.us-
east-1.docdb.amazonaws.com:27017 --sslCAFile global-bundle.pem --
username SampleUser1 --password
```

`mydocdbcluster.cluster-cozt4xr9xv9b.us-east-1`을 클러스터의 동일한 정보로 바꿉니다.

3. 이제 터미널에서 Enter를 누릅니다. 이제 비밀번호를 입력하라는 메시지가 표시됩니다. 암호를 입력합니다.
4. 암호를 입력하고 `rs0 [direct: primary] <env-name>>` 메시지가 표시되면 Amazon DocumentDB 클러스터에 성공적으로 연결된 것입니다.

연결 문제를 해결하려면 섹션을 참조하세요Amazon DocumentDB 문제 해결.

7단계: 데이터 삽입 및 쿼리

이제 클러스터에 연결되었으므로 몇 가지 쿼리를 실행하여 문서 데이터베이스 사용에 익숙해질 수 있습니다.

1. 단일 문서를 삽입하려면 다음을 입력합니다:

```
db.collection.insertOne({"hello":"DocumentDB"})
```

출력은 다음과 같습니다:

```
{
  acknowledged: true,
  insertedId: ObjectId('673657216bdf6258466b128c')
}
```

2. `findOne()` 명령으로 작성한 문서를 읽을 수 있습니다 (단일 문서만 반환하기 때문). 다음 내용을 입력합니다.

```
db.collection.findOne()
```

다음과 같은 출력내용을 얻게 됩니다.

```
{ "_id" : ObjectId("5e401fe56056fda7321fbd67"), "hello" : "DocumentDB" }
```

3. 쿼리를 몇 개 더 수행하려면 게임 프로필 사용 사례를 고려해 보세요. 먼저 `profiles`이라는 제목의 컬렉션에 몇 개의 항목을 삽입합니다. 다음 내용을 입력합니다.

```
db.profiles.insertMany([{ _id: 1, name: 'Matt', status: 'active', level: 12, score:
 202 },
      { _id: 2, name: 'Frank', status: 'inactive', level: 2, score: 9 },
      { _id: 3, name: 'Karen', status: 'active', level: 7, score: 87 },
      { _id: 4, name: 'Katie', status: 'active', level: 3, score: 27 }
])
```

   다음과 같은 출력내용을 얻게 됩니다.

```
{ acknowledged: true, insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4 } }
```

4. `find()` 명령을 사용하여 프로필 컬렉션의 모든 문서를 반환합니다. 다음 내용을 입력합니다.

```
db.profiles.find()
```

   3단계에서 입력한 데이터와 일치하는 출력이 표시됩니다.

5. 필터를 사용하여 단일 문서에 대한 쿼리를 사용하세요. 다음을 입력합니다:

```
db.profiles.find({name: "Katie"})
```

   다음과 같은 출력내용을 얻게 됩니다.

```
{ "_id" : 4, "name" : "Katie", "status": "active", "level": 3, "score":27}
```

6. 이제 프로필을 찾아 `findAndModify` 명령을 사용하여 해당 프로필을 수정해 보겠습니다. 다음 코드를 사용하여 사용자 Matt에게 10점을 추가로 주겠습니다.

```
db.profiles.findAndModify({
        query: { name: "Matt", status: "active"},
        update: { $inc: { score: 10 } }
    })
```

   다음과 같은 결과가 출력됩니다 (참고로 그의 점수는 아직 오르지 않았습니다):

```
{
    [{_id : 1, name : 'Matt', status: 'active', level: 12, score: 202}]
```

7. 다음 쿼리를 통해 그의 점수가 변경되었는지 확인할 수 있습니다:

   ```
   db.profiles.find({name: "Matt"})
   ```

   출력은 다음과 같습니다.

   ```
   { "_id" : 1, "name" : "Matt", "status" : "active", "level" : 12, "score" : 212 }
   ```

8단계: 살펴보기

축하합니다! Amazon DocumentDB에 대한 퀵 스타트 가이드를 성공적으로 완료했습니다.

다음 단계? 몇 가지 인기 있는 기능을 갖춘 이 강력한 데이터베이스를 최대한 활용하는 방법을 알아보세요.

- Amazon DocumentDB 클러스터 및 인스턴스 관리
- 스케일링
- 백업 및 복원

> **Note**
>
> 비용을 절감하려면 Amazon DocumentDB 클러스터를 중지하여 비용을 절감하거나 클러스터를 삭제할 수 있습니다. 기본적으로 30분 동안 사용하지 않으면 AWS Cloud9 환경이 기본 Amazon EC2 인스턴스를 중지합니다.

EC2 인스턴스를 기존 Amazon DocumentDB 데이터베이스에 자동 연결

다음 절차에서는 기존 Amazon DocumentDB 클러스터와 기존 Amazon EC2 인스턴스가 있다고 가정합니다.

Amazon DocumentDB 클러스터에 액세스하고 Amazon EC2 연결을 설정합니다.

1. Amazon DocumentDB 클러스터에 액세스합니다.

   a. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.

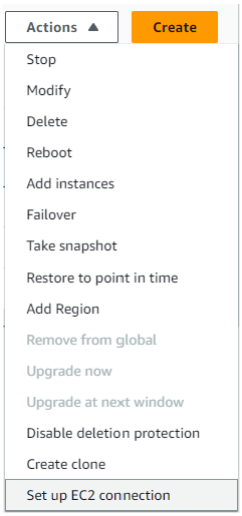
   b. 탐색 창에서 클러스터를 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

c. 클러스터 이름 왼쪽에 있는 버튼을 선택하여 원하는 클러스터를 지정합니다.

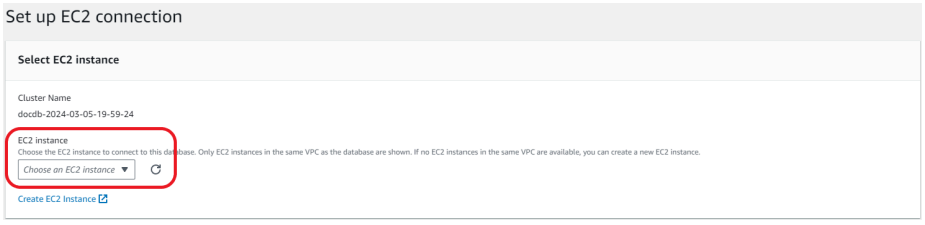
2. Amazon EC2 연결을 설정합니다.

a. 작업을 선택하고 EC2 연결 설정을 선택합니다.



EC2 연결 설정 대화가 나타납니다.

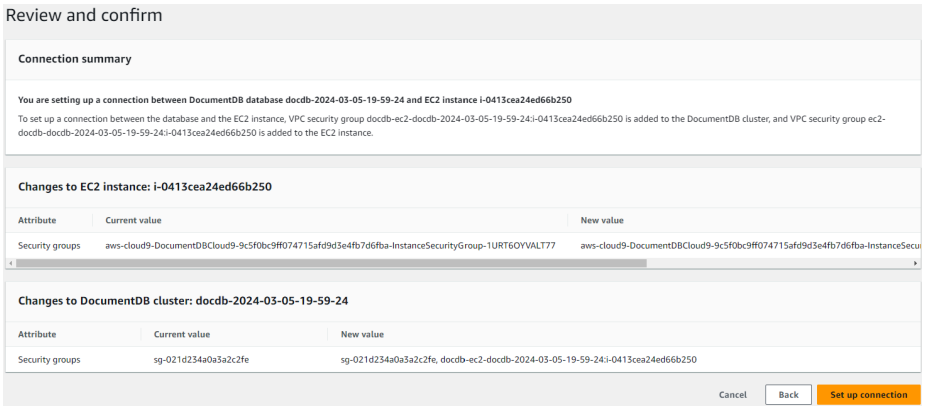
b. EC2 인스턴스 필드에서 클러스터에 연결하려는 EC2 인스턴스를 선택합니다.



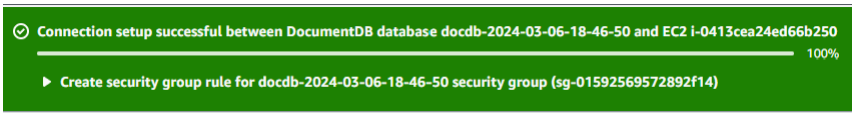
c. 계속을 선택합니다.

검토 및 확인 대화가 나타납니다.

d. 변경 사항이 올바른지 확인합니다. 그런 다음 연결 설정을 선택합니다.



성공하면 다음 확인 메시지가 나타납니다.



EC2 인스턴스와의 자동 연결 개요

EC2 인스턴스와 Amazon DocumentDB 데이터베이스 간의 연결을 설정하는 경우 Amazon DocumentDB는 EC2 인스턴스 및 Amazon DocumentDB 데이터베이스에 대한 VPC 보안 그룹을 자동으로 구성합니다.

다음은 EC2 인스턴스를 Amazon DocumentDB 데이터베이스와 연결하기 위한 요구 사항입니다.

- EC2 인스턴스는 Amazon DocumentDB 데이터베이스와 동일한 VPC가 있어야 합니다.

  EC2 인스턴스가 동일한 VPC에 없는 경우 콘솔은 EC2 인스턴스를 생성하기 위한 링크를 제공합니다.
- 연결을 설정하는 사용자는 다음 Amazon EC2 작업을 수행할 수 있는 권한이 있어야 합니다.
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeSecurityGroups`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

DB 인스턴스와 EC2 인스턴스가 서로 다른 가용 영역에 있는 경우 계정에 교차 가용 영역 비용이 발생할 가능성이 있습니다.

EC2 인스턴스에 대한 연결을 설정하면 Amazon RDS는 다음 테이블에 설명된 대로 Amazon DocumentDB 데이터베이스 및 EC2 인스턴스와 연결된 보안 그룹의 현재 구성을 기반으로 조치를 취합니다.

현재 Amazon DocumentDB 보안 그룹 구성	현재 EC2 보안 그룹 구성	Amazon DocumentDB 작업
이름이 `DocumentDB-ec2-n` 패턴과 일치하며 Amazon DocumentDB 데이터베이스와 연결된 보안 그룹이 하나 이상 있습니다. 해당 패턴과 일치하는 보안 그룹이 수정되지 않았습니다. 이 보안 그룹에는 EC2 인스턴스의 VPC 보안 그룹을 소스로 하는 인바운드 규칙이 하나만 포함됩니다.	`DocumentDB-ec2-n` (n은 숫자를 나타냄) 패턴과 일치하는 이름을 가진 EC2 인스턴스와 연결된 보안 그룹이 하나 이상 있습니다. 해당 패턴과 일치하는 보안 그룹이 수정되지 않았습니다. 이 보안 그룹에는 Amazon DocumentDB 데이터베이스의 VPC 보안 그룹을 원본으로 하는 아웃바운드 규칙이 하나만 포함됩니다.	Amazon DocumentDB는 아무런 조치도 취하지 않습니다. EC2 인스턴스와 Amazon DocumentDB 데이터베이스 간의 연결이 이미 자동으로 구성되었습니다. EC2 인스턴스와 Amazon DocumentDB 데이터베이스 사이에 이미 연결이 존재하기 때문에 보안 그룹은 수정되지 않습니다.
다음 중 하나의 조건이 적용됩니다. • `DocumentDB-ec2-n` 패턴과 일치하는 이름을 가진 Amazon DocumentDB 데이터베이스와 연결된 보안 그룹이 없습니다. • 이름이 `DocumentDB-ec2-n` 패턴과 일치하며 Amazon DocumentDB와 연결된 보안 그룹이 하나 이상 있습니다. 그러나 Amazon DocumentDB는 EC2 인스턴스와의 연결에 이러한 보	다음 중 하나의 조건이 적용됩니다. • `ec2-DocumentDB-n` 패턴과 일치하는 이름을 가진 EC2 인스턴스와 연결된 보안 그룹이 없습니다. • `ec2-DocumentDB-n` 패턴과 일치하는 이름을 가진 EC2 인스턴스와 연결된 보안 그룹이 하나 이상 있습니다. 그러나 Amazon DocumentDB는 Amazon DocumentDB 데이터베이스와의 연결에 이러한 보안 그룹을 사용할 수 없습	Amazon DocumentDB 작업: 새 보안 그룹 생성

현재 Amazon DocumentDB 보안 그룹 구성	현재 EC2 보안 그룹 구성	Amazon DocumentDB 작업
안 그룹을 사용할 수 없습니다. Amazon DocumentDB는 EC2 인스턴스의 VPC 보안 그룹을 소스로 하는 인바운드 규칙 하나가 포함되지 않은 보안 그룹을 사용할 수 없습니다. Amazon DocumentDB는 또한 수정된 보안 그룹을 사용할 수 없습니다. 수정 사항의 예로는 규칙 추가 또는 기존 규칙의 포트 변경이 있습니다.	니다. Amazon DocumentDB는 Amazon DocumentDB 데이터베이스의 VPC 보안 그룹을 원본으로 하는 아웃바운드 규칙 하나가 포함되지 않은 보안 그룹을 사용할 수 없습니다. Amazon DocumentDB는 또한 수정된 보안 그룹을 사용할 수 없습니다.	
이름이 `DocumentDB-ec2-n` 패턴과 일치하며 Amazon DocumentDB 데이터베이스와 연결된 보안 그룹이 하나 이상 있습니다. 해당 패턴과 일치하는 보안 그룹이 수정되지 않았습니다. 이 보안 그룹에는 EC2 인스턴스의 VPC 보안 그룹을 소스로 하는 인바운드 규칙이 하나만 포함됩니다.	`ec2-DocumentDB-n` 패턴과 일치하는 이름을 가진 EC2 인스턴스와 연결된 보안 그룹이 하나 이상 있습니다. 그러나 Amazon DocumentDB는 Amazon DocumentDB 데이터베이스와의 연결에 이러한 보안 그룹을 사용할 수 없습니다. Amazon DocumentDB는 Amazon DocumentDB 데이터베이스의 VPC 보안 그룹을 원본으로 하는 아웃바운드 규칙 하나가 포함되지 않은 보안 그룹을 사용할 수 없습니다. Amazon DocumentDB는 또한 수정된 보안 그룹을 사용할 수 없습니다.	Amazon DocumentDB 작업: 새 보안 그룹 생성

현재 Amazon DocumentDB 보안 그룹 구성	현재 EC2 보안 그룹 구성	Amazon DocumentDB 작업
이름이 `DocumentDB-ec2-n` 패턴과 일치하며 Amazon DocumentDB 데이터베이스와 연결된 보안 그룹이 하나 이상 있습니다. 해당 패턴과 일치하는 보안 그룹이 수정되지 않았습니다. 이 보안 그룹에는 EC2 인스턴스의 VPC 보안 그룹을 소스로 하는 인바운드 규칙이 하나만 포함됩니다.	연결에 유효한 EC2 보안 그룹이 있지만 EC2 인스턴스와 연결되어 있지 않습니다. 이 보안 그룹의 이름이 `DocumentDB-ec2-n` 패턴과 일치합니다. 수정되지 않았습니다. Amazon DocumentDB 데이터베이스의 VPC 보안 그룹을 원본으로 하는 아웃바운드 규칙이 하나만 포함됩니다.	Amazon DocumentDB 작업: EC2 보안 그룹 연결

현재 Amazon DocumentDB 보안 그룹 구성	현재 EC2 보안 그룹 구성	Amazon DocumentDB 작업
다음 중 하나의 조건이 적용됩니다. • `DocumentDB-ec2-n` 패턴과 일치하는 이름을 가진 Amazon DocumentDB 데이터베이스와 연결된 보안 그룹이 없습니다. • 이름이 `DocumentDB-ec2-n` 패턴과 일치하며 Amazon DocumentDB 데이터베이스와 연결된 보안 그룹이 하나 이상 있습니다. 그러나 Amazon DocumentDB는 EC2 인스턴스와의 연결에 이러한 보안 그룹을 사용할 수 없습니다. Amazon DocumentDB는 EC2 인스턴스의 VPC 보안 그룹을 소스로 하는 인바운드 규칙 하나가 포함되지 않은 보안 그룹을 사용할 수 없습니다. 또한 Amazon DocumentDB는 수정된 보안 그룹을 사용할 수 없습니다.	`DocumentDB-ec2-n` 패턴과 일치하는 이름을 가진 EC2 인스턴스와 연결된 보안 그룹이 하나 이상 있습니다. 해당 패턴과 일치하는 보안 그룹이 수정되지 않았습니다. 이 보안 그룹에는 Amazon DocumentDB 데이터베이스의 VPC 보안 그룹을 원본으로 하는 아웃바운드 규칙이 하나만 포함됩니다.	Amazon DocumentDB 작업: 새 보안 그룹 생성

Amazon DocumentDB 작업: 새 보안 그룹 생성

Amazon DocumentDB에서 다음 작업을 수행합니다.

- `DocumentDB-ec2-n` 패턴과 일치하는 새 보안 그룹을 생성합니다. 이 보안 그룹에는 EC2 인스턴스의 VPC 보안 그룹을 소스로 하는 인바운드 규칙이 하나 포함됩니다. 이 보안 그룹은 Amazon DocumentDB 데이터베이스와 연결되어 있으며 EC2 인스턴스가 Amazon DocumentDB 데이터베이스에 액세스하도록 허용합니다.

- `ec2-DocumentDB-n` 패턴과 일치하는 새 보안 그룹을 생성합니다. 이 보안 그룹에는 Amazon DocumentDB 데이터베이스의 VPC 보안 그룹을 원본으로 하는 아웃바운드 규칙이 있습니다. 이 보안 그룹은 EC2 인스턴스와 연결되어 있으며 EC2 인스턴스가 Amazon DocumentDB 데이터베이스에 트래픽을 보내도록 허용합니다.

Amazon DocumentDB 작업: EC2 보안 그룹 연결

Amazon DocumentDB는 유효한 기존 EC2 보안 그룹을 EC2 인스턴스와 연결합니다. 이 보안 그룹은 EC2 인스턴스가 Amazon DocumentDB 데이터베이스에 트래픽을 보내도록 허용합니다.

연결된 컴퓨팅 리소스 보기

AWS Management Console 를 사용하여 Amazon DocumentDB 데이터베이스에 연결된 컴퓨팅 리소스를 볼 수 있습니다. 표시되는 리소스에는 자동으로 설정된 컴퓨팅 리소스 연결이 포함됩니다. 다음과 같은 방법으로 컴퓨팅 리소스와의 연결을 자동으로 설정할 수 있습니다.

- 데이터베이스를 생성할 때 컴퓨팅 리소스를 선택할 수 있습니다. 자세한 내용은 Amazon DocumentDB 클러스터 생성 및 다중 AZ DB 클러스터 생성을 참조하세요.
- 기존 데이터베이스와 컴퓨팅 리소스 간의 연결을 설정할 수 있습니다. 자세한 내용은 Amazon EC2 자동 연결 섹션을 참조하세요.

나열된 컴퓨팅 리소스에는 데이터베이스에 수동으로 연결된 리소스는 포함되지 않습니다. 예를 들어 데이터베이스와 연결된 VPC 보안 그룹에 규칙을 추가하여 컴퓨팅 리소스가 데이터베이스에 수동으로 액세스하도록 허용할 수 있습니다.

컴퓨팅 리소스가 나열되려면 다음 조건이 적용되어야 합니다.

- 컴퓨팅 리소스와 연결된 보안 그룹의 이름은 `ec2-DocumentDB-n` 패턴(n이 숫자)과 일치합니다.
- 컴퓨팅 리소스와 연결된 보안 그룹에는 포트 범위가 Amazon DocumentDB 데이터베이스에서 사용하는 포트로 설정된 아웃바운드 규칙이 있습니다.
- 컴퓨팅 리소스와 연결된 보안 그룹에는 소스가 Amazon DocumentDB 데이터베이스에 연결된 보안 그룹으로 설정된 아웃바운드 규칙이 있습니다.
- Amazon DocumentDB 데이터베이스와 연결된 보안 그룹의 이름은 `DocumentDB-ec2-n` 패턴(n이 숫자)과 일치합니다.
- Amazon DocumentDB 데이터베이스와 연결된 보안 그룹에는 포트 범위가 Amazon DocumentDB 데이터베이스에서 사용하는 포트로 설정된 인바운드 규칙이 있습니다.

- Amazon DocumentDB 데이터베이스와 연결된 보안 그룹에는 원본이 컴퓨팅 리소스에 연결된 보안 그룹으로 설정된 인바운드 규칙이 있습니다.

Amazon DocumentDB 데이터베이스에 연결된 컴퓨팅 리소스 보기

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택한 다음 Amazon DocumentDB 데이터베이스의 이름을 선택합니다.
3. 연결 및 보안 탭의 연결된 컴퓨팅 리소스 섹션에서 컴퓨팅 리소스를 확인합니다.

Amazon EC2 수동 연결

주제

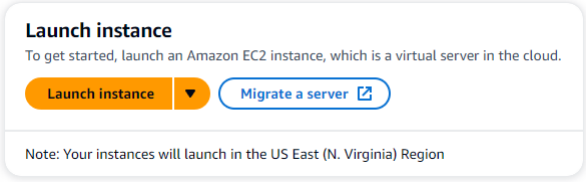
- 1단계: Amazon EC2 인스턴스 생성
- 2단계: 보안 그룹 만들기
- 3단계: Amazon DocumentDB 클러스터 생성
- 4단계: Amazon EC2 인스턴스 구성
- 5단계: MongoDB 쉘 설치
- 6단계: Amazon DocumentDB TLS 관리
- 7단계: Amazon DocumentDB 클러스터에 연결
- 8단계: 데이터 삽입 및 쿼리
- 9단계: 살펴보기

다음 단계에서는 사전 조건 주제의 단계를 완료했다고 가정합니다.

1단계: Amazon EC2 인스턴스 생성

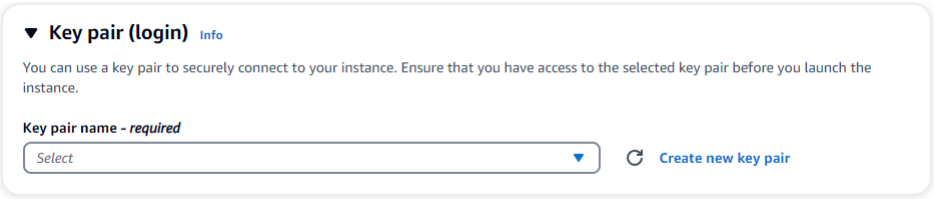
이 단계에서는 나중에 Amazon DocumentDB 클러스터를 프로비저닝하는 데 사용할 동일한 지역 및 Amazon VPC에 Amazon EC2 인스턴스를 생성합니다.

1. Amazon EC2 콘솔 대시보드에서 인스턴스 시작을 선택합니다.



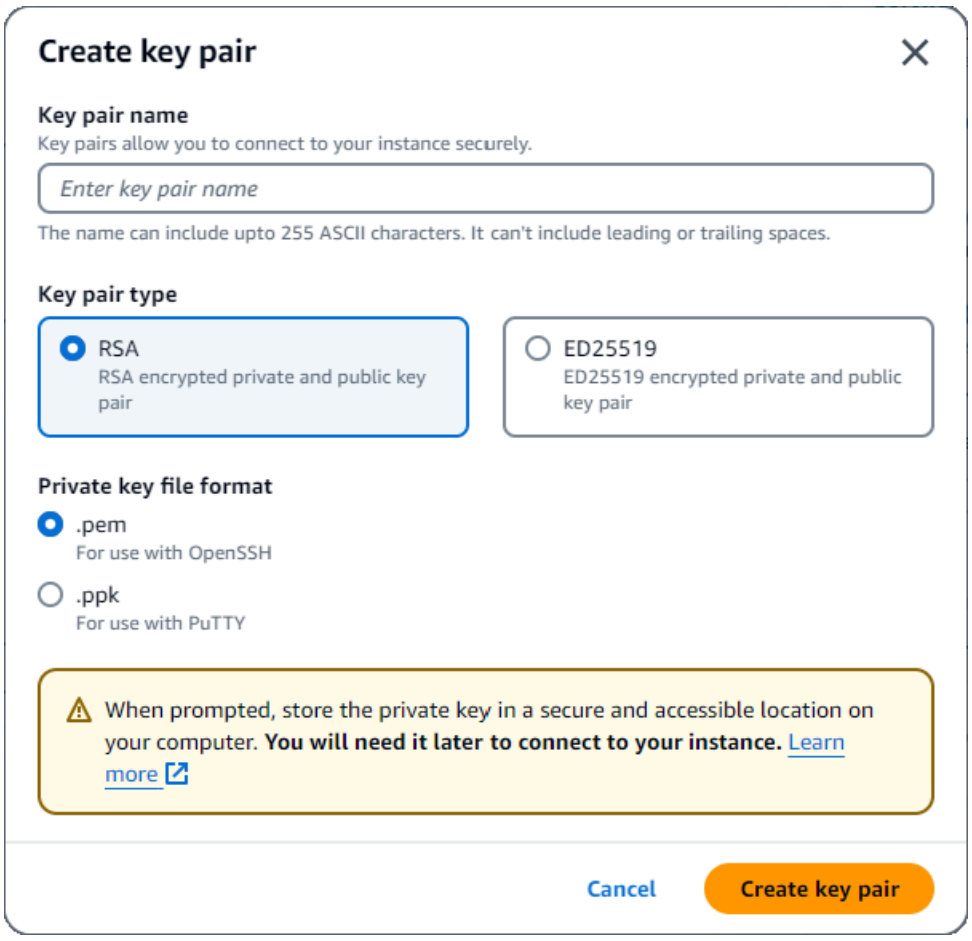
2. 이름 및 태그 섹션에 있는 이름 필드에 이름 또는 식별자를 입력합니다.
3. Amazon Machine Image(AMI) 드롭다운 목록에서 Amazon Linux 2 AMI를 찾아 선택합니다.

4. 인스턴스 유형 드롭다운 목록에서 t3.micro를 찾아 선택합니다.
5. 키 페어(로그인) 섹션에서 기존 키 페어의 식별자를 입력하거나 새 키 페어 생성을 선택합니다.



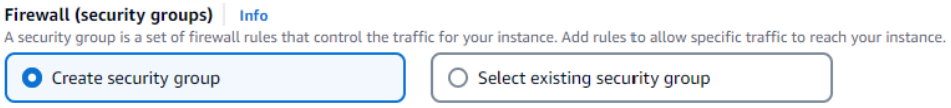
Amazon EC2 키 페어를 제공해야 합니다.

- Amazon EC2 키 페어가 있는 경우:
  1. 키 페어를 선택하고 목록에서 키 페어를 선택합니다.
  2. Amazon EC2 인스턴스에 로그인하려면 프라이빗 키 파일(.pem 또는 .ppk 파일)이 있어야 합니다.
- Amazon EC2 키 페어가 없는 경우:
  1. 새 키 페어 생성을 선택하면 키 페어 생성 대화 상자가 나타납니다.
  2. 키 페어 명칭 필드에 명칭을 입력합니다.
  3. 키 페어 유형 및 프라이빗 키 파일 형식을 선택합니다.
  4. 키 페어 생성(Create key pair)를 선택합니다.



> **Note**
>
> 보안을 위해 SSH 및 EC2 인스턴스에 대한 인터넷 연결 모두에 키 페어를 사용하는 것이 좋습니다.

6. 네트워크 설정 섹션의 방화벽(보안 그룹)에서 보안 그룹 생성 또는 기존 보안 그룹 선택을 선택합니다.



기존 보안 그룹을 선택하기로 선택한 경우 일반 보안 그룹 드롭다운 목록에서 하나를 선택합니다.

새 보안 그룹을 생성하도록 선택한 경우 다음을 수행합니다.

a. EC2 연결에 적용되는 모든 트래픽 허용 규칙을 확인합니다.

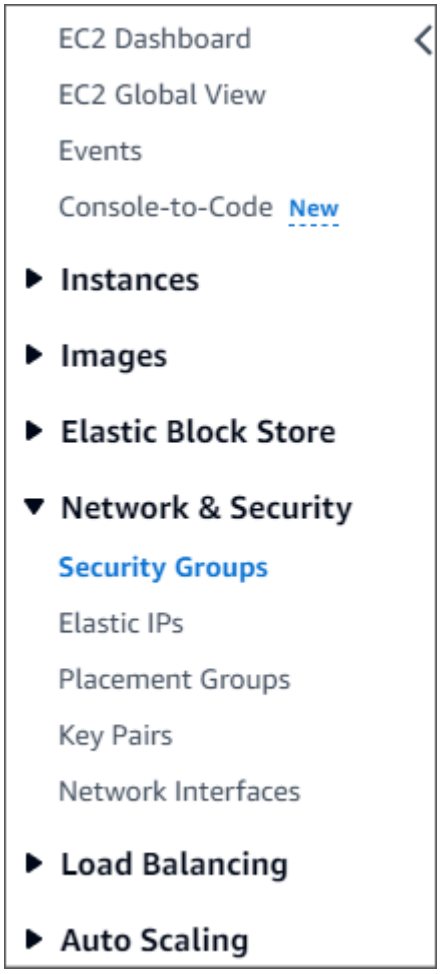
b. IP 필드에서 내 IP를 선택하거나 사용자 지정을 선택하여 CIDR 블록, 접두사 목록 또는 보안 그룹 목록에서 선택합니다. EC2 인스턴스는 EC2 인스턴스에 대한 IP 주소 액세스를 허용하므로 EC2 인스턴스가 격리된 네트워크에 있지 않는 한 Anywhere를 선택하지 않는 것이 좋습니다.

7. 요약 섹션에서 EC2 구성을 검토하고 올바른 경우 인스턴스 시작을 선택합니다.

2단계: 보안 그룹 만들기

이제 기본 Amazon VPC에 새 보안 그룹을 생성합니다. 보안 그룹을 demoDocDB 사용하면 Amazon EC2 인스턴스에서 포트 27017 (Amazon DocumentDB의 기본 포트) 을 통해 Amazon DocumentDB 클러스터에 연결할 수 있습니다.

1. Amazon EC2 관리 콘솔의 네트워크 및 보안에서 보안 그룹을 선택합니다.



2. 보안 그룹 생성을 선택합니다.

3. 기본 세부 정보 섹션에서 다음을 수행합니다.

   a. 보안 그룹 이름에 `demoDocDB`를 입력합니다.

   b. 설명에 설명을 입력합니다.

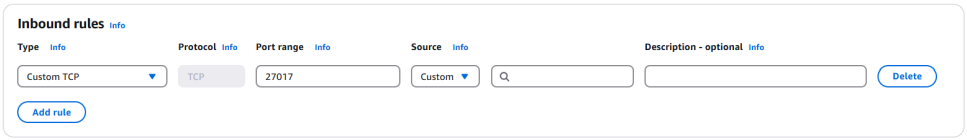
   c. VPC의 경우 기본 VPC 사용을 수락합니다.

4. 인바운드 규칙 섹션에서 규칙 추가를 선택합니다.

   a. 유형의 경우 사용자 지정 TCP 규칙(기본값)을 선택합니다.

   b. 포트 범위에 `27017`를 입력합니다.

   c. 소스(Source)에서 사용자 지정(Custom)을 선택합니다. 옆에 있는 필드에서, 1단계에서 방금 생성한 보안 그룹을 검색합니다. Amazon EC2 콘솔에서 소스 이름을 자동으로 채우려면 브라우저를 새로 고쳐야 할 수 있습니다.



5. 다른 모든 기본값을 그대로 사용하고 보안 그룹 생성을 선택합니다.

3단계: Amazon DocumentDB 클러스터 생성

Amazon EC2 인스턴스가 프로비저닝되는 동안 Amazon DocumentDB 클러스터를 생성해야 합니다.

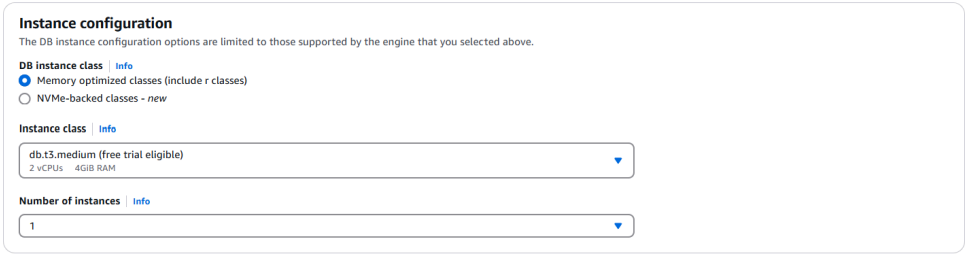
1. Amazon DocumentDB 콘솔로 이동한 다음 탐색 창에서 클러스터를 선택합니다.
2. 생성을 선택합니다.
3. 클러스터 유형 설정을 인스턴스 기반 클러스터 의 기본값으로 둡니다.
4. 클러스터 구성에서 클러스터 식별자에 고유한 이름을 입력합니다. 콘솔은 모든 클러스터 이름을 입력 방식에 관계없이 소문자로 변경합니다.

   엔진 버전을 기본값인 5.0.0으로 그대로 유지합니다.
5. 클러스터 스토리지 구성의 경우 Amazon DocumentDB Standard의 기본 설정을 그대로 유지합니다.

6. 인스턴스 구성에서 다음을 수행합니다.

   - DB 인스턴스 클래스에서 메모리 최적화 클래스(r 클래스 포함)(기본값)를 선택합니다.

     다른 인스턴스 옵션은 NVMe 지원 클래스입니다. 자세한 내용은 NVMe 지원 인스턴스 섹션을 참조하세요.
   - 인스턴스 클래스에서 요구 사항에 맞는 인스턴스 유형을 선택합니다. 인스턴스 클래스에 대한 자세한 설명은 인스턴스 클래스 사양 섹션을 참조하세요.
   - 인스턴스 수에서 요구 사항을 가장 잘 반영하는 숫자를 선택합니다. 숫자가 낮을수록 비용이 낮아지고 클러스터에서 관리할 수 있는 읽기/쓰기 볼륨이 낮아집니다.



7. 연결의 경우 EC2 컴퓨팅 리소스에 연결 안 함을 기본 설정으로 둡니다.

   > **Note**
   >
   > EC2 컴퓨팅 리소스에 연결하면 클러스터에 대한 연결을 위한 보안 그룹이 자동으로 생성됩니다. 이전 단계에서 이러한 보안 그룹을 수동으로 생성했으므로 두 번째 보안 그룹 세트를 생성하지 않도록 EC2 컴퓨팅 리소스에 연결하지 않음을 선택해야 합니다.

8. 인증 섹션에 기본 사용자의 사용자 이름을 입력한 후 자체 관리형을 선택합니다. 암호를 입력한 다음 확인합니다.

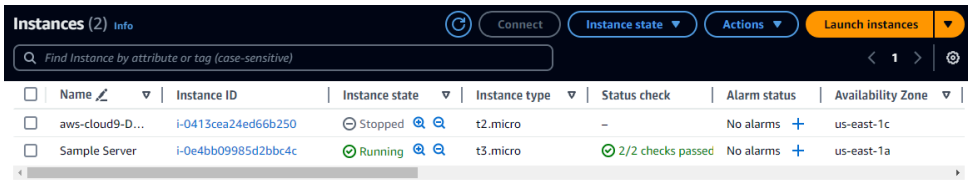
   대신 관리형을 AWS Secrets Manager 선택한 경우 자세한 내용은 Amazon DocumentDB 및를 사용한 암호 관리 AWS Secrets Manager 섹션을 참조하세요.

9. 클러스터 생성을 선택합니다.

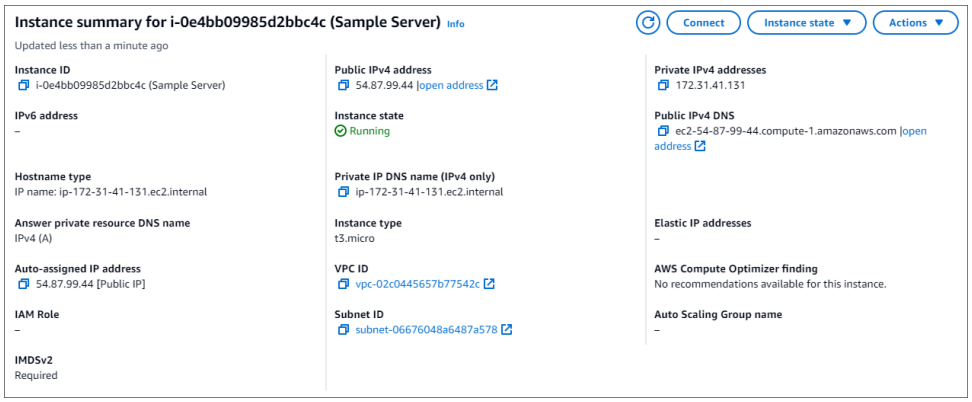
4단계: Amazon EC2 인스턴스 구성

Amazon EC2 인스턴스에 연결하면 MongoDB 쉘을 설치할 수 있습니다. mongo 쉘을 설치하면 Amazon DocumentDB 클러스터에 연결하고 쿼리할 수 있습니다. 다음 단계를 완료합니다.

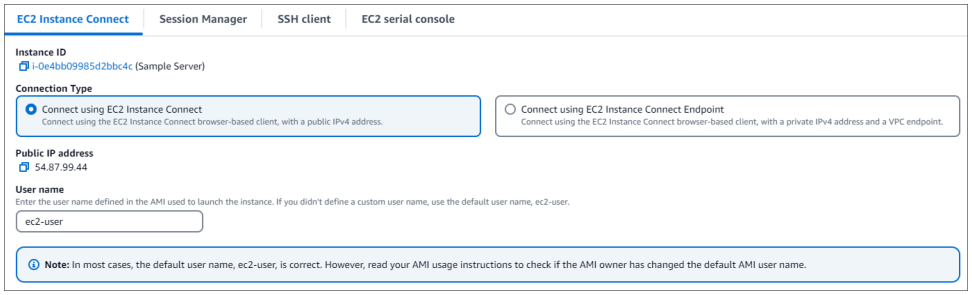
1. Amazon EC2 콘솔에서 인스턴스로 이동하여 방금 생성한 인스턴스가 실행 중인지 확인합니다. 그렇다면 인스턴스 ID를 클릭하여 인스턴스를 선택합니다.



2. 연결을 선택합니다.



3. 연결 방법에는 Amazon EC2 Instance Connect, 세션 관리자, SSH 클라이언트 또는 EC2 직렬 콘솔의 4가지 탭 옵션이 표시됩니다. 하나를 선택하고 지침을 따라야 합니다. 완료되면 연결을 선택합니다.



> **Note**
>
> 이 안내를 시작한 후 IP 주소가 변경되었거나 나중에 해당 환경으로 돌아오려는 경우 새 API 주소로부터의 인바운드 트래픽을 활성화하도록 demoEC2 보안 그룹 인바운드 규칙을 업데이트해야 합니다.

5단계: MongoDB 쉘 설치

이제 Amazon DocumentDB 클러스터에 연결하고 쿼리하는 데 사용하는 명령줄 유틸리티인 MongoDB 쉘을 설치할 수 있습니다. 현재 MongoDB 쉘에는 최신 버전인 mongosh와 이전 버전인 Mongo 쉘의 두 가지 버전이 있습니다.

> **Important**
>
> Amazon DocumentDB에 대한 IAM ID 인증에서 지원되지 않는 Node.js 드라이버(6.13.1 이전 버전)에는 알려진 제한 사항이 있습니다. Node.js 드라이버를 사용하는 Node.js 드라이버 및 도구(예: mongosh)는 Node.js 드라이버 버전 6.13.1 이상을 사용하도록 업그레이드해야 합니다.

아래 지침에 따라 사용 중인 운영 체제에 맞는 MongoDB 쉘을 설치합니다.

On Amazon Linux

Amazon Linux에 MongoDB 쉘 설치

IAM을 사용하지 않고 최신 MongoDB 쉘(mongosh)을 사용하여 Amazon DocumentDB 클러스터에 연결하려면 다음 단계를 따르세요.

1. 리포지토리 파일을 생성합니다. 생성한 EC2 인스턴스의 명령줄에서 다음 명령을 실행합니다.

```
echo -e "[mongodb-org-5.0] \nname=MongoDB Repository\nbaseurl=https://
repo.mongodb.org/yum/amazon/2023/mongodb-org/5.0/x86_64/\ngpgcheck=1 \nenabled=1
 \ngpgkey=https://pgp.mongodb.com/server-5.0.asc" | sudo tee /etc/yum.repos.d/
mongodb-org-5.0.repo
```

2. 완료되면 명령 프롬프트에서 다음 두 명령 옵션 중 하나를 사용하여 mongosh를 설치합니다.

   옵션 1 - Amazon EC2 구성 중에 기본 Amazon Linux 2023을 선택한 경우 다음 명령을 입력합니다.

```
sudo yum install -y mongodb-mongosh-shared-openssl3
```

   옵션 2 - Amazon EC2 구성 중에 Amazon Linux 2를 선택한 경우 다음 명령을 입력합니다.

```
sudo yum install -y mongodb-mongosh
```

IAM을 사용하는 경우 이전 버전의 MongoDB 쉘(5.0)을 사용하여 Amazon DocumentDB 클러스터에 연결해야 합니다. 다음 단계를 따르세요.

1. 리포지토리 파일을 생성합니다. 생성한 EC2 인스턴스의 명령줄에서 다음 명령을 실행합니다.

```
echo -e "[mongodb-org-5.0] \nname=MongoDB Repository\nbaseurl=https://
repo.mongodb.org/yum/amazon/2023/mongodb-org/5.0/x86_64/\ngpgcheck=1 \nenabled=1
 \ngpgkey=https://pgp.mongodb.com/server-5.0.asc" | sudo tee /etc/yum.repos.d/
mongodb-org-5.0.repo
```

2. 완료되면 명령 프롬프트에서 다음 명령 옵션을 사용하여 mongodb 5.0 쉘을 설치합니다.

```
sudo yum install -y mongodb-org-shell
```

On Ubuntu

Ubuntu에 mongosh 설치

1. 패키지 관리 시스템에서 사용할 퍼블릭 키를 가져옵니다.

```
curl -fsSL https://pgp.mongodb.com/server-5.0.asc | sudo gpg --dearmor -o /usr/
share/keyrings/mongodb-server-5.0.gpg
```

2. Ubuntu 버전에 맞는 명령을 사용하여 MongoDB의 목록 파일 `mongodb-org-5.0.list`를 생성합니다.

```
echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-
server-5.0.gpg ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/5.0
 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-5.0.list
```

3. 다음 명령을 사용하여 로컬 패키지 데이터베이스를 가져오고 업데이트합니다.

```
sudo apt-get update
```

4. mongosh를 설치합니다.

```
sudo apt-get install -y mongodb-mongosh
```

Ubuntu 시스템에서 이전 버전의 MongoDB를 설치하는 방법에 대한 자세한 내용은 Install MongoDB Community Edition on Ubuntu를 참조하세요.

On other operating systems

다른 운영 체제에서 mongo 쉘을 설치하려면 MongoDB 설명서의 MongoDB Community Edition 설치를 참조하세요.

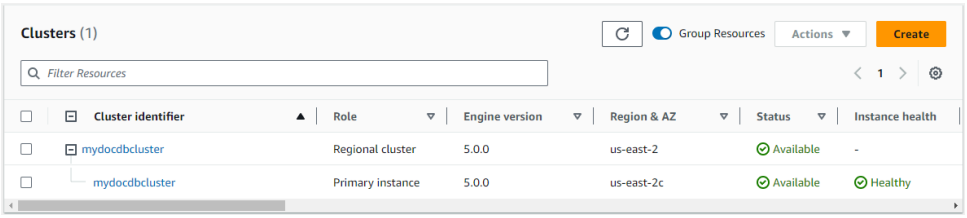
6단계: Amazon DocumentDB TLS 관리

다음 코드를 사용하여 Amazon DocumentDB용 CA 인증서를 다운로드하세요. `wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem`

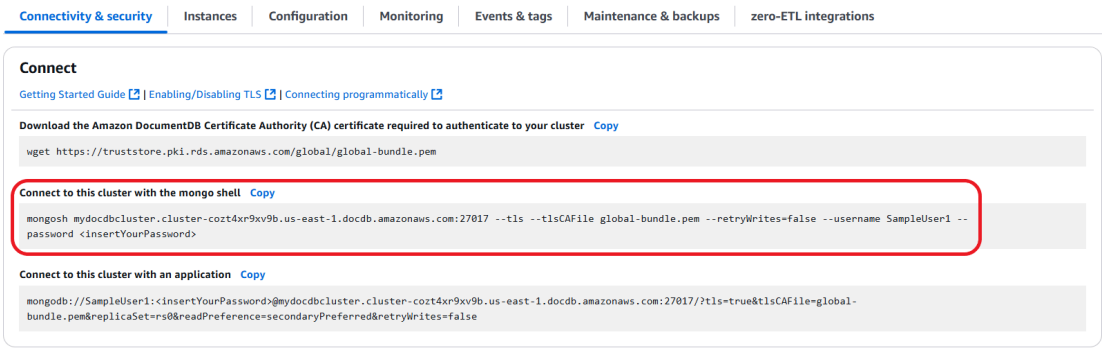
> **Note**
>
> 전송 계층 보안 (TLS) 은 모든 새 Amazon DocumentDB 클러스터에 기본적으로 활성화됩니다. 자세한 내용은 전송 중 데이터 암호화 섹션을 참조하세요.

7단계: Amazon DocumentDB 클러스터에 연결

1. Amazon DocumentDB 콘솔의 클러스터에서 클러스터를 찾습니다. 해당 클러스터의 클러스터 식별자를 클릭하여 생성한 클러스터를 선택합니다.



2. 연결 및 보안 탭에서 연결 상자의 mongo 쉘을 사용하여 이 클러스터에 연결을 찾습니다.



제공된 연결 문자열을 복사하여 터미널에 붙여넣습니다.

해당 항목을 다음과 같이 변경합니다.

a. 문자열에 올바른 사용자 이름이 있는지 확인합니다.

b. 그런 다음 연결할 때 mongo 쉘에서 비밀번호를 입력하라는 메시지가 표시되도록 `<insertYourPassword>` 항목을 생략합니다.

c. 선택 사항: IAM 인증을 사용하거나 이전 버전의 MongoDB 쉘을 사용하는 경우 다음과 같이 연결 문자열을 수정합니다.

```
mongo --ssl --host mydocdbcluster.cluster-cozt4xr9xv9b.us-
east-1.docdb.amazonaws.com:27017 --sslCAFile global-bundle.pem --
username SampleUser1 --password
```

`mydocdbcluster.cluster-cozt4xr9xv9b.us-east-1`을 클러스터의 동일한 정보로 바꿉니다.

3. 이제 터미널에서 Enter를 누릅니다. 이제 비밀번호를 입력하라는 메시지가 표시됩니다. 암호를 입력합니다.

4. 암호를 입력하고 `rs0 [direct: primary] <env-name>>` 메시지가 표시되면 Amazon DocumentDB 클러스터에 성공적으로 연결된 것입니다.

연결 문제를 해결하려면 섹션을 참조하세요Amazon DocumentDB 문제 해결.

8단계: 데이터 삽입 및 쿼리

이제 클러스터에 연결되었으므로 몇 가지 쿼리를 실행하여 문서 데이터베이스 사용에 익숙해질 수 있습니다.

1. 단일 문서를 삽입하려면 다음을 입력합니다:

```
db.collection.insertOne({"hello":"DocumentDB"})
```

출력은 다음과 같습니다:

```
{
  acknowledged: true,
  insertedId: ObjectId('673657216bdf6258466b128c')
}
```

2. findOne() 명령으로 작성한 문서를 읽을 수 있습니다 (단일 문서만 반환하기 때문). 다음 내용을 입력합니다.

```
db.collection.findOne()
```

다음과 같은 출력내용을 얻게 됩니다.

```
{ "_id" : ObjectId("5e401fe56056fda7321fbd67"), "hello" : "DocumentDB" }
```

3. 쿼리를 몇 개 더 수행하려면 게임 프로필 사용 사례를 고려해 보세요. 먼저 profiles이라는 제목의 컬렉션에 몇 개의 항목을 삽입합니다. 다음 내용을 입력합니다.

```
db.profiles.insertMany([{ _id: 1, name: 'Matt', status: 'active', level: 12, score:
 202 },
      { _id: 2, name: 'Frank', status: 'inactive', level: 2, score: 9 },
      { _id: 3, name: 'Karen', status: 'active', level: 7, score: 87 },
      { _id: 4, name: 'Katie', status: 'active', level: 3, score: 27 }
])
```

다음과 같은 출력내용을 얻게 됩니다.

```
{ acknowledged: true, insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4 } }
```

4. find() 명령을 사용하여 프로필 컬렉션의 모든 문서를 반환합니다. 다음 내용을 입력합니다.

```
db.profiles.find()
```

3단계에서 입력한 데이터와 일치하는 출력이 표시됩니다.

5. 필터를 사용하여 단일 문서에 대한 쿼리를 사용하세요. 다음을 입력합니다:

```
db.profiles.find({name: "Katie"})
```

다음과 같은 출력내용을 얻게 됩니다.

```
{ "_id" : 4, "name" : "Katie", "status": "active", "level": 3, "score":27}
```

6. 이제 프로필을 찾아 findAndModify 명령을 사용하여 해당 프로필을 수정해 보겠습니다. 다음 코드를 사용하여 사용자 Matt에게 10점을 추가로 주겠습니다.

```
db.profiles.findAndModify({
        query: { name: "Matt", status: "active"},
        update: { $inc: { score: 10 } }
    })
```

다음과 같은 결과가 출력됩니다 (참고로 그의 점수는 아직 오르지 않았습니다):

```
{
    [{_id : 1, name : 'Matt', status: 'active', level: 12, score: 202}]
```

7. 다음 쿼리를 통해 그의 점수가 변경되었는지 확인할 수 있습니다:

   ```
   db.profiles.find({name: "Matt"})
   ```

   출력은 다음과 같습니다.

   ```
   { "_id" : 1, "name" : "Matt", "status" : "active", "level" : 12, "score" : 212 }
   ```

9단계: 살펴보기

축하합니다! Amazon DocumentDB에 대한 퀵 스타트 가이드를 성공적으로 완료했습니다.

다음 단계? 몇 가지 인기 있는 기능을 갖춘 이 강력한 데이터베이스를 최대한 활용하는 방법을 알아보세요.

- Amazon DocumentDB 클러스터 및 인스턴스 관리
- 스케일링
- 백업 및 복원

> **Note**
>
> 비용을 절감하려면 Amazon DocumentDB 클러스터를 중지하여 비용을 절감하거나 클러스터를 삭제할 수 있습니다. 기본적으로 30분 동안 사용하지 않으면 AWS Cloud9 환경이 기본 Amazon EC2 인스턴스를 중지합니다.

Amazon DocumentDB JDBC 드라이버를 사용하여 연결합니다.

Amazon DocumentDB용 JDBC 드라이버는 개발자를 위한 SQL 관계형 인터페이스를 제공하며, Tableau 및 DbVisualizer와 같은 BI 도구에서 연결할 수 있도록 합니다.

자세한 내용은 GitHub의 Amazon DocumentDB JDBC 드라이버 설명서를 참조하세요.

주제

- 시작하기
- 타블로 데스크톱에서 Amazon DocumentDB에 연결
- DB 비주얼라이저에서 Amazon DocumentDB에 연결
- JDBC 자동 스키마 생성
- SQL 지원 및 제한 사항
- 문제 해결

시작하기

1단계. Amazon DocumentDB 클러스터

Amazon DocumentDB 클러스터가 생성되지 않은 경우의 지침에 따라 클러스터를 생성합니다Amazon DocumentDB로 시작하기.

> **Note**
>
> Amazon DocumentDB는 가상 프라이빗 클라우드(VPC) 전용 서비스입니다. 클러스터 VPC 외부의 로컬 시스템에서 연결하는 경우 Amazon EC2 인스턴스에 대한 SSH 연결을 생성해야 합니다. 이 경우의 지침에 따라 클러스터를 시작합니다Amazon EC2를 사용하여 연결. SSH 터널링에 대한 자세한 내용과 필요할 수 있는 시기에 대해서는 SSH 터널을 사용하여 Amazon DocumentDB에 연결을 참조하세요.

2단계. JRE 또는 JDK 설치

BI 애플리케이션에 따라 컴퓨터에 64비트 JRE 또는 JDK 설치 버전 8 이상이 설치되어 있는지 확인해야 할 수 있습니다. Java SE 런타임 환경 8은 여기에서 다운로드할 수 있습니다.

3단계. DocumentDB JDBC 드라이버 다운로드

여기에서 DocumentDB JDBC 드라이버를 다운로드하세요. 드라이버는 단일 JAR 파일 (예: documentdb-jdbc-1.0.0-all.jar) 로 패키지됩니다.

4단계. SSH 터널을 사용하여 Amazon DocumentDB에 연결

Amazon DocumentDB(MongoDB 호환) 클러스터는 Amazon Virtual Private Cloud (Amazon VPC) 내에 배포됩니다. Amazon EC2 인스턴스 또는 동일한 Amazon VPC에 배포된 다른 AWS 서비스에서 직접 액세스할 수 있습니다. 또한 EC2a 인스턴스 또는 VPCs 피어링을 통해 동일한 AWS 리전 또는 다른 리전의 다른 VPC에 있는 다른 AWS 서비스에서 Amazon DocumentDB에 액세스할 수 있습니다.

SSH 터널링(포트 포워딩이라고도 함)을 사용하여 클러스터의 VPC 외부에서 Amazon DocumentDB 리소스에 액세스할 수 있습니다. 이는 대부분의 사용자가 DocumentDB 클러스터와 동일한 VPC에 있는 VM에서 애플리케이션을 실행하지 않는 경우입니다.

SSH 터널을 만들려면 Amazon DocumentDB 클러스터와 동일한 Amazon VPC에서 실행 중인 Amazon EC2 인스턴스가 필요합니다. 클러스터와 동일한 VPC에서 기존 EC2 인스턴스를 사용하거나 새 인스턴스를 생성할 수 있습니다. 로컬 컴퓨터에서 다음 명령을 실행하여 Amazon DocumentDB 클러스터 `sample-cluster.node.us-east-1.docdb.amazonaws.com`에 대한 SSH 터널을 설정할 수 있습니다.

```
ssh -i "ec2Access.pem" -L 27017:sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017 ubuntu@ec2-34-229-221-164.compute-1.amazonaws.com -
N
```

-L 플래그는 로컬 포트를 전달할 때 사용됩니다. 이는 VPC 외부의 클라이언트에서 실행되는 모든 BI 도구에 연결하기 위한 전제 조건입니다. 위 단계를 실행한 후에는 선택한 BI 도구의 다음 단계로 넘어갈 수 있습니다.

SSH 터널링에 대한 자세한 내용은 SSH 터널을 사용하여 Amazon DocumentDB에 연결에 대한 설명서를 참조하세요.

타블로 데스크톱에서 Amazon DocumentDB에 연결

주제

- Amazon DocumentDB JDBC 드라이버 추가

- 타블로를 사용하여 Amazon DocumentDB에 연결하기 - SSH 터널

Amazon DocumentDB JDBC 드라이버 추가

Tableau Desktop에서 Amazon DocumentDB에 연결하려면 Amazon DocumentDB JDBC 드라이버와 DocumentDB Tableau 커넥터를 다운로드하여 설치해야 합니다.

1. Amazon DocumentDB JDBC 드라이버 리포지토리에서 Amazon DocumentDB JDBC 드라이버 JAR 파일을 다운로드하고 운영 체제에 따라 다음 디렉토리 중 하나에 복사합니다.
   - Windows – `C:\Program Files\Tableau\Drivers`
   - macOS – `~/Library/Tableau/Drivers`
2. Tableau Exchange 웹 사이트에서 DocumentDB Tableau 커넥터(TACO 파일)를 다운로드하여 내 Tableau 리포지토리/커넥터 디렉터리에 복사합니다.
   - Windows – `C:\Users\[user]\Documents\My Tableau Repository\Connectors`
   - macOS – `/Users/[user]/Documents/My Tableau Repository/Connectors`

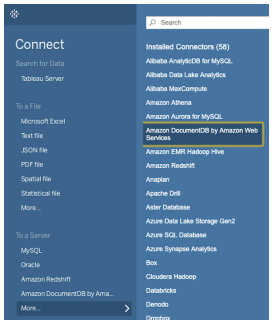
자세한 내용은 Tableau 설명서를 참조하세요.

> **Note**
>
> 최신 CA 인증서를 사용하는 경우 JDBC 드라이버를 v1.4.5(이 AWS GitHub 리포지토리에서 사용 가능)로 업그레이드해야 합니다.

타블로를 사용하여 Amazon DocumentDB에 연결하기 - SSH 터널

DocumentDB 클러스터의 VPC 외부에 있는 클라이언트 컴퓨터에서 Tableau에 연결하려면 아래 단계를 수행하기 전에 SSH 터널을 설정해야 합니다.

1. Tableau Desktop 애플리케이션을 실행합니다.
2. 연결 > 서버로 > 기타로 이동합니다.
3. 설치된 커넥터에서 Amazon Web Services의 Amazon DocumentDB를 선택합니다.

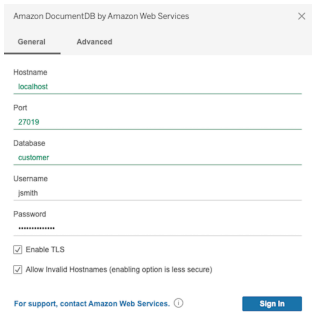


Tableau를 사용하여 Amazon DocumentDB에 연결 - 외부 SSH 터널

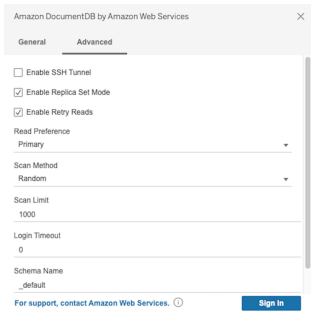
1. 필수 연결 파라미터 호스트 이름, 포트, 데이터베이스, 사용자 이름 및 암호를 입력합니다. 다음 예제의 연결 파라미터는 JDBC 연결 문자열과 동일합니다.

   ```
   jdbc:documentdb://localhost:27019/test?
   tls=true&tlsAllowInvalidHostnames=true&scanMethod=random&scanLimit=1000&logir
   ```

   성 컬렉션에서 사용자 이름 및 암호 파라미터가 별도로 전달됩니다. 연결 문자열 파라미터에 대한 자세한 내용은 Amazon DocumentDB JDBC 드라이버 github 설명서를 참조하세요.



2. (선택 사항) 고급 탭에서 고급 옵션을 찾을 수 있습니다.

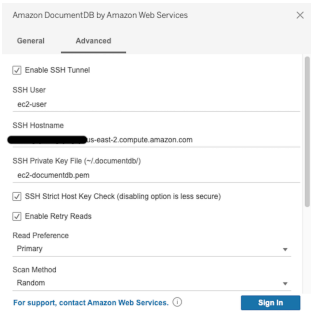


3. 로그인을 선택합니다.

Tableau를 사용하여 Amazon DocumentDB에 연결 - 내부 SSH 터널

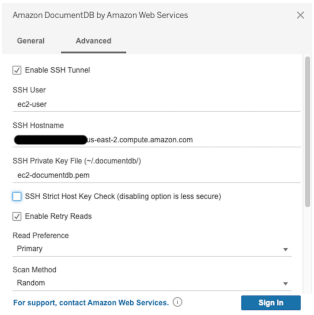
> **Note**
>
> 터미널을 사용하여 SSH 터널을 설정하지 않으려는 경우 Tableau GUI를 사용하여 JDBC 드라이버가 SSH 터널을 생성하는 데 기본적으로 사용할 EC2 인스턴스 세부 정보를 지정할 수 있습니다.

1. 고급 탭에서 SSH 터널 활성화 옵션을 선택하여 추가 속성을 검토하세요.



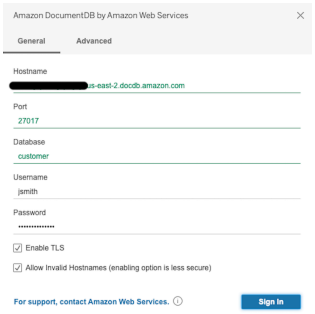
2. SSH 사용자, SSH 호스트 이름 및 SSH 개인 키 파일을 입력합니다.
3. (선택 사항) 알려진 호스트 파일에 대한 호스트 키 검사를 우회하는 SSH 엄격한 호스트 키 검사 옵션을 비활성화할 수 있습니다.

> **Note**
>
> 이 옵션을 비활성화하면 중간자 공격으로 이어질 수 있으므로 보안이 취약합니다.



4. 필수 파라미터 (호스트 이름, 포트, 데이터베이스, 사용자 이름 및 암호) 를 입력합니다.

> **Note**
>
> 내부 SSH 터널 옵션을 사용할 때는 로컬 호스트가 아닌 DocumentDB 클러스터 엔드포인트를 사용해야 합니다.



5. Sign In(로그인)을 선택합니다.

DB 비주얼라이저에서 Amazon DocumentDB에 연결

주제

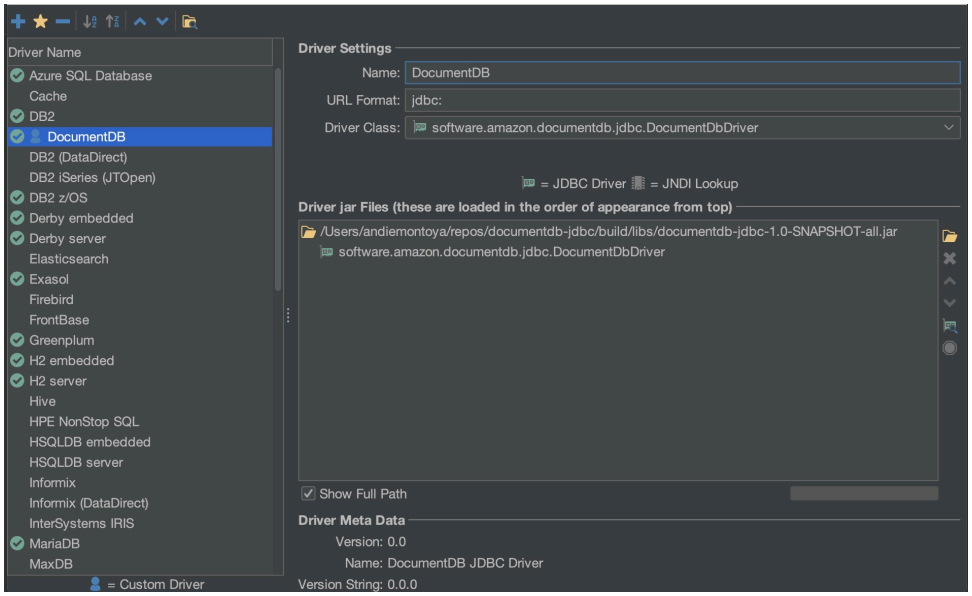
- Amazon DocumentDB JDBC 드라이버 추가
- DB 비주얼라이저를 사용하여 Amazon DocumentDB에 연결

Amazon DocumentDB JDBC 드라이버 추가

DbVisualizer에서 Amazon DocumentDB에 연결하려면 먼저 Amazon DocumentDB JDBC 드라이버를 가져와야 합니다.

1. DbVisualizer 애플리케이션을 시작하고 도구 > 드라이버 관리자... 메뉴 경로로 이동합니다.
2. +를 선택합니다 (또는 메뉴에서 드라이버 > 드라이버 생성을 선택).
3. 이름을 `DocumentDB`으로 설정합니다.
4. URL 형식을 다음으로 설정합니다. `jdbc:documentdb://<host>[:port]/<database>[?option=value[&option=value[...]]]`
5. 폴더 버튼을 선택한 다음 Amazon DocumentDB JDBC 드라이버 JAR 파일을 선택하고 열기 버튼을 선택합니다.

6. 드라이버 클래스 필드가 로 설정되어 있는지 확인하세요.
   `software.amazon.documentdb.jdbc.DocumentDbDriver` DocumentDB의 드라이버 관리자 설정은 다음 예시와 같습니다.



7. 대화 상자를 닫습니다. Amazon DocumentDB JDBC 드라이버가 설치되고 바로 사용할 수 있습니다.

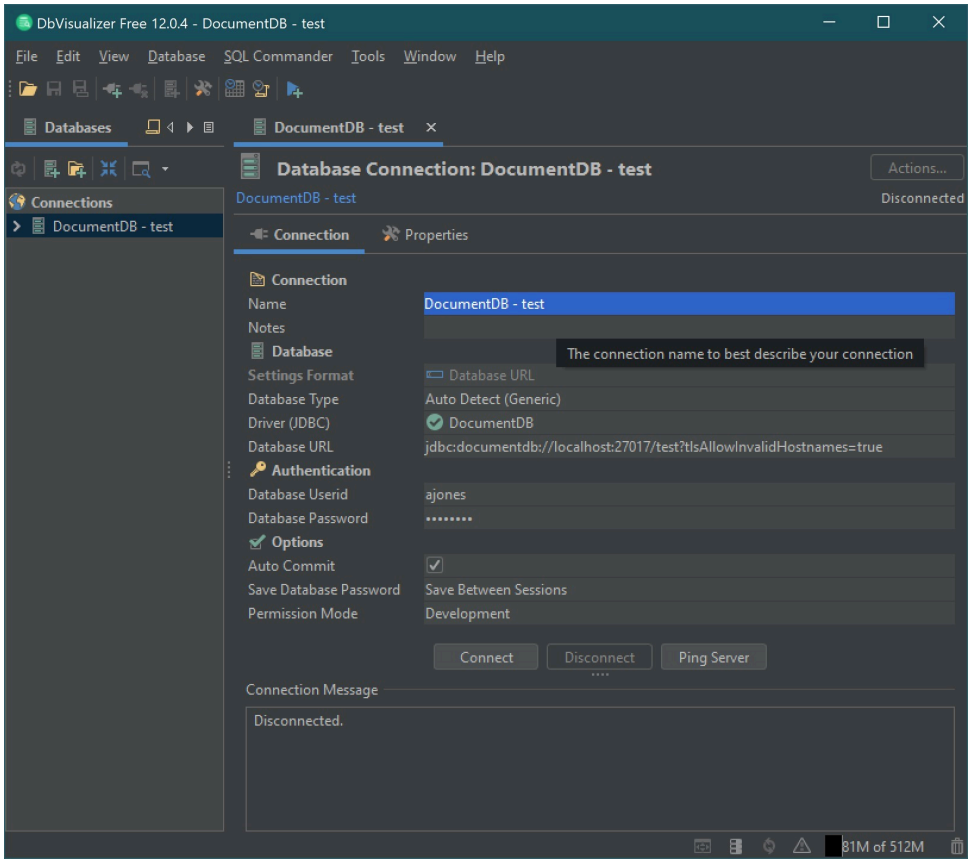
DB 비주얼라이저를 사용하여 Amazon DocumentDB에 연결

DB 비주얼라이저를 사용하여 Amazon DocumentDB에 연결

1. Amazon DocumentDB 클러스터의 VPC 외부에서 연결하는 경우 SSH 터널을 설정했는지 확인하세요.
2. 최상위 메뉴에서 데이터베이스 > 데이터베이스 연결 생성을 선택합니다.
3. 이름 필드에 설명이 포함된 이름을 입력합니다.
4. 드라이버 (JDBC) 를 이전 섹션에서 만든 DocumentDB 드라이버로 설정합니다.
5. 데이터베이스 URL을 JDBC 연결 문자열로 설정합니다.

   예: `jdbc:documentdb://localhost:27017/database?tlsAllowInvalidHostnames=true`
6. 데이터베이스 사용자 ID를 Amazon DocumentDB 사용자 ID로 설정합니다.
7. 데이터베이스 암호를 사용자 ID에 해당하는 암호로 설정합니다.

   데이터베이스 연결 대화 상자는 다음 대화 상자와 비슷합니다.



8. 연결을 선택합니다.

JDBC 자동 스키마 생성

Amazon DocumentDB는 문서 데이터베이스이므로 테이블과 스키마라는 개념이 없습니다. 하지만 Tableau와 같은 BI 도구에서는 연결하는 데이터베이스가 스키마를 제공할 것으로 예상합니다. 특히 JDBC 드라이버 연결에서 데이터베이스의 컬렉션에 대한 스키마를 가져와야 하는 경우 데이터베이스의 모든 컬렉션을 폴링합니다. 드라이버는 해당 컬렉션에 대한 스키마의 캐시된 버전이 이미 존재하는지 확인합니다. 캐시된 버전이 없는 경우 문서 컬렉션을 샘플링하고 다음 동작에 따라 스키마를 만듭니다.

주제

- 스키마 생성 제한
- 스캔 방법 옵션
- Amazon DocumentDB 데이터 유형
- 스칼라 문서 필드 매핑
- 객체 및 배열 데이터 유형 처리

스키마 생성 제한

DocumentDB JDBC 드라이버는 식별자 길이를 128자로 제한합니다. 스키마 생성기는 생성된 식별자(테이블 이름 및 열 이름)의 길이를 줄여 해당 한도에 맞는지 확인할 수 있습니다.

스캔 방법 옵션

연결 문자열 또는 데이터 소스 옵션을 사용하여 샘플링 동작을 수정할 수 있습니다.

- 스캔 방법= <option>
  - random - (기본값) - 샘플 문서가 무작위 순서로 반환됩니다.
  - IDForward - 샘플 문서가 ID 순으로 반환됩니다.
  - IDReverse - 샘플 문서가 ID의 역순으로 반환됩니다.
  - 모두 - 컬렉션의 모든 문서를 샘플링합니다.
- ScanLimit= <n>- 샘플링할 문서 수입니다. 값은 양의 정수여야 합니다. 기본값은 1000입니다. ScanMethod가 모두로 설정된 경우 이 옵션은 무시됩니다.

Amazon DocumentDB 데이터 유형

Amazon DocumentDB 서버는 다양한 MongoDB 데이터 유형을 지원합니다. 지원되는 데이터 유형 및 관련 JDBC 데이터 형식은 다음과 같습니다.

MongoDB 데이터 형식	DocumentDB에서 지원	JDBC 데이터 형식
이진 데이터	예	VARBINARY
부울	예	BOOLEAN
배정밀도 실수	예	DOUBLE
32비트 정수	예	INTEGER
64비트 정수	예	BIGINT
문자열	예	VARCHAR
ObjectId	예	VARCHAR
Date	예	TIMESTAMP

MongoDB 데이터 형식	DocumentDB에서 지원	JDBC 데이터 형식
Null	예	VARCHAR
정규식	예	VARCHAR
타임스탬프	예	VARCHAR
MinKey	예	VARCHAR
MaxKey	예	VARCHAR
객체	예	가상 테이블
배열	예	가상 테이블
Decimal128	아니요	DECIMAL
JavaScript	아니요	VARCHAR
JavaScript (범위 포함)	아니요	VARCHAR
정의되지 않음	아니요	VARCHAR
Symbol	아니요	VARCHAR
DB 포인터 (4.0 이상)	아니요	VARCHAR

스칼라 문서 필드 매핑

컬렉션에서 문서 샘플을 스캔할 때 JDBC 드라이버는 컬렉션의 샘플을 나타내는 스키마를 하나 이상 만듭니다. 일반적으로 문서의 스칼라 필드는 테이블 스키마의 열에 매핑됩니다. 예를 들어 team이라는 컬렉션과 단일 `{ "_id" : "112233", "name" : "Alastair", "age": 25 }` 문서에서는 다음과 같이 스키마에 매핑됩니다.

테이블 이름	열 이름	데이터 형식	Key(키)
팀	team id	VARCHAR	PK
팀	이름	VARCHAR	

테이블 이름	열 이름	데이터 형식	Key(키)
팀	나이	INTEGER	

데이터 유형 충돌 촉진

샘플링된 문서를 스캔할 때 필드의 데이터 유형이 문서마다 일치하지 않을 수 있습니다. 이 경우 JDBC 드라이버는 JDBC 데이터 유형을 샘플 문서의 모든 데이터 유형에 적합한 공통 데이터 유형으로 승격시킵니다.

예:

```
{
"_id" : "112233",
"name" : "Alastair", "age" : 25
}

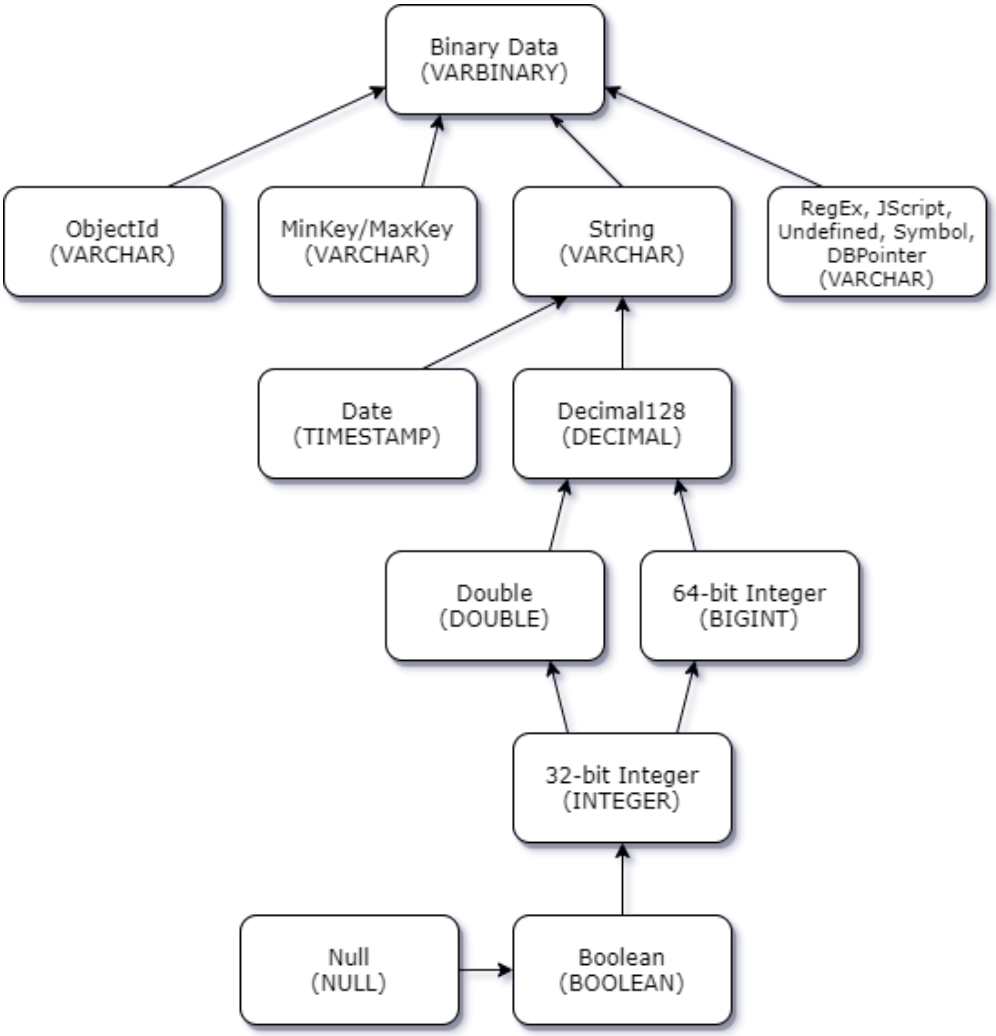
{
"_id" : "112244",
"name" : "Benjamin",
"age" : "32"
}
```

첫 번째 문서에서는 age 필드의 유형이 32비트 정수이고 두 번째 문서에서는 문자열입니다. 여기서 JDBC 드라이버는 JDBC 데이터 유형을 VARCHAR로 승격시켜 두 데이터 유형 중 하나를 발견했을 때 이를 처리합니다.

테이블 이름	열 이름	데이터 형식	Key(키)
팀	team id	VARCHAR	PK
팀	이름	VARCHAR	
팀	나이	VARCHAR	

스칼라-스칼라 충돌 프로모션

다음 다이어그램은 스칼라-스칼라 데이터 유형 충돌을 해결하는 방법을 보여줍니다.



스칼라-복소수 형식 충돌 승격

스칼라-스칼라 형식 충돌과 마찬가지로, 서로 다른 문서에 있는 동일한 필드에서도 복소수 (배열 및 객체) 와 스칼라 (정수, 부울 등) 간에 데이터 유형이 충돌할 수 있습니다. 이러한 모든 충돌은 해당 필드에 대해 VARCHAR로 해결 (승격) 됩니다. 이 경우 배열 및 객체 데이터가 JSON 표현으로 반환됩니다.

임베디드 어레이 - 문자열 필드 충돌 예제:

```
{
   "_id":"112233",
   "name":"George Jackson",
   "subscriptions":[
      "Vogue",
      "People",
      "USA Today"
   ]
```

```
}
{
   "_id":"112244",
   "name":"Joan Starr",
   "subscriptions":1
}
```

앞의 예제는 customer2 테이블의 스키마에 매핑됩니다.

테이블 이름	열 이름	데이터 형식	Key(키)
고객 2	customer2 id	VARCHAR	PK
고객 2	이름	VARCHAR	
고객 2	구독	VARCHAR	

and the customer1_subscriptions 가상 테이블:

테이블 이름	열 이름	데이터 형식	Key(키)
customer1_subscriptions	customer1 id	VARCHAR	PK/FK
customer1_subscriptions	subscriptions_index_lvl0	BIGINT	PK
customer1_subscriptions	값	VARCHAR	
customer_address	city	VARCHAR	
customer_address	리전	VARCHAR	
customer_address	country	VARCHAR	
customer_address	code	VARCHAR	

객체 및 배열 데이터 유형 처리

지금까지는 스칼라 데이터형이 매핑되는 방식만 설명했습니다. 객체 및 배열 데이터 유형은 (현재) 가상 테이블에 매핑되어 있습니다. JDBC 드라이버는 문서의 개체 또는 배열 필드를 나타내는 가상 테이블을 만듭니다. 매핑된 가상 테이블의 이름은 원본 컬렉션의 이름 뒤에 밑줄 문자("_")로 구분된 필드 이름을 연결합니다.

기본 테이블의 프라이머리 키("_id")는 새 가상 테이블에서 새 이름을 사용하며 연결된 기본 테이블에 외래 키로 제공됩니다.

내장된 배열 유형 필드의 경우 배열의 각 수준에서 배열에 대한 인덱스를 나타내는 인덱스 열이 생성됩니다.

내장된 객체 필드 예제

문서의 개체 필드의 경우 JDBC 드라이버가 가상 테이블에 매핑을 만듭니다.

```
{
   "Collection: customer",
   "_id":"112233",
   "name":"George Jackson",
   "address":{
      "address1":"123 Avenue Way",
      "address2":"Apt. 5",
      "city":"Hollywood",
      "region":"California",
      "country":"USA",
      "code":"90210"
   }
}
```

앞의 예제는 고객 테이블의 스키마에 매핑됩니다.

테이블 이름	열 이름	데이터 형식	Key(키)
customer	customer id	VARCHAR	PK
customer	이름	VARCHAR	

및 the customer_address 가상 테이블:

테이블 이름	열 이름	데이터 형식	Key(키)
customer_address	customer id	VARCHAR	PK/FK
customer_address	address1	VARCHAR	
customer_address	address2	VARCHAR	
customer_address	city	VARCHAR	
customer_address	리전	VARCHAR	
customer_address	country	VARCHAR	
customer_address	code	VARCHAR	

임베디드 어레이 필드 예제

문서의 배열 필드의 경우 JDBC 드라이버가 가상 테이블로의 매핑도 생성합니다.

```
{
   "Collection: customer1",
   "_id":"112233",
   "name":"George Jackson",
   "subscriptions":[
      "Vogue",
      "People",
      "USA Today"
   ]
}
```

앞의 예제는 customer1 테이블의 스키마에 매핑됩니다.

테이블 이름	열 이름	데이터 형식	Key(키)
customer1	customer1 id	VARCHAR	PK
customer1	이름	VARCHAR	

and the customer1_subscriptions 가상 테이블:

테이블 이름	열 이름	데이터 형식	Key(키)
customer1_subscriptions	customer1 id	VARCHAR	PK/FK
customer1_subscriptions	subscriptions_index_lvl0	BIGINT	PK
customer1_subscriptions	값	VARCHAR	
customer_address	city	VARCHAR	
customer_address	리전	VARCHAR	
customer_address	country	VARCHAR	
customer_address	code	VARCHAR	

SQL 지원 및 제한 사항

Amazon DocumentDB JDBC 드라이버는 SQL-92 하위 집합과 일부 일반 확장을 지원하는 읽기 전용 드라이버입니다. 자세한 내용은 SQL 제한 설명서 및 JDBC 제한 설명서를 참조하세요.

문제 해결

Amazon DocumentDB JDBC 드라이버 사용에 문제가 있는 경우 문제 해결 안내서를 참조하세요.

Amazon DocumentDB ODBC 드라이버를 사용하여 연결

Amazon DocumentDB용 ODBC 드라이버는 개발자를 위한 SQL 관계형 인터페이스를 제공하며 Power BI Desktop 및 Microsoft Excel과 같은 BI 도구에서 연결할 수 있도록 합니다.

자세한 내용은 GitHub의 Amazon DocumentDB ODBC 드라이버 설명서를 참조하세요.

주제

- 시작하기

- 윈도우에서 Amazon DocumentDB ODBC 드라이버 설정하기
- 마이크로소프트 엑셀에서 Amazon DocumentDB에 연결
- 마이크로소프트 파워 BI 데스크톱에서 Amazon DocumentDB에 연결
- 자동 스키마 생성
- SQL 지원 및 제한 사항
- 문제 해결

시작하기

1단계. Amazon DocumentDB 클러스터 생성

Amazon DocumentDB 클러스터가 아직 없는 경우 여러 가지 방법으로 시작할 수 있습니다.

> **Note**
>
> Amazon DocumentDB는 가상 프라이빗 클라우드(VPC) 전용 서비스입니다. 클러스터의 VPC 외부에 있는 로컬 시스템에서 연결하는 경우 Amazon EC2 인스턴스에 대한 SSH 연결을 생성해야 합니다. 자세한 내용은 Amazon EC2를 사용하여 연결 단원을 참조하십시오. SSH 터널링에 대한 자세한 내용은 SSH 터널을 사용하여 Amazon DocumentDB에 연결을 참조하세요.

2단계. JRE 또는 JDK 설치

BI 애플리케이션에 따라 컴퓨터에 64비트 JRE 또는 JDK 설치 버전 8 이상이 설치되어 있는지 확인해야 할 수 있습니다. Java SE 런타임 환경 8은 여기에서 다운로드할 수 있습니다.

3단계. Amazon DocumentDB ODBC 드라이버 다운로드

여기에서 Amazon DocumentDB ODBC 드라이버를 다운로드하세요. 적절한 설치 프로그램(예: documentdb-odbc-1.0.0.msi)을 선택합니다. 설치 가이드를 따르세요.

4단계. SSH 터널을 사용하여 Amazon DocumentDB에 연결

Amazon DocumentDB 클러스터는 Amazon Virtual Private Cloud(Amazon VPC)에 배포됩니다. Amazon EC2 인스턴스 또는 동일한 Amazon VPC에 배포된 다른 AWS 서비스에서 직접 액세스할 수 있습니다. 또한 Amazon DocumentDB는 Amazon EC2 인스턴스 또는 VPCs 피어링을 통해 동일한 AWS 리전 리전 또는 다른 리전의 다른 VPC에 있는 다른 AWS 서비스에서 액세스할 수 있습니다.

그러나 사용 사례에서 사용자 또는 사용자의 애플리케이션이 클러스터의 VPC 외부에서 Amazon DocumentDB 리소스에 액세스해야 한다고 가정합니다. 이는 대부분의 사용자가 Amazon DocumentDB 클러스터와 동일한 VPC에 있는 VM에서 애플리케이션을 실행하지 않는 경우입니다. VPC 외부에서 연결하는 경우 SSH 터널링(포트 포워딩이라고도 함)을 사용하여 Amazon DocumentDB 리소스에 액세스할 수 있습니다.

SSH 터널을 만들려면 Amazon DocumentDB 클러스터와 동일한 Amazon VPC에서 실행 중인 Amazon EC2 인스턴스가 필요합니다. 클러스터와 동일한 VPC에서 기존 EC2 인스턴스를 사용하거나 새 인스턴스를 생성할 수 있습니다. 로컬 컴퓨터에서 다음 명령을 사용하여 SSH 터널을 Amazon DocumentDB 클러스터 `sample-cluster.node.us-east-1.docdb.amazonaws.com`으로 설정할 수 있습니다.

```
ssh -i "ec2Access.pem" -L 27017:sample-cluster.node.us-
east-1.docdb.amazonaws.com:27017 ubuntu@ec2-34-229-221-164.compute-1.amazonaws.com -
N
```

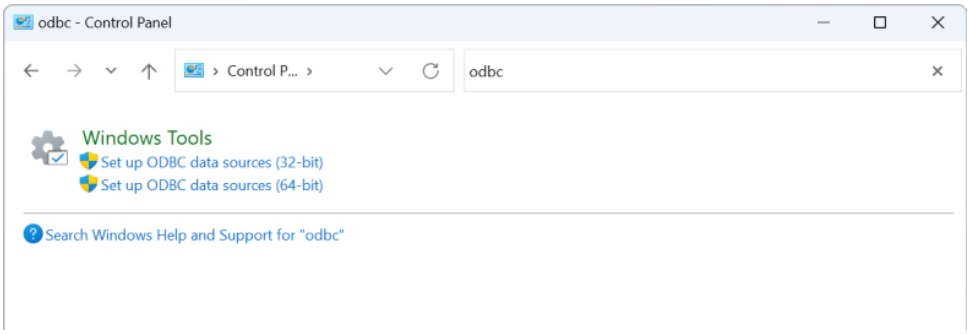
`-L` 플래그는 로컬 포트를 전달하는 데 사용됩니다. 이는 VPC 외부의 클라이언트에서 실행되는 모든 BI 도구에 연결하기 위한 전제 조건입니다. 위 단계를 실행한 후에는 선택한 BI 도구의 다음 단계로 넘어갈 수 있습니다.

SSH 터널링에 대한 자세한 내용은 SSH 터널을 사용하여 Amazon DocumentDB에 연결에 대한 설명서를 참조하세요.

윈도우에서 Amazon DocumentDB ODBC 드라이버 설정하기

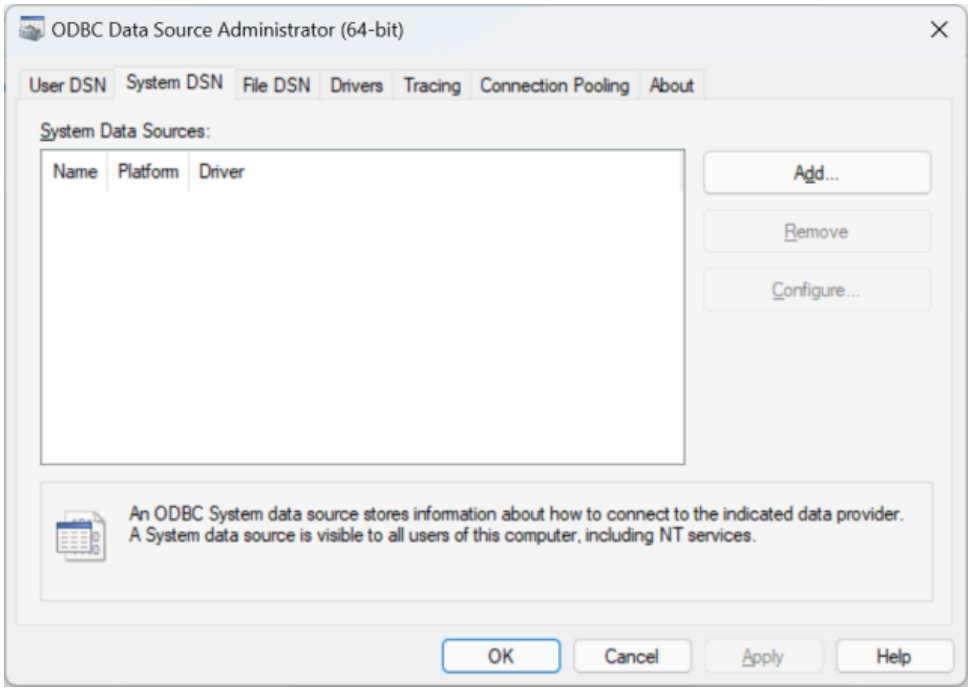
다음 절차를 사용하여 Windows에 Amazon DocumentDB ODBC 드라이버를 설정합니다.

1. Windows에서 제어판을 열고 ODBC를 검색합니다 (또는 메뉴에서 Windows 도구 > ODBC 데이터 소스 (32비트) 또는 ODBC 데이터 소스 (64비트) 를 선택).

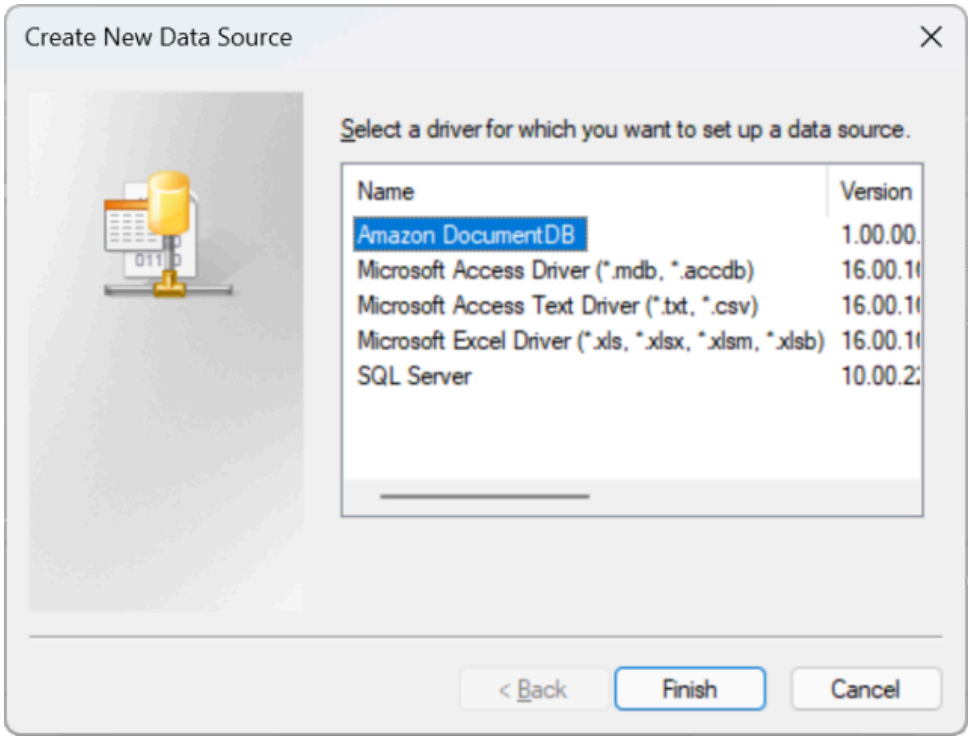


2. 적절한 ODBC 드라이버 데이터 원본 관리자 선택: 설치되어 있는 경우 32비트 버전을 선택하고, 그렇지 않으면 64비트 버전을 선택하세요.

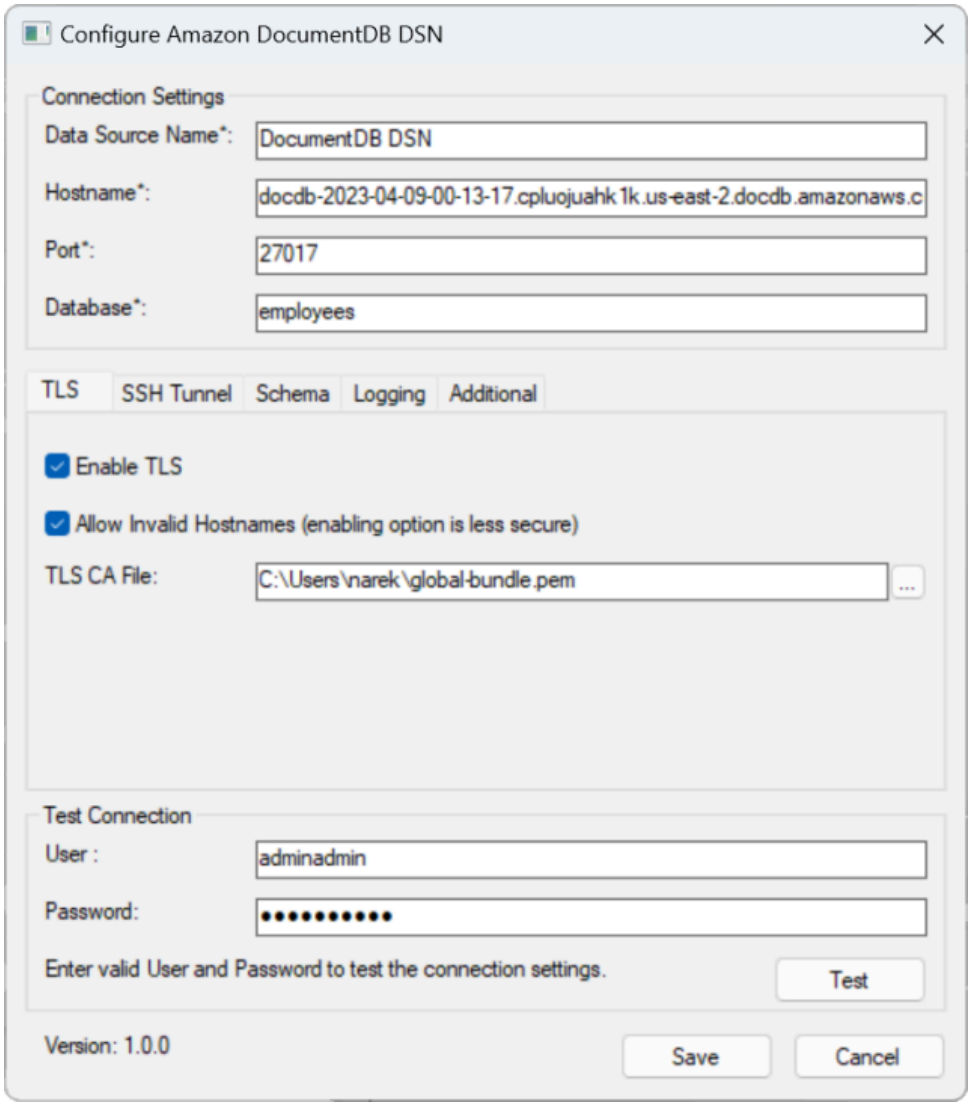
3. Sytem DSN 탭을 선택한 다음 추가...를 선택하여 새 DSN을 추가합니다.



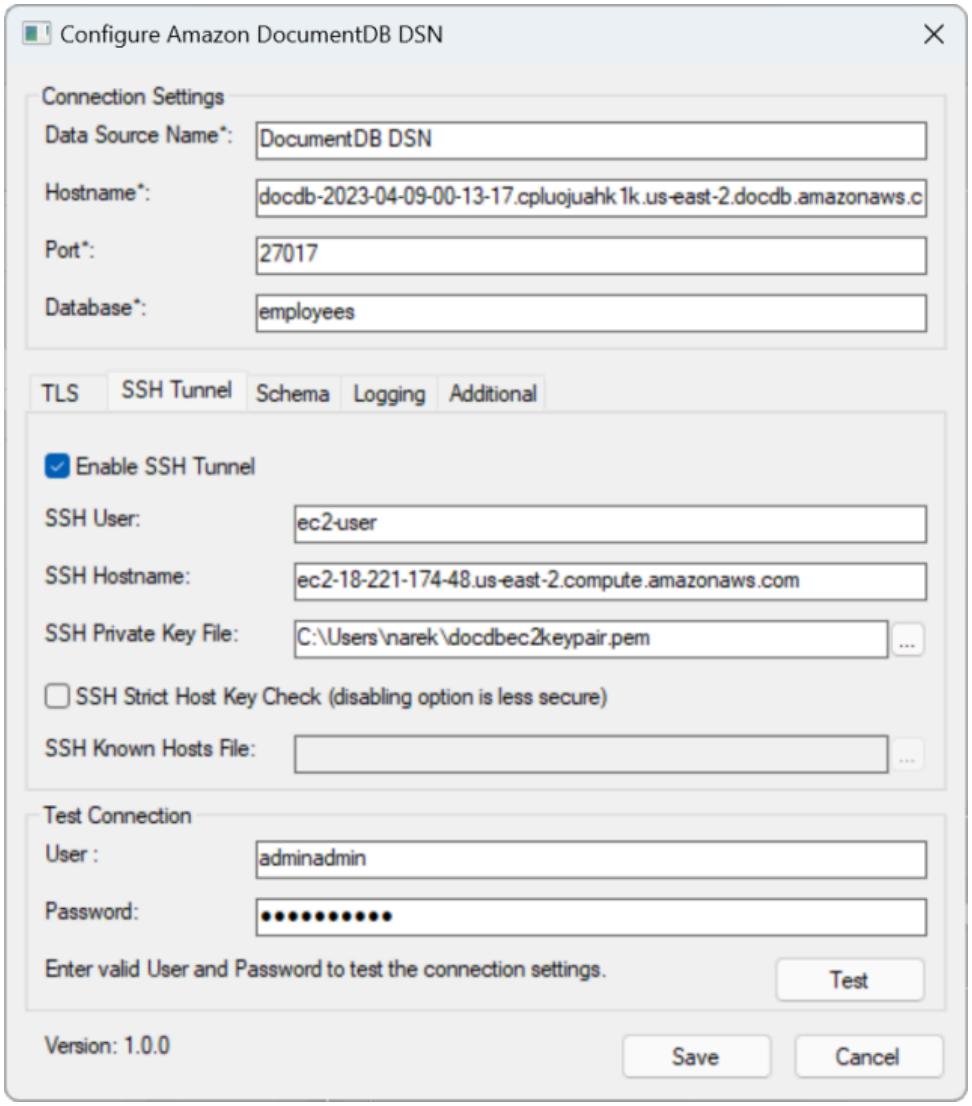
4. 데이터 소스 드라이버 목록에서 Amazon DocumentDB를 선택합니다.



5. Amazon DocumentDB DSN 구성 대화 상자에서 연결 설정, TLS 탭 및 연결 테스트 필드를 완료한 다음 저장을 선택합니다.



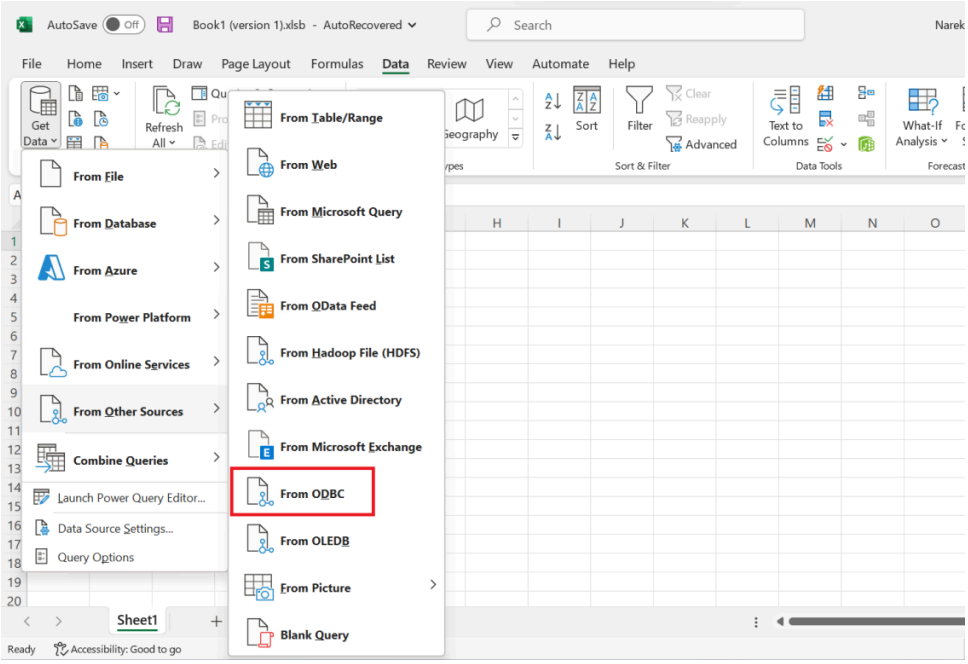
6. 선택한 EC2 인스턴스에 대한 SSH 터널링 방법에 따라 연결 세부 정보가 달라지므로 Windows 양식을 정확하게 작성해야 합니다. 여기에서 SSH 터널링 방법을 참조하세요. 각 속성에 대한 자세한 내용은 연결 문자열 구문 및 옵션을 참조하세요.



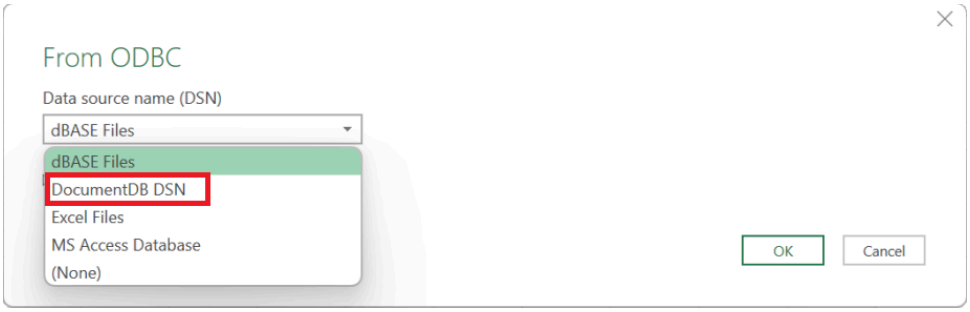
Windows에서 Amazon DocumentDB ODBC 드라이버를 구성하는 방법에 대한 자세한 내용은 여기를 참조하세요.

마이크로소프트 엑셀에서 Amazon DocumentDB에 연결

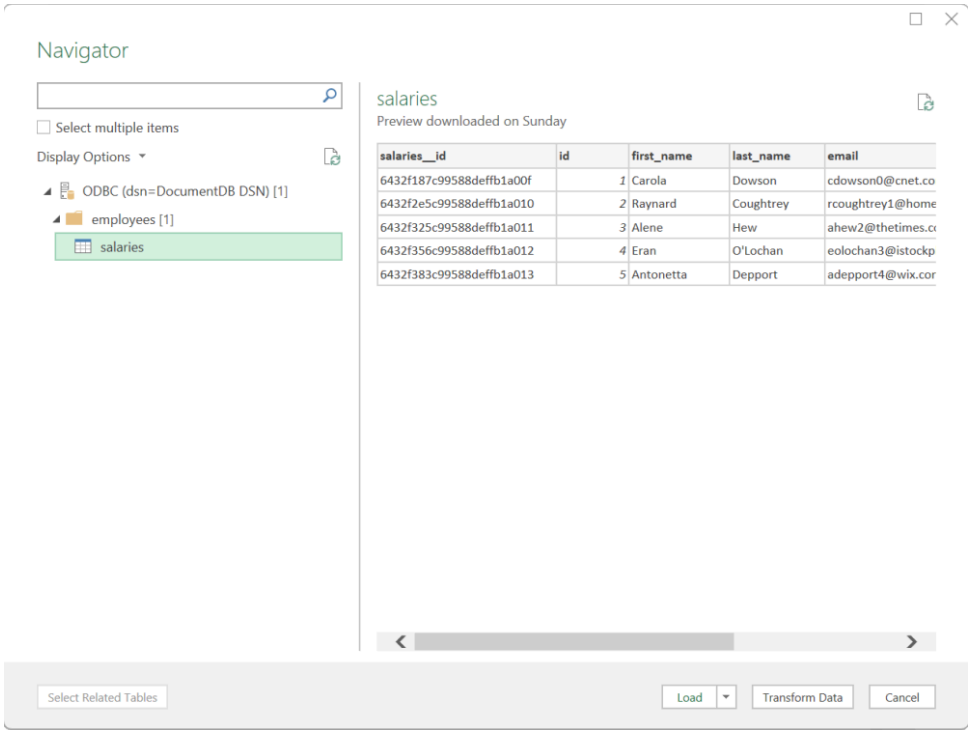
1. Amazon DocumentDB 드라이버가 올바르게 설치 및 구성되었는지 확인하세요. 자세한 내용은 윈도우에서 Amazon DocumentDB ODBC 드라이버 설정하기 단원을 참조하십시오.
2. 마이크로소프트 엑셀을 실행합니다.
3. 데이터 > 데이터 가져오기 > 기타 소스에서로 이동합니다.
4. ODBC에서 선택:



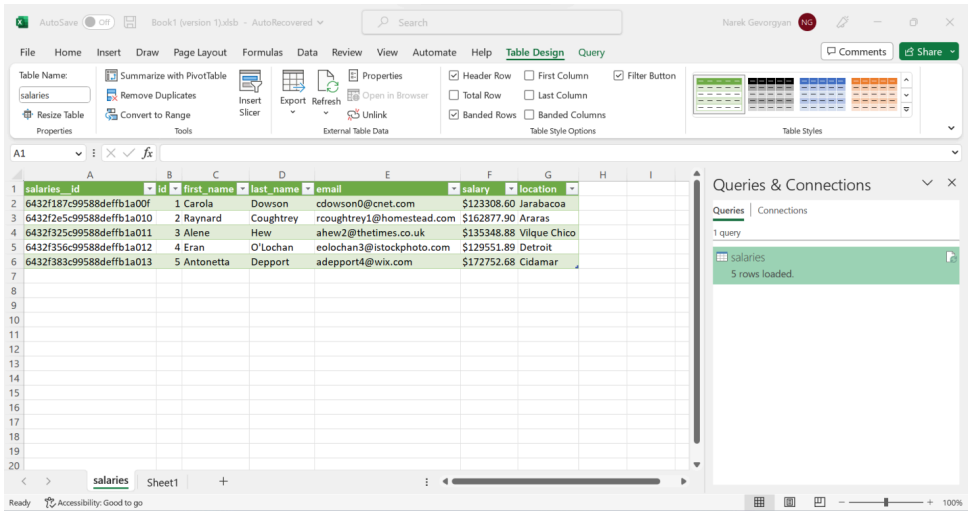
5. Amazon DocumentDB와 연결된 데이터 소스 이름 (DSN) 드롭다운 메뉴에서 데이터 소스를 선택합니다.



6. 데이터를 Excel에 로드할 컬렉션을 선택합니다.



7. 데이터를 Excel에 로드합니다.



마이크로소프트 파워 BI 데스크톱에서 Amazon DocumentDB에 연결

주제

- 사전 조건
- Microsoft Power BI 데스크톱 사용자 지정 커넥터 추가
- Amazon DocumentDB 사용자 지정 커넥터를 사용하여 연결
- 마이크로소프트 파워 BI 게이트웨이 구성

사전 조건

시작하기 전에 Amazon DocumentDB ODBC 드라이버가 올바르게 설치되어 있는지 확인하세요.

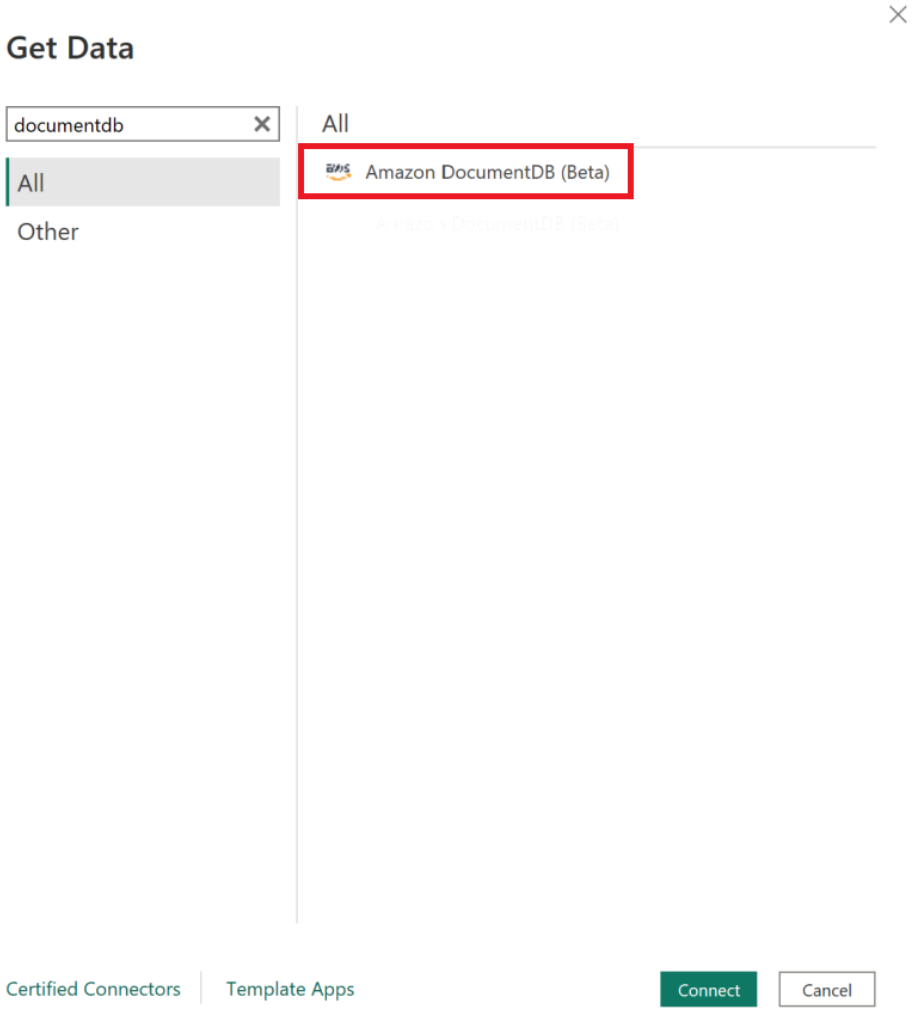
Microsoft Power BI 데스크톱 사용자 지정 커넥터 추가

`AmazonDocumentDBConnector.mez`파일을 `<User>\Documents\Power BI Desktop\Custom Connectors\` 폴더 (또는 OneDrive를 사용하는 `<User>\OneDrive\Documents\Power BI Desktop\Custom Connectors` 경우) 에 복사합니다. 이렇게 하면 Power BI에서 사용자 지정 커넥터에 액세스할 수 있습니다. Power BI 데스크톱에 대한 커넥터는 여기에서 구할 수 있습니다. Power BI Desktop을 다시 시작하여 커넥터가 로드되었는지 확인합니다.

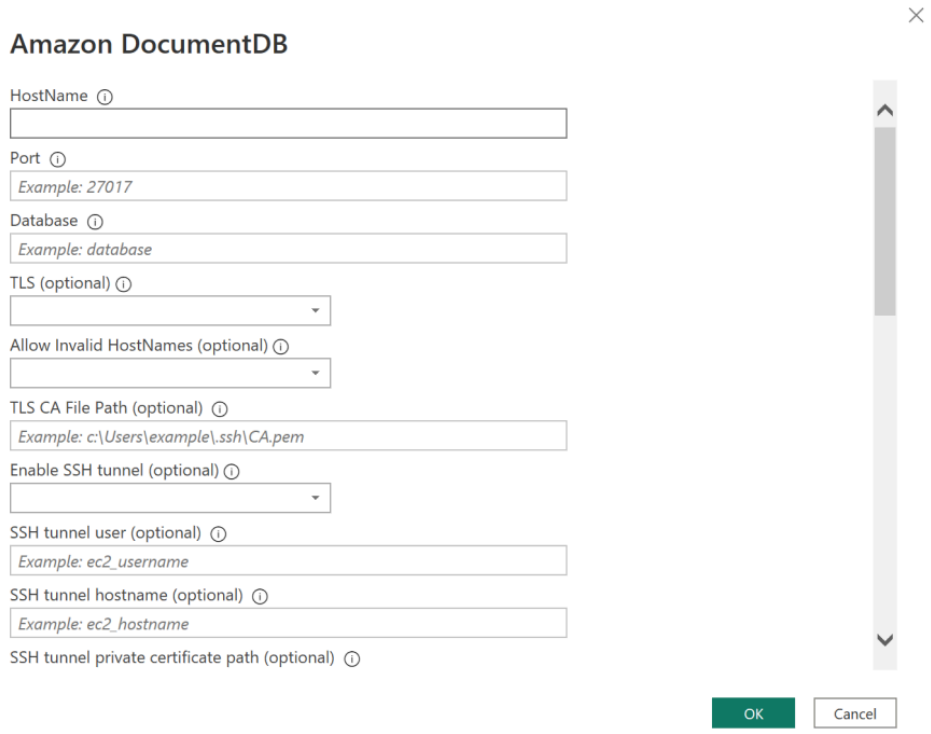
> **Note**
>
> 사용자 지정 커넥터는 인증을 위한 Amazon DocumentDB 사용자 이름 및 암호만 지원합니다.

Amazon DocumentDB 사용자 지정 커넥터를 사용하여 연결

1. 데이터 가져오기에서 Amazon DocumentDB(베타)를 선택하고 연결을 선택합니다. 타사 서비스 사용에 대한 경고가 표시되면 계속을 선택합니다.



2. Amazon DocumentDB 클러스터에 연결하는 데 필요한 모든 정보를 입력한 다음 확인을 선택합니다.



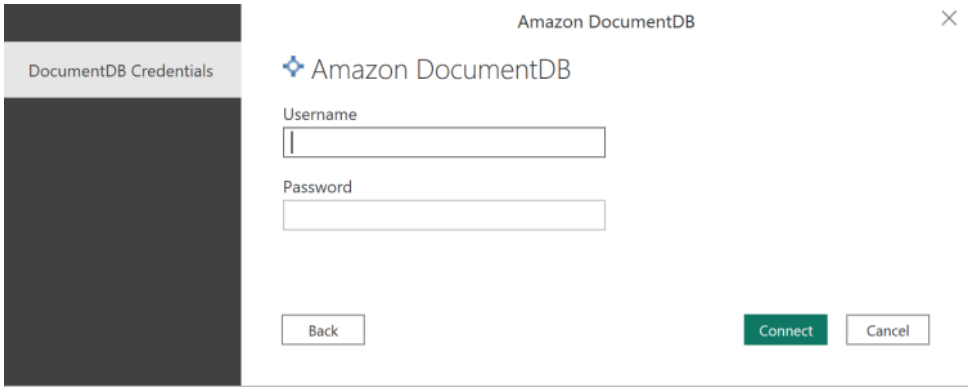
> **Note**
>
> DSN 설정에 필요한 정보를 이미 제공한 경우 ODBC 드라이버의 데이터 소스 이름 (DSN) 구성에 따라 SSH 연결 세부 정보 화면이 표시되지 않을 수 있습니다.

3. 데이터 연결 모드를 선택합니다.

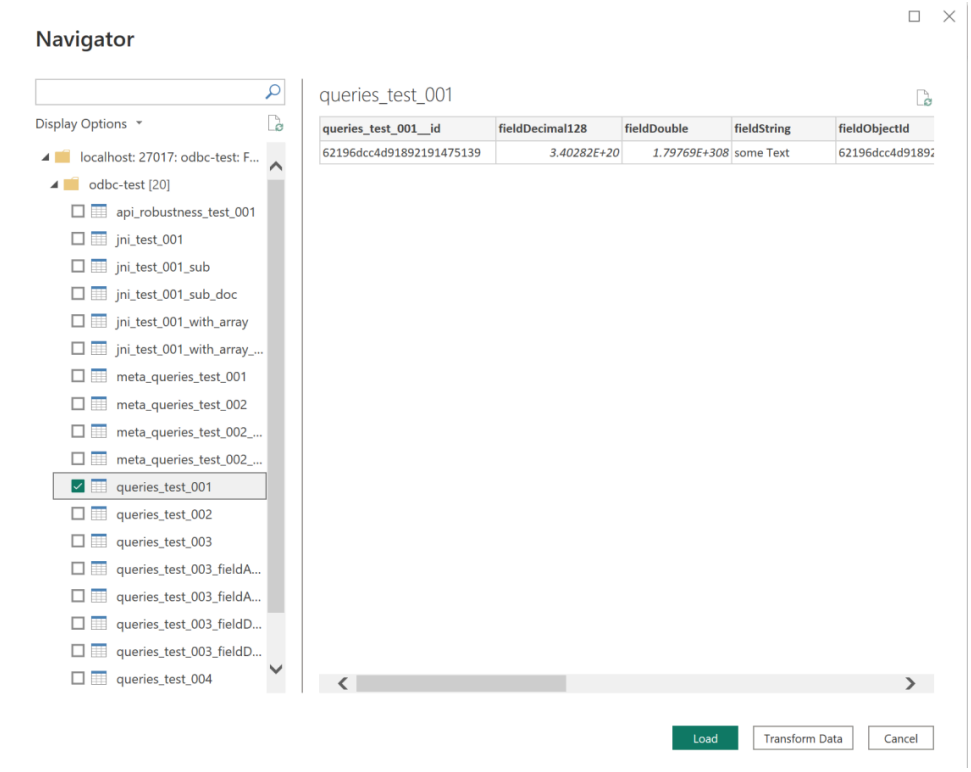
   - 가져오기 - 모든 데이터를 로드하고 정보를 디스크에 저장합니다. 데이터 업데이트를 표시하려면 데이터를 새로 고치고 다시 로드해야 합니다.
   - 직접 쿼리 - 데이터를 로드하지는 않지만 데이터에 대한 실시간 쿼리를 수행합니다. 즉, 데이터 업데이트를 표시하기 위해 데이터를 새로 고치고 다시 로드할 필요가 없습니다.

> **Note**
>
> 매우 큰 데이터셋을 사용하는 경우 모든 데이터를 가져오는 데 시간이 더 오래 걸릴 수 있습니다.

4. 이 데이터 원본에 처음 연결하는 경우 인증 유형을 선택하고 메시지가 표시되면 자격 증명을 입력합니다. 그런 다음 연결을 선택합니다.



5. 탐색기 대화 상자에서 원하는 데이터베이스 테이블을 선택한 다음 로드를 선택하여 데이터를 로드하거나 데이터 변환을 선택하여 데이터 변환을 계속합니다.



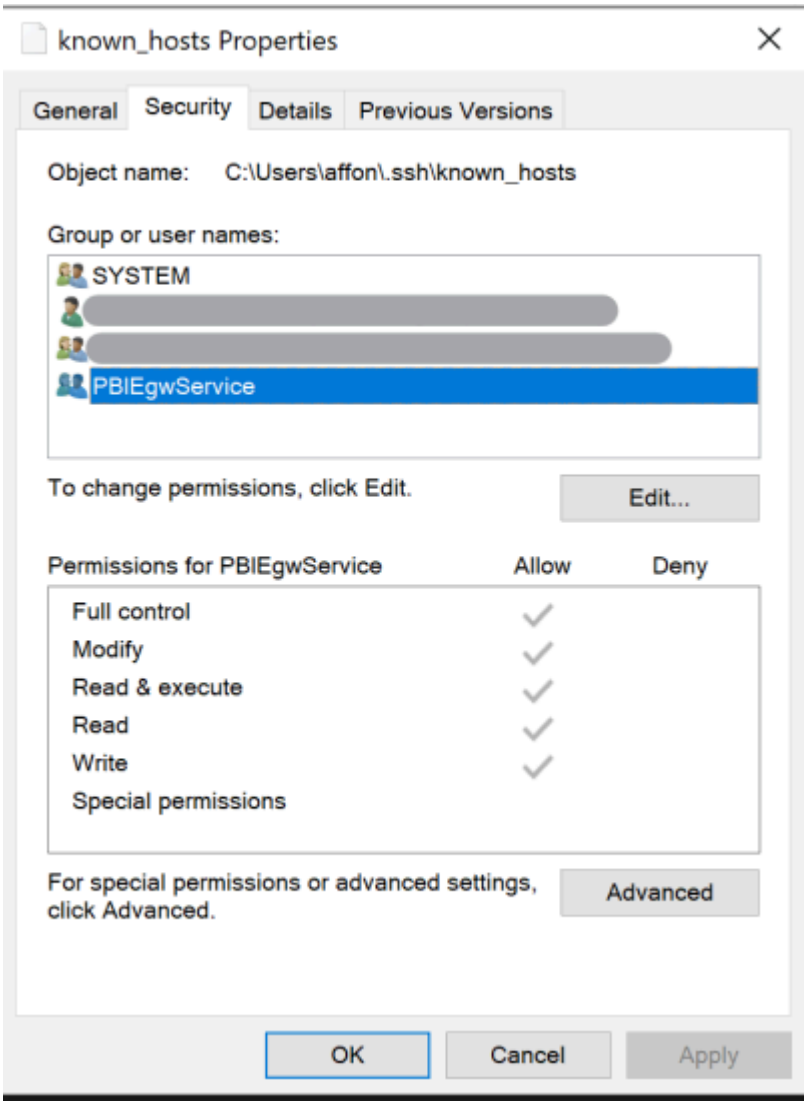
> **Note**
>
> 연결하면 데이터 소스 설정이 저장됩니다. 설정을 수정하려면 데이터 변환 > 데이터 소스 설정을 선택합니다.

마이크로소프트 파워 BI 게이트웨이 구성

사전 조건:

- 사용자 지정 커넥터가 Power BI 게이트웨이와 호환되는지 확인하세요.
- Power BI Gateway가 설치된 컴퓨터의 시스템 탭에 있는 ODBC 데이터 원본에 ODBC DSN이 생성되었는지 확인하세요.

내부 SSH 터널 기능을 사용하는 경우 파일은 `known_hosts` Power BI 서비스 계정이 액세스할 수 있는 위치에 있어야 합니다.



> **Note**
>
> 이는 인증 기관 (CA) 인증서 파일 (pem 파일) 과 같이 Amazon DocumentDB 클러스터에 대한 연결을 설정하는 데 필요한 모든 파일에도 적용됩니다.

자동 스키마 생성

ODBC 드라이버는 JNI (자바 네이티브 인터페이스) 를 통해 Amazon DocumentDB JDBC 드라이버를 활용하므로 자동 스키마 생성 기능이 JDBC 드라이버에서도 비슷하게 작동합니다. 자동 스키마 생성에 대한 자세한 내용은 섹션을 참조하세요JDBC 자동 스키마 생성. ODBC 드라이버 아키텍처에 대한 자세한 내용은 GitHub의 Amazon DocumentDB ODBC 드라이버 아키텍처를 참조하세요.

SQL 지원 및 제한 사항

Amazon DocumentDB ODBC 드라이버는 SQL-92 하위 집합과 일부 일반 확장을 지원하는 읽기 전용 드라이버입니다. 자세한 내용은 ODBC 지원 및 제한 설명서를 참조하세요.

문제 해결

Amazon DocumentDB ODBC 드라이버 사용에 문제가 있는 경우 문제 해결 안내서를 참조하세요.

Amazon DocumentDB를 사용한 프로그램

서비스는 JSON 스키마 검증을 지원합니다.

주제

- Amazon DocumentDB Java 프로그래밍 설명서
- JSON 스키마 검증 사용

Amazon DocumentDB Java 프로그래밍 설명서

이 포괄적인 설명서에서는 MongoDB의 Java 드라이버를 사용하여 Amazon DocumentDB로 작업하기 위한 자세한 안내를 제공하며, 데이터베이스 운영 및 관리의 필수 측면을 다룹니다.

주제

- 소개
- 사전 조건
- 이 설명서의 데이터 모델
- MongoDB Java 드라이버를 사용하여 Amazon DocumentDB에 연결
- Java를 사용하여 Amazon DocumentDB에서 CRUD 작업 수행
- Java를 사용한 Amazon DocumentDB의 인덱스 관리
- Amazon DocumentDB 및 Java를 사용한 이벤트 기반 프로그래밍

소개

이 설명서에서는 연결부터 시작하여 MongoDB Java 드라이버를 통해 DocumentDB 클러스터에 대한 보안 연결을 설정하는 방법에 대해 설명합니다. 연결 문자열 구성 요소, SSL/TLS 구현, IAM 인증 및 연결 풀링을 비롯한 다양한 연결 옵션과 강력한 오류 처리 전략에 대해 자세히 설명합니다.

이 설명서는 생성, 읽기, 업데이트, 삭제(CRUD) 작업 섹션에서 문서 조작에 대해 철저히 다루며, 단일 작업과 대량 작업을 모두 사용하여 문서를 생성, 읽기, 업데이트 및 삭제하는 방법을 보여줍니다. 필터, 쿼리 및 다양한 작업 옵션의 사용에 대해 설명하고, 오류 처리에 대한 모범 사례를 강조하고, 신뢰성 향상을 위한 재시도 로직을 구현합니다. 또한 이 설명서에서는 인덱스 관리에 대해 광범위하게 다루며 단일 필드, 복합, 희소, 텍스트 인덱스를 비롯한 다양한 인덱스 유형의 생성 및 유지 관리에 대해 자세히 설명합니다. 적절한 인덱스 선택 및 `explain()` 함수 사용을 통해 쿼리 실행 계획을 분석하여 쿼리 성능을 최적화하는 방법을 설명합니다.

마지막 섹션에서는 Amazon DocumentDB의 변경 스트림을 사용한 이벤트 기반 프로그래밍에 중점을 두고 Java 애플리케이션에서 실시간 데이터 변경 모니터링을 구현하는 방법을 보여줍니다. 여기에는 변경 스트림 커서 구현, 연속 작업을 위한 재개 토큰 처리, 과거 데이터 처리를 위한 시간 기반 작업이 포함됩니다. 설명서 전체에서 실용적인 코드 예제와 모범 사례가 제공되므로 Amazon DocumentDB로 강력한 Java 애플리케이션을 구축할 때 매우 유용한 리소스가 됩니다.

사전 조건

시작하기 전에 다음 항목이 준비되었는지 확인합니다.

- Amazon DocumentDB 클러스터 AWS 계정 가 구성된 . 클러스터 설정은 AWS 데이터베이스 블로그의 Amazon EC2를 사용하여 Amazon DocumentDB 시작하기Amazon EC2를 참조하세요.
- 설치된 Java Development Kit(JDK)(이 설명서에서는 Amazon Corretto 21을 사용할 예정임).
- 종속성 관리를 위한 Maven.

이 설명서의 데이터 모델

이 설명서의 모든 예제 코드는 'Restaurants' 컬렉션이 있는 'ProgGuideData 테스트 데이터베이스에 대한 연결을 가정합니다. 이 설명서의 모든 샘플 코드는 레스토랑 목록 시스템 이하에서 작동합니다. 이 시스템의 문서는 다음과 같습니다.

```
{
    "_id": "ab6ad8f119b5bca3efa2c7ae",
    "restaurantId": "REST-CRT9BL",
    "name": "Thai Curry Palace",
    "description": "Amazing Restaurant, must visit",
    "cuisine": "Thai",
    "address": {
        "street": "914 Park Street",
        "city": "Bryan",
        "state": "AL",
        "zipCode": "96865",
```

```
        "location": {
            "type": "Point",
            "coordinates": [-25.4619, 8.389]
        }
    },
    "contact": {
        "phone": "(669) 915-9056 x6657"
    },
    "rating": {
        "average": 3.4,
        "totalReviews": 275
    },
    "priceRange": "$",
    "menu": [{
        "category": "Appetizers",
        "items": [{
            "name": "Buffalo Chicken Wings",
            "price": 13.42
        }]
    }],
    "features": [
        "Private Dining"
    ],
    "isActive": false“ michelin”: {“
        star”: 3,
        “ranking_years”: 4
    }
}
```

CRUD, 인덱스 관리 및 이벤트 기반 프로그래밍을 보여주는 모든 코드 샘플은 `MongoClient` 객체 `dbClient`, `MongoDatabase` 객체 `connectionDB`, `MongoCollection` 객체 `collection`이 있다고 가정합니다.

> **Note**
>
> 이 설명서의 모든 코드 예제는 MongoDB Java 드라이버 버전 5.3.0으로 테스트되었습니다.

MongoDB Java 드라이버를 사용하여 Amazon DocumentDB에 연결

이 섹션에서는 Java 드라이버를 사용하여 Amazon DocumentDB에 연결하기 위한 단계별 설명서를 제공합니다. 이를 통해 DocumentDB를 Java 애플리케이션에 통합하는 작업을 시작할 수 있습니다.

주제

- 1단계: 프로젝트 설정
- 2단계: 연결 문자열 생성
- 3단계: 연결 코드 작성
- 4단계: 연결 예외 처리
- 5단계: 코드 실행
- 연결 모범 사례

1단계: 프로젝트 설정

1. Maven을 사용하여 java 프로젝트를 생성합니다.

```
mvn archetype:generate -DgroupId=com.docdb.guide -DartifactId=my-docdb-project -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

2. MongoDB Java 드라이버를 'pom.xml' 파일의 프로젝트에 대한 종속성으로 추가합니다.

```
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-sync</artifactId>
    <version>5.3.0</version>
</dependency>
```

2단계: 연결 문자열 생성

Amazon DocumentDB 연결 문자열은 애플리케이션과 DocumentDB 클러스터 간에 연결을 설정하는 데 필수적입니다. 이 문자열은 클러스터 엔드포인트, 포트, 인증 세부 정보 및 다양한 연결 옵션과 같은 중요한 정보를 캡슐화합니다. DocumentDB 연결 문자열을 빌드하려면 일반적으로 기본 형식으로 시작합니다.

```
"mongodb://username:password@cluster-endpoint:port/?[connection options]"
```

‘사용자 이름’ 및 ‘암호’를 실제 자격 증명으로 바꿔야 합니다. AWS Management Console 와를 통해 클러스터의 엔드포인트 및 포트 번호를 찾을 수 있습니다 AWS CLI. 클러스터의 클러스터 엔드포인트를 찾으려면 클러스터 엔드포인트 찾기 섹션을 참조하세요. Amazon DocumentDB의 기본 포트는 27017 은 입니다.

연결 문자열 예제

- 전송 중 암호화를 사용하여 DocumentDB에 연결하고 읽기 요청이 읽기 복제본으로 이동하고 쓰기 요청이 기본 인스턴스로 이동하도록 합니다.

```
"mongodb://username:password@cluster-endpoint:27017/?tls=true&
   tlsCAFile=global-bundle.pem&
   readPreference=secondaryPreferred&
   retryWrites=false"
```

- IAM 인증을 사용하여 DocumentDB에 연결:

```
"mongodb://cluster-endpoint:27017/?tls=true&
   tlsCAFile=global-bundle.pem&
   readPreference=secondaryPreferred&
   retryWrites=false&
   authSource=%24external&
   authMechanism=MONGODB-AWS"
```

연결 문자열에 사용할 수 있는 다양한 옵션은 다음과 같습니다.

- TLS 인증서
- 읽기 복제본에서 읽기
- 쓰기 문제 및 저널링
- RetryWrites
- IAM 인증
- 연결 풀
- 연결 제한 시간 파라미터

TLS 인증서

`tls=true|false` - 이 옵션은 Transport Layer Security(TLS)을 활성화하거나 비활성화합니다. 기본적으로 Amazon DocumentDB 클러스터에서 전송 중 암호화가 활성화되므로 클러스터 수준에서 TLS가 비활성화되지 않는 한 이 옵션의 값은 `true`여야 합니다.

TLS를 사용하는 경우 DocumentDB 클러스터에 대한 연결을 생성할 때 코드는 SSL 인증서를 제공해야 합니다. 클러스터에 안전하게 연결하는 데 필요한 인증서를 다운로드합니다(`global-bundle.pem`). `global-bundle.pem` 파일을 사용하는 방법에는 두 가지가 있습니다.

- 옵션 1 - `global-bundle.pem` 파일에서 모든 인증서를 추출하고 Java의 keytool을 사용하여 나중에 코드에서 사용할 수 있는 `.jks` 파일에 저장합니다. 이 작업을 수행하는 방법을 보여주는 스크립트는 TLS가 활성화된 상태에서 연결의 Java 탭을 참조하세요.
- 옵션 2 - 코드에 `global-bundle.pem` 파일을 동적으로 추가하고, 인 메모리 키 스토어를 빌드하고, `SSLContext`를 사용하여 연결의 일부로 인증서를 제공합니다.

읽기 복제본에서 읽기

`replicaSet=rs0&readPreference=secondaryPreferred` - 이 두 옵션을 지정하면 모든 읽기 요청이 읽기 복제본으로 라우팅되고 쓰기 요청이 기본 인스턴스로 라우팅됩니다. 연결 문자열에서 `replicaSet=rs0`을 사용하면 MongoDB 드라이버가 클러스터 토폴로지의 자동으로 업데이트된 보기를 유지 관리할 수 있으므로 인스턴스가 추가되거나 제거될 때 애플리케이션이 현재 노드 구성의 가시성을 유지할 수 있습니다. 이러한 옵션을 제공하지 않거나 `readPreference=primary`를 지정하지 않으면 모든 읽기 및 쓰기가 기본 인스턴스로 전송됩니다. `readPreference`에 대한 더 많은 옵션은 읽기 기본 설정 옵션 섹션을 참조하세요.

쓰기 문제 및 저널링

쓰기 문제에 따라 쓰기 작업을 위해 데이터베이스에서 요청된 승인 수준이 결정됩니다. MongoDB 드라이버는 쓰기 문제 및 저널 파일을 조정할 수 있는 옵션을 제공합니다. Amazon DocumentDB는 쓰기 문제 및 저널을 설정할 것으로 예상하지 않으며 `w` 및 `j`(`writeConcern` 및 `journal`)에 대해 전송된 값을 무시합니다. DocumentDB는 항상 `writeConcern: majority` 및 `journal: true`를 사용하여 데이터를 기록하므로 클라이언트에 승인을 보내기 전에 대부분의 노드에 쓰기가 안정적으로 기록됩니다.

RetryWrites

`retryWrites` - 엔진 버전 8.0.2부터 Amazon DocumentDB는 재시도 가능한 쓰기를 지원합니다. 를 활성화`retryWrites=true`로 설정합니다. 8.0.2 이전의 엔진 버전에서는이 속성을 로 설정합니다`false`. 자세한 내용은 Amazon DocumentDB에서 재시도 가능한 쓰기 섹션을 참조하세요.

IAM 인증

`authSource=%24external` 및 **`authMechanism=MONGODB-AWS`** -이 두 파라미터는를 사용하여 인증하는 데 사용됩니다 AWS Identity and Access Management. IAM 인증은 현재 인스턴스 기반 클러스터 버전 5.0에서만 사용할 수 있습니다. 자세한 내용은 IAM 자격 증명을 사용한 인증 섹션을 참조하세요.

연결 풀

연결 풀링에 사용할 수 있는 옵션은 다음과 같습니다.

- **maxPoolSize** - 풀에서 생성할 수 있는 최대 연결 수를 설정합니다. 모든 연결이 사용 중이고 새 요청이 들어오면 연결을 사용할 수 있을 때까지 기다립니다. MongoDB Java 드라이버의 기본값은 100입니다.
- **minPoolSize** - 풀에서 항상 유지해야 하는 최소 연결 수를 나타냅니다. MongoDB Java 드라이버의 기본값은 0입니다.
- **maxIdleTimeMS** - 연결을 닫고 제거하기 전에 풀에서 연결을 유휴 상태로 유지할 수 있는 시간을 결정합니다. MongoDB Java 드라이버의 기본값은 100밀리초입니다.
- **waitQueueTimeoutMS** - 풀이 최대 크기일 때 스레드가 연결을 사용할 수 있을 때까지 기다려야 하는 시간을 구성합니다. 이 시간 내에 연결을 사용할 수 없는 경우 예외가 발생합니다. MongoDB Java 드라이버의 기본값은 120,000밀리초(2분)입니다.

연결 제한 시간 파라미터

제한 시간은 작업 또는 연결 시도가 실패로 간주되기까지 걸릴 수 있는 시간을 제한하는 메커니즘입니다. 무기한 대기를 방지하고 리소스 할당을 관리하는 데 사용할 수 있는 제한 시간 파라미터는 다음과 같습니다.

- **connectTimeoutMS** - 드라이버가 클러스터에 대한 연결을 설정하기 위해 기다리는 시간을 구성합니다. 기본값은 10,000밀리초(10초)입니다.
- **socketTimeoutMS** - 드라이버가 쓰기 작업이 아닌 작업을 위해 서버의 응답을 기다리는 시간을 지정합니다. 기본값은 0입니다(제한 시간 없음 또는 무한).
- **serverSelectionTimeoutMS** - 드라이버가 클러스터에서 사용할 수 있는 서버를 찾기 위해 기다리는 시간을 지정합니다. 이 설정의 기본값은 30초이며 장애 조치 중에 새 기본 인스턴스를 선택하기에 충분합니다.

3단계: 연결 코드 작성

다음 코드 예제에서는 Amazon DocumentDB에 TLS를 연결하는 방법을 보여줍니다.

- Java KeyStore 및 SSLContext> 객체를 생성합니다.
- 또한 ConnectionString 객체에 전달하여 MongoClientSettings 객체를 생성합니다. TLS 연결을 만들려면 MongoClientSettings 객체를 사용하여 connectionstring 및 sslcontext를 바인딩해야 합니다.

- MongoClients를 사용하여 MongoClient 객체를 가져옵니다.

```
public static MongoClient makeDbConnection(String dbName, String DbUserName, String
 DbPassword,
    String DbClusterEndPoint, String keyStorePass) throws Exception {
    MongoClient connectedClient;
    String connectionOptions = "?
replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false";
    String connectionUrl = "mongodb://" + DbUserName + ":" + DbPassword + "@" +
 DbClusterEndPoint + ":27017/" +
        dbName + connectionOptions;

    try {
        KeyStore trustStore = KeyStore.getInstance(KeyStore.getDefaultType());
        try (FileInputStream fis = new FileInputStream("src/main/resources/certs/
truststore.jks")) {
            trustStore.load(fis, keyStorePass.toCharArray());
            TrustManagerFactory tmf =
 TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
            tmf.init(trustStore);

            SSLContext sslContext = SSLContext.getInstance("TLSv1.2");
            sslContext.init(null, tmf.getTrustManagers(), new SecureRandom());
            ConnectionString connectionString = new ConnectionString(connectionUrl);
            MongoClientSettings settings = MongoClientSettings.builder()
                .applyConnectionString(connectionString)
                .applyToSslSettings(builder - > {
                    builder.enabled(true);
                    builder.context(sslContext);
                })
                .build();
            connectedClient = MongoClients.create(settings);
        }
        return connectedClient;
    } catch (MongoException e5) {
        throw new RuntimeException(e5);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

4단계: 연결 예외 처리

Java 애플리케이션에서 DocumentDB로 작업할 때 강력하고 안정적인 데이터베이스 작업을 유지하려면 연결 예외 처리가 중요합니다. 이러한 예외를 적절하게 관리하면 문제를 신속하게 진단하는 데 도움이 될 뿐만 아니라 애플리케이션이 임시 네트워크 중단 또는 서버 가용성을 정상적으로 처리할 수 있으므로 안정성과 사용자 경험이 향상됩니다. 연결 설정과 관련된 몇 가지 중요한 예외는 다음과 같습니다.

- **MongoException** - 일반적인 예외는 보다 구체적인 예외가 적용되지 않는 다양한 시나리오에서 발행될 수 있습니다. 이 예외는 모든 MongoDB 예외를 일반적으로 포착하므로 다른 모든 특정 예외 이후에 처리되어야 합니다.
- **MongoTimeoutException** - 작업 시간이 초과되는 경우에 발생합니다. 존재하지 않는 클러스터 엔드포인트를 쿼리하는 경우를 예로 들 수 있습니다.
- **MongoSocketException** - 네트워크 관련 문제가 있는 경우에 발생합니다. 작업 중 갑작스럽게 네트워크 연결이 해제되는 경우를 예로 들 수 있습니다.
- **MongoSecurityException** - 인증이 실패하는 경우에 발생합니다. 잘못된 자격 증명으로 연결하는 경우를 예로 들 수 있습니다.
- **MongoConfigurationException** - 클라이언트 구성에 오류가 있는 경우에 발생합니다. 잘못된 연결 문자열을 사용하는 경우를 예로 들 수 있습니다.

5단계: 코드 실행

다음 코드 샘플은 Amazon DocumentDB 연결을 생성하고 모든 데이터베이스를 인쇄합니다.

```
public static void TestConnection() {
    try (MongoClient mongoClient = makeDbConnection(DATABASE_NAME, DB_USER_NAME,
 DB_PASSWORD, DB_CLUSTER_ENDPOINT, KEYSTORE_PASSWORD)) {
        List < String > databases = mongoClient.listDatabaseNames().into(new ArrayList
 < > ());
        System.out.println("Databases: " + databases);
    } catch (MongoException e) {
        System.err.println("MongoDB error: " + e.getMessage());
        throw new RuntimeException(e);
    }
}
```

연결 모범 사례

다음은 MongoDB Java 드라이버를 사용하여 Amazon DocumentDB에 연결할 때 고려해야 할 모범 사례입니다.

- 클라이언트가 더 이상 리소스를 릴리스할 필요가 없는 경우 항상 `MongoClient`를 닫습니다.
- 예외를 적절하게 처리하고 적절한 오류 로깅을 구현합니다.
- 환경 변수 또는 AWS Secrets Manager 를 사용하여 사용자 이름 및 암호와 같은 민감한 정보를 저장합니다.

Java를 사용하여 Amazon DocumentDB에서 CRUD 작업 수행

이 섹션에서는 MongoDB Java 드라이버를 사용하여 Amazon DocumentDB에서 생성, 읽기, 업데이트, 삭제(CRUD) 작업을 수행하는 방법을 설명합니다.

주제

- DocumentDB 컬렉션에 문서 생성 및 삽입
- DocumentDB 컬렉션에서 데이터 읽기 및 검색
- DocumentDB 컬렉션의 기존 문서 업데이트
- DocumentDB 컬렉션에서 문서 제거
- 재시도 로직을 사용한 오류 처리

DocumentDB 컬렉션에 문서 생성 및 삽입

Amazon DocumentDB에 문서를 삽입하면 컬렉션에 새 데이터를 추가할 수 있습니다. 요구 사항과 작업 중인 데이터의 양에 따라 삽입을 수행하는 방법에는 여러 가지가 있습니다. 컬렉션에 개별 문서를 삽입하는 가장 기본적인 메서드는 `insertOne()`입니다. 한 번에 여러 문서를 삽입하는 경우 `insertMany()` 메서드를 사용하여 단일 작업으로 문서 배열을 추가할 수 있습니다. DocumentDB 컬렉션에 많은 문서를 삽입하는 또 다른 메서드는 `bulkWrite()`입니다. 이 설명서에서는 DocumentDB 컬렉션에서 문서를 생성하는 모든 메서드에 대해 설명합니다.

`insertOne()`

먼저 Amazon DocumentDBB 컬렉션에 개별 문서를 삽입하는 방법을 살펴보겠습니다. 단일 문서 삽입은 `insertOne()` 메서드를 사용하여 수행됩니다. 이 메서드는 삽입을 위해 BsonDocument를 가져와 새로 삽입된 문서의 객체 ID를 가져오는 데 사용할 수 있는 `InsertOneResult` 객체를 반환합니다. 아래 예제 코드는 컬렉션에 하나의 레스토랑 문서를 삽입하는 작업을 보여줍니다.

```
Document article = new Document()
    .append("restaurantId", "REST-21G145")
    .append("name", "Future-proofed Intelligent Bronze Hat")
    .append("cuisine", "International")
    .append("rating", new Document()
        .append("average", 1.8)
        .append("totalReviews", 267))
    .append("features", Arrays.asList("Outdoor Seating", "Live Music"));

try {
    InsertOneResult result = collection.insertOne(article);
    System.out.println("Inserted document with the following id: " +
 result.getInsertedId());
} catch (MongoWriteException e) {
    // Handle duplicate key or other write errors
    System.err.println("Failed to insert document: " + e.getMessage());
    throw e;
} catch (MongoException e) {
    // Handle other MongoDB errors
    System.err.println("MongoDB error: " + e.getMessage());
    throw e;
}
```

`insertOne()`을 사용할 때는 적절한 오류 처리를 포함해야 합니다. 예를 들어 위 코드에서 "`restaurantId`"에는 고유한 인덱스가 있으므로 이 코드를 다시 실행하면 다음과 같은 `MongoWriteException`이 발생합니다.

```
Failed to insert document: Write operation error on server
 docdbCluster.docdb.amazonaws.com:27017.
Write error: WriteError{code=11000, message='E11000 duplicate key error collection:
 Restaurants index: restaurantId_1', details={}}.
```

insertMany()

컬렉션에 많은 문서를 삽입하는 데 사용되는 기본 메서드는 insertMany() 및 `bulkWrite()`입니다.

`insertMany()` 메서드는 단일 작업으로 여러 문서를 삽입하는 가장 간단한 방법입니다. 문서 목록을 받아 컬렉션에 삽입합니다. 이 메서드는 서로 독립적이며 특별한 처리 또는 혼합 작업이 필요하지 않은 새 문서 배치를 삽입할 때 적합합니다. 다음 코드는 파일에서 JSON 문서를 읽고 컬렉션에 삽입하는 방법을 보여줍니다. `insertMany()` 함수는 삽입된 모든 문서의 ID를 가져오는 데 사용할 수 있는 `InsertManyResult`InsertManyResult 객체를 반환합니다.

```
// Read JSON file content
String content = new String(Files.readAllBytes(Paths.get(jsonFileName)));
JSONArray jsonArray = new JSONArray(content);

// Convert JSON articles to Documents
List < Document > restaurants = new ArrayList < > ();
for (int i = 0; i < jsonArray.length(); i++) {
    JSONObject jsonObject = jsonArray.getJSONObject(i);
    Document doc = Document.parse(jsonObject.toString());
    restaurants.add(doc);
}
//insert documents in collection
InsertManyResult result = collection.insertMany(restaurants);

System.out.println("Count of inserted documents: " + result.getInsertedIds().size());
```

bulkWrite()

bulkWrite() 메서드를 사용하여 단일 배치에서 여러 쓰기 작업(삽입, 업데이트, 삭제)을 수행할 수 있습니다. 다른 문서를 업데이트하는 동안 일부 문서를 삽입하는 등 단일 배치에서 다양한 유형의 작업을 수행해야 하는 경우 bulkWrite()를 사용할 수 있습니다. bulkWrite()는 순서가 지정된 배치 쓰기와 순서가 지정되지 않은 쓰기의 두 가지 유형을 지원합니다.

- 순서가 지정된 작업 - (기본값) Amazon DocumentDB는 쓰기 작업을 순차적으로 처리하고 오류가 처음 발생하면 중지합니다. 이후 작업이 이전 작업에 의존하는 경우와 같이 작업 순서가 중요한 경우에 유용합니다. 그러나 정렬된 작업은 일반적으로 정렬되지 않은 작업보다 느립니다. 순서가 지정된 작업의 경우 첫 번째 오류 시 배치가 중지되어 일부 작업이 처리되지 않을 수 있는 경우를 해결해야 합니다.
- 순서가 지정되지 않은 작업 - Amazon DocumentDB를 사용하면 데이터베이스에서 삽입을 단일 실행으로 처리할 수 있습니다. 문서 하나에 오류가 발생하는 경우 작업은 나머지 문서로 계속됩니다. 대량의 데이터를 삽입할 때 특히 유용하며 중복 키로 인해 일부 문서가 실패할 수 있는 데이터 마이그레이션 또는 대량 가져오기와 같은 일부 오류를 허용할 수 있습니다. 순서가 지정되지 않은 작업의 경우 일부 작업은 성공하고 일부 작업은 실패하는 부분적인 성공 시나리오를 해결해야 합니다.

bulkWrite() 메서드를 사용할 때는 몇 가지 필수 클래스가 필요합니다. 먼저 WriteModel 클래스는 모든 쓰기 작업에 대한 기본 클래스 역할을 하며 InsertOneModel, UpdateOneModel, UpdateManyModel, DeleteOneModel, DeleteManyModel과 같은 특정 구현을 통해 다양한 유형의 작업을 처리합니다.

BulkWriteOptions 클래스는 순서가 지정된/순서가 지정되지 않은 실행 설정 또는 문서 검증 우회와 같은 대량 작업의 동작을 구성하는 데 필요합니다. BulkWriteResult 클래스는 삽입, 업데이트 및 삭제된 문서 수를 포함하여 실행 결과에 대한 자세한 정보를 제공합니다.

오류 처리의 경우 MongoBulkWriteException 클래스는 대량 작업 중 발생하는 모든 장애에 대한 정보를 포함하는 반면, BulkWriteError 클래스는 개별 작업 실패에 대한 특정 세부 정보를 제공하므로 매우 중요합니다. 다음 코드는 단일 bulkWrite() 메서드 호출 실행 내에서 문서 목록을 삽입하고 단일 문서를 업데이트 및 삭제하는 예를 보여줍니다. 또한 코드는 BulkWriteOptions 및 BulkWriteResult 작업 방법과 bulkWrite() 작업의 적절한 오류 처리를 보여줍니다.

```
List < WriteModel < Document >> bulkOperations = new ArrayList < > ();
// get list of 10 documents representing 10 restaurants
List < Document > restaurantsToInsert = getSampleData();

for (Document doc: restaurantsToInsert) {
    bulkOperations.add(new InsertOneModel < > (doc));
}
// Update operation
bulkOperations.add(new UpdateOneModel < > (
    new Document("restaurantId", "REST-Y2E9H5"),
    new Document("", new Document("stats.likes", 20))
    .append("", new Document("rating.average", 4.5))));
// Delete operation
bulkOperations.add(new DeleteOneModel < > (new Document("restaurantId", "REST-
D2L431")));

// Perform bulkWrite operation
try {
    BulkWriteOptions options = new BulkWriteOptions()
        .ordered(false); // Allow unordered inserts

    BulkWriteResult result = collection.bulkWrite(bulkOperations, options);

    System.out.println("Inserted: " + result.getInsertedCount());
    System.out.println("Updated: " + result.getModifiedCount());
    System.out.println("Deleted: " + result.getDeletedCount());
} catch (MongoBulkWriteException e) {
    System.err.println("Bulk write error occurred: " + e.getMessage());
    // Log individual write errors
    for (BulkWriteError error: e.getWriteErrors()) {
        System.err.printf("Error at index %d: %s (Code: %d)%n", error.getIndex(),
 error.getMessage(),
```

```
            error.getCode());

        // Log the problematic document
        Document errorDoc = new Document(error.getDetails());
        if (errorDoc != null) {
            System.err.println("Problematic document: " + errorDoc);
        }
    }
} catch (Exception e) {
    System.err.println("Error during bulkWrite: " + e.getMessage());
}
```

재시도 가능한 쓰기

MongoDB와 달리 Amazon DocumentDB는 재시도 가능한 쓰기를 지원하지 않습니다. 따라서 특히 네트워크 문제 또는 임시 서비스 불가능을 처리하기 위해 애플리케이션에서 사용자 지정 재시도 로직을 구현해야 합니다. 일반적으로 잘 구현된 재시도 전략에는 재시도 간의 지연을 늘리고 총 재시도 횟수를 제한하는 작업이 포함됩니다. 오류 처리로 재시도 로직을 빌드하는 코드 샘플은 아래의 재시도 로직을 사용한 오류 처리 섹션을 참조하세요.

DocumentDB 컬렉션에서 데이터 읽기 및 검색

Amazon DocumentDB에서 문서를 쿼리하는 작업은 데이터를 정확하게 검색하고 조작할 수 있는 몇 가지 주요 구성 요소를 중심으로 수행합니다. `find()` 메서드는 MongoDB Java 드라이버의 기본 쿼리 API입니다. 결과를 필터링, 정렬 및 프로젝션하는 다양한 옵션을 통해 복잡한 데이터를 검색할 수 있습니다. `find()` 메서드 외에도 `Filters` 및 `FindIterable`은 MongoDB Java 드라이버에서 쿼리 작업을 위한 구성 요소를 제공하는 두 가지 다른 기본 구성 요소입니다.

`Filters` 클래스는 쿼리 필터를 구성하기 위한 능숙한 API를 제공하는 MongoDB Java 드라이버의 유틸리티 클래스입니다. 이 클래스는 다양한 쿼리 조건을 나타내는 `Bson` 객체의 인스턴스를 생성하는 정적 팩토리 메서드를 제공합니다. 가장 일반적으로 사용되는 메서드에는 균등 비교를 위한 `eq()`, 숫자 비교를 위한 `gt()`, `lt()`, `gte()`, `lte()`, 여러 조건 결합을 위한 `and()`, `or()`, 배열 멤버십 테스트를 위한 `in()`, `nin()`, 패턴 일치를 위한 `regex()` 등이 있습니다. 클래스는 유형이 안전하도록 설계되었으며 원시 문서 기반 쿼리에 비해 더 나은 컴파일 시간 확인을 제공하므로 Java 애플리케이션에서 DocumentDB 쿼리를 구성하는 데 선호되는 접근 방식입니다. 오류 처리는 견고하며 잘못된 필터 구성에 대해 명확한 예외가 발생합니다.

`FindIterable`은 `find()` 메서드의 결과를 처리하도록 설계된 특수 인터페이스입니다. 쿼리 실행을 구체화하고 제어하기 위한 풍부한 메서드 세트를 제공하여 메서드 체인을 위한 능숙한 API를 제공합니다. 인터페이스에는 반환된 문서 수 제한을 위한 `limit()`, 페이지 매김을 위한 `skip()`, 결과 순서 지정을 위한 `sort()`, 특정 필드 선택을 위한 `projection()`, 인덱스 선택을 위한 `hint()`와 같은 필수

쿼리 수정 메서드가 포함되어 있습니다. FindIterable의 배치, 건너뛰기 및 제한 작업은 데이터베이스에서 문서를 검색하고 처리하는 방법을 제어하는 데 도움이 되는 필수 페이지 매김 및 데이터 관리 도구입니다.

일괄 처리(batchSize)는 DocumentDB가 단일 네트워크 왕복으로 클라이언트에 반환하는 문서 수를 제어합니다. 배치 크기를 설정하면 DocumentDB는 일치하는 모든 문서를 한 번에 반환하지 않고 대신 지정된 배치 크기의 그룹으로 반환합니다.

건너뛰기를 사용하면 결과의 시작점을 오프셋하여 일치 항목을 반환하기 전에 지정된 수의 문서를 건너뛰도록 DocumentDB에 지시할 수 있습니다. 예를 들어 skip(20)은 처음 20개의 일치하는 문서를 우회합니다. 결과의 후속 페이지를 검색하려는 페이지 매김 시나리오에서 일반적으로 사용됩니다.

제한은 쿼리에서 반환할 수 있는 총 문서 수를 제한합니다. limit(n)을 지정하면 데이터베이스에 일치하는 항목이 더 있더라도 DocumentDB가 'n' 문서를 반환한 후 문서 반환을 중지합니다.

FindIterable은 Amazon DocumentDB에서 문서를 검색할 때 반복자와 커서 패턴을 모두 지원합니다. FindIterable을 반복자로 사용하면 문서를 느리게 로드할 수 있고 애플리케이션에서 요청할 때만 문서를 가져올 수 있다는 이점이 있습니다. 반복자 사용의 또 다른 이점은 클러스터에 대한 연결을 유지할 책임이 없으므로 연결을 명시적으로 닫을 필요가 없다는 점입니다.

FindIterable은 Amazon DocumentDB 쿼리로 작업할 때 커서 패턴을 사용할 수 있는 MongoCursor 지원도 제공합니다. MongoCursor는 데이터베이스 작업 및 리소스 관리를 제어하는 MongoDB Java 드라이버별 구현입니다. AutoCloseable 인터페이스를 구현하여 리소스 사용 try-with-resources 블록을 통해 명시적 리소스 관리를 가능하게 합니다. 이 블록은 데이터베이스 연결을 올바르게 닫고 서버 리소스를 확보하는 데 매우 중요합니다. 기본적으로 커서는 10분 후에 시간 초과되며 DocumentDB는 이 시간 초과 동작을 변경할 수 있는 옵션을 제공하지 않습니다. 배치 데이터로 작업할 때는 커서 시간이 초과되기 전에 다음 데이터 배치를 검색해야 합니다. MongoCursor를 사용할 때 한 가지 주요 고려 사항은 리소스 유출을 방지하기 위해 명시적으로 닫아야 한다는 것입니다.

이 섹션에서는 find(), Filters, FindIterable에 대한 몇 가지 예제가 나와 있습니다.

다음 코드 예제에서는 find()를 사용하여 “restaurantId” 필드를 통해 단일 문서를 검색하는 방법을 보여줍니다.

```
Document filter = new Document("restaurantId", "REST-21G145");
Document result = collection.find(filter).first();
```

Filters를 사용하면 컴파일 시간 오류 검사를 개선할 수 있지만 java 드라이버를 사용하면 find() 메서드에서 직접 Bson 필터를 지정할 수 있습니다. 다음 예제 코드는 Bson 문서를 find()에 전달합니다.

```
result = collection.find(new Document("$and", Arrays.asList(
    new Document("rating.totalReviews", new Document("$gt", 1000)),
    new Document("priceRange", "$$"))))
```

다음 예제 코드는 `find()`에서 `Filters` 클래스를 사용하는 몇 가지 예를 보여줍니다.

```
FindIterable < Document > results;

// Exact match
results = collection.find(Filters.eq("name", "Thai Curry Palace"));

// Not equal
results = collection.find(Filters.ne("cuisine", "Thai"));

// find an element in an array
results = collection.find(Filters.in("features", Arrays.asList("Private Dining")));

// Greater than
results = collection.find(Filters.gt("rating.average", 3.5));

// Between (inclusive)
results = collection.find(Filters.and(
    Filters.gte("rating.totalReviews", 100),
    Filters.lte("rating.totalReviews", 200)));
// AND
results = collection.find(Filters.and(
    Filters.eq("cuisine", "Thai"),
    Filters.gt("rating.average", 4.5)));

// OR
results = collection.find(Filters.or(
    Filters.eq("cuisine", "Thai"),
    Filters.eq("cuisine", "American")));


// All document where the Field exists
results = collection.find(Filters.exists("michelin"));

// Regex
results = collection.find(Filters.regex("name", Pattern.compile("Curry",
 Pattern.CASE_INSENSITIVE)));

// Find all document where the array contain the list of value regardless of its order
```

```
results = collection.find(Filters.all("features", Arrays.asList("Private Dining",
 "Parking")));

// Array size
results = collection.find(Filters.size("features", 4));
```

다음 예제에서는 FindIterable 객체에서 sort(), skip(), limit(), batchSize() 작업을 연결하는 방법을 보여줍니다. 이러한 작업이 제공되는 순서는 쿼리 성능에 영향을 미칩니다. 이러한 작업의 순서가 sort(), projection(), skip(), limit(), batchSize() 순이 되는 것이 가장 좋은 방법입니다.

```
FindIterable < Document > results = collection.find(Filters.gt("rating.totalReviews",
 1000))
    // Sorting
    .sort(Sorts.orderBy(
        Sorts.descending("address.city"),
        Sorts.ascending("cuisine")))
    // Field selection
    .projection(Projections.fields(
        Projections.include("name", "cuisine", "priceRange"),
        Projections.excludeId()))

    // Pagination
    .skip(20)
    .limit(10)
    .batchSize(2);
```

다음 코드 예제는 FindIterable에서 반복자를 생성하는 방법을 보여줍니다. Java의 forEach 구문을 사용하여 결과 세트를 순회합니다.

```
collection.find(Filters.eq("cuisine", "American")).forEach(doc ->
 System.out.println(doc.toJson()));
```

마지막 find() 코드 예제에서는 문서 검색에 cursor()를 사용하는 방법을 보여줍니다. 코드가 시도 블록을 종료할 때 커서가 닫히도록 시도 블록에 커서를 생성합니다.

```
try (MongoCursor < Document > cursor = collection.find(Filters.eq("cuisine",
 "American"))
    .batchSize(25)
    .cursor()) {
    while (cursor.hasNext()) {
```

```
        Document doc = cursor.next();
        System.out.println(doc.toJson());
    }
} // Cursor automatically closed
```

DocumentDB 컬렉션의 기존 문서 업데이트

Amazon DocumentDB는 기존 문서를 수정하고 존재하지 않을 때 새 문서를 삽입할 수 있는 유연하고 강력한 메커니즘을 제공합니다. MongoDB Java 드라이버는 단일 문서 업데이트를 위한 `updateOne()`, 여러 문서 업데이트를 위한 `updateMany()`, 전체 문서 교체를 위한 `replaceOne()` 등 여러 가지 업데이트 메서드를 제공합니다. 이 세 가지 메서드 외에도 `Updates`, `UpdateOptions` 및 `UpdateResult`는 MongoDB Java 드라이버의 업데이트 작업을 위한 구성 요소를 제공하는 다른 기본 구성 요소입니다.

MongoDB Java 드라이버의 `Updates` 클래스는 업데이트 연산자를 생성하기 위한 정적 팩토리 메서드를 제공하는 유틸리티 클래스입니다. 유형이 안전하고 읽기 가능한 방식으로 업데이트 작업을 구성하기 위한 기본 빌더 역할을 합니다. `set()`, `unset()`, `inc()`와 같은 기본 메서드를 사용하면 문서를 직접 수정할 수 있습니다. 여러 업데이트 작업을 원자적으로 실행하여 데이터 일관성을 보장할 수 있는 `Updates.combine()` 메서드를 사용하여 여러 작업을 결합하면 이 클래스의 성능이 분명해집니다.

`UpdateOptions`는 문서 업데이트 작업에 필수적인 사용자 지정 기능을 제공하는 MongoDB의 Java 드라이버에 있는 강력한 구성 클래스입니다. 이 클래스의 두 가지 중요한 측면은 업데이트 작업에 대한 업서트 및 배열 필터 지원을 제공한다는 점입니다. `upsert(true)`를 통해 활성화된 업서트 기능을 사용하면 업데이트 작업 중에 일치하는 문서를 찾을 수 없는 경우 새 문서를 생성할 수 있습니다. `arrayFilters()`를 통해 업데이트 작업은 특정 기준을 충족하는 배열 요소를 정확하게 업데이트할 수 있습니다.

MongoDB Java 드라이버의 `UpdateResult`는 업데이트 작업의 결과를 자세히 설명하는 피드백 메커니즘을 제공합니다. 이 클래스는 업데이트 기준과 일치하는 문서 수(`matchedCount`), 실제로 수정된 문서 수(`modifiedCount`), 업서트된 문서에 대한 정보(`upsertedId`)의 세 가지 주요 지표를 캡슐화합니다. 이러한 지표를 이해하는 것은 적절한 오류 처리, 업데이트 작업 확인 및 애플리케이션의 데이터 일관성 유지에 필수적입니다.

단일 문서 업데이트 및 교체

DocumentDB에서는 updateOne() 메서드를 사용하여 단일 문서를 업데이트할 수 있습니다. 이 메서드는 업데이트할 문서를 식별하기 위해 일반적으로 `Filters` 클래스에서 제공하는 필터 파라미터, 업데이트할 필드를 결정하는 `Update` 파라미터, 업데이트에 대한 다양한 옵션을 설정하는 선택적 `UpdateOptions` 파라미터를 사용합니다. `updateOne()` 메서드를 사용하면 선택 기준과 일치하는 첫 번째 문서만 업데이트됩니다. 다음 예제 코드는 한 문서의 단일 필드를 업데이트합니다.

```
collection.updateOne(Filters.eq("restaurantId", "REST-Y2E9H5"),
    Updates.set("name", "Amazing Japanese sushi"));
```

한 문서에서 여러 필드를 업데이트하려면 다음 예제와 같이 updateOne() 및 Update.combine()을 함께 사용합니다. 이 예제에서는 문서의 배열에 항목을 추가하는 방법도 보여줍니다.

```
List<Bson> updates = new ArrayList<>();
// Basic field updates
updates.add(Updates.set("name", "Shanghai Best"));
// Array operations
updates.add(Updates.addEachToSet("features", Arrays.asList("Live Music")));
// Counter updates
updates.add(Updates.inc("rating.totalReviews", 10));
// Combine all updates
Bson combinedUpdates = Updates.combine(updates);
// Execute automic update with one call
collection.updateOne(Filters.eq("restaurantId","REST-1J83NH"), combinedUpdates);
```

다음 코드 예제에서는 데이터베이스에서 문서를 업데이트하는 방법을 보여줍니다. 지정된 문서가 없는 경우 작업은 자동으로 새 문서로 삽입합니다. 이 코드는 UpdateResult 객체를 통해 사용 가능한 지표를 사용하는 방법도 보여줍니다.

```
Bson filter = Filters.eq("restaurantId", "REST-0Y9GL0");
Bson update = Updates.set("cuisine", "Indian");
// Upsert operation
UpdateOptions options = new UpdateOptions().upsert(true);
UpdateResult result = collection.updateOne(filter, update, options);

if (result.getUpsertedId() != null) {
    System.out.println("Inserted document with _id: " + result.getUpsertedId());
} else {
     System.out.println("Updated " + result.getModifiedCount() + " document(s)");
}
```

다음 코드 예제에서는 개별 필드를 업데이트하는 대신 replaceOne() 메서드를 사용하여 기존 문서를 새 문서로 완전히 대체하는 방법을 보여줍니다. replaceOne() 메서드는 원본의 _id 필드만 보존하여 전체 문서를 덮어씁니다. 여러 문서가 필터 기준과 일치하는 경우 처음 발견된 문서만 대체됩니다.

```
Document newDocument = new Document()
```

```
                .append("restaurantId", "REST-0Y9GL0")
                .append("name", "Bhiryani Adda")
                .append("cuisine", "Indian")
                .append("rating", new Document()
                        .append("average", 4.8)
                        .append("totalReviews", 267))
                .append("features", Arrays.asList("Outdoor Seating", "Live Music"));

UpdateResult result = collection.replaceOne(
                    Filters.eq("restaurantId", "REST-0Y9GL0"),
                    newDocument);
System.out.printf("Modified %d document%n", result.getModifiedCount());
```

여러 문서 업데이트

컬렉션에서 여러 문서를 동시에 업데이트하는 메서드에는 두 가지가 있습니다. updateMany() 메서드를 사용하거나 UpdateManyModel 및 bulkWrite() 메서드를 함께 사용할 수 있습니다. updateMany() 메서드는 필터 파라미터를 사용하여 업데이트할 문서를 선택하고, Update 파라미터를 사용하여 업데이트할 필드를 식별하며, 선택적 UpdateOptions 파라미터를 사용하여 업데이트 옵션을 지정합니다.

다음 예제 코드는 updateMany() 메서드의 사용 방법을 보여줍니다.

```
Bson filter = Filters.and(
    Filters.in("features", Arrays.asList("Private Dining")),
    Filters.eq("cuisine", "Thai"));
UpdateResult result1 = collection.updateMany(filter, Updates.set("priceRange", "$$$"));
```

다음 예제 코드는 동일한 업데이트를 사용하는 bulkWrite() 메서드를 보여줍니다.

```
BulkWriteOptions options = new BulkWriteOptions().ordered(false);
List < WriteModel < Document >> updates = new ArrayList < > ();
Bson filter = Filters.and(
    Filters.in("features", Arrays.asList("Private Dining")),
    Filters.eq("cuisine", "Indian"));
Bson updateField = Updates.set("priceRange", "$$$");
updates.add(new UpdateManyModel < > (filter, updateField));
BulkWriteResult result = collection.bulkWrite(updates, options);
System.out.printf("Modified %d document%n", result.getModifiedCount());
```

DocumentDB 컬렉션에서 문서 제거

MongoDB Java 드라이버는 단일 문서를 제거하기 위한 `deleteOne()` 및 특정 기준과 일치하는 여러 문서를 제거하기 위한 `deleteMany()`를 제공합니다. 업데이트와 마찬가지로 삭제 작업을 `bulkWrite()` 메서드와 함께 사용할 수도 있습니다. `deleteOne()` 및 `deleteMany()` 모두 삭제된 문서 수를 포함하여 작업 결과에 대한 정보를 제공하는 `DeleteResult` 객체를 반환합니다. 다음은 `deleteMany()`를 사용하여 여러 문서를 제거하는 예입니다.

```
Bson filter = Filters.and(
    Filters.eq("cuisine", "Thai"),
    Filters.lt("rating.totalReviews", 50));
DeleteResult result = collection.deleteMany(filter);
System.out.printf("Deleted %d document%n", result.getDeletedCount());
```

재시도 로직을 사용한 오류 처리

Amazon DocumentDB에 대한 강력한 오류 처리 전략은 오류 분류를 재시도 가능(예: 네트워크 제한 시간, 연결 문제) 및 재시도 불가능(예: 인증 실패, 잘못된 쿼리)으로 구현해야 합니다. 재시도해야 하는 오류로 인한 작업 실패의 경우 각 재시도와 최대 재시도 간에 시간 지연을 구현해야 합니다. CRUD 작업은 `MongoException` 및 해당 하위 클래스를 포착하는 try-catch 블록에 있어야 합니다. 또한 운영 가시성을 위해 오류 모니터링 및 로깅을 포함해야 합니다. 다음은 재시도 오류 처리를 구현하는 방법을 보여주는 샘플 코드입니다.

```
int MAX_RETRIES = 3;
int INITIAL_DELAY_MS = 1000;
int retryCount = 0;

while (true) {
    try {
        crud_operation(); //perform crud that will throw MongoException or one of its
 subclass
        break;
    } catch (MongoException e) {
        if (retryCount < MAX_RETRIES) {
            retryCount++;
            long delayMs = INITIAL_DELAY_MS * (long) Math.pow(2, retryCount - 1);
            try {
                TimeUnit.MILLISECONDS.sleep(delayMs);
            } catch (InterruptedException t) {
                Thread.currentThread().interrupt();
                throw new RuntimeException("Retry interrupted", t);
            }
```

```
                continue;
            } else
                throw new RuntimeException("Crud operation failed", e);
        }
    }
```

Java를 사용한 Amazon DocumentDB의 인덱스 관리

인덱스를 사용하면 Amazon DocumentDB 컬렉션에서 데이터를 효율적으로 검색할 수 있습니다. 인덱스가 없으면 DocumentDB는 컬렉션의 모든 문서를 스캔하여 지정된 쿼리를 충족하는 결과를 반환해야 합니다. 이 주제에서는 MongoDB Java 드라이버를 사용하여 인덱스를 생성, 삭제 및 나열하는 방법에 대한 정보를 제공합니다. 또한 쿼리에서 특정 인덱스가 사용되고 있는지 확인하는 방법과 특정 인덱스를 사용하도록 Amazon DocumentDB에 힌트를 제공하는 방법도 설명합니다.

주제

- Java를 사용하여 인덱스 생성
- 인덱스 삭제
- 인덱스 선택 결정 및 인덱스 힌트 제공

Amazon DocumentDB는 다양한 유형의 인덱스를 지원합니다. 지원되는 모든 인덱스에 대한 포괄적인 개요는 AWS 데이터베이스 블로그의 Amazon DocumentDB에서 인덱싱하는 방법을 참조하세요.

Java를 사용하여 인덱스 생성

MongoDB Java 드라이버를 사용하여 Amazon DocumentDB에서 인덱스를 생성하는 두 가지 메커니즘은 `runCommand()`를 통한 방법과 단일 인덱스의 경우 `createIndex()` 메서드 또는 여러 인덱스의 경우 `createIndexes()` 메서드를 통한 방법입니다. `createIndex()` 및 `createIndexes()` 메서드를 사용하는 한 가지 이유는 인덱스 생성과 관련된 특정 오류를 포착하여 오류 처리를 개선할 수 있기 때문입니다. `runCommand()`에서 이러한 방법을 사용하는 또 다른 이유는 MongDB Java 드라이버가 인덱스 생성 및 조작을 위한 풍부한 지원 클래스 세트를 제공하기 때문입니다. 이러한 지원 클래스는 `createIndex()` 또는 `createIndexes()` 메서드를 사용하는 경우에만 사용할 수 있습니다. 다음 세 가지 지원 클래스가 있습니다.

- **`Indexes`** - 이 클래스는 다양한 유형의 인덱스를 생성하기 위한 정적 팩토리 메서드를 제공하는 유틸리티 클래스 역할을 합니다. 복잡한 인덱스 정의를 생성하는 프로세스를 간소화하며 일반적으로 다른 인덱스 관련 클래스와 함께 사용됩니다.
- **`IndexModel`** - 이 클래스는 인덱스 키 정의 및 해당 옵션을 모두 캡슐화하는 기본 클래스입니다. 인덱싱할 항목(키)과 인덱싱 방법(옵션)을 결합한 전체 인덱스 사양을 나타냅니다. 이 클래스는

createIndexes() 메서드에 전달할 수 있는 인덱스 사양 모음을 정의할 수 있기 때문에 여러 인덱스를 동시에 생성할 때 특히 유용합니다.

- **IndexOptions** - 인덱스 동작을 사용자 지정하기 위한 다양한 메서드를 제공하는 포괄적인 구성 클래스입니다. 여기에는 고유 인덱스, 희소 인덱스, 만료 시간(TTL) 및 부분 필터 표현식에 대한 설정이 포함됩니다. 메서드 체인을 통해 백그라운드 인덱스 구축 및 고유한 제약 조건과 같은 여러 옵션을 구성할 수 있습니다.

단일 인덱스 생성

이 예제에서는 백그라운드에서 createIndex() 메서드를 사용하여 단일 인덱스를 생성하는 방법을 보여줍니다. 배경 및 포그라운드 인덱스 생성을 이해하려면 섹션을 참조하세요인덱스 빌드 유형. 다음 코드 예제에서는 IndexOptions를 사용하여 백그라운드에서 이름이 “unique_restaurantId_idx”인 고유 인덱스를 생성합니다. 그러면 이 IndexOptions 객체가 createIndex() 메서드로 전달됩니다.

```
collection.createIndex(
    Indexes.ascending("restaurantId"),
    new IndexOptions()
        .unique(true)
        .name("unique_restaurantId_idx")
        .background(true));
```

여러 인덱스 생성

이 예제에서는 createIndexes() 메서드를 사용하여 여러 인덱스를 생성합니다. 먼저 IndexModel 객체를 사용하여 각 인덱스에 대한 옵션을 빌드한 다음 IndexModel 객체 목록을 createIndexes() 메서드에 전달합니다. 다음 코드 예제에서는 Indexes 유틸리티 클래스를 사용하여 복합 인덱스를 생성하는 방법을 보여줍니다. 이 클래스는 오름차순 또는 내림차순 정렬 순서를 사용하여 인덱스를 생성할지 여부를 지정하는 데에도 사용됩니다. 여러 인덱스를 생성한 후 listIndexes() 메서드를 호출하여 인덱스 생성을 확인합니다.

```
// Single Field Index on cuisine
IndexModel singleIndex = new IndexModel(
    Indexes.ascending("cuisine"),
    new IndexOptions().name("cuisine_idx"));

// Compound Index
IndexModel compoundIndex = new IndexModel(
    Indexes.compoundIndex(
        Indexes.ascending("address.state"),
        Indexes.ascending("priceRange")),
```

```
    new IndexOptions().name("location_price_idx"));

// Build a list of IndexModel for the indexes
List < IndexModel > indexes = Arrays.asList(
    singleIndex,
    compoundIndex
);

collection.createIndexes(indexes);

// Verify created indexes
collection.listIndexes().forEach(index - > System.out.println("Created index: " +
 index.toJson()));
```

희소 인덱스 및 부분 인덱스 생성

이 예제에서는 각 인덱스 유형에 `IndexModel`을 생성하여 희소 인덱스와 부분 인덱스를 생성합니다.

```
// Sparse Index Model, this will identify only those documents that have a
// michelin star rating
IndexModel sparseIndex = new IndexModel(
    Indexes.ascending("michelin.star"),
    new IndexOptions()
    .name("michelin_sparse_idx")
    .sparse(true));

// Partial Index Model where the restaurant is active and has a rating of 4 and above
IndexModel partialIndex = new IndexModel(
    Indexes.ascending("rating.average"),
    new IndexOptions()
    .name("high_rated_active_idx")
    .partialFilterExpression(
        Filters.and(
            Filters.eq("isActive", true),
            Filters.gte("rating.average", 4.0))));
```

텍스트 인덱스 생성

이 예제에서는 텍스트 인덱스를 생성하는 방법을 보여줍니다. 컬렉션에는 하나의 텍스트 인덱스만 허용되지만 하나의 텍스트 인덱스는 여러 필드를 포함하는 복합 인덱스일 수 있습니다. 텍스트 인덱스에서 여러 필드를 사용하는 경우 인덱스의 각 필드에 가중치를 할당할 수도 있습니다. 배열 필드의 텍스트 인덱스는 Amazon DocumentDB에서 지원되지 않으며 복합 텍스트 인덱스에 최대 30개의 필드를 사용할 수 있지만 세 개의 필드만 가중치를 할당할 수 있습니다.

```
IndexModel textIndex = new IndexModel(
    new Document()
    .append("name", "text")
    .append("description", "text")
    .append("cuisine", "text"),
    new IndexOptions()
    .name("restaurant_text_idx")
    .weights(new Document()
        .append("name", 10) // Restaurant name gets highest weight
        .append("description", 5) // Description get medium weight
        .append("cuisine", 2) // Cuisine type gets low weight
    ));

collection.createIndex(textIndex.getKeys(), textIndex.getOptions());
```

`runCommand()`를 사용하여 인덱스 생성

Amazon DocumentDB는 병렬 인덱스 생성을 지원하여 인덱스를 생성하는 데 걸리는 시간을 줄입니다. 병렬 인덱싱은 여러 동시 작업자를 사용합니다. 인덱스 생성에 사용되는 기본 작업자는 2명입니다. 자세한 내용은 AWS 데이터베이스 블로그의 Amazon DocumentDB에서 병렬 인덱싱의 성능 잠금 해제를 참조하세요. 현재 MongDB Java 드라이버는 `createIndex()` 또는 `createIndexes()`를 사용할 때 작업자 옵션 지정을 지원하지 않으므로 작업자를 지정하는 유일한 방법은 `runCommand`를 통하는 것입니다. 다음 코드 예제에서는 `runCommand`를 사용하여 작업자를 4명으로 늘리는 인덱스를 생성하는 방법을 보여줍니다.

```
Document command = new Document("createIndexes", "Restaurants")
    .append("indexes", Arrays.asList(
        new Document("key", new Document("name", 1))
        .append("name", "restaurant_name_idx")
        .append("workers", 4) // Specify number of workers
    ));

Document commendResult = connectedDB.runCommand(command);
```

인덱스 삭제

MongoDB Java 드라이버는 인덱스를 삭제하는 여러 방법을 제공하여 다양한 시나리오와 기본 설정에 맞게 조정할 수 있습니다. 이름이나 키 사양별으로 인덱스를 삭제하거나 모든 인덱스를 한 번에 삭제할 수 있습니다. 컬렉션 객체에서 메서드 `dropIndex()` 및 `dropIndexes()`를 호출하여 인덱스를 삭제할 수 있습니다. 이름별로 인덱스를 삭제할 때는 특히 복합 또는 자동 생성된 인덱스의 경우 항상 직관적이지 않을 수 있는 올바른 인덱스 이름을 사용해야 합니다. 존재하지 않는 인덱스를 삭제하려고 하면

MongoCommandException이 발생합니다. default _id 인덱스는 컬렉션 내에서 문서 고유성을 보장하기 때문에 삭제할 수 없습니다.

다음 코드 예제에서는 인덱스가 생성되는 필드 이름을 제공하거나 모든 인덱스를 삭제하여 인덱스를 삭제하는 방법을 보여줍니다.

```
String indexName = "unique_restaurantId_idx";
Document keys = new Document("cuisine", 1);
// Drop index by name
collection.dropIndex(indexName);

// Drop index by keys
collection.dropIndex(keys);

// Drop all indexes
collection.dropIndexes();
```

여러 키를 사용하여 인덱스를 삭제할 때는 지정된 모든 키가 포함된 복합 인덱스가 있고 키 순서가 올바른지 확인합니다. 위의 인덱스 생성 예제 코드는 "cuisine" 및 기능에 대한 복합 키를 보여줍니다. 해당 복합 키를 삭제하려고 시도하지만 해당 순서가 생성에 사용된 순서가 아닌 경우 MongoCommnadException 오류는 다음과 같이 발생합니다.

```
Document keys = new Document("features", 1)
    .append("cuisine", 1);
try {
    // Drop index by keys
    collection.dropIndex(keys);
    System.out.println("Successfully dropped index with keys: " + keys.toJson());

} catch (MongoCommandException commErr) {
    System.out.println("Error dropping index: " + commErr.getErrorMessage());
    throw new RuntimeException("MongoCommandException was thrown while dropping index",
 commErr);
}
```

다음 오류가 표시됩니다.

```
Error dropping index: Cannot drop index: index not found.
Tests run: 3, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.819 sec <<< FAILURE!
com.amazon.docdb.guide.DocDBGuideTest.testindexGuide()  Time elapsed: 0.817 sec  <<<
 FAILURE!
```

```
org.opentest4j.AssertionFailedError: Unexpected exception thrown:
 java.lang.RuntimeException: MongoCommandException was thrown while dropping index
```

인덱스 선택 결정 및 인덱스 힌트 제공

Amazon DocumentDB에서 설명 기능을 사용하는 작업은 쿼리 성능 및 인덱스 사용량을 이해하는 데 매우 중요합니다. 쿼리를 실행할 때 `explain()` 메서드를 추가하여 사용 중인 인덱스를 포함하여 쿼리 계획에 대한 자세한 정보를 얻을 수 있습니다. `explain()` 출력은 쿼리 실행 단계, 검사된 문서 수, 각 단계에 소요된 시간에 대한 인사이트를 제공합니다. 이 정보는 특정 인덱스가 효과적으로 사용되고 있는지 또는 쿼리가 다른 인덱스 구조의 이점을 누릴 수 있는지 식별하는 데 매우 중요합니다.

`explain()` 메서드는 `find()` 메서드와 함께 연결할 수 있습니다. `explain()` 메서드는 `explain()`에서 반환되는 세부 수준을 결정하는 선택적 `ExplainVerbosity` 열거형을 사용할 수 있습니다. 현재 DocumentDB에서는 `EXECUTION_STATS` 및 `QUERY_PLANNER` 열거자만 지원됩니다. 다음 코드 예제에서는 특정 쿼리에 대한 쿼리 플래너를 가져오는 방법을 보여줍니다.

```
// Query we want to analyze
Document query = new Document()
    .append("cuisine", "Thai")
    .append("rating.average", new Document("$gte", 4.0));


Document allPlansExplain =
 collection.find(query).explain(ExplainVerbosity.QUERY_PLANNER);
System.out.println("All Plans Explain:\n" + allPlansExplain.toJson());
```

쿼리 플래너 세부 수준에 대해 다음 JSON 문서가 반환됩니다.

```
{
  "queryPlanner": {
    "plannerVersion": 1,
    "namespace": "ProgGuideData.Restaurants",
    "winningPlan": {
      "stage": "IXSCAN",
      "indexName": "cuisine_idx",
      "direction": "forward"
    }
  },
  "serverInfo": {
    "host": "guidecluster3",
    "port": 27017,
```

```
    "version": "5.0.0"
  },
  "ok": 1,
  "operationTime": {
    "$timestamp": {
      "t": 1739221668,
      "i": 1
    }
  }
}
```

Amazon DocumentDB가 특정 인덱스를 사용하도록 영향을 미치거나 강제하는 몇 가지 옵션이 있습니다. `hint()` 및 `hintString()` 메서드를 사용하면 쿼리에 사용할 인덱스를 명시적으로 지정하여 쿼리 최적화 프로그램의 기본 인덱스 선택 동작을 재정의할 수 있습니다. DocumentDB의 쿼리 최적화 프로그램은 일반적으로 인덱스 선택에 적합한 선택이지만 왜곡된 데이터를 처리하거나 인덱스 성능을 테스트하는 등 `hint()` 또는 `hintString()`을 통해 특정 인덱스를 강제 적용하는 경우 유용할 수 있습니다.

다음 코드 예제에서는 위 코드에서 실행된 동일한 쿼리에 복합 인덱스 “cuisine_features_idx”를 강제로 사용합니다.

```
// Query we want to analyze
Document query = new Document()
    .append("cuisine", "Thai")
    .append("rating.average", new Document("$gte", 4.0));

List < Document > queryDocs = new ArrayList < > ();
collection.find(query).hintString("cuisine_features_idx").forEach(doc - >
 queryDocs.add(doc));
```

Amazon DocumentDB 및 Java를 사용한 이벤트 기반 프로그래밍

Amazon DocumentDB의 컨텍스트에서 이벤트 기반 프로그래밍은 데이터베이스 변경이 후속 비즈니스 로직 및 프로세스를 트리거하는 기본 이벤트 생성기 역할을 하는 강력한 아키텍처 패턴을 나타냅니다. DocumentDB 컬렉션에 레코드가 삽입, 업데이트 또는 삭제되면 이러한 변경 사항은 다양한 다운스트림 프로세스, 알림 또는 데이터 동기화 작업을 자동으로 시작하는 이벤트 역할을 합니다. 이 패턴은 여러 애플리케이션 또는 서비스가 데이터 변경에 실시간으로 대응해야 하는 최신 분산 시스템에서 특히 유용합니다. DocumentDB에서 이벤트 기반 프로그래밍을 구현하는 주요 메커니즘은 변경 스트림입니다.

> **Note**
>
> 이 설명서에서는 작업 중인 컬렉션에서 변경 스트림을 활성화했다고 가정합니다. 컬렉션에서 변경 스트림을 활성화하는 방법은 Amazon DocumentDB에서 변경 스트림 사용 섹션을 참조하세요.

Java 애플리케이션에서 변경 스트림 작업

MongoDB의 Java 드라이버에 있는 `watch()` 메서드는 Amazon DocumentDB에서 실시간 데이터 변경을 모니터링하는 기본 메커니즘입니다. `watch()` 메서드는 `MongoClient`, `MongoDatabase` 및 `MongoCollection` 객체에 의해 호출될 수 있습니다.

`watch()` 메서드는 업데이트 시 전체 문서 조회, 신뢰성을 위한 재개 토큰과 타임스탬프 제공, 변경 사항 필터링을 위한 파이프라인 집계 단계 등 다양한 구성 옵션을 지원하는 `ChangeStreamIterable` 인스턴스를 반환합니다.

`ChangeStreamIterable`은 핵심 Java 인터페이스 `Iterable`을 구현하며 `forEach()`와 함께 사용할 수 있습니다. `forEach()`를 사용하여 이벤트를 캡처하려면 변경된 이벤트를 처리하는 `forEach()`에 콜백 함수를 전달합니다. 다음 코드 조각은 컬렉션에서 변경 스트림을 열어 변경 이벤트 모니터링을 시작하는 방법을 보여줍니다.

```
ChangeStreamIterable < Document > iterator = collection.watch();
iterator.forEach(event - > {
    System.out.println("Received a change: " + event);
});
```

모든 변경 이벤트를 순회하는 또 다른 방법은 클러스터와의 연결을 유지하고 새 변경 이벤트가 발생할 때마다 지속적으로 수신하는 커서를 여는 것입니다. 변경 스트림 커서를 가져오려면 `ChangeStreamIterable` 객체의 `cursor()` 메서드를 사용합니다. 다음 코드 예제에서는 커서를 사용하여 변경 이벤트를 모니터링하는 방법을 보여줍니다.

```
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch().cursor()) {
    System.out.println(cursor.tryNext());
}
```

가장 좋은 방법은 try-with-resource 문에서 MongoChangeStreamCursor를 생성하거나 커서를 수동으로 닫는 것입니다. ChangeStreamIterable에서 cursor() 메서드를 호출하면 ChangeStreamDocument 객체를 통해 생성된 MongoChangeStreamCursor가 반환됩니다.

ChangeStreamDocument 클래스는 스트림의 개별 변경 이벤트를 나타내는 중요한 구성 요소입니다. 여기에는 작업 유형(삽입, 업데이트, 삭제, 교체), 문서 키, 네임스페이스 정보, 사용 가능한 경우 전체 문서 콘텐츠를 포함하여 각 수정에 대한 자세한 정보가 포함되어 있습니다. 클래스는 변경 유형을 결정하는 getOperationType(), 전체 문서 상태에 액세스하는 getFullDocument(), 수정된 문서를 식별하는 getDocumentKey() 등 변경 이벤트의 다양한 측면에 액세스하는 방법을 제공합니다.

ChangeStreamDocument 객체는 두 가지 중요한 정보인 재개 토큰과 변경 이벤트 시간을 제공합니다.

DocumentDB 변경 스트림에서 토큰 재개 및 시간 기반 작업은 연속성을 유지하고 과거 데이터 액세스를 관리하기 위한 중요한 메커니즘을 제공합니다. 재개 토큰은 각 변경 이벤트에 대해 생성된 고유 식별자로, 애플리케이션이 연결 해제 또는 실패 후 특정 지점에서 변경 스트림 처리를 다시 시작할 수 있도록 허용하는 북마크 역할을 합니다. 변경 스트림 커서가 생성되면 resumeAfter() 옵션을 통해 이전에 저장된 재개 토큰을 사용하여 처음부터 시작하거나 이벤트를 손실하는 대신 스트림이 중단된 위치에서 계속될 수 있습니다.

변경 스트림의 시간 기반 작업은 변경 이벤트 모니터링의 시작점을 관리하기 위한 다양한 접근 방식을 제공합니다. startAtOperationTime() 옵션을 사용하면 특정 타임스탬프에서 또는 그 이후에 발생한 변경 사항을 볼 수 있습니다. 이러한 시간 기반 기능은 과거 데이터 처리, 특정 시점으로 복구 또는 시스템 간 동기화가 필요한 시나리오에서 특히 유용합니다.

다음 코드 예제에서는 삽입 문서와 연결된 이벤트를 검색하고 재개 토큰을 캡처한 다음 해당 토큰을 제공하여 삽입 이벤트 후 이벤트 모니터링을 시작합니다. 이벤트는 업데이트 이벤트와 연결되어 업데이트가 발생한 클러스터 시간을 가져오고 해당 타임스탬프를 추가 처리를 위한 시작점으로 사용합니다.

```
BsonDocument resumeToken;
BsonTimestamp resumeTime;

try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch().cursor()) {
    System.out.println("****************** Insert Document ******************");
    ChangeStreamDocument < Document > insertChange = cursor.tryNext();
    resumeToken = insertChange.getResumeToken();
    printJson(cursor.tryNext());
}
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch()
```

```
    .resumeAfter(resumeToken)
    .cursor()) {
    System.out.println("****************** Update Document ******************");
    ChangeStreamDocument < Document > insertChange = cursor.tryNext();
    resumeTime = insertChange.getClusterTime();
    printJson(cursor.tryNext());
}
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch()
    .startAtOperationTime(resumeTime)
    .cursor()) {
    System.out.println("****************** Delete Document ******************");
    printJson(cursor.tryNext());
  }
```

기본적으로 변경 사항 업데이트 이벤트에는 전체 문서가 포함되지 않으며 변경된 내용만 포함됩니다. 업데이트된 전체 문서에 액세스해야 하는 경우 `ChangeStreamIterable` 객체에서 `fullDocument()` 메서드를 호출할 수 있습니다. 업데이트 이벤트에 대해 전체 문서를 반환하도록 요청하면 스트림 변경 호출이 이루어진 시점에 존재하는 문서가 반환된다는 점에 유의하세요.

이 메서드는 `FullDocument` 열거형을 파라미터로 사용합니다. 현재 Amazon DocumentDB는 DEFAULT 및 UPDATE_LOOKUP 값만 지원합니다. 다음 코드 조각은 변경 사항을 감시하기 시작할 때 업데이트 이벤트에 대한 전체 문서를 요청하는 방법을 보여줍니다.

```
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch().fullDocument(FullDocument.UPDATE_LOOKUP).cursor())
```

JSON 스키마 검증 사용

`$jsonSchema`평가 쿼리 연산자를 사용하여, 컬렉션에 삽입되는 문서를 검증할 수 있습니다.

주제

- JSON 스키마 검증 생성 및 사용
- 지원되는 키워드
- bypassDocumentValidation
- 제한 사항

JSON 스키마 검증 생성 및 사용

스키마 검증을 통한 컬렉션 만들기

`createCollection`작업 및 검증 규칙을 사용하여 컬렉션을 만들 수 있습니다. 이러한 검증 규칙은 Amazon DocumentDB 문서를 삽입하거나 업데이트할 때 적용됩니다. 다음 코드 예제에서는 직원 컬렉션에 대한 검증 규칙을 보여줍니다:

```
db.createCollection("employees", {
   "validator": {
      "$jsonSchema": {
         "bsonType": "object",
         "title": "employee validation",
         "required": [ "name", "employeeId"],
         "properties": {
            "name": {
                  "bsonType": "object",
                  "properties": {
                     "firstName": {
                        "bsonType": ["string"]
                     },
                     "lastName": {
                        "bsonType": ["string"]
                     }
                  },
                  "additionalProperties" : false
            },
            "employeeId": {
               "bsonType": "string",
               "description": "Unique Identifier for employee"
            },
             "salary": {
               "bsonType": "double"
            },
            "age": {
               "bsonType": "number"
            }
         },
         "additionalProperties" : true
      }
   },
   "validationLevel": "strict", "validationAction": "error"
} )
```

유효한 문서 삽입

다음 예제는 위의 스키마 검증 규칙을 준수하는 문서를 삽입합니다:

```
db.employees.insert({"name" : { "firstName" : "Carol" , "lastName" : "Smith"},
 "employeeId": "c720a" , "salary": 1000.0 })
db.employees.insert({ "name" : { "firstName" : "William", "lastName" : "Taylor" },
 "employeeId" : "c721a", "age" : 24})
```

잘못된 문서 삽입

다음 예제에서는 위 스키마 유효성 검사 규칙을 준수하지 않는 문서를 삽입합니다. 이 예제에서 employeeId 값은 문자열이 아닙니다:

```
db.employees.insert({
    "name" : { "firstName" : "Carol" , "lastName" : "Smith"},
    "employeeId": 720 ,
    "salary": 1000.0
})
```

이 예제는 문서 내의 잘못된 구문을 보여줍니다.

컬렉션 수정

이 collMod 명령은 기존 컬렉션의 검증 규칙을 추가하거나 수정하는 데 사용됩니다. 다음 예제에서는 필수 필드 목록에 급여 필드를 추가합니다:

```
db.runCommand({"collMod" : "employees",
   "validator": {
      "$jsonSchema": {
         "bsonType": "object",
         "title": "employee validation",
         "required": [ "name", "employeeId", "salary"],
         "properties": {
            "name": {
                  "bsonType": "object",
                  "properties": {
                     "firstName": {
                        "bsonType": ["string"]
                     },
                     "lastName": {
                        "bsonType": ["string"]
```

```
                    }
                },
                "additionalProperties" : false
            },
            "employeeId": {
               "bsonType": "string",
               "description": "Unique Identifier for employee"
            },
             "salary": {
               "bsonType": "double"
            },
            "age": {
               "bsonType": "number"
            }
         },
         "additionalProperties" : true
      }
   }
} )
```

검증 규칙이 변경되기 전에 추가된 주소 지정 문서

검증 규칙이 변경되기 전에 컬렉션에 추가된 문서의 주소를 지정하려면 다음 `validationLevel` 수정자를 사용하십시오:

- 엄격: 모든 삽입 및 업데이트에 검증 규칙을 적용합니다.
- 보통: 기존의 유효한 문서에 검증 규칙을 적용합니다. 업데이트하는 동안 기존의 잘못된 문서는 확인되지 않습니다.

다음 예제에서는 “employees”라는 이름의 컬렉션에 대한 검증 규칙을 업데이트한 후 급여 필드가 필수입니다. 다음 문서 업데이트는 실패합니다:

```
db.runCommand({
    update: "employees",
    updates: [{
        q: { "employeeId": "c721a" },
        u: { age: 25 , salary : 1000},
        upsert: true }]
})
```

Amazon DocumentDB는 다음 출력을 반환합니다:

```
{
"n" : 0,
    "nModified" : 0,
    "writeErrors" : [
        {
"index" : 0,
            "code" : 121,
            "errmsg" : "Document failed validation"
        }
    ],
    "ok" : 1,
    "operationTime" : Timestamp(1234567890, 1)
}
```

검증 수준을 moderate로 업데이트하면 위 문서가 성공적으로 업데이트될 수 있습니다:

```
db.runCommand({
    "collMod" : "employees",
    validationLevel : "moderate"
})

db.runCommand({
    update: "employees",
    updates: [{
        q: { "employeeId": "c721a" },
        u: { age: 25 , salary : 1000},
        upsert: true }]
})
```

Amazon DocumentDB는 다음 출력을 반환합니다:

```
{
"n" : 1,
    "nModified" : 1,
    "ok" : 1,
    "operationTime" : Timestamp(1234567890, 1)
}
```

$jsonSchema를 사용하여 문서 검색

$jsonSchema연산자를 필터로 사용하여 JSON schema와 일치하는 문서를 쿼리할 수 있습니다. 이것은 필터 문서에 최상위 필드로 존재하거나 $and, $or, 및 $nor과 같은 쿼리 연산자와 함께 사용할 수

있는 최상위 연산자입니다. 다음 예제는 $JSONSchema를 개별 필터 및 기타 필터 연산자와 함께 사용하는 방법을 보여줍니다:

"직원" 컬렉션에 삽입된 문서:

```
{ "name" : { "firstName" : "Carol", "lastName" : "Smith" }, "employeeId" : "c720a",
 "salary" : 1000 }
{ "name" : { "firstName" : "Emily", "lastName" : "Brown" }, "employeeId" : "c720b",
 "age" : 25, "salary" : 1050.2 }
{ "name" : { "firstName" : "William", "lastName" : "Taylor" }, "employeeId" : "c721a",
 "age" : 24, "salary" : 1400.5 }
{ "name" : { "firstName" : "Jane", "lastName" : "Doe" }, "employeeId" : "c721a",
 "salary" : 1300 }
```

$jsonSchema연산자로만 필터링된 컬렉션:

```
db.employees.find({
       $jsonSchema: { required: ["age"] } })
```

Amazon DocumentDB는 다음 출력을 반환합니다:

```
{ "_id" : ObjectId("64e5f91c6218c620cf0e8f8b"), "name" : { "firstName" : "Emily",
 "lastName" : "Brown" }, "employeeId" : "c720b", "age" : 25, "salary" : 1050.2 }
{ "_id" : ObjectId("64e5f94e6218c620cf0e8f8c"), "name" : { "firstName" : "William",
 "lastName" : "Taylor" }, "employeeId" : "c721a", "age" : 24, "salary" : 1400.5 }
```

$jsonSchema연산자와 다른 연산자로 필터링된 컬렉션:

```
db.employees.find({
       $or: [{ $jsonSchema: { required: ["age", "name"]}},
            { salary: { $lte:1000}}]});
```

Amazon DocumentDB는 다음 출력을 반환합니다:

```
{ "_id" : ObjectId("64e5f8886218c620cf0e8f8a"), "name" : { "firstName" : "Carol",
 "lastName" : "Smith" }, "employeeId" : "c720a", "salary" : 1000 }
{ "_id" : ObjectId("64e5f91c6218c620cf0e8f8b"), "name" : { "firstName" : "Emily",
 "lastName" : "Brown" }, "employeeId" : "c720b", "age" : 25, "salary" : 1050.2 }
{ "_id" : ObjectId("64e5f94e6218c620cf0e8f8c"), "name" : { "firstName" : "William",
 "lastName" : "Taylor" }, "employeeId" : "c721a", "age" : 24, "salary" : 1400.5 }
```

$jsonSchema 연산자로 그리고 $match로 집합 필터에서 필터링된 집합:

```
db.employees.aggregate(
    [{ $match: {
        $jsonSchema: {
            required: ["name", "employeeId"],
            properties: {"salary" :{"bsonType": "double"}}
        }
       }
    }]
)
```

Amazon DocumentDB는 다음 출력을 반환합니다:

```
{
"_id" : ObjectId("64e5f8886218c620cf0e8f8a"),
 "name" : { "firstName" : "Carol", "lastName" : "Smith" },
"employeeId" : "c720a",
"salary" : 1000
}
{
"_id" : ObjectId("64e5f91c6218c620cf0e8f8b"),
"name" : { "firstName" : "Emily", "lastName" : "Brown" },
"employeeId" : "c720b",
"age" : 25,
"salary" : 1050.2
}
{
"_id" : ObjectId("64e5f94e6218c620cf0e8f8c"),
"name" : { "firstName" : "William", "lastName" : "Taylor" },
"employeeId" : "c721a",
"age" : 24,
"salary" : 1400.5
}
{
"_id" : ObjectId("64e5f9786218c620cf0e8f8d"),
"name" : { "firstName" : "Jane", "lastName" : "Doe" },
"employeeId" : "c721a",
"salary" : 1300
}
```

기존 검증 규칙 보기

컬렉션의 기존 검증 규칙을 보려면 다음을 사용하세요:

```
db.runCommand({
    listCollections: 1,
    filter: { name: 'employees' }
})
```

Amazon DocumentDB는 다음 출력을 반환합니다:

```
{
    "waitedMS" : NumberLong(0),
    "cursor" : {
        "firstBatch" : [
            {
                "name" : "employees",
                "type" : "collection",
                "options" : {
                    "autoIndexId" : true,
                    "capped" : false,
                    "validator" : {
                        "$jsonSchema" : {
                            "bsonType" : "object",
                            "title" : "employee validation",
                            "required" : [
                                "name",
                                "employeeId",
                                "salary"
                            ],
                            "properties" : {
                                "name" : {
                                    "bsonType" : "object",
                                    "properties" : {
                                        "firstName" : {
                                            "bsonType" : [
                                                "string"
                                            ]
                                        },
                                        "lastName" : {
                                            "bsonType" : [
                                                "string"
                                            ]
```

```
                                    }
                                },
                                "additionalProperties" : false
                            },
                            "employeeId" : {
                                "bsonType" : "string",
                                "description" : "Unique Identifier for employee"
                            },
                            "salary" : {
                                "bsonType" : "double"
                            },
                            "age" : {
                                "bsonType" : "number"
                            }
                        },
                        "additionalProperties" : true
                    }
                },
                "validationLevel" : "moderate",
                "validationAction" : "error"
            },
            "info" : {
                "readOnly" : false
            },
            "idIndex" : {
                "v" : 2,
                "key" : {
                    "_id" : 1
                },
                "name" : "_id_",
                "ns" : "test.employees"
            }
        }
    ],
    "id" : NumberLong(0),
    "ns" : "test.$cmd.listCollections"
},
"ok" : 1,
"operationTime" : Timestamp(1692788937, 1)
}
```

Amazon DocumentDB는 또한 $out 집계 단계에서 검증 규칙을 유지합니다.

지원되는 키워드

create및 collMod 명령에서 지원되는 필드는 다음과 같습니다.

- **Validator**— $jsonSchema 연산자를 지원합니다.
- **ValidationLevel**— off, strict, 및 moderate 값을 지원합니다.
- **ValidationAction**— error 값을 지원합니다.

$jsonSchema 연산자는 다음 키워드를 지원합니다:

- additionalItems
- additionalProperties
- allOf
- anyOf
- bsonType
- dependencies
- description
- enum
- exclusiveMaximum
- exclusiveMinimum
- items
- maximum
- minimum
- maxItems
- minItems
- maxLength
- minLength
- maxProperties
- minProperties
- multipleOf
- not
- oneOf

- pattern
- patternProperties
- properties
- required
- title
- type
- uniqueItems

bypassDocumentValidation

Amazon DocumentDB는 다음 명령 및 메서드에 대해 bypassDocumentValidation을 지원합니다.

- insert
- update
- findAndModify
- aggregate 명령 및 db.collection.aggregate() 메서드의 $out 단계

Amazon DocumentDB는 bypassDocumentValidation에 대해 다음 명령을 지원하지 않습니다.

- aggregate 명령 및 db.collection.aggregate() 메서드의 $merge
- mapReduce 명령 및 db.collection.mapReduce() 메서드
- applyOps 명령

제한 사항

$jsonSchema에는 다음과 같은 제한 사항이 적용됩니다.

- Amazon DocumentDB는 작업이 검증 규칙에 실패하면 “문서 검증에 실패했습니다.”라는 오류를 반환합니다.
- Amazon DocumentDB 탄력적 클러스터는 $jsonSchema를 지원하지 않습니다.

Amazon DocumentDB에서 쿼리

이 섹션에서는 Amazon DocumentDB를 사용한 쿼리의 모든 측면을 설명합니다.

주제

- 문서 쿼리
- 쿼리 계획
- 결과의 설명
- 쿼리 플래너 v2
- 쿼리 플래너 v3
- Amazon DocumentDB에서 지리 공간 데이터 쿼리
- 부분 인덱스
- Amazon DocumentDB를 사용하여 텍스트 검색 수행

문서 쿼리

이따금 판매하는 물품을 고객이 보고 구매할 수 있도록 온라인 상점의 재고를 조회해야 할 수도 있습니다. 컬렉션을 쿼리하는 것은 컬렉션의 모든 문서에 대한 것이든 특정 기준을 충족하는 문서에만 대한 것이든 관계없이 상대적으로 간단합니다.

문서를 쿼리하려면 `find()` 작업을 사용합니다. `find()` 명령에는 반환할 문서 선택 시 사용할 기준을 정의하는 단일 문서 파라미터가 있습니다. `find()`의 출력은 줄 바꿈이 없이 한 줄로 된 텍스트 형식의 문서입니다. 쉽게 읽기 위해 출력 문서의 형식을 지정하려면 `find().pretty()`를 사용하십시오. 이 주제의 모든 예제는 `.pretty()`를 사용하여 출력 형식을 지정합니다.

다음 코드 샘플은 이전 두 연습에서 `example` 컬렉션에 삽입`insertMany()`한 문서 4개를 사용`insertOne()`하며에 있습니다문서 추가.

주제

- 컬렉션의 모든 문서 검색
- 피드 값과 일치하는 문서 검색
- 내장 문서와 일치하는 문서 검색
- 내장 문서의 필드 값과 일치하는 문서 검색
- 배열이 일치하는 문서 검색
- 배열의 값과 일치하는 문서 검색
- 연산자를 사용하여 문서 검색

컬렉션의 모든 문서 검색

모음의 모든 문서를 검색하려면 빈 쿼리 문서로 `find()` 작업을 사용하십시오.

다음 쿼리는 `example` 모음의 모든 문서를 반환합니다.

```
db.example.find( {} ).pretty()
```

피드 값과 일치하는 문서 검색

필드 및 값과 일치하는 모든 문서를 검색하려면 일치하는 필드 및 값을 식별하는 쿼리 문서로 `find()` 작업을 사용합니다.

이전 문서를 사용하면 이 쿼리는 "Item" 필드가 "Pen"과 동일한 모든 문서를 반환합니다.

```
db.example.find( { "Item": "Pen" } ).pretty()
```

내장 문서와 일치하는 문서 검색

내장 문서와 일치하는 모든 문서를 찾으려면 내장 문서 이름과 내장 문서의 모든 필드 및 값을 지정하는 쿼리 문서와 함께 `find()` 작업을 사용합니다.

내장 문서를 일치시킬 때 문서의 내장 문서는 쿼리에 있는 것과 동일한 이름을 가져야 합니다. 또한, 내장 문서의 필드 및 값은 쿼리와 일치해야 합니다.

다음 쿼리는 "Poster Paint" 문서만 반환합니다. "Pen"에는 "OnHand" 및 "MinOnHand"에 대한 다른 값이 있으며, "Spray Paint"에는 쿼리 문서보다 필드 하나(`OrderQnty`)가 더 있기 때문입니다.

```
db.example.find({"Inventory": {
    "OnHand": 47,
    "MinOnHand": 50 } } ).pretty()
```

내장 문서의 필드 값과 일치하는 문서 검색

내장 문서와 일치하는 모든 문서를 찾으려면 내장 문서 이름과 내장 문서의 모든 필드 및 값을 지정하는 쿼리 문서와 함께 `find()` 작업을 사용합니다.

이전 문서에서 다음 쿼리는 "점 표기법"을 사용하여 내장 문서와 관심 있는 필드를 지정합니다. 다른 필드가 내장 문서에 표시될 수 있는지 여부와 무관하게 이와 일치하는 모든 문서가 반환됩니다. "Poster Paint" 및 "Spray Paint"가 지정된 필드 및 값과 일치하므로 쿼리가 "Poster Paint" 및 "Spray Paint"를 반환합니다.

```
db.example.find({"Inventory.OnHand": 47, "Inventory.MinOnHand": 50 }).pretty()
```

배열이 일치하는 문서 검색

배열이 일치하는 모든 문서를 찾으려면 관심 있는 배열 이름과 해당 배열의 모든 값을 포함하여 `find()` 작업을 사용합니다. 쿼리가 배열 값이 동일하면서 쿼리와 동일한 순서인 해당 이름을 가진 배열을 포함한 모든 문서를 반환합니다.

"Poster Paint"에는 추가 색상(White)이 있고 "Spray Paint"에는 색상이 다른 순서로 있으므로 다음 쿼리는 "Pen"만을 반환합니다.

```
db.example.find( { "Colors": ["Red","Green","Blue","Black"] } ).pretty()
```

배열의 값과 일치하는 문서 검색

특정 배열 값을 보유한 모든 문서를 찾으려면 관심 있는 배열 이름과 값을 포함하여 `find()` 작업을 사용합니다.

```
db.example.find( { "Colors": "Red" } ).pretty()
```

각각 `Colors`라는 배열과 배열 내에 "Red" 값이 있으므로 이전 작업은 세 문서 모두를 반환합니다. "White" 값을 지정한 경우 쿼리는 "Poster Paint"만 반환합니다.

연산자를 사용하여 문서 검색

다음 쿼리는 "`Inventory.OnHand`" 값이 50 미만인 모든 문서를 반환합니다.

```
db.example.find(
        { "Inventory.OnHand": { $lt: 50 } } )
```

지원되는 쿼리 연산자 목록은 쿼리 및 프로젝션 연산자 섹션을 참조하십시오.

쿼리 계획

쿼리 계획에 **executionStats** 대한를 보려면 어떻게 해야 합니까?

쿼리가 예상보다 느리게 실행되는 이유를 결정할 때 쿼리 계획에 대한 `executionStats`가 무엇인지 이해하는 것이 유용할 수 있습니다. `executionStats`에서는 특정 단계에서 반환된 문

서 수(nReturned), 각 단계에서 소요된 실행 시간(executionTimeMillisEstimate) 및 쿼리 계획을 생성하는 데 걸리는 시간(planningTimeMillis)을 제공합니다. 다음 쿼리 예제와 executionStats같이의 출력에서 최적화 작업에 집중할 수 있도록 쿼리의 가장 시간이 많이 걸리는 단계를 결정할 수 있습니다. executionStats 파라미터는 현재 update 및 delete 명령을 지원하지 않습니다.

> **Note**
>
> Amazon DocumentDB는 분산, 내결함성, 자가 치유 스토리지 시스템을 활용하는 목적으로 만들어진 데이터베이스 엔진에서 MongoDB 3.6 API를 에뮬레이션합니다. 그 결과, 쿼리 계획과 explain()의 출력은 Amazon DocumentDB와 MongoDB 간에 다를 수 있습니다. 쿼리 계획을 제어하려는 고객은 $hint 연산자를 사용하여 기본 인덱스를 선택할 수 있습니다.

다음과 같이 explain() 명령에서 개선하려는 쿼리를 실행합니다.

```
db.runCommand({explain: {query document}}).
explain("executionStats").executionStats;
```

예는 다음과 같습니다.

```
db.fish.find({}).limit(2).explain("executionStats");
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "test.fish",
        "winningPlan" : {
            "stage" : "SUBSCAN",
            "inputStage" : {
                "stage" : "LIMIT_SKIP",
                "inputStage" : {
                    "stage" : "COLLSCAN"
                }
            }
        }
    },
```

```
    "executionStats" : {
        "executionSuccess" : true,
        "executionTimeMillis" : "0.063",
        "planningTimeMillis" : "0.040",
        "executionStages" : {
            "stage" : "SUBSCAN",
            "nReturned" : "2",
            "executionTimeMillisEstimate" : "0.012",
            "inputStage" : {
                "stage" : "LIMIT_SKIP",
                "nReturned" : "2",
                "executionTimeMillisEstimate" : "0.005",
                "inputStage" : {
                    "stage" : "COLLSCAN",
                    "nReturned" : "2",
                    "executionTimeMillisEstimate" : "0.005"
                }
            }
        }
    },
    "serverInfo" : {
        "host" : "enginedemo",
        "port" : 27017,
        "version" : "3.6.0"
    },
    "ok" : 1
}
```

위의 쿼리에서 executionStats만 보는 데 관심이 있다면 다음 명령을 사용할 수 있습니다. 작은 컬렉션에서는 성능 이득이 무시할 수준인 경우 Amazon DocumentDB 쿼리 프로세서는 인덱스를 사용하지 않도록 결정할 수 있습니다.

```
db.fish.find({}).limit(2).explain("executionStats").executionStats;
```

쿼리 계획 캐시

성능을 최적화하고 계획 기간을 줄이기 위해 Amazon DocumentDB는 쿼리 계획을 내부적으로 캐싱합니다. 이렇게 하면 캐시된 계획을 사용하여 동일한 모양의 쿼리를 직접 실행할 수 있습니다.

하지만 이 캐싱으로 인해 동일한 쿼리에 대해 무작위 지연이 발생할 수 있습니다. 예를 들어 쿼리를 실행하는 데 보통 1초가 걸리는 경우 가끔 10초가 걸릴 수 있습니다. 이는 시간이 지나면서 reader 인스턴

스가 다양한 형태의 쿼리를 캐시하여 메모리를 소비하기 때문입니다. 이러한 무작위 속도 저하가 발생하는 경우 메모리를 해제하기 위해 별도의 조치를 취하지 않아도 됩니다. 시스템에서 자동으로 메모리 사용량을 관리하고 메모리가 특정 임계값에 도달하면 자동으로 해제됩니다.

결과의 설명

쿼리 계획에 대한 정보를 반환하려는 경우 Amazon DocumentDB는 세부 정보 표시 모드 `queryPlanner`를 지원합니다. `explain` 결과는 옵티마이저가 선택한 쿼리 계획을 다음과 비슷한 형식으로 반환합니다.

```
{
   "queryPlanner" : {
      "plannerVersion" : <int>,
      "namespace" : <string>,
      "winningPlan" : {
         "stage" : <STAGE1>,
         ...
         "inputStage" : {
            "stage" : <STAGE2>,
            ...
            "inputStage" : {
               ...
            }
         }
      }
   }
}
```

다음 섹션에서는 일반적인 `explain` 결과를 정의합니다.

주제

- 스캔 및 필터 단계
- 인덱스 교집합
- 인덱스 병합
- 다중 인덱스 교집합/병합
- 복합 인덱스
- 정렬 단계

- 그룹 단계

스캔 및 필터 단계

옵티마이저는 다음 스캔 중 하나를 선택할 수 있습니다.

콜스캔

이 단계는 순차적 수집 스캔입니다.

```
{
    "stage" : "COLLSCAN"
}
```

ISSCAN

이 단계에서는 인덱스 키를 스캔합니다. 옵티마이저는 이 단계 내 문서를 검색할 수 있으며, 이 경우 나중에 FETCH 단계가 추가될 수 있습니다.

```
db.foo.find({"a": 1})
{
    "stage" : "IXSCAN",
    "direction" : "forward",
    "indexName" : <idx_name>
}
```

FETCH

옵티마이저가 IXSCAN 이외의 단계에서 문서를 검색한 경우 결과에는 FETCH 단계가 포함됩니다. 예를 들어, 위의 IXSCAN 쿼리는 FETCH와 IXSCAN 단계가 조합된 결과를 초래할 수 있습니다.

```
db.foo.find({"a": 1})
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXSCAN",
        "indexName" : <idx_name>
    }
}
```

IXONLYSCAN은 인덱스 키만 스캔합니다. 복합 인덱스를 생성해도 FETCH를 피할 수는 없습니다.

인덱스 교집합

아일랜드

Amazon DocumentDB에는 인덱스 교차를 활용할 수 있는 경우 IXScan의 InputStage 배열이 포함된 IXAND 스테이지가 포함될 수 있습니다. 예를 들어 다음과 같은 출력이 표시될 수 있습니다.

```
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXAND",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                "indexName" : "a_1"
            },
            {
                "stage" : "IXSCAN",
                "indexName" : "b_1"
            }
        ]
    }
}
```

인덱스 병합

익소르

인덱스 교차와 마찬가지로 Amazon DocumentDB에는 `$or` 연산자를 위한 `inputStages` 배열이 있는 `IXOR` 스테이지가 포함될 수 있습니다.

```
db.foo.find({"$or": [{"a": {"$gt": 2}}, {"b": {"$lt": 2}}]})
```

위 쿼리의 경우 설명 출력은 다음과 같을 수 있습니다.

```
{
```

```
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXOR",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                "indexName" : "a_1"
            },
            {
                "stage" : "IXSCAN",
                "indexName" : "b_1"
            }
        ]
    }
}
```

다중 인덱스 교집합/병합

Amazon DocumentDB는 여러 인덱스 교차 또는 유니온 단계를 함께 결합한 다음 결과를 가져올 수 있습니다. 예제:

```
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXOR",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                ...
            },
            {
                "stage" : "IXAND",
                "inputStages" : [
                    {
                        "stage" : "IXSCAN",
                        ...
                    },
                    {
                        "stage" : "IXSCAN",
                        ...
                    }
                ]
```

```
            }
        ]
    }
}
```

인덱스 교차 또는 유니온 스테이지의 사용은 인덱스 유형(스파스, 복합 등)의 영향을 받지 않습니다.

복합 인덱스

Amazon DocumentDB 복합 인덱스 사용은 인덱싱된 필드의 시작 부분에만 국한되지 않습니다. 접미사 부분이 있는 인덱스를 사용할 수 있지만 그다지 효율적이지 않을 수 있습니다.

예를 들어, `{ a: 1, b: -1 }`의 복합 인덱스는 아래 세 가지 쿼리를 모두 지원할 수 있습니다.

```
db.orders.find( { a: 1 } )

db.orders.find( { b: 1 } )

db.orders.find( { a: 1, b: 1 } )
```

정렬 단계

요청된 정렬 키에 인덱스가 있는 경우 Amazon DocumentDB는 인덱스를 사용하여 순서를 확인할 수 있습니다. 이 경우 결과에는 `SORT` 단계가 아닌 `IXSCAN` 단계가 포함됩니다. 옵티마이저가 일반 정렬을 선호하는 경우 다음과 같은 단계가 포함됩니다.

```
{
    "stage" : "SORT",
    "sortPattern" : {
        "a" : 1,
        "b" : -1
    }
}
```

그룹 단계

Amazon DocumentDB는 다음과 같은 두 가지 그룹 전략을 지원합니다.

- `SORT_AGGREGATE`: 온 디스크 정렬 집계.

- HASH_AGGREGATE: 인 메모리 해시 집계.

쿼리 플래너 v2

새로운 Amazon DocumentDB용 쿼리 플래너(플래너 버전 2.0)는 고급 쿼리 최적화 기능과 향상된 성능을 제공합니다. Amazon DocumentDB 5.0용 플래너 버전 2.0은 인덱스와 함께 `find` 및 `update` 연산자를 사용할 때 이전 버전보다 최대 10배 향상된 성능을 제공합니다. 성능 개선은 주로 보다 최적화된 인덱스 계획 사용 및 부정 연산자(`$neq`, `$nin`), 중첩 `$elementMatch`와 같은 연산자에 대한 인덱스 스캔 지원 활성화에서 비롯됩니다. 플래너 버전 2.0 쿼리는 개선된 비용 추정 기법, 최적화된 알고리즘 및 향상된 안정성을 통해 더 빠르게 실행됩니다. 플래너 버전 2.0은 계획 캐시 필터 API도 지원하므로 플래너 안정성이 향상됩니다. 이 기능을 통해 Amazon DocumentDB 5.0은 이제 다양한 버전의 쿼리 플래너 중에서 선택할 수 있는 기능을 제공합니다.

주제

- 사전 조건
- 플래너 버전 2.0을 기본 쿼리 플래너로 선택
- 모범 사례
- 제한 사항
- Find 및 Update 연산자에 대한 개선 사항
- 캐시 필터 계획
- 플래너 버전 1.0, 2.0 및 MongoDB 간의 잠재적 동작 차이
- 플래너 버전 2.0은 동작 격차를 MongoDB와 연결합니다.

사전 조건

플래너 버전 2.0에는 다음 사전 조건이 적용됩니다.

- 플래너 버전 2.0은 엔진 버전 5.0을 사용할 수 AWS 리전 있는 모든에서 사용할 수 있습니다.
- 버전 2.0을 기본 쿼리 플래너로 사용하도록 선택하려면 클러스터가 Amazon DocumentDB 버전 5.0의 엔진 패치 버전 3.0.15902 이상에 있어야 합니다. 최신 엔진 버전 패치로 업데이트하는 단계는 클러스터의 엔진 버전에 대한 패치 업데이트 수행 섹션을 참조하세요.
- 플래너 버전 2.0을 기본 쿼리 플래너로 설정하려면 클러스터 파라미터 그룹을 업데이트할 수 있는 IAM 권한이 필요합니다.
- Amazon DocumentDB 8.0의 경우 플래너 버전 3.0이 기본 쿼리 플래너입니다. 세부 정보는 쿼리 플래너 v3 섹션을 참조하세요.

플래너 버전 2.0을 기본 쿼리 플래너로 선택

다음 단계에 따라 콘솔 또는 CLI에서 2.0을 기본 쿼리 플래너로 선택합니다.

- Amazon DocumentDB 클러스터 파라미터 수정의 단계에 따라 클러스터의 파라미터 그룹을 수정합니다.
- 'plannerVersion'이라는 파라미터의 경우 값을 플래너 버전을 나타내는 2.0으로 변경합니다.
- 즉시 적용을 선택합니다(재부팅 시 적용을 선택하는 경우 다음에 클러스터를 재부팅할 때까지 선택이 적용되지 않음).

모범 사례

예상 결과를 얻으려면 플래너 버전 2.0을 적용할 때 다음 모범 사례를 사용하세요.

- 글로벌 클러스터에서 두 리전의 클러스터 파라미터 그룹에서 동일한 `plannerVersion` 값(1.0 또는 2.0)을 선택합니다. 기본 및 보조 리전에서 다른 플래너 버전을 선택하면 쿼리 동작과 성능이 일관되지 않을 수 있습니다.
- 예약된 유지 관리 기간 동안 또는 감소된 트래픽 기간 동안 플래너 버전 2.0으로 업데이트하면 워크로드가 활발하게 실행 중일 때 플래너 버전이 변경되는 경우 오류율이 증가할 수 있으므로 중단이 가장 적습니다.
- 플래너 버전 2.0은 MongoDB 쉘 버전 5.0에서 가장 최적으로 작동합니다.

제한 사항

플래너 버전 2.0에는 다음과 같은 제한이 적용됩니다.

- 플래너 버전 2.0은 탄력적 클러스터에서 지원되지 않으며, 플래너 버전 1.0으로 대체됩니다.
- 플래너 버전 2.0은 집계 및 고유 명령에 대해 지원되지 않으며, 플래너 버전 1.0으로 대체됩니다.
- 일부 쿼리 셰이프에는 계획 캐시 필터가 설정되어 있을 수 없습니다. 현재 목록은 섹션을 참조하세요지원되는 쿼리 셰이프.

`Find` 및 **`Update`** 연산자에 대한 개선 사항

플래너 버전 2.0은 `find`, `update`, `delete`, `find-and-modify` 명령을 포함한 기본 작업을 최적화합니다. 다음의 탭 섹션에서는 인덱스에 대한 향상된 기능과 플래너 버전 2.0의 쿼리 성능 개선을 보여줍니다.

Enhanced index support

- 플래너 버전 2.0에는 $nin, $ne, $not {eq}, $not {in} 및 $type, $elemMatch를 포함한 부정 연산자에 대한 인덱스 지원이 추가되었습니다.

```
Sample Document: { "x": 10, "y": [1, 2, 3] }

db.foo.createIndex({ "x": 1, "y": 1 })
db.foo.find({ "x": {$nin: [20, 30] }})
db.foo.find({"x":{ $type: "string" }})

db.foo.createIndex({"x.y": 1})
db.foo.find({"x":{$elemMatch:{"y":{$elemMatch:{"$gt": 3 }}}}})
```

- 플래너 버전 2.0은 $exists가 쿼리 표현식에 없는 경우에도 희소 인덱스 또는 부분 인덱스를 사용합니다.

```
Sample Document: {"name": "Bob", "email": "example@fake.com" }

Using Planner Version 1.0, you can specify the command as shown below:
db.foo.find({email: "example@fake.com", email: {$exists: true}})

Using Planner Version 2.0, you can specify command without $exists:
db.foo.find({ email: "example@fake.com" })
```

- 플래너 버전 2.0은 쿼리 조건이 부분 인덱스 필터 표현식과 정확히 일치하지 않는 경우에도 부분 인덱스를 활용합니다.

```
Sample Document: {"name": "Bob", "age": 34}
db.foo.createIndex({"age":1},{partialFilterExpression:{"age":{$lt:50}}})

With Planner Version 1.0, index is used only when the query condition meets the
 partial
 index filter criterion:
     db.foo.find({"age":{$lt:50}})

With Planner Version 2.0, index is used even when the query condition doesn't meet
 the index
criterion:
db.foo.find({"age":{$lt:30}})
```

- 플래너 버전 2.0은 $elemMatch 쿼리와 함께 부분 인덱스 스캔을 활용합니다.

```
Sample Document: {"name": "Bob", "age": [34,35,36]}
db.foo.createIndex({"age":1},{partialFilterExpression:{"age":{$lt:50,$gt:20}}})
db.foo.find({age:{$elemMatch:{$lt:50,$gt:20}}})
```

- 플래너 버전 2.0에는 애플리케이션 코드에 $hint를 제공할 필요 없이 $regex에 대한 인덱스 스캔 지원이 포함되어 있습니다. $regex는 접두사 검색에서만 인덱스를 지원합니다.

```
Sample Document: { "x": [1, 2, 3], "y": "apple" }
db.foo.createIndex({ "x": 1, "y": 1 })
db.foo.find({"y":{ $regex: "^a" }})
```

- 플래너 버전 2.0은 다중 키 필드에 대한 등식 조건을 사용하여 다중 키 인덱스와 관련된 쿼리의 성능을 개선합니다.

```
Sample Document: {"x": [1, 2, 3],
"y": 5}
db.foo.createIndex({"x": 1, "y":1})
db.foo.find({"x": 2,
"y": {$gt: 1}}).limit(1)
```

- 플래너 버전 2.0은 특히 문서가 8KB를 초과하는 컬렉션에서 여러 필터가 포함된 쿼리의 성능을 개선합니다.

```
Sample Document: {"x": 2,
"y": 4,
"z": 9,
"t": 99}
db.foo.find({$and: [{"x": {$gt : 1}, "y": {$gt : 3}, "z": {$lt : 10},
"t":{$lt : 100}}]})
```

- 플래너 버전 2.0은 정렬 단계를 제거하여 복합 인덱스와 함께 $in 연산자를 사용할 경우의 쿼리 성능을 개선합니다.

```
Sample Document: {"x": 2,
"y": 4,
"z": 9,
"t": 99}
db.foo.createIndex({"x":1, "y":1})
db.foo.find({"x":2,
"y":$in:[1,2,3,4]}).sort({x:1,y:1})
```

또한 $in 요소와 함께 다중 키 인덱스를 사용하는 쿼리의 성능을 개선합니다.

```
Sample Document: {"x": [1, 2, 3]}
db.foo.createIndex({"x": 1})
db.foo.find("x":{$in:[>100 elements]})
```

Query performance improvements

- 플래너 버전 2.0은 특히 문서가 8KB를 초과하는 컬렉션에서 여러 필터가 포함된 쿼리의 성능을 개선합니다.

```
Sample Document: {"x": 2,
"y": 4,
"z": 9,
"t": 99}
db.foo.find({$and: [{"x": {$gt : 1}, "y": {$gt : 3}, "z": {$lt : 10},
"t":{$lt : 100}}]})
```

- 플래너 버전 2.0은 정렬 단계를 제거하여 복합 인덱스와 함께 $in 연산자를 사용할 경우의 쿼리 성능을 개선합니다.

```
Sample Document: {"x": 2,
"y": 4,
"z": 9,
"t": 99}
db.foo.createIndex({"x":1, "y":1})
db.foo.find({"x":2,
"y":$in:[1,2,3,4]}).sort({x:1,y:1})
```

또한 $in 요소와 함께 다중 키 인덱스를 사용하는 쿼리의 성능을 개선합니다.

```
Sample Document: {"x": [1, 2, 3]}
db.foo.createIndex({"x": 1})
db.foo.find("x":{$in:[>100 elements]})
```

캐시 필터 계획

플래너 버전 2.0에는 인덱스 필터라고도 하는 계획 캐시 필터에 대한 지원이 추가되어 플래너가 특정 쿼리 셰이프에 대해 고려하는 인덱스 세트를 제한할 수 있습니다. 필터는 데이터베이스 명령으로 설정되고 서버에 적용되므로 쿼리 회귀가 발생하는 경우 애플리케이션 코드를 수정하지 않고도 완화할 수 있습니다.

플래너 버전 2.0에서는 계획 캐시 필터가 `find` 및 `count` 명령에 적용되고 Amazon DocumentDB 5.0.2 및 8.0.2부터 `update`, `delete`및 `findAndModify` 명령에도 적용됩니다. `distinct` 및 `aggregate` 명령에는 플래너 버전 3.0이 필요합니다.

명령 참조, 지원되는 쿼리 셰이프, 작업 예제 및 필터가 힌트와 상호 작용하는 방법에 대한 자세한 내용은 섹션을 참조하세요캐시 필터 계획.

플래너 버전 1.0, 2.0 및 MongoDB 간의 잠재적 동작 차이

일부 극단적인 경우에는 플래너 버전 2.0이 MongoDB의 결과와 약간 다른 결과를 생성할 수 있습니다. 이 섹션에서는 이러한 가능성의 몇 가지 예를 살펴봅니다.

$(update) and $(projection)

- 경우에 따라 MongoDB의 `$(update)` 및 `$(projection)` 연산자는 Amazon DocumentDB의 플래너 버전 1.0과 다르게 작동할 수 있습니다. 다음은 일부 예제입니다.

  ```
  db.students_list.insertMany( [ { _id: 5, student_ids: [ 100, 200 ], grades: [ 95,
   100 ], grad_year: [ 2024, 2023 ] } ] )
  ```

  ```
  db.students_list.updateOne({ student_ids: 100, grades: 100, grad_year: 2024 },
  { $set: { “grad_year.$”: 2025 } }
  ```

  - 플래너 버전 1.0 - 2022년 필드 업데이트
  - MongoDB - 2022년 필드 업데이트
  - 플래너 버전 2.0 - 2021년 필드 업데이트

- ```
  db.col.insert({x:[1,2,3]})
  db.col.update({$and:[{x:1},{x:3}]},{$set:{"x.$":500}})
  ```

  - 플래너 버전 1.0 - 일치하는 첫 번째 요소를 무작위로 업데이트
  - MongoDB - 일치하는 첫 번째 요소를 무작위로 업데이트
  ```

  - 플래너 버전 2.0 - 업데이트하지 않음
- ```
  db.col.insert({x:[1,2,3]})
  db.col.find()
  ```
  - 플래너 버전 1.0 - 일치하는 요소를 무작위로 선택
  - MongoDB - 일치하는 요소를 무작위로 선택
  - 플래너 버전 2.0 - 선택하지 않음
- ```
  db.col.insert({x:100})
  db.col.update({x:100},{x:100})
  ```
  - 플래너 버전 1.0 - nModified 개수 변경
  - MongoDB - nModified 개수 변경
  - 플래너 버전 2.0 - 동일한 값으로 업데이트해도 nModified 개수는 변경되지 않습니다.
- `$(update)` 연산자를 `$setOnInsert`와 함께 사용하면 플래너 버전 1.0 및 MongoDB에서 오류가 발생하지만 플래너 버전 2.0에서는 오류가 발생하지 않습니다.
- 존재하지 않는 필드의 이름을 `$field`로 변경하면 플래너 버전 2.0에서 오류가 발생하지만 플래너 버전 1.0 및 MongoDB에서는 업데이트를 생성하지 않습니다.

Index behavior

- 플래너 버전 2.0은 `$hint`가 적합하지 않은 인덱스와 함께 적용될 때 오류가 발생하지만 플래너 버전 1.0 및 MongoDB에서는 오류가 발생하지 않습니다.

```
// Insert
db.col.insert({x:1})
db.col.insert({x:2})
db.col.insert({x:3})

// Create index on x with partialFilter Expression {x:{$gt:2}}
db.col.createIndex({x:1},{partialFilterExpression:{x:{$gt:2}}})

// Mongodb allows hint on the following queries
db.col.find({x:1}).hint("x_1")
// result is no documents returned because {x:1} is not indexed by the partial
 index
// Without $hint mongo should return {x:1}, thus the difference in result between
 COLSCAN and IXSCAN
```

```
DocumentDB will error out when $hint is applied on such cases.
db.col.find({x:1}).hint("x_1")
Error: error: {
    "ok" : 0,
    "operationTime" : Timestamp(1746473021, 1),
    "code" : 2,
    "errmsg" : "Cannot use Hint for this Query. Index is multi key index , partial
 index or sparse index and query is not optimized to use this index."
}

rs0:PRIMARY> db.runCommand({"planCacheSetFilter": "col", "query": { location:
 {$nearSphere: {$geometry: {type: "Point", coordinates: [1, 1]}}}}, "indexes":
 ["name_1"]})
{
    "ok" : 0,
    "operationTime" : Timestamp(1750815778, 1),
    "code" : 303,
    "errmsg" : "Unsupported query shape for index filter $nearSphere"
}
```

- $near는 플래너 버전 2.0에서 $hint({“$natural”:1})를 사용할 수 없습니다.

```
// indexes present are index on x and geo index

rs0:PRIMARY> db.usarestaurants.getIndexes()
[
    {
        "v" : 4,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.usarestaurants"
    },
    {
        "v" : 4,
        "key" : {
            "location" : "2dsphere"
        },
        "name" : "location_2dsphere",
        "ns" : "test.usarestaurants",
        "2dsphereIndexVersion" : 1
    }
```

```
]

// Planner Version 2.0 will throw an error when $hint is applied with index "x_1"
rs0:PRIMARY> db.usarestaurants.find({    "location":{       "$nearSphere":{
         "$geometry":{             "type":"Point",             "coordinates":
[                -122.3516,                47.6156             ]          },
          "$minDistance":1,          "$maxDistance":2000       }    } }, {
  "name":1 }).hint({"$natural": 1})
Error: error: {
    "ok" : 0,
    "operationTime" : Timestamp(1746475524, 1),
    "code" : 291,
    "errmsg" : "unable to find index for $geoNear query"
}

// Planner Version 1.0 and MongoDB will not throw an error
 db.usarestaurants.find({    "location":{       "$nearSphere":{
  "$geometry":{             "type":"Point",             "coordinates":[
            -122.3516,                47.6156             ]          },
        "$minDistance":1,          "$maxDistance":2000       }    } }, {
  "name":1 }).hint({"$natural": 1})
{ "_id" : ObjectId("681918e087dadfd99b7f0172"), "name" : "Noodle House" }
```

- MongoDB는 전체 정규식 인덱스 스캔을 지원하지만 플래너 버전 2.0은 접두사 필드에서만 정규식 인덱스 스캔을 지원합니다.

```
// index on x
db.col.createIndex({x:1})

// index scan is used only for prefix regexes
rs0:PRIMARY> db.col.find({x: /^x/}).explain()
{
    "queryPlanner" : {
        "plannerVersion" : 2,
        "namespace" : "test.col",
        "winningPlan" : {
            "stage" : "IXSCAN",
            "indexName" : "x_1",
            "direction" : "forward",
            "indexCond" : {
                "$and" : [
                    {
                        "x" : {
```

```
                            "$regex" : /^x/
                        }
                    }
                ]
            },
            "filter" : {
                "x" : {
                    "$regex" : /^x/
                }
            }
        }
    },
    "indexFilterSet" : false,
    "indexFilterApplied" : false,
    "ok" : 1,
    "operationTime" : Timestamp(1746474527, 1)
}

// COLSCAN is used for non-prefix regexes
rs0:PRIMARY> db.col.find({x: /x$/}).explain()
{
    "queryPlanner" : {
        "plannerVersion" : 2,
        "namespace" : "test.col",
        "winningPlan" : {
            "stage" : "COLLSCAN",
            "filter" : {
                "x" : {
                    "$regex" : /x$/
                }
            }
        }
    },
    "indexFilterSet" : false,
    "indexFilterApplied" : false,
    "ok" : 1,
    "operationTime" : Timestamp(1746474575, 1)
```

- 플래너 버전 2.0에서 계획 캐시 필터를 사용하는 경우 MongoDB에 비해 몇 가지 고유한 차이점이 있습니다. 플래너 버전 2.0은 계획 캐시 필터를 사용한 “projection” 및 “collation” 지정을 지원하지 않지만 MongoDB에서는 지원합니다. 그러나 MongoDB 인덱스 필터는 인 메모리 전용이며 다시 시작한 후 손실됩니다. 플래너 버전 2.0은 재시작 및 패치를 통해 인덱스 필터를 유지합니다.

Others

- 플래너 버전 2.0을 사용할 때 DML 감사 로그의 형식은 플래너 버전 1.0과 약간 다릅니다.

```
command -  db.col.find({x:1})

************** Audit logs generated ******************

// v1 format for dml audit logs
{"atype":"authCheck","ts":1746473479983,"timestamp_utc":"2025-05-05
 19:31:19.983","remote_ip":"127.0.0.1:47022","users":
[{"user":"serviceadmin","db":"test"}],"param":
{"command":"find","ns":"test.col","args":{"batchSize":101,"filter":
{"x":1},"find":"col","limit":18446744073709551615,"lsid":
{"id":{"$binary":"P6RCGz9ZS4iWBSSHWXW15A==","$type":"4"},"uid":
{"$binary":"6Jo8PisnEi3dte03+pJFjdCyn/5cGQL8V2KqaoWsnk8=","$type":"0"}},"maxScan":18446744

// v2 formal for dml audit logs
{"atype":"authCheck","ts":1746473583711,"timestamp_utc":"2025-05-05
 19:33:03.711","remote_ip":"127.0.0.1:37754","users":
[{"user":"serviceadmin","db":"test"}],"param":
{"command":"find","ns":"test.col","args":{"find":"col","filter":{"x":1},"lsid":
{"id":{"$binary":"nJ88TGCSSd
+BeD2+ZtrhQg==","$type":"4"}},"$db":"test"},"result":0}}
```

- 설명 계획의 일부로 사용되는 인덱스 조건:

```
rs0:PRIMARY> db.col.createIndex({index1:1})
{
 "createdCollectionAutomatically" : false,
 "numIndexesBefore" : 1,
 "numIndexesAfter" : 2,
 "ok" : 1,
 "operationTime" : Timestamp(1761149251, 1)
}
```

  플래너 버전 2.0은 인덱스 조건 및 필터를 표시하는 계획 출력을 설명합니다.

```
rs0:PRIMARY> db.col.find({$and:[{price:{$eq:300}},{item:
{$eq:"apples"}}]}).explain()
{
 "queryPlanner" : {
```

```
  "plannerVersion" : 2,
  "namespace" : "test.col",
  "winningPlan" : {
   "stage" : "IXSCAN",
   "indexName" : "price_1",
   "direction" : "forward",
   "indexCond" : {
    "$and" : [
     {
      "price" : {
       "$eq" : 300
      }
     }
    ]
   },
   "filter" : {
    "$and" : [
     {
      "item" : {
       "$eq" : "apples"
      }
     }
    ]
   }
  }
 },
 "indexFilterSet" : false,
 "indexFilterApplied" : false,
 "ok" : 1,
 "operationTime" : Timestamp(1761149497, 1)
}
```

플래너 버전 1.0은 계획 출력을 설명합니다.

```
rs0:PRIMARY> db.col.find({$and:[{price:{$eq:300}},{item:
{$eq:"apples"}}]}).explain()
{
 "queryPlanner" : {
  "plannerVersion" : 1,
  "namespace" : "test.col",
  "winningPlan" : {
   "stage" : "IXSCAN",
   "indexName" : "price_1",
```

```
    "direction" : "forward"
   }
  },
  "ok" : 1,
  "operationTime" : Timestamp(1761149533, 1)
}
```

플래너 버전 2.0은 동작 격차를 MongoDB와 연결합니다.

플래너 버전 2.0이 MongoDB의 동작 격차를 해소하는 몇 가지 영역이 있습니다.

- 플래너 버전 2.0은 $elemMatch의 평면화된 배열에서 숫자 인덱스 조회를 허용합니다.

```
doc: {"x" : [ [ { "y" : 1 } ] ] }

// Planner Version 2 and mongo
> db.bar.find({"x.0": {$elemMatch: {y: 1}}})
{ "_id" : ObjectId("68192947945e5846634c455a"), "x" : [ [ { "y" : 1 } ] ] }
> db.bar.find({"x": {$elemMatch: {"0.y": 1}}})
{ "_id" : ObjectId("68192947945e5846634c455a"), "x" : [ [ { "y" : 1 } ] ] }

//Whereas Planner Version 1 wouldn't return any results.
> db.bar.find({"x.0": {$elemMatch: {y: 1}}})
> db.bar.find({"x": {$elemMatch: {"0.y": 1}}})
```

- 플래너 버전 1.0은 프로젝션에서 문자열을 제외했지만 플래너 버전 2.0의 동작은 MongoDB와 일치하고 이를 리터럴 값으로 취급합니다."

```
// Planner V2/ MongoDB
> db.col.find()
{ "_id" : ObjectId("681537738aa101903ed2fe05"), "x" : 1, "y" : 1 }
> db.col.find({},{x:"string"})
{ "_id" : ObjectId("681537738aa101903ed2fe05"), "x" : "string" }

// Planner V1 treats strings as exclude in projection
rs0:PRIMARY> db.col.find()
{ "_id" : ObjectId("68153744d42969f11d5cca72"), "x" : 1, "y" : 1 }
rs0:PRIMARY> db.col.find({},{x:"string"})
{ "_id" : ObjectId("68153744d42969f11d5cca72"), "y" : 1 }
```

- MongoDB와 같은 플래너 버전 2.0은 동일한 필드 "x" 및 "x.a"에서 프로젝션을 허용하지 않습니다.

```
// Planner version 2/MongoDB will error out
> db.col.find()
{ "_id" : ObjectId("68153da2012265816bc9ba23"), "x" : [ { "a" : 1 }, 3 ] }
 db.col.find({},{"x.a":1,"x":1}) // error

// Planner Version 1 does not error out
db.col.find()
{ "_id" : ObjectId("68153da2012265816bc9ba23"), "x" : [ { "a" : 1 }, 3 ] }

db.col.find({},{"x.a":1,"x":1})
{ "_id" : ObjectId("68153d60143af947c720d099"), "x" : [ { "a" : 1 }, 3 ] }
```

- MongoDB와 같은 플래너 버전 2.0은 하위 문서에 대한 프로젝션을 허용합니다.

```
// Planner Version2/MongoDB supports projections on subdocuments
 db.col.find()
{ "_id" : ObjectId("681542d8f35ace71f0a50004"), "x" : [ { "y" : 100 } ] }
> db.col.find({},{"x":{"y":1}})
{ "_id" : ObjectId("681542b7a22d548e4ac9ddea"), "x" : [ { "y" : 100 } ] }

// Planner V1 throws error if projection is subdocument
 db.col.find()
{ "_id" : ObjectId("681542d8f35ace71f0a50004"), "x" : [ { "y" : 100 } ] }
rs0:PRIMARY> db.col.find({},{"x":{"y":1}})
Error: error: {
    "ok" : 0,
    "operationTime" : Timestamp(1746223914, 1),
    "code" : 2,
    "errmsg" : "Unknown projection operator y"
}
```

- MongoDB와 같은 플래너 버전 2.0에서는 프로젝션이 $ 연산자 이후의 필드를 지원하지 않습니다.

```
// Mongo and Planner Version 2 will error out
  db.col.find()
{ "_id" : ObjectId("68155fa812f843439b593f3f"), "x" : [ { "a" : 100 } ] }
 db.col.find({"x.a":100},{"x.$.a":1}) - // error

// v1 will not error out
 db.col.find()
{ "_id" : ObjectId("68155fa812f843439b593f3f"), "x" : [ { "a" : 100 } ] }
db.col.find({"x.a":100},{"x.$.a":1})
```

```
{ "_id" : ObjectId("68155dee13b051d58239cd0a"), "x" : [ { "a" : 100 } ] }
```

- MongoDB와 같은 플래너 버전 2.0에서는 $hint를 사용할 수 있습니다.

```
// v1 will error out on $hint if there are no filters
db.col.find({}).hint("x_1")
Error: error: {
    "ok" : 0,
    "operationTime" : Timestamp(1746466616, 1),
    "code" : 2,
    "errmsg" : "Cannot use Hint for this Query. Index is multi key index , partial
 index or sparse index and query is not optimized to use this index."
}


// Mongo and Planner Version 2 will allow $hint usage
 db.col.find({}).hint("x_1")
{ "_id" : ObjectId("6818f790d5ba9359d68169cf"), "x" : 1 }
```

쿼리 플래너 v3

Amazon DocumentDB 8.0의 플래너 버전 3은 6개의 새 단계를 포함하여 21개의 집계 단계를 지원합니다. 플래너 V3는 고유한 명령에 대한 내장 지원을 제공합니다. Amazon DocumentDB 5.0에서 플래너 v2에 비해 최대 2배의 전체 성능 개선을 제공합니다. Amazon DocumentDB 8.0의 모든 새로운 기능 및 연산자는 플래너 v3와 호환됩니다. 플래너 v3에서 지원하는 Amazon DocumentDB 8.0의 새로운 집계 단계에는 $replaceWith, $vectorSearch, $merge, $set, $unset, $bucket이 포함됩니다. 또한 플래너 v3는 데이터 정렬, 뷰, $merge, $pow, $rand, $dateTrunc, $dateToParts, $dateFromParts를 포함한 Amazon DocumentDB 8.0의 새로운 기능과 연산자를 지원합니다.

주제

- 사전 조건
- 플래너 버전 3.0을 기본 쿼리 플래너로 선택
- 모범 사례
- 제한 사항
- aggregate 및 distinct 연산자에 대한 개선 사항
- 캐시 필터 계획
- 플래너 버전 1.0, 3.0 및 MongoDB 간의 잠재적 동작 차이

사전 조건

플래너 버전 3.0에는 다음 사전 조건이 적용됩니다.

- 플래너 버전 3.0은 엔진 버전 8.0을 사용할 수 AWS 리전 있는 모든에서 사용할 수 있습니다.
- 플래너 버전 3.0은 엔진 버전 8.0을 선택할 때 기본 쿼리 플래너입니다.

플래너 버전 3.0을 기본 쿼리 플래너로 선택

Amazon DocumentDB 8.0에서 기본 쿼리 플래너를 변경하고 플래너 v3로 되돌려야 하는 경우 콘솔 또는 CLI에서 변경할 수 있습니다.

- Amazon DocumentDB 클러스터 파라미터 수정의 단계에 따라 클러스터의 파라미터 그룹을 수정합니다.
- 'plannerVersion'이라는 파라미터의 경우 플래너 버전 3.0을 나타내는 값을 3.0으로 변경합니다.
- 즉시 적용을 선택합니다(재부팅 시 적용을 선택하는 경우 다음에 클러스터를 재부팅할 때까지 선택이 적용되지 않음).

모범 사례

예상 결과의 경우 플래너 버전 3.0을 적용할 때 다음 모범 사례를 사용하세요.

- 글로벌 클러스터에서 두 리전의 클러스터 파라미터 그룹에서 동일한 `plannerVersion` 값(1.0 또는 2.0 또는 3.0)을 선택합니다. 기본 및 보조 리전에서 다른 플래너 버전을 선택하면 쿼리 동작과 성능이 일관되지 않을 수 있습니다.
- 예약된 유지 관리 기간 동안 또는 감소된 트래픽 기간 동안 플래너 버전 3.0으로 업데이트하면 워크로드가 활발하게 실행 중일 때 플래너 버전이 변경되면 오류율이 증가할 수 있으므로 중단이 가장 적습니다.

제한 사항

플래너 버전 3.0에는 다음 제한 사항이 적용됩니다.

- 플래너 버전 3.0은 탄력적 클러스터에서 지원되지 않으므로 플래너 버전 1.0으로 돌아갑니다.

- 플래너 v1은 쿼리 최적화 프로그램이 특정 쿼리 계획을 선택하도록 'planHint' 사용을 허용하지만, 플래너 v3는 'planHint' 사용을 허용하지 않으며 내부 최적화를 사용하여 지정된 쿼리에 가장 적합한 계획을 선택합니다.

`aggregate` 및 `distinct` 연산자에 대한 개선 사항

플래너 버전 3.0에서는 $aggregate 단계와 $distinct 명령이 개선되었습니다. 다음은 가장 주목할 만한 개선 사항입니다.

- 플래너는 가능한 경우 파이프라인의 앞부분에서 $match 단계를 이동시켜 후속 단계에서 처리되는 문서 수를 줄입니다.

```
//Planner v1
db.orders.aggregate([
  { $project: { customerId: 1, orderDate: 1, totalAmount: 1 } },
  { $match: { customerId: { $gt: 1000 } } }
])

// Planner v3 pulls up the match since customerId exists in original documents
// Optimized internally as:
db.orders.aggregate([
  { $match: { customerId: { $gt: 1000 } } },  // Pulled up before project
  { $project: { customerId: 1, orderDate: 1, totalAmount: 1 } }
])
```

- 플래너는 $lookup 단계와 $unwind 단계를 동일한 필드에서 작동할 때 자동으로 결합하여 중간 데이터 처리를 줄이고 성능을 개선합니다.

```
                Example query:
//Planner v1
db.orders.aggregate([
  { $lookup: { from: "products", localField: "productId", foreignField: "_id", as:
 "productInfo" } },
  { $unwind: "$productInfo" },
  { $project: { orderDate: 1, "productInfo.name": 1, "productInfo.price": 1 } }
])

// Planner version 3.0 optimizes this internally by coalescing the $lookup and
 $unwind stages
```

- 플래너 버전 3.0에는 카디널리티가 낮은 인덱스에서 고유한 작업의 성능을 크게 개선하는 새로운 Distinct Scan 실행 전략이 도입되었습니다.

```
                 Example query:
//// If there is a low cardinality index on "category", you may see a query plan like
 below
db.explain().products.distinct("category")

"queryPlanner" : {
        "plannerVersion" : 3,
        "namespace" : "db.products",
        "winningPlan" : {
                "stage" : "AGGREGATE",
                "inputStage" : {
                        "stage" : "DISTINCT_SCAN",
                        "inputStage" : {
                                "stage" : "IXONLYSCAN",
                                "indexName" : "category_1",
                                "direction" : "forward"
                        }
                }
        }
}
```

캐시 필터 계획

플래너 버전 3.0은 인덱스 필터라고도 하는 계획 캐시 필터를 지원합니다. 이를 통해 플래너가 특정 쿼리 셰이프에 대해 고려하는 인덱스 세트를 제한할 수 있습니다. 필터는 데이터베이스 명령으로 설정되고 서버에 적용되므로 쿼리 회귀가 발생하는 경우 애플리케이션 코드를 수정하지 않고도 완화할 수 있습니다.

플래너 버전 3.0은 플래너 버전 2.0보다 더 많은 명령에 계획 캐시 필터를 적용합니다.

- `distinct` 명령에는 플래너 버전 3.0이 필요합니다.
- `aggregate` 명령에는 플래너 버전 3.0 및 Amazon DocumentDB 8.0.2가 필요합니다. 파이프라인이 선행 `$match` 단계를 통해 컬렉션을 읽을 때 필터가 적용됩니다.

- `$lookup` 또는 `$graphLookup` 단계의 외래 컬렉션에 설정된 필터는 해당 외래 컬렉션을 스캔하는 데 사용되는 인덱스를 구동합니다. 또한 플래너 버전 3.0 및 Amazon DocumentDB 8.0.2가 필요합니다.

명령 참조, 지원되는 쿼리 셰이프, 작업 예제 및 필터가 힌트와 상호 작용하는 방법에 대한 자세한 내용은 섹션을 참조하세요캐시 필터 계획.

플래너 버전 1.0, 3.0 및 MongoDB 간의 잠재적 동작 차이

경우에 따라 플래너 버전 3.0이 플래너 버전 1.0과 약간 다른 결과를 생성할 수 있습니다. 이 섹션에서는 이러한 가능성의 몇 가지 예를 살펴봅니다.

Feature Differences

- 빈 컬렉션에서 또는 매치와 같은 이전 단계가 모든 문서를 필터링할 때 플래너 v1은 출력 “필드”:0을 제공합니다. Planner v3 및 MongoDB는 출력을 제공하지 않습니다.
- Plannerv1의 경우 이전 단계에서 반환된 문서가 n개 미만인 경우 {"$skip":n}은 건너뛰지 않습니다. Plannerv3 및 MongoDB는 반환된 문서 수에 관계없이 올바르게 건너뜁니다.
- $lookup에서 참조되는 외래 컬렉션이 없는 경우 plannerv1에서 오류가 발생합니다. 플래너 v3와 MongoDB는 외래 컬렉션을 빈 컬렉션으로 취급하고 $lookup을 수행합니다.

```
db.coll.aggregate([
    {$lookup: {from: "does_not_exist", localField: "a", foreignField: "a", as:
 "c"}}
])
```

- Planner v1만 파이프라인에서 여러 $search를 사용할 수 있습니다. Planner v2/v3에서는 오류가 발생하고 MongoDB는 이를 지원하지 않습니다.

```
VectorSearch = { "$search": { "vectorSearch": {
   "vector": [0.2, 0.5, 0.8],
   "path": "vectorEmbedding",
   "similarity": "cosine",
   "k": 2,
   "efSearch": 1
   }}}
db.coll.aggregate([VectorSearch, VectorSearch])
```

- vectorSearch 단계가 첫 번째 단계가 아닌 경우에만 Planner v3가 작동합니다. 이 경우 MongoDB에서 오류가 발생하고 플래너 v1은 $vectorSearch 단계를 지원하지 않습니다.

```
VectorSearch = { {"$vectorSearch": {
    "queryVector": [0.2, 0.5, 0.8],
    "path": "vectorEmbedding",
    "similarity": "euclidean",
    "limit": 4,
    "numCandidates": 100} }

db.coll.aggregate([$match:{}, VectorSearch])
```

Amazon DocumentDB에서 지리 공간 데이터 쿼리

이 섹션에서는 Amazon DocumentDB를 사용하여 지리공간 데이터를 쿼리하는 방법을 다룹니다. 이 섹션을 읽고 나면 Amazon DocumentDB에서 지리공간 데이터를 저장, 쿼리 및 인덱싱하는 방법에 대한 답을 얻을 수 있을 것입니다.

주제

- 개요
- 지리공간 데이터 인덱싱 및 저장
- 지리 공간 데이터 쿼리
- 제한 사항

개요

지리정보의 일반적인 사용 사례에는 데이터로부터의 근접성 분석이 포함됩니다. 예: “시애틀에서 50마일 이내에 있는 모든 공항 찾기” 또는 “특정 위치에서 가장 가까운 레스토랑 찾기”. Amazon DocumentDB는 GeoJSON 사양을 사용하여 지리공간 데이터를 나타냅니다. GeoJSON은 좌표 공간에서 도형의 JSON 형식 지정을 위한 오픈 소스 사양입니다. GeoJSON 좌표는 경도와 위도를 모두 캡처하여 지구와 같은 구의 위치를 나타냅니다.

지리공간 데이터 인덱싱 및 저장

Amazon DocumentDB는 '포인트' GeoJSON 유형을 사용하여 지리공간 데이터를 저장합니다. 각 GeoJSON 문서(또는 하위 문서)는 일반적으로 다음 두 필드로 구성됩니다.

- type - 표시되는 도형으로, Amazon DocumentDB에 “좌표” 필드를 해석하는 방법을 알려줍니다. 현재 Amazon DocumentDB는 포인트만 지원합니다.
- 좌표 - 배열에서 객체로 표시되는 위도 및 경도 쌍 - [경도, 위도]

또한 Amazon DocumentDB는 2dsphere 인덱스를 사용하여 지리공간 데이터를 인덱싱합니다. Amazon DocumentDB는 인덱싱 포인트를 지원합니다. Amazon DocumentDB는 2dsphere 인덱싱을 통한 근접 쿼리를 지원합니다.

음식 배달 서비스를 위한 애플리케이션을 구축하는 시나리오를 가정해 보겠습니다. Amazon DocumentDB에 레스토랑 위도 및 경도 페어를 저장하려고 합니다. 이렇게 하려면 먼저 위도 및 경도 페어를 포함하는 지리 공간 필드에 인덱스를 생성합니다.

```
use restaurantsdb
db.usarestaurants.createIndex({location:"2dsphere"})
```

이 명령의 출력은 다음과 같이 표시됩니다:

```
{
 "createdCollectionAutomatically" : true,
 "numIndexesBefore" : 1,
 "numIndexesAfter" : 2,
 "ok" : 1
}
```

색인을 생성한 후에는 Amazon DocumentDB 컬렉션에 데이터 삽입을 시작할 수 있습니다.

```
db.usarestaurants.insert({
   "state":"Washington",
   "city":"Seattle",
   "name":"Thai Palace",
   "rating": 4.8,
   "location":{
      "type":"Point",
      "coordinates":[
         -122.3264,
         47.6009
      ]
   }
});

db.usarestaurants.insert({
   "state":"Washington",
   "city":"Seattle",
   "name":"Noodle House",
   "rating": 4.8,
   "location":{
```

```
      "type":"Point",
      "coordinates":[
        -122.3517,
         47.6159
      ]
   }
});

db.usarestaurants.insert({
   "state":"Washington",
   "city":"Seattle",
   "name":"Curry House",
   "rating": 4.8,
   "location":{
      "type":"Point",
      "coordinates":[
         -121.4517,
         47.6229
      ]
   }
});
```

지리 공간 데이터 쿼리

Amazon DocumentDB는 지리공간 데이터의 근접성, 포함 및 교차 쿼리를 지원합니다. 근접 쿼리의 좋은 예는 특정 거리 미만 및 다른 포인트(도시)와의 거리 초과 거리에 있는 모든 포인트(모든 공항)를 찾는 것입니다. 포함 쿼리의 좋은 예는 지정된 지역/폴리곤(뉴욕주)에 위치한 모든 포인트(모든 공항)을 찾는 것입니다. 교차로 쿼리의 좋은 예는 한 포인트(도시)와 교차하는 한 폴리곤(주)을 찾는 것입니다. 다음 지리공간 연산자를 사용하여 데이터에서 통찰력을 얻을 수 있습니다.

- **$nearSphere** - $nearSphere GeoJSON 포인트에서 가장 가까운 포인트부터 가장 먼 포인트까지 점을 찾을 수 있도록 지원하는 찾기 연산자입니다.
- **$geoNear** - $geoNear GeoJSON 포인트로부터의 거리 계산을 미터 단위로 지원하는 집계 연산자입니다.
- **$minDistance** - $minDistance는 중심점으로부터 지정된 최소 거리에 있는 문서를 필터링하는 데 $nearSphere 또는 $geoNear와 함께 사용되는 찾기 연산자입니다.
- **$maxDistance** - $maxDistance는 중심점으로부터 지정된 최대 거리에 있는 문서를 필터링하는 데 $nearSphere 또는 $geoNear와 함께 사용되는 찾기 연산자입니다.
- **$geoWithin** - $geoWithin 찾기 연산자로, 다각형과 같이 지정된 도형 내에 완전히 존재하는 지리공간 데이터가 포함된 문서를 찾는 데 사용할 수 있습니다.

- **$geoIntersects** - $geoIntersects 지리공간 데이터가 지정된 GeoJSON 객체와 교차하는 문서를 찾을 수 있도록 지원하는 찾기 연산자입니다.

> **Note**
>
> $geoNear 및 $nearSphere는 근접 쿼리에서 사용하는 GeoJSON 필드에는 2dsphere 인덱스가 필요합니다.

예제 1.

이 예제에서는 주소(포인트)로부터 가장 가까운 거리를 기준으로 정렬된 모든 레스토랑(포인트)을 찾는 방법을 알아봅니다.

이러한 쿼리를 수행하려면 $geoNear를 사용하여 포인트 집합과 다른 포인트 간의 거리를 계산할 수 있습니다. distanceMultiplier를 추가하여 킬로미터 단위로 거리를 측정할 수도 있습니다.

```
db.usarestaurants.aggregate([
   {
      "$geoNear":{
         "near":{
            "type":"Point",
            "coordinates":[
               -122.3516,
               47.6156
            ]
         },
         "spherical":true,
         "distanceField":"DistanceKilometers",
         "distanceMultiplier":0.001
      }
   }
])
```

위 명령은 지정된 포인트로부터의 거리(가장 가까운 곳부터 가장 먼 곳까지)를 기준으로 정렬된 식당을 반환합니다. 이 명령의 출력은 다음과 같이 표시됩니다:

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "state" : "Washington", "city" :
 "Seattle", "name" : "Noodle House", "rating" : 4.8, "location" : { "type" : "Point",
 "coordinates" : [ -122.3517, 47.6159 ] }, "DistanceKilometers" : 0.03422834547294996 }
```

```
{ "_id" : ObjectId("611f3da185009a81ad38e74a"), "state" : "Washington", "city" :
 "Seattle", "name" : "Thai Palace", "rating" : 4.8, "location" : { "type" : "Point",
 "coordinates" : [ -122.3264, 47.6009 ] }, "DistanceKilometers" : 2.5009390081704277 }
{ "_id" : ObjectId("611f3dae85009a81ad38e74c"), "state" : "Washington", "city" :
 "Seattle", "name" : "Curry House", "rating" : 4.8, "location" : { "type" : "Point",
 "coordinates" : [ -121.4517, 47.6229 ] }, "DistanceKilometers" : 67.52845344856914 }
```

> **Note**
>
> Amazon DocumentDB 8.0(엔진 패치 버전 4.0.5748) 이상에서는 `$geoNear` 단계의 `limit` 및 num 옵션과 문서 100개의 기본 제한이 더 이상 사용되지 않습니다. `limit` 및를 사용하는 다음 예제는 Amazon DocumentDB 8.0(엔진 패치 버전 4.0.4448) 및 이전 버전에만 num 적용됩니다.

쿼리의 결과 수를 제한하려면 `limit` 또는 num 옵션을 사용합니다.

`limit`:

```
db.usarestaurants.aggregate([
   {
      "$geoNear":{
         "near":{
            "type":"Point",
            "coordinates":[
               -122.3516,
               47.6156
            ]
         },
         "spherical":true,
         "distanceField":"DistanceKilometers",
         "distanceMultiplier":0.001,
         "limit": 10
      }
   }
])
```

num:

```
db.usarestaurants.aggregate([
   {
```

```
      "$geoNear":{
         "near":{
            "type":"Point",
            "coordinates":[
               -122.3516,
               47.6156
            ]
         },
         "spherical":true,
         "distanceField":"DistanceKilometers",
         "distanceMultiplier":0.001,
         "num": 10
      }
   }
])
```

예제 2.

이 예제에서는 특정 주소(포인트)에서 2km 이내에 있는 모든 레스토랑(포인트)을 찾는 방법을 알아봅니다. 이러한 쿼리를 수행하려면 GeoJSON $maxDistance 포인트에서 최소 $minDistance 및 최대 $nearSphere 범위 내에서 사용할 수 있습니다.

```
db.usarestaurants.find({
   "location":{
      "$nearSphere":{
         "$geometry":{
            "type":"Point",
            "coordinates":[
               -122.3516,
               47.6156
            ]
         },
         "$minDistance":1,
         "$maxDistance":2000
      }
   }
},
{
   "name":1
})
```

위 명령은 지정된 포인트로부터 최대 2km 거리에 있는 레스토랑을 반환합니다. 이 명령의 출력은 다음과 같이 표시됩니다:

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "name" : "Noodle House" }
```

제한 사항

Amazon DocumentDB는 폴리곤, 라인스트링, 멀티포인트, 멀티폴리곤, 멀티라인스트링 및 지오메트리 컬렉션의 쿼리 또는 인덱싱을 지원하지 않습니다.

부분 인덱스

부분 인덱스는 지정된 필터 기준을 충족하는 컬렉션의 문서를 색인합니다. 부분 인덱스 기능은 Amazon DocumentDB 5.0 인스턴스 기반 클러스터에서 지원됩니다.

주제

- 부분 인덱스 생성
- 지원되는 연산자
- 부분 인덱스를 사용한 쿼리
- 부분 인덱스 기능
- 부분 인덱스 제한 사항

부분 인덱스 생성

부분 인덱스를 생성하려면 `partialFilterExpression` 옵션과 함께 `createIndex()` 메서드를 사용합니다. 예를 들어, 다음 작업은 `isDelivered` 필드가 `OrderID` true인 문서를 색인하는 명령 컬렉션에 고유한 복합 인덱스를 생성합니다.

```
db.orders.createIndex(
  {"category": 1, "CustomerId": 1, "OrderId": 1},
  {"unique": true, "partialFilterExpression":
    {"$and": [
      {"OrderId": {"$exists": true}},
      {"isDelivered": {"$eq": false}}
    ]}
  }
)
```

지원되는 연산자

- $eq
- $exists
- $and(최상위 수준에서만)
- $gt/$gte/$lt/$lte(인덱스 스캔은 쿼리에 예측된 필터가 부분 필터 식과 정확히 일치하는 경우에만 사용됨)(제한 사항 참조)

부분 인덱스를 사용한 쿼리

부분 인덱스를 사용하면 다음 쿼리 패턴을 사용할 수 있습니다.

- 쿼리 조건자는 부분 인덱스 필터 식과 정확히 일치합니다.

```
db.orders.find({"$and": [
    {"OrderId": {"$exists": true}},
    {"isDelivered": {"$eq": false}}
  ]}).explain()
```

- 쿼리 필터의 예상 결과는 부분 필터의 논리적 하위 집합입니다.

```
db.orders.find({"$and": [
    {"OrderId": {"$exists": true}},
    {"isDelivered": {"$eq": false}},
    {"OrderAmount": {"$eq": "5"}}
  ]}).explain()
```

- 쿼리의 하위 조건자는 다른 인덱스와 함께 사용할 수 있습니다.

```
db.orders.createIndex({"anotherIndex":1})
db.orders.find({ "$or": [
      {"$and": [
        {"OrderId": {"$exists": true}},
        {"isDelivered": {"$eq": false}}
      ]},
      {"anotherIndex": {"$eq": 5}}
    ]
  }).explain()
```

> **Note**
>
> 쿼리 실행 계획기는 효율적인 경우 인덱스 스캔 대신 컬렉션 스캔을 사용하도록 선택할 수 있습니다. 이는 일반적으로 컬렉션의 많은 부분을 반환하는 매우 작은 컬렉션 또는 쿼리에 표시됩니다.

부분 인덱스 기능

부분 인덱스 나열

`getIndex` 작업을 사용하여 partialFilterExpression으로 부분 인덱스를 나열합니다. 예를 들어, 발행된 `getIndex` 작업은 키, 이름 및 partialfilterExpressions 필드가 포함된 부분 인덱스를 나열합니다.

```
db.orders.getIndexes()
```

이 예제는 다음 출력을 반환합니다.

```
[
  {
    "v" : 4,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "ecommerceApp.orders"
  },
  {
    "v" : 4,
    "unique" : true,
    "key" : {
      "category" : 1,
      "" : 1,
      "CustomerId" : 1,
      "OrderId" : 1
    },
    "name" : "category_1_CustID_1_OrderId_1",
    "ns" : "ecommerceApp.orders",
    "partialFilterExpression" : {
      "$and" : [
        {"OrderId": {"$exists": true}},
```

```
        {"isDelivered": {"$eq": false}}
      ]
    }
  }
]
```

동일한 키에 대한 여러 부분 필터 식: 순서

동일한 필드 조합(키: 순서)에 대해 서로 다른 부분 인덱스를 생성할 수 있습니다. 이러한 인덱스의 이름은 달라야 합니다.

```
db.orders.createIndex(
  {"OrderId":1},
  {
    name:"firstPartialIndex",
    partialFilterExpression:{"OrderId":{"$exists": true}}
  }
)
```

```
db.orders.createIndex(
  {"OrderId":1},
  {
    name:"secondPartialIndex",
    partialFilterExpression:{"OrderId":{"$gt": 1000}}
  }
)
```

getIndexes 작업을 실행하여 컬렉션의 모든 인덱스를 나열합니다.

```
db.orders.getIndexes()
```

이러한 예제는 다음 출력을 반환합니다.

```
[
  {
    "v" : 4,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "ecommerceApp.orders"
```

```
  },
  {
    "v" : 4,
    "key" : {
      "OrderId" : 1
    },
    "name" : "firstPartialIndex",
    "ns" : "ecommerceApp.orders",
    "partialFilterExpression" : {"OrderId":{"$exists": true}}
  },
  {
    "v" : 4,
    "key" : {
      "OrderId" : 1
    },
    "name" : "secondPartialIndex",
    "ns" : "ecommerceApp.orders",
    "partialFilterExpression" : {"OrderId":{"$gt": 1000}}
  }
]
```

> **Important**
>
> 인덱스 이름은 달라야 하며 이름으로만 삭제해야 합니다.

부분 및 TTL 속성이 있는 인덱스

인덱스 생성 중에 `partialFilterExpression` 및 `expireAfterSeconds` 옵션을 모두 지정하여 부분 및 TTL 속성을 가진 인덱스를 생성할 수도 있습니다. 이렇게 하면 컬렉션에서 이제 제거되는 문서를 더 잘 제어할 수 있습니다.

예를 들어 특정 기간 이후에 삭제할 문서를 식별하는 TTL 인덱스가 있을 수 있습니다. 이제 부분 인덱스 옵션을 사용하여 문서를 삭제하는 시기에 대한 추가 조건을 지정할 수 있습니다.

```
db.orders.createIndex(
    { "OrderTimestamp": 1 },
    {
        expireAfterSeconds: 3600 ,
        partialFilterExpression: { "isDelivered": { $eq: true } }
    }
)
```

이 예제는 다음 출력을 반환합니다.

```
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1,
        "operationTime" : Timestamp(1234567890, 1)
}
```

getIndexes 작업을 실행하여 컬렉션에 존재하는 인덱스를 나열합니다.

```
db.orders.getIndexes()
[
    {
        "v" : 4,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.orders"
    }
```

이 예제는 다음 출력을 반환합니다.

```
[
    {
        "v": 4,
        "key": {
            "_id": 1
        },
        "name": "_id_",
        "ns": "ecommerceApp.orders"
    },
    {
        "v": 4,
        "key": {
            "OrderTimestamp": 1
        },
        "name": "OrderTimestamp_1",
        "ns": "ecommerceApp.orders",
        "partialFilterExpression": {
            "isDelivered": {
```

```
                "$eq": true
            }
        },
        "expireAfterSeconds": 3600
    }
]
```

부분 인덱스 제한 사항

다음 제한 사항이 부분 인덱스 기능에 적용됩니다.

- Amazon DocumentDB의 불평등 쿼리는 쿼리 필터 예측이 `partialFilterExpression`과 정확히 일치하고 동일한 데이터 유형인 경우에만 부분 인덱스를 사용합니다.

  > **Note**
  >
  > 위 사례에 대해 IXSCAN을 강제로 적용하는 데도 `$hint`를 사용할 수 없습니다.

  다음 예제에서 `partialFilterExpression`은 `field1`에만 적용되고 `field2`에는 적용되지 않습니다.

  ```
  db.orders.createIndex(
    {"OrderAmount": 1},
    {"partialFilterExpression": { OrderAmount : {"$gt" : 5}}}
  )

  db.orders.find({OrderAmount : {"$gt" : 5}}) // Will use partial index
  db.orders.find({OrderAmount : {"$gt" : 6}}) // Will not use partial index
  db.orders.find({OrderAmount : {"$gt" : Decimal128(5.00)}}) // Will not use partial
   index
  ```

- 배열 연산자가 `partialFilterExpression`은 지원되지 않습니다. 다음 작업은 오류를 생성합니다.

  ```
  db.orders.createIndex(
    {"CustomerId":1},
    {'partialFilterExpression': {'OrderId': {'$eq': [1000, 1001, 1002]}}}
  )
  ```

- partialFilterExpression 필드에서는 다음 연산자가 지원되지 않습니다.

- `$all` (배열 연산자)
- `$mod` (배열 연산자)
- `$or`
- `$xor`
- `$not`
- `$nor`

- 필터 식과 필터의 데이터 유형은 동일해야 합니다.

Amazon DocumentDB를 사용하여 텍스트 검색 수행

Amazon DocumentDB의 기본 전체 텍스트 검색 기능(텍스트 인덱스 v1)을 사용하면 특수 용도 텍스트 인덱스를 사용하여 대규모 텍스트 데이터 세트에서 텍스트 검색을 수행할 수 있습니다. 이 섹션에서는 텍스트 인덱스 기능의 기능을 설명하고 Amazon DocumentDB에서 텍스트 인덱스를 생성하고 사용하는 방법에 대한 단계를 제공합니다. 텍스트 검색 제한 사항도 나열됩니다.

주제

- 지원되는 기능
- Amazon DocumentDB 텍스트 인덱스 사용
- MongoDB와의 차이점
- 모범 사례 및 지침
- 텍스트 인덱스 V2
- 제한 사항

지원되는 기능

Amazon DocumentDB 텍스트 검색은 다음 MongoDB API 호환 기능을 지원합니다.

- 단일 필드에 텍스트 인덱스를 생성합니다.
- 둘 이상의 텍스트 필드가 포함된 복합 텍스트 인덱스를 생성합니다.
- 단일 단어 또는 다중 단어 검색을 수행합니다.
- 가중치를 사용하여 검색 결과를 제어합니다.
- 점수별로 검색 결과를 정렬합니다.
- 집계 파이프라인에서 텍스트 인덱스를 사용합니다.

- 정확한 구문을 검색합니다.

Amazon DocumentDB 텍스트 인덱스 사용

문자열 데이터가 포함된 필드에 텍스트 인덱스를 생성하려면 다음과 같이 문자열 '텍스트'를 지정합니다.

단일 필드 인덱스:

```
db.test.createIndex({"comments": "text"})
```

이 인덱스는 지정된 컬렉션의 '의견' 문자열 필드에서 텍스트 검색 쿼리를 지원합니다.

둘 이상의 문자열 필드에 복합 텍스트 인덱스 생성:

```
db.test.createIndex({"comments": "text", "title":"text"})
```

이 인덱스는 지정된 컬렉션의 '의견' 및 '제목' 문자열 필드에서 텍스트 검색 쿼리를 지원합니다. 복합 텍스트 인덱스를 생성할 때 최대 30개의 필드를 지정할 수 있습니다. 생성된 텍스트 검색 쿼리는 모든 인덱싱된 필드를 쿼리합니다.

> **Note**
>
> 각 컬렉션에는 하나의 텍스트 인덱스만 허용됩니다.

Amazon DocumentDB 컬렉션에 텍스트 인덱스 나열

다음 예제와 같이 컬렉션`getIndexes()`에서를 사용하여 텍스트 인덱스를 포함한 인덱스를 식별하고 설명할 수 있습니다.

```
rs0:PRIMARY> db.test.getIndexes()
[
   {
      "v" : 4,
      "key" : {
         "_id" : 1
      },
      "name" : "_id_",
```

```
      "ns" : "test.test"
   },
   {
      "v" : 1,
      "key" : {
         "_fts" : "text",
         "_ftsx" : 1
      },
      "name" : "contents_text",
      "ns" : "test.test",
      "default_language" : "english",
      "weights" : {
         "comments" : 1
      },
      "textIndexVersion" : 1
   }
]
```

색인을 생성한 후에는 Amazon DocumentDB 컬렉션에 데이터 삽입을 시작합니다.

```
db.test.insertMany([{"_id": 1, "star_rating": 4, "comments": "apple is red"},
                    {"_id": 2, "star_rating": 5, "comments": "pie is delicious"},
                    {"_id": 3, "star_rating": 3, "comments": "apples, oranges - healthy
 fruit"},
                    {"_id": 4, "star_rating": 2, "comments": "bake the apple pie in the
 oven"},
                    {"_id": 5, "star_rating": 5, "comments": "interesting couch"},
                    {"_id": 6, "star_rating": 5, "comments": "interested in couch for
 sale, year 2022"}])
```

텍스트 검색 쿼리 실행

단일 단어 텍스트 검색 쿼리 실행

`$text` 및 `$search` 연산자를 사용하여 텍스트 검색을 수행해야 합니다. 다음 예제에서는 텍스트 인덱싱된 필드에 문자열 “apple” 또는 “apples”와 같이 다른 형식의 “apple”이 포함된 모든 문서를 반환합니다.

```
db.test.find({$text: {$search: "apple"}})
```

출력:

이 명령의 출력은 다음과 같이 표시됩니다.

```
{ "_id" : 1, "star_rating" : 4, "comments" : "apple is red" }
{ "_id" : 3, "star_rating" : 3, "comments" : "apples, oranges - healthy fruit" }
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven" }
```

다중 단어 텍스트 검색 실행

Amazon DocumentDB 데이터에 대해 다중 단어 텍스트 검색을 수행할 수도 있습니다. 아래 명령은 “apple” 또는 “pie”가 포함된 텍스트 인덱싱 필드가 있는 문서를 반환합니다.

```
db.test.find({$text: {$search: "apple pie"}})
```

출력:

이 명령의 출력은 다음과 같이 표시됩니다.

```
{ "_id" : 1, "star_rating" : 4, "comments" : "apple is red" }
{ "_id" : 2, "star_rating" : 5, "comments" : "pie is delicious" }
{ "_id" : 3, "star_rating" : 3, "comments" : "apples, oranges - healthy fruit" }
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven" }
```

다중 단어 구문 텍스트 검색 실행

다중 단어 구문 검색의 경우 다음 예제를 사용합니다.

```
db.test.find({$text: {$search: "\"apple pie\""}})
```

출력:

위 명령은 정확한 문구 “apple pie”가 포함된 텍스트 인덱싱 필드가 있는 문서를 반환합니다. 이 명령의 출력은 다음과 같이 표시됩니다.

```
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven" }
```

필터로 텍스트 검색 실행

텍스트 검색을 다른 쿼리 연산자와 결합하여 추가 기준에 따라 결과를 필터링할 수도 있습니다.

```
db.test.find({$and: [{star_rating: 5}, {$text: {$search: "interest"}}]})
```

출력:

위 명령은 모든 형태의 'interest'와 5와 동일한 'star_rating'이 포함된 텍스트 인덱싱 필드가 있는 문서를 반환합니다. 이 명령의 출력은 다음과 같이 표시됩니다.

```
{ "_id" : 5, "star_rating" : 5, "comments" : "interesting couch" }
{ "_id" : 6, "star_rating" : 5, "comments" : "interested in couch for sale, year
 2022" }
```

텍스트 검색에서 반환되는 문서 수 한도

`limit`를 사용하여 반환되는 문서 수를 제한하도록 선택할 수 있습니다.

```
db.test.find({$and: [{star_rating: 5}, {$text: {$search: "couch"}}]}).limit(1)
```

출력:

위 명령은 필터를 충족하는 결과 하나를 반환합니다.

```
{ "_id" : 5, "star_rating" : 5, "comments" : "interesting couch" }
```

텍스트 점수로 결과 정렬

다음 예제에서는 텍스트 검색 결과를 텍스트 점수별로 정렬합니다.

```
db.test.find({$text: {$search: "apple"}}, {score: {$meta: "textScore"}}).sort({score:
 {$meta: "textScore"}})
```

출력:

위 명령은 "apple" 또는 "apple"이 포함된 텍스트 인덱싱 필드가 포함된 문서를 "apples"와 같은 다른 형식으로 반환하고 문서가 검색어와 얼마나 관련이 있는지에 따라 결과를 정렬합니다. 이 명령의 출력은 다음과 같이 표시됩니다.

```
{ "_id" : 1, "star_rating" : 4, "comments" : "apple is red", "score" :
 0.6079270860936958 }
{ "_id" : 3, "star_rating" : 3, "comments" : "apples, oranges - healthy fruit",
 "score" : 0.6079270860936958 }
```

```
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven",
 "score" : 0.6079270860936958 }
```

$text 및 $search는 aggregate, count, findAndModify, update 및 delete 명령에도 지원됩니다.

집계 연산자

$match를 사용한 집계 파이프라인

```
db.test.aggregate(
   [{ $match: { $text: { $search: "apple pie" } } }]
)
```

출력:

위의 명령은 다음 결과를 반환합니다.

```
{ "_id" : 1, "star_rating" : 4, "comments" : "apple is red" }
{ "_id" : 3, "star_rating" : 3, "comments" : "apples, oranges - healthy fruit" }
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven" }
{ "_id" : 2, "star_rating" : 5, "comments" : "pie is delicious" }
```

다른 집계 연산자의 조합

```
db.test.aggregate(
   [
      { $match: { $text: { $search: "apple pie" } } },
      { $sort: { score: { $meta: "textScore" } } },
      { $project: { score: { $meta: "textScore" } } }
   ]
)
```

출력:

위의 명령은 다음 결과를 반환합니다.

```
{ "_id" : 4, "score" : 0.6079270860936958 }
{ "_id" : 1, "score" : 0.3039635430468479 }
{ "_id" : 2, "score" : 0.3039635430468479 }
```

```
{ "_id" : 3, "score" : 0.3039635430468479 }
```

텍스트 인덱스를 생성할 때 여러 필드 지정

복합 텍스트 인덱스에서 최대 3개의 필드에 가중치를 할당할 수 있습니다. 텍스트 인덱스의 필드에 할당된 기본 가중치는 일(1)입니다. 가중치는 선택적 파라미터이며 1~100,000 범위여야 합니다.

```
db.test.createIndex(
   {
     "firstname": "text",
     "lastname": "text",
     ...
   },
   {
     weights: {
       "firstname": 5,
       "lastname":10,
       ...
     },
     name: "name_text_index"
   }
 )
```

MongoDB와의 차이점

Amazon DocumentDB의 텍스트 인덱스 기능은 용어-주파수 알고리즘과 함께 반전된 인덱스를 사용합니다. 텍스트 인덱스는 기본적으로 희소입니다. 구문 분석 로직, 토큰화 구분 기호 등의 차이로 인해 MongoDB와 동일한 결과 세트가 동일한 데이터 세트 또는 쿼리 모양에 대해 반환되지 않을 수 있습니다.

Amazon DocumentDB 텍스트 인덱스와 MongoDB에는 다음과 같은 추가 차이점이 있습니다.

- 텍스트가 아닌 인덱스를 사용하는 복합 인덱스는 지원되지 않습니다.
- Amazon DocumentDB 텍스트 인덱스는 대/소문자를 구분하지 않습니다,
- 텍스트 인덱스에서는 영어만 지원됩니다.
- 배열(또는 다중 키) 필드의 텍스트 인덱싱은 지원되지 않습니다. 예를 들어 문서 {“a”:[“apple”, “pie”]}로 “a“에 텍스트 인덱스를 생성하면 실패합니다.
- 와일드카드 텍스트 인덱싱은 지원되지 않습니다.
- 고유한 텍스트 인덱스는 지원되지 않습니다.

- 용어 제외는 지원되지 않습니다.

모범 사례 및 지침

- 텍스트 점수별 정렬과 관련된 텍스트 검색 쿼리의 성능을 최적화하려면 데이터를 로드하기 전에 텍스트 인덱스를 생성합니다.
- 텍스트 인덱스에는 인덱싱된 데이터의 최적화된 내부 복사본을 위한 추가 스토리지가 필요합니다. 이로 인해 추가 비용이 발생합니다.

텍스트 인덱스 V2

Amazon DocumentDB 8.0에는 MongoDB와의 호환성을 높이기 위해 기본 텍스트 검색 구문 분석기를 변경하는 새로운 버전의 텍스트 인덱스(V2)가 도입되었습니다.

V1 텍스트 인덱스에서 제공하는 기능 외에도 V2 텍스트 인덱스는 다음과 같은 지원을 제공합니다.

- 플래너는 가능한 경우 파이프라인의 앞부분에서 $match 단계를 이동시켜 후속 단계에서 처리되는 문서 수를 줄입니다.
- 이메일, URLs 및 파일 경로와 같은 텍스트 필드의 특수 문자 토큰화를 개선했습니다. V2는 이러한 문자열 내의 개별 토큰을 구문 분석하고 일치시킬 수 있지만 V1은 그렇지 않습니다.

```
rs0:PRIMARY> db.coll.createIndex({ "a": "text" });
rs0:PRIMARY> db.coll.find()
{ "_id" : 1, "a" : "jane.doe_1234@company.com" }
{ "_id" : 2, "a" : "janedoe@company.com" }
{ "_id" : 3, "a" : "/home/user/company/thesis.pdf" }
{ "_id" : 4, "a" : "/home/user/path/jane.pdf" }
{ "_id" : 5, "a" : "http://www.company.com/path" }
{ "_id" : 6, "a" : "https://company.com/path/../home" }

// Sample query
rs0:PRIMARY> db.coll.find({ $text: { $search: "jane" } });

// V1 text index – no results
None

// V2 text index – matches tokens within emails and file paths
{ "_id" : 1, "a" : "jane.doe_1234@company.com" }
{ "_id" : 4, "a" : "/home/user/path/jane.pdf" }
```

제한 사항

Amazon DocumentDB 에서 텍스트 검색에는 다음과 같은 제한이 있습니다.

- 텍스트 인덱스는 어휘와 위치 정보를 저장합니다. 단일 문서 내에서 모든 어휘의 총 크기와 위치 정보는 1MB로 제한됩니다.

인덱스

인덱스는 Amazon DocumentDB가 컬렉션의 모든 문서를 스캔하지 않고도 문서를 빠르게 찾을 수 있도록 하여 쿼리 성능을 개선합니다. 애플리케이션에서 자주 쿼리하는 필드에 인덱스를 생성하여 지연 시간 및 I/O 비용을 줄일 수 있습니다.

추가 인덱스 관련 주제는 다음을 참조하세요.

- Amazon DocumentDB 인덱스 관리 - 인덱스 생성 및 빌드
- 인덱스 작업 - 인덱싱 모범 사례
- 인덱스 문제 해결 - 인덱스 빌드 실패 및 팽창 해결
- 인덱스 사용량을 분석하고 사용하지 않는 인덱스를 식별하려면 어떻게 해야 하나요? - 미사용 및 누락된 인덱스 찾기

주제

- 인덱스 관리
- 인덱스 유형
- 인덱스 속성

인덱스 관리

이 섹션에서는 Amazon DocumentDB의 인덱스 관리 주제를 다룹니다.

주제

- 인덱스 생성

- 인덱스 유지 관리
- 인덱스 작업 모범 사례

인덱스 생성

Amazon DocumentDB에서 인덱스를 구축하려면 다음과 같이 다양한 결정을 내려야 합니다:

- 얼마나 빨리 완료해야 할까요?
- 빌드가 진행되는 동안 컬렉션에 액세스할 수 없나요?
- 인스턴스의 컴퓨팅 성능을 빌드에 얼마나 할당할 수 있습니까?
- 어떤 유형의 인덱스를 만들어야 하나요?

이 섹션에서는 이러한 질문에 답하는 데 도움이 되며 인스턴스 기반 클러스터 컬렉션에서 Amazon DocumentDB 인덱스를 생성하는 명령 및 모니터링 예제를 제공합니다.

지침

다음 지침에는 새 인덱스를 생성할 때의 기본 제한 및 구성 장단점이 포함되어 있습니다:

- Amazon DocumentDB 버전 지원 - 모든 Amazon DocumentDB 버전에서 단일작업자 인덱싱이 지원되지만, 다중 작업자 인덱싱은 Amazon DocumentDB 버전 4.0 및 5.0에서만 지원됩니다.
- 성능 트레이드오프 - 인덱스 생성 프로세스의 작업자 수를 늘리면 Amazon DocumentDB 데이터베이스의 기본 인스턴스에서 CPU 사용률과 읽기 IO가 증가합니다. 새 인덱스를 생성하는 데 필요한 리소스는 실행 중인 워크로드에 사용할 수 없습니다.
- Elastic 클러스터 - Amazon DocumentDB 엘라스틱 클러스터에서는 병렬 인덱싱이 지원되지 않습니다.
- 서버리스 클러스터 - 서버리스 클러스터의 병렬 인덱싱에는 최소 32DCUs 용량이 필요합니다. 이 임계값 미만에서는 `workers` 파라미터가 허용되지만 인덱스 빌드는 단일 작업자로 실행됩니다. 구성된 최소 용량에 따라 조정되는 최대 병렬 작업자 수: 약 명의 `min_DCU / 8` 작업자를 사용할 수 있습니다. 예를 들어 32DCUs에서는 최대 4개의 작업자를 사용할 수 있고, 64DCUs에서는 최대 8개의 작업자를 사용할 수 있습니다.

> **Note**
>
> 새 병렬 작업자 제한이 적용되려면 최소 용량 설정을 변경한 후 클러스터를 다시 시작해야 합니다.

- 최대 작업자 수(인스턴스 기반) - 구성할 수 있는 최대 작업자 수는 데이터베이스 클러스터의 기본 인스턴스 크기에 따라 달라집니다. 데이터베이스 클러스터의 기본 인스턴스에 있는 총 vCPU 수의 절반입니다. 예를 들어 vCPU가 64개 있는 db.r6g.16xlarge 인스턴스에서 최대 32개의 작업자를 실행할 수 있습니다.

> **Note**
>
> 병렬 인덱스 빌드는 인스턴스 2xlarge 이상에서 지원됩니다.

- 최소 작업자 - 구성할 수 있는 최소 작업자 수는 1개입니다. 인스턴스 기반 클러스터의 인덱스 생성에 대한 기본 설정은 작업자 2명입니다. 그러나 "작업자 스레드" 옵션을 사용하여 작업자 수를 하나로 줄일 수 있습니다. 이렇게 하면 단일 작업자로 프로세스가 실행됩니다.
- 인덱스 압축 - Amazon DocumentDB는 인덱스 압축을 지원하지 않습니다. 저장된 데이터 및 인덱스의 데이터 크기는 다른 옵션을 사용할 때보다 클 수 있습니다.
- 여러 컬렉션 인덱싱 - 데이터베이스 클러스터의 기본 인스턴스에 있는 vCPU의 절반을 구성된 작업자가 여러 컬렉션에서 인덱스를 생성하는 데 사용할 수 있습니다.
- 인덱스 유형 - Amazon DocumentDB에서 지원되는 인덱스 유형 목록은 인덱스 유형 섹션을 참조하세요.

인덱스 빌드 유형

인덱스 빌드의 네 가지 유형은 다음과 같습니다.

- 포그라운드: 포그라운드 인덱스 빌드는 인덱스 생성이 완료될 때까지 인덱스가 생성되는 컬렉션의 다른 모든 데이터베이스 작업을 차단합니다. Amazon DocumentDB 포그라운드 빌드는 5단계로 구성되어 있습니다.
- 포그라운드(고유): 단일 문서(고유) 포그라운드 인덱스 빌드는 일반 포그라운드 빌드와 같은 다른 데이터베이스 작업을 차단합니다. 기본 포그라운드 빌드와 달리 고유 빌드는 추가 단계 (정렬 키 2) 를 사용하여 중복 키를 찾습니다. 포그라운드 (고유) 빌드는 6단계로 구성되어 있습니다.
- 배경: 백그라운드 인덱스 빌드를 사용하면 인덱스가 생성되는 동안 다른 데이터베이스 작업을 포그라운드에서 실행할 수 있습니다. Amazon DocumentDB 백그라운드 빌드는 8단계로 구성되어 있습니다.
- 백그라운드(고유): 단일 문서(고유) 백그라운드 인덱스 빌드를 사용하면 인덱스가 생성되는 동안 다른 데이터베이스 작업을 포그라운드에서 실행할 수 있습니다. 기본 백그라운드 빌드와 달리 고유 빌

드는 추가 단계 (정렬 키 2) 를 사용하여 중복 키를 찾습니다. 백그라운드 (고유) 빌드는 9단계로 구성되어 있습니다.

기본 인덱스 빌드 유형은 포그라운드입니다.

백그라운드에서 인덱스를 빌드하려면 인덱스를 생성할 때 "background": true 파라미터를 추가합니다.

시작하기

컬렉션에서 인덱스 생성을 시작하려면 createIndexes 명령을 사용합니다. 기본적으로 명령어는 인덱스 생성 프로세스의 속도를 두 배 증가시키는 두 개의 병렬 작업자를 실행합니다.

예를 들어 다음 명령 프로세스는 문서의 "user_name" 필드에 대한 인덱스를 생성하고 인덱싱 프로세스 속도를 작업자 4명으로 높이는 방법을 보여줍니다.

1. 클러스터에서 두 개의 병렬 작업자를 사용하여 백그라운드에서 인덱스를 생성합니다.

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx", "background":true }]})
```

2. 인덱스 생성 프로세스의 속도를 최적화하기 위해 db.runCommand createIndexes 명령의 '작업자 스레드' 옵션('작업자':<number>)을 사용하여 작업자 수를 지정할 수 있습니다.

   프로세스 속도를 병렬 작업자 4명으로 늘리십시오:

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx", "workers":4}]})
```

> **Note**
>
> 작업자 수가 많을수록 인덱스 생성 진행 속도가 빨라집니다. 하지만 작업자 수가 늘어날수록 기본 인스턴스의 vCPU 및 읽기 IO에 가해지는 부하도 증가합니다. 다른 워크로드를 저하시키지 않으면서 증가된 부담을 감당할 수 있을 만큼 클러스터를 충분히 프로비저닝해야 합니다.

인덱싱 진행 상태

인덱스 생성 프로세스는 컬렉션을 초기화하고, 스캔하고, 키를 정렬하고, 마지막으로 인덱스 빌더를 통해 키를 삽입하는 방식으로 진행됩니다. 프로세스를 포그라운드에서 실행할 경우 최대 6단계, 백그라운드에서 실행할 경우 최대 9단계로 구성됩니다. 단계별로 완료율, 스캔된 저장 블록의 총 수, 정렬된 키 및 삽입된 키와 같은 상태 메트릭을 볼 수 있습니다.

mongo 셸에서 db.currentOp() 명령을 사용하여 인덱싱 프로세스의 진행 상황을 모니터링합니다. 마지막 단계를 100% 완료하면 모든 색인이 성공적으로 생성되었음을 알 수 있습니다.

```
db.currentOp({"command.createIndexes": { $exists : true } })
```

> **Note**
>
> 인덱싱 진행 상태 보기는 Amazon DocumentDB 5.0에서만 지원됩니다.

인덱스 빌드 단계

단계	포그라운드	포그라운드 (고유)	배경	배경 (고유)
초기화 중	1	1	1	1
빌딩 인덱스: 초기화 중	2	2	2	2
빌딩 인덱스: 컬렉션 스캔	3	3	3	3
빌딩 인덱스: 정렬 키 1	4	4	4	4
빌딩 인덱스: 정렬 키 2		5		5

단계	포그라운드	포그라운드 (고유)	배경	배경 (고유)
빌딩 인덱스: 키 삽입	5	6	5	6
검증: 인덱스 스캔			6	7
검증: 튜플 정렬			7	8
검증: 컬렉션 스캔			8	9

- 초기화 중 - createIndex가 인덱스 빌더를 준비하고 있습니다. 이 단계는 매우 간단해야 합니다.
- 인덱스 작성: 초기화 중 - 인덱스 빌더가 인덱스 생성을 준비 중입니다. 이 단계는 매우 간단해야 합니다.
- 인덱스 작성: 컬렉션 스캔 - 인덱스 빌더가 인덱스 키를 수집하기 위해 컬렉션 스캔을 수행합니다. 측정 단위는 "블록"입니다.

> **Note**
>
> 인덱스 빌드에 대해 둘 이상의 작업자가 구성된 경우 이 단계에 표시됩니다. "컬렉션 스캔" 단계는 인덱스 빌드 프로세스 중에 여러 작업자를 사용하는 유일한 단계입니다. 다른 모든 단계에는 작업자 한 명만 표시됩니다.

- 인덱스 작성: 정렬 키 1 - 인덱스 빌더가 수집된 인덱스 키를 정렬합니다. 측정 단위는 "키"입니다.
- 인덱스 작성: 정렬 키 2 - 인덱스 작성기가 죽은 튜플에 해당하는 수집된 인덱스 키를 정렬하고 있습니다. 이 단계는 고유한 인덱스 구축에만 존재합니다. 측정 단위는 "키"입니다.
- 인덱스 작성: 키 삽입 - 인덱스 빌더가 새 인덱스에 인덱스 키를 삽입합니다. 측정 단위는 "키"입니다.
- 유효성 검증: 인덱스 스캔 - createIndex는 인덱스를 스캔하여 검증이 필요한 키를 찾는 것입니다. 측정 단위는 "블록"입니다.
- 유효성 검증: 튜플 정렬 - createIndex는 인덱스 스캔 단계의 출력을 정렬합니다.

- 유효성 검증: 컬렉션 스캔 - createIndex는 컬렉션을 스캔하여 이전 두 단계에서 찾은 인덱스 키의 유효성을 검사합니다. 측정 단위는 "블록"입니다.

인덱스 빌드 출력 예제

다음 출력 예제(포그라운드 인덱스 빌드)에는 인덱스 생성 상태가 표시됩니다. "msg" 필드에는 빌드의 단계와 완료 비율을 표시하여 빌드 진행 상황이 요약되어 있습니다. "작업자" 필드는 인덱스 빌드의 해당 단계에서 사용된 작업자 수를 나타냅니다. "경과" 필드에는 완료 비율을 계산하는 데 사용된 실제 숫자가 표시됩니다.

> **Note**
>
> "currentIndexBuildName", "msg" 및 "progress" 필드는 Amazon DocumentDB 버전 4.0에서 지원되지 않습니다.

```
{
    "inprog" : [{
    …
        "command": {
            "createIndexes": "test",
            "indexes": [{
                "v": 2,
                "key": {
                    "user_name": 1
                },
                "name": "user_name_1"
            }],
            "lsid": {
                "id": UUID("094d0fba-8f41-4373-82c3-7c4c7b5ff13b")
            },
            "$db": "test"
        },
        "currentIndexBuildName": user_name_1,
        "msg": "Index Build: building index number_1, stage 6/6 building index:
 656860/1003520 (keys) 65%",
        "workers": 1,
        "progress": {
            "done": 656861,
            "total": 1003520
```

```
        },
    …
    ],

    "ok" : 1
}
```

문제 해결 리소스

인덱스 생성 문제 해결에 대한 다음 리소스를 참조하세요.

- 인덱스 빌드 실패
- 백그라운드 인덱스 빌드 지연 시간 문제 및 실패

인덱스 유지 관리

인덱스 팽창

Amazon DocumentDB는 다중 버전 동시성 제어(MVCC)를 사용하여 동시 트랜잭션을 관리합니다. 문서가 삭제되거나 업데이트되면 이전 버전은 컬렉션 및 인덱스에 '데드' 버전으로 유지됩니다. 폐영역 회수 프로세스는 향후 작업을 위해 이러한 데드 버전에서 공간을 자동으로 회수합니다.

인덱스 팽창은 데드 버전 또는 더 이상 사용되지 않는 인덱스 항목이 누적되거나 페이지 내에 조각화되면서 컬렉션의 인덱스가 커질 때 발생합니다. 보고된 백분율은 향후 인덱스 항목에서 사용할 수 있는 인덱스 공간의 양을 나타냅니다. 이 팽창은 버퍼 캐시와 스토리지 모두에서 공간을 소비합니다. 팽창을 제거하려면 인덱스를 다시 빌드해야 합니다.

예: 다음 명령을 실행하여 인덱스의 미사용 스토리지를 확인합니다.

```
db.coll.aggregate({$indexStats:{}});
```

다음과 비슷한 결과가 반환됩니다.

```
{
    "name" : "_id_",
    "key" : {
```

```
        "_id" : 1
    },
    "host" : "devbox-test.localhost.a2z.com:27317",
    "size" : NumberLong(827392),
    "accesses" : {
        "ops" : NumberLong(40000),
        "docsRead" : NumberLong(46049),
        "since" : ISODate("2025-04-03T21:44:51.251Z")
    },
    "cacheStats" : {
        "blksRead" : NumberLong(264),
        "blksHit" : NumberLong(140190),
        "hitRatio" : 99.8121
    },
    "unusedStorageSize" : {
        "unusedSizeBytes" : 409600,
        "unusedSizePercent" : 49.51
    }
}
```

전체 컬렉션을 스캔해야 하는 reIndex 명령을 사용하여 가동 중지 없이 인덱스를 재구축할 수 있습니다. reIndex를 사용한 인덱스 유지 관리를 참조하세요.

reIndex를 사용한 인덱스 유지 관리

reIndex는 인덱스를 다시 빌드하는 데 사용되는 명령입니다. 일반적으로 인덱스가 손상되거나 비효율적일 때 사용됩니다. 시간이 지남에 따라 많은 업데이트, 삽입 또는 삭제로 인해 인덱스에 미사용 스페이스가 누적되어 성능이 저하될 수 있습니다. 리인덱싱은 사용하지 않는 스페이스를 제거하고 인덱스의 효율성을 복원하는 데 도움이 됩니다.

reIndex 지침

- reIndex는 Amazon DocumentDB 5.0 이상에서만 지원됩니다.
- reIndex는 항상 백그라운드에서 실행됩니다.
- Amazon DocumentDB는 백그라운드에서 단일 인덱스의 재인덱스를 지원하므로 여러 작업자가 사용할 수 있습니다. 이전 인덱스는 reIndex 프로세스가 실행 중일 때 쿼리에서 사용할 수 있습니다.
- Amazon DocumentDB는 currentOp를 통한 인덱싱 진행률 보고서를 지원합니다. 인덱스 생성 중에 볼 수 있는 인덱스 빌드 단계와 유사한 인덱스 빌드 단계를 볼 수 있습니다. 유일한 차이점은 reIndex에는 고유 여부와 관계없이 항상 8개의 단계가 있다는 것입니다. "빌딩 인덱스: 정렬 키 2" 단계는 없습니다.

- reIndex는 동일한 컬렉션에서 인덱스 관련 명령인 createIndexes, dropIndexes, collMod, renameCollection을 제외한 모든 명령과 동시에 실행할 수 있습니다.
- reIndex는 현재 텍스트, 지리 공간 및 벡터 인덱스에 대해 지원되지 않습니다.
- 부분 인덱스에 대한 reIndex 지원은 엔진 버전 8.0.2 이상에서 사용할 수 있습니다.

reIndex 빌드

다음 명령을 사용하여 인덱스를 재구축합니다.

```
db.runCommand({ reIndex: "collection-name", index: "index-name"})
```

필요에 따라 재구축 프로세스에 할당된 작업자 수를 제어할 수도 있습니다.

```
db.runCommand({ reIndex: "collection-name", index: "index-name", workers: number })
```

Java를 사용한 인덱스 관리에 대한 자세한 내용은 섹션을 참조하세요Java를 사용한 Amazon DocumentDB의 인덱스 관리.

인덱스 작업 모범 사례

인덱스 빌드

Amazon DocumentDB로 데이터를 가져올 때, 대규모 데이터 세트를 가져오기 전에 인덱스를 생성해야 합니다. Amazon DocumentDB 인덱스 도구를 사용하여 실행 중인 MongoDB 인스턴스 또는 mongodump 디렉터리에서 인덱스를 추출하고 Amazon DocumentDB 클러스터에서 해당 인덱스를 생성할 수 있습니다. 마이그레이션에 대한 자세한 지침은 마이그레이션 및 업그레이드 섹션을 참조하세요.

인덱스 선택성

인덱스 생성을 중복 값 수가 컬렉션의 총 문서 수의 1% 미만인 필드로 제한합니다. 예를 들어 컬렉션에 100,000개의 문서가 포함된 경우 동일한 값이 1,000배 이하 발생하는 필드에만 인덱스를 생성합니다.

고유 값 수가 많은 인덱스(즉, 카디널리티가 높은 인덱스)를 선택하면 필터 작업이 적은 수의 문서를 반환하여 인덱스 스캔 중에 우수한 성능을 얻을 수 있습니다. 높은 카디널리티 인덱스의 예는 같음 조건자가 최대 하나의 문서를 반환하도록 보장하는 고유 인덱스입니다. 낮은 카디널리티의 예로는 부울 필

드에 대한 인덱스와 요일에 대한 인덱스가 있습니다. 성능 저하로 인해 데이터베이스의 쿼리 옵티마이저가 낮은 카디널리티 인덱스를 선택할 가능성은 낮습니다. 동시에 낮은 카디널리티 인덱스는 디스크 스페이스 및 I/O와 같은 리소스를 계속 사용합니다. 일반적으로 일반적인 값 빈도가 전체 컬렉션 크기의 1% 이하인 필드의 인덱스를 대상으로 해야 합니다.

또한 일반적으로 필터로 사용되는 필드에만 인덱스를 생성하고, 사용하지 않는 인덱스를 정기적으로 찾는 것이 좋습니다. 자세한 내용은 the section called “인덱스 사용량을 분석하고 사용하지 않는 인덱스를 식별” 섹션을 참조하세요.

인덱스가 데이터 작성에 미치는 영향

인덱스는 컬렉션의 모든 문서를 스캔할 필요가 없으므로 쿼리 성능을 향상할 수 있지만 이러한 개선 시에 다른 불리함이 따를 수 있습니다. 컬렉션의 각 인덱스에 대해 문서가 삽입, 업데이트 또는 삭제될 때마다 데이터베이스는 컬렉션을 업데이트하고 컬렉션의 각 인덱스에 필드를 기록해야 합니다. 예를 들어 컬렉션에 9개의 인덱스가 있는 경우 데이터베이스는 클라이언트에 작업을 승인하기 전에 10개의 쓰기를 수행해야 합니다. 따라서 각 추가 인덱스에는 추가 쓰기 지연 시간, I/O 및 전체 사용 스토리지 증가가 발생합니다.

모든 작업 집합 메모리를 유지하려면 클러스터 인스턴스의 크기를 적절하게 조정해야 합니다. 이에 따라 스토리지 볼륨에서 인덱스 페이지를 지속적으로 읽을 필요가 없으므로 성능에 부정적인 영향을 미치고 I/O 비용이 증가합니다. 자세한 내용은 인스턴스 크기 조정 섹션을 참조하세요.

최상의 성능을 얻으려면 일반 쿼리의 성능을 향상하는 데 필요한 인덱스만 추가하여 컬렉션의 인덱스 수를 최소화합니다. 워크로드는 다양하지만 컬렉션당 인덱스 수를 5 이하로 유지하는 것이 좋습니다.

누락된 인덱스 식별

가장 좋은 방법은 정기적으로 누락된 인덱스를 식별하는 것입니다. 자세한 내용은 the section called “누락된 인덱스를 식별” 섹션을 참조하세요.

사용되지 않는 인덱스 식별

사용하지 않는 인덱스를 정기적으로 식별하고 제거하는 것이 가장 좋습니다. 자세한 내용은 the section called “인덱스 사용량을 분석하고 사용하지 않는 인덱스를 식별” 섹션을 참조하세요.

인덱스 유형

주제

- 단일 필드 인덱스
- 복합 인덱스

- 다중 키 인덱스
- 텍스트 인덱스
- 지역 검색 인덱스
- 벡터 인덱스

단일 필드 인덱스

단일 필드 인덱스는 문서 모음의 단일 필드의 정보를 저장합니다. 기본적으로 모든 컬렉션에는 `_id` 필드에 인덱스가 있습니다. 인덱스를 추가하여 중요한 쿼리와 작업의 속도를 높일 수 있습니다.

단일 필드 인덱스는 다음과 같은 경우에 유용합니다.

- 애플리케이션은 특정 필드의 값을 기반으로 문서를 자주 쿼리하거나 필터링합니다.
- 특정 필드를 기준으로 문서를 효율적으로 정렬해야 합니다.
- 고유 인덱스를 생성하여 컬렉션의 문서 간에 특정 필드의 고유성을 보장하고자 합니다.

지원되는 인덱스 속성

옵션	3.6	4.0	5.0	8.0	탄력적 클러스터
이름	예	예	예	예	예
고유	예	예	예	예	예
스파스 *	예	예	예	예	예
partialFilterExpression*	아니요	아니요	예	예	아니요
expireAfterSeconds	예	예	예	예	예

* `sparse` 및 `partialFilterExpression` 옵션은 동일한 인덱스 정의에서 함께 사용할 수 없습니다. 이러한 옵션으로 인덱스를 생성하려고 하면 다음 오류와 함께 인덱스가 실패합니다.

```
Error in specification: cannot mix partialFilterExpression and sparse options
```

단일 필드 인덱스 생성

`createIndex()` 메서드를 사용하여 단일 필드 인덱스를 생성합니다. 구문은 다음과 같습니다. `db.collection.createIndex(<key>, <options>)`

`key` 파라미터는 필드와 인덱스 정렬 순서를 지정하는 JSON 문서입니다.

```
{
  "<field>": <1 (ascending)|-1 (descending)>
}
```

`options` 파라미터는 인덱스에 대한 옵션을 지정하는 JSON 문서입니다.

```
{
  "name": "<name>",
  "unique": <true | false>,
  "sparse": <true | false>,
  "partialFilterExpression": <filter expression>,
  "expireAfterSeconds": <seconds before expiry>
}
```

단일 필드 인덱스 생성 예제는 인덱스 속성을 참조하세요.

복합 인덱스

복합 인덱스는 문서 모음에 있는 두 개 이상의 필드의 정보를 저장하여 첫 번째 필드 또는 접두사 필드에 대한 쿼리를 허용합니다. 이러한 인덱스는 여러 필드를 동시에 필터링하거나 필터링을 정렬 작업과 결합하는 쿼리의 성능을 최적화합니다. 또한 가장 왼쪽에 인덱싱된 필드의 단일 조건 쿼리에도 효과적입니다. 데이터베이스는 이러한 인덱스 항목을 활용하여 전체 컬렉션 스캔을 수행하지 않고도 일치하는 문서를 효율적으로 찾습니다.

복합 필드 인덱스는 다음과 같은 경우에 유용합니다.

- 여러 필드를 동시에 필터링해야 합니다.
- 필터링을 정렬 작업과 결합해야 합니다.

지원되는 인덱스 속성

옵션	3.6	4.0	5.0	8.0	탄력적 클러스터
이름	예	예	예	예	예
고유	예	예	예	예	예
스파스 *	예	예	예	예	예
partialFilterExpression*	아니요	아니요	예	예	아니요

* `sparse` 및 `partialFilterExpression` 옵션은 동일한 인덱스 정의에서 함께 사용할 수 없습니다. 이러한 옵션으로 인덱스를 생성하려고 하면 다음 오류와 함께 인덱스가 실패합니다.

```
Error in specification: cannot mix partialFilterExpression and sparse options
```

복합 인덱스 생성

`createIndex()` 메서드를 사용하여 복합 인덱스를 생성합니다. 메서드 구문은 다음과 같습니다. `db.collection.createIndex(<keys>, <options>)`

`keys` 파라미터는 필드와 인덱스 정렬 순서를 지정하는 JSON 문서입니다.

```
{
  "<field 1>": <1 (ascending)|-1 (descending)>,
  "<field 2>": <1 (ascending)|-1 (descending)>,
  ...
}
```

단 하나의 필드만 복합 인덱스의 배열일 수 있습니다. 두 개 이상의 배열 필드에 복합 인덱스를 생성하려고 하면 다음 오류와 함께 실패합니다.

```
multiple fields of compound index cannot be arrays
```

`options` 파라미터는 인덱스에 대한 옵션을 지정하는 JSON 문서입니다.

```
{
  "name": "<name>",
  "unique": <true | false>,
  "sparse": <true | false>,
  "partialFilterExpression": <filter expression>
}
```

복합 인덱스 생성 예제는 인덱스 속성을 참조하세요.

다중 키 인덱스

배열 값이 있는 필드의 경우 다중 키 인덱스를 사용하면 배열의 각 요소에 대한 인덱스 키를 생성할 수 있습니다. 배열을 인덱싱하면 배열의 각 요소에 대한 인덱스 항목이 생성됩니다.

다중 키 인덱스는 애플리케이션이 배열의 값을 기반으로 문서를 자주 쿼리하거나 필터링할 때 유용합니다.

지원되는 인덱스 속성

옵션	3.6	4.0	5.0	8.0	탄력적 클러스터
이름	예	예	예	예	예
고유	예	예	예	예	예
스파스 *	예	예	예	예	예
partialFilterExpression*	아니요	아니요	예	예	아니요
expireAfterSeconds	예	예	예	예	예

* `sparse` 및 `partialFilterExpression` 옵션은 동일한 인덱스 정의에서 함께 사용할 수 없습니다. 이러한 옵션으로 인덱스를 생성하려고 하면 다음 오류와 함께 인덱스가 실패합니다.

```
Error in specification: cannot mix partialFilterExpression and sparse options
```

다중 키 인덱스 생성

`createIndex()` 메서드를 사용하여 다중 키 인덱스를 생성합니다. 메서드 구문은 다음과 같습니다. `db.collection.createIndex(<key>, <options>)`

`key` 파라미터는 필드와 인덱스 정렬 순서를 지정하는 JSON 문서입니다.

```
{
  "<field>": <1 (ascending)|-1 (descending)>
}
```

`options` 파라미터는 인덱스에 대한 옵션을 지정하는 JSON 문서입니다.

```
{
  "name": "<name>",
  "unique": <true | false>,
  "sparse": <true | false>,
  "partialFilterExpression": <filter expression>,
  "expireAfterSeconds": <seconds before expiry>
}
```

다음 예제에서는 이름이 인 오름차순으로 정렬된 `categories` 필드에 다중 키 인덱스를 생성합니다. `book_categories`

```
db.collection.createIndex(
  {
    "categories": 1
  },
  {
    "name": "book_categories"
  }
)
```

다중 키 인덱스 생성 예제는 인덱스 속성을 참조하세요.

텍스트 인덱스

텍스트 인덱스는 문서 내의 텍스트 필드에서 단어 또는 구문을 검색하는 데 유용하므로 전체 텍스트 검색 기능을 수행할 수 있습니다. 둘 이상의 텍스트 필드를 사용하여 단일 필드 텍스트 인덱스 또는 복합 텍스트 인덱스를 생성할 수 있습니다. 그러나 컬렉션당 하나의 텍스트 인덱스만 가질 수 있습니다. 자세한 내용은 Amazon DocumentDB를 사용하여 텍스트 검색 수행 섹션을 참조하세요.

지원되는 인덱스 속성

옵션	3.6	4.0	5.0	8.0	탄력적 클러스터
이름	아니요	아니요	예	예	아니요

텍스트 인덱스 생성

`createIndex()` 메서드를 사용하여 텍스트 인덱스를 생성합니다. 메서드 구문은 다음과 같습니다.
`db.collection.createIndex(<keys>, <options>)`

`keys` 파라미터는 필드(들)와 텍스트 인덱스 유형을 지정하는 JSON 문서입니다.

```
{
  "<field 1>": "text",
  "<field 2>": "text"
}
```

`options` 파라미터는 인덱스의 이름을 지정하는 JSON 문서입니다.

```
{
  "name": "<name>"
}
```

텍스트 인덱스 생성 예제는 인덱스 속성을 참조하세요.

지역 검색 인덱스

지리 공간 인덱스는 문서 모음 내에 저장된 지리 공간 데이터를 효율적으로 쿼리하고 관리하도록 설계된 특수한 유형의 인덱스입니다. Amazon DocumentDB는 구체(예: 지구)에서 지리 공간 데이터를 처리하도록 특별히 설계된 2Dsphere 인덱스를 지원합니다. 이렇게 하면 구형 지오메트리를 기반으로 정확한 계산 및 쿼리가 가능합니다.

지리 공간 인덱스는 애플리케이션이 다음과 같은 위치 기반 쿼리를 수행해야 할 때 유용합니다.

- 가까운 관심 지점 찾기
- 위치가 특정 영역에 속하는지 확인

- 위치 간 거리 계산

지원되는 인덱스 속성

옵션	3.6	4.0	5.0	8.0	탄력적 클러스터
이름	예	예	예	예	예

지리 공간 인덱스 생성

`createIndex()` 메서드를 사용하여 지리 공간 인덱스를 생성합니다. 메서드 구문은 다음과 같습니다. `db.collection.createIndex(<key>, <options>)`

`key` 파라미터는 필드와 2dsphere 인덱스 유형을 지정하는 JSON 문서입니다.

```
{
  "<field>": "2dsphere"
}
```

`options` 파라미터는 인덱스에 대한 옵션을 지정하는 JSON 문서입니다.

```
{
  "name": "<name>"
}
```

지리 공간 인덱스 생성 예제는 인덱스 속성을 참조하세요.

벡터 인덱스

벡터 인덱스는 문서 모음 내에 저장된 벡터 데이터를 효율적으로 쿼리하고 관리하도록 설계된 특수한 유형의 인덱스입니다. Amazon DocumentDB는 계층적 탐색 가능 스몰 월드(HNSW) 및 플랫 압축이 포함된 반전 파일(IVFFlat) 인덱스를 지원합니다.

자세한 내용은 Amazon DocumentDB에 대한 벡터 검색 섹션을 참조하세요.

벡터 인덱스는 다음과 같은 기계 학습 및 생성형 AI 사용 사례에 유용합니다.

- 시맨틱 검색

- 제품 권장 사항
- 개인화
- 챗봇
- 사기 탐지
- 이상 탐지

지원되는 인덱스 속성

옵션	3.6	4.0	5.0	8.0	탄력적 클러스터
이름	아니요	아니요	예	예	아니요

벡터 인덱스 생성

createIndex 명령을 `runCommand()` 메서드와 함께 사용하여 벡터 인덱스를 생성합니다. 구문은 다음과 같습니다.

```
db.runCommand({
  "createIndexes": "<collection>",
  "indexes": [{
    "key": {
      "<field>": "vector"
    },
    "name": "<name>",
    "vectorOptions": {
      "type": "<hnsw> | <ivfflat>",
      "dimensions": <number of dimensions>,
      "similarity": "<euclidean>|<cosine>|<dotProduct>",
      "lists": <number_of_lists> [applicable for IVFFlat],
      "m": <max number of connections> [applicable for HNSW],
      "efConstruction": <size of the dynamic list for index build> [applicable for
 HNSW]
    }
  }]
})
```

키 파라미터는 필드 및 벡터 인덱스 유형을 지정하는 JSON 문서입니다.

```
{
  "<field>": "vector"
}
```

벡터 인덱스 생성 예제는 인덱스 속성을 참조하세요.

인덱스 속성

인덱스 속성은 Amazon DocumentDB에서 인덱스의 동작과 특성을 정의합니다. 이러한 속성을 이해하면 애플리케이션의 특정 요구 사항을 충족하는 효율적인 인덱스를 설계하는 데 도움이 됩니다.

주제

- 인덱스 속성: 이름
- 인덱스 속성: 고유
- 인덱스 속성: 희소
- 인덱스 속성: partialFilterExpression
- 인덱스 속성: expireAfterSeconds
- 인덱스 속성: 숨김

인덱스 속성: 이름

지원되는 인덱스 유형

옵션	3.6	4.0	5.0	8.0	탄력적 클러스터
단일 필드	예	예	예	예	예
compound	예	예	예	예	예
다중 키	예	예	예	예	예
텍스트	아니요	아니요	예	예	아니요
지리 공간	예	예	예	예	예

옵션	3.6	4.0	5.0	8.0	탄력적 클러스터
벡터	아니요	아니요	예	예	아니요

이름 옵션을 사용하여 인덱스에 대한 선택적 이름을 제공합니다.

모든 예제에서는 다음 샘플 문서를 사용합니다.

```
{
  "productId": "PROD133726",
  "sku": "SKU24224",
  "name": "Basic Printer",
  "manufacturer": "The Manufacturer",
  "tags": [ "printer", "basic", "electronics", "business" ],
  "barcodes": [ "542364671", "886330670", "437445606" ],
  "reviews": [
    {
      "review_date": ISODate('2024-01-19T21:37:10.585Z'),
      ...
    }
  ],
  "material": "Polycarbonate",
  "color": "Space Gray",
  "supplier": {
    "supplierId": "SUP4",
    "location": {
      "type": "Point",
      "coordinates": [ -71.0589, 42.3601 ]
    }
  },
  "productEmbedding": [
    -0.019320633663838058,
    0.019672111388113596
  ],
  "lastUpdated": ISODate('2025-10-20T21:37:10.585Z')
}
```

단일 필드

```
db.collection.createIndex(
```

```
  {
    "productId": 1
  },
  {
    "name": "single_field_index"
  }
)
```

복합

```
db.collection.createIndex(
  {
    "productId": 1,
    "manufacturer": 1
  },
  {
    "name": "compound_index"
  }
)
```

다중 키

```
db.collection.createIndex(
  {
    "tags": 1
  },
  {
    "name": "multikey_index"
  }
)
```

텍스트

```
db.collection.createIndex(
  {
    "name": "text"
  },
  {
    "name": "text_index"
  }
)
```

Geospatial

```
db.collection.createIndex(
  {
    "supplier.location": "2dsphere"
  },
  {
    "name": "geospatial_index"
  }
)
```

벡터

```
db.runCommand({
  "createIndexes": "collection",
  "indexes": [{
    "key": {
      "productEmbedding": "vector"
    },
    "name": "hnsw_index",
    "vectorOptions": {
      "type": "hnsw",
      "dimensions": 2,
      "similarity": "euclidean",
      "m": 16,
      "efConstruction": 64
    }
  }]
})
```

인덱스 속성: 고유

지원되는 인덱스 유형

옵션	3.6	4.0	5.0	8.0	탄력적 클러스터
단일 필드	예	예	예	예	예
compound	예	예	예	예	예

옵션	3.6	4.0	5.0	8.0	탄력적 클러스터
다중 키	예	예	예	예	예

고유 옵션을 사용하여 컬렉션의 문서 간에 필드(들)의 고유성을 보장합니다.

예제

다음 예제에서는 다음 샘플 문서에서 고유한 인덱스를 생성하는 방법을 보여줍니다.

```
{
  "productId": "PROD133726",
  "sku": "SKU24224",
  "name": "Basic Printer",
  "manufacturer": "The Manufacturer",
  "tags": [ "printer", "basic", "electronics", "business" ],
  "barcodes": [ "542364671", "886330670", "437445606" ],
  "reviews": [
    {
      "review_date": ISODate('2024-01-19T21:37:10.585Z'),
      ...
    }
  ],
  "material": "Polycarbonate",
  "color": "Space Gray",
  "supplier": {
    "supplierId": "SUP4",
    "location": {
      "type": "Point",
      "coordinates": [ -71.0589, 42.3601 ]
    }
  },
  "productEmbedding": [
    -0.019320633663838058,
    0.019672111388113596
  ],
  "lastUpdated": ISODate('2025-10-20T21:37:10.585Z')
}
```

단일 필드

productId에 고유한 단일 필드 인덱스를 생성하여 동일한 productId가 둘 이상의 문서에 존재하지 않도록 합니다.

```
db.collection.createIndex(
  {
    "productId": 1
  },
  {
    name: "productId_unique",
    unique: true
  }
)
```

복합

sku 및 제조업체에서 고유한 복합 인덱스를 생성하여 두 개 이상의 문서에 동일한 sku 및 제조업체 조합이 존재하지 않도록 합니다.

```
db.collection.createIndex(
  {
    "sku": 1,
    "manufacturer": 1
  },
  {
    name: "sku_and_manufacturer_unique",
    unique: true
  }
)
```

다중 키

바코드에 고유한 다중 키 인덱스를 생성하여 바코드 배열의 값이 두 개 이상의 문서에 존재하지 않도록 합니다.

```
db.collection.createIndex(
  {
    "barcodes": 1
  },
  {
    name: "barcodes_unique",
    unique: true
```

```
  }
)
```

배열을 인덱싱하면 배열의 각 요소에 대한 인덱스 항목이 생성됩니다. 예를 들어 배열에 50개의 항목이 있는 경우 50개의 인덱스 항목이 있습니다. 따라서 고유한 다중 키 인덱스는 모든 개별 항목에 고유성을 적용합니다. 예를 들어 다음 문서는 값 배열 필드 인덱스의 고유한 제약 조건을 위반합니다.

```
{ "values": [ 1, 2, 3] }
{ "values": [ 3, 2 ] }   --> 3 and 2 already exist
{ "values": [ 1 ] }      --> 1 already exists
```

고유 인덱스가 있는 다음 동작에 유의하세요.

1. 두 개(또는 그 이상)의 문서가 인덱싱된 필드에 대해 동일한 값을 갖는 기존 데이터에 고유한 인덱스를 생성하면 인덱스 빌드가 실패하고 다음 오류가 발생합니다. `could not create unique index: <collection> index: <index name>`
2. 인덱싱된 필드의 값이 다른 문서의 해당 필드 값과 일치하는 문서를 삽입하면 삽입이 실패하고 다음 오류가 발생합니다. `E11000 duplicate key error collection: <collection> index: <index name>`
3. 인덱싱된 필드의 새 값이 다른 문서의 해당 필드 값과 일치하도록 기존 문서를 업데이트하면 업데이트가 실패하고 다음 오류가 발생합니다. `E11000 duplicate key error collection: <collection> index: <index name>`
4. 인덱싱된 필드가 문서에서 누락된 경우 값은 null로 처리됩니다. 인덱싱된 필드가 두 개(또는 그 이상의) 문서에서 누락된 경우 위에서 설명한 대로 인덱스 빌드, 삽입 및 업데이트가 실패합니다.

인덱스 속성: 희소

지원되는 인덱스 유형

옵션	3.6	4.0	5.0	8.0	탄력적 클러스터
단일 필드	예	예	예	예	예
compound	예	예	예	예	예
다중 키	예	예	예	예	예

희소 옵션을 사용하여 인덱싱된 필드가 누락된 문서의 인덱싱을 건너뛰어 인덱스 크기를 줄이고 메모리 공간을 절약할 수 있습니다. 인덱스 크기가 작기 때문에 인덱스를 사용하는 쿼리가 더 효율적입니다. 쿼리가 희소 인덱스를 활용하려면 인덱싱된 필드에 $exists 절을 사용해야 합니다. $exists 절을 생략하면 Amazon DocumentDB는 희소 인덱스를 사용하지 않습니다.

예제

다음 예제에서는 다음 샘플 문서에서 희소 인덱스를 생성하는 방법을 보여줍니다.

```
{
  "productId": "PROD133726",
  "sku": "SKU24224",
  "name": "Basic Printer",
  "manufacturer": "The Manufacturer",
  "tags": [ "printer", "basic", "electronics", "business" ],
  "barcodes": [ "542364671", "886330670", "437445606" ],
  "reviews": [
    {
      "review_date": ISODate('2024-01-19T21:37:10.585Z'),
      ...
    }
  ],
  "material": "Polycarbonate",
  "color": "Space Gray",
  "supplier": {
    "supplierId": "SUP4",
    "location": {
      "type": "Point",
      "coordinates": [ -71.0589, 42.3601 ]
    }
  },
  "productEmbedding": [
    -0.019320633663838058,
    0.019672111388113596
  ],
  "lastUpdated": ISODate('2025-10-20T21:37:10.585Z')
}
```

검토, 재료 및 색상 필드가 모든 문서에 있는 것은 아닙니다.

단일 필드

재료 필드에 희소 단일 필드 인덱스를 생성합니다.

```
db.collection.createIndex(
  {
    "material": 1
  },
  {
    "name": "material_sparse",
    "sparse": true
  }
)
```

이 인덱스는 재료 값이 있는 모든 제품을 찾을 때 사용됩니다.

```
db.collection.find({
  "material": {
    $exists: true
  }
})
```

복합

재료 및 색상 필드에 희소 복합 인덱스를 생성합니다.

```
db.collection.createIndex(
  {
    "material": 1,
    "color": 1
  },
  {
    "name": "material_and_color_sparse",
    "sparse": true
  }
)
```

이 인덱스는 재료 및 색상 값의 조합이 있는 모든 제품을 찾을 때 사용됩니다.

```
db.collection.find({
  "material": {
    $exists: true
  }
})
```

```
db.collection.find({
  "color": {
    $exists: true
  }
})

db.collection.find({
  $and: [
    {
      "material": {
        $exists: true
      }
    },
    {
      "color": {
        $exists:true
      }
    }
  ]
})

db.collection.find({
  $and: [
    {
      "color": {
        $exists: true
      }
    },
    {
      "material": {
        $exists:true
      }
    }
  ]
})
```

다중 키

리뷰 배열에 희소 다중 키 인덱스를 생성합니다.

```
db.collection.createIndex(
  {
    "reviews": 1
  },
```

```
  {
    "name": "reviews_sparse",
    "sparse": true
  }
)
```

이 인덱스는 리뷰 배열이 있는 모든 제품을 찾을 때 사용됩니다.

```
db.collection.find({
  "reviews": {
    $exists: true
  }
})
```

인덱스 속성: partialFilterExpression

지원되는 인덱스 유형

옵션	3.6	4.0	5.0	8.0	탄력적 클러스터
단일 필드	아니요	아니요	예	예	아니요
compound	아니요	아니요	예	예	아니요
다중 키	아니요	아니요	예	예	아니요
지리 공간	아니요	아니요	아니요	아니요	아니요

partialFilterExpression 옵션을 사용하여 지정된 필터 조건을 충족하는 문서만 포함하는 부분 인덱스를 생성합니다. 이렇게 하면 모든 문서를 인덱싱하고 인덱스 크기를 줄이고 메모리 공간을 절약하는 대신 컬렉션에서 문서의 하위 집합만 인덱싱하여 보다 효율적인 인덱스를 생성할 수 있습니다. 인덱스 크기가 작기 때문에 인덱스를 사용하는 쿼리가 더 효율적입니다. Amazon DocumentDB는 다음 시나리오에서 부분 인덱스를 사용합니다.

- 쿼리 조건자는 부분 인덱스 필터 표현식과 정확히 일치합니다.
- 쿼리 필터의 예상 결과는 부분 필터의 논리적 하위 집합입니다.
- 쿼리의 하위 조건자는 다른 인덱스와 함께 사용할 수 있습니다.

자세한 내용은 부분 인덱스 섹션을 참조하세요.

예제

다음 예제에서는 다음 샘플 문서에서 부분 인덱스를 생성하는 방법을 보여줍니다.

```
{
  "productId": "PROD133726",
  "sku": "SKU24224",
  "name": "Basic Printer",
  "manufacturer": "The Manufacturer",
  "tags": [ "printer", "basic", "electronics", "business" ],
  "barcodes": [ "542364671", "886330670", "437445606" ],
  "reviews": [
    {
      "review_date": ISODate('2024-01-19T21:37:10.585Z'),
      ...
    }
  ],
  "material": "Polycarbonate",
  "color": "Space Gray",
  "supplier": {
    "supplierId": "SUP4",
    "location": {
      "type": "Point",
      "coordinates": [ -71.0589, 42.3601 ]
    }
  },
  "productEmbedding": [
    -0.019320633663838058,
    0.019672111388113596
  ],
  "lastUpdated": ISODate('2025-10-20T21:37:10.585Z')
}
```

단일 필드

스페이스 그레이 색상의 제품에 대해 제조업체에 부분 단일 필드 인덱스를 생성합니다.

```
db.collection.createIndex(
  {
    "manufacturer": 1
  },
  {
```

```
    "name": "manufacturer_space_gray",
    "partialFilterExpression": {
      "color": {
        $eq: "Space Gray"
      }
    }
  }
)
```

이 인덱스는 제조업체별로 스페이스 그레이 색상 제품을 찾을 때 사용됩니다.

```
db.collection.find({
  "manufacturer": "The Manufacturer",
  "color": {
    $eq: "Space Gray"
  }
})
```

복합

폴리카보네이트 재료를 사용하는 제품의 제조업체 및 색상에 부분 복합 인덱스를 생성합니다.

```
db.collection.createIndex(
  {
    "manufacturer": 1,
    "color": 1
  },
  {
    "name": "manufacturer_and_color_polycarbonate",
    "partialFilterExpression": {
      "material": {
        $eq: "Polycarbonate"
      }
    }
  }
)
```

이 인덱스는 제조업체 및 색상별로 폴리카보네이트 재료 제품을 찾을 때 사용됩니다.

```
db.collection.find({
  "manufacturer": "The Manufacturer",
  "color": "Space Gray",
  "material": {
```

```
    $eq: "Polycarbonate"
  }
})
```

다중 키

제조업체가 있는 제품의 태그에 부분 다중 키 인덱스를 생성합니다.

```
db.collection.createIndex(
  {
    "tags": 1
  },
  {
    "name": "tags_the_manufacturer",
    "partialFilterExpression": {
      "manufacturer": {
        $eq: "The Manufacturer"
      }
    }
  }
)
```

이 인덱스는 제조업체가 있는 프린터로 태그가 지정된 제품을 찾을 때 사용됩니다.

```
db.collection.find({
  "tags": "printer",
  "manufacturer": {
    $eq: "The Manufacturer"
  }
})
```

인덱스 속성: expireAfterSeconds

지원되는 인덱스 유형

인덱스 유형	3.6	4.0	5.0	8.0	탄력적 클러스터
단일 필드	예	예	예	예	예
다중 키	예	예	예	예	예

expireAfterSeconds 옵션을 사용하여 TTL(Time To Live) 인덱스를 생성합니다. TTL 인덱스를 사용하면 각 문서의 제한 시간 조건에 따라 기간을 기준으로 문서를 삭제할 수 있습니다. 문서가 지정된 TTL 수명에 도달하면 컬렉션에서 삭제됩니다.

TTL 삭제 모범 사례는 섹션을 참조하세요TTL 및 시계열 워크로드.

예제

다음 예제에서는 다음 샘플 문서에서 TTL 인덱스를 생성하는 방법을 보여줍니다.

```
{
  "productId": "PROD133726",
  "sku": "SKU24224",
  "name": "Basic Printer",
  "manufacturer": "The Manufacturer",
  "tags": [ "printer", "basic", "electronics", "business" ],
  "barcodes": [ "542364671", "886330670", "437445606" ],
  "reviews": [
    {
      "review_date": ISODate('2024-01-19T21:37:10.585Z'),
      "rating": 4,
      "comment": "Good product"
    },
    {
      "review_date": ISODate('2024-03-15T14:22:33.120Z'),
      "rating": 5,
      "comment": "Excellent printer"
    },
    {
      "review_date": ISODate('2024-06-08T09:45:18.890Z'),
      "rating": 3,
      "comment": "Average quality"
    }
  ],
  "material": "Polycarbonate",
  "color": "Space Gray",
  "supplier": {
    "supplierId": "SUP4",
    "location": {
      "type": "Point",
      "coordinates": [ -71.0589, 42.3601 ]
    }
  },
  "productEmbedding": [
```

```
    -0.019320633663838058,
    0.019672111388113596
  ],
  "lastUpdated": ISODate('2025-10-20T21:37:10.585Z')
}
```

단일 필드

lastUpdated에 TTL 인덱스를 생성하여 90일 이내에 업데이트되지 않은 문서를 삭제합니다.

```
db.collection.createIndex(
  {
    "lastUpdated": 1
  },
  {
    "name": "lastUpdated_ttl",
    "expireAfterSeconds": 7776000
  }
)
```

다중 키

review.review_date에 TTL 인덱스를 생성하여 작년에 리뷰가 없는 문서를 삭제합니다.

```
db.collection.createIndex(
  {
    "reviews.review_date": 1
  },
  {
    "name": "reviews_review_date_ttl",
    "expireAfterSeconds": 31536000
  }
)
```

TTL 인덱스가 배열 필드에 있는 경우 배열의 모든 항목이 확인됩니다. 배열의 타임스탬프가 제한 시간 조건을 충족하는 경우 문서가 삭제됩니다.

인덱스 속성: 숨김

Amazon DocumentDB는 Amazon DocumentDB 8.0.1 이상 및 Amazon DocumentDB 5.0.2 이상으로 시작하는 hidden 인덱스 속성을 지원합니다.

숨겨진 인덱스는 쿼리 플래너에 표시되지 않으며 쿼리를 지원하는 데 사용할 수 없습니다. 플래너에서 인덱스를 숨기면 인덱스를 실제로 삭제하지 않고도 인덱스 삭제의 잠재적 영향을 평가할 수 있습니다. 영향이 음수인 경우 삭제된 인덱스를 다시 생성하는 대신 인덱스를 숨길 수 있습니다.

hidden 속성은 단일 필드, 복합, 멀티키, 텍스트, 지리 공간 및 벡터 인덱스를 포함한 모든 인덱스 유형에서 지원됩니다.

동작

플래너에서 숨기는 것 외에도 숨겨진 인덱스는 완전히 유지 관리되고 적용됩니다. 예제:

- 숨겨진 인덱스가 고유 인덱스인 경우 인덱스는 여전히 고유한 제약 조건을 문서에 적용합니다.
- 숨겨진 인덱스가 TTL 인덱스인 경우 인덱스는 여전히 문서를 만료시킵니다.
- 숨겨진 인덱스는 `listIndexes` 및 `db.collection.getIndexes()` 결과에 포함됩니다.
- 숨겨진 인덱스는 컬렉션에 대한 쓰기 작업에서 업데이트되며 디스크 공간과 메모리를 계속 사용합니다. 따라서 `db.collection.stats()` 및와 같은 통계 작업에 포함됩니다`$indexStats`.
- 숨기지 않은 인덱스를 숨기거나 숨김 인덱스를 숨기지 않으면가를 재설정합니다`$indexStats`. 이미 숨겨진 인덱스를 숨기거나 이미 숨기지 않은 인덱스를 숨기면가 재설정되지 않습니다`$indexStats`.

제한 사항

- `_id` 인덱스는 숨길 수 없습니다.
- `cursor.hint()`를 사용하여 숨겨진 인덱스를 강제로 사용할 수 없습니다.
- `$merge` on 필드에 필요한 고유 인덱스가 숨겨져 있으면가 실패합니다. 를 실행하기 전에 인덱스를 표시합니다`$merge`.

예제

숨겨진 인덱스 생성

숨겨진 상태에서 인덱스를 생성하려면 `db.collection.createIndex()` 메서드를 사용하고 인덱스 정의`true`에서 `hidden` 옵션을 로 설정합니다.

예를 들어 다음 작업은 `productId` 필드에 숨겨진 오름차순 인덱스를 생성합니다.

```
db.collection.createIndex(
  { "productId": 1 },
```

```
  { "name": "productId_1", "hidden": true }
)
```

확인하려면 컬렉션listIndexes에서 db.collection.getIndexes() 또는를 실행합니다. 숨겨진 인덱스는 정의"hidden": true에를 포함합니다.

```
db.collection.getIndexes()
[
  {
    "v": 4,
    "key": { "_id": 1 },
    "name": "_id_"
  },
  {
    "v": 4,
    "key": { "productId": 1 },
    "name": "productId_1",
    "hidden": true
  }
]
```

인덱스 옵션은 값이 hidden인 경우에만 반환됩니다true.

기존 인덱스 숨기기

collMod 명령을 사용하여 기존 인덱스를 숨깁니다. name 또는 로 인덱스를 식별하고를 hidden로 keyPattern설정합니다true.

다음 예제에서는 인덱스를 이름으로 숨깁니다.

```
db.runCommand({
  collMod: "collection",
  index: {
    name: "productId_1", // Specify the index name
    hidden: true
  }
})
```

키 패턴으로 인덱스를 식별할 수도 있습니다.

```
db.runCommand({
  collMod: "collection",
```

```
  index: {
    keyPattern: { productId: 1 }, // Specify the index key specification document
    hidden: true
  }
})
```

기존 인덱스 숨기기 해제

숨겨진 인덱스를 다시 표시하려면가 로 hidden 설정된 collMod 명령을 사용합니다false.

```
db.runCommand({
  collMod: "collection",
  index: { name: "productId_1", hidden: false }
})
```

인덱스는 숨겨져 있는 동안 완전히 유지되므로 숨기지 않으면 즉시 사용할 수 있습니다.

Amazon DocumentDB에서 변경 스트림 사용

Amazon DocumentDB(MongoDB 호환)의 변경 스트림 기능은 클러스터의 컬렉션 내에서 시간순으로 발생하는 변경 이벤트 시퀀스를 제공합니다. 변경 스트림에서 이벤트를 읽어 다음을 비롯한 다양한 사용 사례를 구현할 수 있습니다.

- 변경 알림
- Amazon OpenSearch Service(OpenSearch Service)로 전체 텍스트 검색
- Amazon Redshift로 분석

애플리케이션에서는 변경 스트림을 사용하여 개별 컬렉션의 데이터 변경 사항을 구독할 수 있습니다. 변경 스트림 이벤트는 클러스터에서 발생하는 순서대로 정렬되며 이벤트가 기록된 후 3시간(기본값) 동안 저장됩니다. 파라미터를 사용하여 보존 기간을 최대 7일까지 연장할 수 있습니다. change_stream_log_retention_duration 변경 스트림 보존 기간을 수정하려면 섹션을 참조하세요변경 스트림 로그 보존 기간 수정.

주제

- 지원되는 연산자
- 확장된 이벤트
- 결제
- 제한 사항

- 변경 스트림 활성화
- 예: Python에서 변경 스트림 사용
- 전체 문서 조회
- $changeStreamSplitLargeEvent를 사용하여 대규모 이벤트를 조각으로 분할
- 변경 스트림 재개
- startAtOperationTime에서 변경 스트림 재개
- postBatchResumeToken에서 변경 스트림 재개
- 변경 스트림 내 트랜잭션
- 변경 스트림 로그 보존 기간 수정
- 보조 인스턴스에서 변경 스트림 사용

지원되는 연산자

Amazon DocumentDB는 변경 스트림에 대해 다음 작업을 지원합니다.

- MongoDB `db.collection.watch()`, `db.watch()` 및 `client.watch()`API에서 지원되는 모든 변경 이벤트.
- 업데이트를 위한 전체 문서 조회.
- 집계 단계: `$match`, `$project`, `$redact`, `$addFields`, 및 `$replaceRoot`$changeStreamSplitLargeEvent.
- 이력서 토큰에서 변경 스트림 재개
- `startAtOperation` 항목을 사용하여 타임스탬프에서 변경 스트림 재개(Amazon DocumentDB 4.0+에 해당)
- `showExpandedEvents` 파라미터 `createCollection` 사용에 대한 DDL 이벤트 알림(엔진 버전 8.0.2부터 Amazon DocumentDB 8.0에 적용).
- `showExpandedEvents` 파라미터 `createIndexes` 사용에 대한 DDL 이벤트 알림(엔진 버전 8.0.2부터 Amazon DocumentDB 8.0에 적용).

확장된 이벤트

> **Note**
>
> 확장된 이벤트는 엔진 버전 8.0.2부터 Amazon DocumentDB 8.0에서 사용할 수 있습니다.

변경 스트림은 데이터 정의 언어(DDL) 이벤트 알림 createCollection 및를 지원합니다createIndexes. 확장된 이벤트를 포함하려면 showExpandedEvents 옵션이 있는 변경 스트림 커서를 엽니다.

예제:

mongosh Method

```
let cur = db.names.watch( [ ], {
   showExpandedEvents: true
} )
cur.next()
```

Aggregation Stage

```
let cur = db.names.aggregate( [
   { $changeStream: { showExpandedEvents: true } }
] )
cur.next()
```

결제

Amazon DocumentDB 변경 스트림 기능은 기본적으로 비활성화되어 있으며 이 기능이 활성화될 때까지 추가 요금이 발생하지 않습니다. 클러스터에서 변경 스트림을 사용하면 추가 읽기 및 쓰기 IO와 스토리지 비용이 발생합니다. modifyChangeStreams API 작업을 사용하여 클러스터에 대해 이 기능을 활성화할 수 있습니다. 요금에 대한 자세한 내용은 Amazon DocumentDB 요금을 참조하세요.

제한 사항

Amazon DocumentDB에서 변경 스트림에는 다음과 같은 제한 사항이 있습니다.

- Amazon DocumentDB 3.6. 및 Amazon DocumentDB 4.0에서 변경 스트림은 Amazon DocumentDB 클러스터의 기본 인스턴스에 대한 연결에서만 열 수 있습니다. Amazon DocumentDB 3.6. 및 Amazon DocumentDB 4.0에서는 복제본 인스턴스의 변경 스트림에서 읽기 기능이 지원되지 않습니다. watch() API 작업을 호출할 때 모든 읽기가 기본 인스턴스에 대해 수행되도록 primary 읽기 기본 설정을 지정해야 합니다(예제 섹션 참조).
- Amazon DocumentDB 5.0에서는 전역 클러스터를 포함하여 기본 인스턴스와 보조 인스턴스 모두에서 변경 스트림을 열 수 있습니다. 보조 읽기 기본 설정을 지정하여 변경 스트림을 보조 인스턴스로

리디렉션할 수 있습니다. 추가 모범 사례 및 제한 사항은 보조 인스턴스에서 변경 스트림 사용 항목을 참조하세요.

- 모음의 변경 스트림에 작성된 이벤트는 최대 7일 동안 사용할 수 있습니다(기본값은 3시간). 변경 스트림 데이터는 새 변경 사항이 발생하지 않은 경우에도 로그 보존 기간 후에 삭제됩니다.
- 모음에서 updateMany 또는 deleteMany 같은 장기 실행 쓰기 작업을 수행하는 경우, 장기 실행 쓰기 작업이 완료될 때까지 변경 스트림 이벤트의 쓰기를 일시적으로 중단할 수 있습니다.
- Amazon DocumentDB는 MongoDB 작업 로그(oplog)를 지원하지 않습니다.
- Amazon DocumentDB를 사용하면 지정된 컬렉션에서 변경 스트림을 명시적으로 활성화해야 합니다.
- 변경 스트림 이벤트의 총 크기(요청된 경우 변경 데이터 및 전체 문서 포함)가 보다 큰 경우 16 MB클라이언트는 변경 스트림에서 읽기 실패를 경험합니다. 이를 방지하려면 $changeStreamSplitLargeEvent 스테이지를 사용하여 이벤트를 조각으로 분할합니다.
- Amazon DocumentDB v3.6과 client.watch() 및 db.watch() 항목을 사용할 때는 현재 Ruby 드라이버가 지원되지 않습니다.
- 필드의 업데이트된 값이 이전 값과 동일한 경우 변경 스트림의 updateDescription 명령 출력은 Amazon DocumentDB와 MongoDB에서 다릅니다.
  - 제공된 필드가 $set 명령에 지정되어 있고 대상 값이 이미 소스 값과 동일한 경우 Amazon DocumentDB는 updateDescription 출력의 필드를 반환하지 않습니다.
  - MongoDB는 지정된 값이 현재 값과 같더라도 출력의 필드를 반환합니다.

변경 스트림 활성화

해당 데이터베이스 내의 모든 모음 또는 선택한 모음에 대해 Amazon DocumentDB 변경 스트림을 활성화할 수 있습니다. 다음은 mongo 쉘을 사용하여 다른 사용 사례에 변경 스트림을 활성화하는 방법의 예입니다. 데이터베이스 및 모음 이름을 지정할 때 빈 문자열은 와일드카드로 취급됩니다.

```
//Enable change streams for the collection "foo" in database "bar"
db.adminCommand({modifyChangeStreams: 1,
    database: "bar",
    collection: "foo",
    enable: true});
```

```
//Disable change streams on collection "foo" in database "bar"
db.adminCommand({modifyChangeStreams: 1,
    database: "bar",
    collection: "foo",
```

```
    enable: false});
```

```
//Enable change streams for all collections in database "bar"
db.adminCommand({modifyChangeStreams: 1,
    database: "bar",
    collection: "",
    enable: true});
```

```
//Enable change streams for all collections in all databases in a cluster
db.adminCommand({modifyChangeStreams: 1,
    database: "",
    collection: "",
    enable: true});
```

다음과 같은 경우 컬렉션에 대해 변경 스트림이 활성화됩니다.

- 데이터베이스와 컬렉션 모두 명시적으로 활성화되어 있습니다.
- 컬렉션을 포함하는 데이터베이스가 활성화되어 있습니다.
- 모든 데이터베이스가 활성화되어 있습니다.

상위 데이터베이스에도 변경 스트림이 활성화되어 있거나 클러스터의 모든 데이터베이스가 활성화되어 있는 경우 데이터베이스에서 컬렉션을 삭제해도 해당 컬렉션에 대한 변경 스트림이 비활성화되지 않습니다. 삭제된 컬렉션과 동일한 이름으로 새 컬렉션이 생성되면 해당 컬렉션에 대해 변경 스트림이 활성화됩니다.

`$listChangeStreams` 집계 파이프라인 단계를 사용하여 클러스터의 활성화된 모든 변경 스트림을 나열할 수 있습니다. Amazon DocumentDB에서 지원하는 모든 집계 단계는 추가 처리를 위해 파이프라인에서 사용할 수 있습니다. 이전에 활성화된 컬렉션이 비활성화된 경우 `$listChangeStreams` 출력에 나타나지 않습니다.

```
//List all databases and collections with change streams enabled
cursor = new DBCommandCursor(db,
    db.runCommand(
        {aggregate: 1,
        pipeline: [{$listChangeStreams: 1}],
        cursor:{}}));
```

```
//List of all databases and collections with change streams enabled
{ "database" : "test", "collection" : "foo" }
```

```
{ "database" : "bar", "collection" : "" }
{ "database" : "", "collection" : "" }
```

```
//Determine if the database “bar” or collection “bar.foo” have change streams enabled
cursor = new DBCommandCursor(db,
  db.runCommand(
      {aggregate: 1,
       pipeline: [{$listChangeStreams: 1},
                  {$match: {$or: [{database: "bar", collection: "foo"},
                                  {database: "bar", collection: ""},
                                  {database: "", collection: ""}]}}
                 ],
      cursor:{}}));
```

예: Python에서 변경 스트림 사용

다음은 컬렉션 수준의 Python에서 Amazon DocumentDB 변경 스트림을 사용하는 예입니다.

```
import os
import sys
from pymongo import MongoClient, ReadPreference

username = "DocumentDBusername"
password = <Insert your password>

clusterendpoint = "DocumentDBClusterEndpoint”
client = MongoClient(clusterendpoint, username=username, password=password, tls='true',
 tlsCAFile='global-bundle.pem')

db = client['bar']

#While ‘Primary’ is the default read preference, here we give an example of
#how to specify the required read preference when reading the change streams
coll = db.get_collection('foo', read_preference=ReadPreference.PRIMARY)
#Create a stream object
stream = coll.watch()
#Write a new document to the collection to generate a change event
coll.insert_one({'x': 1})
#Read the next change event from the stream (if any)
print(stream.try_next())

"""
```

```
Expected Output:
{'_id': {'_data': '015daf94f600000002010000000200009025'},
'clusterTime': Timestamp(1571788022, 2),
'documentKey': {'_id': ObjectId('5daf94f6ea258751778163d6')},
'fullDocument': {'_id': ObjectId('5daf94f6ea258751778163d6'), 'x': 1},
'ns': {'coll': 'foo', 'db': 'bar'},
'operationType': 'insert'}
"""

#A subsequent attempt to read the next change event returns nothing, as there are no
 new changes
print(stream.try_next())

"""
Expected Output:
None
"""

#Generate a new change event by updating a document
result = coll.update_one({'x': 1}, {'$set': {'x': 2}})
print(stream.try_next())

"""
Expected Output:
{'_id': {'_data': '015daf99d400000001010000000100009025'},
'clusterTime': Timestamp(1571789268, 1),
'documentKey': {'_id': ObjectId('5daf9502ea258751778163d7')},
'ns': {'coll': 'foo', 'db': 'bar'},
'operationType': 'update',
'updateDescription': {'removedFields': [], 'updatedFields': {'x': 2}}}
"""
```

다음은 데이터베이스 수준의 Python에서 Amazon DocumentDB 변경 스트림을 사용하는 예입니다.

```
import os
import sys
from pymongo import MongoClient

username = "DocumentDBusername"
password = <Insert your password>
clusterendpoint = "DocumentDBClusterEndpoint”
client = MongoClient(clusterendpoint, username=username, password=password, tls='true',
 tlsCAFile='global-bundle.pem')
```

```
db = client['bar']
#Create a stream object
stream = db.watch()
coll = db.get_collection('foo')
#Write a new document to the collection foo to generate a change event
coll.insert_one({'x': 1})

#Read the next change event from the stream (if any)
print(stream.try_next())

"""
Expected Output:
{'_id': {'_data': '015daf94f600000002010000000200009025'},
'clusterTime': Timestamp(1571788022, 2),
'documentKey': {'_id': ObjectId('5daf94f6ea258751778163d6')},
'fullDocument': {'_id': ObjectId('5daf94f6ea258751778163d6'), 'x': 1},
'ns': {'coll': 'foo', 'db': 'bar'},
'operationType': 'insert'}
"""
#A subsequent attempt to read the next change event returns nothing, as there are no
 new changes
print(stream.try_next())

"""
Expected Output:
None
"""

coll = db.get_collection('foo1')

#Write a new document to another collection to generate a change event
coll.insert_one({'x': 1})
print(stream.try_next())

"""
Expected Output: Since the change stream cursor was the database level you can see
 change events from different collections in the same database
{'_id': {'_data': '015daf94f600000002010000000200009025'},
'clusterTime': Timestamp(1571788022, 2),
'documentKey': {'_id': ObjectId('5daf94f6ea258751778163d6')},
'fullDocument': {'_id': ObjectId('5daf94f6ea258751778163d6'), 'x': 1},
'ns': {'coll': 'foo1', 'db': 'bar'},
'operationType': 'insert'}
```

```
"""
```

전체 문서 조회

변경 사항 업데이트 이벤트에는 전체 문서가 포함되지 않으며 변경된 내용만 포함됩니다. 사용 사례에 업데이트의 영향을 받는 전체 문서가 필요한 경우, 스트림을 열 때 전체 문서 조회를 활성화할 수 있습니다.

변경 스트림 업데이트 이벤트에 대한 `fullDocument` 문서는 문서 조회 시 업데이트된 문서의 최신 버전을 나타냅니다. 업데이트 작업과 `fullDocument` 조회 사이에 변경 사항이 발생한 경우 `fullDocument` 문서가 업데이트 당시의 문서 상태를 나타내지 않을 수 있습니다.

업데이트 조회가 활성화된 스트림 객체를 생성하려면 다음 예제를 사용하세요.

```
stream = coll.watch(full_document='updateLookup')

#Generate a new change event by updating a document
result = coll.update_one({'x': 2}, {'$set': {'x': 3}})

stream.try_next()
```

스트림 객체의 결과는 다음과 비슷합니다.

```
{'_id': {'_data': '015daf9b7c000000010100000100009025'},
'clusterTime': Timestamp(1571789692, 1),
'documentKey': {'_id': ObjectId('5daf9502ea258751778163d7')},
'fullDocument': {'_id': ObjectId('5daf9502ea258751778163d7'), 'x': 3},
'ns': {'coll': 'foo', 'db': 'bar'},
'operationType': 'update',
'updateDescription': {'removedFields': [], 'updatedFields': {'x': 3}}}
```

$changeStreamSplitLargeEvent를 사용하여 대규모 이벤트를 조각으로 분할

> **Note**
>
> $changeStreamSplitLargeEvent 단계는 엔진 버전 8.0.2부터 Amazon DocumentDB 8.0에서 사용할 수 있습니다.

$changeStreamSplitLargeEvent 스테이지를 사용하면 16MB보다 큰 변경 이벤트가 변경 스트림 커서를 통해 순차적으로 반환되는 조각으로 분할됩니다. 16MB보다 작은 이벤트는 분할되지 않은 상태로 반환됩니다. 스테이지는 최상위 필드 경계를 따라 이벤트를 분할합니다. 단일 필드 내에서 분할되거나 하위 문서로 반복되지 않습니다. 따라서 단일 최상위 필드 자체가 16MB보다 큰 경우 스테이지는 유효한 조각을 생성할 수 없으며 대신 오류를 반환합니다.

가장 좋은 방법은 이벤트 크기를 줄여 $changeStreamSplitLargeEvent 가능하면 사용을 피하는 것입니다.

$changeStreamSplitLargeEvent 단계는 $changeStream 파이프라인에서만 사용할 수 있으며 마지막 단계로 한 번만 사용해야 합니다.

각 조각에는 자체 resumeToken 및 자체이 있습니다postBatchResumeToken. 비최종 조각 토큰에서 재개하면(일반 재개 토큰 또는 postBatchResumeToken 사용) 후속 조각이 전달됩니다. 토큰이 이벤트의 최종 조각에 해당하는 경우 변경 스트림 커서는 후속 이벤트를 반환합니다.

변경 스트림 재개

마지막으로 검색된 변경 이벤트 문서의 _id 필드와 동일한 다시 시작 토큰을 사용하여 나중에 변경 스트림을 다시 시작할 수 있습니다.

```
import os
import sys
from pymongo import MongoClient

username = "DocumentDBusername"
password = <Insert your password>
clusterendpoint = "DocumentDBClusterEndpoint”
client = MongoClient(clusterendpoint, username=username, password=password, tls='true',
 tlsCAFile='global-bundle.pem', retryWrites='false')

db = client['bar']
coll = db.get_collection('foo')
#Create a stream object
stream = db.watch()
coll.update_one({'x': 1}, {'$set': {'x': 4}})
event = stream.try_next()
token = event['_id']
print(token)

"""
Output: This is the resume token that we will later us to resume the change stream
```

```
{'_data': '015daf9c5b0000000101000000100009025'}
"""
#Python provides a nice shortcut for getting a stream's resume token
print(stream.resume_token)

"""
Output
{'_data': '015daf9c5b0000000101000000100009025'}
"""
#Generate a new change event by updating a document
result = coll.update_one({'x': 4}, {'$set': {'x': 5}})
#Generate another change event by inserting a document
result = coll.insert_one({'y': 5})
#Open a stream starting after the selected resume token
stream = db.watch(full_document='updateLookup', resume_after=token)
#Our first change event is the update with the specified _id
print(stream.try_next())

"""
#Output: Since we are resuming the change stream from the resume token, we will see all
 events after the first update operation. In our case, the change stream will resume
 from the update operation {x:5}

{'_id': {'_data': '015f7e8f0c0000000601000000006000fe038'},
'operationType': 'update',
'clusterTime': Timestamp(1602129676, 6),
'ns': {'db': 'bar', 'coll': 'foo'},
'documentKey': {'_id': ObjectId('5f7e8f0ac423bafbfd9adba2')},
'fullDocument': {'_id': ObjectId('5f7e8f0ac423bafbfd9adba2'), 'x': 5},
'updateDescription': {'updatedFields': {'x': 5}, 'removedFields': []}}
"""
#Followed by the insert
print(stream.try_next())

"""
#Output:
{'_id': {'_data': '015f7e8f0c0000000701000000007000fe038'},
'operationType': 'insert',
'clusterTime': Timestamp(1602129676, 7),
'ns': {'db': 'bar', 'coll': 'foo'},
'documentKey': {'_id': ObjectId('5f7e8f0cbf8c233ed577eb94')},
'fullDocument': {'_id': ObjectId('5f7e8f0cbf8c233ed577eb94'), 'y': 5}}
"""
```

startAtOperationTime에서 변경 스트림 재개

startAtOperationTime을 사용하여 특정 타임스탬프에서 나중에 변경 스트림을 재개할 수 있습니다.

> **Note**
>
> startAtOperationTime을 사용하는 기능은 Amazon DocumentDB 4.0 이상에서 사용할 수 있습니다. startAtOperationTime을 사용하는 경우, 변경 스트림 커서는 지정된 타임스탬프 시점 또는 이후에 발생한 변경 사항만 반환합니다. startAtOperationTime 및 resumeAfter 명령은 상호 배타적이므로 함께 사용할 수 없습니다.

```
import os
import sys
from pymongo import MongoClient

username = "DocumentDBusername"
password = <Insert your password>
clusterendpoint = "DocumentDBClusterEndpoint”
client = MongoClient(clusterendpoint, username=username, password=password, tls='true',
 tlsCAFile='rds-root-ca-2020.pem',retryWrites='false')
db = client['bar']
coll = db.get_collection('foo')
#Create a stream object
stream = db.watch()
coll.update_one({'x': 1}, {'$set': {'x': 4}})
event = stream.try_next()
timestamp = event['clusterTime']
print(timestamp)
"""
Output
Timestamp(1602129114, 4)
"""
#Generate a new change event by updating a document
result = coll.update_one({'x': 4}, {'$set': {'x': 5}})
result = coll.insert_one({'y': 5})
#Generate another change event by inserting a document
#Open a stream starting after specified time stamp

stream = db.watch(start_at_operation_time=timestamp)
print(stream.try_next())
```

```
"""
#Output: Since we are resuming the change stream at the time stamp of our first update
 operation (x:4), the change stream cursor will point to that event
{'_id': {'_data': '015f7e941a0000000301000000003000fe038'},
'operationType': 'update',
'clusterTime': Timestamp(1602130970, 3),
'ns': {'db': 'bar', 'coll': 'foo'},
'documentKey': {'_id': ObjectId('5f7e9417c423bafbfd9adbb1')},
'updateDescription': {'updatedFields': {'x': 4}, 'removedFields': []}}
"""

print(stream.try_next())
"""
#Output: The second event will be the subsequent update operation (x:5)
{'_id': {'_data': '015f7e9502000000050100000005000fe038'},
'operationType': 'update',
'clusterTime': Timestamp(1602131202, 5),
'ns': {'db': 'bar', 'coll': 'foo'},
'documentKey': {'_id': ObjectId('5f7e94ffc423bafbfd9adbb2')},
'updateDescription': {'updatedFields': {'x': 5}, 'removedFields': []}}
"""

print(stream.try_next())

"""
#Output: And finally the last event will be the insert operation (y:5)
{'_id': {'_data': '015f7e9502000000060100000006000fe038'},
'operationType': 'insert',
'clusterTime': Timestamp(1602131202, 6),
'ns': {'db': 'bar', 'coll': 'foo'},
'documentKey': {'_id': ObjectId('5f7e95025c4a569e0f6dde92')},
'fullDocument': {'_id': ObjectId('5f7e95025c4a569e0f6dde92'), 'y': 5}}
"""
```

postBatchResumeToken에서 변경 스트림 재개

이제 Amazon DocumentDB 변경 스트림은 postBatchResumeToken이라는 추가 필드를 반환합니다. 이 필드는 $changestream 명령 및 getMore 명령에서 반환됩니다.

Python의 $changestream 명령 예제:

```
db.command({"aggregate": "sales", "pipeline": [{ "$changeStream": {}}], "cursor":
 {"batchSize": 1}})
```

예상 결과:

```
cursor" : {
   "firstBatch" : [ ],
   "postBatchResumeToken" : {"_data" : "0167c8cbe60000000004"},
   "id" : NumberLong("9660788144470"),
   "ns" : "test.sales"
}
```

Python의 getMore 명령 예제:

```
db.command({"getMore": NumberLong(<cursor id>), "collection": "sales", "batchSize":
 1 })
```

예상 결과

```
cursor" : {
   "nextBatch" : [ ],
   "postBatchResumeToken" : {"_data" : "0167c8cbe60000000004"},
   "id" : NumberLong("9660788144470"),
   "ns" : "test.sales"
}
```

postBatchResumeToken 필드는 재개 토큰이 사용되는 방식과 마찬가지로 resumeAfter 필드에서 새 변경 스트림 커서를 여는 데 사용할 수 있습니다.

선택한 postBatchResumeToken 이후에 시작하는 스트림을 엽니다.

```
post_batch_resume_token = output['cursor']['postBatchResumeToken']
stream = db.watch(full_document='updateLookup', resume_after=post_batch_resume_token)
```

실제 이벤트를 반영하는 작업 로그(oplog) 항목에 항상 해당하는 일반 재개 토큰과 달리, postBatchResumeToken은 변경 스트림이 서버에서 스캔한 oplog 항목에 해당하며, 반드시 일치하는 변경 사항은 아닙니다.

이전 일반 재개 토큰으로 다시 시작하려고 시도하는 경우 데이터베이스가 지정된 타임스탬프 및 현재 시간 간의 모든 oplog 항목을 강제로 스캔합니다. 이를 통해 짧은 시간 동안 각 하위 쿼리 스캔을 통해

내부적으로 많은 쿼리가 생성될 수 있습니다. 이로 인해 CPU 사용량이 급증하고 데이터베이스 성능이 저하됩니다. 마지막 `postBatchResumeToken`으로 재개하면 일치하지 않는 oplog 항목의 스캔을 건너뜁니다.

변경 스트림 내 트랜잭션

변경 스트림 이벤트에는 커밋되지 않았거나 중단된 트랜잭션의 이벤트는 포함되지 않습니다. 예를 들어 하나의 `INSERT` 작업과 하나의 `UPDATE` 작업으로 트랜잭션을 시작하고 `INSERT` 작업이 성공하지만 `UPDATE` 작업이 실패하면 트랜잭션이 롤백됩니다. 이 트랜잭션이 롤백되었으므로 변경 스트림에는 이 트랜잭션에 대한 이벤트가 포함되지 않습니다.

변경 스트림 로그 보존 기간 수정

AWS Management Console 또는를 사용하여 변경 스트림 로그 보존 기간을 1시간에서 7일로 수정할 수 있습니다 AWS CLI.

Using the AWS Management Console

변경 스트림 로그 보존 기간을 수정하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택합니다.

3. 파라미터 그룹 창에서 클러스터와 연결된 클러스터 파라미터 그룹을 선택합니다. 클러스터와 연결된 클러스터 파라미터 그룹을 식별하려면 Amazon DocumentDB 클러스터의 파라미터 그룹 확인 단원을 참조하십시오.
4. 결과 페이지에는 이 클러스터 파라미터 그룹에 대한 파라미터 및 해당 세부 정보가 표시됩니다. `change_stream_log_retention_duration` 파라미터를 선택합니다.
5. 페이지 오른쪽 상단에서 편집을 선택하여 파라미터의 값을 변경합니다. `change_stream_log_retention_duration` 파라미터는 1시간~7일 사이로 수정할 수 있습니다.

6. 변경한 다음 클러스터 파라미터 수정을 선택하여 변경 사항을 저장합니다. 변경 사항을 취소하려면 취소를 선택합니다.

Using the AWS CLI

클러스터 파라미터 그룹의 change_stream_log_retention_duration 파라미터를 수정하려면 다음 파라미터와 함께 modify-db-cluster-parameter-group 작업을 사용합니다.

- **--db-cluster-parameter-group-name** - 필수입니다. 수정하려는 클러스터 파라미터 그룹의 이름입니다. 클러스터와 연결된 클러스터 파라미터 그룹을 식별하려면 Amazon DocumentDB 클러스터의 파라미터 그룹 확인 단원을 참조하십시오.
- **--parameters** - 필수입니다. 수정 중인 파라미터입니다. 각 파라미터 요소는 다음을 포함해야 합니다.
  - **ParameterName** — 수정 중인 파라미터의 이름입니다. 이 경우에는 change_stream_log_retention_duration입니다.
  - **ParameterValue** — 이 파라미터의 새 값입니다.
  - **ApplyMethod** - 파라미터에 대한 변경 사항을 적용할 방법입니다. 허용된 값은 immediate 및 pending-reboot입니다.

> **Note**
>
> static의 ApplyType 파라미터에는 pending-reboot의 ApplyMethod이 있어야 합니다.

1. 파라미터 change_stream_log_retention_duration의 값을 변경하려면 다음 명령을 실행하고 parameter-value를 파라미터를 수정할 값으로 바꿉니다.

   Linux, macOS, Unix의 경우:

```
aws docdb modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name sample-parameter-group \
    --parameters
 "ParameterName=change_stream_log_retention_duration,ParameterValue=<parameter-
value>,ApplyMethod=immediate"
```

   Windows의 경우:

```
aws docdb modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name sample-parameter-group ^
    --parameters
 "ParameterName=change_stream_log_retention_duration,ParameterValue=<parameter-
value>,ApplyMethod=immediate"
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "DBClusterParameterGroupName": "sample-parameter-group"
}
```

2. 적어도 5분을 기다립니다.
3. `sample-parameter-group`의 파라미터 값을 나열하여 변경 사항을 확인합니다.

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

Windows의 경우:

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name sample-parameter-group
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "Parameters": [
        {
            "ParameterName": "audit_logs",
            "ParameterValue": "disabled",
            "Description": "Enables auditing on cluster.",
            "Source": "system",
            "ApplyType": "dynamic",
            "DataType": "string",
            "AllowedValues": "enabled,disabled",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot"
        },
```

```
        {
            "ParameterName": "change_stream_log_retention_duration",
            "ParameterValue": "12345",
            "Description": "Duration of time in seconds that the change stream
 log is retained and can be consumed.",
            "Source": "user",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "3600-86400",
            "IsModifiable": true,
            "ApplyMethod": "immediate"
        }
    ]
}
```

> **Note**
>
> 변경 스트림 로그 보존은 로그 크기가 (>) 51,200MB보다 커질 때까지 구성된 `change_stream_log_retention_duration` 값보다 오래된 로그를 삭제하지 않습니다.

보조 인스턴스에서 변경 스트림 사용

보조 인스턴스에서 변경 스트림 사용을 시작하려면 `readPreference` 항목을 보조 인스턴스로 사용하여 변경 스트림 커서를 엽니다.

변경 스트림 커서를 열어 클러스터 또는 데이터베이스의 특정 컬렉션 또는 모든 컬렉션에 대한 변경 이벤트를 감시할 수 있습니다. Amazon DocumentDB 인스턴스에서 변경 스트림 커서를 열고 라이터 인스턴스와 리더 인스턴스 모두에서 변경 스트림 문서를 가져올 수 있습니다. 라이터 및 리더 인스턴스에서 열린 다양한 변경 스트림 커서에서 변경 스트림 토큰(예: `resumeToken` 또는 `startOperationTime`)을 공유할 수 있습니다.

예제

```
import os
import sys
from pymongo import MongoClient, ReadPreference

username = "DocumentDBusername"
password = <Your password>
```

```
clusterendpoint = "DocumentDBClusterEndpoint"

client = MongoClient(clusterendpoint, username=username, password=password, tls='true',
 tlsCAFile='global-bundle.pem')

db = client['bar']

# Make sure to use SECONDARY to redirect cursor reads from secondary instances
coll = db.get_collection('foo', read_preference=ReadPreference.SECONDARY)

# Create a stream object on RO. The token needs to generated from PRIMARY.
stream = coll.watch(resumeAfter=token)

for event in stream:
   print(event)
```

보조 인스턴스의 변경 스트림에 대한 지침 및 제한 사항

- 스트림 변경 이벤트는 기본 인스턴스에서 보조 인스턴스로 복제해야 합니다. Amazon CloudWatch를 사용하여 `DBInstanceReplicaLag` 지표를 모니터링할 수도 있습니다.
- 보조 인스턴스의 타임스탬프가 항상 기본 인스턴스와 동기화되지는 않을 수 있습니다. 이 경우 보조 인스턴스 타임스탬프가 지연될 수 있어 따라잡을 수 있습니다. 가장 좋은 방법은 `startAtOperationTime` 또는 `resumeToken`를 사용하여 보조 인스턴스에서 감시를 시작하는 것입니다.
- 문서 크기가 크고, `fullDocumentLookup` 작업을 수행 중이며, 기본 인스턴스에 동시 쓰기 워크로드가 많은 경우 기본 인스턴스에 비해 보조 인스턴스의 처리량이 낮은 상황을 경험할 수도 있습니다. 가장 좋은 방법은 보조에서 버퍼 캐시 적중률을 모니터링하고 버퍼 캐시 적중률이 높은지 확인하는 것입니다.

Amazon DocumentDB의 트랜잭션

Amazon DocumentDB(MongoDB 호환)는 Amazon DocumentDB 4.0 이상에서 트랜잭션을 지원합니다. 여러 문서, 문, 컬렉션 및 데이터베이스에서 트랜잭션을 수행할 수 있습니다. 트랜잭션은 Amazon DocumentDB 클러스터 내의 하나 이상의 문서에서 원자적이고 일관적이며 격리되고 내구성이 뛰어난(ACID) 작업을 수행할 수 있도록 하여 애플리케이션 개발을 단순화합니다. 트랜잭션의 일반적인 사용 사례로는 재무 처리, 주문 이행 및 관리, 멀티플레이어 게임 구축 등이 있습니다.

트랜잭션을 활성화하는 데 추가 비용이 들지 않습니다. 트랜잭션의 일부로 사용한 읽기 및 쓰기 IO에 대한 비용만 지불하면 됩니다.

주제

- 요구 사항
- 모범 사례
- 제한 사항
- 모니터링 및 진단
- 트랜잭션 격리 수준
- 사용 사례
- 지원되는 명령
- 지원되지 않는 기능
- 세션
- 트랜잭션 오류

요구 사항

트랜잭션을 사용하려면 다음과 같은 요구 사항을 충족해야 합니다.

- Amazon DocumentDB 4.0 이상을 사용해야 합니다.
- MongoDB 4.0 이상과 호환되는 드라이버를 사용해야 합니다.

모범 사례

다음은 Amazon DocumentDB를 통한 트랜잭션을 최대한 활용할 수 있는 몇 가지 모범 사례입니다.

- 트랜잭션이 완료된 후에는 항상 트랜잭션을 커밋하거나 중단하십시오. 트랜잭션을 불완전한 상태로 두면 데이터베이스 리소스가 묶이고 쓰기 충돌이 발생할 수 있습니다.
- 필요한 명령 수를 최소한으로 줄여 트랜잭션을 유지하는 것이 좋습니다. 여러 개의 작은 트랜잭션으로 나눌 수 있는 여러 개의 명령문이 포함된 트랜잭션이 있는 경우, 시간 초과가 발생할 가능성을 줄이기 위해 여러 개의 작은 트랜잭션으로 나누는 것이 좋습니다. 항상 오래 실행되는 읽기가 아닌 짧은 트랜잭션을 생성하는 것을 목표로 하십시오.

제한 사항

- Amazon DocumentDB는 트랜잭션 내에서 커서를 지원하지 않습니다.
- Amazon DocumentDB는 트랜잭션에서 새 컬렉션을 생성할 수 없으며 존재하지 않는 컬렉션에 대해 쿼리/업데이트할 수 없습니다.
- 문서 수준 쓰기 잠금에는 사용자가 구성할 수 없는 1분의 제한 시간이 적용됩니다.
- 8.0.2 이전의 엔진 버전에서는 재시도 가능한 쓰기가 지원되지 않습니다. 엔진 버전 8.0.2부터는 각 개별 쓰기가 최대 하나의 문서에 영향을 미치는 쓰기 작업에 대해 트랜잭션 외부에서 재시도 가능한 쓰기가 지원되며 트랜잭션 커밋 및 중단 작업은 재시도 가능합니다. 자세한 내용은 Amazon DocumentDB에서 재시도 가능한 쓰기 섹션을 참조하세요.
- 각 Amazon DocumentDB 인스턴스에는 인스턴스에서 열 수 있는 동시 트랜잭션 수에 대한 상한이 있습니다. 자세한 내용은 인스턴스 할당량 섹션을 참조하세요.
- 특정 트랜잭션의 트랜잭션 로그 크기는 32MB 미만이어야 합니다.
- Amazon DocumentDB는 트랜잭션 내에서 `count()`을 지원하지만 모든 드라이버가 이 기능을 지원하는 것은 아닙니다. 대안은 개수 쿼리를 클라이언트 측의 집계 쿼리로 변환하는 `countDocuments()` API를 사용하는 것입니다.
- 트랜잭션의 실행 제한은 1분이고 세션의 제한 시간은 30분입니다. 트랜잭션 제한 시간이 초과되면 트랜잭션은 중단되며 기존 트랜잭션에 대해 세션 내에서 후속 명령을 실행하면 다음 오류가 발생합니다.

```
WriteCommandError({
"ok" : 0,
"operationTime" : Timestamp(1603491424, 627726),
"code" : 251,
"errmsg" : "Given transaction number 0 does not match any in-progress transactions."
}
```

모니터링 및 진단

Amazon DocumentDB 4.0의 트랜잭션 지원과 함께, 트랜잭션을 모니터링하는 데 도움이 되는 추가 CloudWatch 지표가 추가되었습니다.

새로운 CloudWatch 지표

- `DatabaseTransactions`: 1분 동안 수행된 열린 트랜잭션 수입니다.
- `DatabaseTransactionsAborted`: 1분 동안 수행된 중단된 트랜잭션 수입니다.

- `DatabaseTransactionsMax`: 1분 동안 수행된 열린 최대 트랜잭션 수입니다.
- `TransactionsAborted`: 1분 동안 인스턴스에 수행된 중단된 트랜잭션 수입니다.
- `TransactionsCommitted`: 1분 동안 인스턴스에 수행된 커밋된 트랜잭션 수입니다.
- `TransactionsOpen`: 1분 동안 인스턴스에 수행된 열린 트랜잭션 수입니다.
- `TransactionsOpenMax`: 1분 동안 인스턴스에 수행된 열린 최대 트랜잭션 수입니다.
- `TransactionsStarted`: 1분 동안 인스턴스에 수행된 시작된 트랜잭션 수입니다.

> **Note**
>
> Amazon DocumentDB에 대한 더 많은 CloudWatch 지표를 보려면 CloudWatch를 사용하여 Amazon DocumentDB 모니터링로 이동하십시오.

또한 두 가지 `currentOp` `lsid`, `transactionThreadId` 모두에 새 필드가 추가되었고 , “`idle transaction`” 및 `serverStatus` 트랜잭션에 대한 새 상태: `currentActive`, `currentInactive`, `currentOpen`, `totalAborted`, `totalCommitted` 및 `totalStarted`가 추가되었습니다.

트랜잭션 격리 수준

트랜잭션을 시작할 때 다음 예제`writeConcern`와 같이 `readConcern` 및를 모두 지정할 수 있습니다.

```
mySession.startTransaction({readConcern: {level: 'snapshot'}, writeConcern: {w: 'majority'}});
```

`readConcern`의 경우, Amazon DocumentDB는 기본적으로 스냅샷 격리를 지원합니다. 로컬, 사용 가능 또는 과반수의 `readConcern`이 지정된 경우 Amazon DocumentDB는 `readConcern` 레벨을 스냅샷으로 업그레이드합니다. Amazon DocumentDB는 선형화 가능한 `readConcern`을 지원하지 않으므로 이러한 읽기 문제를 지정하면 오류가 발생합니다.

`writeConcern`의 경우, Amazon DocumentDB는 기본적으로 과반수를 지원하며 데이터 사본 4개가 AZ에 걸쳐 유지되면 쓰기 쿼럼이 달성됩니다. 더 낮은 `writeConcern`을 지정하면 Amazon DocumentDB가 `writeConcern`를 과반수로 업그레이드합니다. 또한 모든 Amazon DocumentDB 쓰기는 저널링되며 저널링을 비활성화할 수 없습니다.

사용 사례

이 섹션에서는 다중 문과 다중 컬렉션이라는 두 가지 트랜잭션 사용 사례를 살펴보겠습니다.

다중 문 트랜잭션

Amazon DocumentDB 트랜잭션은 다중 문이므로 명시적인 커밋 또는 롤백을 통해 여러 문에 걸친 트랜잭션을 작성할 수 있습니다. `insert`, `update`, `delete` 및 `findAndModify` 작업을 단일 원자 연산으로 그룹화할 수 있습니다.

다중 문 트랜잭션의 일반적인 사용 사례는 직불 크레딧 트랜잭션입니다. 예를 들어, 친구에게 옷을 산 돈을 빚지고 있습니다. 따라서 귀하의 계좌에서 500달러를 인출(출금)하고 친구의 계좌에 500달러를 입금(예치금)해야 합니다. 이 작업을 수행하려면 단일 트랜잭션 내에서 부채 및 입금 작업을 모두 수행하여 원자성을 확보해야 합니다. 이렇게 하면 사용자 계정에서 500달러가 차감되지만 친구의 계좌에는 입금되지 않는 상황을 방지할 수 있습니다. 이 사용 사례는 다음과 같습니다.

```
// *** Transfer $500 from Alice to Bob inside a transaction: Success Scenario***
// Setup bank account for Alice and Bob. Each have $1000 in their account

var databaseName = "bank";
var collectionName = "account";
var amountToTransfer = 500;

var session = db.getMongo().startSession({causalConsistency: false});
var bankDB = session.getDatabase(databaseName);
var accountColl = bankDB[collectionName];
accountColl.drop();

accountColl.insert({name: "Alice", balance: 1000});
accountColl.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountColl.find({"name": "Alice"}).next().balance;
var newAliceBalance = aliceBalance - amountToTransfer;
accountColl.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountColl.find({"name": "Alice"}).next().balance;

// add $500 to Bob's account
var bobBalance = accountColl.find({"name": "Bob"}).next().balance;
var newBobBalance = bobBalance + amountToTransfer;
accountColl.update({"name": "Bob"},{"$set": {"balance": newBobBalance}});
var findBobBalance = accountColl.find({"name": "Bob"}).next().balance;
```

```
session.commitTransaction();

accountColl.find();

// *** Transfer $500 from Alice to Bob inside a transaction: Failure Scenario***

// Setup bank account for Alice and Bob. Each have $1000 in their account
var databaseName = "bank";
var collectionName = "account";
var amountToTransfer = 500;

var session = db.getMongo().startSession({causalConsistency: false});
var bankDB = session.getDatabase(databaseName);
var accountColl = bankDB[collectionName];
accountColl.drop();

accountColl.insert({name: "Alice", balance: 1000});
accountColl.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountColl.find({"name": "Alice"}).next().balance;
var newAliceBalance = aliceBalance - amountToTransfer;
accountColl.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountColl.find({"name": "Alice"}).next().balance;

session.abortTransaction();
```

다중 컬렉션 트랜잭션

또한 당사의 트랜잭션은 다중 컬렉션이므로 단일 트랜잭션 내에서 여러 컬렉션에 걸쳐 여러 작업을 수행하는 데 사용될 수 있습니다. 이를 통해 일관된 데이터 보기를 제공하고 데이터 무결성을 유지할 수 있습니다. 명령을 한 번의 <>로 커밋하면 트랜잭션은 모두 실행되거나 아예 실행되지 않습니다. 즉, 트랜잭션은 모두 성공하거나 모두 실패합니다.

다음은 다중 문 트랜잭션에 예제의 동일한 시나리오와 데이터를 사용하는 다중 컬렉션 트랜잭션의 예입니다.

```
// *** Transfer $500 from Alice to Bob inside a transaction: Success Scenario***
```

```
// Setup bank account for Alice and Bob. Each have $1000 in their account
var amountToTransfer = 500;
var collectionName = "account";

var session = db.getMongo().startSession({causalConsistency: false});
var accountCollInBankA = session.getDatabase("bankA")[collectionName];
var accountCollInBankB = session.getDatabase("bankB")[collectionName];

accountCollInBankA.drop();
accountCollInBankB.drop();

accountCollInBankA.insert({name: "Alice", balance: 1000});
accountCollInBankB.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountCollInBankA.find({"name": "Alice"}).next().balance;
var newAliceBalance = aliceBalance - amountToTransfer;
accountCollInBankA.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountCollInBankA.find({"name": "Alice"}).next().balance;

// add $500 to Bob's account
var bobBalance = accountCollInBankB.find({"name": "Bob"}).next().balance;
var newBobBalance = bobBalance + amountToTransfer;
accountCollInBankB.update({"name": "Bob"},{"$set": {"balance": newBobBalance}});
var findBobBalance = accountCollInBankB.find({"name": "Bob"}).next().balance;

session.commitTransaction();

accountCollInBankA.find(); // Alice holds $500 in bankA
accountCollInBankB.find(); // Bob holds $1500 in bankB

// *** Transfer $500 from Alice to Bob inside a transaction: Failure Scenario***

// Setup bank account for Alice and Bob. Each have $1000 in their account
var collectionName = "account";
var amountToTransfer = 500;

var session = db.getMongo().startSession({causalConsistency: false});
var accountCollInBankA = session.getDatabase("bankA")[collectionName];
var accountCollInBankB = session.getDatabase("bankB")[collectionName];

accountCollInBankA.drop();
```

```
accountCollInBankB.drop();

accountCollInBankA.insert({name: "Alice", balance: 1000});
accountCollInBankB.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountCollInBankA.find({"name": "Alice"}).next().balance;
var newAliceBalance = aliceBalance - amountToTransfer;
accountCollInBankA.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountCollInBankA.find({"name": "Alice"}).next().balance;

// add $500 to Bob's account
var bobBalance = accountCollInBankB.find({"name": "Bob"}).next().balance;
var newBobBalance = bobBalance + amountToTransfer;
accountCollInBankB.update({"name": "Bob"},{"$set": {"balance": newBobBalance}});
var findBobBalance = accountCollInBankB.find({"name": "Bob"}).next().balance;

session.abortTransaction();

accountCollInBankA.find(); // Alice holds $1000 in bankA
accountCollInBankB.find(); // Bob holds $1000 in bankB
```

콜백 API의 트랜잭션 API 예제

콜백 API는 4.2+ 드라이버에만 사용할 수 있습니다.

Javascript

다음 코드는 Amazon DocumentDB 트랜잭션 API를 자바스크립트와 함께 활용하는 방법을 보여줍니다.

```
// *** Transfer $500 from Alice to Bob inside a transaction: Success ***
// Setup bank account for Alice and Bob. Each have $1000 in their account
var databaseName = "bank";
var collectionName = "account";
var amountToTransfer = 500;

var session = db.getMongo().startSession({causalConsistency: false});
var bankDB = session.getDatabase(databaseName);
var accountColl = bankDB[collectionName];
```

```
accountColl.drop();

accountColl.insert({name: "Alice", balance: 1000});
accountColl.insert({name: "Bob", balance: 1000});

session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountColl.find({"name": "Alice"}).next().balance;
assert(aliceBalance >= amountToTransfer);
var newAliceBalance = aliceBalance - amountToTransfer;
accountColl.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountColl.find({"name": "Alice"}).next().balance;
assert.eq(newAliceBalance, findAliceBalance);

// add $500 to Bob's account
var bobBalance = accountColl.find({"name": "Bob"}).next().balance;
var newBobBalance = bobBalance + amountToTransfer;
accountColl.update({"name": "Bob"},{"$set": {"balance": newBobBalance}});
var findBobBalance = accountColl.find({"name": "Bob"}).next().balance;
assert.eq(newBobBalance, findBobBalance);

session.commitTransaction();

accountColl.find();
```

Node.js

다음 코드는 Amazon DocumentDB 트랜잭션 API를 Node.js와 함께 활용하는 방법을 보여줍니다.

```
// Node.js callback API:

const bankDB = await mongoclient.db("bank");
var accountColl = await bankDB.createCollection("account");
var amountToTransfer = 500;

const session = mongoclient.startSession({causalConsistency: false});
await accountColl.drop();

await accountColl.insertOne({name: "Alice", balance: 1000}, { session });
await accountColl.insertOne({name: "Bob", balance: 1000}, { session });

const transactionOptions = {
    readConcern: { level: 'snapshot' },
```

```
    writeConcern: { w: 'majority' }
    };

// deduct $500 from Alice's account
var aliceBalance = await accountColl.findOne({name: "Alice"}, {session});
assert(aliceBalance.balance >= amountToTransfer);
var newAliceBalance = aliceBalance - amountToTransfer;
session.startTransaction(transactionOptions);
await accountColl.updateOne({name: "Alice"}, {$set: {balance: newAliceBalance}},
 {session });
await session.commitTransaction();
aliceBalance = await accountColl.findOne({name: "Alice"}, {session});
assert(newAliceBalance == aliceBalance.balance);

// add $500 to Bob's account
var bobBalance = await accountColl.findOne({name: "Bob"}, {session});
var newBobBalance = bobBalance.balance + amountToTransfer;
session.startTransaction(transactionOptions);
await accountColl.updateOne({name: "Bob"}, {$set: {balance: newBobBalance}},
 {session });
await session.commitTransaction();
bobBalance = await accountColl.findOne({name: "Bob"}, {session});
assert(newBobBalance == bobBalance.balance);
```

C#

다음 코드는 Amazon DocumentDB 트랜잭션 API를 C#와 함께 활용하는 방법을 보여줍니다.

```
// C# Callback API

var dbName = "bank";
var collName = "account";
var amountToTransfer = 500;

using (var session = client.StartSession(new ClientSessionOptions{CausalConsistency
 = false}))
{
    var bankDB = client.GetDatabase(dbName);
    var accountColl = bankDB.GetCollection<BsonDocument>(collName);
    bankDB.DropCollection(collName);
    accountColl.InsertOne(session, new BsonDocument { {"name", "Alice"}, {"balance",
 1000 } });
    accountColl.InsertOne(session, new BsonDocument { {"name", "Bob"}, {"balance",
 1000 } });
```

```
   // start transaction
   var transactionOptions = new TransactionOptions(
           readConcern: ReadConcern.Snapshot,
           writeConcern: WriteConcern.WMajority);
   var result = session.WithTransaction(
       (sess, cancellationtoken) =>
       {
           // deduct $500 from Alice's account
           var aliceBalance = accountColl.Find(sess,
Builders<BsonDocument>.Filter.Eq("name",
"Alice")).FirstOrDefault().GetValue("balance");
           Debug.Assert(aliceBalance >= amountToTransfer);
           var newAliceBalance = aliceBalance.AsInt32 - amountToTransfer;
           accountColl.UpdateOne(sess, Builders<BsonDocument>.Filter.Eq("name",
"Alice"),
                                 Builders<BsonDocument>.Update.Set("balance",
newAliceBalance));
           aliceBalance = accountColl.Find(sess,
Builders<BsonDocument>.Filter.Eq("name",
"Alice")).FirstOrDefault().GetValue("balance");
           Debug.Assert(aliceBalance == newAliceBalance);

           // add $500 from Bob's account
           var bobBalance = accountColl.Find(sess,
Builders<BsonDocument>.Filter.Eq("name",
"Bob")).FirstOrDefault().GetValue("balance");
           var newBobBalance = bobBalance.AsInt32 + amountToTransfer;
           accountColl.UpdateOne(sess, Builders<BsonDocument>.Filter.Eq("name",
"Bob"),
                                 Builders<BsonDocument>.Update.Set("balance",
newBobBalance));
           bobBalance = accountColl.Find(sess,
Builders<BsonDocument>.Filter.Eq("name",
"Bob")).FirstOrDefault().GetValue("balance");
           Debug.Assert(bobBalance == newBobBalance);

           return "Transaction committed";
       }, transactionOptions);
  // check values outside of transaction
   var aliceNewBalance = accountColl.Find(Builders<BsonDocument>.Filter.Eq("name",
"Alice")).FirstOrDefault().GetValue("balance");
   var bobNewBalance = accountColl.Find(Builders<BsonDocument>.Filter.Eq("name",
"Bob")).FirstOrDefault().GetValue("balance");
```

```
    Debug.Assert(aliceNewBalance ==  500);
    Debug.Assert(bobNewBalance ==  1500);
}
```

Ruby

다음 코드는 Amazon DocumentDB 트랜잭션 API를 Ruby와 함께 활용하는 방법을 보여줍니다.

```
// Ruby Callback API

dbName = "bank"
collName = "account"
amountToTransfer = 500

session = client.start_session(:causal_consistency=> false)
bankDB = Mongo::Database.new(client, dbName)
accountColl = bankDB[collName]
accountColl.drop()

accountColl.insert_one({"name"=>"Alice", "balance"=>1000})
accountColl.insert_one({"name"=>"Bob", "balance"=>1000})

    # start transaction
    session.with_transaction(read_concern: {level: :snapshot}, write_concern:
 {w: :majority}) do
        # deduct $500 from Alice's account
        aliceBalance = accountColl.find({"name"=>"Alice"}, :session=>
 session).first['balance']
        assert aliceBalance >= amountToTransfer
        newAliceBalance = aliceBalance - amountToTransfer
        accountColl.update_one({"name"=>"Alice"}, { "$set" =>
 {"balance"=>newAliceBalance} }, :session=> session)
        aliceBalance = accountColl.find({"name"=>>"Alice"}, :session=>
 session).first['balance']
        assert_equal(newAliceBalance, aliceBalance)

        # add $500 from Bob's account
        bobBalance = accountColl.find({"name"=>"Bob"}, :session=>
 session).first['balance']
        newBobBalance = bobBalance + amountToTransfer
        accountColl.update_one({"name"=>"Bob"}, { "$set" =>
 {"balance"=>newBobBalance} }, :session=> session)
        bobBalance = accountColl.find({"name"=>"Bob"}, :session=>
 session).first['balance']
```

```
        assert_equal(newBobBalance, bobBalance)
    end

   # check results outside of transaction
    aliceBalance = accountColl.find({"name"=>"Alice"}).first['balance']
    bobBalance = accountColl.find({"name"=>"Bob"}).first['balance']
    assert_equal(aliceBalance, 500)
    assert_equal(bobBalance, 1500)

session.end_session
```

Go

다음 코드는 Amazon DocumentDB 트랜잭션 API를 Go와 함께 활용하는 방법을 보여줍니다.

```
// Go - Callback API
type Account struct {
    Name string
    Balance  int
}

ctx := context.TODO()

dbName := "bank"
collName := "account"
amountToTransfer := 500

session, err := client.StartSession(options.Session().SetCausalConsistency(false))
assert.NilError(t, err)
defer session.EndSession(ctx)

bankDB := client.Database(dbName)
accountColl := bankDB.Collection(collName)
accountColl.Drop(ctx)

_, err = accountColl.InsertOne(ctx, bson.M{"name" : "Alice", "balance":1000})
_, err = accountColl.InsertOne(ctx, bson.M{"name" : "Bob", "balance":1000})

transactionOptions := options.Transaction().SetReadConcern(readconcern.Snapshot()).

 SetWriteConcern(writeconcern.New(writeconcern.WMajority()))
_, err = session.WithTransaction(ctx, func(sessionCtx mongo.SessionContext)
 (interface{}, error) {
    var result Account
```

```
    // deduct $500 from Alice's account
    err = accountColl.FindOne(sessionCtx, bson.M{"name": "Alice"}).Decode(&result)
    aliceBalance := result.Balance
    newAliceBalance := aliceBalance - amountToTransfer
    _, err = accountColl.UpdateOne(sessionCtx, bson.M{"name": "Alice"},
 bson.M{"$set": bson.M{"balance": newAliceBalance}})
    err = accountColl.FindOne(sessionCtx, bson.M{"name": "Alice"}).Decode(&result)
    aliceBalance = result.Balance
    assert.Equal(t, aliceBalance, newAliceBalance)

    // add $500 to Bob's account
    err = accountColl.FindOne(sessionCtx, bson.M{"name": "Bob"}).Decode(&result)
    bobBalance := result.Balance
    newBobBalance := bobBalance + amountToTransfer
    _, err = accountColl.UpdateOne(sessionCtx, bson.M{"name": "Bob"}, bson.M{"$set":
 bson.M{"balance": newBobBalance}})
    err = accountColl.FindOne(sessionCtx, bson.M{"name": "Bob"}).Decode(&result)
    bobBalance = result.Balance
    assert.Equal(t, bobBalance, newBobBalance)

    if err != nil {
        return nil, err
    }
    return "transaction committed", err
}, transactionOptions)

// check results outside of transaction
var result Account
err = accountColl.FindOne(ctx, bson.M{"name": "Alice"}).Decode(&result)
aliceNewBalance := result.Balance
err = accountColl.FindOne(ctx, bson.M{"name": "Bob"}).Decode(&result)
bobNewBalance := result.Balance
assert.Equal(t, aliceNewBalance, 500)
assert.Equal(t, bobNewBalance, 1500)
```

Java

다음 코드는 Amazon DocumentDB 트랜잭션 API를 Java와 함께 활용하는 방법을 보여줍니다.

```
// Java (sync) - Callback API
MongoDatabase bankDB = mongoClient.getDatabase("bank");
MongoCollection accountColl = bankDB.getCollection("account");
accountColl.drop();
int amountToTransfer = 500;
```

```
// add sample data
accountColl.insertOne(new Document("name", "Alice").append("balance", 1000));
accountColl.insertOne(new Document("name", "Bob").append("balance", 1000));

TransactionOptions txnOptions = TransactionOptions.builder()
        .readConcern(ReadConcern.SNAPSHOT)
        .writeConcern(WriteConcern.MAJORITY)
        .build();
ClientSessionOptions sessionOptions =
 ClientSessionOptions.builder().causallyConsistent(false).build();
try ( ClientSession clientSession = mongoClient.startSession(sessionOptions) ) {
    clientSession.withTransaction(new TransactionBody<Void>() {
        @Override
        public Void execute() {
            // deduct $500 from Alice's account
            List<Document> documentList = new ArrayList<>();
            accountColl.find(clientSession, new Document("name",
 "Alice")).into(documentList);
            int aliceBalance = (int) documentList.get(0).get("balance");
            int newAliceBalance = aliceBalance - amountToTransfer;

            accountColl.updateOne(clientSession, new Document("name", "Alice"), new
 Document("$set", new Document("balance", newAliceBalance)));

            // check Alice's new balance
            documentList = new ArrayList<>();
            accountColl.find(clientSession, new Document("name",
 "Alice")).into(documentList);
            int updatedBalance = (int) documentList.get(0).get("balance");
            Assert.assertEquals(updatedBalance, newAliceBalance);

            // add $500 to Bob's account
            documentList = new ArrayList<>();
            accountColl.find(clientSession, new Document("name",
 "Bob")).into(documentList);
            int bobBalance = (int) documentList.get(0).get("balance");
            int newBobBalance = bobBalance + amountToTransfer;

            accountColl.updateOne(clientSession, new Document("name", "Bob"), new
 Document("$set", new Document("balance", newBobBalance)));

            // check Bob's new balance
            documentList = new ArrayList<>();
```

```
            accountColl.find(clientSession, new Document("name",
 "Bob")).into(documentList);
            updatedBalance = (int) documentList.get(0).get("balance");
            Assert.assertEquals(updatedBalance, newBobBalance);

            return null;
        }
    }, txnOptions);
}
```

C

다음 코드는 Amazon DocumentDB 트랜잭션 API를 C와 함께 활용하는 방법을 보여줍니다.

```
// Sample Code for C with Callback

#include <bson.h>
#include <mongoc.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>

typedef struct {
    int64_t balance;
    bson_t *account;
    bson_t *opts;
    mongoc_collection_t *collection;
} ctx_t;

bool callback_session (mongoc_client_session_t *session, void *ctx, bson_t **reply,
 bson_error_t *error)
{
    bool r = true;
    ctx_t *data = (ctx_t *) ctx;
    bson_t local_reply;
    bson_t *selector = data->account;
    bson_t *update = BCON_NEW ("$set", "{", "balance", BCON_INT64 (data->balance),
 "}");

    mongoc_collection_update_one (data->collection, selector, update, data->opts,
 &local_reply, error);

    *reply = bson_copy (&local_reply);
    bson_destroy (&local_reply);
```

```
    bson_destroy (update);
    return r;
}

void test_callback_money_transfer(mongoc_client_t* client, mongoc_collection_t*
 collection, int amount_to_transfer){

    bson_t reply;
    bool r = true;
    const bson_t *doc;
    bson_iter_t iter;
    ctx_t alice_ctx;
    ctx_t bob_ctx;
    bson_error_t error;

    // find query
    bson_t *alice_query = bson_new ();
    BSON_APPEND_UTF8(alice_query, "name", "Alice");

    bson_t *bob_query = bson_new ();
    BSON_APPEND_UTF8(bob_query, "name", "Bob");

    // create session
    // set causal consistency to false
    mongoc_session_opt_t *session_opts = mongoc_session_opts_new ();
    mongoc_session_opts_set_causal_consistency (session_opts, false);
    // start the session
    mongoc_client_session_t *client_session = mongoc_client_start_session (client,
 session_opts, &error);

    // add session to options
    bson_t *opts = bson_new();
    mongoc_client_session_append (client_session, opts, &error);

    // deduct 500 from Alice
    // find account balance of Alice
    mongoc_cursor_t *cursor = mongoc_collection_find_with_opts (collection,
 alice_query, NULL, NULL);
    mongoc_cursor_next (cursor, &doc);
    bson_iter_init (&iter, doc);
    bson_iter_find (&iter, "balance");
    int64_t alice_balance = (bson_iter_value (&iter))->value.v_int64;
    assert(alice_balance >= amount_to_transfer);
    int64_t new_alice_balance = alice_balance - amount_to_transfer;
```

```
   // set variables which will be used by callback function
   alice_ctx.collection = collection;
   alice_ctx.opts = opts;
   alice_ctx.balance = new_alice_balance;
   alice_ctx.account = alice_query;

   // callback
   r = mongoc_client_session_with_transaction (client_session, &callback_session,
NULL, &alice_ctx, &reply, &error);
   assert(r);

   // find account balance of Alice after transaction
   cursor = mongoc_collection_find_with_opts (collection, alice_query, NULL, NULL);
   mongoc_cursor_next (cursor, &doc);
   bson_iter_init (&iter, doc);
   bson_iter_find (&iter, "balance");
   alice_balance = (bson_iter_value (&iter))->value.v_int64;
   assert(alice_balance == new_alice_balance);
   assert(alice_balance == 500);

       // add 500 to bob's balance
   // find account balance of Bob
   cursor = mongoc_collection_find_with_opts (collection, bob_query, NULL, NULL);
   mongoc_cursor_next (cursor, &doc);
   bson_iter_init (&iter, doc);
   bson_iter_find (&iter, "balance");
   int64_t bob_balance = (bson_iter_value (&iter))->value.v_int64;
   int64_t new_bob_balance = bob_balance + amount_to_transfer;

   bob_ctx.collection = collection;
   bob_ctx.opts = opts;
   bob_ctx.balance = new_bob_balance;
   bob_ctx.account = bob_query;

   // set read & write concern
   mongoc_read_concern_t *read_concern = mongoc_read_concern_new ();
   mongoc_write_concern_t *write_concern = mongoc_write_concern_new ();
   mongoc_transaction_opt_t *txn_opts = mongoc_transaction_opts_new ();

   mongoc_write_concern_set_w(write_concern, MONGOC_WRITE_CONCERN_W_MAJORITY);
   mongoc_read_concern_set_level(read_concern, MONGOC_READ_CONCERN_LEVEL_SNAPSHOT);
   mongoc_transaction_opts_set_write_concern (txn_opts, write_concern);
   mongoc_transaction_opts_set_read_concern (txn_opts, read_concern);
```

```
    // callback
    r = mongoc_client_session_with_transaction (client_session, &callback_session,
 txn_opts, &bob_ctx, &reply, &error);
    assert(r);

 // find account balance of Bob after transaction
    cursor = mongoc_collection_find_with_opts (collection, bob_query, NULL, NULL);
    mongoc_cursor_next (cursor, &doc);
    bson_iter_init (&iter, doc);
    bson_iter_find (&iter, "balance");
    bob_balance = (bson_iter_value (&iter))->value.v_int64;
    assert(bob_balance == new_bob_balance);
    assert(bob_balance == 1500);

    // cleanup
    bson_destroy(alice_query);
    bson_destroy(bob_query);
    mongoc_client_session_destroy(client_session);
    bson_destroy(opts);
    mongoc_transaction_opts_destroy(txn_opts);
    mongoc_read_concern_destroy(read_concern);
    mongoc_write_concern_destroy(write_concern);
    mongoc_cursor_destroy(cursor);
    bson_destroy(doc);
}
int main(int argc, char* argv[]) {
    mongoc_init ();
    mongoc_client_t* client = mongoc_client_new (<connection uri>);
    bson_error_t error;

    // connect to bank db
    mongoc_database_t *database = mongoc_client_get_database (client, "bank");
    // access account collection
    mongoc_collection_t* collection = mongoc_client_get_collection(client, "bank",
 "account");
    // set amount to transfer
    int64_t amount_to_transfer = 500;
    // delete the collection if already existing
    mongoc_collection_drop(collection, &error);

    // open Alice account
    bson_t *alice_account = bson_new ();
    BSON_APPEND_UTF8(alice_account, "name", "Alice");
```

```
    BSON_APPEND_INT64(alice_account, "balance", 1000);

    // open Bob account
    bson_t *bob_account = bson_new ();
    BSON_APPEND_UTF8(bob_account, "name", "Bob");
    BSON_APPEND_INT64(bob_account, "balance", 1000);

    bool r = true;

    r = mongoc_collection_insert_one(collection, alice_account, NULL, NULL, &error);
    if (!r) {printf("Error encountered:%s", error.message);}
    r = mongoc_collection_insert_one(collection, bob_account, NULL, NULL, &error);
    if (!r) {printf("Error encountered:%s", error.message);}

    test_callback_money_transfer(client, collection, amount_to_transfer);

}
```

Python

다음 코드는 Amazon DocumentDB 트랜잭션 API를 Python과 함께 활용하는 방법을 보여줍니다.

```
// Sample Python code with callback api

import pymongo

def callback(session, balance, query):
    collection.update_one(query, {'$set': {"balance": balance}}, session=session)

client = pymongo.MongoClient(<connection uri>)
rc_snapshot = pymongo.read_concern.ReadConcern('snapshot')
wc_majority = pymongo.write_concern.WriteConcern('majority')

# To start, drop and create an account collection and insert balances for both Alice
 and Bob
collection = client.get_database("bank").get_collection("account")
collection.drop()
collection.insert_one({"_id": 1, "name": "Alice", "balance": 1000})
collection.insert_one({"_id": 2, "name": "Bob", "balance": 1000})

amount_to_transfer = 500

# deduct 500 from Alice's account
alice_balance = collection.find_one({"name": "Alice"}).get("balance")
```

```
assert alice_balance >= amount_to_transfer
new_alice_balance = alice_balance - amount_to_transfer

with client.start_session({'causalConsistency':False}) as session:
    session.with_transaction(lambda s: callback(s, new_alice_balance, {"name":
 "Alice"}), read_concern=rc_snapshot, write_concern=wc_majority)

updated_alice_balance = collection.find_one({"name": "Alice"}).get("balance")
assert updated_alice_balance == new_alice_balance

# add 500 to Bob's account
bob_balance = collection.find_one({"name": "Bob"}).get("balance")
assert bob_balance >= amount_to_transfer
new_bob_balance = bob_balance + amount_to_transfer

with client.start_session({'causalConsistency':False}) as session:
    session.with_transaction(lambda s: callback(s, new_bob_balance, {"name":
 "Bob"}), read_concern=rc_snapshot, write_concern=wc_majority)

updated_bob_balance = collection.find_one({"name": "Bob"}).get("balance")
assert updated_bob_balance == new_bob_balance
```

코어 API의 트랜잭션 API 예제

Javascript

다음 코드는 Amazon DocumentDB 트랜잭션 API를 자바스크립트와 함께 활용하는 방법을 보여줍니다.

```
// *** Transfer $500 from Alice to Bob inside a transaction: Success ***
// Setup bank account for Alice and Bob. Each have $1000 in their account
var databaseName = "bank";
var collectionName = "account";
var amountToTransfer = 500;

var session = db.getMongo().startSession({causalConsistency: false});
var bankDB = session.getDatabase(databaseName);
var accountColl = bankDB[collectionName];
accountColl.drop();

accountColl.insert({name: "Alice", balance: 1000});
accountColl.insert({name: "Bob", balance: 1000});
```

```
session.startTransaction();

// deduct $500 from Alice's account
var aliceBalance = accountColl.find({"name": "Alice"}).next().balance;
assert(aliceBalance >= amountToTransfer);
var newAliceBalance = aliceBalance - amountToTransfer;
accountColl.update({"name": "Alice"},{"$set": {"balance": newAliceBalance}});
var findAliceBalance = accountColl.find({"name": "Alice"}).next().balance;
assert.eq(newAliceBalance, findAliceBalance);

// add $500 to Bob's account
var bobBalance = accountColl.find({"name": "Bob"}).next().balance;
var newBobBalance = bobBalance + amountToTransfer;
accountColl.update({"name": "Bob"},{"$set": {"balance": newBobBalance}});
var findBobBalance = accountColl.find({"name": "Bob"}).next().balance;
assert.eq(newBobBalance, findBobBalance);

session.commitTransaction();

accountColl.find();
```

C#

다음 코드는 Amazon DocumentDB 트랜잭션 API를 C#와 함께 활용하는 방법을 보여줍니다.

```
// C# Core API

public void TransferMoneyWithRetry(IMongoCollection<bSondocument> accountColl,
 IClientSessionHandle session)
{
    var amountToTransfer = 500;

    // start transaction
   var transactionOptions = new TransactionOptions(
                readConcern: ReadConcern.Snapshot,
                writeConcern: WriteConcern.WMajority);
    session.StartTransaction(transactionOptions);
   try
    {
        // deduct $500 from Alice's account
        var aliceBalance = accountColl.Find(session,
 Builders<bSondocument>.Filter.Eq("name",
 "Alice")).FirstOrDefault().GetValue("balance");
```

```
        Debug.Assert(aliceBalance >= amountToTransfer);
        var newAliceBalance = aliceBalance.AsInt32 - amountToTransfer;
        accountColl.UpdateOne(session, Builders<bSondocument>.Filter.Eq("name",
 "Alice"),
                                  Builders<bSondocument>.Update.Set("balance",
 newAliceBalance));
        aliceBalance = accountColl.Find(session,
 Builders<bSondocument>.Filter.Eq("name",
 "Alice")).FirstOrDefault().GetValue("balance");
        Debug.Assert(aliceBalance == newAliceBalance);

        // add $500 from Bob's account
        var bobBalance = accountColl.Find(session,
 Builders<bSondocument>.Filter.Eq("name",
 "Bob")).FirstOrDefault().GetValue("balance");
        var newBobBalance = bobBalance.AsInt32 + amountToTransfer;
        accountColl.UpdateOne(session, Builders<bSondocument>.Filter.Eq("name",
 "Bob"),
                                  Builders<bSondocument>.Update.Set("balance",
 newBobBalance));
        bobBalance = accountColl.Find(session,
 Builders<bSondocument>.Filter.Eq("name",
 "Bob")).FirstOrDefault().GetValue("balance");
        Debug.Assert(bobBalance == newBobBalance);

    }
    catch (Exception e)
    {
        session.AbortTransaction();
        throw;
    }

    session.CommitTransaction();
 }

}
public void DoTransactionWithRetry(MongoClient client)
{
    var dbName = "bank";
    var collName = "account";
    using (var session = client.StartSession(new
 ClientSessionOptions{CausalConsistency = false}))
    {
        try
```

```
        {
            var bankDB = client.GetDatabase(dbName);
            var accountColl = bankDB.GetCollection<bSondocument>(collName);
            bankDB.DropCollection(collName);
            accountColl.InsertOne(session, new BsonDocument { {"name", "Alice"},
{"balance", 1000 } });
            accountColl.InsertOne(session, new BsonDocument { {"name", "Bob"},
{"balance", 1000 } });

            while(true) {
                try
                {
                        TransferMoneyWithRetry(accountColl, session);
                        break;
                }
                catch (MongoException e)
                {
                    if(e.HasErrorLabel("TransientTransactionError"))
                    {
                        continue;
                    }
                    else
                    {
                        throw;
                    }
                }
            }

            // check values outside of transaction
            var aliceNewBalance =
accountColl.Find(Builders<bSondocument>.Filter.Eq("name",
"Alice")).FirstOrDefault().GetValue("balance");
            var bobNewBalance =
accountColl.Find(Builders<bSondocument>.Filter.Eq("name",
"Bob")).FirstOrDefault().GetValue("balance");
            Debug.Assert(aliceNewBalance ==  500);
            Debug.Assert(bobNewBalance ==  1500);
        }
        catch (Exception e)
        {
            Console.WriteLine("Error running transaction: " + e.Message);
        }
    }
```

```
}
```

Ruby

다음 코드는 Amazon DocumentDB 트랜잭션 API를 Ruby와 함께 활용하는 방법을 보여줍니다.

```
# Ruby Core API

def transfer_money_w_retry(session, accountColl)
    amountToTransfer = 500

    session.start_transaction(read_concern: {level: :snapshot}, write_concern:
 {w: :majority})
    # deduct $500 from Alice's account
    aliceBalance = accountColl.find({"name"=>"Alice"}, :session=>
 session).first['balance']
    assert aliceBalance >= amountToTransfer
    newAliceBalance = aliceBalance - amountToTransfer
    accountColl.update_one({"name"=>"Alice"}, { "$set" =>
 {"balance"=>newAliceBalance} }, :session=> session)
    aliceBalance = accountColl.find({"name"=>"Alice"}, :session=>
 session).first['balance']
    assert_equal(newAliceBalance, aliceBalance)

    # add $500 to Bob's account
    bobBalance = accountColl.find({"name"=>"Bob"}, :session=>
 session).first['balance']
    newBobBalance = bobBalance + amountToTransfer
    accountColl.update_one({"name"=>"Bob"}, { "$set" =>
 {"balance"=>newBobBalance} }, :session=> session)
    bobBalance = accountColl.find({"name"=>"Bob"}, :session=>
 session).first['balance']
    assert_equal(newBobBalance, bobBalance)

    session.commit_transaction

end

def do_txn_w_retry(client)
     dbName = "bank"
    collName = "account"

    session = client.start_session(:causal_consistency=> false)
    bankDB = Mongo::Database.new(client, dbName)
```

```
    accountColl = bankDB[collName]
    accountColl.drop()

    accountColl.insert_one({"name"=>"Alice", "balance"=>1000})
    accountColl.insert_one({"name"=>"Bob", "balance"=>1000})

    begin
        transferMoneyWithRetry(session, accountColl)
        puts "transaction committed"
    rescue Mongo::Error => e
        if e.label?('TransientTransactionError')
            retry
        else
            puts "transaction failed"
            raise
        end
    end

    # check results outside of transaction
    aliceBalance = accountColl.find({"name"=>"Alice"}).first['balance']
    bobBalance = accountColl.find({"name"=>"Bob"}).first['balance']
    assert_equal(aliceBalance, 500)
    assert_equal(bobBalance, 1500)

end
```

Go

다음 코드는 Amazon DocumentDB 트랜잭션 API를 Go와 함께 활용하는 방법을 보여줍니다.

```
// Go - Core API
type Account struct {
    Name string
    Balance  int
}

func transferMoneyWithRetry(sessionContext mongo.SessionContext, accountColl
 *mongo.Collection, t *testing.T) error {
    amountToTransfer := 500

    transactionOptions :=
 options.Transaction().SetReadConcern(readconcern.Snapshot()).

 SetWriteConcern(writeconcern.New(writeconcern.WMajority()))
```

```
    if err := sessionContext.StartTransaction(transactionOptions); err != nil {
        panic(err)
    }

    var result Account
    // deduct $500 from Alice's account
    err := accountColl.FindOne(sessionContext, bson.M{"name":
 "Alice"}).Decode(&result)
    aliceBalance := result.Balance
    newAliceBalance := aliceBalance - amountToTransfer
    _, err = accountColl.UpdateOne(sessionContext, bson.M{"name": "Alice"},
 bson.M{"$set": bson.M{"balance": newAliceBalance}})
    if err != nil {
        sessionContext.AbortTransaction(sessionContext)
    }
    err = accountColl.FindOne(sessionContext, bson.M{"name":
 "Alice"}).Decode(&result)
    aliceBalance = result.Balance
    assert.Equal(t, aliceBalance, newAliceBalance)

    // add $500 to Bob's account
    err = accountColl.FindOne(sessionContext, bson.M{"name": "Bob"}).Decode(&result)
    bobBalance := result.Balance
    newBobBalance := bobBalance + amountToTransfer
    _, err = accountColl.UpdateOne(sessionContext, bson.M{"name": "Bob"},
 bson.M{"$set": bson.M{"balance": newBobBalance}})
    if err != nil {
        sessionContext.AbortTransaction(sessionContext)
    }
    err = accountColl.FindOne(sessionContext, bson.M{"name": "Bob"}).Decode(&result)
    bobBalance = result.Balance
    assert.Equal(t, bobBalance, newBobBalance)

    err = sessionContext.CommitTransaction(sessionContext)
    return err
}

func doTransactionWithRetry(t *testing.T) {
    ctx := context.TODO()

    dbName := "bank"
    collName := "account"
    bankDB := client.Database(dbName)
    accountColl := bankDB.Collection(collName)
```

```
    client.UseSessionWithOptions(ctx, options.Session().SetCausalConsistency(false),
 func(sessionContext mongo.SessionContext) error {
        accountColl.Drop(ctx)
        accountColl.InsertOne(sessionContext, bson.M{"name" : "Alice",
 "balance":1000})
        accountColl.InsertOne(sessionContext, bson.M{"name" : "Bob",
 "balance":1000})
        for {
            err := transferMoneyWithRetry(sessionContext, accountColl, t)
            if err == nil {
                println("transaction committed")
                return nil
            }
            if mongoErr := err.(mongo.CommandError);
 mongoErr.HasErrorLabel("TransientTransactionError") {
                continue
            }
            println("transaction failed")
            return err
        }
    })

    // check results outside of transaction
    var result Account
    accountColl.FindOne(ctx, bson.M{"name": "Alice"}).Decode(&result)
    aliceBalance := result.Balance
    assert.Equal(t, aliceBalance, 500)
    accountColl.FindOne(ctx, bson.M{"name": "Bob"}).Decode(&result)
    bobBalance := result.Balance
    assert.Equal(t, bobBalance, 1500)
}
```

Java

다음 코드는 Amazon DocumentDB 트랜잭션 API를 Java와 함께 활용하는 방법을 보여줍니다.

```
// Java (sync) - Core API

public void transferMoneyWithRetry() {
   // connect to server
    MongoClientURI mongoURI = new MongoClientURI(uri);
    MongoClient mongoClient = new MongoClient(mongoURI);
```

```
    MongoDatabase bankDB = mongoClient.getDatabase("bank");
    MongoCollection accountColl = bankDB.getCollection("account");
    accountColl.drop();

   // insert some sample data
    accountColl.insertOne(new Document("name", "Alice").append("balance", 1000));
    accountColl.insertOne(new Document("name", "Bob").append("balance", 1000));

    while (true) {
        try {
            doTransferMoneyWithRetry(accountColl, mongoClient);
            break;
        } catch (MongoException e) {
            if (e.hasErrorLabel(MongoException.TRANSIENT_TRANSACTION_ERROR_LABEL)) {
                continue;
            } else {
                throw e;
            }
        }
    }
}

public void doTransferMoneyWithRetry(MongoCollection accountColl, MongoClient
 mongoClient) {
    int amountToTransfer = 500;

   TransactionOptions txnOptions = TransactionOptions.builder()
      .readConcern(ReadConcern.SNAPSHOT)
      .writeConcern(WriteConcern.MAJORITY)
      .build();
    ClientSessionOptions sessionOptions =
 ClientSessionOptions.builder().causallyConsistent(false).build();
    try ( ClientSession clientSession = mongoClient.startSession(sessionOptions) ) {
        clientSession.startTransaction(txnOptions);

        // deduct $500 from Alice's account
        List<Document> documentList = new ArrayList<>();
        accountColl.find(clientSession, new Document("name",
 "Alice")).into(documentList);
        int aliceBalance = (int) documentList.get(0).get("balance");
        Assert.assertTrue(aliceBalance >= amountToTransfer);
        int newAliceBalance = aliceBalance - amountToTransfer;
        accountColl.updateOne(clientSession, new Document("name", "Alice"), new
 Document("$set", new Document("balance", newAliceBalance)));
```

```
        // check Alice's new balance
        documentList = new ArrayList<>();
        accountColl.find(clientSession, new Document("name",
 "Alice")).into(documentList);
        int updatedBalance = (int) documentList.get(0).get("balance");
        Assert.assertEquals(updatedBalance, newAliceBalance);

        // add $500 to Bob's account
        documentList = new ArrayList<>();
        accountColl.find(clientSession, new Document("name",
 "Bob")).into(documentList);
        int bobBalance = (int) documentList.get(0).get("balance");
        int newBobBalance = bobBalance + amountToTransfer;
        accountColl.updateOne(clientSession, new Document("name", "Bob"), new
 Document("$set", new Document("balance", newBobBalance)));

        // check Bob's new balance
        documentList = new ArrayList<>();
        accountColl.find(clientSession, new Document("name",
 "Bob")).into(documentList);
        updatedBalance = (int) documentList.get(0).get("balance");
        Assert.assertEquals(updatedBalance, newBobBalance);

        // commit transaction
        clientSession.commitTransaction();
    }
}
// Java (async) -- Core API
public void transferMoneyWithRetry() {
    // connect to the server
    MongoClient mongoClient = MongoClients.create(uri);

    MongoDatabase bankDB = mongoClient.getDatabase("bank");
    MongoCollection accountColl = bankDB.getCollection("account");
    SubscriberLatchWrapper<Void> dropCallback = new SubscriberLatchWrapper<>();
    mongoClient.getDatabase("bank").drop().subscribe(dropCallback);
    dropCallback.await();

    // insert some sample data
    SubscriberLatchWrapper<InsertOneResult> insertionCallback = new
 SubscriberLatchWrapper<>();
    accountColl.insertOne(new Document("name", "Alice").append("balance",
 1000)).subscribe(insertionCallback);
```

```
    insertionCallback.await();

    insertionCallback = new SubscriberLatchWrapper<>();
    accountColl.insertOne(new Document("name", "Bob").append("balance",
 1000)).subscribe(insertionCallback);;
    insertionCallback.await();

    while (true) {
        try {
            doTransferMoneyWithRetry(accountColl, mongoClient);
            break;
        } catch (MongoException e) {
            if (e.hasErrorLabel(MongoException.TRANSIENT_TRANSACTION_ERROR_LABEL)) {
                continue;
            } else {
                throw e;
            }
        }
    }
}

public void doTransferMoneyWithRetry(MongoCollection accountColl, MongoClient
 mongoClient) {
    int amountToTransfer = 500;

    // start the transaction
    TransactionOptions txnOptions = TransactionOptions.builder()
            .readConcern(ReadConcern.SNAPSHOT)
            .writeConcern(WriteConcern.MAJORITY)
            .build();
    ClientSessionOptions sessionOptions =
 ClientSessionOptions.builder().causallyConsistent(false).build();

    SubscriberLatchWrapper<ClientSession> sessionCallback = new
 SubscriberLatchWrapper<>();
    mongoClient.startSession(sessionOptions).subscribe(sessionCallback);
    ClientSession session = sessionCallback.get().get(0);
    session.startTransaction(txnOptions);

    // deduct $500 from Alice's account
    SubscriberLatchWrapper<Document> findCallback = new SubscriberLatchWrapper<>();
    accountColl.find(session, new Document("name",
 "Alice")).first().subscribe(findCallback);
    Document documentFound = findCallback.get().get(0);
```

```
   int aliceBalance = (int) documentFound.get("balance");
   int newAliceBalance = aliceBalance - amountToTransfer;

   SubscriberLatchWrapper<UpdateResult> updateCallback = new
SubscriberLatchWrapper<>();
   accountColl.updateOne(session, new Document("name",
"Alice"), new Document("$set", new Document("balance",
newAliceBalance))).subscribe(updateCallback);
   updateCallback.await();

   // check Alice's new balance
   findCallback = new SubscriberLatchWrapper<>();
   accountColl.find(session, new Document("name",
"Alice")).first().subscribe(findCallback);
   documentFound = findCallback.get().get(0);
   int updatedBalance = (int) documentFound.get("balance");
   Assert.assertEquals(updatedBalance, newAliceBalance);

   // add $500 to Bob's account
   findCallback = new SubscriberLatchWrapper<>();
   accountColl.find(session, new Document("name",
"Bob")).first().subscribe(findCallback);
   documentFound = findCallback.get().get(0);
   int bobBalance = (int) documentFound.get("balance");
   int newBobBalance = bobBalance + amountToTransfer;

   updateCallback = new SubscriberLatchWrapper<>();
   accountColl.updateOne(session, new Document("name", "Bob"), new Document("$set",
new Document("balance", newBobBalance))).subscribe(updateCallback);
   updateCallback.await();

   // check Bob's new balance
   findCallback = new SubscriberLatchWrapper<>();
   accountColl.find(session, new Document("name",
"Bob")).first().subscribe(findCallback);
   documentFound = findCallback.get().get(0);
   updatedBalance = (int) documentFound.get("balance");
   Assert.assertEquals(updatedBalance, newBobBalance);

   // commit the transaction
   SubscriberLatchWrapper<Void> transactionCallback = new
SubscriberLatchWrapper<>();
   session.commitTransaction().subscribe(transactionCallback);
   transactionCallback.await();
```

```
}

public class SubscriberLatchWrapper<T> implements Subscriber<T> {

    /**
     * A Subscriber that stores the publishers results and provides a latch so can
 block on completion.
     *
     * @param <T> The publishers result type
     */
    private final List<T> received;
    private final List<RuntimeException> errors;
    private final CountDownLatch latch;
    private volatile Subscription subscription;
    private volatile boolean completed;

    /**
     * Construct an instance
     */
    public SubscriberLatchWrapper() {
        this.received = new ArrayList<>();
        this.errors = new ArrayList<>();
        this.latch = new CountDownLatch(1);
    }

    @Override
    public void onSubscribe(final Subscription s) {
        subscription = s;
        subscription.request(Integer.MAX_VALUE);
    }

    @Override
    public void onNext(final T t) {
        received.add(t);
    }

    @Override
    public void onError(final Throwable t) {
        if (t instanceof RuntimeException) {
            errors.add((RuntimeException) t);
        } else {
            errors.add(new RuntimeException("Unexpected exception", t));
        }
        onComplete();
```

```
    }

    @Override
    public void onComplete() {
        completed = true;
        subscription.cancel();
        latch.countDown();
    }

    /**
     * Get received elements
     *
     * @return the list of received elements
     */
    public List<T> getReceived() {
        return received;
    }

    /**
     * Get received elements.
     *
     * @return the list of receive elements
     */
    public List<T> get() {
        return await().getReceived();
    }

    /**
     * Await completion or error
     *
     * @return this
     */
    public SubscriberLatchWrapper<T> await() {
        subscription.request(Integer.MAX_VALUE);
        try {
            if (!latch.await(300, TimeUnit.SECONDS)) {
                throw new MongoTimeoutException("Publisher onComplete timed out for
300 seconds");
            }
        } catch (InterruptedException e) {
            throw new MongoInterruptedException("Interrupted waiting for
observeration", e);
        }
        if (!errors.isEmpty()) {
```

```
            throw errors.get(0);
        }
        return this;
    }

    public boolean getCompleted() {
        return this.completed;
    }

    public void close() {
        subscription.cancel();
        received.clear();
    }
}
```

C

다음 코드는 Amazon DocumentDB 트랜잭션 API를 C와 함께 활용하는 방법을 보여줍니다.

```
// Sample C code with core session

bool core_session(mongoc_client_session_t *client_session, mongoc_collection_t*
 collection, bson_t *selector, int64_t balance){
    bool r = true;
    bson_error_t error;
    bson_t *opts = bson_new();
    bson_t *update = BCON_NEW ("$set", "{", "balance", BCON_INT64 (balance), "}");

    // set read & write concern
    mongoc_read_concern_t *read_concern = mongoc_read_concern_new ();
    mongoc_write_concern_t *write_concern = mongoc_write_concern_new ();
    mongoc_transaction_opt_t *txn_opts = mongoc_transaction_opts_new ();

    mongoc_write_concern_set_w(write_concern, MONGOC_WRITE_CONCERN_W_MAJORITY);
    mongoc_read_concern_set_level(read_concern, MONGOC_READ_CONCERN_LEVEL_SNAPSHOT);
    mongoc_transaction_opts_set_write_concern (txn_opts, write_concern);
    mongoc_transaction_opts_set_read_concern (txn_opts, read_concern);

    mongoc_client_session_start_transaction (client_session, txn_opts, &error);
    mongoc_client_session_append (client_session, opts, &error);

    r = mongoc_collection_update_one (collection, selector, update, opts, NULL,
 &error);
```

```
    mongoc_client_session_commit_transaction (client_session, NULL, &error);
    bson_destroy (opts);
    mongoc_transaction_opts_destroy(txn_opts);
    mongoc_read_concern_destroy(read_concern);
    mongoc_write_concern_destroy(write_concern);
    bson_destroy (update);
    return r;
}

void test_core_money_transfer(mongoc_client_t* client, mongoc_collection_t*
 collection, int amount_to_transfer){

    bson_t reply;
    bool r = true;
    const bson_t *doc;
    bson_iter_t iter;
    bson_error_t error;

    // find query
    bson_t *alice_query = bson_new ();
    BSON_APPEND_UTF8(alice_query, "name", "Alice");

    bson_t *bob_query = bson_new ();
    BSON_APPEND_UTF8(bob_query, "name", "Bob");

    // create session
    // set causal consistency to false
    mongoc_session_opt_t *session_opts = mongoc_session_opts_new ();
    mongoc_session_opts_set_causal_consistency (session_opts, false);
    // start the session
    mongoc_client_session_t *client_session = mongoc_client_start_session (client,
 session_opts, &error);

    // add session to options
    bson_t *opts = bson_new();
    mongoc_client_session_append (client_session, opts, &error);

    // deduct 500 from Alice
    // find account balance of Alice
    mongoc_cursor_t *cursor = mongoc_collection_find_with_opts (collection,
 alice_query, NULL, NULL);
    mongoc_cursor_next (cursor, &doc);
    bson_iter_init (&iter, doc);
```

```
    bson_iter_find (&iter, "balance");
    int64_t alice_balance = (bson_iter_value (&iter))->value.v_int64;
    assert(alice_balance >= amount_to_transfer);
    int64_t new_alice_balance = alice_balance - amount_to_transfer;

    // core
    r = core_session (client_session, collection, alice_query, new_alice_balance);
    assert(r);

    // find account balance of Alice after transaction
    cursor = mongoc_collection_find_with_opts (collection, alice_query, NULL, NULL);
    mongoc_cursor_next (cursor, &doc);
    bson_iter_init (&iter, doc);
    bson_iter_find (&iter, "balance");
    alice_balance = (bson_iter_value (&iter))->value.v_int64;
    assert(alice_balance == new_alice_balance);
    assert(alice_balance == 500);

    // add 500 to Bob's balance
    // find account balance of Bob
    cursor = mongoc_collection_find_with_opts (collection, bob_query, NULL, NULL);
    mongoc_cursor_next (cursor, &doc);
    bson_iter_init (&iter, doc);
    bson_iter_find (&iter, "balance");
    int64_t bob_balance = (bson_iter_value (&iter))->value.v_int64;
    int64_t new_bob_balance = bob_balance + amount_to_transfer;

    //core
    r = core_session (client_session, collection, bob_query, new_bob_balance);
    assert(r);

    // find account balance of Bob after transaction
    cursor = mongoc_collection_find_with_opts (collection, bob_query, NULL, NULL);
    mongoc_cursor_next (cursor, &doc);
    bson_iter_init (&iter, doc);
    bson_iter_find (&iter, "balance");
    bob_balance = (bson_iter_value (&iter))->value.v_int64;
    assert(bob_balance == new_bob_balance);
    assert(bob_balance == 1500);

    // cleanup
    bson_destroy(alice_query);
    bson_destroy(bob_query);
    mongoc_client_session_destroy(client_session);
```

```
    bson_destroy(opts);
    mongoc_cursor_destroy(cursor);
    bson_destroy(doc);
}

int main(int argc, char* argv[]) {
    mongoc_init ();
    mongoc_client_t* client = mongoc_client_new (<connection uri>);
    bson_error_t error;

    // connect to bank db
    mongoc_database_t *database = mongoc_client_get_database (client, "bank");
    // access account collection
    mongoc_collection_t* collection = mongoc_client_get_collection(client, "bank",
 "account");
    // set amount to transfer
    int64_t amount_to_transfer = 500;
    // delete the collection if already existing
    mongoc_collection_drop(collection, &error);

    // open Alice account
    bson_t *alice_account = bson_new ();
    BSON_APPEND_UTF8(alice_account, "name", "Alice");
    BSON_APPEND_INT64(alice_account, "balance", 1000);

    // open Bob account
    bson_t *bob_account = bson_new ();
    BSON_APPEND_UTF8(bob_account, "name", "Bob");
    BSON_APPEND_INT64(bob_account, "balance", 1000);

    bool r = true;

    r = mongoc_collection_insert_one(collection, alice_account, NULL, NULL, &error);
    if (!r) {printf("Error encountered:%s", error.message);}
    r = mongoc_collection_insert_one(collection, bob_account, NULL, NULL, &error);
    if (!r) {printf("Error encountered:%s", error.message);}

    test_core_money_transfer(client, collection, amount_to_transfer);

}
```

Scala

다음 코드는 Amazon DocumentDB 트랜잭션 API를 Scala와 함께 활용하는 방법을 보여줍니다.

```
// Scala Core API
def transferMoneyWithRetry(sessionObservable: SingleObservable[ClientSession] ,
 database: MongoDatabase ): Unit = {
    val accountColl = database.getCollection("account")
    var amountToTransfer = 500

    var transactionObservable: Observable[ClientSession] =
 sessionObservable.map(clientSession => {
    clientSession.startTransaction()

    // deduct $500 from Alice's account
    var aliceBalance = accountColl.find(clientSession, Document("name" ->
 "Alice")).await().head.getInteger("balance")
    assert(aliceBalance >= amountToTransfer)
    var newAliceBalance = aliceBalance - amountToTransfer
    accountColl.updateOne(clientSession, Document("name" -> "Alice"),
 Document("$set" -> Document("balance" -> newAliceBalance))).await()
    aliceBalance = accountColl.find(clientSession, Document("name" ->
 "Alice")).await().head.getInteger("balance")
    assert(aliceBalance == newAliceBalance)

    // add $500 to Bob's account
    var bobBalance = accountColl.find(clientSession, Document("name" ->
 "Bob")).await().head.getInteger("balance")
    var newBobBalance = bobBalance + amountToTransfer
    accountColl.updateOne(clientSession, Document("name" -> "Bob"), Document("$set"
 -> Document("balance" -> newBobBalance))).await()
    bobBalance = accountColl.find(clientSession, Document("name" ->
 "Bob")).await().head.getInteger("balance")
    assert(bobBalance == newBobBalance)

    clientSession
    })

    transactionObservable.flatMap(clientSession =>
 clientSession.commitTransaction()).await()
}

def doTransactionWithRetry(): Unit = {
    val client: MongoClient = MongoClientWrapper.getMongoClient()
    val database: MongoDatabase = client.getDatabase("bank")
    val accountColl = database.getCollection("account")
    accountColl.drop().await()
```

```
   val sessionOptions =
ClientSessionOptions.builder().causallyConsistent(false).build()
   var  sessionObservable: SingleObservable[ClientSession] =
client.startSession(sessionOptions)
   accountColl.insertOne(Document("name" -> "Alice", "balance" -> 1000)).await()
   accountColl.insertOne(Document("name" -> "Bob", "balance" -> 1000)).await()

   var retry = true
   while (retry) {
       try {
       transferMoneyWithRetry(sessionObservable, database)
       println("transaction committed")
       retry = false
       }
       catch {
       case e: MongoException if
e.hasErrorLabel(MongoException.TRANSIENT_TRANSACTION_ERROR_LABEL) => {
           println("retrying transaction")
       }
       case other: Throwable => {
           println("transaction failed")
           retry = false
           throw other

       }
       }
   }

   // check results outside of transaction
   assert(accountColl.find(Document("name" ->
"Alice")).results().head.getInteger("balance") == 500)
   assert(accountColl.find(Document("name" ->
"Bob")).results().head.getInteger("balance") == 1500)

   accountColl.drop().await()

}
```

Python

다음 코드는 Amazon DocumentDB 트랜잭션 API를 Python과 함께 활용하는 방법을 보여줍니다.

```
// Sample Python code with Core api
```

```
import pymongo

client = pymongo.MongoClient(<connection_string>)
rc_snapshot = pymongo.read_concern.ReadConcern('snapshot')
wc_majority = pymongo.write_concern.WriteConcern('majority')

# To start, drop and create an account collection and insert balances for both Alice
 and Bob
collection = client.get_database("bank").get_collection("account")
collection.drop()
collection.insert_one({"_id": 1, "name": "Alice", "balance": 1000})
collection.insert_one({"_id": 2, "name": "Bob", "balance": 1000})

amount_to_transfer = 500

# deduct 500 from Alice's account
alice_balance = collection.find_one({"name": "Alice"}).get("balance")
assert alice_balance >= amount_to_transfer
new_alice_balance = alice_balance - amount_to_transfer

with client.start_session({'causalConsistency':False}) as session:
    session.start_transaction(read_concern=rc_snapshot, write_concern=wc_majority)
    collection.update_one({"name": "Alice"}, {'$set': {"balance":
 new_alice_balance}}, session=session)
    session.commit_transaction()

updated_alice_balance = collection.find_one({"name": "Alice"}).get("balance")
assert updated_alice_balance == new_alice_balance

# add 500 to Bob's account
bob_balance = collection.find_one({"name": "Bob"}).get("balance")
assert bob_balance >= amount_to_transfer
new_bob_balance = bob_balance + amount_to_transfer

with client.start_session({'causalConsistency':False}) as session:
    session.start_transaction(read_concern=rc_snapshot, write_concern=wc_majority)
    collection.update_one({"name": "Bob"}, {'$set': {"balance": new_bob_balance}},
 session=session)
    session.commit_transaction()

updated_bob_balance = collection.find_one({"name": "Bob"}).get("balance")
assert updated_bob_balance == new_bob_balance
```

지원되는 명령

명령	지원됨
abortTransaction	예
commitTransaction	예
endSessions	예
killSession	예
killAllSession	예
killAllSessionsByPattern	아니요
refreshSessions	아니요
startSession	예

지원되지 않는 기능

메서드	단계 또는 명령
db.collection.aggregate()	$collStats $currentOp $indexStats $listSessions $out
db.collection.count() db.collection.countDocuments()	$where $near $nearSphere

메서드	단계 또는 명령
`db.collection.insert()`	기존 컬렉션에 대해 실행되지 않는 경우 `insert`은 지원되지 않습니다. 이 메서드는 기존 컬렉션을 대상으로 하는 경우 지원됩니다.

세션

MongoDB 세션은 재시도 가능한 쓰기, 인과 관계 일관성, 트랜잭션을 지원하고 데이터베이스 전반의 작업을 관리하는 데 사용되는 프레임워크입니다. 세션이 생성되면 클라이언트에서 lsid(논리적 세션 식별자)를 생성하고 서버로 명령을 보낼 때 해당 세션 내의 모든 작업에 태그를 지정하는 데 사용됩니다.

Amazon DocumentDB는 세션 사용을 지원하여 트랜잭션을 활성화하지만 인과관계 일관성은 지원하지 않습니다. 엔진 버전 8.0.2부터 Amazon DocumentDB는 각 개별 쓰기가 최대 하나의 문서에 영향을 미치는 쓰기 작업에 대해 트랜잭션 외부에서 재시도 가능한 쓰기를 지원합니다. 이전 엔진 버전에서는 재시도 가능한 쓰기가 지원되지 않습니다. 자세한 내용은 Amazon DocumentDB에서 재시도 가능한 쓰기 섹션을 참조하세요.

Amazon DocumentDB 내에서 트랜잭션을 사용하는 경우 `session.startTransaction()` API를 사용하여 세션 내에서 트랜잭션이 시작되며 세션은 한 번에 하나의 트랜잭션만 지원합니다. 마찬가지로, 트랜잭션은 커밋(`session.commitTransaction()`) 또는 중단(`session.abortTransaction()`) API를 사용하여 완료됩니다.

인과 관계 일관성

인과 관계 일관성을 통해 클라이언트는 단일 클라이언트 세션 내에서 읽기 후 쓰기 일관성, 단일 원자 읽기/쓰기 및 읽기 후 쓰기가 수행되며 이러한 보장은 기본 인스턴스뿐만 아니라 클러스터의 모든 인스턴스에 적용됩니다. Amazon DocumentDB는 인과 관계의 일관성을 지원하지 않으므로 다음 명령문을 실행하면 오류가 발생합니다.

```
var mySession = db.getMongo().startSession();
var mySessionObject = mySession.getDatabase('test').getCollection('account');

mySessionObject.updateOne({"_id": 2}, {"$inc": {"balance": 400}});
//Result:{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

mySessionObject.find()
//Error: error: {
```

```
//         "ok" : 0,
//         "code" : 303,
//         "errmsg" : "Feature not supported: 'causal consistency'",
//         "operationTime" : Timestamp(1603461817, 493214)
//}

mySession.endSession()
```

세션 내에서 인과 관계 일관성을 비활성화할 수 있습니다. 이렇게 하면 세션 프레임워크를 활용할 수 있지만 읽기에 대한 인과적 일관성은 보장되지 않습니다. Amazon DocumentDB를 사용할 경우 기본 인스턴스에서의 읽기는 쓰기 후 읽기 일관성이 유지되고 복제본 인스턴스에서의 읽기도 최종적으로 일관됩니다. 트랜잭션은 세션을 활용하는 주요 사용 사례입니다.

```
var mySession = db.getMongo().startSession({causalConsistency: false});
var mySessionObject = mySession.getDatabase('test').getCollection('account');

mySessionObject.updateOne({"_id": 2}, {"$inc": {"balance": 400}});
//Result:{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

mySessionObject.find()
//{ "_id" : 1, "name" : "Bob", "balance" : 100 }
//{ "_id" : 2, "name" : "Alice", "balance" : 1700 }
```

재시도 가능한 쓰기

재시도 가능한 쓰기를 사용하면 네트워크 오류가 발생하거나 드라이버가 기본 쓰기 작업을 찾을 수 없는 경우 드라이버가 자동으로 쓰기 작업을 한 번 재시도합니다. 엔진 버전 8.0.2부터 Amazon DocumentDB는 재시도 가능한 쓰기를 지원합니다. 이전 엔진 버전에서는 연결 문자열`retryWrites=false`에서를 설정하여 재시도 가능한 쓰기를 비활성화해야 합니다. 자세한 내용은 Amazon DocumentDB에서 재시도 가능한 쓰기 섹션을 참조하세요.

트랜잭션 오류

트랜잭션을 사용할 때, 트랜잭션 번호가 진행 중인 트랜잭션과 일치하지 않는다는 오류가 발생하는 상황이 있습니다.

이 오류는 최소한 두 가지 상황에서 발생할 수 있습니다.

- 1분 트랜잭션 제한 시간 후입니다.

- 인스턴스를 다시 시작한 후(패칭, 충돌 복구 등으로 인해) 트랜잭션이 성공적으로 커밋된 경우에도 이 오류를 수신할 수 있습니다. 인스턴스를 다시 시작하는 동안 데이터베이스는 성공적으로 완료된 트랜잭션과 중단된 트랜잭션 간의 차이를 알 수 없습니다. 즉, 트랜잭션 완료 상태가 모호합니다.

이 오류를 처리하는 가장 좋은 방법은 트랜잭션 업데이트를 멱등적으로 만드는 것입니다. 예를 들어 증가/감소 작업 대신 `$set` 뮤테이터를 사용하는 것입니다. 아래 내용을 참조하십시오.

```
{ "ok" : 0,
"operationTime" : Timestamp(1603938167, 1),
"code" : 251,
"errmsg" : "Given transaction number 1 does not match any in-progress transactions."
}
```

Amazon DocumentDB에서 재시도 가능한 쓰기

엔진 버전 8.0.2부터 Amazon DocumentDB는 재시도 가능한 쓰기를 지원합니다. 일시적인 네트워크 오류 또는 기본 선택으로 인해 쓰기 작업이 실패하면 드라이버는 작업을 정확히 한 번 재시도할 수 있습니다. Amazon DocumentDB는 재시도된 쓰기를 중복 제거하므로 작업이 최대 한 번 적용되어 멱등성을 유지합니다.

재시도 가능한 쓰기에는 재시도 가능한 쓰기 프로토콜을 지원하는 MongoDB 호환 드라이버가 필요합니다. 대부분의 현재 MongoDB 호환 드라이버는 기본적으로 재시도 가능한 쓰기를 활성화합니다 (`retryWrites=true`연결 문자열에서). 이전 버전에서 엔진 8.0.2로 업그레이드하는 경우 연결 문자열`retryWrites=false`에서를 제거하여이 기능을 활성화할 수 있습니다.

요구 사항

재시도 가능한 쓰기를 사용하려면 다음 요구 사항을 충족해야 합니다.

- Amazon DocumentDB 엔진 버전 8.0.2 이상.
- 재시도 가능한 쓰기를 지원하는 MongoDB 호환 드라이버입니다. 최소 버전은 드라이버 설명서를 참조하세요.
- 연결 문자열에가 포함`retryWrites=true`되거나 드라이버가 기본적으로 재시도 가능한 쓰기(대부분의 현재 드라이버가 수행)여야 합니다.

지원되는 연산자

재시도 가능한 쓰기는 각 개별 쓰기가 최대 하나의 문서에 영향을 미치는 쓰기 작업에 적용됩니다. 배치 내의 각 쓰기는 별도로 중복 제거되므로 배치 작업은 한정됩니다.

다음 쓰기 작업을 재시도할 수 있습니다.

- `insertOne`
- `insertMany`
- `updateOne`
- `deleteOne`
- `findOneAndUpdate`
- `findOneAndDelete`
- `findOneAndReplace`
- `bulkWrite` (`insertOne`, `updateOnedeleteOne`, 또는 `replaceOne` 작업으로 구성된 경우)

`insertMany` 및의 경우 배치의 `bulkWrite`각 문서는 개별적으로 중복 제거됩니다. 재시도가 필요한 경우 아직 적용되지 않은 문서만 삽입됩니다.

> **Note**
>
> `commitTransaction` 및 `abortTransaction`는 쓰기가 아닌 트랜잭션 제어 명령으로 재시도할 수도 있습니다. 트랜잭션 내의 개별 쓰기는 재시도할 수 없습니다. 자세한 내용은 제한 사항 섹션을 참조하세요.

제한 사항

Amazon DocumentDB에서 재시도 가능한 쓰기에는 다음 제한 사항이 적용됩니다.

- `updateMany` 및 `deleteMany`는 재시도할 수 없습니다.
- 다중 문 트랜잭션 내의 쓰기는 재시도할 수 없습니다. 트랜잭션 커밋 및 중단 작업은 별도로 재시도할 수 있습니다.
- 문서에는 재시도 가능한 삽입에 대한 `_id` 필드가 포함되어야 합니다.
- 재시도 가능한 쓰기는 엔진 버전 8.0.2 이상에서만 사용할 수 있습니다. 이전 엔진 버전에서는 오류를 방지하기 위해 연결 문자열`retryWrites=false`에를 설정합니다.

- 쿼리 플래너 버전 1.0을 사용하는 클러스터에서는 쓰기 작업을 재시도할 수 없습니다.
- 대용량 문서를 반환하는 `findOneAndUpdate`, `findOneAndDelete`, 및 `findOneAndReplace` 작업의 경우 전체 결과 문서가 중복 제거를 위해 캐시되므로 재시도 가능한 쓰기로 인해 쓰기 지연 시간이 증가할 수 있습니다. 쓰기 성능이 중요하고 반환된 문서가 큰 경우 해당 워크로드에 `retryWrites=false` 대한 설정을 고려하세요.

재시도 가능한 쓰기 활성화

재시도 가능한 쓰기를 활성화하려면 연결 문자열`retryWrites=true`에를 포함합니다.

```
mongodb://<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=true
```

드라이버의 기본값이 인 경우 연결 문자열`retryWrites=false`에서 명시적인를 제거할 `retryWrites=true`수 있습니다.

전체 연결 문자열 연습은 섹션을 참조하세요Amazon DocumentDB에 프로그래밍 방식으로 연결.

재시도 가능한 쓰기 작동 방식

애플리케이션이를 사용하여 쓰기`retryWrites=true`를 전송하면 드라이버는 논리적 세션 ID(`lsid`)와 트랜잭션 번호(`txnNumber`)를 쓰기 명령에 연결합니다. Amazon DocumentDB는 다음 식별자를 사용하여 쓰기를 중복 제거합니다.

- 첫 번째 시도에서 Amazon DocumentDB는 쓰기를 실행하고 결과를 캐싱합니다.
- 드라이버가 동일한 쓰기(`lsid` 및 동일`txnNumber`)를 재시도하면 Amazon DocumentDB는 쓰기를 다시 실행하지 않고 캐시된 결과를 반환합니다.
- Amazon DocumentDB는 원본 쓰기 후 60분 동안 재시도를 수행합니다. 드라이버는 즉시 재시도하므로이 기간은 드라이버에 필요한 시간보다 훨씬 더 깁니다.

원래 쓰기 후 60분 이상 후에 재시도가 도착하면 Amazon DocumentDB에 캐시된 결과가 더 이상 없으며 쓰기를 새 작업으로 실행합니다.

오류 처리

다음 오류는 Amazon DocumentDB에서 재시도 가능한 쓰기에만 해당됩니다.

오류 코드	이름	설명
225	TransactionTooOld	동일한 세션 및 문에 대한 최신 트랜잭션 번호가 이미 커밋되었으므로이 재시도가 너무 오래되어 적용할 수 없습니다.
301	재시도 가능한 쓰기는 지원되지 않음	쓰기에는 재시도 가능한 쓰기 필드가 포함되어 있지만 엔진 버전은 재시도 가능한 쓰기를 지원하지 않거나 작업을 재시도할 수 없습니다. 8.0.2 이전의 엔진 버전에 연결할 `retryWrites=false` 때를 설정합니다.

retryWrites=false에서 마이그레이션

엔진 버전 8.0.2로 업그레이드하고 현재 연결 문자열`retryWrites=false`에서를 사용하는 경우:

1. 클러스터를 엔진 버전 8.0.2 이상으로 업그레이드합니다.
2. 애플리케이션의 연결 문자열`retryWrites=false`에서를 제거하거나 로 변경합니다`retryWrites=true`.
3. 애플리케이션 코드를 변경할 필요가 없습니다. 드라이버는 재시도를 자동으로 처리합니다.

Amazon DocumentDB 8.0에서 데이터 정렬 사용

Amazon DocumentDB 8.0은 이제 데이터 정렬을 지원합니다. 데이터 정렬을 사용하면 문자열 비교를 위한 언어별 규칙을 구성할 수 있습니다. 데이터 정렬을 사용하면 대소문자를 구분하는 비교에 대한 규칙을 지정하거나 언어 로캘을 지정할 수 있습니다. 데이터 정렬은 DocumentDB 8.0의 컬렉션 수준 또는 인덱스 수준에서 구성할 수 있습니다. DocumentDB에서 데이터 정렬을 사용하면 내부적으로 다음 파라미터와 함께 데이터 정렬 문서가 생성됩니다.

```
{
locale: string,
caseLevel: boolean,
caseFirst: string,
strength: int,
numericOrdering: boolean,
alternate: string,
maxVariable: string,
```

```
backwards: boolean,
normalization: boolean
}
```

제한 사항

데이터 정렬에는 Amazon DocumentDB에서 다음과 같은 제한 사항이 있습니다.

- 데이터 정렬은 Amazon DocumentDB 8.0에서 사용할 수 있는 플래너 v3와 호환됩니다. plannerv2 또는 plannerv1로 전환하면 “인덱스를 찾을 수 없음” 오류를 비롯한 일관되지 않은 동작이 발생할 수 있습니다.
- 고유한 라이브러리 차이로 인해 데이터 정렬이 포함된 mongodb 컬렉션을 내보내는 경우 마이그레이션 전에 metadata.bson 파일을 업데이트하고 해당 버전을 57.1에서 60.2로 변경해야 합니다.
- 드문 경우지만 데이터 정렬 설정이 문자 수에 대한 내부 제한을 위반하여 다음 오류가 발생할 수 있습니다. “오류: 데이터 정렬 문서에 기본값이 아닌 속성이 지원되는 속성보다 많습니다. 옵션 수를 줄이세요.” 이 경우 데이터 정렬 문서에 제공하는 옵션을 줄이거나 가능하면 기본값을 사용해 볼 수 있습니다.

Amazon DocumentDB 8.0에서 뷰 사용

- 이제 Amazon DocumentDB 8.0에서 뷰를 지원합니다. 지정된 집계 작업을 기반으로 데이터를 제공하는 가상 컬렉션으로 함수를 봅니다. 뷰를 생성할 때 하나 이상의 소스 컬렉션에서 데이터를 변환하는 쿼리를 정의합니다. Amazon DocumentDB 8.0은 추가 스토리지 리소스를 소비하지 않고 뷰에 액세스할 때마다이 쿼리를 실행합니다. 표준 컬렉션과 달리 Amazon DocumentDB 8.0의 뷰는 문서를 디스크에 저장하지 않으므로 변환되거나 필터링된 데이터를 애플리케이션에 제공하는 효율적인 솔루션입니다. Amazon DocumentDB에서 뷰를 생성하려면 createView 명령 또는 db.createView() 헬퍼 메서드를 사용할 수 있습니다.

```
db.createView("viewName","sourceCollection",
[
    { $match: { status: "active" } },
    { $project: { _id: 1, name: 1, email: 1 } }
]
)
```

이렇게 하면 활성 문서만 포함하고 _id, 이름 및 이메일 필드만 프로젝트하는 "sourceCollection"을 기반으로 "viewName"이라는 뷰가 생성됩니다. Amazon DocumentDB의 보기는 읽기 전용입니다. 뷰에 대한 쓰기 작업은 오류를 반환합니다. 대규모 데이터 세트로 최적의 성능을 위해 뷰 파이프라인을 구조화하여 효율성을 극대화할 수 있습니다. 복잡한 집계 파이프라인의 경우 $match 단계를 파이프라인의 첫 번째 단계 또는 초기 단계로 사용하여 후속 단계에서 처리해야 하는 문서 수를 줄여 쿼리 성능을 개선하는 것이 좋습니다.

모범 사례

보기에서 따라야 할 몇 가지 모범 사례는 다음과 같습니다.

- 조기 필터링: 뷰 파이프라인 초기에 $match 단계를 사용하여 처리되는 데이터의 양을 줄입니다.
- 복잡한 집계 방지: 복잡한 집계로 자주 액세스하는 뷰의 경우 주기적으로 업데이트되는 사전 계산된 결과로 별도의 컬렉션을 생성하는 것이 좋습니다.
- 인덱스 계획: 뷰 파이프라인, 특히 $match 및 $sort 작업에 사용되는 필드가 소스 컬렉션에서 제대로 인덱싱되었는지 확인합니다.
- 쿼리 최적화: explain 명령을 사용하여 뷰 쿼리가 실행되는 방식을 이해하고 그에 따라 최적화합니다.
- 뷰의 대안: Amazon DocumentDB 뷰와 MongoDB 뷰의 기능적 차이를 고려할 때 제한이 발생할 때 뷰 대신 예약된 업데이트가 있는 정기 컬렉션을 사용하는 것이 좋습니다.

집계자 연산자 호환성

Amazon DocumentDB는 호환성을 계속 확장하면서 뷰 정의에서 많은 집계 연산자를 지원합니다. 뷰를 사용할 때는 지원되는 다음 연산자에 집중하세요.

- 문서 필터링을 위한 $match
- 필드 선택 및 변환을 위한 $project
- 계산된 필드를 추가하기 위한 $addFields
- 결과 주문에 대한 $sort
- 페이지 매김에 대한 $limit 및 $skip

$currentOp, $replaceRoot, $geoNear와 같은 일부 특수 연산자는 현재 뷰 정의가 아닌 직접 집계 쿼리에서 작동합니다.

인덱스 및 뷰 활용

Amazon DocumentDB 8.0의 뷰는 기본 컬렉션의 인덱스를 사용합니다. 따라서 뷰에서 인덱스를 직접 생성, 삭제 또는 재구축할 수 없습니다. 그러나 소스 컬렉션에서 잘 설계된 인덱스는 보기 쿼리 성능을 크게 개선할 수 있습니다. 다음은 보기에서 쿼리 성능을 최적화하기 위한 몇 가지 단계입니다.

- 뷰의 파이프라인, 특히 $match 및 $sort 작업에 사용되는 소스 컬렉션 필드에 적절한 인덱스가 있는지 확인합니다.
- explain() 메서드를 사용하여 쿼리 실행 계획을 분석하고 인덱스 사용량을 확인합니다. 예: `db.viewName.find({...}).explain()`

Amazon DocumentDB 생성형 인공 지능

Amazon DocumentDB는 기계 학습(ML) 및 생성형 인공 지능(AI) 모델이 Amazon DocumentDB에 저장된 데이터를 실시간으로 사용할 수 있는 기능을 제공합니다. 고객은 더 이상 별도의 인프라를 관리하고, 다른 서비스와 연결하기 위한 코드를 작성하고, 기본 데이터베이스에서 데이터를 복제하는 데 시간을 소비할 필요가 없습니다.

인공 지능과가 AI 요구 사항을 AWS 지원하는 방법에 대한 자세한 내용은이 "What-is" 문서를 참조하세요.

주제

- Amazon SageMaker AI Canvas를 사용한 코드 없는 기계 학습
- Amazon DocumentDB에 대한 벡터 검색

Amazon SageMaker AI Canvas를 사용한 코드 없는 기계 학습

Amazon SageMaker AI Canvas를 사용하면 한 줄의 코드를 작성하지 않고도 자체 AI/ML 모델을 구축할 수 있습니다. 회귀 및 예측과 같은 일반적인 사용 사례를 위한 ML 모델을 구축하고 Amazon Bedrock에서 파운데이션 모델(FM) 액세스하고 평가할 수 있습니다. 콘텐츠 생성, 텍스트 추출 및 텍스트 요약을 위해 Amazon SageMaker AI JumpStart에서 퍼블릭 FM에 액세스하여 생성형 AI 솔루션을 지원할 수도 있습니다.

SageMaker AI Canvas를 사용하여 코드 없는 ML 모델을 빌드하는 방법

Amazon DocumentDB는 이제 Amazon SageMaker AI Canvas와 통합되어 Amazon DocumentDB에 저장된 데이터와 함께 코드 없는 기계 학습(ML)을 활성화합니다. 이제 회귀 및 예측 요구 사항을 위한

ML 모델을 구축하고 단일 코드 줄을 작성하지 않고도 Amazon DocumentDB에 저장된 데이터를 사용하여 콘텐츠 요약 및 생성을 위한 파운데이션 모델을 사용할 수 있습니다.

SageMaker AI Canvas는 Amazon DocumentDB 고객이 AI/ML 전문 지식 없이 예측을 생성하거나 단일 코드 줄을 작성할 수 있는 시각적 인터페이스를 제공합니다. 이제 고객은에서 SageMaker AI Canvas 워크스페이스를 시작하고 데이터 준비 및 모델 훈련을 위해 Amazon DocumentDB 데이터를 AWS Management Console가져오고 조인할 수 있습니다. 이제 SageMaker AI Canvas에서 Amazon DocumentDB의 데이터를 사용하여 모델을 구축 및 보강하여 고객 이탈을 예측하고, 사기를 감지하고, 유지 관리 실패를 예측하고, 비즈니스 지표를 예측하고, 콘텐츠를 생성할 수 있습니다. 이제 고객은 SageMaker AI Canvas와 Quick의 기본 통합을 사용하여 팀 간에 ML 기반 인사이트를 게시하고 공유할 수 있습니다. SageMaker AI Canvas의 데이터 수집 파이프라인은 기본적으로 Amazon DocumentDB 보조 인스턴스에서 실행되므로 애플리케이션 및 SageMaker AI Canvas 수집 워크로드의 성능이 저하되지 않습니다.

Amazon DocumentDB 고객은 새 Amazon DocumentDB 코드 없는 ML 콘솔 페이지로 이동하여 새 SageMaker AI Canvas 워크스페이스 또는 사용 가능한 SageMaker AI Canvas 워크스페이스에 연결하여 SageMaker AI Canvas를 시작할 수 있습니다.

SageMaker AI 도메인 및 사용자 프로필 구성

VPC 전용 모드에서 실행 중인 SageMaker AI 도메인에서 Amazon DocumentDB 클러스터에 연결할 수 있습니다. VPC에서 SageMaker AI 도메인을 시작하면 SageMaker AI Studio 및 Canvas 환경에서 데이터 흐름을 제어할 수 있습니다. 이를 통해 인터넷 액세스를 제한하고, 표준 AWS 네트워킹 및 보안 기능을 사용하여 트래픽을 모니터링 및 검사하고, VPC 엔드포인트를 통해 다른 AWS 리소스에 연결할 수 있습니다. Amazon SageMaker AI Canvas 시작하기 및 Amazon SageMaker AI 개발자 안내서에 있는 인터넷 액세스 없이 VPC에서 Amazon SageMaker AI Canvas 구성을 참조하여 Amazon DocumentDB 클러스터에 연결할 SageMaker AI 도메인을 생성합니다. Amazon SageMaker

Amazon DocumentDB 및 SageMaker AI Canvas에 대한 IAM 액세스 권한 구성

연결된 역할 및 자격 증명에 `AmazonDocDBConsoleFullAccess`가 연결된 Amazon DocumentDB 사용자는 AWS Management Console에 액세스할 수 있습니다. 앞서 언급한 역할 또는 자격 증명에 다음 작업을 추가하여 Amazon SageMaker AI Canvas를 사용한 코드 없는 기계 학습에 대한 액세스를 제공합니다.

```
"sagemaker:CreatePresignedDomainUrl",
"sagemaker:DescribeDomain",
"sagemaker:ListDomains",
"sagemaker:ListUserProfiles"
```

SageMaker AI Canvas에 대한 데이터베이스 사용자 및 역할 생성

Amazon DocumentDB의 역할 기반 액세스 제어(RBAC)를 사용하여 데이터베이스에서 사용자가 수행할 수 있는 작업에 대한 액세스를 제한할 수 있습니다. RBAC는 사용자에게 하나 이상의 역할을 부여하여 작동합니다. 이러한 역할은 사용자가 데이터베이스 리소스에서 수행할 수 있는 작업을 결정합니다.

Canvas 사용자는 사용자 이름과 암호 자격 증명을 사용하여 Amazon DocumentDB 데이터베이스에 연결합니다. Amazon DocumentDB RBAC 기능을 사용하여 특정 데이터베이스에 대한 읽기 액세스 권한이 있는 Canvas 사용자의 데이터베이스 사용자/역할을 생성할 수 있습니다.

예를 들어 `createUser`를 사용합니다.

```
db.createUser({
user: "canvas_user",
pwd: "<insert-password>",
roles: [{role: "read", db: "sample-database-1"}]
})
```

이렇게 하면 `sample-database-1` 데이터베이스에 대한 읽기 권한이 있는 `canvas_user`가 생성됩니다. Canvas 분석가는 이 자격 증명을 사용하여 Amazon DocumentDB 클러스터의 데이터에 액세스할 수 있습니다. 자세한 내용은 역할 기반 액세스 제어를 사용한 데이터베이스 액세스 섹션을 참조하세요.

사용 가능한 리전

노코드 통합은 Amazon DocumentDB와 Amazon SageMaker AI Canvas가 모두 지원되는 리전에서 사용할 수 있습니다. 리전에는 다음이 포함됩니다.

- us-east-1(버지니아 북부)
- us-east-2(오하이오)
- us-west-2(오리건)
- ap-northeast-1(도쿄)
- ap-northeast-2(서울)
- ap-south-1(뭄바이)
- ap-southeast-1(싱가포르)
- ap-southeast-2(시드니)
- eu-central-1(프랑크푸르트)
- eu-west-1(아일랜드)

최신 리전 가용성은 Amazon SageMaker AI 개발자 안내서의 Amazon SageMaker AI Canvas를 참조하세요. Amazon SageMaker

Amazon DocumentDB에 대한 벡터 검색

벡터 검색은 기계 학습에서 거리 또는 유사성 지표를 사용하여 벡터 표현을 비교하여 지정된 데이터 포인트와 유사한 데이터 포인트를 찾는 데 사용되는 방법입니다. 벡터 공간에 두 벡터가 가까울수록 기본 항목이 더 유사한 것으로 간주됩니다. 이 기법은 데이터의 의미 체계적 의미를 캡처하는 데 도움이 됩니다. 이 접근 방식은 추천 시스템, 자연어 처리 및 이미지 인식과 같은 다양한 애플리케이션에 유용합니다.

Amazon DocumentDB에 대한 벡터 검색은 JSON 기반 문서 데이터베이스의 유연성과 풍부한 쿼리 기능을 벡터 검색 기능과 결합합니다. 기존 Amazon DocumentDB 데이터 또는 유연한 문서 데이터 구조를 사용하여 의미 검색 경험, 제품 권장 사항, 개인화, 챗봇, 사기 탐지 및 이상 탐지와 같은 기계 학습 및 생성형 AI 사용 사례를 구축하려는 경우 Amazon DocumentDB에 대한 벡터 검색이 이상적인 선택입니다. 벡터 검색은 Amazon DocumentDB 5.0 인스턴스 기반 클러스터에서 사용할 수 있습니다.

주제

- 벡터 삽입
- 벡터 인덱스 생성
- 인덱스 정의 가져오기
- 벡터 쿼리
- 특성 및 제한 사항
- 모범 사례

벡터 삽입

Amazon DocumentDB 데이터베이스에 벡터를 삽입하려면 기존 삽입 방법을 사용할 수 있습니다.

예제

다음 예제에서는 테스트 데이터베이스 내에 5개의 문서 모음이 생성됩니다. 각 문서에는 제품 이름과 해당 벡터 임베딩의 두 필드가 포함되어 있습니다.

```
db.collection.insertMany([
  {"product_name": "Product A", "vectorEmbedding": [0.2, 0.5, 0.8]},
  {"product_name": "Product B", "vectorEmbedding": [0.7, 0.3, 0.9]},
  {"product_name": "Product C", "vectorEmbedding": [0.1, 0.2, 0.5]},
```

```
  {"product_name": "Product D", "vectorEmbedding": [0.9, 0.6, 0.4]},
  {"product_name": "Product E", "vectorEmbedding": [0.4, 0.7, 0.2]}
]);
```

벡터 인덱스 생성

Amazon DocumentDB는 계층적 탐색 가능 스몰 월드(HNSW) 인덱싱과 플랫 압축을 사용한 역 파일(IVFFlat) 인덱싱 방법을 모두 지원합니다. IVFFlat 인덱스는 벡터를 목록으로 분리한 다음 쿼리 벡터에 가장 가까운 해당 목록의 선택된 하위 집합을 검색합니다. 반면 HNSW 인덱스는 벡터 데이터를 다층 그래프로 구성합니다. HNSW는 IVFFlat 에 비해 구축 시간이 느리지만 쿼리 성능 및 리콜을 개선합니다. IVFFlat과 달리 HNSW에는 훈련 단계가 포함되어 있지 않으므로 초기 데이터 로드 없이 인덱스를 생성할 수 있습니다. 대부분의 사용 사례에서는 벡터 검색에 HNSW 인덱스 유형을 사용합니다.

벡터 인덱스를 생성하지 않으면 Amazon DocumentDB가 정확히 가장 가까운 이웃 검색을 수행하여 완벽한 호출을 보장합니다. 그러나 프로덕션 시나리오에서는 속도가 중요합니다. 속도 향상을 위해 일부 재현율을 상쇄할 수 있는 벡터 인덱스를 사용합니다. 벡터 인덱스를 추가하면 쿼리 결과가 달라질 수 있다는 점에 유의해야 합니다.

템플릿

다음 `createIndex` 또는 `runCommand` 템플릿을 사용하여 벡터 필드에 벡터 인덱스를 빌드할 수 있습니다.

Using createIndex

mongosh 및 Java와 같은 특정 드라이버에서의 `vectorOptions` 파라미터를 사용하면 오류가 발생할 `createIndex` 수 있습니다. 이러한 경우를 사용합니다`runCommand`.

```
db.collection.createIndex(
  { "<vectorField>": "vector" },
  { "name": "<indexName>",
    "vectorOptions": {
      "type": " <hnsw> | <ivfflat> ",
      "dimensions": <number_of_dimensions>,
      "similarity": " <euclidean> | <cosine> | <dotProduct> ",
      "lists": <number_of_lists> [applicable for IVFFlat],
      "m": <max number of connections> [applicable for HNSW],
      "efConstruction": <size of the dynamic list for index build> [applicable for
 HNSW]
    }
  }
```

```
);
```

Using runCommand

mongosh 및 Java와 같은 특정 드라이버에서의 `vectorOptions` 파라미터를 사용하면 오류가 발생할 `createIndex` 수 있습니다. 이러한 경우를 사용합니다`runCommand`.

```
db.runCommand(
  { "createIndexes": "<collection>",
  "indexes": [{
      key: { "<vectorField>": "vector" },
      vectorOptions: {
          type: " <hnsw> | <ivfflat> ",
          dimensions: <number of dimensions>,
          similarity: " <euclidean> | <cosine> | <dotProduct> ",
          lists: <number_of_lists> [applicable for IVFFlat],
          m: <max number of connections> [applicable for HNSW],
          efConstruction: <size of the dynamic list for index build> [applicable for
 HNSW]
          },
      name: "myIndex"
      }]
  }
);
```

파라미터	요구 사항	데이터 유형	설명	값
name	선택 사항	문자열	인덱스의 이름을 지정합니다.	영숫자
type	선택 사항		인덱스의 유형을 지정합니다.	지원: hnsw 또는 ivfflat 기본값: HNSW(엔진 패치 3.0.4574 이상)
dimensions	필수	정수	벡터 데이터의 차원 수를 지정합니다.	최대 2,000개의 차원.

파라미터	요구 사항	데이터 유형	설명	값
similarity	필수	문자열	유사성 계산에 사용되는 거리 지표를 지정합니다.	• **euclidean** • **cosine** • **dotProduct**
lists	IVFFlat에 필요	정수	IVFFlat 인덱스가 벡터 데이터를 그룹화하는 데 사용하는 클러스터 수를 지정합니다. 권장 설정은 최대 100만 개 문서의 경우 문서 수/1000, 그리고 100만 개가 넘는 문서의 경우 `sqrt(# of documents)` 입니다.	최소: 1 최대: 특성 및 제한 사항 아래의 인스턴스 유형별 목록 표를 참조하세요.
m	선택 사항	정수	HNSW 인덱스의 최대 연결 수를 지정합니다.	기본값: 16 범위 [2, 100]
efConstruction	선택 사항	정수	HNSW 인덱스에 대한 그래프를 구성할 동적 후보 목록의 크기를 지정합니다. `efConstruction` 은 (2 * m)보다 크거나 같아야 합니다.	기본값: 64 범위 [4, 1000]

검색의 정확도/리콜, 빌드 시간 및 성능에 영향을 미치므로 IVFFlat `lists`, `m` 및 HNSW `efConstruction`과 같은 하위 파라미터의 값을 적절하게 설정하는 것이 중요합니다. 목록 값이 높을수록 각 목록의 벡터 수가 줄어들어 리전이 작아지므로 쿼리 속도가 증가합니다. 그러나 리전 크기가 작을수록 더 많은 리콜 오류가 발생하여 정확도가 낮아질 수 있습니다. HNSW의 경우 `m` 및 `efConstruction` 값을 늘리면 정확도가 높아지지만 인덱스 구축 시간과 크기도 증가합니다. 다음 예를 참조하세요.

예제

HNSW

```
db.collection.createIndex(
  { "vectorEmbedding": "vector" },
  { "name": "myIndex",
    "vectorOptions": {
      "type": "hnsw",
      "dimensions": 3,
      "similarity": "euclidean",
      "m": 16,
      "efConstruction": 64
    }
  }
);
```

IVFFlat

```
db.collection.createIndex(
  { "vectorEmbedding": "vector" },
  { "name": "myIndex",
    "vectorOptions": {
      "type": "ivfflat",
      "dimensions": 3,
      "similarity": "euclidean",
      "lists":1
    }
  }
)
```

인덱스 정의 가져오기

다음 `getIndexes` 명령을 사용하여 벡터 인덱스를 포함한 인덱스의 세부 정보를 볼 수 있습니다.

예제

```
db.collection.getIndexes()
```

출력 예

```
[
 {
  "v" : 4,
  "key" : {
   "_id" : 1
  },
  "name" : "_id_",
  "ns" : "test.collection"
 },
 {
  "v" : 4,
  "key" : {
   "vectorEmbedding" : "vector"
  },
  "name" : "myIndex",
  "vectorOptions" : {
   "type" : "ivfflat",
   "dimensions" : 3,
   "similarity" : "euclidean",
   "lists" : 1
  },
  "ns" : "test.collection"
 }
]
```

벡터 쿼리

Amazon DocumentDB는 벡터 쿼리를 위해 두 개의 벡터 검색 연산자를 지원합니다.

클래식 벡터 검색 연산자

다음 템플릿을 사용하여 벡터를 쿼리합니다.

```
db.collection.aggregate([
  {
    $search: {
      "vectorSearch": {
```

```
        "vector": <query vector>,
        "path": "<vectorField>",
        "similarity": "<distance metric>",
        "k": <number of results>,
        "probes":<number of probes> [applicable for IVFFlat],
        "efSearch":<size of the dynamic list during search> [applicable for HNSW]
      }
    }
  }
]);
```

파라미터	요구 사항	유형	설명	값
vectorSearch	필수	연산자	$search 명령 내에서 벡터를 쿼리하는 데 사용됩니다.	
vector	필수	배열	유사한 벡터를 찾는 데 사용할 쿼리 벡터를 나타냅니다.	
path	필수	문자열	벡터 필드의 이름을 정의합니다.	
k	필수	정수	검색에서 반환하는 결과의 수를 지정합니다.	
similarity	필수	문자열	유사성 계산에 사용되는 거리 지표를 지정합니다.	• **euclidean** • **cosine** • **dotProduct**
probes	선택 사항	정수	벡터 검색에서 검사할 클러스터 수입니다. 값이 높을수록 속도 비용으로 더 나은 리	기본값: 1

파라미터	요구 사항	유형	설명	값
			콜을 제공합니다. 정확히 가장 가까운 이웃 검색을 위한 목록 수로 설정할 수 있습니다(실행 계획기가 인덱스를 사용하지 않는 시점). 미세 조정을 시작하기 위한 권장 설정은 `sqrt(# of lists)`입니다.	
`efSearch`	선택 사항	정수	검색 중에 HNSW 인덱스가 사용하는 동적 후보 목록의 크기를 지정합니다. `efSearch` 값이 높을수록 속도 비용으로 더 나은 리콜을 제공합니다.	기본값: 40 범위 [1, 1000]

원하는 성능과 정확도를 달성하려면 `efSearch`(HNSW) 또는 `probes`(IVFFlat)의 값을 미세 조정하는 것이 중요합니다. 다음 예제 작업을 참조하세요.

HNSW

```
db.collection.aggregate([
  {
    $search: {
      "vectorSearch": {
        "vector": [0.2, 0.5, 0.8],
        "path": "vectorEmbedding",
        "similarity": "euclidean",
```

```
        "k": 2,
        "efSearch": 40
      }
    }
  }
]);
```

IVFFlat

```
db.collection.aggregate([
  {
    $search: {
      "vectorSearch": {
        "vector": [0.2, 0.5, 0.8],
        "path": "vectorEmbedding",
        "similarity": "euclidean",
        "k": 2,
        "probes": 1
      }
    }
  }
]);
```

출력 예

이 작업의 출력은 다음과 같이 표시됩니다.

```
{ "_id" : ObjectId("653d835ff96bee02cad7323c"), "product_name" : "Product A",
 "vectorEmbedding" : [ 0.2, 0.5, 0.8 ] }
{ "_id" : ObjectId("653d835ff96bee02cad7323e"), "product_name" : "Product C",
 "vectorEmbedding" : [ 0.1, 0.2, 0.5 ] }
```

$vectorSearch 연산자(Amazon DocumentDB 8.0 이상에서 사용 가능)

다음 템플릿을 사용하여 벡터를 쿼리합니다.

```
db.collection.aggregate([
{
  "$vectorSearch": {
    "exact": true | false,
    "index": "<index-name>" [supports only HNSW index],
    "limit": <number-of-results> [same as k],
```

```
    "path": "<vector field-to-search>",
    "queryVector": <array-of-numbers>,
    "numCandidates": <number-of-candidates> [same as efSearch],
  }
}])
```

특성 및 제한 사항

버전 호환성

- Amazon DocumentDB에 대한 벡터 검색은 Amazon DocumentDB 5.0+ 인스턴스 기반 클러스터에서만 사용할 수 있습니다.

벡터

- Amazon DocumentDB는 최대 2,000개의 차원으로 벡터를 인덱싱할 수 있습니다. 하지만 인덱스 없이 최대 16,000개의 차원을 저장할 수 있습니다.

인덱스

- IVFFlat 인덱스 생성의 경우 목록 파라미터의 권장 설정은 최대 100만 개의 문서의 경우 문서 수/1000 및 100만 개가 넘는 문서의 경우 `sqrt(# of documents)`입니다. 작동 메모리 제한으로 인해 Amazon DocumentDB는 차원 수에 따라 목록 파라미터의 특정 최대값을 지원합니다. 참고로 다음 표에서는 500, 1,000 및 2,000 차원의 벡터에 대한 목록 파라미터의 최대값을 제공합니다.

인스턴스 유형	500개 차원이 있는 목록	1,000개 차원이 있는 목록	2,000개 차원이 있는 목록
t3.med	372	257	150
r5.l	915	741	511
r5.xl	1,393	1,196	901
r5.2xl	5,460	5,230	4,788
r5.4xl	7,842	7,599	7,138
r5.8xl	11,220	10,974	10,498

인스턴스 유형	500개 차원이 있는 목록	1,000개 차원이 있는 목록	2,000개 차원이 있는 목록
r5.12xl	13,774	13,526	13,044
r5.16xl	15,943	15,694	15,208
r5.24xl	19,585	19,335	18,845

- compound, sparse 또는 partial과 같은 다른 인덱스 옵션은 벡터 인덱스에서 지원되지 않습니다.
- Amazon DocumentDB 5.0의 HNSW 인덱스에는 병렬 인덱스 빌드가 지원되지 않습니다.

벡터 쿼리

- 벡터 검색 쿼리의 경우 최적의 결과를 위해 probes 또는 efSearch와 같은 파라미터를 미세 조정하는 것이 중요합니다. probes 또는 efSearch 파라미터의 값이 높을수록 재호출이 높고 속도가 낮아집니다. 프로브 파라미터 미세 조정을 시작하기 위한 권장 설정은 sqrt(# of lists)입니다.

모범 사례

Amazon DocumentDB의 벡터 검색을 사용한 작업 모범 사례에 대해서 알아봅니다. 이 섹션은 새로운 모범 사례가 확인되는 대로 지속적으로 업데이트됩니다.

- 플랫 압축(IVFFlat) 인덱스 생성이 포함된 반전된 파일에는 유사성을 기반으로 데이터 포인트를 클러스터링하고 구성하는 작업이 포함됩니다. 인덱스의 효과를 높이려면 인덱스를 생성하기 전에 일부 데이터를 로드합니다.
- 벡터 검색 쿼리의 경우 최적의 결과를 위해 probes 또는 efSearch와 같은 파라미터를 미세 조정하는 것이 중요합니다. probes 또는 efSearch 파라미터의 값이 높을수록 재호출이 높고 속도가 낮아집니다. probes 파라미터 미세 조정을 시작하기 위한 권장 설정은 sqrt(lists)입니다.

리소스

- 이제 AWS 블로그에서 Amazon DocumentDB에 대한 벡터 검색을 정식으로 사용할 수 있습니다.
- 의미 검색 코드 샘플
- Amazon DocumentDB 벡터 검색 코드 샘플

변경 스트림과 AWS Lambda 함께 사용

Amazon DocumentDB는와 통합되어 Lambda 함수를 사용하여 변경 스트림의 레코드를 처리할 AWS Lambda 수 있습니다. Lambda 이벤트 소스 매핑은 Lambda 함수를 호출하여 Lambda를 직접 호출하지 않는 Amazon DocumentDB 이벤트를 처리하는 데 사용할 수 있는 리소스입니다. Amazon DocumentDB 변경 스트림을 이벤트 소스로 사용하면 데이터 변경에 응답하는 이벤트 기반 애플리케이션을 구축할 수 있습니다. 예를 들어 Lambda 함수를 사용하여 새 문서를 처리하거나, 기존 문서에 대한 업데이트를 추적하거나, 삭제된 문서를 로그할 수 있습니다.

이벤트 소스 매핑을 구성하여 Amazon DocumentDB 변경 스트림의 레코드를 Lambda 함수로 전송하도록 할 수 있습니다. 이벤트는 효율성 향상을 위해 한 번에 하나씩 보내거나 배치할 수 있으며 순서대로 처리됩니다. 이벤트 소스 매핑의 배치 동작은 특정 시간대(0 - 300초) 또는 배치 레코드 수(최대 10,000개 레코드 제한)를 기준으로 구성할 수 있습니다. 여러 개의 이벤트 소스 매핑을 생성하여 동일한 데이터를 여러 개의 Lambda 함수로 처리하거나 여러 스트림에서 구별되는 항목을 단일 함수로 처리할 수 있습니다.

함수가 오류를 반환하면 Lambda는 처리가 성공할 때까지 배치를 재시도합니다. 변경 스트림의 이벤트가 만료된 경우 Lambda는 이벤트 소스 매핑을 비활성화합니다. 이 경우 새 이벤트 소스 매핑을 생성하고 원하는 시작 위치로 구성할 수 있습니다. Lambda 이벤트 소스 매핑은 폴러의 분산 특성으로 인해 이벤트를 한 번 이상 처리합니다. 결과적으로 Lambda 함수는 드물게 중복 이벤트를 수신할 수 있습니다. AWS Lambda 함수 작업에 대한 모범 사례를 따르고 멱등성 함수를 구축하여 중복 이벤트와 관련된 문제를 방지합니다. 자세한 내용은 AWS Lambda 개발자 안내서의 Amazon DocumentDB AWS Lambda console 에서 사용을 참조하세요.

성능 모범 사례에 따라 Lambda 함수는 수명이 짧아야 합니다. 불필요한 처리 지연을 방지하기 위해 복잡한 로직도 실행하지 않아야 합니다. 특히 고속 스트림의 경우 장기 실행 중인 동기식 Lambda보다 비동기식 사후 처리 단계 함수 워크플로를 트리거하는 것이 좋습니다. 에 대한 자세한 내용은 AWS Lambda 개발자 안내서를 AWS Lambda참조하세요.

제한 사항

다음은 Amazon DocumentDB 및 작업 시 고려해야 할 제한 사항입니다. AWS Lambda

- AWS Lambda 는 현재 Amazon DocumentDB 4.0 및 5.0에서만 지원됩니다.
- AWS Lambda 는 현재 탄력적 클러스터 또는 글로벌 클러스터에서 지원되지 않습니다.
- AWS Lambda 페이로드 크기는 6MB를 초과할 수 없습니다. Lambda 배치 크기에 대한 자세한 내용은 AWS Lambda 개발자 가이드의 Lambda 이벤트 소스 매핑 섹션의 "배치 동작"을 참조하세요.

Amazon DocumentDB의 MongoDB 호환성

Amazon DocumentDB는 MongoDB 4.0, MongoDB 5.0 및 MongoDB 8.0을 포함한 MongoDB 호환성을 지원합니다. MongoDB 호환은 MongoDB 데이터베이스에서 현재 이미 사용하고 있는 애플리케이션, 드라이버 및 도구의 대다수를 거의 변경하지 않고 Amazon DocumentDB에서 사용할 수 있음을 의미합니다. 이 섹션에서는 MongoDB와의 Amazon DocumentDB 호환성에 대해 알아야 할 모든 것을 설명합니다. 여기에는 새로운 기능과 특성, 시작, 마이그레이션 경로 및 기능 차이 등이 포함됩니다.

주제

- MongoDB 8.0 호환성
- MongoDB 5.0 호환성
- MongoDB 4.0 호환성
- 기능적 차이: Amazon DocumentDB 및 MongoDB
- Amazon DocumentDB에서 지원되는 MongoDB API, 작업 및 데이터 형식

MongoDB 8.0 호환성

주제

- Amazon DocumentDB 8.0의 새로운 기능
- Amazon DocumentDB 8.0 시작하기
- Amazon DocumentDB 5.0 또는 8.0으로 업그레이드 또는 마이그레이션
- 기능적 차이

Amazon DocumentDB 8.0의 새로운 기능

Amazon DocumentDB 8.0은 쿼리 성능을 최대 7배 개선하고 압축률을 최대 5배 개선하여 더 저렴한 비용으로 고성능 애플리케이션을 구축할 수 있습니다. Amazon DocumentDB 8.0은 MongoDB 8.0과 완전한 유선 프로토콜 호환성을 제공합니다. 아래 요약에서는 Amazon DocumentDB 8.0에 도입된 몇 가지 주요 기능을 소개합니다. 새로운 기능의 전체 목록을 보려면 릴리스 노트 섹션을 참조하세요.

- MongoDB 8.0 API 드라이버에 대한 지원을 추가하여 MongoDB 8.0과의 호환성을 제공합니다. 또한 MongoDB API 버전 6.0 및 7.0을 사용하여 빌드된 애플리케이션을 지원합니다.
- Amazon DocumentDB 8.0의 새로운 쿼리 플래너는 집계 파이프라인 최적화 및 개별 명령을 지원하는 동시에 성능 개선을 집계 단계 연산자로 확장합니다.

- Amazon DocumentDB 8.0은 데이터 정렬 및 보기를 지원합니다.
- $replaceWith, $vectorSearch, $merge, $set, $unset, $bucket의 6가지 새로운 집계 단계와 $pow, $rand, $dateTrunc의 3가지 새로운 집계 연산자를 제공합니다.
- 새로운 버전의 텍스트 인덱스: Amazon DocumentDB 8.0의 텍스트 인덱스 v2에는 추가 토큰이 도입되어 텍스트 검색 기능이 향상되었습니다.
- 병렬 벡터 인덱스 빌드를 통해 Amazon DocumentDB 8.0은 인덱스 빌드 시간을 최대 30배 단축합니다.
- 스토리지 한도를 256TiB로 늘렸습니다.

Amazon DocumentDB 8.0 시작하기

Amazon DocumentDB 8.0을 시작하려면 섹션을 참조하세요Amazon DocumentDB로 시작하기. AWS Management Console 또는 AWS SDK AWS CLI또는를 사용하여 새 Amazon DocumentDB 8.0 클러스터를 생성할 수 있습니다 CloudFormation. Amazon DocumentDB에 연결할 경우 MongoDB 5.0 이상과 호환되는 MongoDB 드라이버 또는 유틸리티를 사용해야 합니다.

> **Note**
>
> AWS SDK AWS CLI또는 CloudFormation를 사용하는 경우 엔진 버전은 기본적으로 5.0.0으로 설정됩니다. 파라미터를 명시적으로 지정`engineVersion = 8.0.0`하여 새 Amazon DocumentDB 8.0 클러스터를 생성하거나 새 Amazon DocumentDB 4.0 클러스터를 `engineVersion = 4.0.0` 생성하거나 새 Amazon DocumentDB 3.6 클러스터를 `engineVersion = 3.6.0` 생성해야 합니다. 지정된 Amazon DocumentDB 클러스터의 경우 AWS CLI 를 사용하여 클러스터 버전을 확인하거나 Amazon DocumentDB 관리 콘솔을 `describe-db-clusters` 사용하여 특정 클러스터의 엔진 버전 번호를 볼 수 있습니다.

Amazon DocumentDB 5.0 및 8.0은 클러스터의 및 `t4.medium` 인스턴스 유형과 같은 Amazon EC2 `r8g` Graviton4 프로세서`r6g`와 Graviton2 프로세서를 지원하며 지원되는 모든에서 사용할 수 있습니다 AWS 리전 (참조인스턴스). 요금에 대한 자세한 내용은 Amazon DocumentDB(MongoDB 호환) 요금을 참조하세요.

Amazon DocumentDB 5.0 또는 8.0으로 업그레이드 또는 마이그레이션

MongoDB, mongorestore, mongoimport 및 mongoexport와 같은 또는 유틸리티를 사용하여 MongoDB 3.6 또는 MongoDBgoAmazon DocumentDB 4.0에서 Amazon DocumentDB 5.0 AWS DMS 또는

Amazon DocumentDB 8.0으로 마이그레이션할 수 있습니다. ??? 마이그레이션에 대한 지침은 섹션을 참조하세요를 사용하여 Amazon DocumentDB 클러스터 업그레이드 AWS Database Migration Service.

기능적 차이

Amazon DocumentDB 5.0과 8.0의 기능적 차이점

Amazon DocumentDB 8.0 릴리스에서는 Amazon DocumentDB 5.0과 Amazon DocumentDB 8.0 간에 기능적 차이가 있습니다.

- 플래너 v1은 Amazon DocumentDB 5.0의 기본 쿼리 플래너인 반면, 성능이 더 뛰어난 플래너 v3는 Amazon DocumentDB 8.0의 기본입니다.
- $merge와 같은 보기, 데이터 정렬 및 연산자를 포함한 Amazon DocumentDB 8.0의 새로운 기능은 플래너 v3와만 호환됩니다.
- 압축은 Amazon DocumentDB 8.0에서 기본적으로 켜져 있으며 Zstandard 알고리즘을 사용하도록 설정되어 있습니다. 또한 Amazon DocumentDB 8.0에서 'enabled'는 더 이상 유효하지 않습니다. Zstd, LZ4 및 none 중에서 선택할 수 있습니다.
- Amazon DocumentDB 8.0은 데이터베이스 연결에 TLS 1.2 이상만 지원합니다.

> **Important**
>
> Amazon DocumentDB 8.0에는 TLS 1.2 이상이 필요합니다. TLS 1.0 및 TLS 1.1은 더 이상 지원되지 않습니다.

MongoDB 5.0 호환성

주제

- Amazon DocumentDB 5.0의 새로운 기능
- Amazon DocumentDB 5.0.1의 새로운 기능
- Amazon DocumentDB 5.0 시작하기
- Amazon DocumentDB 5.0으로 업그레이드 또는 마이그레이션
- 기능적 차이

Amazon DocumentDB 5.0의 새로운 기능

Amazon DocumentDB 5.0은 스토리지 제한 및 클라이언트 측 필드 레벨 암호화를 포함하는 새로운 기능과 특성을 소개합니다. 아래 요약은 Amazon DocumentDB 5.0에서 도입된 몇 가지 주요 특성을 소개합니다. 새로운 기능의 전체 목록을 보려면 릴리스 노트 섹션을 참조하세요.

- 인스턴스 기반의 모든 Amazon DocumentDB 클러스터 및 샤드 기반의 탄력적 클러스터에 대해 128TiB로 스토리지 제한을 높였습니다.
- Amazon DocumentDB 5.0 Engine Version 3.0.775) 소개
  - MongoDB 5.0 API 드라이버 지원
  - 클라이언트 측 FLE(Field Level Encryption)를 지원합니다. 이제 Amazon DocumentDB 클러스터에 데이터를 쓰기 전에 클라이언트 측에서 필드를 암호화할 수 있습니다. 자세한 내용은 클라이언트 측 필드 레벨 암호화 단원을 참조하십시오.
  - 새 집계 연산자:`$dateAdd`, `$dateSubtract`
  - 연산자를 사용한 `$elemMatch` 인덱스 지원. 따라서 쿼리가 `$elemMatch`이면 인덱스 스캔이 발생합니다.

Amazon DocumentDB는 모든 MongoDB 5.0 기능을 지원하지는 않습니다. Amazon DocumentDB 5.0을 구축할 당시에는 고객이 가장 많이 구축하라고 요청한 기능과 특성을 다시 연구했습니다. 앞으로도 고객들이 구축을 요청하는 내용을 바탕으로 MongoDB 5.0 기능을 추가할 예정입니다. 지원되는 APIsAmazon DocumentDB에서 지원되는 MongoDB API, 작업 및 데이터 형식.

Amazon DocumentDB 5.0.1의 새로운 기능

Amazon DocumentDB 5.0.1은 Amazon DocumentDB 5.0 메이저 버전의 첫 번째 마이너 버전 릴리스입니다. 마이너 버전에는 새로운 기능, 보안 수정 및 버그 수정과 같은 개선 사항이 포함되어 있습니다. 마이너 버전 업그레이드를 사용하여 5.0.0에서 5.0.1로 업그레이드할 수 있습니다. 자세한 내용은 Amazon DocumentDB 마이너 버전 업그레이드 단원을 참조하십시오.

- 새 연산자(`$rand`, , `$pow`, `$dateFromParts`)를 사용하여 집계 기능을 개선`$dateToParts`했습니다.
- 인스턴스를 모니터링하기 위한 활성 연결 지표입니다. 자세한 내용은 인스턴스 할당량 단원을 참조하십시오.
- CRUD 작업에 대한 CloudWatch의 세분화된 명령 수준 성능 지표(`find`, , `insert`, `findAndModifyupdate`, `delete`, `aggregatecount`, `distinct`, `getMore`,

abortTransaction, , commitTransaction), 향상된 운영 모니터링을 위한 노출 AvgDurationP100Duration, 및 MaxConcurrent차원.

- Amazon DocumentDB 5.0.1은 데이터베이스 연결에 대해 TLS 1.2 이상만 지원합니다.
- 가비지 수집 프로세스의 전체 상태에 대한 추가 정보를 표시collstats하기 위해 및 gcRuntimeStatsstorageSegmentExtended에 수집 수준 지표의 documentFragmentStats, storageSegmentBase 및 분류를 추가했습니다.

Amazon DocumentDB 5.0 시작하기

Amazon DocumentDB 5.0을 시작하려면 섹션을 참조하세요Amazon DocumentDB로 시작하기. AWS Management Console 또는 AWS SDK AWS CLI또는를 사용하여 새 Amazon DocumentDB 5.0 클러스터를 생성할 수 있습니다 CloudFormation. Amazon DocumentDB에 연결할 경우 MongoDB 5.0 이상과 호환되는 MongoDB 드라이버 또는 유틸리티를 사용해야 합니다.

> **Note**
>
> AWS SDK AWS CLI또는 CloudFormation를 사용하는 경우 엔진 버전은 기본적으로 5.0.0으로 설정됩니다. 새 Amazon DocumentDB 4.0 클러스터 또는 engineVersion = 3.6.0를 생성하거나 새 Amazon DocumentDB 3.6 클러스터를 생성하려면 engineVersion = 4.0.0 파라미터를 명시적으로 지정해야 합니다. 지정된 Amazon DocumentDB 클러스터의 경우 AWS CLI 를 사용하여 클러스터 버전을 확인하거나 Amazon DocumentDB 관리 콘솔을 describe-db-clusters 사용하여 특정 클러스터의 엔진 버전 번호를 볼 수 있습니다.

Amazon DocumentDB 5.0 및 8.0은 클러스터의 및 t4.medium 인스턴스 유형과 같은 Amazon EC2 r8g Graviton4 프로세서r6g와 Graviton2 프로세서를 지원하며 지원되는 모든에서 사용할 수 있습니다 AWS 리전 (참조인스턴스). 요금에 대한 자세한 내용은 Amazon DocumentDB(MongoDB 호환) 요금을 참조하세요.

Amazon DocumentDB 5.0으로 업그레이드 또는 마이그레이션

MongoDB, mongorestore, mongoimport 및 mongoexport와 같은 AWS DMS 또는 유틸리티를 사용하여 MongoDB 3.6 또는 MongoDB 4.0에서 Amazon DocumentDB 5.0으로 마이그레이션할 수 있습니다. ??? 마이그레이션에 대한 지침은 섹션을 참조하세요를 사용하여 Amazon DocumentDB 클러스터 업그레이드 AWS Database Migration Service.

기능적 차이

Amazon DocumentDB 4.0 및 5.0 간의 기능적 차이

Amazon DocumentDB 5.0 릴리스는 Amazon DocumentDB 4.0 및 Amazon DocumentDB 5.0과 비교해 다음과 같은 기능적 차이가 있습니다.

- 이제 백업 기본 제공 역할이 `serverStatus`를 지원합니다. 작업 - 백업 역할을 가진 개발자 및 응용 프로그램은 Amazon DocumentDB 클러스터의 상태에 대한 통계를 수집할 수 있습니다.
- `replSetGetConfig` 출력에서 `SecondaryDelaySecs` 필드가 `slaveDelay`를 대체합니다.
- hello 명령이 `isMaster`를 대체함 - hello는 Amazon DocumentDB 클러스터의 역할을 설명하는 문서를 반환합니다.
- Amazon DocumentDB 5.0은 이제 첫 번째 중첩 수준에서 `$elemMatch` 연산자를 사용한 인덱스 스캔을 지원합니다. 쿼리 전용 필터에 한 레벨의 `$elemMatch` 필터가 있는 경우 인덱스 스캔이 지원되지만 중첩된 `$elemMatch` 쿼리가 포함된 경우에는 필터가 지원되지 않습니다.

  예를 들어, Amazon DocumentDB 5.0에서 `$elemMatch` 연산자를 중첩 수준에 포함하면 Amazon DocumentDB 4.0에서처럼 값이 반환되지 않습니다.

```
db.foo.insert(
[
    {a: {b: 5}},
    {a: {b: [5]}},
    {a: {b: [3, 7]}},
    {a: [{b: 5}]},
    {a: [{b: 3}, {b: 7}]},
    {a: [{b: [5]}]},
    {a: [{b: [3, 7]}]},
    {a: [[{b: 5}]]},
    {a: [[{b: 3}, {b: 7}]]},
    {a: [[{b: [5]}]]},
    {a: [[{b: [3, 7]}]]}
]);

// DocumentDB 5.0
> db.foo.find({a: {$elemMatch: {b: {$elemMatch: {$lt: 6, $gt: 4}}}}}, {_id: 0})
{ "a" : [ { "b" : [ 5 ] } ] }

// DocumentDB 4.0
> db.foo.find({a: {$elemMatch: {b: {$elemMatch: {$lt: 6, $gt: 4}}}}}, {_id: 0})
{ "a" : [ { "b" : [ 5 ] } ] }
```

```
{ "a" : [ [ { "b" : [ 5 ] } ] ] }
```

- Amazon DocumentDB 4.0의 "$" 프로젝션은 모든 필드가 포함된 모든 문서를 반환합니다. Amazon DocumentDB 5.0에서는 "$" 프로젝션이 있는 find 명령은 "$" 프로젝션과 일치하는 필드만 포함된 쿼리 매개 변수와 일치하는 문서를 반환합니다.
- Amazon DocumentDB 5.0에서 `$regex` 및 `$options` 쿼리 매개 변수가 있는 find 명령은 "`$regex` 및 `$options` 모두에서 옵션을 설정할 수 없습니다"라는 오류를 반환합니다.
- Amazon DocumentDB 5.0에서 `$indexOfCP`는 이제 다음과 같은 경우에 "-1"을 반환합니다.
  - 문자열 표현식에서 하위 문자열을 찾을 수 없음, 또는
  - 시작이 끝보다 큰 숫자임, 또는
  - 시작이 문자열의 바이트 길이보다 큰 숫자임.
- Amazon DocumentDB 4.0에서 `$indexOfCP`는 시작 위치가 끝 또는 문자열의 바이트 길이보다 큰 경우 "0"을 반환합니다.
- Amazon DocumentDB 5.0에서 `_id fields`의 프로젝션 연산(예: `{"_id.nestedField" : 1}`)은 프로젝션 필드만 포함하는 문서를 반환합니다. 단, Amazon DocumentDB 4.0에서는 중첩 필드 프로젝션 명령이 문서를 필터링하지 않습니다.

MongoDB 4.0 호환성

주제

- Amazon DocumentDB 4.0 특성
- Amazon DocumentDB 4.0 시작하기
- Amazon DocumentDB 4.0으로 업그레이드 또는 마이그레이션
- 기능적 차이

Amazon DocumentDB 4.0 특성

Amazon DocumentDB 4.0에는 ACID 트랜잭션과 스트림 변경을 위한 개선 사항을 포함한 많은 새로운 기능과 특성이 도입되었습니다. 아래 요약은 Amazon DocumentDB 4.0에서 소개된 주요 기능 중 일부 특성을 보여줍니다. 기능의 전체 목록을 보려면 릴리스 노트 섹션을 참조하세요.

- ACID 트랜잭션: Amazon DocumentDB는 이제 여러 문서, 명령문, 컬렉션 및 데이터베이스에서 트랜잭션을 수행하는 기능을 지원합니다. 트랜잭션은 Amazon DocumentDB 클러스터 내의 하나 이상의 문서에서 원자적이고 일관적이며 격리되고 내구성이 뛰어난(ACID) 작업을 수행할 수 있도록 하

여 애플리케이션 개발을 단순화합니다. 자세한 내용은 Amazon DocumentDB의 트랜잭션 섹션을 참조하십시오.

- 변경 스트림: 이제 클러스터 수준(`client.watch()` 또는 `mongo.watch()`)과 데이터베이스(`db.watch()`)에서 변경 스트림을 열 수 있고, 변경 스트림 커서를 열도록 `startAtOperationTime`을 지정할 수 있으며, 마지막으로 변경 스트림 보존 기간을 7일(이전에는 24시간)까지 연장할 수 있습니다. 자세한 내용은 Amazon DocumentDB에서 변경 스트림 사용 단원을 참조하십시오.
- AWS Database Migration Service (AWS DMS): 이제 AWS DMS 를 사용하여 MongoDB 4.0 워크로드를 Amazon DocumentDB로 마이그레이션할 수 있습니다. AWS DMS 이제는 MongoDB 4.0 소스, Amazon DocumentDB 4.0 대상 및 Amazon DocumentDB 3.6과 4.0 간의 업그레이드를 수행하기 위한 Amazon DocumentDB 3.6 소스를 지원합니다. 자세한 내용은 AWS DMS 설명서를 참조하세요.
- 성능 및 인덱싱: 이제 `$lookup`을 통한 인덱스를 활용하고, 하나의 필드 또는 하나의 필드와 `_id` 필드(컬렉션에서 읽을 필요 없이 인덱스에서 직접 제공 가능(가려진 쿼리))를 포함하는 프로젝션이 있는 쿼리를 찾을 수 있으며, `findAndModify`와 함께 `hint()`를 실행할 수 있고, `$addToSet`에 대한 성능 최적화 및 전체 인덱스 크기를 줄일 수 있는 개선 사항이 제공됩니다. 자세한 내용은 릴리스 노트 섹션을 참조하십시오.
- 연산자: Amazon DocumentDB 4.0은 이제 다양한 다음과 같은 새로운 집계 연산자를 지원합니다: `$ifNull`, `$replaceRoot`, `$setIsSubset`, `$setIntersection`, `$setUnion`, `$setEquals`. Amazon DocumentDB에서 지원되는 MongoDB API, 작업 및 데이터 형식에서 지원하는 모든 MongoDB API, 작업 및 데이터 유형을 볼 수 있습니다.
- 역할 기반 액세스 제어(RBAC): 이제 `ListCollection` 및 `ListDatabase` 명령을 모두 사용하여 `authorizedCollections` 및 `authorizedDatabases` 파라미터를 선택적으로 사용하여 사용자가 `listCollections` 및 `listDatabase` 역할을 필요로 하지 않고 액세스 권한이 있는 컬렉션 및 데이터베이스를 나열할 수 있습니다. 또한 `KillCursor` 역할 필요 없이 자신의 커서를 종료할 수 있습니다.

Amazon DocumentDB는 일부 MongoDB 4.0 기능을 지원하지 않습니다. Amazon DocumentDB 4.0을 구축할 때 고객이 가장 많이 구축하도록 요구했던 기능과 특성을 거꾸로 수행했습니다. 앞으로도 고객들이 구축을 요청하는 내용을 바탕으로 MongoDB 4.0 기능을 추가적으로 추가할 예정입니다. 예를 들어, Amazon DocumentDB 4.0은 현재 MongoDB 4.0에 도입된 유형 변환 연산자나 문자열 연산자를 지원하지 않습니다. 지원되는 APIsAmazon DocumentDB에서 지원되는 MongoDB API, 작업 및 데이터 형식.

Amazon DocumentDB 4.0 시작하기

Amazon DocumentDB 4.0을 시작하려면 섹션을 참조하세요Amazon DocumentDB로 시작하기. AWS Management Console 또는 AWS SDK AWS CLI또는를 사용하여 새 Amazon DocumentDB 4.0 클러스터를 생성할 수 있습니다 CloudFormation. Amazon DocumentDB에 연결할 경우 MongoDB 4.0 이상과 호환되는 MongoDB 드라이버 또는 유틸리티를 사용해야 합니다.

> **Note**
>
> AWS SDK AWS CLI또는 CloudFormation를 사용하는 경우 엔진 버전은 기본적으로 5.0.0으로 설정됩니다. 새 Amazon DocumentDB 4.0 클러스터 또는 `engineVersion = 3.6.0`를 생성하거나 새 Amazon DocumentDB 3.6 클러스터를 생성하려면 `engineVersion = 4.0.0` 파라미터를 명시적으로 지정해야 합니다. 지정된 Amazon DocumentDB 클러스터의 경우 AWS CLI 를 사용하여 클러스터 버전을 확인하거나 Amazon DocumentDB 관리 콘솔을 `describe-db-clusters` 사용하여 특정 클러스터의 엔진 버전 번호를 볼 수 있습니다.

Amazon DocumentDB 4.0은 클러스터에 대해 `r5`, `r6gt3.medium`, 및 `t4g.medium` 인스턴스 유형을 지원하며 지원되는 모든에서 사용할 수 있습니다 AWS 리전. Amazon DocumentDB 4.0 사용에 대한 추가 요금은 없습니다. 요금에 대한 자세한 내용은 Amazon DocumentDB(MongoDB 호환) 요금을 참조하십시오.

Amazon DocumentDB 4.0으로 업그레이드 또는 마이그레이션

MongoDB, mongorestore, mongoimport 및 mongoexport와 같은 AWS DMS 또는 유틸리티를 사용하여 MongoDB 3.6 또는 MongoDB 4.0에서 Amazon DocumentDB 4.0으로 마이그레이션할 수 있습니다. ??? 마찬가지로 동일한 도구를 사용하여 Amazon DocumentDB 3.6에서 Amazon DocumentDB 4.0으로 업그레이드할 수 있습니다. 마이그레이션에 대한 지침은 섹션을 참조하세요를 사용하여 Amazon DocumentDB 클러스터 업그레이드 AWS Database Migration Service.

기능적 차이

Amazon DocumentDB 3.6 및 4.0 간의 기능적 차이

Amazon DocumentDB 4.0의 릴리스에서는 Amazon DocumentDB 3.6 과 Amazon DocumentDB 4.0 간에 기능적 차이가 있습니다:

- 중첩 문서에 대한 프로젝션: Amazon DocumentDB 3.6은 프로젝션을 적용할 때 중첩된 문서의 첫 번째 필드를 고려합니다. 그러나 Amazon DocumentDB 4.0은 하위 문서를 구문 분석하고 각 하위 문서

에도 프로젝션을 적용합니다. 예를 들어"a.b.c": 1, 프로젝션이 다음과 같으면 두 버전의 동작은 동일합니다. 하지만 프로젝션이 프로젝션인 {a:{b:{c:1}}} 경우 Amazon DocumentDB 3.6은 프로젝션을 'a'에만 적용하고 'b' 또는 'c'에는 적용하지 않습니다.

- **minKey**, **maxKey**에 대한 동작: Amazon DocumentDB 4.0에서 {x:{$gt:MaxKey}}에 대한 동작은 아무것도 반환하지 않으며, {x:{$lt:MaxKey}}에 대한 동작은 모든 것을 반환합니다.
- 문서 비교 차이: 하위 문서(예: {"_id" :1, "a" :{"b":1}}의 b)에서 서로 다른 유형(double, int, long)의 수치를 비교함으로써, 이제 수치 데이터 유형과 문서의 각 수준에 걸쳐 일관된 출력이 제공됩니다.

Amazon DocumentDB 4.0 및 MongoDB 4.0 간의 기능적 차이

다음은 Amazon DocumentDB 4.0 및 MongoDB 4.0 간의 기능적 차이입니다.

- 경로에서 빈 키로 조회: 컬렉션에 배열 내에 빈 컨테이너가 있는 문서가 포함되어 있고(예:{"x" : [{ "" : 10 }, { "b" : 20 }]}) 쿼리에 사용된 키가 빈 문자열로 끝나는 경우(예:x.), Amazon DocumentDB는 배열의 모든 문서를 순회하므로 해당 문서를 반환하지만 MongoDB는 해당 문서를 반환하지 않습니다.
- 경로에서 **$setOnInsert** 및 **$** 함께: $setOnInsert 필드 연산자는 Amazon DocumentDB의 경로에서 $와 함께 작동하지 않으며, 이는 MongoDB 4.0과도 일치합니다.

기능적 차이: Amazon DocumentDB 및 MongoDB

다음은 Amazon DocumentDB(MongoDB 호환)과 MongoDB 간의 기능적 차이입니다.

주제

- Amazon DocumentDB의 기능적 이점
- 업데이트된 기능적 차이
- MongoDB와 기능적 차이

Amazon DocumentDB의 기능적 이점

암시적 트랜잭션

Amazon DocumentDB에서 모든 CRUD 문(findAndModify, update, insert, delete)은 여러 문서를 수정하는 작업의 경우에도 원자성과 일관성을 보장합니다. Amazon DocumentDB 4.0이 출시되

면서, 이제 다중 명령문 및 다중 컬렉션 작업에 대한 ACID 속성을 제공하는 명시적 트랜잭션이 지원됩니다. Amazon DocumentDB에서 트랜잭션을 사용하는 방법에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB의 트랜잭션.

다음은 Amazon DocumentDB에서 원자성 동작과 일관성 동작을 모두 충족하는 여러 문서를 수정하는 작업의 예입니다.

```
db.miles.update(
    { "credit_card": { $eq: true } },
    { $mul: { "flight_miles.$[]": NumberInt(2) } },
    { multi: true }
)
```

```
db.miles.updateMany(
    { "credit_card": { $eq: true } },
    { $mul: { "flight_miles.$[]": NumberInt(2) } }
)
```

```
db.runCommand({
  update: "miles",
  updates: [
    {
      q: { "credit_card": { $eq: true } },
      u: { $mul: { "flight_miles.$[]": NumberInt(2) } },
      multi: true
    }
  ]
})
```

```
db.products.deleteMany({
  "cost": { $gt: 30.00 }
})
```

```
db.runCommand({
  delete: "products",
  deletes: [{ q: { "cost": { $gt: 30.00 } }, limit: 0 }]
```

```
})
```

updateMany 및 deleteMany와 같은 대량 작업을 구성하는 개별 작업은 원자성이지만 전체 대량 작업은 원자성이 아닙니다. 예를 들어, 개별 삽입 작업이 오류 없이 성공적으로 실행되면 전체 insertMany 작업이 원자성입니다. insertMany 작업에 오류가 발생하면 insertMany 작업 내의 각 개별 삽입 명령문이 원자성 작업으로 실행됩니다. insertMany, updateMany 및 deleteMany 작업에 대한 ACID 속성이 필요한 경우 트랜잭션을 사용하는 것이 좋습니다.

업데이트된 기능적 차이

Amazon DocumentDB에서는 고객이 빌드하도록 요청한 기능에서 거꾸로 작업함으로써 MongoDB와의 호환성을 지속적으로 개선하고 있습니다. 이 섹션에는 고객이 더 쉽게 마이그레이션하고 애플리케이션을 빌드할 수 있도록 Amazon DocumentDB에서 제거한 기능적 차이가 포함되어 있습니다.

주제

- 배열 인덱싱
- 다중 키 인덱스
- 문자열의 null 문자
- 역할 기반 액세스 제어
- $regex 인덱싱
- 중첩된 문서에 대한 프로젝션

배열 인덱싱

2020년 4월 23일 현재 Amazon DocumentDB에서는 2,048바이트보다 큰 배열을 인덱싱하는 기능을 지원합니다. 배열의 개별 키에 대한 제한은 여전히 MongoDB와 일치하는 2,048바이트로 유지됩니다.

새 인덱스를 생성하는 경우, 향상된 기능을 활용하기 위해 아무런 작업도 필요하지 않습니다. 기존 인덱스가 있는 경우, 인덱스를 삭제한 다음, 다시 생성하여 향상된 기능을 활용할 수 있습니다. 향상된 기능을 갖춘 현재 인덱스 버전은 "v" : 3입니다.

> **Note**
>
> 프로덕션 클러스터의 경우 인덱스를 삭제하면 애플리케이션 성능에 영향을 미칠 수 있습니다. 먼저 테스트하고 프로덕션 시스템을 변경할 때 주의해서 진행합니다. 또한 인덱스를 다시 생성하는 데 걸리는 시간은 컬렉션의 전체 데이터 크기에 따라 달라집니다.

다음 명령을 사용하여 인덱스의 버전을 쿼리할 수 있습니다.

```
db.collection.getIndexes()
```

이 작업의 출력은 다음과 같이 표시됩니다. 이 출력에서 인덱스의 버전은 가장 최신 인덱스 버전인 "v" : 3입니다,

```
[
    {
        "v" : 3,
        "key" : {
        "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.test"
    }
]
```

다중 키 인덱스

2020년 4월 23일 현재 Amazon DocumentDB에서는 동일한 배열에 여러 키가 있는 복합 인덱스를 만드는 기능을 지원합니다.

새 인덱스를 생성하는 경우, 향상된 기능을 활용하기 위해 아무런 작업도 필요하지 않습니다. 기존 인덱스가 있는 경우, 인덱스를 삭제한 다음, 다시 생성하여 향상된 기능을 활용할 수 있습니다. 향상된 기능을 갖춘 현재 인덱스 버전은 "v" : 3입니다.

> **Note**
>
> 프로덕션 클러스터의 경우 인덱스를 삭제하면 애플리케이션 성능에 영향을 미칠 수 있습니다. 먼저 테스트하고 프로덕션 시스템을 변경할 때 주의해서 진행합니다. 또한 인덱스를 다시 생성하는 데 걸리는 시간은 컬렉션의 전체 데이터 크기에 따라 달라집니다.

다음 명령을 사용하여 인덱스의 버전을 쿼리할 수 있습니다.

```
db.collection.getIndexes()
```

이 작업의 출력은 다음과 같이 표시됩니다. 이 출력에서 인덱스의 버전은 가장 최신 인덱스 버전인 "v" : 3입니다,

```
[
    {
        "v" : 3,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.test"
    }
]
```

문자열의 null 문자

2020년 6월 22일부터 Amazon DocumentDB는 문자열에서 null 문자('\0')를 지원합니다.

역할 기반 액세스 제어

2020년 3월 26일부터 Amazon DocumentDB는 기본 제공 역할에 대한 RBAC(역할 기반 액세스 제어)를 지원합니다. 자세한 내용은 역할 기반 액세스 제어을 참조하세요.

$regex 인덱싱

2020년 6월 22일부터 Amazon DocumentDB는 $regex 연산자가 인덱스를 활용하는 기능을 지원합니다.

$regex 연산자와 함께 인덱스를 활용하려면 hint() 명령을 사용해야 합니다. hint()를 사용할 때 $regex를 적용할 필드의 이름을 지정해야 합니다. 예를 들어 product 필드에 인덱스 이름이 p_1인 인덱스가 있으면 db.foo.find({product: /^x.*/}).hint({product:1})은 p_1 인덱스를 사용하지만 db.foo.find({product: /^x.*/}).hint("p_1")은 인덱스를 사용하지 않습니다. explain() 명령을 사용하거나 느린 쿼리를 로깅하는 프로파일러를 사용하여 인덱스가 선택되었는지 확인할 수 있습니다. 예: db.foo.find({product: /^x.*/}).hint("p_1").explain()

> **Note**
>
> hint() 메서드는 한 번에 하나의 인덱스에만 사용할 수 있습니다.

$regex 쿼리에 대한 인덱스 사용은 접두사를 사용하고 i, m 또는 정규식 옵션을 지정하지 않는 o 정규식 쿼리에 최적화됩니다.

$regex와 함께 인덱스를 사용하는 경우 중복 값 수가 컬렉션에 있는 총 문서 수의 1% 미만인 매우 선택적인 필드에 인덱스를 생성하는 것이 좋습니다. 예를 들어 컬렉션에 100,000개의 문서가 포함되어 있는 경우, 동일한 값이 1000회 이하로 나타나는 필드에만 인덱스를 생성하세요.

중첩된 문서에 대한 프로젝션

버전 3.6의 Amazon DocumentDB와 MongoDB 간에는 $project 연산자와의 기능적 차이가 있습니다. 이 차이는 Amazon DocumentDB 4.0에서는 해결되었지만 Amazon DocumentDB 3.6에서는 지원되지 않습니다.

Amazon DocumentDB 3.6은 프로젝션을 적용할 때 중첩 문서의 첫 번째 필드만 고려하는 반면, MongoDB 3.6은 하위 문서를 구문 분석하여 각 하위 문서에도 프로젝션을 적용합니다.

예를 들어, 프로젝션이 "a.b.c": 1이면, Amazon DocumentDB와 MongoDB 모두에서 동작이 예상대로 작동합니다. 그러나 프로젝션이 {a:{b:{c:1}}}이면, Amazon DocumentDB 3.6은 프로젝션을 a에만 적용하고 b 또는 c에는 적용하지 않습니다. Amazon DocumentDB 4.0에서는 프로젝션 {a:{b:{c:1}}}, a, b, 및 c에 적용됩니다.

MongoDB와 기능적 차이

주제

- $vectorSearch 연산자
- OpCountersCommand
- 관리자 데이터베이스 및 컬렉션
- cursormaxTimeMS
- explain()
- 인덱스 빌드
- 경로에 빈 키를 사용하여 조회
- MongoDB API, 작업 및 데이터 형식
- mongodump및 mongorestore 유틸리티
- 결과 주문
- 재시도 가능한 쓰기
- 희소 인덱스
- $all 표현식 내에서 $elemMatch 사용
- 필드 이름의 달러($) 및 점(.)

- $lookup
- $natural 및 역 정렬

`$vectorSearch` 연산자

Amazon DocumentDB는 독립 연산자로 `$vectorSearch` 항목을 지원하지 않습니다. 대신 `$search` 연산자 내에서 `vectorSearch` 항목을 지원합니다. 자세한 내용은 Amazon DocumentDB에 대한 벡터 검색 섹션을 참조하세요.

`OpCountersCommand`

Amazon DocumentDB의 `OpCountersCommand` 동작은 다음과 같이 MongoDB의 `opcounters.command`에서 벗어납니다.

- MongoDB `opcounters.command` 개수는 삽입, 업데이트 및 삭제를 제외한 모든 명령을 계산하는 반면, Amazon DocumentDB의 `OpCountersCommand`는 해당 `find` 명령도 제외합니다.
- Amazon DocumentDB는 내부 명령을 `OpCountersCommand`로 계산합니다.

관리자 데이터베이스 및 컬렉션

Amazon DocumentDB는 각각 관리자 또는 로컬 데이터베이스, MongoDB `system.*` 또는 `startup_log` 컬렉션을 지원하지 않습니다.

`cursormaxTimeMS`

Amazon DocumentDB에서 `cursor.maxTimeMS`은 각 `getMore` 요청에 대해 카운터를 재설정합니다. 따라서 3000MS `maxTimeMS`을 지정하면 쿼리에 2800MS가 걸리고 이후의 각 `getMore` 요청에는 300MS가 걸리므로, 커서가 타임아웃되지 않습니다. 쿼리 또는 개별 `getMore` 요청 중 단일 작업에 지정된 `maxTimeMS`보다 많은 시간이 걸리는 경우에만 커서가 시간 초과됩니다. 또한, 커서 실행 시간을 확인하는 스위퍼는 5분 단위로 실행됩니다.

explain()

Amazon DocumentDB는 분산된 내결함성 자가 복구 스토리지 시스템을 활용하는 목적별 데이터베이스 엔진에서 MongoDB 3.6, 4.0, 5.0 및 8.0 APIs를 에뮬레이션합니다. 그 결과, 쿼리 계획과 `explain()`의 출력은 Amazon DocumentDB와 MongoDB 간에 다를 수 있습니다. 쿼리 계획을 제어하려는 고객은 `$hint` 연산자를 사용하여 기본 인덱스를 선택할 수 있습니다.

인덱스 빌드

Amazon DocumentDB는 특정 시간에 모음에서 발생하는 배경 인덱스 빌드를 하나만 허용합니다. 포그라운드 또는 백그라운드 둘 다에서. 인덱스 빌드가 현재 진행 중일 때 같은 컬렉션에서 `createIndex()` 또는 `dropIndex()`와 같은 작업이 발생하면 새로 시도한 작업이 실패합니다.

기본적으로 Amazon DocumentDB 및 MongoDB 버전 4.0의 인덱스 빌드는 포그라운드에서 실행됩니다. MongoDB 버전 4.2 이상에서는 createIndexes 또는 해당 쉘 도우미 `createIndex()` 및 `createIndexes()`에 지정된 경우, 백그라운드 인덱스 빌드 옵션을 무시합니다.

TTL(Time to Live) 인덱스는 배경 인덱스 빌드가 완료된 이후에 문서 만료를 시작합니다.

경로에 빈 키를 사용하여 조회

경로의 일부로 빈 문자열이 포함된 키(예: `x.`, `x..b`)를 찾고, 객체의 배열 내에 빈 문자열 키 경로가 있는 경우(예: `{"x" : [ { "" : 10 }, { "b" : 20 } ]}`), Amazon DocumentDB는 MongoDB에서 동일한 검색을 실행한 경우와 다른 결과를 반환합니다.

MongoDB에서, 배열 내 빈 키 경로 조회는 빈 문자열 키가 경로 조회의 끝에 있지 않을 때 예상대로 작동합니다. 하지만 경로 조회 끝에 빈 문자열 키가 있으면 배열을 들여다보지 않습니다.

하지만 Amazon DocumentDB에서는 `getArrayIndexFromKeyString`은 빈 문자열을 `0`로 변환하므로 배열 내의 첫 번째 요소만 읽습니다. 따라서 문자열 키 조회는 배열 인덱스 검색으로 취급됩니다.

MongoDB API, 작업 및 데이터 형식

Amazon DocumentDB는 MongoDB 3.6, 4.0, 5.0 및 8.0 APIs. 지원되는 기능의 최신 목록을 보려면 Amazon DocumentDB에서 지원되는 MongoDB API, 작업 및 데이터 형식 섹션을 참조하세요.

`mongodump`및 `mongorestore` 유틸리티

Amazon DocumentDB는 관리자 데이터베이스를 지원하지 않으며, 따라서 mongodump 또는 `mongorestore` 유틸리티를 사용할 때 관리자 데이터베이스를 덤프하거나 복원하지 않습니다. Amazon DocumentDB에서 `mongorestore`을 사용하여 새 데이터베이스를 생성할 경우 복원 작업 이외에 사용자 역할을 다시 생성해야 합니다.

> **MongoDB 데이터베이스 도구 버전 요구 사항**
>
> Amazon DocumentDB용 버전 100.11.0까지 MongoDB 데이터베이스 도구를 사용합니다. Amazon DocumentDB 다운로드하려면 MongoDB 웹 사이트의 MongoDB 데이터베이스 도구 릴리스를 참조하세요. MongoDB

결과 주문

Amazon DocumentDB는 결과 집합의 암시적 결과 정렬 순서를 보장하지 않습니다. 결과 집합의 순서를 보장하려면 `sort()`를 사용하여 정렬 순서를 명시적으로 지정합니다.

다음 예에서는 재고 수집의 항목을 재고 필드를 기준으로 내림차순으로 정렬합니다.

```
db.inventory.find().sort({ stock: -1 })
```

`$sort` 집계 단계를 사용할 경우 해당 단계가 집계 파이프라인의 마지막 `$sort` 단계가 아니면 정렬 순서가 보존되지 않습니다. `$sort` 집계 단계를 `$group`집계 단계와 함께 사용하는 경우 `$sort` 집계 단계는 `$first` 및 `$last` 누적기에만 적용됩니다. Amazon DocumentDB 4.0에서는 이전 `$sort` 단계의 정렬 순서를 준수하기 위한 지원이 `$push`에 추가되었습니다.

재시도 가능한 쓰기

대부분의 현재 MongoDB 호환 드라이버는 기본적으로 재시도 가능한 쓰기를 활성화합니다. Amazon DocumentDB는 엔진 버전 8.0.2부터 재시도 가능한 쓰기를 지원합니다. 이전 엔진 버전에서는 재시도 가능한 쓰기가 지원되지 않으므로 오류를 방지하려면 해당 쓰기를 비활성화해야 합니다.

8.0.2 이전 엔진 버전에서 재시도 가능한 쓰기를 활성화하면 다음과 유사한 오류 메시지가 표시됩니다.

```
{"ok":0,"errmsg":"Retryable writes are not supported","code":301,"name":"MongoError"}
```

재시도 가능한 쓰기는 연결 문자열(예: `MongoClient("mongodb://my.mongodb.cluster/db?retryWrites=false")`) 또는 MongoClient 생성자의 키워드 인수(예: `MongoClient("mongodb://my.mongodb.cluster/db", retryWrites=False)`)를 통해 비활성화할 수 있습니다.

엔진 버전 8.0.2 이상에서 재시도 가능한 쓰기에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB에서 재시도 가능한 쓰기.

희소 인덱스

쿼리에서 생성된 희소 인덱스를 사용하려면 인덱스를 포함하는 필드에서 `$exists` 절을 사용해야 합니다. `$exists`를 생략하면 Amazon DocumentDB는 희소 인덱스를 사용하지 않습니다.

다음은 예입니다.

```
db.inventory.count({ "stock": { $exists: true }})
```

스파스 다중 키 인덱스의 경우 Amazon DocumentDB는 문서 조회 결과로 값 집합이 생성되고 인덱싱된 필드의 하위 집합만 누락된 경우 고유 키 제약 조건을 지원하지 않습니다. 예를 들어, `"a.c"`가 인덱스에 저장되므로 `"a" : [ { "b" : 2 }, { "c" : 1 } ]`의 입력이 주어지면 `createIndex({"a.b" : 1 }, { unique : true, sparse :true })`는 지원되지 않습니다.

$all 표현식 내에서 **$elemMatch** 사용

Amazon DocumentDB는 현재 `$all` 표현식 내에서 `$elemMatch` 연산자 사용을 지원하지 않습니다. 차선책으로 다음과 같이 `$elemMatch`와 함께 `$and` 연산자를 사용할 수 있습니다.

원래 작업:

```
db.col.find({
  qty: {
    $all: [
      { "$elemMatch": { part: "xyz", qty: { $lt: 11 } } },
      { "$elemMatch": { num: 40, size: "XL" } }
    ]
  }
})
```

업데이트된 작업:

```
db.col.find({
  $and: [
    { qty: { "$elemMatch": { part: "xyz", qty: { $lt: 11 } } } },
    { qty: { "$elemMatch": { qty: 40, size: "XL" } } }
  ]
})
```

필드 이름의 달러($) 및 점(.)

Amazon DocumentDB는 중첩 객체에서 $in, $nin 및 $all의 달러($) 접두사 필드 쿼리를 지원하지 않습니다. 예를 들어 다음 쿼리는 Amazon DocumentDB에서 유효하지 않습니다.

```
coll.find({"field": {"$all": [{ "$a": 1 }]}})
```

$lookup

Amazon DocumentDB는 동등 일치(예: 왼쪽 외부 조인)를 수행할 수 있는 기능을 지원하고 상관 관계가 없는 하위 쿼리도 지원합니다. 버전 8.0.2부터 Amazon DocumentDB는 상관관계가 있는 간결한 하위 쿼리 구문을 포함하여 상관관계가 있는 하위 쿼리도 지원합니다. 상관관계가 있는 하위 쿼리에서는 표현식을 평가하는 집계 단계의 하위 집합만 `let` 변수를 참조할 수 있습니다. 지원되는 단계 및 관련 제한 목록은 섹션을 참조하세요상관관계가 있는 하위 쿼리에서 지원되는 단계. 8.0.2 이전 버전에서는 상관관계가 있는 하위 쿼리가 지원되지 않습니다.

$lookup과 인덱스를 활용하면

이제 `$lookup` 스테이지 연산자를 사용하여 인덱스를 활용할 수 있습니다. 사용 사례에 따라 성능을 최적화하는 데 사용할 수 있는 여러 인덱싱 알고리즘이 있습니다. 이 섹션에서는 `$lookup`을 위한 다양한 인덱싱 알고리즘을 설명하고 워크로드에 가장 적합한 알고리즘을 선택하는 데 도움이 됩니다.

기본적으로 Amazon DocumentDB는 `allowDiskUse:false`을 사용할 때는 해시 알고리즘을 활용하고, `allowDiskUse:true`를 사용할 때는 정렬 병합을 사용합니다.

> **Note**
>
> 이 `allowDiskUse` 옵션은 현재 `find` 명령에 지원되지 않습니다. 옵션은 집계의 일부로만 지원됩니다. 와 함께 집계 프레임워크`allowDiskUse:true`를 사용하여 메모리 제한을 초과할 수 있는 대규모 쿼리를 처리합니다.

일부 사용 사례의 경우 쿼리 최적화 프로그램이 다른 알고리즘을 사용하도록 강제하는 것이 바람직할 수 있습니다. `$lookup` 집계 연산자가 이용할 수 있는 다양한 인덱싱 알고리즘은 다음과 같습니다.

- 중첩 루프: 외부 컬렉션이 1GB 미만이고 외부 컬렉션의 필드에 인덱스가 있는 경우, 일반적으로 중첩 루프 계획이 워크로드에 유용합니다. 중첩 루프 알고리즘을 사용하는 경우 계획 설명에는 단계가 `NESTED_LOOP_LOOKUP`로 표시됩니다.
- 정렬 병합: 외부 컬렉션에 조회에 사용되는 필드에 대한 색인이 없고 작업 데이터셋이 메모리에 맞지 않는 경우 일반적으로 정렬 병합 계획이 워크로드에 유용합니다. 정렬 병합 알고리즘을 사용하는 경우 설명 계획에는 단계가 `SORT_LOOKUP`로 표시됩니다.
- 해시: 일반적으로 해시 계획은 외부 컬렉션이 1GB 미만이고 작업 데이터셋이 메모리에 들어갈 수 있는 경우 워크로드에 유용합니다. 해시 알고리즘을 사용하는 경우 설명 계획에는 단계가 `HASH_LOOKUP`로 표시됩니다.

쿼리에서 explain을 사용하여 $lookup 연산자에 사용되는 인덱싱 알고리즘을 식별할 수 있습니다. 아래는 하나의 예제입니다.

```
db.localCollection.explain().aggregate(
   [
      {
         $lookup:
            {
               from: "foreignCollection",
               localField: "a",
               foreignField: "b",
               as: "joined"
            }
      }
   ]
)

output
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "test.localCollection",
        "winningPlan" : {
            "stage" : "SUBSCAN",
            "inputStage" : {
                "stage" : "SORT_AGGREGATE",
                "inputStage" : {
                    "stage" : "SORT",
                    "inputStage" : {
                        "stage" : "NESTED_LOOP_LOOKUP",
                        "inputStages" : [
                            {
                                "stage" : "COLLSCAN"
                            },
                            {
                                "stage" : "FETCH",
                                "inputStage" : {
                                    "stage" : "COLLSCAN"
                                }
                            }
                        ]
                    }
```

```
                }
            }
        }
    },
    "serverInfo" : {
        "host" : "devbox-test",
        "port" : 27317,
        "version" : "3.6.0"
    },
    "ok" : 1
}
```

explain() 메서드를 사용하는 대신 프로파일러를 사용하여 $lookup 연산자 사용 시 사용 중인 알고리즘을 검토할 수 있습니다. 프로파일러에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB 작업 프로파일링.

planHint 사용

쿼리 최적화 프로그램에서 $lookup에 다른 인덱싱 알고리즘을 사용하도록 하려면 planHint를 사용할 수 있습니다. 이렇게 하려면 집계 단계 옵션의 설명을 사용하여 다른 계획을 적용하세요. 다음은 주석 구문의 예시입니다.

```
comment : {
    comment :  "<string>",
    lookupStage : { planHint : "SORT" | "HASH" | "NESTED_LOOP" }
}
```

다음은 planHint을 사용하여 쿼리 최적화 프로그램이 HASH 인덱싱 알고리즘을 사용하도록 강제하는 예입니다.

```
db.foo.aggregate(
   [
      {
         $lookup:
            {
               from: "foo",
               localField: "_id",
               foreignField: "_id",
               as: "joined"
            },
```

```
      }
   ]
),
{
   comment : "{ \"lookupStage\" : { \"planHint\": \"HASH\" }}"
```

워크로드에 가장 적합한 알고리즘을 테스트하려면 explain 메서드의 executionStats 파라미터를 사용하여 $lookup 스테이지의 실행 시간을 측정하고 인덱싱 알고리즘을 수정할 수 있습니다(즉, HASH/SORT/NESTED_LOOP).

다음 예제는 SORT 알고리즘을 사용하여 $lookup 스테이지의 실행 시간을 측정하는 데 executionStats을 사용하는 방법을 보여줍니다.

```
db.foo.explain("executionStats").aggregate(
   [
      {
         $lookup:
            {
               from: "foo",
               localField: "_id",
               foreignField: "_id",
               as: "joined"
            },
      }
   ]
),
{
   comment : "{ \"lookupStage\" : { \"planHint\": \"SORT\" }}"
```

$natural 및 역 정렬

Amazon DocumentDB는 포워드 컬렉션 스캔에만 $natural을 지원합니다. 역수집 스캔({$natural: -1})은 MongoServerError로 이어집니다.

Amazon DocumentDB에서 지원되는 MongoDB API, 작업 및 데이터 형식

Amazon DocumentDB(MongoDB 호환)은 MongoDB 워크로드를 지원하는 빠르고, 확장 가능하며, 가용성이 높은 완전관리형 문서 데이터베이스 서비스입니다. Amazon DocumentDB는 MongoDB 3.6,

4.0, 5.0 및 8.0 APIs. 이 단원에서는 지원되는 기능에 대해 설명합니다. MongoDB APIs 및 드라이버 사용에 대한 지원은 MongoDB 커뮤니티 포럼을 참조하세요. Amazon DocumentDB 서비스 사용에 대한 지원은 해당 AWS 지원 팀에 문의하십시오. Amazon DocumentDB와 MongoDB의 기능적 차이점은 섹션을 참조하세요기능적 차이: Amazon DocumentDB 및 MongoDB.

내부 전용이거나 완전 관리형 서비스에 해당되지 않는 MongoDB 명령 및 연산자는 지원되지 않으며, 지원 기능 목록에도 포함되지 않습니다.

출시 이후 50개 이상의 기능을 더 추가했으며 고객이 원하는 기능을 제공하기 위해 앞으로도 계속 노력할 것입니다. 최신 출시에 대한 자세한 내용은 Amazon DocumentDB 공지 사항을 참조하십시오.

지원되지 않는 기능 중 구축을 원하는 기능이 있는 경우, 계정 ID, 요청된 기능, 사용 사례를 포함한 이메일을 Amazon DocumentDB 서비스 팀에 보내 알려주십시오.

주제

- 데이터베이스 명령
- 쿼리 및 프로젝션 연산자
- 업데이트 연산자
- 지리 공간
- 커서 메서드
- 집계 파이프라인 연산자
- 데이터 형식
- 인덱스

데이터베이스 명령

주제

- 관리 명령
- 집계
- 인증
- 진단 명령
- 쿼리 및 쓰기 작업
- 역할 관리 명령
- 세션 명령

- 사용자 관리
- 샤딩 명령

관리 명령

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
제한 컬렉션	아니요	아니요	아니요	아니요	아니요
cloneCollectionAsCapped	아니요	아니요	아니요	아니요	아니요
collMod	부분적	부분적	부분적	부분적	부분적
collMod: expireAfterSeconds	예	예	예	예	예
collMod: 숨김	아니요	아니요	예(5.0.2 이상)	예(8.0.1 이상)	아니요
convertToCapped	아니요	아니요	아니요	아니요	아니요
copydb	아니요	아니요	아니요	아니요	아니요
create	예	예	예	예	예
createView	아니요	아니요	아니요	예	아니요
createIndexes	예	예	예	예	예
currentOp	예	예	예	예	예
drop	예	예	예	예	예

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
dropDatabase	예	예	예	예	예
dropIndexes	예	예	예	예	예
filemd5	아니요	아니요	아니요	아니요	아니요
getAuditConfig	아니요	예	예	예	아니요
killCursors	예	예	예	예	예
killOp	예	예	예	예	예
listCollections*	예	예	예	예	예
listDatabases	예	예	예	예	예
listIndexes	예	예	예	예	예
reIndex	아니요	아니요	예	예	아니요
renameCollection	예	예	예	예	아니요
setAuditConfig	아니요	예	예	예	아니요

* 필터 옵션의 `type` 키는 지원되지 않습니다.

집계

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
aggregate	예	예	예	예	예
count	예	예	예	예	예
distinct	예	예	예	예	예
mapReduce	아니요	아니요	아니요	예	아니요

인증

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
authenticate	예	예	예	예	예
로그아웃	예	예	예	예	예

진단 명령

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
buildInfo	예	예	예	예	예
collStats	예	예	예	예	예
connPoolStats	아니요	아니요	아니요	아니요	아니요
connectionStatus	예	예	예	예	예

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
dataSize	예	예	예	예	예
dbHash	아니요	아니요	아니요	아니요	아니요
dbStats	예	예	예	예	예
explain	예	예	예	예	예
explain: executionStats	예	예	예	예	예
기능	아니요	아니요	아니요	아니요	아니요
hostInfo	예	예	예	예	예
listCommands	예	예	예	예	예
planCacheClearFilters	아니요	아니요	예	예	아니요
planCacheListFilters	아니요	아니요	예	예	아니요
planCacheSetFilter	아니요	아니요	예	예	아니요
profiler	예	예	예	예	아니요
serverStatus	예	예	예	예	예
top	예	예	예	예	예

쿼리 및 쓰기 작업

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
Change streams	예	예	예	예	아니요
delete	예	예	예	예	예
find	예	예	예	예	예
findAndModify	예	예	예	예	예
getLastError	아니요	아니요	아니요	아니요	아니요
getMore	예	예	예	예	예
getPrevError	아니요	아니요	아니요	아니요	아니요
GridFS	예	예	예	예	아니요
삽입	예	예	예	예	예
parallelCollectionScan	아니요	아니요	아니요	아니요	아니요
resetError	아니요	아니요	아니요	아니요	아니요
업데이트	예	예	예	예	예
ReplaceOne	예	예	예	예	예

역할 관리 명령

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
createRole	예	예	예	예	아니요
createUser	예	예	예	예	아니요
dropAllRoles	예	예	예	예	아니요
dropAllUsers	예	예	예	예	아니요
dropRole	예	예	예	예	아니요
dropUser	예	예	예	예	아니요
grantRole	예	예	예	예	아니요
revokeRole	예	예	예	예	아니요
updateRole	예	예	예	예	아니요
updateUser	예	예	예	예	아니요

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
abortTransaction	아니요	예	예	예	아니요
commitTransaction	아니요	예	예	예	아니요
endSessions	아니요	아니요	아니요	아니요	아니요
killAllSessions	아니요	예	예	예	아니요
killAllSessionsByPattern	아니요	아니요	아니요	아니요	아니요
killSessions	아니요	예	예	예	아니요
refreshSessions	아니요	아니요	아니요	아니요	아니요

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
startSession	아니요	예	예	예	아니요

사용자 관리

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
createUser	예	예	예	예	예
dropAllUsersFromDatabase	예	예	예	예	예
dropUser	예	예	예	예	예
grantRolesToUser	예	예	예	예	예
revokeRolesFromUser	예	예	예	예	예
updateUser	예	예	예	예	예
usersInfo	예	예	예	예	예

샤딩 명령

명령	엘라스틱 클러스터
abortReshardCollection	아니요
addShard	아니요
addShardToZone	아니요

명령	엘라스틱 클러스터
balancerCollectionStatus	아니요
balancerStart	아니요
balancerStatus	아니요
balancerStop	아니요
checkShardingIndex	아니요
clearJumboFlag	아니요
cleanupOrphaned	아니요
cleanupReshardCollection	아니요
commitReshardCollection	아니요
enableSharding	예
flushRouterConfig	아니요
getShardMap	아니요
getShardVersion	아니요
isdbgrid	아니요
listShards	아니요
medianKey	아니요
moveChunk	아니요
movePrimary	아니요
mergeChunks	아니요
refineCollectionShardKey	아니요

명령	엘라스틱 클러스터
removeShard	아니요
removeShardFromZone	아니요
reshardCollection	아니요
setAllowMigrations	아니요
setShardVersion	아니요
shardCollection	예
shardingState	아니요
split	아니요
splitVector	아니요
unsetSharding	아니요
updateZoneKeyRange	아니요

쿼리 및 프로젝션 연산자

주제

- 배열 연산자
- Bitwise 연산자
- 설명 연산자
- 비교 연산자
- 요소 연산자
- 평가 쿼리 연산자
- 논리 연산자
- 프로젝션 연산자

배열 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$all	예	예	예	예	예
$elemMatch	예	예	예	예	예
$size	예	예	예	예	예

Bitwise 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$bitsAllSet	예	예	예	예	예
$bitsAnySet	예	예	예	예	예
$bitsAllClear	예	예	예	예	예
$bitsAnyClear	예	예	예	예	예

설명 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$comment	예	예	예	예	예

비교 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$eq	예	예	예	예	예
$gt	예	예	예	예	예
$gte	예	예	예	예	예
$in	예	예	예	예	예
$lt	예	예	예	예	예
$lte	예	예	예	예	예
$ne	예	예	예	예	예
$nin	예	예	예	예	예

요소 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$exists	예	예	예	예	예
$type	예	예	예	예	예

평가 쿼리 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$expr	아니요	예	예	예	아니요

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$jsonSchema	아니요	예	예	예	아니요
$mod	예	예	예	예	예
$regex	예	예	예	예	예
$text	아니요	아니요	예	예	아니요
$where	아니요	아니요	아니요	아니요	아니요

논리 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$and	예	예	예	예	예
$nor	예	예	예	예	예
$not	예	예	예	예	예
$or	예	예	예	예	예

프로젝션 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$	예	예	예	예	예
$elemMatch	예	예	예	예	예
$meta	아니요	아니요	예	예	아니요
$slice	예	예	예	예	예

업데이트 연산자

주제

- 배열 연산자
- Bitwise 연산자
- 필드 연산자
- 업데이트 한정자

배열 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$	예	예	예	예	예
$[]	예	예	예	예	예
$[<identifier>]	예	예	예	예	예
$addToSet	예	예	예	예	예
$pop	예	예	예	예	예
$pullAll	예	예	예	예	예
$pull	예	예	예	예	예
$push	예	예	예	예	예

Bitwise 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$bit	예	예	예	예	예

필드 연산자

연산자	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$currentDate	예	예	예	예	예
$inc	예	예	예	예	예
$max	예	예	예	예	예
$min	예	예	예	예	예
$mul	예	예	예	예	예
$rename	예	예	예	예	예
$set	예	예	예	예	예
$setOnInsert	예	예	예	예	예
$unset	예	예	예	예	예

업데이트 한정자

연산자	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$each	예	예	예	예	예
$position	예	예	예	예	예
$slice	예	예	예	예	예
$sort	예	예	예	예	예

지리 공간

지오메트리 지정자

쿼리 선택기	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$box	아니요	아니요	아니요	아니요	아니요
$center	아니요	아니요	아니요	아니요	아니요
$centerSphere	아니요	아니요	아니요	아니요	아니요
$geometry	예	예	예	예	예
$maxDistance	예	예	예	예	예
$minDistance	예	예	예	예	예
$nearSphere	예	예	예	예	예
$polygon	아니요	아니요	아니요	아니요	아니요
$uniqueDocs	아니요	아니요	아니요	아니요	아니요

쿼리 선택기

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$geoIntersects	예	예	예	예	예
$geoWithin	예	예	예	예	예
$near	예	예	예	예	예

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$nearSphere	예	예	예	예	예
$polygon	아니요	아니요	아니요	아니요	아니요
$uniqueDocs	아니요	아니요	아니요	아니요	아니요

커서 메서드

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
cursor.batchSize()	예	예	예	예	예
cursor.close()	예	예	예	예	예
cursor.collation()	아니요	아니요	아니요	예	아니요
cursor.comment()	예	예	예	예	예
cursor.count()	예	예	예	예	예
cursor.explain()	예	예	예	예	아니요
cursor.forEach()	예	예	예	예	예
cursor.hasNext()	예	예	예	예	예
cursor.hint()	예	예	예	예	예*

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
cursor.isClosed()	예	예	예	예	예
cursor.isExhausted()	예	예	예	예	아니요
cursor.itcount()	예	예	예	예	아니요
cursor.limit()	예	예	예	예	아니요
cursor.map()	예	예	예	예	아니요
cursor.max()	아니요	아니요	예(5.0.2 이상)	예(8.0.1 이상)	아니요
cursor.maxScan()	예	예	예	예	아니요
cursor.maxTimeMS()	예	예	예	예	아니요
cursor.min()	아니요	아니요	예(5.0.2 이상)	예(8.0.1 이상)	아니요
cursor.next()	예	예	예	예	예
cursor.noCursorTimeout()	아니요	아니요	아니요	아니요	아니요
cursor.objsLeftInBatch()	예	예	예	예	아니요
cursor.pretty()	예	예	예	예	아니요

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
cursor.readConcern()	예	예	예	예	아니요
cursor.readPref()	예	예	예	예	아니요
cursor.returnKey()	아니요	아니요	아니요	아니요	아니요
cursor.showRecordId()	아니요	아니요	아니요	아니요	아니요
cursor.size()	예	예	예	예	아니요
cursor.skip()	예	예	예	예	아니요
cursor.sort()	예	예	예	예	아니요
cursor.tailable()	아니요	아니요	아니요	아니요	아니요
cursor.toArray()	예	예	예	예	아니요

* `hint` 인덱스는 인덱스 표현식과 함께 지원됩니다. 예: `db.foo.find().hint({x:1})`

`cursor.min()` 및 `cursor.max()` 메서드에는 사용할 인덱스의 이름을 `hint` 지정하는가 필요합니다. 두 방법 모두 동일한 필드를 지정해야 하며 정렬과 결합할 수 없습니다`$natural`. 명령`find`에서만 지원됩니다.

집계 파이프라인 연산자

주제

- 누적기 식
- 산술 연산자

- 배열 연산자
- 부울 연산
- 비교 연산자
- 조건식 연산자
- 데이터 유형 연산자
- 데이터 크기 연산자
- 날짜 연산자
- 리터럴 연산자
- 병합 연산자
- 자연 연산자
- 집합 연산자
- 단계 연산자
- 문자열 연산자
- 시스템 변수
- 텍스트 검색 연산자
- 유형 변환 연산자
- 변수 연산자
- 삼각법 연산자
- 비트 연산자(집계)
- 타임스탬프 연산자
- 기타 연산자

누적기 식

표현식	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$accumulator	-	-	아니요	아니요	아니요
$addToSet	예	예	예	예	예
$avg	예	예	예	예	예

표현식	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$하위	-	-	-	예(8.0.1 이상)	아니요
$bottomN	-	-	-	예(8.0.1 이상)	아니요
$count	-	-	아니요	예(8.0.1 이상)	아니요
$covarianCePop	아니요	아니요	아니요	예(8.0.2 이상)	아니요
$covarianCeSamp	아니요	아니요	아니요	예(8.0.2 이상)	아니요
$denseRank	아니요	아니요	아니요	예(8.0.2 이상)	아니요
$derivative	아니요	아니요	아니요	아니요	아니요
$documentNumber	아니요	아니요	아니요	예(8.0.2 이상)	아니요
$expMovingAvg	아니요	아니요	아니요	아니요	아니요
$first	예	예	예	예	예
$firstN	-	-	-	예(8.0.1 이상)	아니요
$integral	아니요	아니요	아니요	아니요	아니요
$last	예	예	예	예	예
$lastN	-	-	-	예(8.0.1 이상)	아니요
$linearFill	아니요	아니요	아니요	아니요	아니요
$locf	아니요	아니요	아니요	아니요	아니요
$max	예	예	예	예	예

표현식	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$maxN	-	-	-	예(8.0.1 이상)	아니요
$median	-	-	-	예(8.0.1 이상)	아니요
$min	예	예	예	예	예
$minN	-	-	-	예(8.0.1 이상)	아니요
$백분위수	-	-	-	예(8.0.1 이상)	아니요
$push	예	예	예	예	예
$rank	아니요	아니요	아니요	예(8.0.2 이상)	아니요
$shift	아니요	아니요	아니요	아니요	아니요
$stdDevPop	아니요	아니요	아니요	예(8.0.1 이상)	아니요
$stdDevSamp	아니요	아니요	아니요	예(8.0.1 이상)	아니요
$sum	예	예	예	예	예
$상위	-	-	-	예(8.0.1 이상)	아니요
$topN	-	-	-	예(8.0.1 이상)	아니요

산술 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$abs	예	예	예	예	예
$add	예	예	예	예	예
$ceil	아니요	예	예	예	예

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$divide	예	예	예	예	예
$exp	아니요	예	예	예	예
$floor	아니요	예	예	예	예
$ln	아니요	예	예	예	예
$log	아니요	예	예	예	예
$log10	아니요	예	예	예	예
$mod	예	예	예	예	예
$multiply	예	예	예	예	예
$pow	아니요	아니요	예(5.0.1 이상)	예	아니요
$round	-	-	아니요	예(8.0.1 이상)	아니요
$sqrt	아니요	예	예	예	예
$subtract	예	예	예	예	예
$trunc	아니요	아니요	아니요	예(8.0.1 이상)	아니요

배열 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$arrayElemAt	예	예	예	예	예
$arrayToObject	예	예	예	예	예

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$concatArrays	예	예	예	예	예
$filter	예	예	예	예	예
$first	-	-	예	예	아니요
$in	예	예	예	예	예
$indexOfArray	예	예	예	예	예
$isArray	예	예	예	예	예
$last	-	-	예	예	아니요
$objectToArray	예	예	예	예	예
$range	예	예	예	예	예
$reverseArray	예	예	예	예	예
$sortArray	아니요	아니요	아니요	예	아니요
$reduce	예	예	예	예	예
$size	예	예	예	예	예
$slice	예	예	예	예	예
$zip	예	예	예	예	예

부울 연산

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$and	예	예	예	예	예
$not	예	예	예	예	예
$or	예	예	예	예	예

비교 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$cmp	예	예	예	예	예
$eq	예	예	예	예	예
$gt	예	예	예	예	예
$gte	예	예	예	예	예
$lt	예	예	예	예	예
$lte	예	예	예	예	예
$ne	예	예	예	예	예

조건식 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$cond	예	예	예	예	예

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$ifNull	예	예	예	예	예
$switch	아니요	예	예	예	아니요

데이터 유형 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$type	예	예	예	예	예

데이터 크기 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$binarySize	-	-	아니요	예(8.0.1 이상)	아니요
$bsonSize	-	-	아니요	예(8.0.1 이상)	아니요

날짜 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$dateAdd	아니요	아니요	예	예	예
$dateDiff	-	-	예	예	아니요
$dateFromParts	아니요	아니요	예(5.0.1 이상)	아니요	아니요

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$dateFromString	예	예	예	예	예
$dateSubtract	아니요	아니요	예	예	예
$dateToParts	아니요	아니요	예(5.0.1 이상)	아니요	아니요
$dateToString	예	예	예	예	예
$dateTrunc	-	-	아니요	예	아니요
$dayOfMonth	예	예	예	예	예
$dayOfWeek	예	예	예	예	예
$dayOfYear	예	예	예	예	예
$hour	예	예	예	예	예
$isoDayOfWeek	예	예	예	예	예
$isoWeek	예	예	예	예	예
$isoWeekYear	예	예	예	예	예
$millisecond	예	예	예	예	예
$minute	예	예	예	예	예
$month	예	예	예	예	예
$second	예	예	예	예	예
$week	예	예	예	예	예

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$year	예	예	예	예	예

리터럴 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$literal	예	예	예	예	예

병합 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$mergeObjects	예	예	예	예	예

자연 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$natural	예	예	예	예	예

집합 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$allElementsTrue	아니요	예	예	예	예
$anyElementTrue	아니요	예	예	예	예
$setDifference	아니요	예	예	예	예
$setEquals	예	예	예	예	예
$setIntersection	예	예	예	예	예
$setIsSubset	예	예	예	예	예
$setUnion	예	예	예	예	예

단계 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$addFields	예	예	예	예	예
$bucket	아니요	아니요	아니요	예	아니요
$bucketAuto	아니요	아니요	아니요	예(8.0.2 이상)	아니요
$changeStream	예	예	예	예	아니요

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$changeStreamSplitLargeEvent	아니요	아니요	아니요	예(8.0.2 이상)	아니요
$collStats	아니요	예	예	예	아니요
$count	예	예	예	예	예
$currentOp	예	예	예	예	예
$densify	아니요	아니요	아니요	아니요	아니요
$facet	아니요	아니요	아니요	예(8.0.2 이상)	아니요
$fill	아니요	아니요	아니요	아니요	아니요
$geoNear	예	예	예	예	예
$graphLookup	아니요	아니요	아니요	예	아니요
$group	예	예	예	예	예
$indexStats	예	예	예	예	아니요
$limit	예	예	예	예	예
$listLocalSessions	아니요	아니요	아니요	아니요	아니요
$listSessions	아니요	아니요	아니요	아니요	아니요
$lookup	예	예	예	예	예
$match	예	예	예	예	예
$merge	-	-	아니요	예	아니요

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$out	예	예	예	예	아니요
$planCacheStats	-	-	아니요	아니요	아니요
$project	예	예	예	예	예
$redact	예	예	예	예	예
$replaceRoot	예	예	예	예	예
$sample	예	예	예	예	예
$set	-	-	아니요	예	아니요
$setWindowFields	-	-	아니요	예(8.0.2 이상)	아니요
$skip	예	예	예	예	예
$sort	예	예	예	예	예
$sortByCount	아니요	아니요	아니요	예(8.0.1 이상)	아니요
$unionWith	-	-	아니요	아니요	아니요
$unset	-	-	아니요	예	아니요
$unwind	예	예	예	예	예
$replaceWith	아니요	아니요	아니요	예	아니요
$vectorSearch	아니요	아니요	아니요	예	아니요
$listSearchIndexes	-	-	-	예(8.0.1 이상)	아니요

문자열 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$concat	예	예	예	예	예
$indexOfBytes	예	예	예	예	예
$indexOfCP	예	예	예	예	예
$ltrim	아니요	예	예	예	아니요
$regexFind	-	-	예	예	아니요
$regexFindAll	-	-	예	예	아니요
$regexMatch	-	-	예	예	아니요
$replaceAll	-	-	예	예	아니요
$replaceOne	-	-	예	예	아니요
$rtrim	아니요	예	예	예	아니요
$split	예	예	예	예	예
$strcasecmp	예	예	예	예	예
$strLenBytes	예	예	예	예	예
$strLenCP	예	예	예	예	예
$substr	예	예	예	예	예
$substrBytes	예	예	예	예	예
$substrCP	예	예	예	예	예
$toLower	예	예	예	예	예

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$toUpper	예	예	예	예	예
$trim	아니요	예	예	예	아니요

시스템 변수

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$$CURRENT	아니요	아니요	아니요	아니요	아니요
$$DESCEND	예	예	예	예	예
$$KEEP	예	예	예	예	예
$$PRUNE	예	예	예	예	예
$$REMOVE	아니요	아니요	아니요	아니요	아니요
$ROOT	예	예	예	예	예

텍스트 검색 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$meta	아니요	아니요	예	예	아니요
$search	아니요	아니요	예	예	아니요

유형 변환 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$convert	아니요	예	예	예	예
$isNumber	-	-	아니요	예(8.0.1 이상)	아니요
$toBool	아니요	예	예	예	예
$toDate	아니요	예	예	예	예
$toDecimal	아니요	예	예	예	예
$toDouble	아니요	예	예	예	예
$toInt	아니요	예	예	예	예
$toLong	아니요	예	예	예	예
$toObjectId	아니요	예	예	예	예
$toString	아니요	예	예	예	예
$toUUID	-	-	-	예(8.0.1 이상)	아니요

변수 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$let	예	예	예	예	예
$map	예	예	예	예	예

삼각법 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$acos	-	-	-	예(8.0.1 이상)	아니요
$acosh	-	-	-	예(8.0.1 이상)	아니요
$asin	-	-	-	예(8.0.1 이상)	아니요
$asinh	-	-	-	예(8.0.1 이상)	아니요
$atan	-	-	-	예(8.0.1 이상)	아니요
$atan2	-	-	-	예(8.0.1 이상)	아니요
$atanh	-	-	-	예(8.0.1 이상)	아니요
$cos	-	-	-	예(8.0.1 이상)	아니요
$cosh	-	-	-	예(8.0.1 이상)	아니요
$degreesToRadians	-	-	-	예(8.0.1 이상)	아니요
$radiansToDegrees	-	-	-	예(8.0.1 이상)	아니요
$sin	-	-	-	예(8.0.1 이상)	아니요
$sinh	-	-	-	예(8.0.1 이상)	아니요
$tan	-	-	-	예(8.0.1 이상)	아니요
$tanh	-	-	-	예(8.0.1 이상)	아니요

비트 연산자(집계)

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$bitAnd	-	-	-	예(8.0.1 이상)	아니요
$bitNot	-	-	-	예(8.0.1 이상)	아니요
$bitOr	-	-	-	예(8.0.1 이상)	아니요
$bitXor	-	-	-	예(8.0.1 이상)	아니요

타임스탬프 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$tsIncrement	-	-	-	예(8.0.1 이상)	아니요
$tsSecond	-	-	-	예(8.0.1 이상)	아니요

기타 연산자

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
$getField	-	-	예	예	아니요
$rand	-	-	예(5.0.1 이상)	예	아니요
$sampleRate	-	-	아니요	예(8.0.1 이상)	아니요
$sigmoid	-	-	-	예(8.0.1 이상)	아니요

데이터 형식

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
32비트 정수 (int)	예	예	예	예	예
64비트 정수 (int)	예	예	예	예	예
배열	예	예	예	예	예
이진 데이터	예	예	예	예	예
부울	예	예	예	예	예
Date	예	예	예	예	예
DBPointer	아니요	아니요	아니요	아니요	아니요
DBRefs	아니요	아니요	아니요	아니요	아니요
Decimal128	예	예	예	예	예
배정밀도 실수	예	예	예	예	예
JavaScript	아니요	아니요	아니요	아니요	아니요
JavaScript(범위 포함)	아니요	아니요	아니요	아니요	아니요
MaxKey	예	예	예	예	예
MinKey	예	예	예	예	예
Null	예	예	예	예	예
객체	예	예	예	예	예

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
ObjectId	예	예	예	예	예
정규식	예	예	예	예	예
문자열	예	예	예	예	예
Symbol	아니요	아니요	아니요	아니요	아니요
타임스탬프	예	예	예	예	예
정의되지 않음	아니요	아니요	아니요	아니요	아니요

인덱스 및 인덱스 속성

주제

- 인덱스
- 인덱스 속성

인덱스

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
2dsphere	예	예	예	예	예
2d 인덱스	아니요	아니요	아니요	아니요	아니요
복합 인덱스	예	예	예	예	예
해시된 인덱스	아니요	아니요	아니요	아니요	아니요

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
Multikey 인덱스	예	예	예	예	예
단일 필드 인덱스	예	예	예	예	예
텍스트 인덱스	아니요	아니요	예	예	아니요
와일드카드	아니요	아니요	아니요	아니요	아니요

인덱스 속성

명령	3.6	4.0	5.0	8.0	엘라스틱 클러스터
배경	예	예	예	예	예
대소문자 구분 안 함	아니요	아니요	아니요	예	아니요
숨김	아니요	아니요	아니요	예(8.0.1 이상)	아니요
부분적	아니요	아니요	예	예	아니요
희소	예	예	예	예	예
텍스트	아니요	아니요	예	예	아니요
TTL	예	예	예	예	예
고유	예	예	예	예	예
벡터	아니요	아니요	예	예	아니요

특정 MongoDB 연산자에 대한 자세한 내용은 다음 주제를 참조하세요.

- 집계 파이프라인 연산자
- Geospatial
- 프로젝션 연산자
- 업데이트 연산자

집계 파이프라인 연산자

이 섹션에서는 Amazon DocumentDB에서 지원하는 집계 파이프라인 연산자에 대한 자세한 정보를 제공합니다.

주제

- $$DESCEND
- $$KEEP
- $$PRUNE
- $ROOT
- $abs
- $acos
- $acosh
- $add
- $addToSet
- $addFields
- $allElementsTrue
- $and
- $anyElementTrue
- $arrayElemAt
- $arrayToObject
- $asin
- $asinh
- $atan

- $atan2
- $atanh
- $avg
- $binarySize
- $bitAnd
- $bitNot
- $bitOr
- $bitXor
- $하위
- $bottomN
- $bsonSize
- $bucket
- $bucketAuto
- $ceil
- $changeStream
- $changeStreamSplitLargeEvent
- $cmp
- $collStats
- $concat
- $concatArrays
- $cond
- $convert
- $cos
- $cosh
- $count(스테이지)
- $count(누적기)
- $covariancePop
- $covarianceSamp

- $currentOp
- $dateAdd
- $dateDiff
- $dateFromParts
- $dateFromString
- $dateSubtract
- $dateToParts
- $dateToString
- $dateTrunc
- $dayOfMonth
- $dayOfWeek
- $dayOfYear
- $degreesToRadians
- $denseRank
- $divide
- $documentNumber
- $eq
- $exp
- $facet
- $filter
- $first
- $firstN
- $floor
- $geoNear
- $graphLookup
- $group
- $gt
- $gte
- $hour

- $ifNull
- $in
- $indexOfArray
- $indexOfBytes
- $indexOfCP
- $indexStats
- $isArray
- $isNumber
- $isoDayOfWeek
- $isoWeek
- $isoWeekYear
- $last
- $lastN
- $let
- $limit
- $listSearchIndexes
- $literal
- $ln
- $log
- $log10
- $lt
- $lte
- $lookup
- $ltrim
- $map
- $match
- $max
- $maxN
- $median

- $meta
- $merge
- $mergeObjects
- $millisecond
- $min
- $minN
- $minute
- $mod
- $month
- $multiply
- $natural
- $ne
- $not
- $objectToArray
- $or
- $out
- $백분위수
- $pow
- $push
- $project
- $radiansToDegrees
- $rand
- $range
- $rank
- $redact
- $reduce
- $regexFind
- $regexFindAll
- $regexMatch

- $replaceAll
- $replaceOne
- $replaceRoot
- $replaceWith
- $reverseArray
- $round
- $rtrim
- $sample
- $search
- $second
- $set
- $setDifference
- $setEquals
- $setIntersection
- $setIsSubset
- $setUnion
- $setWindowFields
- $sigmoid
- $sin
- $sinh
- $skip
- $slice
- $size
- $sort
- $sortArray
- $sortByCount
- $split
- $sqrt
- $stdDevPop

- $stdDevSamp
- $strLenBytes
- $strLenCP
- $strcasecmp
- $substr
- $substrBytes
- $substrCP
- $subtract
- $sum
- $switch
- $tan
- $tanh
- $상위
- $topN
- $trunc
- $tsIncrement
- $tsSecond
- $toBool
- $toDate
- $toDecimal
- $toDouble
- $toInt
- $toLong
- $toLower
- $toObjectId
- $toString
- $toUpper
- $toUUID
- $trim

- $type
- $unset
- $unwind
- $vectorSearch
- $week
- $year
- $zip

$$DESCEND

Amazon DocumentDB의 $$DESCEND 연산자는 $redact 파이프라인 단계 내에서 사용되는 특수 위치 배열 연산자입니다. 집계 파이프라인에 현재 문서로 내림하고 중첩 수준에 관계없이 모든 필드를 처리하도록 지시합니다.

$redact 스테이지에서 $$DESCEND 연산자를 발견하면 현재 문서의 모든 필드를 표시하고 파이프라인에서 더 아래로 처리합니다. 이는 문서의 구조를 유지하면서 조건에 따라 특정 필드를 선택적으로 수정하거나 정리하려는 경우에 유용합니다.

파라미터

없음.

예제(MongoDB 쉘)

이 예제에서는 연$$DESCEND산자와 함께 $redact 단계를 사용하여 code 필드가 "Reg"와 같은 문서를 선택적으로 표시합니다.

샘플 문서 생성

```
db.patient.insertMany([
  { "_id": 1, "code": "Emp", "patient": "John Doe", "DOB": "1/1/1980", "Hospital":
 "Main" },
  { "_id": 2, "code": "Reg", "patient": "Jane Doe", "DOB": "3/27/1989", "Hospital":
 "Cherry Hill" },
  { "_id": 3, "code": "Emp", "patient": "Bob Smith", "DOB": "6/15/1975", "Hospital":
 "Downtown" }
]);
```

쿼리 예제

```
db.patient.aggregate([
  { $redact: {
    $cond: {
      if: { $eq: ["Reg", "$code"] },
      then: "$$DESCEND",
      else: "$$PRUNE"
    }
  }}
]);
```

출력

```
{
  "_id": 2,
  "code": "Reg",
  "patient": "Jane Doe",
  "DOB": "3/27/1989",
  "Hospital": "Cherry Hill"
}
```

코드 예제

$$DESCEND 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function redactPatients() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('patient');

  const result = await collection.aggregate([
    { $redact: {
      $cond: {
        if: { $eq: ["Reg", "$code"] },
        then: "$$DESCEND",
        else: "$$PRUNE"
```

```
      }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

redactPatients();
```

Python

```
from pymongo import MongoClient

def redact_patients():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.patient

    result = list(collection.aggregate([
        { "$redact": {
            "$cond": {
                "if": { "$eq": ["Reg", "$code"] },
                "then": "$$DESCEND",
                "else": "$$PRUNE"
            }
        }}
    ]))

    print(result)
    client.close()

redact_patients()
```

$$KEEP

$$KEEP 시스템 변수는 집계 파이프라인의 $redact 스테이지와 함께 사용되어 현재 문서 또는 필드를 변경하지 않고 출력에 포함합니다.

파라미터

없음

예제(MongoDB 쉘)

다음 예제에서는 Amazon DocumentDB 집계 파이프라인$$KEEP에서의 사용을 보여줍니다. 문서는 액세스가 "퍼블릭"과 같을 때만 보관되며, 그렇지 않으면 제거됩니다.

샘플 문서 생성

```
db.articles.insertMany([
  { title: "Article A", access: "public", content: "Visible content" },
  { title: "Article B", access: "private", content: "Hidden content" }
]);
```

쿼리 예제

```
db.articles.aggregate([
  {
    $redact: {
      $cond: [
        { $eq: ["$access", "public"] },
        "$$KEEP",
        "$$PRUNE"
      ]
    }
  }
]);
```

출력

```
[
  {
    "_id" : ObjectId("..."),
    "title" : "Article A",
    "access" : "public",
    "content" : "Visible content"
  }
]
```

코드 예제

$$KEEP 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function run() {
  const client = new MongoClient(
    'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-bundle.pem&replicaSet=rs0'
  );

  try {
    await client.connect();
    const db = client.db('test');
    const articles = db.collection('articles');

    const pipeline = [
      {
        $redact: {
          $cond: [
            { $eq: ["$access", "public"] },
            "$$KEEP",
            "$$PRUNE"
          ]
        }
      }
    ];

    const results = await articles.aggregate(pipeline).toArray();
    console.log(results);
  } finally {
    await client.close();
  }
}

run().catch(console.error);
```

Python

```
from pymongo import MongoClient

client = MongoClient(
    "mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-bundle.pem&replicaSet=rs0"
```

```
)

db = client.test
articles = db.articles

pipeline = [
    {
        "$redact": {
            "$cond": [
                {"$eq": ["$access", "public"]},
                "$$KEEP",
                "$$PRUNE"
            ]
        }
    }
]

results = list(articles.aggregate(pipeline))
print(results)

client.close()
```

$$PRUNE

$$PRUNE 시스템 변수는 집계 파이프라인의 $redact 단계와 함께 사용되어 결과에서 문서 또는 포함된 문서 수준을 제외합니다. 조건이 로 평가되면 $$PRUNE현재 문서 또는 하위 문서가 출력에서 제거됩니다. 일반적으로 $$DESCEND (문서를 보관 및 통과) 또는 $$KEEP (문서를 모든 수준에서 유지)와 함께 사용됩니다.

파라미터

없음. $$PRUNE 시스템 변수는 파라미터 없이 사용되며와 함께 사용해야 합니다$redact.

예제(MongoDB 쉘)

다음 예제에서는와 $$PRUNE 함께를 사용하여 결과에서 30세 이상의 사용자를 제외$redact하는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insert([
```

```
  { _id:1, name: "Carlos Salazar", age: 35, address: { street: "123 Main St", city:
 "Anytown", state: "CA" } },
  { _id:2, name: "Saanvi Sarkar", age: 28, address: { street: "456 Oak Rd", city:
 "Someplace", state: "NY" } },
  { _id:3, name: "Li Juan", age: 42, address: { street: "789 Pine Ave", city:
 "Springfield", state: "TX" } }
])
```

쿼리 예제

```
db.users.aggregate([
  {
    $redact: {
      $cond: {
        if: { $gt: ["$age", 30] },
        then: "$$PRUNE",
        else: "$$DESCEND"
      }
    }
  }
])
```

출력

```
[
  {
    "_id": 2,
    "name": "Saanvi Sarkar",
    "age": 28,
    "address": {
      "street": "456 Oak Rd",
      "city": "Someplace",
      "state": "NY"
    }
  }
]
```

코드 예제

$$PRUNE 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

async function main() {
  await client.connect();
  const db = client.db('test');
  const users = db.collection('users');

  const result = await users.aggregate([
    {
      $redact: {
        $cond: {
          if: { $gt: ["$age", 30] },
          then: "$$PRUNE",
          else: "$$DESCEND"
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
users = db['users']

result = list(users.aggregate([
    {
```

```
        '$redact': {
            '$cond': {
                'if': { '$gt': ['$age', 30] },
                'then': '$$PRUNE',
                'else': '$$DESCEND'
            }
        }
    }
]))

print(result)
client.close()
```

$ROOT

Amazon DocumentDB의 $ROOT 연산자는 집계 파이프라인 내의 전체 입력 문서를 참조하는 데 사용됩니다. 이를 통해 중첩된 모든 필드 및 구조를 포함하여 전체 문서에 액세스하고 조작할 수 있습니다.

파라미터

없음

예제(MongoDB 쉘)

이 예제에서는 $ROOT를 사용하여 처리 시기에 대한 메타데이터와 함께 전체 원본 문서를 캡처하는 감사 로그를 생성하는 방법을 보여줍니다.

샘플 문서 생성

```
db.orders.insertMany([
  {
    _id: 1,
    orderId: "ORD-2024-001",
    customer: "María García",
    email: "maría@example.com",
    items: [
      { product: "Laptop", quantity: 1, price: 1299.99 }
    ],
    totalAmount: 1299.99
  },
  {
    _id: 2,
```

```
    orderId: "ORD-2024-002",
    customer: "Arnav Desai",
    email: "arnav@example.com",
    items: [
      { product: "Mouse", quantity: 2, price: 29.99 },
      { product: "Keyboard", quantity: 1, price: 89.99 }
    ],
    totalAmount: 149.97
  }
]);
```

쿼리 예제

```
db.orders.aggregate([
  {
    $project: {
      processedAt: new Date(),
      originalDocument: "$$ROOT",
      summary: {
        $concat: [
          "Order ",
          "$orderId",
          " for ",
          "$customer",
          " - Total: $",
          { $toString: "$totalAmount" }
        ]
      }
    }
  }
]);
```

출력

```
[
  {
    _id: 1,
    processedAt: ISODate('2025-11-24T20:43:51.492Z'),
    originalDocument: {
      _id: 1,
      orderId: 'ORD-2024-001',
      customer: 'María García',
      email: 'maría@example.com',
```

```
      items: [ { product: 'Laptop', quantity: 1, price: 1299.99 } ],
      totalAmount: 1299.99
    },
    summary: 'Order ORD-2024-001 for María García - Total: $1299.99'
  },
  {
    _id: 2,
    processedAt: ISODate('2025-11-24T20:43:51.492Z'),
    originalDocument: {
      _id: 2,
      orderId: 'ORD-2024-002',
      customer: 'Arnav Desai',
      email: 'arnav@example.com',
      items: [
        { product: 'Mouse', quantity: 2, price: 29.99 },
        { product: 'Keyboard', quantity: 1, price: 89.99 }
      ],
      totalAmount: 149.97
    },
    summary: 'Order ORD-2024-002 for Arnav Desai - Total: $149.97'
  }
]
```

코드 예제

$ROOT 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function createAuditLog() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const orders = db.collection('orders');

  const result = await orders.aggregate([
    {
      $project: {
        processedAt: new Date(),
        originalDocument: "$$ROOT",
        summary: {
```

```
          $concat: [
            "Order ",
            "$orderId",
            " for ",
            "$customer",
            " - Total: $",
            { $toString: "$totalAmount" }
          ]
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

createAuditLog();
```

Python

```
from pymongo import MongoClient
from datetime import datetime

def create_audit_log():
    client = MongoClient('mongodb://username:password@docdb-
cluster.cluster-123456789.us-east-1.docdb.amazonaws.com:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['orders']

    result = list(collection.aggregate([
        {
            '$project': {
                'processedAt': datetime.now(),
                'originalDocument': '$$ROOT',
                'summary': {
                    '$concat': [
                        "Order ",
                        "$orderId",
                        " for ",
                        "$customer",
```

```
                        " - Total: $",
                        { '$toString': "$totalAmount" }
                    ]
                }
            }
        }
    ]))

    print(result)
    client.close()

create_audit_log()
```

$abs

Amazon DocumentDB의 $abs 연산자는 숫자의 절대값을 반환합니다. 집계 파이프라인에서 숫자 필드에 수학 작업을 수행하는 데 사용할 수 있습니다.

파라미터

- number: 절대 값이 반환될 숫자 표현식입니다.

예제(MongoDB 쉘)

이 예제에서는 연$abs산자를 사용하여 숫자 필드의 절대값을 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.numbers.insertMany([
  { "_id": 1, "value": -5 },
  { "_id": 2, "value": 10 },
  { "_id": 3, "value": -3.14 },
  { "_id": 4, "value": 0 }
]);
```

쿼리 예제

```
db.numbers.aggregate([
  { $project: {
    "_id": 1,
    "absolute_value": { $abs: "$value" }
```

```
  }}
]);
```

출력

```
[
  { "_id": 1, "absolute_value": 5 },
  { "_id": 2, "absolute_value": 10 },
  { "_id": 3, "absolute_value": 3.14 },
  { "_id": 4, "absolute_value": 0 }
]
```

코드 예제

$abs 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $project: {
      "_id": 1,
      "absolute_value": { $abs: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient
```

```
def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['numbers']

    result = list(collection.aggregate([
        { '$project': {
            "_id": 1,
            "absolute_value": { "$abs": "$value" }
        }}
    ]))

    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$acos

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $acos 연산자는 값의 아크코사인(역코사인)을 반환합니다. 입력 값은 -1에서 1 사이여야 합니다.

파라미터

- expression: -1에서 1 사이의 숫자로 확인되는 표현식입니다.

결과는 라디안 단위입니다. 각도$radiansToDegrees를 구하려면 출력에를 적용합니다.

반환 유형은 double 기본적으로 입니다. 입력이 128비트 소수인 경우 출력도 128비트 소수입니다.

예제(MongoDB 쉘)

다음 예제에서는 $acos 연산자를 사용하여 숫자 값의 아크코사인을 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.values.insertMany([
  { "_id": 1, "value": 1 },
  { "_id": 2, "value": 0 },
  { "_id": 3, "value": -1 }
]);
```

쿼리 예제

```
db.values.aggregate([
  { $project: {
    "angle": { $acos: "$value" }
  }}
]);
```

출력

```
[
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 1.5707963267948966 },
  { "_id": 3, "angle": 3.141592653589793 }
]
```

코드 예제

$acos 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "angle": { $acos: "$value" }
    }}
```

```
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'angle': { '$acos': '$value' }
        }}
    ]))

    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$acosh

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $acosh 연산자는 값의 역 하이퍼볼릭 코사인(하이퍼볼릭 아크코사인)을 반환합니다. 입력 값은 1보다 크거나 같아야 합니다.

파라미터

- expression: 1보다 크거나 같은 숫자로 확인되는 표현식입니다.

반환 유형은 double 기본적으로 입니다. 입력이 128비트 10진수인 경우 출력도 128비트 10진수입니다.

예제(MongoDB 쉘)

다음 예제에서는 $acosh 연산자를 사용하여 숫자 값의 역 하이퍼볼릭 코사인을 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.values.insertMany([
  { "_id": 1, "value": 1 },
  { "_id": 2, "value": 2 },
  { "_id": 3, "value": 10 }
]);
```

쿼리 예제

```
db.values.aggregate([
  { $project: {
    "result": { $acosh: "$value" }
  }}
]);
```

출력

```
[
  { "_id": 1, "result": 0 },
  { "_id": 2, "result": 1.3169578969248166 },
  { "_id": 3, "result": 2.993222846126381 }
]
```

코드 예제

$acosh 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "result": { $acosh: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'result': { '$acosh': '$value' }
        }}
    ]))

    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$add

Amazon DocumentDB의 $add 연산자는 숫자 또는 날짜를 함께 추가하는 데 사용됩니다. 숫자 필드에 산술 연산을 수행하거나 날짜 필드에 여러 시간 단위를 추가하여 날짜 산술을 수행하는 데 사용할 수 있습니다.

파라미터

- `expression1`: 추가할 첫 번째 숫자 또는 날짜입니다.
- `expression2`: 추가할 두 번째 숫자 또는 날짜입니다.
- `expression3`: (선택 사항) 추가할 추가 번호 또는 날짜입니다.

예제(MongoDB 쉘)

다음 예제에서는 $add 연산자를 사용하여 두 숫자를 함께 추가하는 방법을 보여줍니다.

샘플 문서 생성

```
db.numbers.insertMany([
  { _id: 1, a: 5, b: 3 },
  { _id: 2, a: 10, b: 7 },
  { _id: 3, a: 2, b: 8 }
]);
```

쿼리 예제

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    sum: { $add: ["$a", "$b"] }
  }}
])
```

출력

```
[
  { "_id" : 1, "sum" : 8 },
  { "_id" : 2, "sum" : 17 },
  { "_id" : 3, "sum" : 10 }
]
```

이 예제에서는 $add 연산자를 사용하여 각 문서에 대한 a 및 b 필드의 값을 추가하고 결과를 sum 필드에 저장합니다.

코드 예제

$add 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $project: {
      _id: 1,
      sum: { $add: ['$a', '$b'] }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['numbers']

result = list(collection.aggregate([
    { '$project': {
```

```
        '_id': 1,
        'sum': { '$add': ['$a', '$b'] }
    }}
]))

print(result)
client.close()
```

$addToSet

$addToSet 집계 연산자는 각 그룹에 대해 지정된 표현식에서 고유한 값의 배열을 반환합니다. $group 스테이지 내에서 고유한 값을 누적하여 중복을 자동으로 제거하는 데 사용됩니다.

파라미터

- expression: 그룹의 각 문서에 대해 평가할 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $addToSet 연산자를 사용하여 각 고객에 대해 주문이 이루어진 고유한 도시를 수집하는 방법을 보여줍니다.

샘플 문서 생성

```
db.orders.insertMany([
  { _id: 1, customer: "Alice", city: "Seattle", amount: 100 },
  { _id: 2, customer: "Alice", city: "Portland", amount: 150 },
  { _id: 3, customer: "Bob", city: "Seattle", amount: 200 },
  { _id: 4, customer: "Alice", city: "Seattle", amount: 75 },
  { _id: 5, customer: "Bob", city: "Boston", amount: 300 }
]);
```

쿼리 예제

```
db.orders.aggregate([
  {
    $group: {
      _id: "$customer",
      cities: { $addToSet: "$city" }
    }
  }
```

```
]);
```

출력

```
[
  { _id: 'Bob', cities: [ 'Seattle', 'Boston' ] },
  { _id: 'Alice', cities: [ 'Seattle', 'Portland' ] }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$addToSet산자는 $setWindowFields 스테이지에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 문서에 대해 지정된 표현식의 고유 값 배열을 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

> **Note**
>
> 에서 창 연산자로 사용할 경우 $setWindowFields$addToSet는 100MB의 중간 데이터로 제한됩니다. 이 제한을 초과하는 작업은 오류를 반환합니다.

샘플 문서 생성

```
db.visits.insertMany([
  { _id: 1, customer: "Alice", day: 1, city: "Seattle" },
  { _id: 2, customer: "Alice", day: 2, city: "Portland" },
  { _id: 3, customer: "Alice", day: 3, city: "Seattle" },
  { _id: 4, customer: "Bob", day: 1, city: "Boston" },
  { _id: 5, customer: "Bob", day: 2, city: "Boston" }
]);
```

쿼리 예제

다음 예제에서는 문서를 기준으로 분할하고customer, 각 파티션을 기준으로 정렬하고day, 파티션 시작부터 현재 문서까지 고유한 city 값 집합을 반환합니다.

```
db.visits.aggregate([
  {
    $setWindowFields: {
```

```
      partitionBy: "$customer",
      sortBy: { day: 1 },
      output: {
        citiesSoFar: {
          $addToSet: "$city",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "customer": "Alice", "day": 1, "city": "Seattle", "citiesSoFar":
 [ "Seattle" ] },
  { "_id": 2, "customer": "Alice", "day": 2, "city": "Portland", "citiesSoFar":
 [ "Portland", "Seattle" ] },
  { "_id": 3, "customer": "Alice", "day": 3, "city": "Seattle", "citiesSoFar":
 [ "Portland", "Seattle" ] },
  { "_id": 4, "customer": "Bob", "day": 1, "city": "Boston", "citiesSoFar":
 [ "Boston" ] },
  { "_id": 5, "customer": "Bob", "day": 2, "city": "Boston", "citiesSoFar":
 [ "Boston" ] }
]
```

각 문서는 시작부터 현재 문서까지 파티션 내의 고유 city 값 집합citiesSoFar인 로 보강됩니다. 누적기 양식과 마찬가지로 반환된 배열의 값 순서는 보장되지 않습니다.

코드 예제

$addToSet 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누산기 사용량()과 창 연산자 사용량($group)을 모두 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
```

```
const db = client.db('test');

// Accumulator usage: collect unique values per group
const orders = db.collection('orders');
await orders.insertMany([
  { _id: 1, customer: "Alice", city: "Seattle", amount: 100 },
  { _id: 2, customer: "Alice", city: "Portland", amount: 150 },
  { _id: 3, customer: "Bob", city: "Seattle", amount: 200 },
  { _id: 4, customer: "Alice", city: "Seattle", amount: 75 },
  { _id: 5, customer: "Bob", city: "Boston", amount: 300 }
]);
const accumulatorResult = await orders.aggregate([
  {
    $group: {
      _id: "$customer",
      cities: { $addToSet: "$city" }
    }
  }
]).toArray();
console.log('Accumulator result:', accumulatorResult);

// Window operator usage: running set of unique values within each partition
const visits = db.collection('visits');
await visits.insertMany([
  { _id: 1, customer: "Alice", day: 1, city: "Seattle" },
  { _id: 2, customer: "Alice", day: 2, city: "Portland" },
  { _id: 3, customer: "Alice", day: 3, city: "Seattle" },
  { _id: 4, customer: "Bob", day: 1, city: "Boston" },
  { _id: 5, customer: "Bob", day: 2, city: "Boston" }
]);
const windowResult = await visits.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$customer",
      sortBy: { day: 1 },
      output: {
        citiesSoFar: {
          $addToSet: "$city",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]).toArray();
```

```
  console.log('Window result:', windowResult);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']

    # Accumulator usage: collect unique values per group
    orders = db['orders']
    orders.insert_many([
        { '_id': 1, 'customer': 'Alice', 'city': 'Seattle', 'amount': 100 },
        { '_id': 2, 'customer': 'Alice', 'city': 'Portland', 'amount': 150 },
        { '_id': 3, 'customer': 'Bob', 'city': 'Seattle', 'amount': 200 },
        { '_id': 4, 'customer': 'Alice', 'city': 'Seattle', 'amount': 75 },
        { '_id': 5, 'customer': 'Bob', 'city': 'Boston', 'amount': 300 }
    ])
    accumulator_result = list(orders.aggregate([
        {
            '$group': {
                '_id': '$customer',
                'cities': { '$addToSet': '$city' }
            }
        }
    ]))
    print('Accumulator result:', accumulator_result)

    # Window operator usage: running set of unique values within each partition
    visits = db['visits']
    visits.insert_many([
        { '_id': 1, 'customer': 'Alice', 'day': 1, 'city': 'Seattle' },
        { '_id': 2, 'customer': 'Alice', 'day': 2, 'city': 'Portland' },
        { '_id': 3, 'customer': 'Alice', 'day': 3, 'city': 'Seattle' },
        { '_id': 4, 'customer': 'Bob', 'day': 1, 'city': 'Boston' },
```

```
        { '_id': 5, 'customer': 'Bob', 'day': 2, 'city': 'Boston' }
    ])
    window_result = list(visits.aggregate([
        {
            '$setWindowFields': {
                'partitionBy': '$customer',
                'sortBy': { 'day': 1 },
                'output': {
                    'citiesSoFar': {
                        '$addToSet': '$city',
                        'window': { 'documents': ['unbounded', 'current'] }
                    }
                }
            }
        }
    ]))
    print('Window result:', window_result)

    client.close()

example()
```

$addFields

Amazon DocumentDB 집계 파이프라인의 $addFields 단계를 사용하면 문서에 계산된 새 필드를 추가할 수 있습니다. 이는 파생되거나 변환된 데이터를 문서에 추가하는 데 유용할 수 있습니다.

파라미터

- newField: 추가할 새 필드의 이름입니다.
- expression: 새 필드의 값으로 확인되는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는를 $addFields 사용하여 Inventory.OnHand 및 필드를 기반으로 총 인벤토리를 계산TotalInventory하는 새 Inventory.OrderQnty 필드를 추가하는 방법을 보여줍니다.

샘플 문서 생성

```
db.example.insertMany([
```

```
  {
    "Item": "Spray Paint",
    "Colors": ["Black", "Red", "Green", "Blue"],
    "Inventory": {
      "OnHand": 47,
      "MinOnHand": 50,
      "OrderQnty": 36
    },
    "UnitPrice": 3.99
  },
  {
    "Item": "Ruler",
    "Colors": ["Red", "Green", "Blue", "Clear", "Yellow"],
    "Inventory": {
      "OnHand": 47,
      "MinOnHand": 40
    },
    "UnitPrice": 0.89
  }
]);
```

쿼리 예제

```
db.example.aggregate([
  {
    $addFields: {
      TotalInventory: { $add: ["$Inventory.OnHand", "$Inventory.OrderQnty"] }
    }
  }
])
```

출력

```
[
  {
    "_id" : ObjectId("5bedafbcf65ff161707de24f"),
    "Item" : "Ruler",
    "Colors" : [ "Red", "Green", "Blue", "Clear", "Yellow" ],
    "Inventory" : {
      "OnHand" : 47,
      "MinOnHand" : 40
    },
    "UnitPrice" : 0.89,
```

```
    "TotalInventory" : 47
  },
  {
    "_id" : ObjectId("5bedafbcf65ff161707de250"),
    "Item" : "Spray Paint",
    "Colors" : [ "Black", "Red", "Green", "Blue" ],
    "Inventory" : {
      "OnHand" : 47,
      "MinOnHand" : 50,
      "OrderQnty" : 36
    },
    "UnitPrice" : 3.99,
    "TotalInventory" : 83
  }
]
```

코드 예제

$addFields 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
const db = client.db('test');
const collection = db.collection('example');

const result = await collection.aggregate([
  {
    $addFields: {
      TotalInventory: { $add: ['$Inventory.OnHand', '$Inventory.OrderQnty'] }
    }
  }
]).toArray();

console.log(result);

await client.close();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['example']

result = list(collection.aggregate([
    {
        '$addFields': {
            'TotalInventory': { '$add': ['$Inventory.OnHand',
 '$Inventory.OrderQnty'] }
        }
    }
]))

print(result)
client.close()
```

$allElementsTrue

버전 4.0의 새로운 기능

연$allElementsTrue산자는 배열의 모든 요소가 true 값으로 평가되는지 확인하는 데 사용됩니다.

파라미터

- expression: 배열로 평가되는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는를 사용하여 배열의 모든 요소가 true인지 확인하는 $allElementsTrue 방법을 보여줍니다.

샘플 문서 생성

```
db.collection.insert([
  { "name": "John", "scores": [100, 90, 80] },
```

```
  { "name": "Jane", "scores": [80, 85, 0] },
  { "name": "Bob", "scores": [90, 95, null] }
])
```

쿼리 예제

```
db.collection.find({
  "scores": { "$allElementsTrue": [{ "$gt": 0 }] }
})
```

출력

```
[
  { "_id" : ObjectId("..."), "name" : "John", "scores" : [ 100, 90, 80 ] },
  { "_id" : ObjectId("..."), "name" : "Bob", "scores" : [ 90, 95, null ] }
]
```

이 예제에서 쿼리는 scores 배열의 모든 요소가 0보다 큰지 확인합니다. scores 배열에 거짓 값인 0이 포함되어 있기 때문에가 있는 문서는 "name": "Jane" 제외됩니다.

코드 예제

$allElementsTrue 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.find({
    "scores": { "$allElementsTrue": [{ "$gt": 0 }] }
  }).toArray();

  console.log(result);
```

```
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['collection']

    result = list(collection.find({
        "scores": {"$allElementsTrue": [{"$gt": 0}]}
    }))

    print(result)

    client.close()

example()
```

$and

$and 집계 연산자는 여러 표현식을 평가하고 모든 표현식이 로 평가되는 true 경우에만를 반환합니다true. 표현식이 이면 false를 반환합니다false.

파라미터

- expressions: 평가할 표현식의 배열입니다.

예제(MongoDB 쉘)

다음 예제에서는 $and 연산자를 사용하여 제품이 여러 기준을 충족하는지 확인하는 방법을 보여줍니다.

샘플 문서 생성

```
db.products.insertMany([
  { _id: 1, name: "Laptop", price: 1200, inStock: true },
  { _id: 2, name: "Mouse", price: 25, inStock: false },
  { _id: 3, name: "Keyboard", price: 75, inStock: true }
]);
```

쿼리 예제

```
db.products.aggregate([
  {
    $project: {
      name: 1,
      price: 1,
      inStock: 1,
      affordable: {
        $and: [
          { $lt: ["$price", 100] },
          { $eq: ["$inStock", true] }
        ]
      }
    }
  }
]);
```

출력

```
[
  { _id: 1, name: 'Laptop', price: 1200, inStock: true, affordable: false },
  { _id: 2, name: 'Mouse', price: 25, inStock: false, affordable: false },
  { _id: 3, name: 'Keyboard', price: 75, inStock: true, affordable: true }
]
```

코드 예제

$and 집계 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
        price: 1,
        inStock: 1,
        affordable: {
          $and: [
            { $lt: ["$price", 100] },
            { $eq: ["$inStock", true] }
          ]
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['products']

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
```

```
                'price': 1,
                'inStock': 1,
                'affordable': {
                    '$and': [
                        { '$lt': ['$price', 100] },
                        { '$eq': ['$inStock', True] }
                    ]
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$anyElementTrue

버전 4.0의 새로운 기능

`$anyElementTrue` 연산자는 배열의 요소가 true인지 확인하는 데 사용됩니다.

파라미터

- `field`: 평가할 배열 필드입니다.

예제(MongoDB 쉘)

다음 예제에서는를 사용하여 배열의 요소가 true인지 확인하는 `$anyElementTrue` 방법을 보여줍니다.

샘플 문서 생성

```
db.grades.insertMany([
  { _id: 1, student: 'Tim', scores: [false, false, null] },
  { _id: 2, student: 'Bob', scores: [false, 0, false] },
  { _id: 3, student: 'Ivy', scores: [false, true, 0] }
])
```

쿼리 예제

```
db.grades.aggregate([
  {
    $project: {
      student: 1,
      isAnyTrue: { $anyElementTrue: ["$scores"] },
      _id: 0
    }
  }
])
```

출력

```
[
  { student: 'Tim', isAnyTrue: false },
  { student: 'Bob', isAnyTrue: false },
  { student: 'Ivy', isAnyTrue: true }
]
```

코드 예제

$anyElementTrue 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('grades');

  const result = await collection.aggregate([
    {
      $project: {
        student: 1,
        isAnyTrue: { $anyElementTrue: ["$scores"] },
        _id: 0
      }
    }
  ]).toArray();
```

```
  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['grades']

    result = list(collection.aggregate([
      {
        "$project": {
          "student": 1,
          "isAnyTrue": { "$anyElementTrue": ["$scores"] },
          "_id": 0
        }
      }
    ]))

    print(result)

    client.close()

example()
```

$arrayElemAt

Amazon DocumentDB의 `$arrayElemAt` 연산자를 사용하면 인덱스 위치를 기준으로 배열에서 요소를 검색할 수 있습니다. 이는 문서의 배열 필드 내에서 특정 요소에 액세스해야 할 때 특히 유용합니다.

파라미터

- `array`: 요소를 검색할 입력 배열입니다.

- index: 검색할 요소의 0 기반 인덱스 위치입니다. 이 값은 음수가 아닌 정수여야 합니다.

예제(MongoDB 쉘)

이 예제에서는 $arrayElemAt 연산자를 사용하여 miles 컬렉션의 flight_miles 배열에서 특정 요소를 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.miles.insertMany([
  { "_id" : 1, "member_since" : ISODate("1987-01-01T00:00:00Z"), "credit_card" : false,
 "flight_miles" : [ 1205, 2560, 880 ]},
  { "_id" : 2, "member_since" : ISODate("1982-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 1205, 2560, 890, 2780 ]},
  { "_id" : 3, "member_since" : ISODate("1999-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 1205, 880 ]}
]);
```

쿼리 예제

```
db.miles.aggregate([
  { $project: {
    "_id": 1,
    "first_mile": { $arrayElemAt: [ "$flight_miles", 0 ] },
    "last_mile": { $arrayElemAt: [ "$flight_miles", -1 ] }
  }}
]);
```

출력

```
{ "_id" : 1, "first_mile" : 1205, "last_mile" : 880 }
{ "_id" : 2, "first_mile" : 1205, "last_mile" : 2780 }
{ "_id" : 3, "first_mile" : 1205, "last_mile" : 880 }
```

이 예제에서는 $arrayElemAt 연산자를 사용하여 각 문서에 대한 flight_miles 배열의 첫 번째 및 마지막 요소를 검색합니다.

코드 예제

$arrayElemAt 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('miles');

  const result = await collection.aggregate([
    { $project: {
      "_id": 1,
      "first_mile": { $arrayElemAt: [ "$flight_miles", 0 ] },
      "last_mile": { $arrayElemAt: [ "$flight_miles", -1 ] }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.mydatabase
    collection = db.miles

    result = list(collection.aggregate([
        { "$project": {
            "_id": 1,
            "first_mile": { "$arrayElemAt": [ "$flight_miles", 0 ] },
            "last_mile": { "$arrayElemAt": [ "$flight_miles", -1 ] }
        }}
    ]))
```

```
    print(result)
    client.close()

example()
```

$arrayToObject

Amazon DocumentDB의 $arrayToObject 연산자는 $objectToArray 연산자의 반대입니다. 키-값 페어 문서의 배열을 가져와서 단일 문서로 변환합니다. 이는 키-값 페어의 배열을 객체 또는 문서 구조로 다시 변환해야 할 때 특히 유용합니다.

파라미터

- array expression: 배열로 확인되는 표현식입니다. 배열 요소는 (키) 및 k (v값)의 두 필드가 있는 문서여야 합니다.

예제(MongoDB 쉘)

다음 예제에서는를 $arrayToObject 사용하여 키-값 페어의 배열을 다시 문서로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.videos.insertMany([
  { "_id": 1, "name": "Live Soft", "inventory": { "Des Moines": 1000, "Ames": 500 } },
  { "_id": 2, "name": "Top Pilot", "inventory": { "Mason City": 250, "Des Moines":
 1000 } },
  { "_id": 3, "name": "Romancing the Rock", "inventory": { "Mason City": 250, "Ames":
 500 } },
  { "_id": 4, "name": "Bravemind", "inventory": { "Mason City": 250, "Des Moines":
 1000, "Ames": 500 } }
]);
```

쿼리 예제

```
db.videos.aggregate([
  { $project: { name: 1, videos: { $objectToArray: "$inventory" } } },
  { $project: { name: 1, inventory: { $arrayToObject: "$videos" } } }
]);
```

출력

```
{ "_id" : 1, "name" : "Live Soft", "inventory" : { "Des Moines" : 1000, "Ames" :
 500 } }
{ "_id" : 2, "name" : "Top Pilot", "inventory" : { "Mason City" : 250, "Des Moines" :
 1000 } }
{ "_id" : 3, "name" : "Romancing the Rock", "inventory" : { "Mason City" : 250,
 "Ames" : 500 } }
{ "_id" : 4, "name" : "Bravemind", "inventory" : { "Mason City" : 250, "Des Moines" :
 1000, "Ames" : 500 } }
```

이 예제에서는 $objectToArray 연산자를 사용하여 inventory 객체를 키-값 페어의 배열로 변환합니다. 그런 다음 $arrayToObject 연산자를 사용하여 배열을 다시 문서로 변환하여 원래 객체 구조를 복원합니다.

코드 예제

$arrayToObject 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('videos');

  const result = await collection.aggregate([
    { $project: { name: 1, videos: { $objectToArray: '$inventory' } } },
    { $project: { name: 1, inventory: { $arrayToObject: '$videos' } } }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['videos']

    result = list(collection.aggregate([
        { '$project': { 'name': 1, 'videos': { '$objectToArray': '$inventory' } } },
        { '$project': { 'name': 1, 'inventory': { '$arrayToObject': '$videos' } } }
    ]))

    print(result)
    client.close()

example()
```

$asin

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $asin 연산자는 값의 아크신(역사인)을 반환합니다. 입력 값은 -1에서 1 사이여야 합니다.

파라미터

- expression: -1과 1 사이의 숫자로 확인되는 표현식입니다.

결과는 라디안 단위입니다. 각도$radiansToDegrees를 구하려면 출력에를 적용합니다.

반환 유형은 double 기본적으로 입니다. 입력이 128비트 10진수인 경우 출력도 128비트 10진수입니다.

예제(MongoDB 쉘)

다음 예제에서는 $asin 연산자를 사용하여 숫자 값의 아크신을 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 0.5 },
  { "_id": 3, "value": 1 }
]);
```

쿼리 예제

```
db.values.aggregate([
  { $project: {
    "angle": { $asin: "$value" }
  }}
]);
```

출력

```
[
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 0.5235987755982989 },
  { "_id": 3, "angle": 1.5707963267948966 }
]
```

코드 예제

$asin 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "angle": { $asin: "$value" }
    }}
```

```
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'angle': { '$asin': '$value' }
        }}
    ]))

    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$asinh

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $asinh 연산자는 값의 역 하이퍼볼릭 사인을 반환합니다. 집계 파이프라인에서 이를 사용하여 숫자 필드에 대해 하이퍼볼릭 삼각형 계산을 수행합니다.

파라미터

- expression: 숫자로 확인되는 표현식입니다.

결과는 라디안 단위입니다. 각도$radiansToDegrees를 구하려면 출력에를 적용합니다.

반환 유형은 double 기본적으로 입니다. 입력이 128비트 10진수인 경우 출력도 128비트 10진수입니다.

예제(MongoDB 쉘)

다음 예제에서는 $asinh 연산자를 사용하여 숫자 값의 역 하이퍼볼릭 사인을 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 1 },
  { "_id": 3, "value": -1 }
]);
```

쿼리 예제

```
db.values.aggregate([
  { $project: {
    "result": { $asinh: "$value" }
  }}
]);
```

출력

```
[
  { "_id": 1, "result": 0 },
  { "_id": 2, "result": 0.881373587019543 },
  { "_id": 3, "result": -0.881373587019543 }
]
```

코드 예제

$asinh 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "result": { $asinh: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'result': { '$asinh': '$value' }
        }}
    ]))

    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$atan

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $atan 연산자는 값의 역탄젠트(아크탄젠트)를 반환합니다.

입력 표현식은 숫자 값으로 확인되어야 합니다.

결과는 라디안 단위입니다. 각도$radiansToDegrees를 구하려면 출력에를 적용합니다.

반환 유형은 double 기본적으로 입니다. 입력이 128비트 10진수인 경우 출력도 128비트 10진수입니다.

파라미터

- expression: 숫자로 확인되는 표현식입니다.

동작

null 및 NaN

예제	결과
{ $atan: NaN }	NaN
{ $atan: null }	null

입력이 null 이거나 참조된 필드가 누락된 경우 결과는 입니다null. 의 입력은를 NaN 생성합니다NaN.

예제(MongoDB 쉘)

다음 예제에서는 $atan 연산자를 사용하여 값의 아크탄젠트를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 1 },
  { "_id": 3, "value": -1 }
]);
```

쿼리 예제

```
db.values.aggregate([
  { $project: {
    "angle": { $atan: "$value" }
  }}
]);
```

출력

```
[
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 0.7853981633974483 },
  { "_id": 3, "angle": -0.7853981633974483 }
]
```

코드 예제

$atan 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "angle": { $atan: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'angle': { '$atan': '$value' }
        }}
    ]))

    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$atan2

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $atan2 연산자는의 역탄젠트(아크탄젠트)를 반환합니다. y / x여기서 y 및 x는 각각 표현식에 전달되는 첫 번째 및 두 번째 값입니다.

두 표현식 모두 숫자 값으로 확인되어야 합니다. 연산자는 두 인수의 부호를 사용하여 올바른 사분면을 결정합니다.

결과는 라디안 단위입니다. 각도$radiansToDegrees를 구하려면 출력에를 적용합니다.

반환 유형은 double 기본적으로 입니다. 입력이 128비트 소수인 경우 출력도 128비트 소수입니다.

파라미터

- expression 1 (y): y 좌표를 나타내는 숫자로 확인되는 표현식입니다.
- expression 2 (x): x 좌표를 나타내는 숫자로 확인되는 표현식입니다.

동작

null 및 NaN

인수 중 하나가 인 경우null(또는 누락된 필드를 참조하는 경우) 결과는 입니다null. 두 인수 중 하나가 인 경우 NaN결과는 입니다NaN. 한 인수는 null이고 다른 인수는 인 경우 NaNnull가 우선합니다.

예제	결과
{ $atan2: [NaN, <value>] }	NaN
{ $atan2: [<value>, NaN] }	NaN
{ $atan2: [null, <value>] }	null
{ $atan2: [<value>, null] }	null
{ $atan2: [NaN, null] }	null
{ $atan2: [null, NaN] }	null

예제(MongoDB 쉘)

다음 예제에서는 $atan2 연산자를 사용하여 점 집합에 대한 y/x의 아크탄젠트를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.points.insertMany([
  { "_id": 1, "y": 1, "x": 1 },
  { "_id": 2, "y": 1, "x": 0 },
  { "_id": 3, "y": -1, "x": -1 }
]);
```

쿼리 예제

```
db.points.aggregate([
  { $project: {
    "angle": { $atan2: ["$y", "$x"] }
  }}
]);
```

출력

```
[
  { "_id": 1, "angle": 0.7853981633974483 },
  { "_id": 2, "angle": 1.5707963267948966 },
  { "_id": 3, "angle": -2.356194490192345 }
]
```

코드 예제

$atan2 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('points');

  const result = await collection.aggregate([
    { $project: {
      "angle": { $atan2: ["$y", "$x"] }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
    db = client['test']
    collection = db['points']

    result = list(collection.aggregate([
        { '$project': {
            'angle': { '$atan2': ['$y', '$x'] }
        }}
    ]))

    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$atanh

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $atanh 연산자는 값의 역 하이퍼볼릭 탄젠트(하이퍼볼릭 아크 탄젠트)를 반환합니다.

입력 표현식은 ~1(포함) 범위의 숫자-1로 확인되어야 합니다. 이 범위를 벗어나는 값은 오류를 일으킵니다.

반환 유형은 double 기본적으로 입니다. 입력이 128비트 10진수인 경우 출력도 128비트 10진수입니다.

파라미터

- expression: -1 ~ 사이의 숫자로 확인되는 표현1식입니다.

동작

null, NaN 및 +/- 무한대

예제	결과
{ $atanh: NaN }	NaN
{ $atanh: null }	null

예제	결과
`{ $atanh: 1 }`	`Infinity`
`{ $atanh: -1 }`	`-Infinity`
`{ $atanh: Infinity }` 또는 `{ $atanh: -Infinity }`	오류가 발생합니다.

입력이 `null` 이거나 참조된 필드가 누락된 경우 결과는 입니다`null`. 의 입력은를 `NaN` 생성합니다`NaN`. 경계 값과는 `-Infinity` 각각 `Infinity` 및 `1`를 `-1` 생성합니다. `[-1, 1]` 범위를 벗어나는 값(포함`±Infinity`)이 있으면 오류가 발생합니다.

예제(MongoDB 쉘)

다음 예제에서는 `$atanh` 연산자를 사용하여 값의 역 하이퍼볼릭 탄젠트를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 0.5 },
  { "_id": 3, "value": -0.5 }
]);
```

쿼리 예제

```
db.values.aggregate([
  { $project: {
    "result": { $atanh: "$value" }
  }}
]);
```

출력

```
[
  { "_id": 1, "result": 0 },
  { "_id": 2, "result": 0.5493061443340548 },
  { "_id": 3, "result": -0.5493061443340548 }
```

```
]
```

코드 예제

$atanh 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "result": { $atanh: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'result': { '$atanh': '$value' }
```

```
        }}
    ]))

    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$avg

Amazon DocumentDB의 $avg 집계 연산자는 단계에 입력되는 문서 전체에서 지정된 표현식의 평균 값을 계산합니다. 이 연산자는 문서 세트에서 숫자 필드 또는 표현식의 평균을 계산하는 데 유용합니다.

파라미터

- expression: 평균을 계산하는 데 사용할 표현식입니다. 필드 경로(예: "$field") 또는 표현식(예: { $multiply: ["$field1", "$field2"] })일 수 있습니다.

예제(MongoDB 쉘)

다음 예제에서는 $avg 연산자를 사용하여 학생 문서 세트의 평균 점수를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.students.insertMany([
  { name: "John", score: 85 },
  { name: "Jane", score: 92 },
  { name: "Bob", score: 78 },
  { name: "Alice", score: 90 }
]);
```

쿼리 예제

```
db.students.aggregate([
  { $group: {
    _id: null,
    avgScore: { $avg: "$score" }
```

```
  }}
]);
```

출력

```
[
  {
    "_id": null,
    "avgScore": 86.25
  }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$avg산자는 $setWindowFields 스테이지에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 문서에 대해 지정된 표현식의 평균을 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

샘플 문서 생성

```
db.readings.insertMany([
  { _id: 1, city: "Denver", day: 1, temp: 70 },
  { _id: 2, city: "Denver", day: 2, temp: 80 },
  { _id: 3, city: "Seattle", day: 1, temp: 60 },
  { _id: 4, city: "Seattle", day: 2, temp: 64 },
  { _id: 5, city: "Seattle", day: 3, temp: 68 }
]);
```

쿼리 예제

다음 예제에서는를 기준으로 문서를 분할하고city,를 기준으로 각 파티션을 정렬하고day, 파티션 시작temp부터 현재 문서까지의 실행 평균을 반환합니다.

```
db.readings.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$city",
      sortBy: { day: 1 },
      output: {
```

```
        runningAvg: {
          $avg: "$temp",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "city": "Denver", "day": 1, "temp": 70, "runningAvg": 70 },
  { "_id": 2, "city": "Denver", "day": 2, "temp": 80, "runningAvg": 75 },
  { "_id": 3, "city": "Seattle", "day": 1, "temp": 60, "runningAvg": 60 },
  { "_id": 4, "city": "Seattle", "day": 2, "temp": 64, "runningAvg": 62 },
  { "_id": 5, "city": "Seattle", "day": 3, "temp": 68, "runningAvg": 64 }
]
```

각 문서는 현재 문서를 포함하여 파티션 temp 내의 평균runningAvg인 로 보강됩니다.

코드 예제

$avg 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누산기 사용량()과 창 연산자 사용량($group)을 모두 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function calculateAvgScore() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');

  // Accumulator usage: average across grouped documents
  const students = db.collection('students');
  await students.insertMany([
    { name: "John", score: 85 },
    { name: "Jane", score: 92 },
    { name: "Bob", score: 78 },
    { name: "Alice", score: 90 }
```

```
  ]);
  const accumulatorResult = await students.aggregate([
    { $group: {
      _id: null,
      avgScore: { $avg: '$score' }
    }}
  ]).toArray();
  console.log('Accumulator result:', accumulatorResult);

  // Window operator usage: running average within each partition
  const readings = db.collection('readings');
  await readings.insertMany([
    { _id: 1, city: "Denver", day: 1, temp: 70 },
    { _id: 2, city: "Denver", day: 2, temp: 80 },
    { _id: 3, city: "Seattle", day: 1, temp: 60 },
    { _id: 4, city: "Seattle", day: 2, temp: 64 },
    { _id: 5, city: "Seattle", day: 3, temp: 68 }
  ]);
  const windowResult = await readings.aggregate([
    {
      $setWindowFields: {
        partitionBy: "$city",
        sortBy: { day: 1 },
        output: {
          runningAvg: {
            $avg: "$temp",
            window: { documents: ["unbounded", "current"] }
          }
        }
      }
    }
  ]).toArray();
  console.log('Window result:', windowResult);

  await client.close();
}

calculateAvgScore();
```

Python

```
from pymongo import MongoClient
```

```
def calculate_avg_score():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test

    # Accumulator usage: average across grouped documents
    students = db.students
    students.insert_many([
        { 'name': 'John', 'score': 85 },
        { 'name': 'Jane', 'score': 92 },
        { 'name': 'Bob', 'score': 78 },
        { 'name': 'Alice', 'score': 90 }
    ])
    accumulator_result = list(students.aggregate([
        { '$group': {
            '_id': None,
            'avgScore': { '$avg': '$score' }
        }}
    ]))
    print('Accumulator result:', accumulator_result)

    # Window operator usage: running average within each partition
    readings = db.readings
    readings.insert_many([
        { '_id': 1, 'city': 'Denver', 'day': 1, 'temp': 70 },
        { '_id': 2, 'city': 'Denver', 'day': 2, 'temp': 80 },
        { '_id': 3, 'city': 'Seattle', 'day': 1, 'temp': 60 },
        { '_id': 4, 'city': 'Seattle', 'day': 2, 'temp': 64 },
        { '_id': 5, 'city': 'Seattle', 'day': 3, 'temp': 68 }
    ])
    window_result = list(readings.aggregate([
        {
            '$setWindowFields': {
                'partitionBy': '$city',
                'sortBy': { 'day': 1 },
                'output': {
                    'runningAvg': {
                        '$avg': '$temp',
                        'window': { 'documents': ['unbounded', 'current'] }
                    }
                }
            }
        }
```

```
    ]))
    print('Window result:', window_result)

    client.close()

calculate_avg_score()
```

$binarySize

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $binarySize 연산자는 지정된 문자열 또는 이진 데이터 값의 크기를 바이트 단위로 반환합니다. 문자열 값의 경우 문자 수가 아닌 UTF-8 인코딩 바이트 수입니다. 멀티바이트 UTF-8 문자(예: 액센트가 있는 문자 또는 CJK 문자)는 문자열의 문자 수보다 큰 바이트 수를 생성합니다.

파라미터

- expression: 문자열 또는 이진 데이터 값으로 확인되는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $binarySize 연산자를 사용하여 문자열 필드의 바이트 크기를 반환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.docs.insertMany([
  {_id: 1, content: "hello"},
  {_id: 2, content: "Amazon DocumentDB"},
  {_id: 3, content: ""}
]);
```

쿼리 예제

```
db.docs.aggregate([
  { $project: { size: { $binarySize: "$content" } } }
]);
```

출력

```
[
  {_id: 1, size: 5},
  {_id: 2, size: 17},
  {_id: 3, size: 0}
]
```

코드 예제

$binarySize 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('docs');
    const result = await collection.aggregate([
      { $project: { size: { $binarySize: "$content" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
```

```
        collection = db['docs']
        result = list(collection.aggregate([
            {'$project': {'size': {'$binarySize': '$content'}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$bitAnd

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $bitAnd 연산자는 정수 또는 긴 값에 대해 비트 단위 AND 작업을 수행합니다.

파라미터

- expressions: 정수 또는 길이로 해석할 수 있는 2개 이상의 표현식 배열입니다.

예제(MongoDB 쉘)

다음 예제에서는 $bitAnd 연산자를 사용하여 두 필드에 대해 비트 단위 AND를 수행하는 방법을 보여줍니다.

샘플 문서 생성

```
db.flags.insertMany([
  {_id: 1, a: 13, b: 10},
  {_id: 2, a: 7, b: 5},
  {_id: 3, a: 15, b: 9}
]);
```

쿼리 예제

```
db.flags.aggregate([
  { $project: { result: { $bitAnd: ["$a", "$b"] } } }
]);
```

출력

```
[
  {_id: 1, result: 8},
  {_id: 2, result: 5},
  {_id: 3, result: 9}
]
```

바이너리: 13(1101) AND 10(1010) = 8(1000), 7(0111) AND 5(0101) = 5(0101), 15(1111) AND 9(1001) = 9(1001).

코드 예제

$bitAnd 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('flags');
    const result = await collection.aggregate([
      { $project: { result: { $bitAnd: ["$a", "$b"] } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
    try:
        db = client['test']
        collection = db['flags']
        result = list(collection.aggregate([
            {'$project': {'result': {'$bitAnd': ['$a', '$b']}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$bitNot

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $bitNot 연산자는 정수 또는 긴 값에 대해 비트 단위 NOT 작업을 수행하여 비트 단위 보완을 반환합니다. 지정된 정수 또는 긴 n의 경우 결과는 -(n+1)입니다.

파라미터

- expression: 정수 또는 길이로 확인되는 표현식입니다. 지정된 정수 또는 긴 n에 대해 -(n+1)인 비트 보완을 반환합니다.

예제(MongoDB 쉘)

다음 예제에서는 $bitNot 연산자를 사용하여 정수 값의 비트 단위 보완을 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.numbers.insertMany([
  {_id: 1, value: 0},
  {_id: 2, value: 5},
  {_id: 3, value: -3}
]);
```

쿼리 예제

```
db.numbers.aggregate([
  { $project: { result: { $bitNot: "$value" } } }
```

```
]);
```

출력

```
[
  {_id: 1, result: -1},
  {_id: 2, result: -6},
  {_id: 3, result: 2}
]
```

바이너리: NOT 0 = -1, NOT 5 = -6, NOT -3 = 2.

코드 예제

$bitNot 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('numbers');
    const result = await collection.aggregate([
      { $project: { result: { $bitNot: "$value" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['numbers']
        result = list(collection.aggregate([
            {'$project': {'result': {'$bitNot': '$value'}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$bitOr

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 `$bitOr` 연산자는 정수 또는 긴 값에 대해 비트 단위 OR 작업을 수행합니다.

파라미터

- `expressions`: 정수 또는 길이로 해석할 수 있는 2개 이상의 표현식 배열입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$bitOr` 연산자를 사용하여 두 필드에서 비트 단위 OR을 수행하는 방법을 보여줍니다.

샘플 문서 생성

```
db.flags.insertMany([
  {_id: 1, a: 13, b: 10},
  {_id: 2, a: 7, b: 5},
  {_id: 3, a: 15, b: 9}
]);
```

쿼리 예제

```
db.flags.aggregate([
```

```
  { $project: { result: { $bitOr: ["$a", "$b"] } } }
]);
```

출력

```
[
  {_id: 1, result: 15},
  {_id: 2, result: 7},
  {_id: 3, result: 15}
]
```

바이너리: 13(1101) 또는 10(1010) = 15(1111), 7(0111) 또는 5(0101) = 7(0111), 15(1111) 또는 9(1001) = 15(1111).

코드 예제

`$bitOr` 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('flags');
    const result = await collection.aggregate([
      { $project: { result: { $bitOr: ["$a", "$b"] } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient
```

```
def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['flags']
        result = list(collection.aggregate([
            {'$project': {'result': {'$bitOr': ['$a', '$b']}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$bitXor

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 `$bitXor` 연산자는 정수 또는 긴 값에 대해 비트 단위 XOR 작업을 수행합니다.

파라미터

- `expressions`: 정수 또는 길이로 해석할 수 있는 2개 이상의 표현식 배열입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$bitXor` 연산자를 사용하여 두 필드에서 비트 단위 XOR을 수행하는 방법을 보여줍니다.

샘플 문서 생성

```
db.flags.insertMany([
  {_id: 1, a: 13, b: 10},
  {_id: 2, a: 7, b: 5},
  {_id: 3, a: 15, b: 9}
]);
```

쿼리 예제

```
db.flags.aggregate([
  { $project: { result: { $bitXor: ["$a", "$b"] } } }
]);
```

출력

```
[
  {_id: 1, result: 7},
  {_id: 2, result: 2},
  {_id: 3, result: 6}
]
```

바이너리: 13(1101) XOR 10(1010) = 7(0111), 7(0111) XOR 5(0101) = 2(0010), 15(1111) XOR 9(1001) = 6(0110).

코드 예제

$bitXor 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('flags');
    const result = await collection.aggregate([
      { $project: { result: { $bitXor: ["$a", "$b"] } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['flags']
        result = list(collection.aggregate([
            {'$project': {'result': {'$bitXor': ['$a', '$b']}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$하위

버전 8.0.1에서 새로 추가되었습니다.

$group 단계의 $bottom 누적기를 사용하여 지정된 정렬 순서에 따라 그룹당 가장 낮은 순위의 문서를 반환합니다.

파라미터

- sortBy: 정렬 순서를 지정하는 문서입니다. 오름차순1에는를 사용하고 내림차순에는 -1를 사용합니다.
- output: 하단 문서에서 반환할 필드를 지정하는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $bottom 누적기를 사용하여 판매 컬렉션의 항목당 최저 판매량(최저 수량)을 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.sales.insertMany([
```

```
  { item: "abc", quantity: 10, price: 5 },
  { item: "abc", quantity: 5, price: 8 },
  { item: "xyz", quantity: 15, price: 3 },
  { item: "xyz", quantity: 7, price: 6 }
])
```

쿼리 예제

```
db.sales.aggregate([
  { $group: { _id: "$item", bottomSale: { $bottom: { sortBy: { quantity: -1 }, output:
 { quantity: "$quantity", price: "$price" } } } } }
])
```

출력

```
[
  { "_id": "xyz", "bottomSale": { "quantity": 7, "price": 6 } },
  { "_id": "abc", "bottomSale": { "quantity": 5, "price": 8 } }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$bottom산자는 $setWindowFields 스테이지에서 창 연산자로도 사용할 수 있습니다. 이 컨텍스트에서는 각 창의 문서 중에서 가장 낮은 순위output의 문서(연산자 자신의에 따라sortBy)에서를 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

> **Note**
>
> 에서 창 연산자로 사용할 경우 $setWindowFields$bottom는 100MB의 중간 데이터로 제한됩니다. 이 제한을 초과하는 작업은 오류를 반환합니다.

샘플 문서 생성

```
db.matchScores.insertMany([
  { _id: 1, player: "Alice", round: 1, score: 20 },
  { _id: 2, player: "Alice", round: 2, score: 35 },
  { _id: 3, player: "Alice", round: 3, score: 28 },
```

```
  { _id: 4, player: "Bob", round: 1, score: 15 },
  { _id: 5, player: "Bob", round: 2, score: 40 }
]);
```

쿼리 예제

다음 예제에서는를 기준으로 문서를 분할하고player,를 기준으로 각 파티션을 정렬하고round, 파티션 시작부터 현재 문서까지 가장 낮은 점수를 반환합니다.

```
db.matchScores.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$player",
      sortBy: { round: 1 },
      output: {
        worstRoundSoFar: {
          $bottom: { sortBy: { score: -1 }, output: { round: "$round", score:
 "$score" } },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "player": "Alice", "round": 1, "score": 20, "worstRoundSoFar": { "round":
 1, "score": 20 } },
  { "_id": 2, "player": "Alice", "round": 2, "score": 35, "worstRoundSoFar": { "round":
 1, "score": 20 } },
  { "_id": 3, "player": "Alice", "round": 3, "score": 28, "worstRoundSoFar": { "round":
 1, "score": 20 } },
  { "_id": 4, "player": "Bob", "round": 1, "score": 15, "worstRoundSoFar": { "round":
 1, "score": 15 } },
  { "_id": 5, "player": "Bob", "round": 2, "score": 40, "worstRoundSoFar": { "round":
 1, "score": 15 } }
]
```

각 문서는 현재 문서를 포함하여 파티션 내에서 가장 낮은 점수 라운드worstRoundSoFar인 로 보강됩니다.

코드 예제

$bottom 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누산기 사용량()과 창 연산자 사용량($group)을 모두 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const db = client.db('test');

    // Accumulator usage: bottom document per group
    const sales = db.collection('sales');
    await sales.insertMany([
      { item: "abc", quantity: 10, price: 5 },
      { item: "abc", quantity: 5, price: 8 },
      { item: "xyz", quantity: 15, price: 3 },
      { item: "xyz", quantity: 7, price: 6 }
    ]);
    const accumulatorResult = await sales.aggregate([
      { $group: { _id: "$item", bottomSale: { $bottom: { sortBy: { quantity: -1 },
 output: { quantity: "$quantity", price: "$price" } } } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Window operator usage: worst round so far within each partition
    const matchScores = db.collection('matchScores');
    await matchScores.insertMany([
      { _id: 1, player: "Alice", round: 1, score: 20 },
      { _id: 2, player: "Alice", round: 2, score: 35 },
      { _id: 3, player: "Alice", round: 3, score: 28 },
      { _id: 4, player: "Bob", round: 1, score: 15 },
      { _id: 5, player: "Bob", round: 2, score: 40 }
    ]);
    const windowResult = await matchScores.aggregate([
```

```
      {
        $setWindowFields: {
          partitionBy: "$player",
          sortBy: { round: 1 },
          output: {
            worstRoundSoFar: {
              $bottom: { sortBy: { score: -1 }, output: { round: "$round", score:
 "$score" } },
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
    ]).toArray();
    console.log('Window result:', windowResult);

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client['test']

        # Accumulator usage: bottom document per group
        sales = db['sales']
        sales.insert_many([
            { 'item': 'abc', 'quantity': 10, 'price': 5 },
```

```
            { 'item': 'abc', 'quantity': 5, 'price': 8 },
            { 'item': 'xyz', 'quantity': 15, 'price': 3 },
            { 'item': 'xyz', 'quantity': 7, 'price': 6 }
        ])
        accumulator_result = list(sales.aggregate([
            { '$group': { '_id': '$item', 'bottomSale': { '$bottom':
{ 'sortBy': { 'quantity': -1 }, 'output': { 'quantity': '$quantity', 'price':
'$price' } } } } }
        ]))
        pprint(accumulator_result)

        # Window operator usage: worst round so far within each partition
        match_scores = db['matchScores']
        match_scores.insert_many([
            { '_id': 1, 'player': 'Alice', 'round': 1, 'score': 20 },
            { '_id': 2, 'player': 'Alice', 'round': 2, 'score': 35 },
            { '_id': 3, 'player': 'Alice', 'round': 3, 'score': 28 },
            { '_id': 4, 'player': 'Bob', 'round': 1, 'score': 15 },
            { '_id': 5, 'player': 'Bob', 'round': 2, 'score': 40 }
        ])
        window_result = list(match_scores.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$player',
                    'sortBy': { 'round': 1 },
                    'output': {
                        'worstRoundSoFar': {
                            '$bottom': { 'sortBy': { 'score': -1 }, 'output':
{ 'round': '$round', 'score': '$score' } },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        pprint(window_result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()
```

```
example()
```

$bottomN

버전 8.0.1에서 새로 추가되었습니다.

$group 스테이지의 $bottomN 누적기를 사용하여 지정된 정렬 순서에 따라 그룹 내의 하단 N 요소를 반환합니다. 그룹에 N개 미만의 요소가 포함된 경우는 그룹의 모든 요소를 $bottomN 반환합니다.

파라미터

- n: 그룹당 반환할 하위 결과 수를 지정하여 양수 또는 1로 확인되는 표현식입니다.
- sortBy: 정렬 순서를 지정하는 문서입니다. 오름차순1에는를 사용하고 내림차순에는 -1를 사용합니다.
- output: 각 하단 N개 문서에서 반환할 필드를 지정하는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $bottomN 누적기를 사용하여 판매 컬렉션의 항목당 하위 2개 판매(최저 수량)를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.sales.insertMany([
  { item: "abc", quantity: 10, price: 5 },
  { item: "abc", quantity: 7, price: 8 },
  { item: "abc", quantity: 5, price: 10 },
  { item: "xyz", quantity: 15, price: 3 },
  { item: "xyz", quantity: 9, price: 6 },
  { item: "xyz", quantity: 3, price: 12 }
])
```

쿼리 예제

```
db.sales.aggregate([
  { $group: { _id: "$item", bottomTwoSales: { $bottomN: { n: 2, sortBy: { quantity:
 -1 }, output: { quantity: "$quantity", price: "$price" } } } } }
])
```

출력

```
[
  { "_id": "xyz", "bottomTwoSales": [{ "quantity": 9, "price": 6 }, { "quantity": 3,
 "price": 12 }] },
  { "_id": "abc", "bottomTwoSales": [{ "quantity": 7, "price": 8 }, { "quantity": 5,
 "price": 10 }] }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$bottomN산자는 $setWindowFields 스테이지에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 문서 간에 output 하단 N 문서(연산자 자신의에 따라sortBy)에서의 배열을 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

> **Note**
>
> 에서 창 연산자로 사용할 경우 $setWindowFields$bottomN는 100MB의 중간 데이터로 제한됩니다. 이 제한을 초과하는 작업은 오류를 반환합니다.

샘플 문서 생성

```
db.matchScores.insertMany([
  { _id: 1, player: "Alice", round: 1, score: 20 },
  { _id: 2, player: "Alice", round: 2, score: 35 },
  { _id: 3, player: "Alice", round: 3, score: 28 },
  { _id: 4, player: "Bob", round: 1, score: 15 },
  { _id: 5, player: "Bob", round: 2, score: 40 }
]);
```

쿼리 예제

다음 예제에서는 문서를 로 분할하고player, 각 파티션을 로 정렬하고round, 파티션 시작부터 현재 문서까지 가장 낮은 점수를 받은 두 라운드를 반환합니다.

```
db.matchScores.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$player",
      sortBy: { round: 1 },
```

```
      output: {
        bottomTwoRoundsSoFar: {
          $bottomN: { n: 2, sortBy: { score: -1 }, output: { round: "$round", score:
 "$score" } },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "player": "Alice", "round": 1, "score": 20, "bottomTwoRoundsSoFar":
 [{ "round": 1, "score": 20 }] },
  { "_id": 2, "player": "Alice", "round": 2, "score": 35, "bottomTwoRoundsSoFar":
 [{ "round": 2, "score": 35 }, { "round": 1, "score": 20 }] },
  { "_id": 3, "player": "Alice", "round": 3, "score": 28, "bottomTwoRoundsSoFar":
 [{ "round": 3, "score": 28 }, { "round": 1, "score": 20 }] },
  { "_id": 4, "player": "Bob", "round": 1, "score": 15, "bottomTwoRoundsSoFar":
 [{ "round": 1, "score": 15 }] },
  { "_id": 5, "player": "Bob", "round": 2, "score": 40, "bottomTwoRoundsSoFar":
 [{ "round": 2, "score": 40 }, { "round": 1, "score": 15 }] }
]
```

각 문서는 현재 문서를 포함하여 파티션 내에서 bottomTwoRoundsSoFar가장 낮은 두 라운드인 로 보강됩니다. 창에 문서가 2개 미만인 경우 모든 문서가 반환됩니다.

코드 예제

$bottomN 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누산기 사용량()과 창 연산자 사용량($group)을 모두 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);
```

```
 try {
   await client.connect();

   const db = client.db('test');

   // Accumulator usage: bottom N documents per group
   const sales = db.collection('sales');
   await sales.insertMany([
     { item: "abc", quantity: 10, price: 5 },
     { item: "abc", quantity: 7, price: 8 },
     { item: "abc", quantity: 5, price: 10 },
     { item: "xyz", quantity: 15, price: 3 },
     { item: "xyz", quantity: 9, price: 6 },
     { item: "xyz", quantity: 3, price: 12 }
   ]);
   const accumulatorResult = await sales.aggregate([
     { $group: { _id: "$item", bottomTwoSales: { $bottomN: { n: 2, sortBy:
{ quantity: -1 }, output: { quantity: "$quantity", price: "$price" } } } } }
   ]).toArray();
   console.log('Accumulator result:', accumulatorResult);

   // Window operator usage: bottom two rounds so far within each partition
   const matchScores = db.collection('matchScores');
   await matchScores.insertMany([
     { _id: 1, player: "Alice", round: 1, score: 20 },
     { _id: 2, player: "Alice", round: 2, score: 35 },
     { _id: 3, player: "Alice", round: 3, score: 28 },
     { _id: 4, player: "Bob", round: 1, score: 15 },
     { _id: 5, player: "Bob", round: 2, score: 40 }
   ]);
   const windowResult = await matchScores.aggregate([
     {
       $setWindowFields: {
         partitionBy: "$player",
         sortBy: { round: 1 },
         output: {
           bottomTwoRoundsSoFar: {
             $bottomN: { n: 2, sortBy: { score: -1 }, output: { round: "$round",
score: "$score" } },
             window: { documents: ["unbounded", "current"] }
           }
         }
       }
```

```
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client['test']

        # Accumulator usage: bottom N documents per group
        sales = db['sales']
        sales.insert_many([
            { 'item': 'abc', 'quantity': 10, 'price': 5 },
            { 'item': 'abc', 'quantity': 7, 'price': 8 },
            { 'item': 'abc', 'quantity': 5, 'price': 10 },
            { 'item': 'xyz', 'quantity': 15, 'price': 3 },
            { 'item': 'xyz', 'quantity': 9, 'price': 6 },
            { 'item': 'xyz', 'quantity': 3, 'price': 12 }
        ])
        accumulator_result = list(sales.aggregate([
            { '$group': { '_id': '$item', 'bottomTwoSales': { '$bottomN': { 'n':
 2, 'sortBy': { 'quantity': -1 }, 'output': { 'quantity': '$quantity', 'price':
 '$price' } } } } }
        ]))
        pprint(accumulator_result)
```

```
        # Window operator usage: bottom two rounds so far within each partition
        match_scores = db['matchScores']
        match_scores.insert_many([
            { '_id': 1, 'player': 'Alice', 'round': 1, 'score': 20 },
            { '_id': 2, 'player': 'Alice', 'round': 2, 'score': 35 },
            { '_id': 3, 'player': 'Alice', 'round': 3, 'score': 28 },
            { '_id': 4, 'player': 'Bob', 'round': 1, 'score': 15 },
            { '_id': 5, 'player': 'Bob', 'round': 2, 'score': 40 }
        ])
        window_result = list(match_scores.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$player',
                    'sortBy': { 'round': 1 },
                    'output': {
                        'bottomTwoRoundsSoFar': {
                            '$bottomN': { 'n': 2, 'sortBy': { 'score': -1 },
 'output': { 'round': '$round', 'score': '$score' } },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        pprint(window_result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$bsonSize

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $bsonSize 연산자는 BSON으로 인코딩될 때 문서의 크기를 바이트 단위로 반환합니다.

파라미터

- `expression`: 문서 또는 객체로 확인되는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $bsonSize 연산자를 사용하여 각 문서의 BSON 크기를 반환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.items.insertMany([
  {_id: 1, name: "a", tags: ["x"]},
  {_id: 2, name: "ab", tags: ["x", "y", "z"]},
  {_id: 3, name: "abc", tags: []}
]);
```

쿼리 예제

```
db.items.aggregate([
  { $project: { docSize: { $bsonSize: "$$ROOT" } } }
]);
```

출력

```
[
  {_id: 1, docSize: 46},
  {_id: 2, docSize: 65},
  {_id: 3, docSize: 39}
]
```

> **Note**
>
> 정확한 크기는 문서 필드의 BSON 인코딩에 따라 달라집니다.

코드 예제

$bsonSize 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('items');
    const result = await collection.aggregate([
      { $project: { docSize: { $bsonSize: "$$ROOT" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['items']
        result = list(collection.aggregate([
            {'$project': {'docSize': {'$bsonSize': '$$ROOT'}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$bucket

버전 8.0의 새로운 기능

Elastic 클러스터에서는 지원되지 않습니다.

Amazon DocumentDB의 $bucket 집계 단계를 사용하면 지정된 표현식 및 버킷 경계를 기반으로 입력 문서를 버킷으로 그룹화할 수 있습니다. 이는 특정 값 범위 또는 범주에 속하는 데이터를 분석하는 데 유용할 수 있습니다.

파라미터

- groupBy (필수): 그룹화할 값을 지정하는 표현식입니다.
- boundaries (필수): 버킷 경계를 정의하는 이중 값의 배열입니다. 문서는 지정된 경계 내에 속하는 groupBy 표현식 값을 기반으로 버킷에 할당됩니다.
- default (선택 사항): groupBy 표현식 값이 지정된 경계에 속하지 않는 문서에 대해 출력되는 리터럴 값입니다.
- output (선택 사항): 각 버킷에 대해 출력할 정보를 지정하는 객체입니다. $sum, avgmin, 및 같은 누적기 연산자를 사용하여 각 버킷에 대한 집계를 계산$max할 수 있습니다.

예제(MongoDB 쉘)

다음 예제에서는 $bucket 스테이지를 사용하여 판매 데이터를 가격 범위별로 그룹화하는 방법을 보여줍니다.

샘플 문서 생성

```
db.sales.insertMany([
  { item: "abc", price: 10, quantity: 2, date: new Date("2020-09-01") },
  { item: "def", price: 20, quantity: 1, date: new Date("2020-10-01") },
  { item: "ghi", price: 5, quantity: 3, date: new Date("2020-11-01") },
  { item: "jkl", price: 15, quantity: 2, date: new Date("2020-12-01") },
  { item: "mno", price: 25, quantity: 1, date: new Date("2021-01-01") }
]);
```

쿼리 예제

```
db.sales.aggregate([
  {
    $bucket: {
```

```
      groupBy: "$price",
      boundaries: [0, 10, 20, 30],
      default: "Other",
      output: {
        "count": { $sum: 1 },
        "totalQuantity": { $sum: "$quantity" }
      }
    }
  },
  {
    $sort: { _id: 1 }
  }
])
```

출력

```
[
  { _id: 0, count: 1, totalQuantity: 3 },
  { _id: 10, count: 2, totalQuantity: 4 },
  { _id: 20, count: 2, totalQuantity: 2 }
]
```

코드 예제

$bucket 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const sales = db.collection('sales');

  const result = await sales.aggregate([
    {
      $bucket: {
        groupBy: "$price",
        boundaries: [0, 10, 20, 30],
        default: "Other",
```

```
        output: {
          "count": { $sum: 1 },
          "totalQuantity": { $sum: "$quantity" }
        }
      }
    },
    {
      $sort: { _id: 1 }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&lsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    sales = db['sales']

    result = list(sales.aggregate([
        {
            '$bucket': {
                'groupBy': '$price',
                'boundaries': [0, 10, 20, 30],
                'default': 'Other',
                'output': {
                    'count': {'$sum': 1},
                    'totalQuantity': {'$sum': '$quantity'}
                }
            }
        },
        {
            "$sort": { "_id": 1 }
        }
```

```
    ]))

    print(result)
    client.close()

example()
```

$bucketAuto

버전 8.0.2에서 새로 추가되었습니다.

Amazon DocumentDB의 $bucketAuto 집계 단계는 지정된 표현식을 기반으로 입력 문서를 지정된 수의 버킷으로 자동으로 그룹화합니다. 버킷 경계를 명시적으로 정의하는와 달리 $bucket는 문서가 요청된 버킷 수에 최대한 균등하게 배포되도록 경계를 $bucketAuto 자동으로 결정합니다. 이는 값의 분포를 미리 모르고 데이터를 고정된 수의 그룹으로 분류하려는 경우에 유용할 수 있습니다.

파라미터

- groupBy (필수): 그룹화할 값을 지정하는 표현식입니다.
- buckets (필수): 문서가 그룹화되는 버킷 수를 지정하는 양의 정수입니다. 입력 문서와 groupBy 표현식에 따라 실제 버킷 수는 지정된 수보다 적을 수 있습니다.
- output (선택 사항): 각 버킷에 대해 출력할 정보를 지정하는 객체입니다. $sum, avgmin, 및 같은 누적기 연산자를 사용하여 각 버킷에 대한 집계를 계산$max할 수 있습니다. 생략하면 각 버킷은 기본적으로 count 필드를 출력합니다.
- granularity (선택 사항): 버킷 경계에 사용할 기본 숫자 시리즈를 지정하는 문자열입니다. 이렇게 하면 계산된 경계가 표준 수치적 진행에 맞춰집니다. 이 파라미터는 모든 groupBy 값이 숫자이고 모두 인 경우에만 사용할 수 있습니다NaN. 유효한 값은 R5, , R10, R20R40, R80, , 1-2-5, E6, E12, E24E48, E96, 및 E192입니다POWERSOF2.

출력

각 버킷은 버킷의 경계를 설명하는 _id 객체가 포함된 별도의 문서로 반환됩니다.

- _id.min: 버킷의 포함 하한입니다.
- _id.max: 버킷의 상한입니다. 이 바인딩은 포함된 시리즈의 최종 버킷을 제외한 모든 버킷에 대해 제외됩니다.

예제(MongoDB 쉘)

다음 예제에서는 $bucketAuto 스테이지를 사용하여 판매 데이터를 가격별로 버킷으로 그룹화하는 방법을 보여줍니다.

샘플 문서 생성

```
db.sales.insertMany([
  { item: "abc", price: 10, quantity: 2, date: new Date("2020-09-01") },
  { item: "def", price: 20, quantity: 1, date: new Date("2020-10-01") },
  { item: "ghi", price: 5, quantity: 3, date: new Date("2020-11-01") },
  { item: "jkl", price: 15, quantity: 2, date: new Date("2020-12-01") },
  { item: "mno", price: 25, quantity: 1, date: new Date("2021-01-01") }
]);
```

쿼리 예제

다음 명령은 샘플 문서를 가격별로 세 개의 버킷으로 그룹화합니다.

```
db.sales.aggregate([
  {
    $bucketAuto: {
      groupBy: "$price",
      buckets: 3,
      output: {
        "count": { $sum: 1 },
        "totalQuantity": { $sum: "$quantity" }
      }
    }
  }
])
```

출력

명령은 다음 출력을 반환합니다.

```
[
  { _id: { min: 5, max: 15 }, count: 2, totalQuantity: 5 },
  { _id: { min: 15, max: 25 }, count: 2, totalQuantity: 3 },
  { _id: { min: 25, max: 25 }, count: 1, totalQuantity: 1 }
]
```

세분화된 쿼리 예제

다음 예제에서는 계산된 버킷 경계를 2의 출력에 맞추는 granularity 옵션을 추가합니다.

```
db.sales.aggregate([
  {
    $bucketAuto: {
      groupBy: "$price",
      buckets: 2,
      granularity: "POWERSOF2",
      output: {
        "count": { $sum: 1 },
        "totalQuantity": { $sum: "$quantity" }
      }
    }
  }
])
```

출력

명령은 다음 출력을 반환합니다.

```
[
  { _id: { min: 4, max: 16 }, count: 3, totalQuantity: 7 },
  { _id: { min: 16, max: 32 }, count: 2, totalQuantity: 2 }
]
```

코드 예제

$bucketAuto 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const sales = db.collection('sales');
```

```
  const result = await sales.aggregate([
    {
      $bucketAuto: {
        groupBy: "$price",
        buckets: 3,
        output: {
          "count": { $sum: 1 },
          "totalQuantity": { $sum: "$quantity" }
        }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    sales = db['sales']

    result = list(sales.aggregate([
        {
            '$bucketAuto': {
                'groupBy': '$price',
                'buckets': 3,
                'output': {
                    'count': {'$sum': 1},
                    'totalQuantity': {'$sum': '$quantity'}
                }
            }
        }
    ]))
```

```
    print(result)
    client.close()

example()
```

$ceil

버전 4.0의 새로운 기능

MongoDB와 마찬가지로 Amazon DocumentDB의 `$ceil` 연산자는 숫자를 가장 가까운 정수로 반올림합니다. 이는 숫자 필드에 수학 연산을 수행하고 결과가 정수인지 확인해야 할 때 유용합니다.

파라미터

- `expression`: 반올림할 숫자 표현식입니다.

예제(MongoDB 쉘)

이 예제에서는 `$ceil` 연산자를 사용하여 숫자 필드를 반올림하는 방법을 보여줍니다.

샘플 문서 생성

```
db.numbers.insertMany([
  { "_id": 1, "value": 3.14 },
  { "_id": 2, "value": -2.7 },
  { "_id": 3, "value": 0 }
])
```

쿼리 예제

```
db.numbers.aggregate([
  { $project: {
    "roundedUp": { $ceil: "$value" }
  }}
])
```

출력

```
{ "_id": 1, "roundedUp": 4 }
{ "_id": 2, "roundedUp": -2 }
```

```
{ "_id": 3, "roundedUp": 0 }
```

코드 예제

$ceil 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $project: {
      "roundedUp": { $ceil: "$value" }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.numbers

    result = list(collection.aggregate([
        { '$project': {
            "roundedUp": { '$ceil': "$value" }
```

```
        }}
    ]))

    print(result)
    client.close()

example()
```

$changeStream

Elastic 클러스터에서는 지원되지 않습니다.

$changeStream 집계 단계에서는 변경 스트림 커서를 열어 컬렉션에 대한 실시간 변경 사항을 모니터링합니다. 삽입, 업데이트, 교체 또는 삭제 작업이 발생할 때 변경 이벤트 문서를 반환합니다.

파라미터

- fullDocument: 업데이트 작업을 위해 전체 문서를 반환할지 여부를 지정합니다. 옵션은 default 또는 updateLookup입니다.
- resumeAfter: 선택 사항입니다. 토큰을 재개하여 변경 스트림의 특정 지점에서 계속합니다.
- startAtOperationTime: 선택 사항입니다. 변경 스트림을 시작할 타임스탬프입니다.
- allChangesForCluster: 선택 사항입니다. 부울 값입니다. true인 경우 클러스터 전체의 모든 변경 사항을 감시합니다(관리자 데이터베이스의 경우). false (기본값)인 경우는 지정된 컬렉션만 감시합니다.
- showExpandedEvents: 선택 사항입니다. 부울 값입니다. true인 경우는 createCollection 및와 같은 확장된 데이터 정의 언어(DDL) 이벤트 알림을 변경 스트림createIndexes에 포함합니다. false (기본값)인 경우 이러한 이벤트는 반환되지 않습니다. 엔진 버전 8.0.2부터 Amazon DocumentDB 8.0에서 사용할 수 있습니다.

예제(MongoDB 쉘)

다음 예제에서는 $changeStream 스테이지를 사용하여 컬렉션에 대한 변경 사항을 모니터링하는 방법을 보여줍니다.

쿼리 예제

```
// Open change stream first
const changeStream = db.inventory.aggregate([
```

```
  { $changeStream: { fullDocument: "updateLookup" } }
]);

// In another session, insert a document
db.inventory.insertOne({ _id: 1, item: "Widget", qty: 10 });

// Back in the first session, read the change event
if (changeStream.hasNext()) {
  print(tojson(changeStream.next()));
}
```

출력

```
{
  _id: { _data: '...' },
  operationType: 'insert',
  clusterTime: Timestamp(1, 1234567890),
  fullDocument: { _id: 1, item: 'Widget', qty: 10 },
  ns: { db: 'test', coll: 'inventory' },
  documentKey: { _id: 1 }
}
```

코드 예제

$changeStream 집계 단계 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('inventory');

  // Open change stream
  const changeStream = collection.watch([]);

  changeStream.on('change', (change) => {
    console.log('Change detected:', change);
  });
```

```
  // Simulate insert in another operation
  setTimeout(async () => {
    await collection.insertOne({ _id: 1, item: 'Widget', qty: 10 });
  }, 1000);

  // Keep connection open to receive changes
  // In production, handle cleanup appropriately
}

example();
```

Python

```
from pymongo import MongoClient
import threading
import time

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['inventory']

    # Open change stream
    change_stream = collection.watch([])

    # Insert document in separate thread after delay
    def insert_doc():
        time.sleep(1)
        collection.insert_one({'_id': 1, 'item': 'Widget', 'qty': 10})

    threading.Thread(target=insert_doc).start()

    # Watch for changes
    for change in change_stream:
        print('Change detected:', change)
        break  # Exit after first change

    client.close()

example()
```

$changeStreamSplitLargeEvent

이 집계 단계는 탄력적 클러스터에서 지원되지 않습니다.

이 단계는 Amazon DocumentDB 엔진 버전 8.0.2부터 사용할 수 있습니다.

$changeStreamSplitLargeEvent 집계 단계는 $changeStream 파이프라인 내에서 16MB를 초과하는 이벤트를 더 작은 조각으로 분할하는 데 사용됩니다. 변경 스트림 커서를 통해 각 조각을 순차적으로 반환합니다.

파라미터

없음. $changeStreamSplitLargeEvent 스테이지에 대한 입력은 빈 문서여야 합니다.

예 - MongoDB 쉘

다음 예제에서는 $changeStreamSplitLargeEvent 스테이지를 사용하여 크기 초과 변경 이벤트를 여러 조각으로 분할하는 방법을 보여줍니다.

쿼리 예제

```
// Insert a document
db.inventory.insertOne({ _id: 1, item: "Widget", payload: "x".repeat(9 * 1024 *
 1024) })

// Open change stream with the $changeStreamSplitLargeEvent stage
var changeStream = db.inventory.aggregate([
  { $changeStream: { fullDocument: "updateLookup" } },
  { $changeStreamSplitLargeEvent: {} }
]);

// Update the document
db.inventory.updateOne({ _id: 1 }, { $set: { payload: "y".repeat(9 * 1024 * 1024) } })

// Read the change event
if (changeStream.hasNext()) {
  print(tojson(changeStream.next()));
}
```

출력

```
{
  _id: { _data: '...' },
  splitEvent: { fragment: 1, of: 2},
```

```
  ns: { db: 'test', coll: 'inventory' },
  clusterTime: Timestamp(4, 1789154779),
  documentKey: { _id: 1 },
  fullDocument: { _id: 1, item: 'Widget', payload: yyyyyyyy...(9437184 chars) },
  operationType: 'update',
}
{
  _id: { _data: '...' },
  splitEvent: { fragment: 2, of: 2},
  updateDescription: { updatedFields: { payload: yyyyyyyy...(9437184 chars) },
 removedFields: [] }
}
```

코드 예제

$changeStreamSplitLargeEvent 집계 단계 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('inventory');

  // Insert a large document first
  await collection.insertOne({ _id: 1, item: 'Widget', payload: 'x'.repeat(9 * 1024
 * 1024) });

  // Open change stream with $changeStreamSplitLargeEvent
  const changeStream = collection.watch(
    [{ $changeStreamSplitLargeEvent: {} }],
    { fullDocument: 'updateLookup' }
  );

  changeStream.on('change', (change) => {
    console.log('Change detected:', change);
  });
```

```
  // Update to trigger a split event — fullDocument + updateDescription > 16 MB
  setTimeout(async () => {
    console.log('Triggering update...');
    await collection.updateOne({ _id: 1 }, { $set: { payload: 'y'.repeat(9 * 1024 *
 1024) } });
  }, 1000);

  // Keep connection open to receive changes
  // In production, handle cleanup appropriately
}

example();
```

Python

```
from pymongo import MongoClient
import threading
import time

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['inventory']

    # Insert large document first
    collection.drop()
    collection.insert_one({'_id': 1, 'item': 'Widget', 'payload': 'x' * (9 * 1024 *
 1024)})

    # Open change stream with $changeStreamSplitLargeEvent
    change_stream = collection.watch(
        [{'$changeStreamSplitLargeEvent': {}}],
        full_document='updateLookup'
    )

    # Update in separate thread after delay
    def update_doc():
        collection.update_one({'_id': 1}, {'$set': {'payload': 'y' * (9 * 1024 *
 1024)}})

    threading.Thread(target=update_doc).start()
```

```
    # Watch for changes
    for change in change_stream:
        print('Change detected:', change)

    client.close()

example()
```

$cmp

Amazon DocumentDB의 $cmp 연산자는 두 값을 비교하고 상대 순서를 나타내는 정수 값을 반환하는 데 사용됩니다. 첫 번째 값이 각각 두 번째 값보다 작은지, 같은지 또는 큰지에 따라 두 표현식을 비교하고 -1, 0 또는 1의 정수 값을 반환하는 비교 연산자입니다.

파라미터

- `expression1`: 비교할 첫 번째 표현식입니다.
- `expression2`: 비교할 두 번째 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 연$cmp산자를 사용하여 두 숫자 값을 비교하는 방법을 보여줍니다.

샘플 문서 생성

```
db.collection.insertMany([
  { _id: 1, value1: 10, value2: 20 },
  { _id: 2, value1: 15, value2: 15 },
  { _id: 3, value1: 20, value2: 10 }
]);
```

쿼리 예제

```
db.collection.find({
  $expr: {
    $cmp: ["$value1", "$value2"]
  }
})
```

출력

```
[
  { "_id" : 1, "value1" : 10, "value2" : 20 },
  { "_id" : 3, "value1" : 20, "value2" : 10 }
]
```

이 예제에서 $cmp 연산자는 각 문서의 value1 및 value2 필드를 비교합니다. 결과는 다음과 같습니다.

```
- `$cmp: ["$value1", "$value2"]` returns -1 for the first document (10 < 20), 0 for the
 second document (15 = 15), and 1 for the third document (20 > 10).
```

코드 예제

$cmp 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

다음은 mongodb 드라이버와 함께 Node.js 애플리케이션에서 $cmp 연산자를 사용하는 예입니다.

```
const { MongoClient } = require('mongodb');

const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

async function main() {
  await client.connect();
  const db = client.db('test');
  const collection = db.collection('mycollection');

  // Insert sample documents
  await collection.insertMany([
    { _id: 1, value1: 10, value2: 20 },
    { _id: 2, value1: 15, value2: 15 },
    { _id: 3, value1: 20, value2: 10 }
  ]);

  // Query using $cmp operator
  const result = await collection.find({
    $expr: {
      $cmp: ['$value1', '$value2']
```

```
    }
  }).toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

다음은 pymongo 드라이버와 함께 Python 애플리케이션에서 $cmp 연산자를 사용하는 예입니다.

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['mycollection']

# Insert sample documents
collection.insert_many([
    {'_id': 1, 'value1': 10, 'value2': 20},
    {'_id': 2, 'value1': 15, 'value2': 15},
    {'_id': 3, 'value1': 20, 'value2': 10}
])

# Query using $cmp operator
result = list(collection.find({
    '$expr': {
        '$cmp': ['$value1', '$value2']
    }
}))

print(result)

client.close()
```

Node.js 및 Python 예제의 출력은 MongoDB Shell 예제와 동일합니다.

```
[
```

```
  { "_id" : 1, "value1" : 10, "value2" : 20 },
  { "_id" : 2, "value1" : 15, "value2" : 15 },
  { "_id" : 3, "value1" : 20, "value2" : 10 }
]
```

$collStats

버전 4.0의 새로운 기능

Amazon DocumentDB의 `$collStats` 집계 단계는 MongoDB 셸의 `db.collection.stats()` 명령과 마찬가지로 지정된 컬렉션에 대한 통계를 제공합니다. 이 단계는 문서 수, 컬렉션의 총 크기, 다양한 성능 지표와 같은 컬렉션에 대한 정보를 검색하는 데 사용할 수 있습니다.

파라미터

- `latencyStats`: (선택 사항) 지연 시간 통계 수집 옵션을 지정하는 문서입니다. 이 파라미터는 Amazon DocumentDB에서 지원되지 않습니다.
- `recordStats`: (선택 사항) 레코드 통계 수집 옵션을 지정하는 문서입니다. 이 파라미터는 Amazon DocumentDB에서 지원되지 않습니다.
- `queryExecStats`: (선택 사항) 쿼리 실행 통계 수집 옵션을 지정하는 문서입니다. 이 파라미터는 Amazon DocumentDB에서 지원되지 않습니다.

예제(MongoDB 쉘)

다음 예제에서는 `$collStats` 집계 단계를 사용하여 db 데이터베이스`test`에서 라는 컬렉션에 대한 통계를 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.test.insertMany([
  { "name": "John", "age": 30 },
  { "name": "Jane", "age": 25 },
  { "name": "Bob", "age": 35 }
]);
```

쿼리 예제

```
db.test.aggregate([
```

```
  { $collStats: {} }
]);
```

출력

```
{
  "ns" : "db.test",
  "count" : 3,
  "size" : 87,
  "avgObjSize" : 29,
  "storageSize" : 40960,
  "capped" : false,
  "nindexes" : 1,
  "totalIndexSize" : 8192,
  "indexSizes" : {
    "_id_" : 8192
  },
  "collScans" : 0,
  "idxScans" : 0,
  "opCounter" : {
    "numDocsIns" : 3,
    "numDocsUpd" : 0,
    "numDocsDel" : 0
  },
  "cacheStats" : {
    "collBlksHit" : 0,
    "collBlksRead" : 0,
    "collHitRatio" : 0,
    "idxBlksHit" : 0,
    "idxBlksRead" : 0,
    "idxHitRatio" : 0
  },
  "lastReset" : "2023-04-11T12:00:00Z",
  "ok" : 1,
  "operationTime" : Timestamp(1681206000, 1)
}
```

코드 예제

`$collStats` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

다음은 공식 MongoDB Node.js 드라이버를 사용하여 Node.js 애플리케이션에서 `$collStats` 집계 단계를 사용하는 방법의 예입니다.

```
const { MongoClient } = require('mongodb');

async function runCollStatsExample() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('db');
  const collection = db.collection('test');

  const result = await collection.aggregate([
    { $collStats: {} }
  ]).toArray();

  console.log(result);

  await client.close();
}

runCollStatsExample();
```

Python

다음은 PyMongo 드라이버를 사용하여 Python 애플리케이션에서 `$collStats` 집계 단계를 사용하는 방법의 예입니다.

```
from pymongo import MongoClient

def run_coll_stats_example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['db']
    collection = db['test']

    result = list(collection.aggregate([
        { '$collStats': {} }
    ]))
```

```
    print(result)

    client.close()

run_coll_stats_example()
```

$concat

Amazon DocumentDB의 $concat 집계 연산자는 문서의 여러 문자열을 연결(또는 결합)하여 애플리케이션에 반환할 수 있는 단일 문자열을 생성합니다. 이렇게 하면 문자열 조작이 데이터베이스 수준에서 수행되므로 애플리케이션에서 수행되는 작업이 줄어듭니다.

파라미터

- expression1: 연결할 첫 번째 문자열입니다.
- expression2: 연결할 두 번째 문자열입니다.
- ...: 연결할 추가 문자열(선택 사항).

예제(MongoDB 쉘)

이 예제에서는 사용자의 이름과 성을 연결하여 각 사용자의 전체 이름을 생성합니다.

샘플 문서 생성

```
db.people.insertMany([
  { "_id":1, "first_name":"Jane", "last_name":"Doe", "DOB":"2/1/1999", "Desk": "MSP102-
MN"},
  { "_id":2, "first_name":"John", "last_name":"Doe", "DOB":"12/21/1992", "Desk":
 "DSM301-IA"},
  { "_id":3, "first_name":"Steve", "last_name":"Smith", "DOB":"3/21/1981",
 "Desk":"MKE233-WI"}
])
```

쿼리 예제

```
db.people.aggregate([
  { $project: { full_name: { $concat: [ "$first_name", " ", "$last_name"] } } }
])
```

출력

```
{ "_id" : 1, "full_name" : "Jane Doe" }
{ "_id" : 2, "full_name" : "John Doe" }
{ "_id" : 3, "full_name" : "Steve Smith" }
```

코드 예제

$concat 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function concatenateNames() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');

  const result = await db.collection('people').aggregate([
    { $project: { full_name: { $concat: [ "$first_name", " ", "$last_name"] } } }
  ]).toArray();

  console.log(result);

  await client.close();
}

concatenateNames();
```

Python

```
from pymongo import MongoClient

def concatenate_names():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']

    result = list(db.people.aggregate([
```

```
        { '$project': { 'full_name': { '$concat': [ '$first_name', ' ',
  '$last_name' ] } } }
    ]))

    print(result)

    client.close()

concatenate_names()
```

$concatArrays

Amazon DocumentDB의 $concatArrays 집계 연산자는 두 개 이상의 배열을 단일 배열로 연결하는 데 사용됩니다. 이는 추가 처리 또는 분석을 위해 여러 데이터 배열을 단일 배열로 결합해야 할 때 유용할 수 있습니다.

파라미터

- array1: 연결할 첫 번째 배열입니다.
- array2: 연결할 두 번째 배열입니다.
- [array3, ...]: (선택 사항) 연결할 추가 배열입니다.

예제(MongoDB 쉘)

다음 예제에서는 $concatArrays 연산자를 사용하여 두 배열을 단일 배열로 결합하는 방법을 보여줍니다.

샘플 문서 생성

```
db.collection.insertMany([
  {
    "_id": 1,
    "name": "John Doe",
    "hobbies": ["reading", "swimming"],
    "skills": ["programming", "design"]
  },
  {
    "_id": 2,
    "name": "Jane Smith",
    "hobbies": ["hiking", "cooking"],
```

```
    "skills": ["marketing", "analysis"]
  }
]);
```

쿼리 예제

```
db.collection.aggregate([
  {
    $project: {
      _id: 0,
      name: 1,
      all_activities: { $concatArrays: ["$hobbies", "$skills"] }
    }
  }
]);
```

출력

```
[
  {
    "name": "John Doe",
    "all_activities": [
      "reading",
      "swimming",
      "programming",
      "design"
    ]
  },
  {
    "name": "Jane Smith",
    "all_activities": [
      "hiking",
      "cooking",
      "marketing",
      "analysis"
    ]
  }
]
```

코드 예제

$concatArrays 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('mydb');
  const collection = db.collection('mycollection');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 0,
        name: 1,
        all_activities: { $concatArrays: ['$hobbies', '$skills'] }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['mydb']
    collection = db['mycollection']

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 0,
                'name': 1,
```

```
                'all_activities': { '$concatArrays': ['$hobbies', '$skills'] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$cond

Amazon DocumentDB의 $cond 연산자는 조건 표현식을 평가하고 두 개의 가능한 결과 표현식 중 하나를 반환하는 데 사용됩니다.

파라미터

- `if`: 평가할 부울 표현식입니다.
- `then`: 표현식이 true인 경우 반환할 `if` 표현식입니다.
- `else`: 표현식이 false인 경우 반환할 `if` 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 연$cond산자를 사용하여 사람의 연령에 따라 값을 반환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.people.insertMany([
  { _id: 1, name: "John Doe", age: 35 },
  { _id: 2, name: "Jane Doe", age: 25 },
  { _id: 3, name: "Bob Smith", age: 65 }
]);
```

쿼리 예제

```
db.people.aggregate([
  {
    $project: {
      name: 1,
      ageGroup: {
```

```
        $cond: {
          if: { $lt: ["$age", 30] },
          then: "young",
          else: {
            $cond: {
              if: { $lt: ["$age", 65] },
              then: "middle-aged",
              else: "elderly"
            }
          }
        }
      }
    }
  }
])
```

출력

```
[
  { "_id" : 1, "name" : "John Doe", "ageGroup" : "middle-aged" },
  { "_id" : 2, "name" : "Jane Doe", "ageGroup" : "young" },
  { "_id" : 3, "name" : "Bob Smith", "ageGroup" : "elderly" }
]
```

코드 예제

$cond 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const result = await collection.aggregate([
    { $project: {
        name: 1,
        ageGroup: {
```

```
            $cond: {
                if: { $lt: ["$age", 30] },
                then: "young",
                else: {
                    $cond: {
                        if: { $lt: ["$age", 65] },
                        then: "middle-aged",
                        else: "elderly"
                    }
                }
            }
        }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient


def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.people

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'ageGroup': {
                    '$cond': {
                        'if': { '$lt': ["$age", 30]},
                        'then': "young",
                        'else': {
```

```
                            '$cond': {
                                'if': { '$lt': ["$age", 65]},
                                'then': "middle-aged",
                                'else': "elderly"
                            }
                        }
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$convert

버전 4.0의 새로운 기능

Amazon DocumentDB의 `$convert` 연산자는 값을 한 데이터 유형에서 다른 데이터 유형으로 변환하는 데 사용됩니다. 이 연산자는 문자열을 숫자로 변환하거나 날짜를 타임스탬프로 변환하는 등 다양한 유형의 데이터에 대한 작업을 수행해야 하는 경우에 유용합니다.

파라미터

- `to`: 값을 변환할 대상 데이터 유형입니다. 지원되는 값은 `"string"`, `"double"`, `"long"`, `"int"`, `"date"` 및 `"boolean"`입니다.
- `from`: 값의 현재 데이터 형식입니다. 지정하지 않으면 Amazon DocumentDB는 데이터 유형을 자동으로 감지하려고 시도합니다.
- `onError`: (선택 사항) 변환에 실패할 경우 반환할 값입니다. 특정 값이거나 `"null"`, `"zerofill"`또는 특수 값 중 하나일 수 있습니다`"error"`.
- `onNull`: (선택 사항) 입력 값이 인 경우 반환할 값입니다`null`. 특정 값이거나 `"null"`, `"zerofill"`또는 특수 값 중 하나일 수 있습니다`"error"`.

예제(MongoDB 쉘)

다음 예제에서는 `$convert` 연산자를 사용하여 문자열 값을 날짜로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { _id: 1, name: "John Doe", joinedOn: "2022-01-01" },
  { _id: 2, name: "Jane Smith", joinedOn: "2023-02-15" },
  { _id: 3, name: "Bob Johnson", joinedOn: "invalid date" }
]);
```

쿼리 예제

```
db.users.aggregate([
  {
    $project: {
      _id: 1,
      name: 1,
      joinedOn: {
        $convert: {
          input: "$joinedOn",
          to: "date",
          onError: "null",
          onNull: "null"
        }
      }
    }
  }
])
```

출력

```
[
  { "_id" : 1, "name" : "John Doe", "joinedOn" : ISODate("2022-01-01T00:00:00Z") },
  { "_id" : 2, "name" : "Jane Smith", "joinedOn" : ISODate("2023-02-15T00:00:00Z") },
  { "_id" : 3, "name" : "Bob Johnson", "joinedOn" : null }
]
```

코드 예제

$convert 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require("mongodb");
```

```
async function example() {
  const client = await MongoClient.connect("mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false");
  const db = client.db("test");
  const users = db.collection("users");

  const results = await users.aggregate([
    {
      $project: {
        _id: 1,
        name: 1,
        joinedOn: {
          $convert: {
            input: "$joinedOn",
            to: "date",
            onError: "null",
            onNull: "null"
          }
        }
      }
    }
  ]).toArray();

  console.log(results);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient("mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false")
    db = client["test"]
    users = db["users"]

    results = list(users.aggregate([
```

```
        {
            "$project": {
                "_id": 1,
                "name": 1,
                "joinedOn": {
                    "$convert": {
                        "input": "$joinedOn",
                        "to": "date",
                        "onError": "null",
                        "onNull": "null"
                    }
                }
            }
        }
    ]))

    print(results)
    client.close()

example()
```

$cos

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $cos 연산자는 라디안 단위로 측정된 값의 코사인을 반환합니다. 집계 파이프라인에서 이를 사용하여 숫자 필드에 대한 삼각 계산을 수행합니다.

파라미터

- expression: 라디안 단위의 숫자로 확인되는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $cos 연산자를 사용하여 각도 값의 코사인을 라디안 단위로 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.angles.insertMany([
  { "_id": 1, "angle": 0 },
```

```
  { "_id": 2, "angle": 1.0471975511965976 },
  { "_id": 3, "angle": 3.141592653589793 }
]);
```

쿼리 예제

```
db.angles.aggregate([
  { $project: {
    "cosine": { $cos: "$angle" }
  }}
]);
```

출력

```
[
  { "_id": 1, "cosine": 1 },
  { "_id": 2, "cosine": 0.5000000000000001 },
  { "_id": 3, "cosine": -1 }
]
```

코드 예제

$cos 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('angles');

  const result = await collection.aggregate([
    { $project: {
      "cosine": { $cos: "$angle" }
    }}
  ]).toArray();

  console.log(result);
```

```
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['angles']

    result = list(collection.aggregate([
        { '$project': {
            'cosine': { '$cos': '$angle' }
        }}
    ]))

    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$cosh

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $cosh 연산자는 값의 하이퍼볼릭 코사인을 반환합니다. 집계 파이프라인에서 이를 사용하여 숫자 필드에 대해 하이퍼볼릭 삼각형 계산을 수행합니다.

파라미터

- expression: 숫자로 확인되는 표현식입니다.

반환 유형은 double 기본적으로 입니다. 입력이 128비트 10진수인 경우 출력도 128비트 10진수입니다.

예제(MongoDB 쉘)

다음 예제에서는 $cosh 연산자를 사용하여 숫자 값의 하이퍼볼릭 코사인을 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 1 },
  { "_id": 3, "value": -1 }
]);
```

쿼리 예제

```
db.values.aggregate([
  { $project: {
    "result": { $cosh: "$value" }
  }}
]);
```

출력

```
[
  { "_id": 1, "result": 1 },
  { "_id": 2, "result": 1.5430806348152437 },
  { "_id": 3, "result": 1.5430806348152437 }
]
```

코드 예제

$cosh 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
```

```
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "result": { $cosh: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'result': { '$cosh': '$value' }
        }}
    ]))

    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$count(스테이지)

Amazon DocumentDB의 $count 집계 단계는가 단계로 전달하는 문서 수를 계산하는 데 사용됩니다. 집계 파이프라인의 최종 단계로 사용되어 이전 단계와 일치하는 총 문서 수를 반환하는 경우가 많습니다.

파라미터

- `field`: 계산할 필드입니다. 이 파라미터는 선택 사항이며 제공되지 않은 경우 스테이지는 총 입력 문서 수를 계산합니다.

예제(MongoDB 쉘)

다음 예제에서는 `$count` 스테이지를 사용하여 컬렉션의 총 문서 수를 가져오는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { name: "John", age: 30 },
  { name: "Jane", age: 25 },
  { name: "Bob", age: 35 },
  { name: "Alice", age: 28 }
]);
```

쿼리 예제

```
db.users.aggregate([
  { $count: "total" }
]);
```

출력

```
{ "total" : 4 }
```

이 예제에서는 `users` 컬렉션을 집계하고 `$count` 스테이지를 사용하여 총 문서 수를 계산합니다.

코드 예제

`$count` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function countDocuments() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
```

```
  const db = client.db('mydb');
  const collection = db.collection('users');

  const result = await collection.aggregate([
    { $count: "total" }
  ]).toArray();

  console.log(result[0].total);

  await client.close();
}

countDocuments();
```

Python

```
from pymongo import MongoClient

def count_documents():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['mydb']
    collection = db['users']

    result = list(collection.aggregate([
        { '$count': "total" }
    ]))

    print(result[0]['total'])

    client.close()

count_documents()
```

$count(누적기)

버전 8.0.1에서 새로 추가되었습니다.

$group 단계 내의 $count 누적기를 사용하여 각 그룹의 문서 수를 반환합니다. 빈 객체를 인수{}로 허용합니다.

이는 $count 파이프라인을 통과하는 모든 문서를 계산하는 독립 실행형 단계인 파이프라인 단계와 다릅니다. 대신 누적기는 $count에서 생성한 각 그룹 내의 문서를 계산합니다$group.

구문

```
{ $count: {} }
```

파라미터

- $count 누적기는 인수를 취하지 않습니다. 빈 객체를 수락{}하고 각 그룹의 문서 수를 반환합니다.

예제(MongoDB 쉘)

다음 예제에서는 $count 누적기를 사용하여 각 범주의 제품 수를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.products.insertMany([
  { name: "Widget", category: "A" },
  { name: "Gadget", category: "A" },
  { name: "Doohickey", category: "B" },
  { name: "Thingamajig", category: "B" },
  { name: "Whatsit", category: "B" }
])
```

쿼리 예제

```
db.products.aggregate([
  { $group: { _id: "$category", count: { $count: {} } } }
])
```

출력

```
[
  { "_id": "A", "count": 2 },
  { "_id": "B", "count": 3 }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

$count 누적기는 $setWindowFields 스테이지에서 창 연산자로도 사용할 수 있습니다. 이 컨텍스트에서는 각 창의 문서 수를 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

샘플 문서 생성

```
db.events.insertMany([
  { _id: 1, category: "A", day: 1 },
  { _id: 2, category: "A", day: 2 },
  { _id: 3, category: "A", day: 3 },
  { _id: 4, category: "B", day: 1 },
  { _id: 5, category: "B", day: 2 }
])
```

쿼리 예제

다음 예제에서는를 기준으로 문서를 분할하고category,를 기준으로 각 파티션을 정렬하고day, 파티션 시작부터 현재 문서까지 실행 중인 문서 수를 반환합니다.

```
db.events.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$category",
      sortBy: { day: 1 },
      output: {
        runningCount: {
          $count: {},
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
])
```

출력

```
[
  { "_id": 1, "category": "A", "day": 1, "runningCount": 1 },
  { "_id": 2, "category": "A", "day": 2, "runningCount": 2 },
  { "_id": 3, "category": "A", "day": 3, "runningCount": 3 },
  { "_id": 4, "category": "B", "day": 1, "runningCount": 1 },
  { "_id": 5, "category": "B", "day": 2, "runningCount": 2 }
```

```
]
```

각 문서는 현재 문서를 포함하여 파티션 내의 문서 runningCount수인 로 보강됩니다.

코드 예제

$count 누적기 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누산기 사용량()과 창 연산자 사용량($group)을 모두 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const db = client.db('test');

    // Accumulator usage: count per group
    const products = db.collection('products');
    await products.insertMany([
      { name: "Widget", category: "A" },
      { name: "Gadget", category: "A" },
      { name: "Doohickey", category: "B" },
      { name: "Thingamajig", category: "B" },
      { name: "Whatsit", category: "B" }
    ]);
    const accumulatorResult = await products.aggregate([
      { $group: { _id: "$category", count: { $count: {} } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Window operator usage: running count within each partition
    const events = db.collection('events');
    await events.insertMany([
      { _id: 1, category: "A", day: 1 },
      { _id: 2, category: "A", day: 2 },
```

```
      { _id: 3, category: "A", day: 3 },
      { _id: 4, category: "B", day: 1 },
      { _id: 5, category: "B", day: 2 }
    ]);
    const windowResult = await events.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$category",
          sortBy: { day: 1 },
          output: {
            runningCount: {
              $count: {},
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client['test']
```

```
        # Accumulator usage: count per group
        products = db['products']
        products.insert_many([
            { 'name': 'Widget', 'category': 'A' },
            { 'name': 'Gadget', 'category': 'A' },
            { 'name': 'Doohickey', 'category': 'B' },
            { 'name': 'Thingamajig', 'category': 'B' },
            { 'name': 'Whatsit', 'category': 'B' }
        ])
        accumulator_result = list(products.aggregate([
            { '$group': { '_id': '$category', 'count': { '$count': {} } } }
        ]))
        pprint(accumulator_result)

        # Window operator usage: running count within each partition
        events = db['events']
        events.insert_many([
            { '_id': 1, 'category': 'A', 'day': 1 },
            { '_id': 2, 'category': 'A', 'day': 2 },
            { '_id': 3, 'category': 'A', 'day': 3 },
            { '_id': 4, 'category': 'B', 'day': 1 },
            { '_id': 5, 'category': 'B', 'day': 2 }
        ])
        window_result = list(events.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$category',
                    'sortBy': { 'day': 1 },
                    'output': {
                        'runningCount': {
                            '$count': {},
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        pprint(window_result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
```

```
            client.close()

example()
```

$covariancePop

버전 8.0.2에서 새로 추가되었습니다.

Amazon DocumentDB의 $covariancePop 연산자는 두 숫자 표현식의 모집단 공분산을 반환합니다. $setWindowFields 단계에서만 사용되는 창 연산자이며 $group 또는 $bucket 단계에서는 유효하지 않습니다. 단계의 output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

파라미터

- $covariancePop는 2개 요소 배열을 사용합니다. [<expression1>, <expression2>]여기서 각 요소는 숫자 표현식("$x", 표현식 또는 상수와 같은 필드 경로)입니다. 창의 문서에 걸쳐 두 표현식의 모집단 공분산을 반환합니다.

예제(MongoDB 쉘)

다음 예제에서는 문서를 기준으로 분할하고series, 각 파티션을 기준으로 정렬하고t, 전체 파티션에 y 대한 x 및의 모집단 공분산을 계산합니다.

샘플 문서 생성

```
db.measurements.insertMany([
  { _id: 1, series: "A", t: 1, x: 1, y: 2 },
  { _id: 2, series: "A", t: 2, x: 2, y: 5 },
  { _id: 3, series: "A", t: 3, x: 3, y: 8 },
  { _id: 4, series: "B", t: 1, x: 1, y: 8 },
  { _id: 5, series: "B", t: 2, x: 2, y: 5 },
  { _id: 6, series: "B", t: 3, x: 3, y: 2 }
]);
```

쿼리 예제

```
db.measurements.aggregate([
  {
    $setWindowFields: {
```

```
      partitionBy: "$series",
      sortBy: { t: 1 },
      output: {
        covariance: {
          $covariancePop: [ "$x", "$y" ],
          window: { documents: ["unbounded", "unbounded"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "series": "A", "t": 1, "x": 1, "y": 2, "covariance": 2 },
  { "_id": 2, "series": "A", "t": 2, "x": 2, "y": 5, "covariance": 2 },
  { "_id": 3, "series": "A", "t": 3, "x": 3, "y": 8, "covariance": 2 },
  { "_id": 4, "series": "B", "t": 1, "x": 1, "y": 8, "covariance": -2 },
  { "_id": 5, "series": "B", "t": 2, "x": 2, "y": 5, "covariance": -2 },
  { "_id": 6, "series": "B", "t": 3, "x": 3, "y": 2, "covariance": -2 }
]
```

파티션의 모든 문서는 시리즈(x x 및 y가 함께 증가) 및 시리즈A(x가 증가2하면 By가 감소)와 같이 동일한 covariance, 즉 해당 파티션의 y 모든 문서에 -2 대한 및의 모집단 공분산을 수신합니다.

코드 예제

$setWindowFields 단계에서 $covariancePop 연산자를 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
```

```
    const db = client.db('test');
    const measurements = db.collection('measurements');

    await measurements.insertMany([
      { _id: 1, series: "A", t: 1, x: 1, y: 2 },
      { _id: 2, series: "A", t: 2, x: 2, y: 5 },
      { _id: 3, series: "A", t: 3, x: 3, y: 8 },
      { _id: 4, series: "B", t: 1, x: 1, y: 8 },
      { _id: 5, series: "B", t: 2, x: 2, y: 5 },
      { _id: 6, series: "B", t: 3, x: 3, y: 2 }
    ]);

    const result = await measurements.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$series",
          sortBy: { t: 1 },
          output: {
            covariance: {
              $covariancePop: [ "$x", "$y" ],
              window: { documents: ["unbounded", "unbounded"] }
            }
          }
        }
      }
    ]).toArray();

    console.log(result);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
    try:
        db = client['test']
        measurements = db['measurements']

        measurements.insert_many([
            { '_id': 1, 'series': 'A', 't': 1, 'x': 1, 'y': 2 },
            { '_id': 2, 'series': 'A', 't': 2, 'x': 2, 'y': 5 },
            { '_id': 3, 'series': 'A', 't': 3, 'x': 3, 'y': 8 },
            { '_id': 4, 'series': 'B', 't': 1, 'x': 1, 'y': 8 },
            { '_id': 5, 'series': 'B', 't': 2, 'x': 2, 'y': 5 },
            { '_id': 6, 'series': 'B', 't': 3, 'x': 3, 'y': 2 }
        ])

        result = list(measurements.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$series',
                    'sortBy': { 't': 1 },
                    'output': {
                        'covariance': {
                            '$covariancePop': [ '$x', '$y' ],
                            'window': { 'documents': ['unbounded', 'unbounded'] }
                        }
                    }
                }
            }
        ]))

        print(result)
    finally:
        client.close()

example()
```

$covarianceSamp

버전 8.0.2에서 새로 추가되었습니다.

Amazon DocumentDB의 $covarianceSamp 연산자는 두 숫자 표현식의 샘플 공분산을 반환합니다. $setWindowFields 단계에서만 사용되는 창 연산자이며 $group 또는 $bucket 단계에서는 유효

하지 않습니다. 단계의 output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

파라미터

- $covarianceSamp는 2개 요소 배열을 사용합니다. [<expression1>, <expression2>]여기서 각 요소는 숫자 표현식("$x", 표현식 또는 상수와 같은 필드 경로)입니다. 창의 문서에 걸쳐 두 표현식의 샘플 공분산을 반환합니다.

예제(MongoDB 쉘)

다음 예제에서는를 기준으로 문서를 분할하고series,를 기준으로 각 파티션을 정렬하고t, 전체 파티션에 y 대한 x 및의 샘플 공분산을 계산합니다.

샘플 문서 생성

```
db.measurements.insertMany([
  { _id: 1, series: "A", t: 1, x: 1, y: 2 },
  { _id: 2, series: "A", t: 2, x: 2, y: 5 },
  { _id: 3, series: "A", t: 3, x: 3, y: 8 },
  { _id: 4, series: "B", t: 1, x: 1, y: 8 },
  { _id: 5, series: "B", t: 2, x: 2, y: 5 },
  { _id: 6, series: "B", t: 3, x: 3, y: 2 }
]);
```

쿼리 예제

```
db.measurements.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$series",
      sortBy: { t: 1 },
      output: {
        covariance: {
          $covarianceSamp: [ "$x", "$y" ],
          window: { documents: ["unbounded", "unbounded"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "series": "A", "t": 1, "x": 1, "y": 2, "covariance": 3 },
  { "_id": 2, "series": "A", "t": 2, "x": 2, "y": 5, "covariance": 3 },
  { "_id": 3, "series": "A", "t": 3, "x": 3, "y": 8, "covariance": 3 },
  { "_id": 4, "series": "B", "t": 1, "x": 1, "y": 8, "covariance": -3 },
  { "_id": 5, "series": "B", "t": 2, "x": 2, "y": 5, "covariance": -3 },
  { "_id": 6, "series": "B", "t": 3, "x": 3, "y": 2, "covariance": -3 }
]
```

파티션의 모든 문서는 시리즈(x x 및 y가 함께 증가) 및 3 시리즈A(x가 증가하면 By가 감소)와 같이 동일한 covariance, 즉 해당 파티션의 y 모든 문서에 -3 대한 및의 샘플 공분산을 수신합니다. 샘플 공분산은 n - 1 대신 로 나뉘n므로 동일한 데이터의 모집단 공분산보다 크기가 더 큽니다.

코드 예제

$setWindowFields 단계에서 $covarianceSamp 연산자를 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const measurements = db.collection('measurements');

    await measurements.insertMany([
      { _id: 1, series: "A", t: 1, x: 1, y: 2 },
      { _id: 2, series: "A", t: 2, x: 2, y: 5 },
      { _id: 3, series: "A", t: 3, x: 3, y: 8 },
      { _id: 4, series: "B", t: 1, x: 1, y: 8 },
      { _id: 5, series: "B", t: 2, x: 2, y: 5 },
      { _id: 6, series: "B", t: 3, x: 3, y: 2 }
    ]);
```

```
    const result = await measurements.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$series",
          sortBy: { t: 1 },
          output: {
            covariance: {
              $covarianceSamp: [ "$x", "$y" ],
              window: { documents: ["unbounded", "unbounded"] }
            }
          }
        }
      }
    ]).toArray();

    console.log(result);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']
        measurements = db['measurements']

        measurements.insert_many([
            { '_id': 1, 'series': 'A', 't': 1, 'x': 1, 'y': 2 },
            { '_id': 2, 'series': 'A', 't': 2, 'x': 2, 'y': 5 },
            { '_id': 3, 'series': 'A', 't': 3, 'x': 3, 'y': 8 },
            { '_id': 4, 'series': 'B', 't': 1, 'x': 1, 'y': 8 },
            { '_id': 5, 'series': 'B', 't': 2, 'x': 2, 'y': 5 },
            { '_id': 6, 'series': 'B', 't': 3, 'x': 3, 'y': 2 }
```

```
        ])

        result = list(measurements.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$series',
                    'sortBy': { 't': 1 },
                    'output': {
                        'covariance': {
                            '$covarianceSamp': [ '$x', '$y' ],
                            'window': { 'documents': ['unbounded', 'unbounded'] }
                        }
                    }
                }
            }
        ]))

        print(result)
    finally:
        client.close()

example()
```

$currentOp

$currentOp 집계 단계에서는 데이터베이스에서 현재 실행 중인 작업에 대한 정보를 반환합니다. 이 단계는 집계 파이프라인에서 활성 쿼리 및 작업을 모니터링하는 데 유용합니다.

파라미터

- allUsers (선택 사항): 로 설정하면 모든 사용자에 대한 작업을 true반환합니다. 기본값은 false입니다.
- idleConnections (선택 사항): 로 설정하면에 유휴 연결이 true포함됩니다. 기본값은 false입니다.
- idleCursors (선택 사항): 로 설정하면 유휴 커서에 대한 정보가 true포함됩니다. 기본값은 false입니다.
- idleSessions (선택 사항): 로 설정하면 유휴 세션에 대한 정보가 true포함됩니다. 기본값은 true입니다.
- localOps (선택 사항): 로 설정하면에 로컬 작업이 true포함됩니다. 기본값은 false입니다.

예제(MongoDB 쉘)

다음 예제에서는 $currentOp 집계 단계를 사용하여 활성 읽기 작업에 대한 정보를 검색하는 방법을 보여줍니다.

쿼리 예제

```
db.aggregate([
  { $currentOp: { allUsers: true, idleConnections: false } },
  { $match: { op: "query" } }
])
```

출력

```
[
  {
    "opid": "12345",
    "active": true,
    "op": "query",
    "ns": "test.users",
    "secs_running": 2
  }
]
```

코드 예제

$currentOp 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('admin');

  const result = await db.aggregate([
    { $currentOp: { allUsers: true, idleConnections: false } },
    { $match: { op: "query" } }
  ]).toArray();
```

```
  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['admin']

    result = list(db.aggregate([
        { '$currentOp': { 'allUsers': True, 'idleConnections': False } },
        { '$match': { 'op': 'query' } }
    ]))

    print(result)
    client.close()

example()
```

$dateAdd

버전 5.0의 새로운 기능

Amazon DocumentDB의 $dateAdd 집계 연산자를 사용하면 날짜 및 시간 값에 기간을 추가할 수 있습니다.

파라미터

- date: 기간을 추가할 날짜 및 시간 값입니다.
- duration: date 값에 추가할 기간입니다. , years, , , months, dayshours, weeks에 대한 키가 있는 객체로 지정할 수 minutes있습니다seconds.
- timezone: (선택 사항) 날짜 추가를 수행할 때 사용할 시간대입니다. 지정하지 않으면 Amazon DocumentDB 클러스터의 기본 시간대가 사용됩니다.

예제(MongoDB 쉘)

다음 예제에서는 $dateAdd 연산자를 사용하여 날짜에 2일 및 12시간을 추가하는 방법을 보여줍니다.

샘플 문서 생성

```
db.events.insertMany([
  { _id: 1, eventDate: ISODate("2023-04-01T10:00:00Z") },
  { _id: 2, eventDate: ISODate("2023-04-02T12:00:00Z") },
  { _id: 3, eventDate: ISODate("2023-04-03T14:00:00Z") }
]);
```

쿼리 예제

```
db.events.aggregate([
  {
    $project: {
      _id: 1,
      eventDate: 1,
      eventDatePlustwodaysandtwelvehours: {
        $dateAdd: {
          startDate: {
            $dateAdd: {
              startDate: "$eventDate",
              unit: "day",
              amount: 2
            }
          },
          unit: "hour",
          amount: 12
        }
      }
    }
  }
])
```

출력

```
[
  {
    "_id": 1,
    "eventDate": "2023-04-01T10:00:00Z",
```

```
    "eventDatePlustwodaysandtwelvehours": ISODate("2023-04-03T22:00:00Z)"
  },
  {
    "_id": 2,
    "eventDate": "2023-04-02T12:00:00Z",
    "eventDatePlustwodaysandtwelvehours": ISODate("2023-04-05T00:00:00Z)"
  },
  {
    "_id": 3,
    "eventDate": "2023-04-03T14:00:00Z",
    "eventDatePlustwodaysandtwelvehours": ISODate("2023-04-06T02:00:00Z)"
  }
]
```

코드 예제

$dateAdd 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('events');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        eventDate: 1,
        eventDatePlustwodaysandtwelvehours: {
          $dateAdd: {
            startDate: {
              $dateAdd: {
                startDate: "$eventDate",
                unit: "day",
                amount: 2
              }
            },
            unit: "hour",
```

```
            amount: 12
          }
        }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient
from bson.date_time import datetime

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['events']

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'eventDate': 1,
                'eventDatePlustwodaysandtwelvehours': {
                    '$dateAdd': {
                        'startDate': {
                            '$dateAdd' : {
                                'startDate': '$eventDate',
                                'unit': 'day',
                                'amount': 2,
                            }
                        },
                        'unit': 'hour',
                        'amount': 12,
                    }
```

```
                }
            }
        }
    ]))

    print(result)

    client.close()

example()
```

$dateDiff

버전 5.0의 새로운 기능

Elastic 클러스터에서는 지원되지 않습니다.

`$dateDiff` 집계 연산자는 지정된 단위로 두 날짜 간의 차이를 계산합니다. 시작 날짜와 종료 날짜 사이에 교차된 단위 경계 수를 반환합니다.

파라미터

- `startDate`: 시작 날짜 표현식입니다.
- `endDate`: 종료 날짜 표현식입니다.
- `unit`: 차이의 시간 단위입니다. 지원되는 단위는 `year`, `quarter`, `month`, `week`, `day`, `hour`, `minute`, 및 `second`입니다`millisecond`.

예제(MongoDB 쉘)

다음 예제에서는 `$dateDiff` 연산자를 사용하여 주문 배치와 배송 사이의 일수를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.shipments.insertMany([
  {
    orderId: 1001,
    orderDate: ISODate("2025-01-10T08:00:00Z"),
    deliveryDate: ISODate("2025-01-15T14:30:00Z")
  },
```

```
  {
    orderId: 1002,
    orderDate: ISODate("2025-02-05T10:00:00Z"),
    deliveryDate: ISODate("2025-02-12T16:45:00Z")
  }
]);
```

쿼리 예제

```
db.shipments.aggregate([
  {
    $project: {
      orderId: 1,
      orderDate: 1,
      deliveryDate: 1,
      daysToDeliver: {
        $dateDiff: {
          startDate: "$orderDate",
          endDate: "$deliveryDate",
          unit: "day"
        }
      }
    }
  }
]);
```

출력

```
[
  {
    _id: ObjectId('6924a5f2d66dcae121d29517'),
    orderId: 1001,
    orderDate: ISODate('2025-01-10T08:00:00.000Z'),
    deliveryDate: ISODate('2025-01-15T14:30:00.000Z'),
    daysToDeliver: 5
  },
  {
    _id: ObjectId('6924a5f2d66dcae121d29518'),
    orderId: 1002,
    orderDate: ISODate('2025-02-05T10:00:00.000Z'),
    deliveryDate: ISODate('2025-02-12T16:45:00.000Z'),
    daysToDeliver: 7
  }
```

```
]
```

코드 예제

$dateDiff 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const shipments = db.collection('shipments');

  const result = await shipments.aggregate([
    {
      $project: {
        orderId: 1,
        orderDate: 1,
        deliveryDate: 1,
        daysToDeliver: {
          $dateDiff: {
            startDate: "$orderDate",
            endDate: "$deliveryDate",
            unit: "day"
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient
```

```
def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    shipments = db['shipments']

    result = list(shipments.aggregate([
        {
            "$project": {
                "orderId": 1,
                "orderDate": 1,
                "deliveryDate": 1,
                "daysToDeliver": {
                    "$dateDiff": {
                        "startDate": "$orderDate",
                        "endDate": "$deliveryDate",
                        "unit": "day"
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$dateFromParts

버전 5.0.1 및 8.0의 새로운 기능

Amazon DocumentDB의 `$dateFromParts` 집계 연산자는 연도, 월, 일, 시간, 분, 초, 밀리초와 같은 구성 요소에서 날짜 및 시간 값을 구성하고 반환합니다.

파라미터

- `year`: 역년입니다. ISO 8601 부분을 사용하지 않을 때 필요합니다.
- `month`, `day`, `hour`, `minute`, `second`, `millisecond`: (선택 사항) 나머지 날짜 및 시간 구성 요소입니다. 생략된 구성 요소는 기본적으로 가장 낮은 유효 값으로 설정됩니다.

- `isoWeekYear`, `isoWeek`, `isoDayOfWeek`: (선택 사항) ISO 8601 날짜 구성 요소입니다. ISO 8601 주 날짜에서 날짜를 생성day하려면 `yearmonth`, 및 대신이 옵션을 사용합니다.
- `timezone`: (선택 사항) 날짜를 생성할 때 사용할 시간대입니다. 지정하지 않으면 UTC가 사용됩니다.

예제(MongoDB 쉘)

다음 예제에서는 `$dateFromParts` 연산자를 사용하여 개별 구성 요소에서 날짜를 구성하는 방법을 보여줍니다.

샘플 문서 생성

```
db.parts.insertMany([
  { _id: 1, y: 2023, m: 4, d: 1 },
  { _id: 2, y: 2023, m: 12, d: 25 }
]);
```

쿼리 예제

```
db.parts.aggregate([
  {
    $project: {
      _id: 1,
      constructedDate: {
        $dateFromParts: { year: "$y", month: "$m", day: "$d" }
      }
    }
  }
])
```

출력

```
[
  { "_id": 1, "constructedDate": ISODate("2023-04-01T00:00:00Z") },
  { "_id": 2, "constructedDate": ISODate("2023-12-25T00:00:00Z") }
]
```

코드 예제

`$dateFromParts` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('parts');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        constructedDate: {
          $dateFromParts: { year: "$y", month: "$m", day: "$d" }
        }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['parts']

    result = list(collection.aggregate([
        {
            '$project': {
```

```
                '_id': 1,
                'constructedDate': {
                    '$dateFromParts': { 'year': '$y', 'month': '$m', 'day': '$d' }
                }
            }
        }
    ]))

    print(result)

    client.close()

example()
```

$dateFromString

Amazon DocumentDB의 `$dateFromString` 집계 연산자를 사용하면 날짜-시간 문자열을 날짜 객체로 변환할 수 있습니다. 이는 데이터가 문자열로 저장되지만 데이터에 대해 날짜 기반 작업을 수행해야 하는 경우에 유용합니다.

파라미터

- `dateString`: 날짜 및 시간을 나타내는 문자열입니다.
- `format`: (선택 사항) 형식을 지정하는 문자열입니다`dateString`. 제공되지 않으면 Amazon DocumentDB는 ISO-8601 형식으로 문자열을 구문 분석하려고 시도합니다.
- `timezone`: (선택 사항) 시간대를 지정하는 문자열입니다. 제공되지 않은 경우 Amazon DocumentDB는 서버의 시간대를 사용합니다.
- `onError`: (선택 사항) 변환이 실패할 경우 수행할 작업을 지정합니다. 가능한 값은 `'error'` (오류를 발생시키는 기본값), `'null'` (를 반환) 또는 `'replace'` (값을 `onErrorMessage` 옵션에 지정된 대체 문자열로 대체`null`)입니다.
- `onErrorMessage`: (선택 사항) `onError`이 로 설정된 경우 `'replace'`이 옵션은 대체 문자열을 지정합니다.

예제(MongoDB 쉘)

다음 예제에서는 `$dateFromString`를 사용하여 Amazon DocumentDB에서 날짜 문자열을 날짜 객체로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.missionLog.insertMany([
{ _id: 1, event: "missionStart", logDate: "2020-03-15T13:41:33"},
{ _id: 2, event: "jumpPoint1", logDate: "2020-03-15T13:45:34"},
{ _id: 3, event: "jumpPoint2", logDate: "2020-03-15T13:48:21"},
{ _id: 4, event: "jumpPoint3", logDate: "2020-03-15T13:52:09"},
{ _id: 5, event: "missionEnd", logDate: "2020-03-15T13:58:44"}
]);
```

쿼리 예제

```
db.missionLog.aggregate([
  {
    $project: {
      event: '$event',
      logDate: {
        $dateFromString: {
          dateString: '$logDate'
        }
      }
    }
  }
]);
```

출력

```
[
  {
    "_id": 1,
    "event": "missionStart",
    "logDate": ISODate("2020-03-15T13:41:33Z")
  },
  {
    "_id": 2,
    "event": "jumpPoint1",
    "logDate": ISODate("2020-03-15T13:45:34Z")
  },
  {
    "_id": 3,
    "event": "jumpPoint2",
    "logDate": ISODate("2020-03-15T13:48:21Z")
  },
  {
```

```
    "_id": 4,
    "event": "jumpPoint3",
    "logDate": ISODate("2020-03-15T13:52:09Z")
  },
  {
    "_id": 5,
    "event": "missionEnd",
    "logDate": ISODate("2020-03-15T13:58:44Z")
  }
]
```

코드 예제

$dateFromString 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('missionLog');

  const result = await collection.aggregate([
    {
      $project: {
        event: '$event',
        logDate: {
          $dateFromString: {
            dateString: '$logDate'
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}
```

```
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['missionLog']

    result = list(collection.aggregate([
        {
            '$project': {
                'event': '$event',
                'logDate': {
                    '$dateFromString': {
                        'dateString': '$logDate'
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$dateSubtract

버전 5.0의 새로운 기능

Amazon DocumentDB의 `$dateSubtract` 집계 연산자를 사용하면 날짜 값에서 지정된 기간을 뺄 수 있습니다.

파라미터

- `date`: 날짜 또는 타임스탬프로 확인되는 날짜 표현식입니다.
- `subtrahend`: 표현식에서 뺄 시간을 지정하는 기간 `date` 표현식입니다.

- unit: subtrahend 표현식의 시간 단위를 지정하는 문자열입니다. 지원되는 단위는 "연도", "분기", "월", "주", "일", "시간", "분", "초", "밀리초"입니다.

예제(MongoDB 쉘)

다음 예제에서는 $dateSubtract 연산자를 사용하여 현재 날짜로부터 1년 전의 날짜를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.events.insertOne({
  eventName: "Player joined",
  eventTime: ISODate("2023-04-01T12:00:00Z")
});
```

쿼리 예제

```
db.events.aggregate([
  {
    $project: {
      eventName: 1,
      oneYearAgo: {
        $dateSubtract: {
          startDate: "$eventTime",
          amount: 1,
          unit: "year"
        }
      }
    }
  }
])
```

출력

```
{
  "_id" : ObjectId("64567890abcdef012345678"),
  "eventName" : "Player joined",
  "oneYearAgo" : ISODate("2022-04-01T12:00:00Z")
}
```

코드 예제

$dateSubtract 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const events = db.collection('events');

  const result = await events.aggregate([
    {
      $project: {
        eventName: 1,
        oneYearAgo: {
          $dateSubtract: {
            startDate: "$eventTime",
            amount: 1,
            unit: "year"
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient
from bson.date_time import datetime

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    events = db['events']

    result = list(events.aggregate([
        {
            '$project': {
                'eventName': 1,
                'oneYearAgo': {
                    '$dateSubtract': {
                        'startDate': '$eventTime',
                        'amount': 1,
                        'unit': 'year'
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$dateToParts

버전 5.0.1 및 8.0의 새로운 기능

Amazon DocumentDB의 $dateToParts 집계 연산자는 연도, 월, 일, 시간, 분, 초, 밀리초와 같은 날짜 및 시간 값의 구성 요소가 포함된 문서를 반환합니다.

파라미터

- date: 여러 부분으로 나눌 날짜 및 시간 값입니다.
- timezone: (선택 사항) 날짜 부분을 추출할 때 사용할 시간대입니다. 지정하지 않으면 파트가 UTC로 반환됩니다.
- iso8601: (선택 사항) 로 설정하면 반환true된 문서는 ISO 8601 날짜 부분(isoWeekYear, isoWeek,)을 사용합니다isoDayOfWeek. 기본값은 false입니다.

예제(MongoDB 쉘)

다음 예제에서는 $dateToParts 연산자를 사용하여 날짜의 개별 부분을 반환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.events.insertMany([
  { _id: 1, eventDate: ISODate("2023-04-01T10:30:15Z") },
  { _id: 2, eventDate: ISODate("2023-12-25T18:45:00Z") }
]);
```

쿼리 예제

```
db.events.aggregate([
  {
    $project: {
      _id: 1,
      parts: { $dateToParts: { date: "$eventDate" } }
    }
  }
])
```

출력

```
[
  {
    "_id": 1,
    "parts": { "year": 2023, "month": 4, "day": 1, "hour": 10, "minute": 30, "second":
 15, "millisecond": 0 }
  },
  {
    "_id": 2,
    "parts": { "year": 2023, "month": 12, "day": 25, "hour": 18, "minute": 45,
 "second": 0, "millisecond": 0 }
  }
]
```

코드 예제

$dateToParts 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('events');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        parts: { $dateToParts: { date: "$eventDate" } }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['events']

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'parts': { '$dateToParts': { 'date': '$eventDate' } }
```

```
            }
        }
    ]))

    print(result)

    client.close()

example()
```

$dateToString

Amazon DocumentDB의 $dateToString 집계 연산자는 날짜 또는 타임스탬프 값을 문자열 표현으로 변환하는 데 사용됩니다. 이는 표시 또는 추가 처리를 위해 특정 방식으로 날짜 및 시간을 포맷해야 하는 경우에 유용합니다.

파라미터

- date: 문자열로 변환할 날짜 또는 타임스탬프 값입니다.
- format: 날짜를 표시해야 하는 형식을 지정하는 문자열입니다. 형식 문자열에는 4자리 연도의 %Y 경우 , 2자리 월의 %m 경우 , %d2자리 월의 경우와 같은 다양한 형식 지정자가 포함될 수 있습니다.
- timezone: (선택 사항) 변환에 사용할 시간대입니다. 지정하지 않으면 Amazon DocumentDB 클러스터를 호스팅하는 서버의 시간대가 사용됩니다.
- onNull: (선택 사항) date 파라미터가 인 경우 반환되는 값입니다null.

예제(MongoDB 쉘)

다음 예제에서는 연$dateToString산자를 사용하여 missionLog 컬렉션의 logDate 필드를 포맷하는 방법을 보여줍니다.

샘플 문서 생성

```
db.missionLog.insertMany([
  { _id: 1, "event":"missionStart", logDate: new Date("2020-03-15T13:41:33Z") },
  { _id: 2, "event":"jumpPoint1", logDate: new Date("2020-03-15T13:45:34Z") },
  { _id: 3, "event":"jumpPoint2", logDate: new Date("2020-03-15T13:48:21Z") },
  { _id: 4, "event":"jumpPoint3", logDate: new Date("2020-03-15T13:52:09Z") },
  { _id: 5, "event":"missionEnd", logDate: new Date("2020-03-15T13:58:44Z") }
]);
```

쿼리 예제

```
db.missionLog.aggregate([
  {
    $project: {
      event: "$event",
      logDateFormatted: {
        $dateToString: {
          format: "%Y-%m-%d %H:%M:%S",
          date: "$logDate"
        }
      }
    }
  }
])
```

출력

```
[
  {
    "_id": 1,
    "event": "missionStart",
    "logDateFormatted": "2020-03-15 13:41:33"
  },
  {
    "_id": 2,
    "event": "jumpPoint1",
    "logDateFormatted": "2020-03-15 13:45:34"
  },
  {
    "_id": 3,
    "event": "jumpPoint2",
    "logDateFormatted": "2020-03-15 13:48:21"
  },
  {
    "_id": 4,
    "event": "jumpPoint3",
    "logDateFormatted": "2020-03-15 13:52:09"
  },
  {
    "_id": 5,
    "event": "missionEnd",
    "logDateFormatted": "2020-03-15 13:58:44"
```

```
  }
]
```

코드 예제

$dateToString 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

다음은 Node.js 애플리케이션에서 $dateToString 연산자를 사용하는 예입니다.

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('missionLog');

  const result = await collection.aggregate([
    {
      $project: {
        event: "$event",
        logDateFormatted: {
          $dateToString: {
            format: "%Y-%m-%d %H:%M:%S",
            date: "$logDate"
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

다음은 Python 애플리케이션에서 $dateToString 연산자를 사용하는 예입니다.

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['missionLog']

    pipeline = [
        {
            '$project': {
                'event': '$event',
                'logDateFormatted': {
                    '$dateToString': {
                        'format': '%Y-%m-%d %H:%M:%S',
                        'date': '$logDate'
                    }
                }
            }
        }
    ]

    result = list(collection.aggregate(pipeline))
    print(result)
    client.close()

example()
```

$dateTrunc

버전 8.0의 새로운 기능

Elastic 클러스터에서는 지원되지 않습니다.

Amazon DocumentDB의 `$dateTrunc` 집계 연산자는 날짜를 지정된 단위로 잘라냅니다.

파라미터

- `date`: 날짜 또는 타임스탬프로 확인되는 날짜 표현식입니다.
- `unit`: 하위 추적 표현식의 시간 단위를 지정하는 문자열입니다. 지원되는 단위는 `year`, `quarter`, `month`, `week`, `day`, `hour`, `minute`, 및 `second`입니다`millisecond`.

예제(MongoDB 쉘)

다음 예제에서는 $dateTrunc 연산자를 사용하여 날짜를 시간으로 자르는 방법을 보여줍니다.

샘플 문서 생성

```
db.events.insertMany([
  {
    eventName: "Event 1",
    eventTime: ISODate("2025-04-01T12:15:00Z")
  },
  {
    eventName: "Event 2",
    eventTime: ISODate("2025-08-15T14:33:22Z")
  },
]);
```

쿼리 예제

```
db.events.aggregate([
  {
    $project: {
      eventName: 1,
      eventTime: 1,
      truncatedToHour: {
        $dateTrunc: {
          date: "$eventTime",
          unit: "hour"
        }
      }
    }
  }
]);
```

출력

```
[
  {
    _id: ObjectId('6924a258d66dcae121d29515'),
    eventName: 'Event 1',
    eventTime: ISODate('2025-04-01T12:15:00.000Z'),
    truncatedToHour: ISODate('2025-04-01T12:00:00.000Z')
  },
```

```
  {
    _id: ObjectId('6924a258d66dcae121d29516'),
    eventName: 'Event 2',
    eventTime: ISODate('2025-08-15T14:33:22.000Z'),
    truncatedToHour: ISODate('2025-08-15T14:00:00.000Z')
  }
]
```

코드 예제

$dateTrunc 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const events = db.collection('events');

  const result = await events.aggregate([
    {
      $project: {
        eventName: 1,
        eventTime: 1,
        truncatedToHour: {
          $dateTrunc: {
            date: "$eventTime",
            unit: "hour"
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    events = db['events']

    result = list(events.aggregate([
        {
            "$project": {
                "eventName": 1,
                "eventTime": 1,
                "truncatedToHour": {
                    "$dateTrunc": {
                        "date": "$eventTime",
                        "unit": "hour"
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$dayOfMonth

Amazon DocumentDB의 `$dayOfMonth` 집계 연산자는 지정된 날짜의 월중 날짜(1~31)를 검색합니다. 이 연산자는 문서의 날짜 필드에서 월의 날짜를 그룹화, 필터링 또는 추출하는 데 유용합니다.

파라미터

- `date expression`: 날짜 표현식은 문서의 날짜 필드, 날짜 객체 또는 날짜 문자열일 수 있습니다.

예제(MongoDB 쉘)

이 예제에서는 $dayOfMonth 연산자를 사용하여 문서의 날짜 필드에서 월의 날짜를 추출하는 방법을 보여줍니다.

샘플 문서 생성

```
db.events.insertMany([
  { _id: 1, eventDate: new Date("2022-01-15T12:00:00Z") },
  { _id: 2, eventDate: new Date("2022-02-28T15:30:00Z") },
  { _id: 3, eventDate: new Date("2022-03-01T09:45:00Z") },
  { _id: 4, eventDate: new Date("2022-04-30T23:59:59Z") }
]);
```

쿼리 예제

```
db.events.aggregate([
  { $project: {
    eventDay: { $dayOfMonth: "$eventDate" }
  }}
])
```

출력

```
[
  { "_id" : 1, "eventDay" : 15 },
  { "_id" : 2, "eventDay" : 28 },
  { "_id" : 3, "eventDay" : 1 },
  { "_id" : 4, "eventDay" : 30 }
]
```

코드 예제

$dayOfMonth 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const events = db.collection('events');

  const result = await events.aggregate([
    { $project: {
      eventDay: { $dayOfMonth: "$eventDate" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.mydatabase
    events = db.events

    result = list(events.aggregate([
        { "$project": {
            "eventDay": { "$dayOfMonth": "$eventDate" }
        }}
    ]))

    print(result)
    client.close()

example()
```

$dayOfWeek

Amazon DocumentDB의 $dayOfWeek 연산자는 지정된 날짜 필드에서 요일을 추출합니다. MongoDB와 동일한 동작인 1(일요일)에서 7(토요일) 사이의 숫자로 요일을 반환합니다.

파라미터

- date field: 요일을 추출할 날짜 필드입니다.

예제(MongoDB 쉘)

이 예제에서는 $dayOfWeek 연산자를 사용하여 weather 컬렉션의 date 필드에서 요일을 추출하는 방법을 보여줍니다.

샘플 문서 생성

```
db.weather.insertMany([
  {
    "temperature": 97.5,
    "humidity": 0.60,
    "date": new Date("2023-04-01")
  },
  {
    "temperature": 95.2,
    "humidity": 0.55,
    "date": new Date("2023-04-02")
  },
  {
    "temperature": 92.8,
    "humidity": 0.65,
    "date": new Date("2023-04-03")
  }
]);
```

쿼리 예제

```
db.weather.aggregate([
  {
    $project: {
      dayOfWeek: { $dayOfWeek: "$date" }
    }
```

```
  }
]).pretty();
```

출력

```
{ "_id" : ObjectId("64272c6663f4f8ce422c2d91"), "dayOfWeek" : 7 }
{ "_id" : ObjectId("64272c6663f4f8ce422c2d92"), "dayOfWeek" : 1 }
{ "_id" : ObjectId("64272c6663f4f8ce422c2d93"), "dayOfWeek" : 2 }
```

코드 예제

$dayOfWeek 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('weather');

  const result = await collection.aggregate([
    {
      $project: {
        dayOfWeek: { $dayOfWeek: '$date' }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient
```

```
def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.weather

    result = list(collection.aggregate([
        {
            '$project': {
                'dayOfWeek': { '$dayOfWeek': '$date' }
            }
        }
    ]))

    print(result)

    client.close()

if __name__ == '__main__':
    main()
```

$dayOfYear

Amazon DocumentDB의 `$dayOfYear` 연산자는 날짜의 날짜를 1에서 366 사이의 숫자로 반환합니다(윤년이 아닌 경우 365).

파라미터

- `expression`: 연도의 날짜를 추출할 날짜 필드 또는 표현식입니다.

예제(MongoDB 쉘)

이 예제에서는 `$dayOfYear` 연산자를 사용하여 Amazon DocumentDB 컬렉션의 날짜 필드에서 연도의 날짜를 추출하는 방법을 보여줍니다.

샘플 문서 생성

```
db.weather.insert([
  {
```

```
    "temperature" : 97.5,
    "humidity": 0.60,
    "date" : new Date("2023-03-15")
  },
  {
    "temperature" : 82.3,
    "humidity": 0.75,
    "date" : new Date("2023-12-31")
  }
])
```

쿼리 예제

```
db.weather.aggregate([
  {
    $project: {
      dayOfYear: { $dayOfYear: "$date" }
    }
  }
]).pretty()
```

출력

```
{ "_id" : ObjectId("642b86fc7d8e07af279bbe63"), "dayOfYear" : 74 }
{ "_id" : ObjectId("642b86fc7d8e07af279bbe64"), "dayOfYear" : 365 }
```

코드 예제

$dayOfYear 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('weather');

  const result = await collection.aggregate([
```

```
    {
      $project: {
        dayOfYear: { $dayOfYear: "$date" }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['weather']

    result = list(collection.aggregate([
        {
            '$project': {
                'dayOfYear': { '$dayOfYear': '$date' }
            }
        }
    ]))

    print(result)

    client.close()

example()
```

$degreesToRadians

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $degreesToRadians 연산자는 값을 각도에서 라디안으로 변환합니다.

파라미터

- expression: 각도의 숫자 값으로 확인되는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $degreesToRadians 연산자를 사용하여 도 값을 라디안으로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.angles.insertMany([
  { "_id": 1, "degrees": 0 },
  { "_id": 2, "degrees": 90 },
  { "_id": 3, "degrees": 180 },
  { "_id": 4, "degrees": 360 }
]);
```

쿼리 예제

```
db.angles.aggregate([
  { $project: {
    "radians": { $degreesToRadians: "$degrees" }
  }}
]);
```

출력

```
[
  { "_id": 1, "radians": 0 },
  { "_id": 2, "radians": 1.5707963267948966 },
  { "_id": 3, "radians": 3.141592653589793 },
  { "_id": 4, "radians": 6.283185307179586 }
]
```

코드 예제

$degreesToRadians 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('angles');

  const result = await collection.aggregate([
    { $project: {
      "radians": { $degreesToRadians: "$degrees" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['angles']

    result = list(collection.aggregate([
        { '$project': {
            'radians': { '$degreesToRadians': '$degrees' }
        }}
    ]))

    print(result)
    client.close()
```

```
if __name__ == "__main__":
    main()
```

$denseRank

버전 8.0.2에서 새로 추가되었습니다.

Amazon DocumentDB의 $denseRank 창 연산자는 $setWindowFields 단계 sortBy 순서에 따라 파티션 내의 각 문서의 위치를 반환합니다. sortBy 값이 동일한 문서는 동일한 순위를 받으며, 다음 고유 값은 바로 다음 위치를 사용하여 공백을 남기지 않습니다(예: 1, 2, 2, 3). $setWindowFields 스테이지의 output 필드 아래에 지정됩니다.

$setWindowFields 스테이지에서만 사용되는 창 연산자입니다.

파라미터

- $denseRank는 빈 문서를 {} 값으로 사용합니다.
- window 프레임은 허용되지 않습니다.
- $setWindowFields 스테이지에 정확히 하나의 sortBy 키가 필요합니다.

예제(MongoDB 쉘)

다음 예제에서는를 기준으로 문서를 분할하고game,를 기준으로 각 파티션score을 내림차순으로 정렬하며, 각 플레이어에게 밀집 순위를 할당합니다.

샘플 문서 생성

```
db.players.insertMany([
  { _id: 1, game: "A", player: "Alice", score: 90 },
  { _id: 2, game: "A", player: "Bob", score: 85 },
  { _id: 3, game: "A", player: "Carol", score: 85 },
  { _id: 4, game: "A", player: "Dave", score: 70 },
  { _id: 5, game: "B", player: "Erin", score: 95 },
  { _id: 6, game: "B", player: "Frank", score: 80 }
]);
```

쿼리 예제

```
db.players.aggregate([
```

```
  {
    $setWindowFields: {
      partitionBy: "$game",
      sortBy: { score: -1 },
      output: {
        denseRank: { $denseRank: {} }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "game": "A", "player": "Alice", "score": 90, "denseRank": 1 },
  { "_id": 2, "game": "A", "player": "Bob", "score": 85, "denseRank": 2 },
  { "_id": 3, "game": "A", "player": "Carol", "score": 85, "denseRank": 2 },
  { "_id": 4, "game": "A", "player": "Dave", "score": 70, "denseRank": 3 },
  { "_id": 5, "game": "B", "player": "Erin", "score": 95, "denseRank": 1 },
  { "_id": 6, "game": "B", "player": "Frank", "score": 80, "denseRank": 2 }
]
```

각 game 파티션 내에서 플레이어는 내림차순으로 순위를 매깁니다score. Bob과 Carol은를 묶85고 순위를 공유하며2, Dave는 격차 3 없이 순위를 따릅니다.

코드 예제

$setWindowFields 단계에서 $denseRank 연산자를 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
```

```
    const players = db.collection('players');

    await players.insertMany([
      { _id: 1, game: "A", player: "Alice", score: 90 },
      { _id: 2, game: "A", player: "Bob", score: 85 },
      { _id: 3, game: "A", player: "Carol", score: 85 },
      { _id: 4, game: "A", player: "Dave", score: 70 },
      { _id: 5, game: "B", player: "Erin", score: 95 },
      { _id: 6, game: "B", player: "Frank", score: 80 }
    ]);

    const result = await players.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$game",
          sortBy: { score: -1 },
          output: {
            denseRank: { $denseRank: {} }
          }
        }
      }
    ]).toArray();

    console.log(result);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']
        players = db['players']
```

```
        players.insert_many([
            { '_id': 1, 'game': 'A', 'player': 'Alice', 'score': 90 },
            { '_id': 2, 'game': 'A', 'player': 'Bob', 'score': 85 },
            { '_id': 3, 'game': 'A', 'player': 'Carol', 'score': 85 },
            { '_id': 4, 'game': 'A', 'player': 'Dave', 'score': 70 },
            { '_id': 5, 'game': 'B', 'player': 'Erin', 'score': 95 },
            { '_id': 6, 'game': 'B', 'player': 'Frank', 'score': 80 }
        ])

        result = list(players.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$game',
                    'sortBy': { 'score': -1 },
                    'output': {
                        'denseRank': { '$denseRank': {} }
                    }
                }
            }
        ]))

        print(result)
    finally:
        client.close()

example()
```

$divide

Amazon DocumentDB 집계 파이프라인의 $divide 연산자는 한 숫자를 다른 숫자로 나누는 데 사용됩니다. 문서 내의 숫자 필드에 수학 연산을 수행하는 데 유용한 연산자입니다.

파라미터

- numerator: 분할할 배당 또는 숫자입니다.
- denominator: 나눕니다.

예제(MongoDB 쉘)

이 예제에서는 $divide 연산자를 사용하여 연간 급여와 연간 근무 시간을 기준으로 직원의 시간당 요금을 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.employees.insertMany([
  { _id: 1, name: "John Doe", salary: 80000, hoursPerYear: 2080 },
  { _id: 2, name: "Jane Smith", salary: 90000, hoursPerYear: 2080 },
  { _id: 3, name: "Bob Johnson", salary: 75000, hoursPerYear: 2080 }
]);
```

쿼리 예제

```
db.employees.aggregate([
  {
    $project: {
      name: 1,
      hourlyRate: { $divide: ["$salary", "$hoursPerYear"] }
    }
  }
])
```

출력

```
[
  { "_id" : 1, "name" : "John Doe", "hourlyRate" : 38.46153846153846 },
  { "_id" : 2, "name" : "Jane Smith", "hourlyRate" : 43.26923076923077 },
  { "_id" : 3, "name" : "Bob Johnson", "hourlyRate" : 36.05769230769231 }
]
```

코드 예제

$divide 명령 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function calculateHourlyRate() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const employees = db.collection('employees');

  const result = await employees.aggregate([
    {
      $project: {
        name: 1,
        hourlyRate: { $divide: ["$salary", "$hoursPerYear"] }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

calculateHourlyRate();
```

Python

```
from pymongo import MongoClient

def calculate_hourly_rate():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.mydatabase
    employees = db.employees

    result = list(employees.aggregate([
        {
            '$project': {
                'name': 1,
                'hourlyRate': { '$divide': ['$salary', '$hoursPerYear'] }
            }
        }
    ]))

    print(result)
    client.close()
```

```
calculate_hourly_rate()
```

$documentNumber

버전 8.0.2에서 새로 추가되었습니다.

Amazon DocumentDB의 $documentNumber 창 연산자는 $setWindowFields 단계 sortBy 순서에 따라 파티션의 각 문서에 대해 고유한 순차 번호를 반환합니다. $rank 및와 달리$denseRank, 모든 문서는 고유한 번호(예: 1, 2, 3, 4)를 수신하고 sortBy 값이 동일한 문서는 발생한 순서대로 번호가 매겨지므로 특별히 매듭을 취급하지 않습니다. $setWindowFields 스테이지의 output 필드 아래에 지정됩니다.

$setWindowFields 스테이지에서만 사용되는 창 연산자입니다.

파라미터

- $documentNumber는 빈 문서를 {} 값으로 사용합니다.
- window 프레임은 허용되지 않습니다.
- $setWindowFields 스테이지에 정확히 하나의 sortBy 키가 필요합니다.

예제(MongoDB 쉘)

다음 예제에서는를 기준으로 문서를 분할하고game,를 기준으로 각 파티션score을 내림차순으로 정렬하며, 각 플레이어에게 순차적 문서 번호를 할당합니다.

샘플 문서 생성

```
db.players.insertMany([
  { _id: 1, game: "A", player: "Alice", score: 90 },
  { _id: 2, game: "A", player: "Bob", score: 85 },
  { _id: 3, game: "A", player: "Carol", score: 85 },
  { _id: 4, game: "A", player: "Dave", score: 70 },
  { _id: 5, game: "B", player: "Erin", score: 95 },
  { _id: 6, game: "B", player: "Frank", score: 80 }
]);
```

쿼리 예제

```
db.players.aggregate([
```

```
  {
    $setWindowFields: {
      partitionBy: "$game",
      sortBy: { score: -1 },
      output: {
        documentNumber: { $documentNumber: {} }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "game": "A", "player": "Alice", "score": 90, "documentNumber": 1 },
  { "_id": 2, "game": "A", "player": "Bob", "score": 85, "documentNumber": 2 },
  { "_id": 3, "game": "A", "player": "Carol", "score": 85, "documentNumber": 3 },
  { "_id": 4, "game": "A", "player": "Dave", "score": 70, "documentNumber": 4 },
  { "_id": 5, "game": "B", "player": "Erin", "score": 95, "documentNumber": 1 },
  { "_id": 6, "game": "B", "player": "Frank", "score": 80, "documentNumber": 2 }
]
```

각 game 파티션 내에서 각 플레이어는 고유한 순차적 번호를 내림차score순으로 받습니다. Bob과 Carol은에 묶여 85 있지만 여전히 번호가 매겨2져 있고 발생한 3 순서대로 묶여 있습니다.

코드 예제

$setWindowFields 단계에서 $documentNumber 연산자를 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
```

```
    const players = db.collection('players');

    await players.insertMany([
      { _id: 1, game: "A", player: "Alice", score: 90 },
      { _id: 2, game: "A", player: "Bob", score: 85 },
      { _id: 3, game: "A", player: "Carol", score: 85 },
      { _id: 4, game: "A", player: "Dave", score: 70 },
      { _id: 5, game: "B", player: "Erin", score: 95 },
      { _id: 6, game: "B", player: "Frank", score: 80 }
    ]);

    const result = await players.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$game",
          sortBy: { score: -1 },
          output: {
            documentNumber: { $documentNumber: {} }
          }
        }
      }
    ]).toArray();

    console.log(result);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']
        players = db['players']
```

```
        players.insert_many([
            { '_id': 1, 'game': 'A', 'player': 'Alice', 'score': 90 },
            { '_id': 2, 'game': 'A', 'player': 'Bob', 'score': 85 },
            { '_id': 3, 'game': 'A', 'player': 'Carol', 'score': 85 },
            { '_id': 4, 'game': 'A', 'player': 'Dave', 'score': 70 },
            { '_id': 5, 'game': 'B', 'player': 'Erin', 'score': 95 },
            { '_id': 6, 'game': 'B', 'player': 'Frank', 'score': 80 }
        ])

        result = list(players.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$game',
                    'sortBy': { 'score': -1 },
                    'output': {
                        'documentNumber': { '$documentNumber': {} }
                    }
                }
            }
        ]))

        print(result)
    finally:
        client.close()

example()
```

$eq

$eq 집계 연산자는 두 값을 비교하고 값이 같`true`으면를 반환하고 그렇지 않으면를 반환합니다`false`.

파라미터

- `expression1`: 비교할 첫 번째 값입니다.
- `expression2`: 비교할 두 번째 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 $eq 연산자를 사용하여 제품 수량이 대상 값과 일치하는지 확인하는 방법을 보여줍니다.

샘플 문서 생성

```
db.inventory.insertMany([
  { _id: 1, item: "Widget", qty: 50, target: 50 },
  { _id: 2, item: "Gadget", qty: 30, target: 50 },
  { _id: 3, item: "Tool", qty: 50, target: 40 }
]);
```

쿼리 예제

```
db.inventory.aggregate([
  {
    $project: {
      item: 1,
      qty: 1,
      target: 1,
      meetsTarget: { $eq: ["$qty", "$target"] }
    }
  }
]);
```

출력

```
[
  { _id: 1, item: 'Widget', qty: 50, target: 50, meetsTarget: true },
  { _id: 2, item: 'Gadget', qty: 30, target: 50, meetsTarget: false },
  { _id: 3, item: 'Tool', qty: 50, target: 40, meetsTarget: false }
]
```

코드 예제

$eq 집계 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('inventory');

  const result = await collection.aggregate([
    {
      $project: {
        item: 1,
        qty: 1,
        target: 1,
        meetsTarget: { $eq: ["$qty", "$target"] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['inventory']

    result = list(collection.aggregate([
        {
            '$project': {
                'item': 1,
                'qty': 1,
                'target': 1,
                'meetsTarget': { '$eq': ['$qty', '$target'] }
```

```
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$exp

버전 4.0의 새로운 기능

Amazon DocumentDB의 $exp 연산자를 사용하면 상수 e를 지정된 숫자로 올릴 수 있습니다.

파라미터

- expression: 평가할 표현식입니다. 필드 참조, 산술 연산 및 기타 집계 단계를 포함한 모든 유효한 집계 표현식일 수 있습니다.

예제(MongoDB 쉘)

다음 예제에서는 연$exp산자를 사용하여 quantity 필드가 price 필드보다 큰 모든 문서를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.items.insertMany([
  { item: "canvas", quantity: 4 },
  { item: "journal", quantity: 2 }
]);
```

쿼리 예제

```
db.items.aggregate([
  { $project: {
    "quantityRaised": {$exp: "$quantity"}}
  }
]);
```

출력

```
[
  {
    _id: ObjectId('6920b785311cf98b79d2950d'),
    quantityRaised: 54.598150033144236
  },
  {
    _id: ObjectId('6920b785311cf98b79d2950e'),
    quantityRaised: 7.38905609893065
  }
]
```

코드 예제

$exp 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function aggregateExp() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const items = db.collection('items');

  const result = await items.aggregate([
    { $project: {
      "quantityRaised": {$exp: "$quantity"}}
    }
    ]).toArray();

  console.log(result);
  client.close();
}

aggregateExp();
```

Python

```
from pymongo import MongoClient
```

```
def aggregate_exp():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    items = db.items

    result = list(items.aggregate([
      { "$project": {
        "quantityRaised": {"$exp": "$quantity"}}
      }
    ]))

    print(result)
    client.close()

aggregate_exp()
```

$facet

버전 8.0.2에서 새로 추가되었습니다.

Amazon DocumentDB의 `$facet` 집계 단계는 동일한 입력 문서 세트의 단일 단계 내에서 여러 집계 파이프라인을 처리합니다. 각 출력 필드는 자체 파이프라인을 정의하며 모든 파이프라인은 동일한 입력 문서를 수신합니다. 이를 통해 여러 개의 개별 쿼리를 통해 컬렉션을 실행하는 대신 단일 쿼리에서 개수, 그룹화 및 요약 통계와 같은 여러 개의 독립적인 집계를 계산할 수 있습니다.

각 하위 파이프라인은 독립적으로 실행되며 해당 출력 필드에 저장된 문서 배열을 생성합니다. 하위 파이프라인은 상태를 공유하지 않으므로 한 파이프라인의 출력은 다른 파이프라인에 영향을 주지 않습니다.

구문

```
{
  $facet: {
    <outputField1>: [ <stage1>, <stage2>, ... ],
    <outputField2>: [ <stage1>, <stage2>, ... ],
    ...
  }
}
```

파라미터

- `outputField` (필수): 연결된 집계 파이프라인의 결과 배열을 포함하는 출력 필드의 이름입니다. 하나 이상의 출력 필드를 지정할 수 있습니다. 각 출력 필드는 입력 문서에 적용되는 집계 파이프라인 단계의 배열에 매핑됩니다.

출력

`$facet` 스테이지는 단일 문서를 반환합니다. 지정하는 각 출력 필드에는 해당 하위 파이프라인에서 생성된 문서 배열이 포함되어 있습니다.

제한 사항

- `$facet` 하위 파이프라인에는 `$changeStream`, `$collStats`, `$facet` (중첩은 허용되지 않음), , `$geoNear`, `$indexStats`, `$listSearchIndexes`, `$merge`, `$out`, `$planCacheStats`, `$search$searchMeta`, 단계가 포함될 수 없습니다`$vectorSearch`.
- `$facet` 스테이지가 반환하는 문서는 16MB BSON 문서 크기 제한 내에 있어야 합니다.
- `$sort` 단계 앞에 오는 `$facet` 단계는 해당 순서를 `$facet` 경계 전체에 적용해야 하는 경우 `$first`, 또는 `$bucketAuto` 단계에서 주문에 민감한 누산기(`$last$push$firstN$lastN`, `$group`, , `$bucket`, 또는 `$mergeObjects`)의 순서를 결정할 수 없습니다. 이는 그룹화 단계가 `$facet` 하위 파이프라인 내에 있는지 아니면 `$facet` 단계 뒤에 있는지에 관계없이 적용됩니다. 이 경우 Amazon DocumentDB는 파이프라인을 실행하는 대신 오류를 반환합니다. 이러한 누적기를 주문하려면 그룹화 `$sort` 단계 바로 앞에 `$facet` 있는 하위 파이프라인 내에 단계를 배치합니다. , `$sum`, 및 `$min$max`와 같은 주문 비구분 누적기는 영향을 받지 `$count`않습니다.

예제(MongoDB 쉘)

다음 예제에서는 `$facet` 스테이지를 사용하여 단일 쿼리에서 동일한 판매 데이터에 대해 여러 집계를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.sales.insertMany([
  { item: "abc", price: 10, quantity: 2, date: new Date("2020-09-01") },
  { item: "def", price: 20, quantity: 1, date: new Date("2020-10-01") },
  { item: "ghi", price: 5, quantity: 3, date: new Date("2020-11-01") },
  { item: "jkl", price: 15, quantity: 2, date: new Date("2020-12-01") },
  { item: "abc", price: 25, quantity: 1, date: new Date("2021-01-01") }
]);
```

쿼리 예제

다음 명령은 $facet를 사용하여 단일 단계에서 항목당 판매된 총 수량, 전체 가격 통계 및 총 문서 수를 계산합니다.

```
db.sales.aggregate([
  {
    $facet: {
      "totalByItem": [
        { $group: { _id: "$item", totalQuantity: { $sum: "$quantity" } } }
      ],
      "priceStats": [
        { $group: { _id: null, avgPrice: { $avg: "$price" }, maxPrice: { $max:
 "$price" } } }
      ],
      "documentCount": [
        { $count: "total" }
      ]
    }
  }
])
```

출력

명령은 다음 출력을 반환합니다.

```
[
  {
    totalByItem: [
      { _id: "abc", totalQuantity: 3 },
      { _id: "def", totalQuantity: 1 },
      { _id: "ghi", totalQuantity: 3 },
      { _id: "jkl", totalQuantity: 2 }
    ],
    priceStats: [
      { _id: null, avgPrice: 15, maxPrice: 25 }
    ],
    documentCount: [
      { total: 5 }
    ]
  }
]
```

코드 예제

`$facet` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const sales = db.collection('sales');

  const result = await sales.aggregate([
    {
      $facet: {
        "totalByItem": [
          { $group: { _id: "$item", totalQuantity: { $sum: "$quantity" } } }
        ],
        "priceStats": [
          { $group: { _id: null, avgPrice: { $avg: "$price" }, maxPrice: { $max:
 "$price" } } }
        ],
        "documentCount": [
          { $count: "total" }
        ]
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    sales = db['sales']

    result = list(sales.aggregate([
        {
            '$facet': {
                'totalByItem': [
                    {'$group': {'_id': '$item', 'totalQuantity': {'$sum':
 '$quantity'}}}
                ],
                'priceStats': [
                    {'$group': {'_id': None, 'avgPrice': {'$avg': '$price'},
 'maxPrice': {'$max': '$price'}}}
                ],
                'documentCount': [
                    {'$count': 'total'}
                ]
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$filter

Amazon DocumentDB의 `$filter` 연산자는 배열의 각 요소에 필터 표현식을 적용하고 지정된 조건과 일치하는 요소만 포함하는 배열을 반환하는 데 사용됩니다. 이 연산자는 문서 내의 배열 필드에 대해 복잡한 필터링 작업을 수행해야 할 때 유용합니다.

파라미터

- `input`: 필터링할 배열 필드입니다.
- `as`: `cond` 표현식 내에서 `input` 배열의 각 요소에 사용할 변수 이름입니다.
- `cond`: 지정된 요소를 출력 배열에 포함할지 여부를 결정하는 부울 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$filter` 연산자를 사용하여 각 주문의 고객을 프로젝션하고 가격이 15보다 큰 항목 배열의 항목만 포함하는 새 배열 필드 paidItems를 생성하는 방법을 보여줍니다. 기본적으로 각 주문의 항목을 필터링하여 15개가 넘는 제품만 포함합니다.

샘플 문서 생성

```
db.orders.insertMany([
  { _id: 1, customer: "abc123", items: [
    { name: "Product A", price: 10, qty: 2 },
    { name: "Product B", price: 20, qty: 1 }
  ]},
  { _id: 2, customer: "def456", items: [
    { name: "Product C", price: 5, qty: 3 },
    { name: "Product D", price: 15, qty: 4 }
  ]},
  { _id: 3, customer: "ghi789", items: [
    { name: "Product E", price: 8, qty: 3 },
    { name: "Product F", price: 12, qty: 1 }
  ]}
]);
```

쿼리 예제

```
db.orders.aggregate([
  {
    $project: {
      customer: 1,
      paidItems: {
        $filter: {
          input: "$items",
          as: "item",
          cond: { $gt: ["$$item.price", 15] }
        }
      }
    }
  }
]).pretty();
```

출력

```
[
```

```
  {
    _id: 1,
    customer: 'abc123',
    paidItems: [ { name: 'Product B', price: 20, qty: 1 } ]
  },
  { _id: 2, customer: 'def456', paidItems: [] },
  { _id: 3, customer: 'ghi789', paidItems: [] }
]
```

코드 예제

`$filter` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const db = client.db('test');
    const collection = db.collection('orders');

    const result = await collection.aggregate([
      {
        $project: {
          customer: 1,
          paidItems: {
            $filter: {
              input: "$items",
              as: "item",
              cond: { $gt: ["$$item.price", 15] }
            }
          }
        }
      }
    ]).toArray();
```

```
    console.log(JSON.stringify(result, null, 2));

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false'

    with MongoClient(uri) as client:
        db = client.test
        collection = db.orders

        result = list(collection.aggregate([
            {
                '$project': {
                    'customer': 1,
                    'paidItems': {
                        '$filter': {
                            'input': '$items',
                            'as': 'item',
                            'cond': { '$gt': ['$$item.price', 15] }
                        }
                    }
                }
            }
        ]))

        for doc in result:
            pprint(doc)
```

```
example()
```

$first

버전 5.0의 새 버전입니다.

Elastic 클러스터에서는 지원되지 않습니다.

Amazon DocumentDB의 $first 연산자는 그룹화된 문서 세트에서 첫 번째 문서를 반환합니다. 일반적으로 집계 파이프라인에서 특정 조건과 일치하는 첫 번째 문서를 검색하는 데 사용됩니다.

파라미터

- expression: 각 그룹의 첫 번째 값으로 반환할 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 연$first산자를 사용하여 집계 중에 각 범주에 대해 발생한 첫 번째 항목 값을 검색하는 방법을 보여줍니다.

참고:는 파이프라인의 현재 문서 순서를 기반으로 첫 번째 문서를 $first 반환합니다. 특정 주문(예: 날짜, 가격 등)을 보장하려면 $sort 스테이지 전에 $group 스테이지를 사용해야 합니다.

샘플 문서 생성

```
db.products.insertMany([
  { _id: 1, item: "abc", price: 10, category: "food" },
  { _id: 2, item: "jkl", price: 20, category: "food" },
  { _id: 3, item: "xyz", price: 5, category: "toy" },
  { _id: 4, item: "abc", price: 5, category: "toy" }
]);
```

쿼리 예제

```
db.products.aggregate([
  { $group: { _id: "$category", firstItem: { $first: "$item" } } }
]);
```

출력

```
[
```

```
  { "_id" : "food", "firstItem" : "abc" },
  { "_id" : "toy", "firstItem" : "xyz" }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$first산자는 $setWindowFields 스테이지에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 expression 첫 번째 문서에서의 값을 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

> **Note**
>
> 에서 창 연산자로 사용할 경우 $setWindowFields$first는 100MB의 중간 데이터로 제한됩니다. 이 제한을 초과하는 작업은 오류를 반환합니다.

샘플 문서 생성

```
db.stockPrices.insertMany([
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

쿼리 예제

다음 예제에서는 문서를 기준으로 분할하고ticker, 각 파티션을 기준으로 정렬하고hour, 각 파티션에 기록된 첫 번째(가장 빠른) 가격을 반환합니다.

```
db.stockPrices.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        firstPrice: {
```

```
          $first: "$price",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "firstPrice": 50 },
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "firstPrice": 50 },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "firstPrice": 50 },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "firstPrice": 50 },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "firstPrice": 30 },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "firstPrice": 30 },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "firstPrice": 30 }
]
```

각 문서는 창의 첫 번째 문서(파티션에서 가장 빠른 문서)의 가격`firstPrice`인 hour로 보강됩니다.

코드 예제

`$first` 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누산기 사용량()과 창 연산자 사용량(`$group`)을 모두 보여줍니다. `$setWindowFields`

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const db = client.db('test');

    // Accumulator usage: first value per group
```

```
    const products = db.collection('products');
    await products.insertMany([
      { _id: 1, item: "abc", price: 10, category: "food" },
      { _id: 2, item: "jkl", price: 20, category: "food" },
      { _id: 3, item: "xyz", price: 5, category: "toy" },
      { _id: 4, item: "abc", price: 5, category: "toy" }
    ]);
    const accumulatorResult = await products.aggregate([
      { $group: { _id: "$category", firstItem: { $first: "$item" } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Window operator usage: first value within each partition
    const stockPrices = db.collection('stockPrices');
    await stockPrices.insertMany([
      { _id: 1, ticker: "ABC", hour: 1, price: 50 },
      { _id: 2, ticker: "ABC", hour: 2, price: 45 },
      { _id: 3, ticker: "ABC", hour: 3, price: 60 },
      { _id: 4, ticker: "ABC", hour: 4, price: 40 },
      { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
      { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
      { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
    ]);
    const windowResult = await stockPrices.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$ticker",
          sortBy: { hour: 1 },
          output: {
            firstPrice: {
              $first: "$price",
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
```

```
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client['test']

        # Accumulator usage: first value per group
        products = db['products']
        products.insert_many([
            { '_id': 1, 'item': 'abc', 'price': 10, 'category': 'food' },
            { '_id': 2, 'item': 'jkl', 'price': 20, 'category': 'food' },
            { '_id': 3, 'item': 'xyz', 'price': 5, 'category': 'toy' },
            { '_id': 4, 'item': 'abc', 'price': 5, 'category': 'toy' }
        ])
        accumulator_result = list(products.aggregate([
            { '$group': { '_id': '$category', 'firstItem': { '$first': '$item' } } }
        ]))
        pprint(accumulator_result)

        # Window operator usage: first value within each partition
        stock_prices = db['stockPrices']
        stock_prices.insert_many([
            { '_id': 1, 'ticker': 'ABC', 'hour': 1, 'price': 50 },
            { '_id': 2, 'ticker': 'ABC', 'hour': 2, 'price': 45 },
            { '_id': 3, 'ticker': 'ABC', 'hour': 3, 'price': 60 },
            { '_id': 4, 'ticker': 'ABC', 'hour': 4, 'price': 40 },
            { '_id': 5, 'ticker': 'XYZ', 'hour': 1, 'price': 30 },
            { '_id': 6, 'ticker': 'XYZ', 'hour': 2, 'price': 35 },
            { '_id': 7, 'ticker': 'XYZ', 'hour': 3, 'price': 25 }
        ])
        window_result = list(stock_prices.aggregate([
```

```
            {
                '$setWindowFields': {
                    'partitionBy': '$ticker',
                    'sortBy': { 'hour': 1 },
                    'output': {
                        'firstPrice': {
                            '$first': '$price',
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        pprint(window_result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$firstN

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 `$firstN` 연산자는 첫 번째 N 요소를 반환합니다. `$group` 스테이지에서 누적기로 사용할 경우 각 그룹의 첫 번째 N 값의 배열을 반환합니다. 배열 표현식 연산자로 사용할 경우 배열의 첫 번째 N 요소를 반환합니다.

파라미터

- `input`: 값을 반환할 필드 또는 배열로 확인되는 표현식입니다.
- `n`: 반환할 값의 수를 지정하는 양의 정수입니다. `$group` 단계에서 누적기로 사용할 경우 그룹 `_id` 필드를 기반으로 양의 정수로 확인되는 한 `n`도 표현식이 될 수 있습니다.

예제(MongoDB 쉘)

다음 예제에서는 집계 중에 $firstN 누적기를 사용하여 각 항목의 처음 두 수량을 검색하는 방법을 보여줍니다.

> **Note**
>
> $firstN는 문서가 $group 단계에 도달하는 순서대로 값을 선택합니다. 특정 순서(예: 날짜 또는 점수)에 대한 첫 번째 N 값을 반환하려면 앞에 $sort 스테이지를 추가합니다$group.

샘플 문서 생성

```
db.sales.insertMany([
  { item: "abc", quantity: 10, date: ISODate("2023-01-01") },
  { item: "abc", quantity: 5, date: ISODate("2023-01-02") },
  { item: "abc", quantity: 8, date: ISODate("2023-01-03") },
  { item: "xyz", quantity: 15, date: ISODate("2023-01-01") },
  { item: "xyz", quantity: 7, date: ISODate("2023-01-02") },
  { item: "xyz", quantity: 3, date: ISODate("2023-01-03") }
]);
```

쿼리 예제

```
db.sales.aggregate([
  { $group: { _id: "$item", firstTwoQuantities: { $firstN: { input: "$quantity", n:
 2 } } } }
]);
```

출력

```
[
  { "_id": "abc", "firstTwoQuantities": [10, 5] },
  { "_id": "xyz", "firstTwoQuantities": [15, 7] }
]
```

표현식 사용 예제(MongoDB Shell)

연$firstN산자를 $project 스테이지 내의 표현식으로 사용하여 배열 필드의 첫 번째 N 요소를 반환할 수도 있습니다.

샘플 문서 생성

```
db.inventory.insertMany([
  { _id: 1, item: "abc", tags: ["red", "green", "blue", "yellow", "purple"] },
  { _id: 2, item: "xyz", tags: ["alpha", "beta", "gamma"] }
]);
```

쿼리 예제

```
db.inventory.aggregate([
  { $project: {
      firstThreeTags: { $firstN: { input: "$tags", n: 3 } }
    }}
]);
```

출력

```
[
  { "_id": 1, "firstThreeTags": ["red", "green", "blue"] },
  { "_id": 2, "firstThreeTags": ["alpha", "beta", "gamma"] }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연`$firstN`산자는 `$setWindowFields` 스테이지에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 문서에 대해 최대 첫 번째 n 값의 배열을 반환합니다. `output` 필드 아래에 연산자를 지정하고 선택적으로 `window` 문서로 창 경계를 정의합니다.

> **Note**
>
> 에서 창 연산자로 사용할 경우 `$setWindowFields`$firstN는 100MB의 중간 데이터로 제한됩니다. 이 제한을 초과하는 작업은 오류를 반환합니다.

샘플 문서 생성

```
db.stockPrices.insertMany([
```

```
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

쿼리 예제

다음 예제에서는를 기준으로 문서를 분할하고ticker,를 기준으로 각 파티션을 정렬하고hour, 파티션 시작부터 현재 문서를 통해 표시되는 처음 두 가격을 반환합니다.

```
db.stockPrices.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        firstTwoSoFar: {
          $firstN: { input: "$price", n: 2 },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "firstTwoSoFar": [50] },
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "firstTwoSoFar": [50, 45] },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "firstTwoSoFar": [50, 45] },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "firstTwoSoFar": [50, 45] },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "firstTwoSoFar": [30] },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "firstTwoSoFar": [30, 35] },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "firstTwoSoFar": [30, 35] }
]
```

각 문서는 창 내의 `firstTwoSoFar`처음 두 `price` 값(hour순서대로)인 로 보강됩니다.

코드 예제

$firstN 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누적기 사용량(의 $group), 표현식 사용량(의 $project) 및 창 연산자 사용량(의)을 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');

    // Accumulator usage: first N values per group
    const sales = db.collection('sales');
    await sales.insertMany([
      { item: "abc", quantity: 10, date: new Date("2023-01-01") },
      { item: "abc", quantity: 5, date: new Date("2023-01-02") },
      { item: "abc", quantity: 8, date: new Date("2023-01-03") },
      { item: "xyz", quantity: 15, date: new Date("2023-01-01") },
      { item: "xyz", quantity: 7, date: new Date("2023-01-02") },
      { item: "xyz", quantity: 3, date: new Date("2023-01-03") }
    ]);
    const accumulatorResult = await sales.aggregate([
      { $group: { _id: "$item", firstTwoQuantities: { $firstN: { input: "$quantity",
 n: 2 } } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Expression usage: first N elements of an array field
    const inventory = db.collection('inventory');
    await inventory.insertMany([
      { _id: 1, item: "abc", tags: ["red", "green", "blue", "yellow", "purple"] },
      { _id: 2, item: "xyz", tags: ["alpha", "beta", "gamma"] }
    ]);
    const expressionResult = await inventory.aggregate([
      { $project: { firstThreeTags: { $firstN: { input: "$tags", n: 3 } } } }
```

```
    ]).toArray();
    console.log('Expression result:', expressionResult);

    // Window operator usage: first N values over a window in each partition
    const stockPrices = db.collection('stockPrices');
    await stockPrices.insertMany([
      { _id: 1, ticker: "ABC", hour: 1, price: 50 },
      { _id: 2, ticker: "ABC", hour: 2, price: 45 },
      { _id: 3, ticker: "ABC", hour: 3, price: 60 },
      { _id: 4, ticker: "ABC", hour: 4, price: 40 },
      { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
      { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
      { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
    ]);
    const windowResult = await stockPrices.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$ticker",
          sortBy: { hour: 1 },
          output: {
            firstTwoSoFar: {
              $firstN: { input: "$price", n: 2 },
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from datetime import datetime

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']

        # Accumulator usage: first N values per group
        sales = db['sales']
        sales.insert_many([
            { 'item': 'abc', 'quantity': 10, 'date': datetime(2023, 1, 1) },
            { 'item': 'abc', 'quantity': 5, 'date': datetime(2023, 1, 2) },
            { 'item': 'abc', 'quantity': 8, 'date': datetime(2023, 1, 3) },
            { 'item': 'xyz', 'quantity': 15, 'date': datetime(2023, 1, 1) },
            { 'item': 'xyz', 'quantity': 7, 'date': datetime(2023, 1, 2) },
            { 'item': 'xyz', 'quantity': 3, 'date': datetime(2023, 1, 3) }
        ])
        accumulator_result = list(sales.aggregate([
            { '$group': { '_id': '$item', 'firstTwoQuantities': { '$firstN':
 { 'input': '$quantity', 'n': 2 } } } }
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: first N elements of an array field
        inventory = db['inventory']
        inventory.insert_many([
            { '_id': 1, 'item': 'abc', 'tags': ['red', 'green', 'blue', 'yellow',
 'purple'] },
            { '_id': 2, 'item': 'xyz', 'tags': ['alpha', 'beta', 'gamma'] }
        ])
        expression_result = list(inventory.aggregate([
            { '$project': { 'firstThreeTags': { '$firstN': { 'input': '$tags', 'n':
 3 } } } }
        ]))
        print('Expression result:', expression_result)

        # Window operator usage: first N values over a window in each partition
        stock_prices = db['stockPrices']
        stock_prices.insert_many([
            { '_id': 1, 'ticker': 'ABC', 'hour': 1, 'price': 50 },
            { '_id': 2, 'ticker': 'ABC', 'hour': 2, 'price': 45 },
            { '_id': 3, 'ticker': 'ABC', 'hour': 3, 'price': 60 },
            { '_id': 4, 'ticker': 'ABC', 'hour': 4, 'price': 40 },
            { '_id': 5, 'ticker': 'XYZ', 'hour': 1, 'price': 30 },
```

```
            { '_id': 6, 'ticker': 'XYZ', 'hour': 2, 'price': 35 },
            { '_id': 7, 'ticker': 'XYZ', 'hour': 3, 'price': 25 }
        ])
        window_result = list(stock_prices.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$ticker',
                    'sortBy': { 'hour': 1 },
                    'output': {
                        'firstTwoSoFar': {
                            '$firstN': { 'input': '$price', 'n': 2 },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()

example()
```

$floor

버전 4.0의 새 버전입니다.

Amazon DocumentDB의 `$floor` 연산자는 지정된 숫자보다 작거나 같은 가장 큰 정수를 반환합니다. 이 연산자는 숫자 값을 내림하는 데 유용합니다.

파라미터

- `expression`: 반올림할 숫자 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$floor` 연산자를 사용하여 십진수 값을 가장 가까운 정수로 반올림하는 방법을 보여 줍니다.

샘플 문서 생성

```
db.numbers.insertOne({ value: 3.14 });
```

쿼리 예제

```
db.numbers.aggregate([
  { $project: { _id: 0, floored: { $floor: "$value" } } }
]);
```

출력

```
{ "floored" : 3 }
```

코드 예제

$floor 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const db = client.db('test');
    const collection = db.collection('numbers');

    const result = await collection.aggregate([
      { $project: { _id: 0, floored: { $floor: "$value" } } }
    ]).toArray();

    console.log(result);

  } catch (error) {
    console.error('Error:', error);
  } finally {
```

```
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client.test
        collection = db.numbers

        result = list(collection.aggregate([
            { '$project': { '_id': 0, 'floored': { '$floor': '$value' }}}
        ]))

        pprint(result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$geoNear

$geoNear 집계 단계는 지정된 지점에 근접한 순서대로 문서를 반환합니다. 포인트로부터의 거리를 계산하고 출력 문서에 거리를 포함합니다.

파라미터

- near: GeoJSON 또는 레거시 좌표로 지정된 거리를 계산할 지점입니다.
- distanceField: 계산된 거리를 저장할 필드 이름입니다.
- spherical: 구형 지오메트리를 사용할지 여부를 나타내는 부울입니다(GeoJSON 포인트에 필요).
- maxDistance: 선택 사항입니다. 중심점으로부터의 최대 거리입니다.
- minDistance: 선택 사항입니다. 중심점과의 최소 거리입니다.
- query: 선택 사항입니다. 적용할 추가 필터 기준입니다.
- limit: 선택 사항입니다. 반환할 최대 문서 수입니다.
- key: 선택 사항입니다. 여러 지리 공간 인덱스가 있는 경우 지리 공간 쿼리에 사용할 필드입니다.

예제(MongoDB 쉘)

다음 예제에서는 $geoNear 스테이지를 사용하여 지정된 위치에 가장 가까운 스토어를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.stores.createIndex({ location: "2dsphere" });

db.stores.insertMany([
  { _id: 1, name: "Store A", location: { type: "Point", coordinates: [-122.4,
 37.8] } },
  { _id: 2, name: "Store B", location: { type: "Point", coordinates: [-122.5,
 37.7] } },
  { _id: 3, name: "Store C", location: { type: "Point", coordinates: [-122.3, 37.9] } }
]);
```

쿼리 예제

```
db.stores.aggregate([
  {
    $geoNear: {
      near: { type: "Point", coordinates: [-122.4, 37.8] },
      distanceField: "distance",
      spherical: true
    }
  }
]);
```

출력

```
[
  { _id: 1, name: 'Store A', location: { type: 'Point', coordinates: [ -122.4,
 37.8 ] }, distance: 0 },
  { _id: 3, name: 'Store C', location: { type: 'Point', coordinates: [ -122.3,
 37.9 ] }, distance: 13877.82 },
  { _id: 2, name: 'Store B', location: { type: 'Point', coordinates: [ -122.5,
 37.7 ] }, distance: 15557.89 }
]
```

코드 예제

$geoNear 집계 단계 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('stores');

  const result = await collection.aggregate([
    {
      $geoNear: {
        near: { type: "Point", coordinates: [-122.4, 37.8] },
        distanceField: "distance",
        spherical: true
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient
```

```
def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['stores']

    result = list(collection.aggregate([
        {
            '$geoNear': {
                'near': { 'type': 'Point', 'coordinates': [-122.4, 37.8] },
                'distanceField': 'distance',
                'spherical': True
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$graphLookup

버전 8.0.2에서 새로 추가되었습니다.

Amazon DocumentDB의 $graphLookup 집계 단계는 컬렉션에 대한 재귀 검색을 수행합니다. 각 입력 문서에 대해는 하나 이상의 값에서 $graphLookup 시작하여 일치하는 문서가 더 이상 없거나 선택적 재귀 깊이에 도달할 때까지 일치하는 문서 체인에 따라 연결 필드를 대상 필드와 반복적으로 일치시킵니다. 이는 조직 보고 구조, 범주 트리 또는 네트워크 관계와 같은 그래프형 또는 계층적 데이터를 통과하는 데 유용합니다.

파라미터

- from: 재귀적으로 검색할 대상 컬렉션입니다. from 모음은 동일한 데이터베이스에 있어야 합니다.
- startWith: 재귀 검색을 시작할 값을 지정하는 표현식입니다. 이 값은 from 컬렉션에 있는 문서의 connectToField와 비교됩니다. 가 배열로 startWith 평가되면 검색은 모든 배열 요소에서 동시에 실행됩니다.

- connectFromField: 값이 재귀 검색에서 다음 문서에 연결하는 데 사용되는 필드 이름입니다. 값이 배열인 경우 각 요소는 순회를 통해 개별적으로 추적됩니다.
- connectToField:의 값과 일치하는 from 컬렉션의 필드 이름입니다connectFromField.
- as: 각 출력 문서에 추가된 배열 필드의 이름입니다. 이 필드에는 재귀 검색 중에 찾은 문서가 포함되어 있습니다. 이러한 문서의 순서에 대한 자세한 내용은 다음 동작 섹션의 결과 순서 항목을 참조하세요.
- maxDepth: 초기 조회 후 수행할 최대 재귀 반복 횟수를 지정하는 음수가 아닌 정수입니다. 이 파라미터는 선택 사항입니다. 값은 초기 일치만 0 수행하고 재귀는 수행하지 않습니다. maxDepth를 생략하면 일치하는 문서가 더 이상 없을 때까지 검색이 재귀적으로 계속됩니다.
- depthField: 문서가 발견된 재귀 깊이를 포함하는 as 배열의 각 문서에 추가할 필드의 이름입니다. 이 파라미터는 선택 사항입니다. 깊이는 로 표시되고 0부터 NumberLong 시작하므로 초기 조회는 깊이에 해당합니다0.
- restrictSearchWithMatch: 검색 문서에 적용되는 추가 일치 조건을 지정하는 문서입니다. 이 파라미터는 선택 사항입니다. 조건은 $match 스테이지와 동일한 쿼리 구문을 사용하며 집계 표현식을 포함할 수 없습니다.

> **Note**
>
> $graphLookup 스테이지에는 100MB 메모리 제한이 적용되며를 지원하지 않습니다allowDiskUse:true. 이 제한을 초과하는 작업은 오류를 반환합니다. 집계에 다른 단계가 포함된 경우 allowDiskUse:true 옵션은 해당 단계에 적용됩니다.

동작

- 컬렉션 간 순회: 검색하려는 컬렉션from으로 설정합니다. 파이프라인이 실행되는 것과 동일한 컬렉션 또는 동일한 데이터베이스의 다른 컬렉션일 수 있습니다.
- 결과 순서 지정: as 배열의 문서는 어떤 보장된 순서로도 반환되지 않습니다.

예제(MongoDB 쉘)

다음 예제에서는 $graphLookup를 사용하여 직원 보고 계층 구조를 통과하고 각 직원에 대해 위의 전체 관리자 체인을 반환합니다.

다음 명령은 mongosh 셸에서 실행되고 보고 계층 구조를 설정하는 샘플 직원 문서를 삽입합니다.

```
db.employees.insertMany([
  { _id: 1, name: "Dev", reportsTo: "Eliot" },
  { _id: 2, name: "Eliot", reportsTo: "Ron" },
  { _id: 3, name: "Ron", reportsTo: "Andrew" },
  { _id: 4, name: "Andrew", reportsTo: "Asya" },
  { _id: 5, name: "Asya" }
]);
```

다음 집계는 새 reportingHierarchy 배열의 각 직원에 대한 전체 관리 체인을 반환합니다.

```
db.employees.aggregate([
  {
    $graphLookup: {
      from: "employees",
      startWith: "$reportsTo",
      connectFromField: "reportsTo",
      connectToField: "name",
      as: "reportingHierarchy",
      depthField: "level"
    }
  }
]);
```

이 작업은 다음 출력을 반환합니다.

```
[
  {
    _id: 1,
    name: 'Dev',
    reportsTo: 'Eliot',
    reportingHierarchy: [
      { _id: 2, name: 'Eliot', reportsTo: 'Ron', level: 0 },
      { _id: 3, name: 'Ron', reportsTo: 'Andrew', level: 1 },
      { _id: 4, name: 'Andrew', reportsTo: 'Asya', level: 2 },
      { _id: 5, name: 'Asya', level: 3 }
    ]
  },
  {
    _id: 2,
    name: 'Eliot',
    reportsTo: 'Ron',
    reportingHierarchy: [
```

```
      { _id: 3, name: 'Ron', reportsTo: 'Andrew', level: 0 },
      { _id: 4, name: 'Andrew', reportsTo: 'Asya', level: 1 },
      { _id: 5, name: 'Asya', level: 2 }
    ]
  },
  {
    _id: 3,
    name: 'Ron',
    reportsTo: 'Andrew',
    reportingHierarchy: [
      { _id: 4, name: 'Andrew', reportsTo: 'Asya', level: 0 },
      { _id: 5, name: 'Asya', level: 1 }
    ]
  },
  {
    _id: 4,
    name: 'Andrew',
    reportsTo: 'Asya',
    reportingHierarchy: [
      { _id: 5, name: 'Asya', level: 0 }
    ]
  },
  {
    _id: 5,
    name: 'Asya',
    reportingHierarchy: []
  }
]
```

각 직원 문서는 재귀적으로 일치하는 관리자 문서가 포함된 `reportingHierarchy` 배열을 얻습니다. `level` 필드는 각 관리자가 발견된 재귀 깊이를 기록합니다. Asya에는 `reportsTo` 값이 없으므로 `reportingHierarchy` 배열이 비어 있습니다.

maxDepth 및 restrictSearchWithMatch를 사용한 예제(MongoDB Shell)

다음 예제에서는 사람 간의 연결 네트워크를 통과하고를 사용하여 재귀`maxDepth`를 제한하고 추가 조건과 일치하는 문서만 `restrictSearchWithMatch` 포함합니다. 검색은 시작하는 사람의 연결을 최대 두 수준 깊이까지 찾아 `hobbies` 배열에가 포함된 사람만 유지합니다`golf`.

다음 명령은 mongosh 셸에서 실행되며 각 사람이 단방향 연결 체인을 통해 앞으로 `friends` 가리키는 샘플 문서를 삽입합니다.

```
db.people.insertMany([
```

```
  { _id: 1, name: "Tanya", friends: [ "Carole", "Shirley" ], hobbies: [ "tennis",
 "golf" ] },
  { _id: 2, name: "Carole", friends: [ "Joseph" ], hobbies: [ "archery", "golf" ] },
  { _id: 3, name: "Joseph", friends: [ "Angelo" ], hobbies: [ "tennis", "golf" ] },
  { _id: 4, name: "Angelo", friends: [ "Miguel" ], hobbies: [ "travel", "golf" ] },
  { _id: 5, name: "Miguel", friends: [], hobbies: [ "golf" ] },
  { _id: 6, name: "Shirley", friends: [], hobbies: [ "frisbee" ] }
]);
```

다음 집계는 Tanya에서 시작하여 최대 두 레벨 깊이까지 연결 네트워크를 통과하고를 취미golf로 나열하는 사람만 유지합니다.

```
db.people.aggregate([
  { $match: { name: "Tanya" } },
  {
    $graphLookup: {
      from: "people",
      startWith: "$friends",
      connectFromField: "friends",
      connectToField: "name",
      as: "golfConnections",
      maxDepth: 2,
      restrictSearchWithMatch: { hobbies: "golf" }
    }
  },
  { $project: { name: 1, "golfConnections.name": 1 } }
]);
```

이 작업은 다음 출력을 반환합니다.

```
[
  {
    _id: 1,
    name: 'Tanya',
    golfConnections: [
      { name: 'Carole' },
      { name: 'Joseph' },
      { name: 'Angelo' }
    ]
  }
]
```

hobbies 배열에 포함된 연결만 포함되며 두 수준의 재귀 후 순회가 중지golf됩니다. Shirley는를 나열하지 않기 때문에 제외golf되고 Miguel은 maxDepth의를 넘어 Tanya에서 3개 레벨 떨어져 있기 때문에 제외됩니다2. 연결은 Tanya를 가리키지 않는 단방향 체인을 구성하므로 재귀 검색 중에 시작 문서가 일치하지 않습니다.

코드 예제

MongoDB 드라이버를 사용하여 첫 번째 예제에서 직원 보고 계층 구조를 빌드하는 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const employees = db.collection('employees');

    const result = await employees.aggregate([
      {
        $graphLookup: {
          from: 'employees',
          startWith: '$reportsTo',
          connectFromField: 'reportsTo',
          connectToField: 'name',
          as: 'reportingHierarchy',
          depthField: 'level'
        }
      }
    ]).toArray();

    console.log(JSON.stringify(result, null, 2));
  } catch (err) {
    console.error('Failed to run $graphLookup aggregation:', err.message);
  } finally {
    await client.close();
  }
```

```
}

example();
```

Python

```
from pymongo import MongoClient
from pymongo.errors import PyMongoError

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']
        employees = db['employees']

        pipeline = [
            {
                "$graphLookup": {
                    "from": "employees",
                    "startWith": "$reportsTo",
                    "connectFromField": "reportsTo",
                    "connectToField": "name",
                    "as": "reportingHierarchy",
                    "depthField": "level"
                }
            }
        ]

        result = list(employees.aggregate(pipeline))

        for doc in result:
            print(doc)
    except PyMongoError as error:
        print(f"Failed to run $graphLookup aggregation: {error}")
    finally:
        client.close()

example()
```

$group

Amazon DocumentDB의 $group 집계 단계를 사용하면 지정된 표현식을 기준으로 문서를 그룹화하고 그룹화된 데이터에 대해 다양한 누적 작업을 수행할 수 있습니다. 이는 그룹화된 데이터를 기반으로 합계, 평균 또는 기타 통계를 계산하는 등의 작업에 유용할 수 있습니다.

파라미터

- `_id`: 입력 문서를 그룹화해야 하는 표현식을 지정합니다. 필드 이름, 계산된 표현식 또는 둘 다일 수 있습니다.
- `accumulator expressions`: (선택 사항) 그룹화된 데이터에 적용해야 하는 하나 이상의 누적기 표현식입니다. 이러한 표현식은 위에서 언급한 누적기 연산자를 사용합니다.

예제(MongoDB 쉘)

다음 예시에서는 고객을 도시별로 그룹화하고 각 도시의 총 주문 금액을 계산합니다.

샘플 문서 생성

```
db.customers.insertMany([
  { name: "John Doe", city: "New York", orders: [{ amount: 100 }, { amount: 200 }] },
  { name: "Jane Smith", city: "Los Angeles", orders: [{ amount: 150 }, { amount:
 300 }] },
  { name: "Bob Johnson", city: "New York", orders: [{ amount: 75 }, { amount: 125 }] },
  { name: "Samantha Lee", city: "Chicago", orders: [{ amount: 50 }, { amount: 100 }] }
]);
```

쿼리 예제

```
db.customers.aggregate([
  {
    $group: {
      _id: "$city",
      totalOrders: { $sum: { $sum: "$orders.amount" } }
    }
  }
]);
```

출력

```
[
  { _id: 'Chicago', totalOrders: 150 },
  { _id: 'Los Angeles', totalOrders: 450 },
  { _id: 'New York', totalOrders: 500 }
]
```

코드 예제

$group 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function groupByCity() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();
    const db = client.db('test');
    const result = await db.collection('customers').aggregate([
      { $unwind: '$orders' },
      {
        $group: {
          _id: '$city',
          totalOrders: { $sum: '$orders.amount' }
        }
      }
    ]).toArray();

    console.log(result);
  } catch (err) {
    console.error('Error during aggregation:', err);
  } finally {
    await client.close();
  }
}

groupByCity();
```

Python

```
from pymongo import MongoClient

def group_by_city():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client.test
        result = list(db.customers.aggregate([
            {'$unwind': '$orders'},
            {
                '$group': {
                    '_id': '$city',
                    'totalOrders': {'$sum': '$orders.amount'}
                }
            }
        ]))
        print(result)
    except Exception as e:
        print(f"Error during aggregation: {e}")
    finally:
        client.close()

group_by_city()
```

$gt

$gt 집계 연산자는 두 값을 비교하고 첫 번째 값이 두 번째 값보다 크true면를 반환하고, 그렇지 않으면를 반환합니다false.

파라미터

- expression1: 비교할 첫 번째 값입니다.
- expression2: 비교할 두 번째 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 $gt 연산자를 사용하여 가격 임계값을 초과하는 제품을 식별하는 방법을 보여줍니다.

샘플 문서 생성

```
db.products.insertMany([
  { _id: 1, name: "Laptop", price: 1200 },
  { _id: 2, name: "Mouse", price: 25 },
  { _id: 3, name: "Monitor", price: 400 }
]);
```

쿼리 예제

```
db.products.aggregate([
  {
    $project: {
      name: 1,
      price: 1,
      expensive: { $gt: ["$price", 100] }
    }
  }
]);
```

출력

```
[
  { _id: 1, name: 'Laptop', price: 1200, expensive: true },
  { _id: 2, name: 'Mouse', price: 25, expensive: false },
  { _id: 3, name: 'Monitor', price: 400, expensive: true }
]
```

코드 예제

$gt 집계 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
        price: 1,
        expensive: { $gt: ["$price", 100] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['products']

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'price': 1,
                'expensive': { '$gt': ['$price', 100] }
            }
        }
    ]))
```

```
    print(result)
    client.close()

example()
```

$gte

$gte 집계 연산자는 두 값을 비교하고 첫 번째 값이 두 번째 값보다 크거나 같true으면를 반환하고, 그렇지 않으면를 반환합니다false.

파라미터

- expression1: 비교할 첫 번째 값입니다.
- expression2: 비교할 두 번째 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 $gte 연산자를 사용하여 학생이 시험을 통과했는지 확인하는 방법을 보여줍니다.

샘플 문서 생성

```
db.students.insertMany([
  { _id: 1, name: "Alice", score: 85 },
  { _id: 2, name: "Bob", score: 60 },
  { _id: 3, name: "Charlie", score: 72 }
]);
```

쿼리 예제

```
db.students.aggregate([
  {
    $project: {
      name: 1,
      score: 1,
      passed: { $gte: ["$score", 70] }
    }
  }
]);
```

출력

```
[
  { _id: 1, name: 'Alice', score: 85, passed: true },
  { _id: 2, name: 'Bob', score: 60, passed: false },
  { _id: 3, name: 'Charlie', score: 72, passed: true }
]
```

코드 예제

$gte 집계 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('students');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
        score: 1,
        passed: { $gte: ["$score", 70] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient
```

```
def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['students']

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'score': 1,
                'passed': { '$gte': ['$score', 70] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$hour

$hour 연산자는 날짜 또는 타임스탬프 필드에서 시간 구성 요소를 추출합니다.

파라미터

- dateExpression: 연산자가 적용되는 날짜입니다. 이는 유효한 BSON 날짜(예: $createdAt 또는 날짜 리터럴과 같은 필드)로 확인되어야 합니다.

파라미터는 다음 형식의 문서로 지정할 수도 있습니다.

{ 날짜: <dateExpression>, 시간대: <timezoneExpression> }

이렇게 하면가 시간대 인식 날짜 작업을 적용할 수 있습니다.

```
- `<tzExpression>`: (optional) The timezone of the operation result. It must be a valid
 expression that resolves to a string formatted as either an Olson Timezone Identifier
 or a UTC Offset. If no timezone is provided, the result is in UTC.
```

예제(MongoDB 쉘)

다음 예제에서는 $hour 연산자를 사용하여 날짜 필드에서 시간 구성 요소를 추출하고 그에 따라 데이터를 그룹화하는 방법을 보여줍니다.

샘플 문서 생성

```
db.events.insertMany([
  { timestamp: new Date("2023-04-01T10:30:00Z") },
  { timestamp: new Date("2023-04-01T12:45:00Z") },
  { timestamp: new Date("2023-04-02T08:15:00Z") },
  { timestamp: new Date("2023-04-02T16:20:00Z") },
  { timestamp: new Date("2023-04-03T23:59:00Z") }
]);
```

쿼리 예제

```
db.events.aggregate([
  {
    $project: {
      hour: { $hour: "$timestamp" }
    }
  },
  {
    $group: {
      _id: "$hour",
      count: { $sum: 1 }
    }
  },
  {
    $sort: { _id: 1 }
  }
]);
```

출력

```
[
  { "_id": 8, "count": 1 },
  { "_id": 10, "count": 1 },
  { "_id": 12, "count": 1 },
  { "_id": 16, "count": 1 },
  { "_id": 23, "count": 1 }
```

```
]
```

이 쿼리는 timestamp 필드의 시간 구성 요소를 기준으로 이벤트를 그룹화하고 각 시간에 대한 이벤트 수를 계산합니다.

코드 예제

$hour 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('events');

    const result = await collection.aggregate([
      {
        $project: {
          hour: { $hour: "$timestamp" }
        }
      },
      {
        $group: {
          _id: "$hour",
          count: { $sum: 1 }
        }
      },
      {
        $sort: { _id: 1 }
      }
    ]).toArray();

    console.log(result);
  } catch (error) {
    console.error('Error occurred:', error);
```

```
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from datetime import datetime

def example():
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client.test
        collection = db.events

        result = list(collection.aggregate([
            {
                "$project": {
                    "hour": {"$hour": "$timestamp"}
                }
            },
            {
                "$group": {
                    "_id": "$hour",
                    "count": {"$sum": 1}
                }
            },
            {
                "$sort": {"_id": 1}
            }
        ]))

        print(result)

    except Exception as e:
        print(f"An error occurred: {e}")
```

```
    finally:
        client.close()

example()
```

$ifNull

이 $ifNull 연산자는 입력 표현식이 null 또는 정의되지 않은 것으로 평가되는 경우 지정된 값을 반환하는 데 사용됩니다. 이 연산자는 기본값을 제공하거나 null/정의되지 않은 사례를 처리하려는 시나리오에서 유용할 수 있습니다.

파라미터

- expression: 평가할 표현식입니다.
- replacement:가 null 또는 정의되지 않은 것으로 <expression> 평가되는 경우 반환할 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 name 필드가 null이거나 정의되지 않은 경우 연$ifNull산자를 사용하여 기본값을 제공하는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { _id: 1, name: "John" },
  { _id: 2, name: null },
  { _id: 3 }
]);
```

쿼리 예제

```
db.users.aggregate([
  {
    $project: {
      _id: 1,
      name: { $ifNull: ["$name", "No Name"] }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "name": "John" },
  { "_id": 2, "name": "No Name" },
  { "_id": 3, "name": "No Name" }
]
```

코드 예제

`$ifNull` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  const db = client.db('test');

  const collection = db.collection('users');

  const pipeline = [
    {
      $project: {
        _id: 1,
        name: { $ifNull: ["$name", "No Name"] }
      }
    }
  ];

  const cursor = await collection.aggregate(pipeline);

  await cursor.forEach(doc => {
    console.log(doc);
  });

  await client.close();

}
```

```
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    db = client.test
    collection = db.users

    pipeline = [
        {
            "$project": {
                "_id": 1,
                "name": { "$ifNull": ["$name", "No Name"] }
            }
        }
    ]

    result = collection.aggregate(pipeline)

    for doc in result:
        print(doc)

    client.close()

example()
```

$in

`$in` 집계 연산자는 배열 내에 지정된 값이 존재하는지 확인합니다. 배열에서 값이 발견`true`되면를 반환하고, `false` 그렇지 않으면를 반환합니다.

파라미터

- `value`: 검색할 값입니다.
- `array`: 검색할 배열입니다.

예제(MongoDB 쉘)

다음 예제에서는 $in 연산자를 사용하여 각 직원의 스킬 세트에 특정 스킬이 있는지 확인하는 방법을 보여줍니다.

샘플 문서 생성

```
db.employees.insertMany([
  { _id: 1, name: "Sarah", skills: ["Python", "JavaScript", "SQL"] },
  { _id: 2, name: "Mike", skills: ["Java", "C++", "Go"] },
  { _id: 3, name: "Emma", skills: ["Python", "Ruby", "Rust"] }
]);
```

쿼리 예제

```
db.employees.aggregate([
  {
    $project: {
      name: 1,
      hasPython: { $in: ["Python", "$skills"] }
    }
  }
]);
```

출력

```
[
  { _id: 1, name: 'Sarah', hasPython: true },
  { _id: 2, name: 'Mike', hasPython: false },
  { _id: 3, name: 'Emma', hasPython: true }
]
```

코드 예제

$in 집계 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('employees');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
        hasPython: { $in: ["Python", "$skills"] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['employees']

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'hasPython': { '$in': ['Python', '$skills'] }
            }
        }
    ]))

    print(result)
    client.close()
```

```
example()
```

$indexOfArray

Amazon DocumentDB의 `$indexOfArray` 연산자는 배열에서 지정된 요소의 첫 번째 발생 인덱스를 찾는 데 사용됩니다. 이 연산자는 지정된 값과 일치하는 배열에서 첫 번째 요소의 0 기반 인덱스 위치를 반환합니다. 값을 찾을 수 없는 경우 -1을 반환합니다.

파라미터

- `array`: 검색할 배열입니다.
- `value`: 배열에서 검색할 값입니다.
- `start`: (선택 사항) 검색을 시작할 배열의 위치입니다. 기본값은 0입니다.

예제(MongoDB 쉘)

다음 예제에서는 $indexOfArray 연산자를 사용하여 각 문서의 "fruits" 배열에서 요소 "mango"의 첫 번째 발생 인덱스를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.collection.insertMany([
  { _id: 1, fruits: ["apple", "banana", "cherry", "durian"] },
  { _id: 2, fruits: ["mango", "orange", "pineapple"] },
  { _id: 3, fruits: ["kiwi", "lemon", "mango"] }
]);
```

쿼리 예제

```
db.collection.aggregate([
  {
    $project: {
      _id: 1,
      fruitIndex: { $indexOfArray: ["$fruits", "mango"] }
    }
  }
]);
```

출력

```
{ "_id" : 1, "fruitIndex" : 1 }
{ "_id" : 2, "fruitIndex" : 0 }
{ "_id" : 3, "fruitIndex" : 2 }
```

코드 예제

$indexOfArray 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        fruitIndex: { $indexOfArray: ["$fruits", "mango"] }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
    db = client['test']
    collection = db['collection']

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'fruitIndex': { '$indexOfArray': ["$fruits", "mango"] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$indexOfBytes

Amazon DocumentDB의 $indexOfBytes 연산자는 문자의 바이트 위치에 따라 문자열 내에서 하위 문자열의 시작 인덱스를 찾는 데 사용됩니다. 이는 비 라틴어 스크립트에 있는 것과 같이 멀티바이트 문자를 포함할 수 있는 텍스트 데이터로 작업할 때 유용할 수 있습니다.

파라미터

- `string`: 검색할 입력 문자열입니다.
- `substring`: 입력 문자열 내에서 검색할 하위 문자열입니다.
- `[<start>]`: (선택 사항) 검색의 시작 위치(0 기반)입니다. 지정하지 않으면 문자열의 시작 부분에서 검색이 시작됩니다.

예제(MongoDB 쉘)

다음 예제에서는를 사용하여 데스크 위치를 나타내는 문자열 집합에서 첫 번째 하이픈 문자의 인덱스를 찾는 `$indexOfBytes` 방법을 보여줍니다.

샘플 문서 생성

```
db.people.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-BVV-021" },
```

```
  { "_id": 2, "Desk": "Munich-HGG-32a" },
  { "_id": 3, "Desk": "Cologne-ayu-892.50" },
  { "_id": 4, "Desk": "Dortmund-Hop-78" }
]);
```

쿼리 예제

```
db.people.aggregate([
  { $project: { stateLocation: { $indexOfBytes: ["$Desk", "-"] } } }
]);
```

출력

```
{ "_id" : 1, "stateLocation" : 11 }
{ "_id" : 2, "stateLocation" : 6 }
{ "_id" : 3, "stateLocation" : 7 }
{ "_id" : 4, "stateLocation" : 8 }
```

코드 예제

$indexOfBytes 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const events = db.collection('people');

  const result = await db.collection('people').aggregate([
    { $project: { stateLocation: { $indexOfBytes: ["$Desk", "-"] } } }
  ]).toArray();
  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['people']

    result = list(db.people.aggregate([
        { '$project': { 'stateLocation': { '$indexOfBytes': ['$Desk', '-'] } } }
    ]))
    print(result)
    client.close()

example()
```

$indexOfCP

Amazon DocumentDB의 `$indexOfCP` 연산자는 문자열 표현식 내에서 지정된 하위 문자열이 처음 발생한 인덱스를 코드 포인트(CP)에서 찾는 데 사용됩니다. 이는 문자열 필드에서 콘텐츠를 구문 분석하고 추출할 때 유용할 수 있습니다.

파라미터

- `string expression`: 검색할 문자열입니다.
- `substring`: 검색할 하위 문자열입니다.
- `[<start>]`: (선택 사항) 검색을 시작할 위치(제로 기반 인덱스). 기본값은 0.

예제(MongoDB 쉘)

이 예제에서는 `$indexOfCP` 연산자를 사용하여 각 문서의 데스크 필드에서 하이픈 문자 -의 첫 번째 발생 인덱스를 찾습니다.

샘플 문서 생성

```
db.people.insertMany([
```

```
  { "_id":1, "name":"John Doe", "Manager":"Jane Doe", "Role":"Developer", "Desk":
 "Düsseldorf-BVV-021"},
  { "_id":2, "name":"John Stiles", "Manager":"Jane Doe", "Role":"Manager", "Desk":
 "Munich-HGG-32a"},
  { "_id":3, "name":"Richard Roe", "Manager":"Jorge Souza", "Role":"Product", "Desk":
 "Cologne-ayu-892.50"},
  { "_id":4, "name":"Mary Major", "Manager":"Jane Doe", "Role":"Solution Architect",
 "Desk": "Dortmund-Hop-78"}
])
```

쿼리 예제

```
db.people.aggregate([
  { $project: { stateLocation: { $indexOfCP: [ "$Desk", "-"] } } }
])
```

출력

```
{ "_id" : 1, "stateLocation" : 10 }
{ "_id" : 2, "stateLocation" : 6 }
{ "_id" : 3, "stateLocation" : 7 }
{ "_id" : 4, "stateLocation" : 8 }
```

코드 예제

`$indexOfCP` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const result = await collection.aggregate([
    { $project: { stateLocation: { $indexOfCP: [ "$Desk", "-"] } } }
  ]).toArray();

  console.log(result);
```

```
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['people']

result = list(collection.aggregate([
    { '$project': { 'stateLocation': { '$indexOfCP': [ '$Desk', '-' ] } } }
]))

print(result)

client.close()
```

$indexStats

Amazon DocumentDB의 `$indexStats` 집계 단계는 컬렉션 내 인덱스 사용에 대한 인사이트를 제공합니다. 이 연산자를 사용하면 인덱스의 액세스 패턴을 모니터링할 수 있으므로 인덱스 관리 및 최적화에 대해 정보에 입각한 결정을 내리는 데 도움이 될 수 있습니다.

파라미터

없음

예제(MongoDB 쉘)

다음 예제에서는 `$indexStats` 연산자를 사용하여 Amazon DocumentDB 컬렉션의 인덱스 사용량을 분석하는 방법을 보여줍니다.

샘플 문서 생성

```
db.grocery.insertMany([
```

```
  { _id: 1, product: "milk", quantity: 10 },
  { _id: 2, product: "eggs", quantity: 20 },
  { _id: 3, product: "bread", quantity: 5 },
  { _id: 4, product: "cheese", quantity: 15 },
  { _id: 5, product: "apple", quantity: 8 }
]);
```

쿼리 예제

```
db.grocery.aggregate([
  { $indexStats: {} }
]);
```

출력

```
[
  {
    "name": "_id_",
    "key": {
      "_id": 1
    },
    "host": "docdb-cluster-1.cluster-123456789.us-west-2.docdb.amazonaws.com",
    "accesses": {
      "ops": NumberLong(5),
      "since": ISODate("2023-04-06T12:34:56.789Z")
    }
  },
  {
    "name": "product_1",
    "key": {
      "product": 1
    },
    "host": "docdb-cluster-1.cluster-123456789.us-west-2.docdb.amazonaws.com",
    "accesses": {
      "ops": NumberLong(10),
      "since": ISODate("2023-04-06T12:34:56.789Z")
    }
  }
]
```

이 예제에서 `$indexStats` 연산자는 `_id_` 인덱스에 5번 액세스했고 마지막 재설정 또는 서버 재부팅 이후 `product_1` 인덱스에 10번 액세스했음을 보여줍니다.

코드 예제

`$indexStats` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function indexStats() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const result = await db.collection('grocery').aggregate([
    { $indexStats: {} }
  ]).toArray();
  console.log(result);
  await client.close();
}

indexStats();
```

Python

```
from pymongo import MongoClient

def index_stats():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    result = list(db.grocery.aggregate([
        { '$indexStats': {} }
    ]))
    print(result)
    client.close()

index_stats()
```

$isArray

Amazon DocumentDB의 `$isArray` 연산자는 문서의 필드가 배열인지 확인하는 데 사용됩니다. 이 연산자는 집계 파이프라인 및 조건식에서 배열 유형 필드를 처리하는 데 유용할 수 있습니다.

파라미터

- `field`: 배열인지 확인할 필드 경로입니다.

예제(MongoDB 쉘)

이 예제에서는 `$isArray` 연산자를 사용하여 "인벤토리" 필드가 배열인 문서를 식별하는 방법을 보여줍니다.

샘플 문서 생성

```
db.videos.insertMany([
  { "_id":1, "name":"Live Soft", "inventory": {"Des Moines": 1000, "Ames" : 500}},
  { "_id":2, "name":"Top Pilot", "inventory": {"Mason City": 250, "Des Moines": 1000}},
  { "_id":3, "name":"Romancing the Rock", "inventory": {"Mason City": 250, "Ames" :
 500}},
  { "_id":4, "name":"Bravemind", "inventory": [{"location": "Mason City", "count":
 250}, {"location": "Des Moines", "count": 1000}, {"location": "Ames", "count": 500}]}
]);
```

쿼리 예제

```
db.videos.aggregate([
  {
    $match: {
      $isArray: "$inventory"
    }
  },
  {
    $project: {
      _id: 1,
      name: 1,
      "inventory.location": 1,
      "inventory.count": 1
    }
  }
]).pretty();
```

출력

```
{
  "_id": 4,
  "name": "Bravemind",
  "inventory": [
    {
      "location": "Mason City",
      "count": 250
    },
    {
      "location": "Des Moines",
      "count": 1000
    },
    {
      "location": "Ames",
      "count": 500
    }
  ]
}
```

코드 예제

$isArray 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function run() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('videos');

  const result = await collection.aggregate([
    {
      $match: {
        $isArray: '$inventory'
      }
    },
    {
```

```
      $project: {
        _id: 1,
        name: 1,
        "inventory.location": 1,
        "inventory.count": 1
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

run();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['videos']

result = list(collection.aggregate([
    {
        '$match': {
            '$isArray': '$inventory'
        }
    },
    {
        '$project': {
            '_id': 1,
            'name': 1,
            'inventory.location': 1,
            'inventory.count': 1
        }
    }
]))

print(result)
```

```
client.close()
```

$isNumber

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $isNumber 연산자는 지정된 표현식이 숫자 유형(정수, 소수, 이중, 길이)으로 확인되는지 여부를 나타내는 부울을 반환합니다. 배열의 모든 요소가 숫자인 경우에도 배열은 숫자로 간주되지 않습니다. 예를 들어 값에 $isNumber 적용된는를 [1, 2, 3] 반환합니다false.

파라미터

- expression: 숫자 유형으로 확인되는지 여부를 확인하는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $isNumber 연산자를 사용하여 필드 값이 숫자인지 확인하는 방법을 보여줍니다.

샘플 문서 생성

```
db.data.insertMany([
  {_id: 1, value: 42},
  {_id: 2, value: "hello"},
  {_id: 3, value: 3.14},
  {_id: 4, value: null}
]);
```

쿼리 예제

```
db.data.aggregate([
  { $project: { isNum: { $isNumber: "$value" } } }
]);
```

출력

```
[
  {_id: 1, isNum: true},
  {_id: 2, isNum: false},
  {_id: 3, isNum: true},
```

```
  {_id: 4, isNum: false}
]
```

코드 예제

$isNumber 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('data');
    const result = await collection.aggregate([
      { $project: { isNum: { $isNumber: "$value" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['data']
        result = list(collection.aggregate([
            {'$project': {'isNum': {'$isNumber': '$value'}}}
        ]))
```

```
        print(result)
    finally:
        client.close()

example()
```

$isoDayOfWeek

Amazon DocumentDB의 $isoDayOfWeek 연산자는 날짜의 ISO 요일을 정수 값으로 반환합니다. ISO 주 날짜 시스템은 매주 월요일부터 일요일까지를 정의하며, 1주차는 해당 연도의 첫 번째 목요일이 포함된 주입니다.

파라미터

- expression: ISO 요일을 반환할 날짜 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $isoDayOfWeek 연산자를 사용하여 이벤트 문서 세트에 대한 ISO 요일을 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.events.insertMany([
  { _id: 1, eventDate: ISODate("2023-04-01T12:00:00Z") },
  { _id: 2, eventDate: ISODate("2023-04-02T12:00:00Z") },
  { _id: 3, eventDate: ISODate("2023-04-03T12:00:00Z") },
  { _id: 4, eventDate: ISODate("2023-04-04T12:00:00Z") },
  { _id: 5, eventDate: ISODate("2023-04-05T12:00:00Z") },
  { _id: 6, eventDate: ISODate("2023-04-06T12:00:00Z") },
  { _id: 7, eventDate: ISODate("2023-04-07T12:00:00Z") }
]);
```

쿼리 예제

```
db.events.aggregate([
  { $project: {
    _id: 1,
    eventDate: 1,
    isoDayOfWeek: { $isoDayOfWeek: "$eventDate" }
  }}
```

```
]);
```

출력

```
[
  { "_id": 1, "eventDate": ISODate("2023-04-01T12:00:00Z"), "isoDayOfWeek": 6 },
  { "_id": 2, "eventDate": ISODate("2023-04-02T12:00:00Z"), "isoDayOfWeek": 7 },
  { "_id": 3, "eventDate": ISODate("2023-04-03T12:00:00Z"), "isoDayOfWeek": 1 },
  { "_id": 4, "eventDate": ISODate("2023-04-04T12:00:00Z"), "isoDayOfWeek": 2 },
  { "_id": 5, "eventDate": ISODate("2023-04-05T12:00:00Z"), "isoDayOfWeek": 3 },
  { "_id": 6, "eventDate": ISODate("2023-04-06T12:00:00Z"), "isoDayOfWeek": 4 },
  { "_id": 7, "eventDate": ISODate("2023-04-07T12:00:00Z"), "isoDayOfWeek": 5 }
]
```

코드 예제

`$isoDayOfWeek` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const events = db.collection('events');

  const result = await events.aggregate([
    {
      $project: {
        _id: 1,
        eventDate: 1,
        isoDayOfWeek: { $isoDayOfWeek: '$eventDate' }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}
```

```
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    events = db.events

    result = list(events.aggregate([
        {
            '$project': {
                '_id': 1,
                'eventDate': 1,
                'isoDayOfWeek': { '$isoDayOfWeek': '$eventDate' }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$isoWeek

Amazon DocumentDB의 `$isoWeek` 연산자는 날짜의 ISO 주 번호를 반환합니다. ISO 주 날짜 시스템은 한 해의 주 번호를 매기는 방법으로, 새해의 첫 번째 주가 해당 연도의 첫 번째 목요일이 포함된 주입니다. 이는 새해의 첫 주가 1월 1일이 포함된 주인 그레고리 달력과 다릅니다.

파라미터

없음

예제(MongoDB 쉘)

다음 예제에서는 `$isoWeek` 연산자를 사용하여 지정된 날짜의 ISO 주 번호를 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.dates.insertMany([
  { _id: 1, date: new ISODate("2022-01-01") },
  { _id: 2, date: new ISODate("2022-12-31") },
  { _id: 3, date: new ISODate("2023-01-01") }
])
```

쿼리 예제

```
db.dates.aggregate([
  {
    $project: {
      _id: 1,
      isoWeek: { $isoWeek: "$date" }
    }
  }
])
```

출력

```
[
  { "_id": 1, "isoWeek": 52 },
  { "_id": 2, "isoWeek": 52 },
  { "_id": 3, "isoWeek": 1 }
]
```

코드 예제

`$isoWeek` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('dates');
```

```
  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        isoWeek: { $isoWeek: "$date" }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['dates']

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'isoWeek': { '$isoWeek': '$date' }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$isoWeekYear

Amazon DocumentDB의 `$isoWeekYear` 연산자는 지정된 날짜에 대한 ISO 8601 주 연도 번호를 반환합니다. ISO 주 연도 수는 특히 연중 시작 및 종료 시 해당 주 연도가 해당 연도와 다를 수 있다는 점에서 그레고리 역년과 다릅니다.

파라미터

- `expression`: ISO 8601 주 연도 번호를 반환할 날짜 표현식입니다.

예제(MongoDB 쉘)

이 예제에서는 `$isoWeekYear` 연산자를 사용하여 이벤트 컬렉션에 있는 각 문서의 날짜 필드에 대한 ISO 8601 주 연도를 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.events.insertMany([
  { _id: 1, name: "Event 1", date: ISODate("2022-12-31T00:00:00Z") },
  { _id: 2, name: "Event 2", date: ISODate("2023-01-01T00:00:00Z") },
  { _id: 3, name: "Event 3", date: ISODate("2023-01-02T00:00:00Z") }
]);
```

쿼리 예제

```
db.events.aggregate([
  { $project: {
    name: 1,
    isoWeekYear: { $isoWeekYear: "$date" }
  }}
]);
```

출력

```
[
  { "_id" : 1, "name" : "Event 1", "isoWeekYear" : 2023 },
  { "_id" : 2, "name" : "Event 2", "isoWeekYear" : 2023 },
  { "_id" : 3, "name" : "Event 3", "isoWeekYear" : 2023 }
]
```

코드 예제

$isoWeekYear 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const events = db.collection('events');

  const result = await events.aggregate([
    { $project: {
      name: 1,
      isoWeekYear: { $isoWeekYear: '$date' }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    events = db.events

    result = list(events.aggregate([
        { '$project': {
            'name': 1,
            'isoWeekYear': { '$isoWeekYear': '$date' }
```

```
        }}
    ]))

    print(result)
    client.close()

example()
```

$last

Amazon DocumentDB의 $last 연산자는 쿼리 기준과 일치하는 배열의 마지막 요소를 반환하는 데 사용됩니다. 특정 조건을 충족하는 배열에서 가장 최근 또는 마지막 요소를 검색하는 데 특히 유용합니다.

파라미터

- expression: 배열 요소와 일치하는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 연$last산자를와 함께 사용하여 특정 조건(예: 제목이 '과학')을 충족하는 배열에서 마지막 요소를 $filter 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.collection.insertMany([
  {
    "_id": 1,
    "name": "John",
    "scores": [
      { "subject": "math", "score": 82 },
      { "subject": "english", "score": 85 },
      { "subject": "science", "score": 90 }
    ]
  },
  {
    "_id": 2,
    "name": "Jane",
    "scores": [
      { "subject": "math", "score": 92 },
      { "subject": "english", "score": 88 },
```

```
      { "subject": "science", "score": 87 }
    ]
  },
  {
    "_id": 3,
    "name": "Bob",
    "scores": [
      { "subject": "math", "score": 75 },
      { "subject": "english", "score": 80 },
      { "subject": "science", "score": 85 }
    ]
  }
]);
```

쿼리 예제

```
db.collection.aggregate([
    { $match: { name: "John" } },
    {
      $project: {
        name: 1,
        lastScienceScore: {
          $last: {
            $filter: {
              input: "$scores",
              as: "score",
              cond: { $eq: ["$$score.subject", "science"] }
            }
          }
        }
      }
    }
  ]);
```

출력

```
[
  {
    _id: 1,
    name: 'John',
    lastScienceScore: { subject: 'science', score: 90 }
  }
```

```
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$last산자는 $setWindowFields 스테이지에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 expression 마지막 문서에서의 값을 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

샘플 문서 생성

```
db.stockPrices.insertMany([
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

쿼리 예제

다음 예제에서는 문서를 기준으로 분할하고ticker, 각 파티션을 기준으로 정렬하고hour, 각 파티션에 기록된 마지막(최신) 가격을 반환합니다.

```
db.stockPrices.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        lastPrice: {
          $last: "$price",
          window: { documents: ["unbounded", "unbounded"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "lastPrice": 40 },
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "lastPrice": 40 },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "lastPrice": 40 },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "lastPrice": 40 },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "lastPrice": 25 },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "lastPrice": 25 },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "lastPrice": 25 }
]
```

각 문서는 창의 마지막 문서(hour파티션의 최신 문서)의 가격lastPrice인 로 보강됩니다.

코드 예제

$last 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 표현식 사용량()과 창 연산자 사용량($project)을 모두 보여줍니다$setWindowFields.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  const db = client.db('test');

  // Expression usage: last element of a filtered array
  const collection = db.collection('collection');
  await collection.insertMany([
    {
      _id: 1,
      name: "John",
      scores: [
        { subject: "math", score: 82 },
        { subject: "english", score: 85 },
        { subject: "science", score: 90 }
      ]
    },
    {
      _id: 2,
      name: "Jane",
```

```
    scores: [
      { subject: "math", score: 92 },
      { subject: "english", score: 88 },
      { subject: "science", score: 87 }
    ]
  },
  {
    _id: 3,
    name: "Bob",
    scores: [
      { subject: "math", score: 75 },
      { subject: "english", score: 80 },
      { subject: "science", score: 85 }
    ]
  }
]);
const expressionResult = await collection.aggregate([
  { $match: { name: "John" } },
  {
    $project: {
      name: 1,
      lastScienceScore: {
        $last: {
          $filter: {
            input: "$scores",
            as: "score",
            cond: { $eq: ["$$score.subject", "science"] }
          }
        }
      }
    }
  }
]).toArray();
console.log('Expression result:', JSON.stringify(expressionResult, null, 2));

// Window operator usage: last value within each partition
const stockPrices = db.collection('stockPrices');
await stockPrices.insertMany([
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
```

```
    { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
  ]);
  const windowResult = await stockPrices.aggregate([
    {
      $setWindowFields: {
        partitionBy: "$ticker",
        sortBy: { hour: 1 },
        output: {
          lastPrice: {
            $last: "$price",
            window: { documents: ["unbounded", "unbounded"] }
          }
        }
      }
    }
  ]).toArray();
  console.log('Window result:', JSON.stringify(windowResult, null, 2));

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():

    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    db = client.test

    # Expression usage: last element of a filtered array
    collection = db.collection
    collection.insert_many([
        {
            "_id": 1,
            "name": "John",
            "scores": [
                { "subject": "math", "score": 82 },
```

```
                { "subject": "english", "score": 85 },
                { "subject": "science", "score": 90 }
            ]
        },
        {
            "_id": 2,
            "name": "Jane",
            "scores": [
                { "subject": "math", "score": 92 },
                { "subject": "english", "score": 88 },
                { "subject": "science", "score": 87 }
            ]
        },
        {
            "_id": 3,
            "name": "Bob",
            "scores": [
                { "subject": "math", "score": 75 },
                { "subject": "english", "score": 80 },
                { "subject": "science", "score": 85 }
            ]
        }
    ])
    expression_pipeline = [
        { "$match": { "name": "John" } },
        {
            "$project": {
                "name": 1,
                "lastScienceScore": {
                    "$last": {
                        "$filter": {
                            "input": "$scores",
                            "as": "score",
                            "cond": { "$eq": ["$$score.subject", "science"] }
                        }
                    }
                }
            }
        }
    ]
    expression_result = list(collection.aggregate(expression_pipeline))
    print('Expression result:', expression_result)

    # Window operator usage: last value within each partition
```

```
    stock_prices = db.stockPrices
    stock_prices.insert_many([
        { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50 },
        { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45 },
        { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60 },
        { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40 },
        { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30 },
        { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35 },
        { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25 }
    ])
    window_pipeline = [
        {
            "$setWindowFields": {
                "partitionBy": "$ticker",
                "sortBy": { "hour": 1 },
                "output": {
                    "lastPrice": {
                        "$last": "$price",
                        "window": { "documents": ["unbounded", "unbounded"] }
                    }
                }
            }
        }
    ]
    window_result = list(stock_prices.aggregate(window_pipeline))
    print('Window result:', window_result)

    client.close()

example()
```

$lastN

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 `$lastN` 연산자는 마지막 N 요소를 반환합니다. `$group` 단계에서 누적기로 사용할 경우 각 그룹의 마지막 N 값의 배열을 반환합니다. 배열 표현식 연산자로 사용하면 배열의 마지막 N 요소를 반환합니다.

파라미터

- `input`: 값을 반환할 필드 또는 배열로 확인되는 표현식입니다.

- n: 반환할 값의 수를 지정하는 양의 정수입니다. $group 스테이지에서 누적기로 사용할 경우 그룹 _id 필드를 기반으로 양의 정수로 확인되는 한 n도 표현식이 될 수 있습니다.

예제(MongoDB 쉘)

다음 예제에서는 집계 중에 $lastN 누적기를 사용하여 각 항목의 마지막 두 수량을 검색하는 방법을 보여줍니다.

> **Note**
>
> $lastN는 문서가 $group 단계에 도달하는 순서대로 값을 선택합니다. 특정 순서(예: 날짜 또는 점수)에 대한 마지막 N 값을 반환하려면 앞에 $sort 스테이지를 추가합니다$group.

샘플 문서 생성

```
db.sales.insertMany([
  { item: "abc", quantity: 10, date: ISODate("2023-01-01") },
  { item: "abc", quantity: 5, date: ISODate("2023-01-02") },
  { item: "abc", quantity: 8, date: ISODate("2023-01-03") },
  { item: "xyz", quantity: 15, date: ISODate("2023-01-01") },
  { item: "xyz", quantity: 7, date: ISODate("2023-01-02") },
  { item: "xyz", quantity: 3, date: ISODate("2023-01-03") }
]);
```

쿼리 예제

```
db.sales.aggregate([
  { $group: { _id: "$item", lastTwoQuantities: { $lastN: { input: "$quantity", n:
 2 } } } }
]);
```

출력

```
[
  { "_id": "abc", "lastTwoQuantities": [5, 8] },
  { "_id": "xyz", "lastTwoQuantities": [7, 3] }
]
```

표현식 사용 예제(MongoDB Shell)

연$lastN산자를 $project 스테이지 내의 표현식으로 사용하여 배열 필드의 마지막 N 요소를 반환할 수도 있습니다.

샘플 문서 생성

```
db.inventory.insertMany([
  { _id: 1, item: "abc", tags: ["red", "green", "blue", "yellow", "purple"] },
  { _id: 2, item: "xyz", tags: ["alpha", "beta", "gamma"] }
]);
```

쿼리 예제

```
db.inventory.aggregate([
  { $project: {
      lastThreeTags: { $lastN: { input: "$tags", n: 3 } }
    }}
]);
```

출력

```
[
  { "_id": 1, "lastThreeTags": ["blue", "yellow", "purple"] },
  { "_id": 2, "lastThreeTags": ["alpha", "beta", "gamma"] }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$lastN산자는 $setWindowFields 스테이지에서 창 연산자로도 사용할 수 있습니다. 이 컨텍스트에서는 각 창의 문서에 대한 마지막 n 값까지 배열을 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

> **Note**
>
> 에서 창 연산자로 사용할 경우 $setWindowFields$lastN는 100MB의 중간 데이터로 제한됩니다. 이 제한을 초과하는 작업은 오류를 반환합니다.

샘플 문서 생성

```
db.stockPrices.insertMany([
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

쿼리 예제

다음 예제에서는를 기준으로 문서를 분할하고ticker,를 기준으로 각 파티션을 정렬하고hour, 파티션 시작부터 현재 문서까지 가장 최근 가격 2개를 반환합니다.

```
db.stockPrices.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        lastTwoSoFar: {
          $lastN: { input: "$price", n: 2 },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "lastTwoSoFar": [50] },
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "lastTwoSoFar": [50, 45] },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "lastTwoSoFar": [45, 60] },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "lastTwoSoFar": [60, 40] },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "lastTwoSoFar": [30] },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "lastTwoSoFar": [30, 35] },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "lastTwoSoFar": [35, 25] }
```

```
]
```

각 문서는 창 내의 마지막 두 price 값lastTwoSoFar인 로 보강됩니다(hour순서대로).

코드 예제

$lastN 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누적기 사용량(의 $group), 표현식 사용량(의 $project) 및 창 연산자 사용량(의)을 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');

    // Accumulator usage: last N values per group
    const sales = db.collection('sales');
    await sales.insertMany([
      { item: "abc", quantity: 10, date: new Date("2023-01-01") },
      { item: "abc", quantity: 5, date: new Date("2023-01-02") },
      { item: "abc", quantity: 8, date: new Date("2023-01-03") },
      { item: "xyz", quantity: 15, date: new Date("2023-01-01") },
      { item: "xyz", quantity: 7, date: new Date("2023-01-02") },
      { item: "xyz", quantity: 3, date: new Date("2023-01-03") }
    ]);
    const accumulatorResult = await sales.aggregate([
      { $group: { _id: "$item", lastTwoQuantities: { $lastN: { input: "$quantity",
 n: 2 } } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Expression usage: last N elements of an array field
    const inventory = db.collection('inventory');
    await inventory.insertMany([
      { _id: 1, item: "abc", tags: ["red", "green", "blue", "yellow", "purple"] },
```

```
      { _id: 2, item: "xyz", tags: ["alpha", "beta", "gamma"] }
    ]);
    const expressionResult = await inventory.aggregate([
      { $project: { lastThreeTags: { $lastN: { input: "$tags", n: 3 } } } }
    ]).toArray();
    console.log('Expression result:', expressionResult);

    // Window operator usage: last N values over a window in each partition
    const stockPrices = db.collection('stockPrices');
    await stockPrices.insertMany([
      { _id: 1, ticker: "ABC", hour: 1, price: 50 },
      { _id: 2, ticker: "ABC", hour: 2, price: 45 },
      { _id: 3, ticker: "ABC", hour: 3, price: 60 },
      { _id: 4, ticker: "ABC", hour: 4, price: 40 },
      { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
      { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
      { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
    ]);
    const windowResult = await stockPrices.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$ticker",
          sortBy: { hour: 1 },
          output: {
            lastTwoSoFar: {
              $lastN: { input: "$price", n: 2 },
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from datetime import datetime

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']

        # Accumulator usage: last N values per group
        sales = db['sales']
        sales.insert_many([
            { 'item': 'abc', 'quantity': 10, 'date': datetime(2023, 1, 1) },
            { 'item': 'abc', 'quantity': 5, 'date': datetime(2023, 1, 2) },
            { 'item': 'abc', 'quantity': 8, 'date': datetime(2023, 1, 3) },
            { 'item': 'xyz', 'quantity': 15, 'date': datetime(2023, 1, 1) },
            { 'item': 'xyz', 'quantity': 7, 'date': datetime(2023, 1, 2) },
            { 'item': 'xyz', 'quantity': 3, 'date': datetime(2023, 1, 3) }
        ])
        accumulator_result = list(sales.aggregate([
            { '$group': { '_id': '$item', 'lastTwoQuantities': { '$lastN':
 { 'input': '$quantity', 'n': 2 } } } }
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: last N elements of an array field
        inventory = db['inventory']
        inventory.insert_many([
            { '_id': 1, 'item': 'abc', 'tags': ['red', 'green', 'blue', 'yellow',
 'purple'] },
            { '_id': 2, 'item': 'xyz', 'tags': ['alpha', 'beta', 'gamma'] }
        ])
        expression_result = list(inventory.aggregate([
            { '$project': { 'lastThreeTags': { '$lastN': { 'input': '$tags', 'n':
 3 } } } }
        ]))
        print('Expression result:', expression_result)

        # Window operator usage: last N values over a window in each partition
        stock_prices = db['stockPrices']
```

```
        stock_prices.insert_many([
            { '_id': 1, 'ticker': 'ABC', 'hour': 1, 'price': 50 },
            { '_id': 2, 'ticker': 'ABC', 'hour': 2, 'price': 45 },
            { '_id': 3, 'ticker': 'ABC', 'hour': 3, 'price': 60 },
            { '_id': 4, 'ticker': 'ABC', 'hour': 4, 'price': 40 },
            { '_id': 5, 'ticker': 'XYZ', 'hour': 1, 'price': 30 },
            { '_id': 6, 'ticker': 'XYZ', 'hour': 2, 'price': 35 },
            { '_id': 7, 'ticker': 'XYZ', 'hour': 3, 'price': 25 }
        ])
        window_result = list(stock_prices.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$ticker',
                    'sortBy': { 'hour': 1 },
                    'output': {
                        'lastTwoSoFar': {
                            '$lastN': { 'input': '$price', 'n': 2 },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()

example()
```

$let

Amazon DocumentDB의 `$let` 연산자는 변수를 값에 바인딩하고 표현식에서 해당 변수를 사용하는 데 사용됩니다. 이를 통해 집계 파이프라인의 동일한 단계 내에서 후속 표현식에 사용할 수 있는 로컬 변수를 정의할 수 있습니다.

파라미터

- `vars`: 표현식에 사용할 변수를 정의하는 객체입니다.
- `in`: vars 파라미터에 정의된 변수가 사용되는 표현식입니다.

예제(MongoDB 쉘)

이 예제에서는 $let 연산자를 사용하여 직사각형의 면적을 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.shapes.insertMany([
  { name: "Rectangle 1", length: 5, width: 3 },
  { name: "Rectangle 2", length: 7, width: 4 },
  { name: "Rectangle 3", length: 6, width: 2 }
]);
```

쿼리 예제

```
db.shapes.aggregate([
  {
    $project: {
      name: 1,
      area: {
        $let: {
          vars: {
            length: "$length",
            width: "$width"
          },
          in: {
            $multiply: ["$$length", "$$width"]
          }
        }
      }
    }
  }
])
```

출력

```
[
  {
    "_id": ObjectId("6161e5b1a3eba3c7f2960d03"),
    "name": "Rectangle 1",
    "area": 15
  },
  {
```

```
    "_id": ObjectId("6161e5b1a3eba3c7f2960d04"),
    "name": "Rectangle 2",
    "area": 28
  },
  {
    "_id": ObjectId("6161e5b1a3eba3c7f2960d05"),
    "name": "Rectangle 3",
    "area": 12
  }
]
```

코드 예제

`$let` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function calculateRectangleAreas() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('shapes');

  const result = await shapes.aggregate([
    {
      $project: {
        name: 1,
        area: {
          $let: {
            vars: {
              length: '$length',
              width: '$width'
            },
            in: {
              $multiply: ['$$length', '$$width']
            }
          }
        }
      }
    }
  ]).toArray();
```

```
  console.log(result);
  client.close();
}

calculateRectangleAreas();
```

Python

```
from pymongo import MongoClient

def calculate_rectangle_areas():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.shapes

    result = list(shapes.aggregate([
        {
            '$project': {
                'name': 1,
                'area': {
                    '$let': {
                        'vars': {
                            'length': '$length',
                            'width': '$width'
                        },
                        'in': {
                            '$multiply': ['$$length', '$$width']
                        }
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

calculate_rectangle_areas()
```

$limit

Amazon DocumentDB의 `$limit` 연산자는 쿼리에서 반환되는 문서 수를 제한하는 데 사용됩니다. MongoDB `$limit` 연산자와 비슷하지만 Amazon DocumentDB와 함께 사용할 때 몇 가지 구체적인 고려 사항이 있습니다.

Amazon DocumentDB에서 `$limit` 연산자는 일치하는 전체 문서의 하위 집합을 검색하려는 페이지 매김에 유용합니다. 이를 통해 각 응답에서 반환되는 문서 수를 제어하여 성능을 개선하고 네트워크를 통해 전송되는 데이터의 양을 줄일 수 있습니다.

파라미터

- `limit`: 반환할 최대 문서 수입니다. 음수가 아닌 정수 값이어야 합니다.

예제(MongoDB 쉘)

다음 예제에서는 `$limit` 연산자를 사용하여 지정된 필터와 일치하는 최대 하나의 문서를 반환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.test.insertMany([
  { "_id": 1, "star_rating": 4, "comments": "apple is red" },
  { "_id": 2, "star_rating": 5, "comments": "comfortable couch" },
  { "_id": 3, "star_rating": 3, "comments": "apples, oranges - healthy fruit" },
  { "_id": 4, "star_rating": 5, "comments": "this is a great couch" },
  { "_id": 5, "star_rating": 5, "comments": "interesting couch" }
]);
```

쿼리 예제

```
db.test.createIndex({ comments: "text" });

db.test.find({
  $and: [
    { star_rating: 5 },
    { $text: { $search: "couch" } }
  ]
}).limit(1);
```

출력

```
[ { _id: 2, star_rating: 5, comments: 'comfortable couch' } ]
```

코드 예제

`$limit` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function limitExample() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
      await client.connect();

      const db = client.db('test');
      const collection = db.collection('test');

      await collection.createIndex({ comments: 'text' });

      const query = {
        $and: [
          { star_rating: 5 },
          { $text: { $search: "couch" } }
        ]
      };

      const result = await collection.find(query).limit(1).toArray();

      console.log(result);

    } catch (err) {
        console.error("Error:", err);
    } finally {
        await client.close();
    }

}
```

```
limitExample();
```

Python

```
from pymongo import MongoClient

def limit_example():
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client['test']
        collection = db['test']

        collection.create_index([('comments', 'text')])

        query = {
            '$and': [
                {'star_rating': 5},
                {'$text': {'$search': 'couch'}}
            ]
        }

        result = collection.find(query).limit(1)

        for doc in result:
            print(doc)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        client.close()

limit_example()
```

$listSearchIndexes

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $listSearchIndexes 집계 단계는 컬렉션의 기존 검색 인덱스에 대한 정보를 반환합니다. 집계 파이프라인의 첫 번째 단계여야 합니다.

파라미터

- name: (선택 사항) 정보를 반환할 검색 인덱스의 이름입니다. 생략하면 컬렉션의 모든 검색 인덱스가 반환됩니다.

출력 필드

반환된 각 문서에는 다음 필드가 포함됩니다.

- name: 검색 인덱스의 이름입니다.
- status: 인덱스의 빌드 상태입니다. READY (빌드 및 유효), BUILDING (동시 빌드 진행 중) 또는 FAILED (인덱스 빌드가 성공적으로 완료되지 않음) 중 하나.
- queryable: 인덱스를 현재 쿼리에 사용할 수 있는지 여부를 나타내는 부울입니다. 이는 false 숨겨진 인덱스(로 유지READY) 및 FAILED 인덱스에 대한 것입니다. 숨겨진 인덱스는가 READY 이지만 status가 인 경우queryable입니다false.
- latestDefinitionVersion: version (인덱스 형식 버전) 및 createdAt (인덱스가 생성된 시간)을 포함하는 문서입니다.

구문

```
db.collection.aggregate([
  { $listSearchIndexes: {} }
])

// Or filter by name:
db.collection.aggregate([
  { $listSearchIndexes: { name: "mySearchIndex" } }
])
```

예제(MongoDB 쉘)

다음 예제에서는 $listSearchIndexes 스테이지를 사용하여 컬렉션의 모든 검색 인덱스를 나열하는 방법을 보여줍니다.

쿼리 예제

```
db.movies.aggregate([
  { $listSearchIndexes: {} }
]);
```

출력

```
[
  {
    "name": "default",
    "status": "READY",
    "queryable": true,
    "latestDefinitionVersion": {
      "version": 2,
      "createdAt": ISODate("2026-05-11T21:12:57.974Z")
    }
  }
]
```

숨겨진 인덱스는 유지READY되지만 queryable: false숨겨진 쿼리를 처리하는 데 사용할 수 없으므로는 보고합니다.

```
[
  {
    "name": "myHiddenIndex",
    "status": "READY",
    "queryable": false,
    "latestDefinitionVersion": {
      "version": 2,
      "createdAt": ISODate("2026-05-11T21:12:57.974Z")
    }
  }
]
```

특정 인덱스 이름을 기준으로 필터링하려면:

```
db.movies.aggregate([
  { $listSearchIndexes: { name: "default" } }
]);
```

코드 예제

`$listSearchIndexes` 스테이지 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('movies');

    // List all search indexes
    const allIndexes = await collection.aggregate([
      { $listSearchIndexes: {} }
    ]).toArray();
    console.log('All search indexes:', allIndexes);

    // List a specific search index by name
    const namedIndex = await collection.aggregate([
      { $listSearchIndexes: { name: "default" } }
    ]).toArray();
    console.log('Named index:', namedIndex);

  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']
        collection = db['movies']

        # List all search indexes
        all_indexes = list(collection.aggregate([
            { '$listSearchIndexes': {} }
        ]))
        print('All search indexes:', all_indexes)

        # List a specific search index by name
        named_index = list(collection.aggregate([
            { '$listSearchIndexes': { 'name': 'default' } }
        ]))
        print('Named index:', named_index)

    finally:
        client.close()

example()
```

$literal

Amazon DocumentDB의 `$literal` 연산자는 집계 파이프라인 단계 내의 리터럴 값을 나타내는 데 사용됩니다. 필드 참조 또는 표현식으로 해석하지 않고 숫자, 문자열 또는 부울과 같은 특정 값을 포함할 수 있습니다.

이 연산자는 동적 쿼리 필터를 빌드하거나 계산을 수행할 때와 같이 보다 복잡한 집계 파이프라인의 일부로 리터럴 값을 포함해야 하는 경우에 특히 유용합니다.

파라미터

없음

예제(MongoDB 쉘)

다음 예제에서는 $literal 연산자를 사용하여 집계 파이프라인에 리터럴 값을 포함하는 방법을 보여 줍니다. 연$literal산자는 값 18을 $gt 표현식에 리터럴 값으로 포함하는 데 사용됩니다. 이렇게 하면 집계 파이프라인이 연령 필드를 리터럴 값 18과 비교하여 사람이 성인인지 확인할 수 있습니다.

샘플 문서 생성

```
db.collection.insertMany([
  { "name": "John Doe", "age": 30, "city": "New York" },
  { "name": "Jane Doe", "age": 25, "city": "Los Angeles" },
  { "name": "Bob Smith", "age": 35, "city": "Chicago" }
]);
```

쿼리 예제

```
db.collection.aggregate([
  {
    $project: {
      name: 1,
      age: 1,
      city: 1,
      isAdult: { $gt: ["$age", { $literal: 18 }] }
    }
  }
]);
```

출력

```
[
  {
    "_id": ObjectId("601234567890abcdef012345"),
    "name": "John Doe",
    "age": 30,
    "city": "New York",
    "isAdult": true
  },
  {
    "_id": ObjectId("601234567890abcdef012346"),
    "name": "Jane Doe",
    "age": 25,
```

```
    "city": "Los Angeles",
    "isAdult": true
  },
  {
    "_id": ObjectId("601234567890abcdef012347"),
    "name": "Bob Smith",
    "age": 35,
    "city": "Chicago",
    "isAdult": true
  }
]
```

코드 예제

`$literal` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
        age: 1,
        city: 1,
        isAdult: { $gt: ["$age", { $literal: 18 }] }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.collection

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'age': 1,
                'city': 1,
                'isAdult': { '$gt': ["$age", { '$literal': 18 }] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$ln

버전 4.0의 새 버전입니다.

Amazon DocumentDB의 `$ln` 연산자는 지정된 숫자의 자연 로그(기본 e)를 계산합니다. 숫자의 로그를 기본 e로 반환합니다.

파라미터

- `expression`: 자연 로그를 계산할 숫자입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$log` 연산자를 사용하여 숫자의 자연 로그를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.numbers.insertMany([
  { _id: 1, value: 10 },
  { _id: 2, value: 100 },
  { _id: 3, value: 1000 }
]);
```

쿼리 예제

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    naturalLog: { $ln: "$value" }
  }}
]);
```

출력

```
[
  { "_id" : 1, "naturalLog" : 2.302585092994046 },
  { "_id" : 2, "naturalLog" : 4.605170185988092 },
  { "_id" : 3, "naturalLog" : 6.907755278982137 }
]
```

코드 예제

$ln 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  const db = client.db('test');
  const collection = db.collection('numbers');
```

```
  const result = await collection.aggregate([
    { $project: {
      _id: 1,
      naturalLog: { $ln: "$value" }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():

    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    db = client.test
    collection = db.numbers

    result = list(collection.aggregate([
        { "$project": {
            "_id": 1,
            "naturalLog": { "$ln": "$value" }
        }}
    ]))

    print(result)
    client.close()

example()
```

$log

버전 4.0의 새 버전입니다.

Amazon DocumentDB의 $log 연산자는 숫자의 자연 로그를 계산합니다. 지정된 숫자의 base-e 로그를 반환합니다.

파라미터

- expression: 자연 로그를 계산할 숫자입니다.
- base: 로그를 계산하기 위한 기본값입니다.

예제(MongoDB 쉘)

다음 예제에서는 $log 연산자를 사용하여 숫자의 자연 로그를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.numbers.insertMany([
  { _id: 1, value: 10 },
  { _id: 2, value: 100 },
  { _id: 3, value: 1000 }
]);
```

쿼리 예제

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    naturalLog: { $log: ["$value", 10] }
  }}
]);
```

출력

```
[
  { "_id" : 1, "naturalLog" : 1 },
  { "_id" : 2, "naturalLog" : 2 },
  { "_id" : 3, "naturalLog" : 2.9999999999999996 }
]
```

코드 예제

$log 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $project: {
      _id: 1,
      naturalLog: { $log: ["$value", 10] }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    db = client.test
    collection = db.numbers

    result = list(collection.aggregate([
        { "$project": {
            "_id": 1,
            "naturalLog": { "$log": ["$value", 10] }
        }}
    ]))
```

```
    print(result)
    client.close()

example()
```

$log10

버전 4.0의 새 버전입니다.

Amazon DocumentDB의 `$log10` 연산자는 숫자의 base-10 로그를 계산하는 데 사용됩니다. 집계 파이프라인 내의 숫자 필드에 대한 로그 계산을 수행하는 데 유용합니다.

파라미터

- `expression`: base-10 로그를 계산할 숫자 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$log10` 연산자를 사용하여 숫자 필드의 base-10 로그를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.numbers.insertMany([
  { _id: 1, value: 1 },
  { _id: 2, value: 10 },
  { _id: 3, value: 100 },
  { _id: 4, value: 1000 }
]);
```

쿼리 예제

```
db.numbers.aggregate([
  {
    $project: {
      _id: 1,
      log10Value: { $log10: "$value" }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "log10Value": 0 },
  { "_id": 2, "log10Value": 1 },
  { "_id": 3, "log10Value": 2 },
  { "_id": 4, "log10Value": 3 }
]
```

코드 예제

$log10 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  let client;

  try {
    client = await MongoClient.connect('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

    const db = client.db('test');
    const collection = db.collection('numbers');

    const result = await collection.aggregate([
      {
        $project: {
          _id: 1,
          log10Value: { $log10: "$value" }
        }
      }
    ]).toArray();

    console.log(result);
  } catch (error) {
    console.error("An error occurred:", error);
  } finally {
    if (client) {
      await client.close();
```

```
    }
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client.test
        collection = db.numbers

        result = list(collection.aggregate([
            {
                '$project': {
                    '_id': 1,
                    'log10Value': { '$log10': '$value' }
                }
            }
        ]))

        print(result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$lt

$lt 집계 연산자는 두 값을 비교하고 첫 번째 값이 두 번째 값보다 작true으면를 반환하고, 그렇지 않으면를 반환합니다false.

파라미터

- expression1: 비교할 첫 번째 값입니다.
- expression2: 비교할 두 번째 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 $lt 연산자를 사용하여 재고가 적은 항목을 식별하는 방법을 보여줍니다.

샘플 문서 생성

```
db.warehouse.insertMany([
  { _id: 1, item: "Bolts", stock: 5 },
  { _id: 2, item: "Nuts", stock: 25 },
  { _id: 3, item: "Screws", stock: 8 }
]);
```

쿼리 예제

```
db.warehouse.aggregate([
  {
    $project: {
      item: 1,
      stock: 1,
      lowStock: { $lt: ["$stock", 10] }
    }
  }
]);
```

출력

```
[
  { _id: 1, item: 'Bolts', stock: 5, lowStock: true },
  { _id: 2, item: 'Nuts', stock: 25, lowStock: false },
  { _id: 3, item: 'Screws', stock: 8, lowStock: true }
]
```

코드 예제

`$lt` 집계 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('warehouse');

  const result = await collection.aggregate([
    {
      $project: {
        item: 1,
        stock: 1,
        lowStock: { $lt: ["$stock", 10] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['warehouse']

    result = list(collection.aggregate([
```

```
        {
            '$project': {
                'item': 1,
                'stock': 1,
                'lowStock': { '$lt': ['$stock', 10] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$lte

$lte 집계 연산자는 두 값을 비교하고 첫 번째 값이 두 번째 값보다 작거나 같true으면를 반환하고, 그렇지 않으면를 반환합니다false.

파라미터

- expression1: 비교할 첫 번째 값입니다.
- expression2: 비교할 두 번째 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 $lte 연산자를 사용하여 예산 친화적인 항목을 식별하는 방법을 보여줍니다.

샘플 문서 생성

```
db.menu.insertMany([
  { _id: 1, dish: "Salad", price: 8 },
  { _id: 2, dish: "Pasta", price: 12 },
  { _id: 3, dish: "Soup", price: 6 }
]);
```

쿼리 예제

```
db.menu.aggregate([
  {
```

```
    $project: {
      dish: 1,
      price: 1,
      affordable: { $lte: ["$price", 10] }
    }
  }
]);
```

출력

```
[
  { _id: 1, dish: 'Salad', price: 8, affordable: true },
  { _id: 2, dish: 'Pasta', price: 12, affordable: false },
  { _id: 3, dish: 'Soup', price: 6, affordable: true }
]
```

코드 예제

`$lte` 집계 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('menu');

  const result = await collection.aggregate([
    {
      $project: {
        dish: 1,
        price: 1,
        affordable: { $lte: ["$price", 10] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
```

```
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['menu']

    result = list(collection.aggregate([
        {
            '$project': {
                'dish': 1,
                'price': 1,
                'affordable': { '$lte': ['$price', 10] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$lookup

Amazon DocumentDB의 $lookup 집계 단계를 사용하면 두 컬렉션 간에 왼쪽 외부 조인을 수행할 수 있습니다. 이 작업을 사용하면 일치하는 필드 값을 기반으로 여러 컬렉션의 데이터를 결합할 수 있습니다. 관련 컬렉션의 데이터를 쿼리 결과에 통합해야 할 때 특히 유용합니다.

파라미터

- `from`: 조인을 수행할 컬렉션의 이름입니다.
- `localField`:와 일치시킬 입력 문서의 필드입니다`foreignField`.
- `foreignField`: `from` 컬렉션에 있는 문서의 필드로,와 일치합니다`localField`.

- as: from 컬렉션의 일치하는 문서가 포함된 출력 문서에 추가할 새 필드의 이름입니다.
- pipeline: localField 및가 지정된 경우 선택 사항foreignField입니다. from 컬렉션에서 실행할 집계 파이프라인입니다. 파이프라인은 from 컬렉션에서 문서를 반환합니다. 모든 문서를 반환하려면 빈 파이프라인을 지정합니다[]. 파이프라인에는 $out 또는 $merge 단계가 포함될 수 없습니다. 조인 조건 및 상관관계가 있는 하위 쿼리(MongoDB Shell) 섹션을 참조하세요.
- let: 선택 사항입니다. pipeline 단계에서 사용할 변수를 정의하는 문서입니다. 변수 표현식을 사용하여에 조인된 입력 문서의 필드에 액세스합니다pipeline. 파이프라인 단계의 변수를 참조하려면 "$$<variable>" 구문을 사용합니다. 조인 조건 및 상관관계가 있는 하위 쿼리(MongoDB Shell) 섹션을 참조하세요.

예제(MongoDB 쉘)

다음 예제에서는 orders 컬렉션의 데이터를 customers 컬렉션에 조인하는 간단한 $lookup 작업을 보여줍니다.

샘플 문서 생성

```
db.customers.insertMany([
  { _id: 1, name: "Alice" },
  { _id: 2, name: "Bob" },
  { _id: 3, name: "Charlie" }
]);

db.orders.insertMany([
  { _id: 1, customer_id: 1, total: 50 },
  { _id: 2, customer_id: 1, total: 100 },
  { _id: 3, customer_id: 2, total: 75 }
]);
```

쿼리 예제

```
db.customers.aggregate([
  {
    $lookup: {
      from: "orders",
      localField: "_id",
      foreignField: "customer_id",
      as: "orders"
    }
  }
```

```
]);
```

출력

```
[
  {
    _id: 1,
    name: 'Alice',
    orders: [
      { _id: 2, customer_id: 1, total: 100 },
      { _id: 1, customer_id: 1, total: 50 }
    ]
  },
  { _id: 3, name: 'Charlie', orders: [] },
  {
    _id: 2,
    name: 'Bob',
    orders: [ { _id: 3, customer_id: 2, total: 75 } ]
  }
]
```

조인 조건 및 상관관계가 있는 하위 쿼리(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

Amazon DocumentDB$lookup에서는 단일 등식 일치 외에도 조인된 컬렉션에서 집계 파이프라인을 실행할 수 있습니다. 이는 모든 입력 문서에 대해 동일한 하위 쿼리를 실행하는 상관관계가 없는 하위 쿼리와 각 입력 문서의 필드에서 정의된 변수를 사용하여 해당 문서와 상관관계가 있는 결과를 반환하는 상관관계가 있는 하위 쿼리를 지원합니다.

구문

상관관계가 있는 하위 쿼리를 실행하려면를 사용하여 입력 문서의 필드에서 변수를 정의하고 `pipeline` 단계에서 `let`참조합니다.

```
{
  $lookup: {
    from: <collection to join>,
    let: { <var_1>: <expression>, ..., <var_n>: <expression> },
    pipeline: [ <pipeline to run on the joined collection> ],
    as: <output array field>
  }
```

```
}
```

파라미터

- `let`: 입력 문서의 필드에서 변수를 정의하는 문서입니다. 파이프라인 단계의 변수를 참조하려면 "$$<variable>" 구문을 사용합니다. 변수는에 중첩된 추가 단계를 `pipeline`포함하여의 `$lookup` 단계에서 액세스할 수 있습니다`pipeline`. `$match` 단계에서는 연`$expr`산자를 사용하여 변수에 액세스해야 합니다. 상관 관계가 없는 하위 쿼리를 실행`let`하려면 생략합니다.
- `pipeline`: `from` 컬렉션에서 실행할 집계 파이프라인입니다. 파이프라인은 입력 문서의 필드에 직접 액세스할 수 없습니다. 로 변수를 정의하고 대신 변수를 `let` 참조합니다. 모든 문서를 반환하려면 빈 파이프라인을 지정합니다`[]`.

`$match` 단계 내의 `$expr` 연산자에 배치된 `$eq`, `$gt``$gte`, `$lt`, 및 `$lte` 비교 연산자는 조인된 `from` 컬렉션의 필드를 `let` 변수 또는 상수와 비교할 때 컬렉션의 인덱스를 사용할 수 있습니다.

샘플 문서 생성

```
db.customers.insertMany([
  { _id: 1, name: "Alice", min_total: 60 },
  { _id: 2, name: "Bob", min_total: 50 },
  { _id: 3, name: "Charlie", min_total: 20 }
]);

db.orders.insertMany([
  { _id: 1, customer_id: 1, total: 50 },
  { _id: 2, customer_id: 1, total: 100 },
  { _id: 3, customer_id: 2, total: 75 }
]);
```

쿼리 예제

다음 예시에서는 각 고객을 주문과 조인하고 `total`가 고객의 자체 `min_total` 값보다 크거나 같은 주문만 유지합니다. 조인 조건과 고객별 필터는 내의 두 참조 `let` 변수를 모두 필터링합니다`$expr`.

```
db.customers.aggregate([
  {
    $lookup: {
      from: "orders",
      let: { customer_id: "$_id", min_total: "$min_total" },
      pipeline: [
```

```
        {
          $match: {
            $expr: {
              $and: [
                { $eq: [ "$customer_id", "$$customer_id" ] },
                { $gte: [ "$total", "$$min_total" ] }
              ]
            }
          }
        },
        { $project: { _id: 0, total: 1 } }
      ],
      as: "qualifying_orders"
    }
  }
]);
```

출력

```
[
  { _id: 1, name: 'Alice', min_total: 60, qualifying_orders: [ { total: 100 } ] },
  { _id: 2, name: 'Bob', min_total: 50, qualifying_orders: [ { total: 75 } ] },
  { _id: 3, name: 'Charlie', min_total: 20, qualifying_orders: [] }
]
```

각 고객에 대해 하위 쿼리는 해당 고객의 `_id` 및 `min_total` 값으로 평가되므로 각 문서는 상관관계가 있는 자체 결과를 수신합니다.

간결한 구문을 사용한 상관관계가 있는 하위 쿼리(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

Amazon DocumentDB는 간결한 상관관계가 있는 하위 쿼리 구문도 지원합니다.이 구문은 `localField` 및의 등식 일치를 `foreignField`와 결합합니다`pipeline`. 간결한 구문을 사용하면 `$match` 단계의 `$expr` 연산자 내에서 등식 일치를 표현할 필요가 없습니다.

구문

```
{
  $lookup: {
    from: <collection to join>,
    localField: <field from the input documents>,
```

```
    foreignField: <field from the documents of the "from" collection>,
    let: { <var_1>: <expression>, ..., <var_n>: <expression> },
    pipeline: [ <pipeline to run> ],
    as: <output array field>
  }
}
```

쿼리 예제

다음 예제에서는 상관관계가 있는 이전 하위 쿼리 예제와 동일한 결과를 반환합니다. `_id`와 간의 등식 일치`customer_id`는 `$expr` 조건 `foreignField` 대신 `localField` 및 로 표현됩니다.

```
db.customers.aggregate([
  {
    $lookup: {
      from: "orders",
      localField: "_id",
      foreignField: "customer_id",
      let: { min_total: "$min_total" },
      pipeline: [
        { $match: { $expr: { $gte: [ "$total", "$$min_total" ] } } },
        { $project: { _id: 0, total: 1 } }
      ],
      as: "qualifying_orders"
    }
  }
]);
```

출력

```
[
  { _id: 1, name: 'Alice', min_total: 60, qualifying_orders: [ { total: 100 } ] },
  { _id: 2, name: 'Bob', min_total: 50, qualifying_orders: [ { total: 75 } ] },
  { _id: 3, name: 'Charlie', min_total: 20, qualifying_orders: [] }
]
```

상관관계가 있는 하위 쿼리에서 지원되는 단계

`$lookup` 파이프라인에서 `let`를 사용하는 경우 다음 단계에서 "$$<variable>" 변수를 참조할 수 있습니다.

- `$match` (`$expr` 연산자 사용)

- $project
- $addFields
- $set
- $group
- $bucket
- $bucketAuto
- $sortByCount
- $lookup(중첩)
- $graphLookup

$sort, , , , $unset, $skip $limit$count$sample$unwind, 등의 변수 표현식을 평가하지 않는 단계도 파이프라인에서 사용할 수 있지만 변수를 참조할 수는 없습니다.

, let$facet, , , , $redact, 및를와 $replaceRoot $replaceWith $geoNear $setWindowFields $fill $match 함께 사용하는 $lookup 파이프라인에서는 다음 단계가 지원되지 않습니다$text.

제한 사항

- 에는 $out 또는 $merge 단계가 포함될 수 pipeline 없습니다.

코드 예제

$lookup 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  await client.connect();

  const db = client.db('test');
```

```
  const result = await db.collection('customers').aggregate([
    {
      $lookup: {
        from: 'orders',
        localField: '_id',
        foreignField: 'customer_id',
        as: 'orders'
      }
    }
  ]).toArray();

  console.log(JSON.stringify(result, null, 2));
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    db = client.test

    collection = db.customers

    pipeline = [
        {
            "$lookup": {
                "from": "orders",
                "localField": "_id",
                "foreignField": "customer_id",
                "as": "orders"
            }
        }
    ]

    result = collection.aggregate(pipeline)
```

```
    for doc in result:
        print(doc)

    client.close()

example()
```

상관관계가 있는 하위 쿼리 코드 예제

상관관계가 있는 하위 쿼리와 함께 $lookup 스테이지를 사용하기 위한 코드 예제를 보려면(let 및 사용pipeline) 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  const db = client.db('test');

  const customers = db.collection('customers');
  await customers.insertMany([
    { _id: 1, name: "Alice", min_total: 60 },
    { _id: 2, name: "Bob", min_total: 50 },
    { _id: 3, name: "Charlie", min_total: 20 }
  ]);
  const orders = db.collection('orders');
  await orders.insertMany([
    { _id: 1, customer_id: 1, total: 50 },
    { _id: 2, customer_id: 1, total: 100 },
    { _id: 3, customer_id: 2, total: 75 }
  ]);

  // Correlated subquery: filter each customer's orders with a per-customer
 threshold
  const result = await customers.aggregate([
    {
      $lookup: {
        from: 'orders',
```

```
        localField: '_id',
        foreignField: 'customer_id',
        let: { min_total: '$min_total' },
        pipeline: [
          { $match: { $expr: { $gte: [ '$total', '$$min_total' ] } } },
          { $project: { _id: 0, total: 1 } }
        ],
        as: 'qualifying_orders'
      }
    }
  ]).toArray();

  console.log(JSON.stringify(result, null, 2));
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    db = client.test

    customers = db.customers
    customers.insert_many([
        { "_id": 1, "name": "Alice", "min_total": 60 },
        { "_id": 2, "name": "Bob", "min_total": 50 },
        { "_id": 3, "name": "Charlie", "min_total": 20 }
    ])
    orders = db.orders
    orders.insert_many([
        { "_id": 1, "customer_id": 1, "total": 50 },
        { "_id": 2, "customer_id": 1, "total": 100 },
        { "_id": 3, "customer_id": 2, "total": 75 }
    ])
```

```
    # Correlated subquery: filter each customer's orders with a per-customer
 threshold
    pipeline = [
        {
            "$lookup": {
                "from": "orders",
                "localField": "_id",
                "foreignField": "customer_id",
                "let": { "min_total": "$min_total" },
                "pipeline": [
                    { "$match": { "$expr": { "$gte": [ "$total", "$
$min_total" ] } } },
                    { "$project": { "_id": 0, "total": 1 } }
                ],
                "as": "qualifying_orders"
            }
        }
    ]

    result = customers.aggregate(pipeline)

    for doc in result:
        print(doc)

    client.close()

example()
```

$ltrim

버전 4.0의 새 버전입니다.

Elastic 클러스터에서는 지원되지 않습니다.

Amazon DocumentDB의 `$ltrim` 연산자는 문자열에서 선행 문자를 제거하는 데 사용됩니다. 기본적으로 선행 공백 문자를 제거하지만 문자 인수를 전달하여 제거할 문자 집합을 지정할 수도 있습니다.

파라미터

- `input`: 선행 공백 문자를 제거할 입력 문자열입니다.
- `chars`: (선택 사항) 특정 문자를 제거합니다.

예제(MongoDB 쉘)

다음 예제에서는를 사용하여 문자열의 시작 부분에서 지정된 문자(" *")를 제거하는 $ltrim 방법을 보여줍니다.

샘플 문서 생성

```
db.collection.insertMany([
  { name: " *John Doe", age: 30 },
  { name: "Jane Doe*", age: 25 },
  { name: "  Bob Smith  ", age: 35 }
]);
```

쿼리 예제

```
db.collection.aggregate([
  {
    $project: {
      _id: 0,
      name: {
        $ltrim: { input: "$name", chars: " *" }
      },
      age: 1
    }
  }
]);
```

출력

```
[
  { "name": "John Doe", "age": 30 },
  { "name": "Jane Doe ", "age": 25 },
  { "name": "Bob Smith  ", "age": 35 }
]
```

코드 예제

$ltrim 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();

    const db = client.db('test');
    const collection = db.collection('collection');

    const pipeline = [
      {
        $project: {
          _id: 0,
          name: {
            $ltrim: {
              input: '$name',
              chars: ' *'
            }
          },
          age: 1
        }
      }
    ];

    const result = await collection.aggregate(pipeline).toArray();

    console.dir(result, { depth: null });

  } finally {
    await client.close();
  }
}

example().catch(console.error);
```

Python

```
from pymongo import MongoClient

def example():
```

```
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client.test
        collection = db.collection

        pipeline = [
            {
                "$project": {
                    "_id": 0,
                    "name": {
                        "$ltrim": {
                            "input": "$name",
                            "chars": " *"
                        }
                    },
                    "age": 1
                }
            }
        ]

        results = collection.aggregate(pipeline)

        for doc in results:
            print(doc)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        client.close()

example()
```

$map

Amazon DocumentDB의 $map 연산자를 사용하면 배열의 각 요소에 지정된 표현식을 적용하고 변환된 요소가 있는 새 배열을 반환할 수 있습니다. 이 연산자는 배열 내에서 데이터를 조작하고 변환하는 데 특히 유용하며, 이는 애플리케이션 코드를 단순화하고 배열 처리를 데이터베이스 수준으로 푸시하여 쿼리 성능을 개선하는 데 도움이 될 수 있습니다.

파라미터

- `input`: 변환할 배열입니다.
- `as`: (선택 사항) 처리 중인 현재 요소를 나타내는 표현식의 내에서 사용할 변수의 이름입니다.
- `in`: 입력 배열의 각 요소에 적용할 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $map 연산자를 사용하여 숫자 배열을 변환하여 각 값을 두 배로 늘리는 방법을 보여 줍니다.

샘플 문서 생성

```
db.collection.insertMany([
  { _id: 1, numbers: [1, 2, 3, 4, 5] },
  { _id: 2, numbers: [10, 20, 30, 40, 50] }
])
```

쿼리 예제

```
db.collection.aggregate([
  {
    $project: {
      doubledNumbers: { $map: { input: "$numbers", as: "num", in: { $multiply: ["$
$num", 2] } } }
    }
  }
])
```

출력

```
[
  { _id: 1, doubledNumbers: [2, 4, 6, 8, 10] },
  { _id: 2, doubledNumbers: [20, 40, 60, 80, 100] }
]
```

코드 예제

$map 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.aggregate([
    {
      $project: {
        doubledNumbers: { $map: { input: "$numbers", as: "num", in: { $multiply: ["$
$num", 2] } } }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.collection

    result = list(collection.aggregate([
        {
            '$project': {
                'doubledNumbers': { '$map': { 'input': '$numbers', 'as': 'num',
 'in': { '$multiply': ['$$num', 2] } } }
            }
```

```
        }
    ]))

    print(result)
    client.close()

example()
```

$match

Amazon DocumentDB의 $match 파이프라인 단계는 입력 문서를 지정된 쿼리 기준과 일치하는 문서로만 필터링하는 데 사용됩니다. 집계 작업에서 가장 일반적으로 사용되는 파이프라인 단계 중 하나입니다. $match 단계는 다른 파이프라인 단계보다 먼저 적용되므로 후속 단계에서 처리해야 하는 문서 수를 효율적으로 줄일 수 있습니다.

파라미터

- query: 작업의 선택 기준을 표현하는 문서입니다. 쿼리 문서는 find() 메서드와 동일한 구문을 사용합니다.

예제(MongoDB 쉘)

다음 예제에서는 $match 스테이지를 사용하여 특정 필드 값을 기반으로 문서를 필터링하는 방법을 보여줍니다.

샘플 문서 생성

```
db.collection.insertMany([
  { _id: 1, name: "John", age: 25, city: "New York" },
  { _id: 2, name: "Jane", age: 30, city: "Los Angeles" },
  { _id: 3, name: "Bob", age: 35, city: "Chicago" },
  { _id: 4, name: "Alice", age: 40, city: "Miami" }
]);
```

쿼리 예제

```
db.collection.aggregate([
  { $match: { age: { $gt: 30 } } },
  { $project: { _id: 1, name: 1, city: 1 } }
```

```
]);
```

출력

```
[
  { "_id": 3, "name": "Bob", "city": "Chicago" },
  { "_id": 4, "name": "Alice", "city": "Miami" }
]
```

$match 단계는 age 필드가 30보다 큰 문서만 포함하도록 문서를 필터링합니다.

코드 예제

$match 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.aggregate([
    { $match: { age: { $gt: 30 } } },
    { $project: { _id: 1, name: 1, city: 1 } }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient
```

```
def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    db = client['test']
    collection = db['collection']

    result = list(collection.aggregate([
        { '$match': { 'age': { '$gt': 30 } } },
        { '$project': { '_id': 1, 'name': 1, 'city': 1 } }
    ]))

    print(result)
    client.close()

example()
```

$max

$max 집계 연산자는 문서 세트에서 지정된 필드의 최대값을 반환합니다. 이 연산자는 문서 세트에서 가장 높은 값을 찾는 데 유용합니다.

파라미터

- expression: 최대값을 계산하는 데 사용할 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $max 연산자를 사용하여 학생 문서 모음에서 최대 점수를 찾는 방법을 보여줍니다. $group 스테이지는 모든 문서를 그룹화하고 연$max산자는 모든 문서에서 score 필드의 최대값을 계산하는 데 사용됩니다.

샘플 문서 생성

```
db.students.insertMany([
  { name: "John", score: 85 },
  { name: "Jane", score: 92 },
  { name: "Bob", score: 78 },
  { name: "Alice", score: 90 }
])
```

쿼리 예제

```
db.students.aggregate([
  { $group: { _id: null, maxScore: { $max: "$score" } } },
  { $project: { _id: 0, maxScore: 1 } }
])
```

출력

```
[ { maxScore: 92 } ]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$max산자는 $setWindowFields 스테이지에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 문서에 대한 최대값을 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

> **Note**
>
> 에서 창 연산자로 사용할 경우 $setWindowFields$max는 100MB의 중간 데이터로 제한됩니다. 이 제한을 초과하는 작업은 오류를 반환합니다.

샘플 문서 생성

```
db.stockPrices.insertMany([
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

쿼리 예제

다음 예제에서는 문서를 로 분할하고ticker, 각 파티션을 로 정렬하고hour, 파티션 시작부터 현재 문서까지 가장 높은 가격을 반환합니다.

```
db.stockPrices.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        highestSoFar: {
          $max: "$price",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "highestSoFar": 50 },
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "highestSoFar": 50 },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "highestSoFar": 60 },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "highestSoFar": 60 },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "highestSoFar": 30 },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "highestSoFar": 35 },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "highestSoFar": 35 }
]
```

각 문서는 현재 문서를 포함하여 파티션 `price` 내에서 `highestSoFar`실행 중인 최대 인 로 보강됩니다.

코드 예제

`$max` 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누산기 사용량()과 창 연산자 사용량(`$group`)을 모두 보여줍니다. `$setWindowFields`

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');

  // Accumulator usage: maximum value across a group
  const students = db.collection('students');
  await students.insertMany([
    { name: "John", score: 85 },
    { name: "Jane", score: 92 },
    { name: "Bob", score: 78 },
    { name: "Alice", score: 90 }
  ]);
  const accumulatorResult = await students.aggregate([
    { $group: { _id: null, maxScore: { $max: "$score" } } }
  ]).toArray();
  console.log('Accumulator result:', accumulatorResult);

  // Window operator usage: running maximum within each partition
  const stockPrices = db.collection('stockPrices');
  await stockPrices.insertMany([
    { _id: 1, ticker: "ABC", hour: 1, price: 50 },
    { _id: 2, ticker: "ABC", hour: 2, price: 45 },
    { _id: 3, ticker: "ABC", hour: 3, price: 60 },
    { _id: 4, ticker: "ABC", hour: 4, price: 40 },
    { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
    { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
    { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
  ]);
  const windowResult = await stockPrices.aggregate([
    {
      $setWindowFields: {
        partitionBy: "$ticker",
        sortBy: { hour: 1 },
        output: {
          highestSoFar: {
            $max: "$price",
            window: { documents: ["unbounded", "current"] }
          }
        }
      }
    }
  ]).toArray();
  console.log('Window result:', windowResult);
```

```
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']

    # Accumulator usage: maximum value across a group
    students = db.students
    students.insert_many([
        { "name": "John", "score": 85 },
        { "name": "Jane", "score": 92 },
        { "name": "Bob", "score": 78 },
        { "name": "Alice", "score": 90 }
    ])
    accumulator_result = list(students.aggregate([
        { "$group": { "_id": None, "maxScore": { "$max": "$score" } } }
    ]))
    print('Accumulator result:', accumulator_result)

    # Window operator usage: running maximum within each partition
    stock_prices = db.stockPrices
    stock_prices.insert_many([
        { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50 },
        { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45 },
        { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60 },
        { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40 },
        { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30 },
        { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35 },
        { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25 }
    ])
    window_result = list(stock_prices.aggregate([
        {
            "$setWindowFields": {
```

```
                "partitionBy": "$ticker",
                "sortBy": { "hour": 1 },
                "output": {
                    "highestSoFar": {
                        "$max": "$price",
                        "window": { "documents": ["unbounded", "current"] }
                    }
                }
            }
        }
    ]))
    print('Window result:', window_result)

    client.close()

example()
```

$maxN

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $maxN 연산자는 N개의 가장 큰 값을 반환합니다. $group 스테이지에서 누적기로 사용할 경우 각 그룹의 N 최대값 배열을 반환합니다. 배열 표현식 연산자로 사용할 경우 배열의 N 최대 요소를 반환합니다.

파라미터

- input: 최대값을 반환할 배열 또는 필드로 확인되는 표현식입니다n.
- n: 반환할 최대 값 수를 지정하는 양의 정수로 확인되는 표현식입니다.

동작

배열 표현식 연산자 동작

- 보다 n 작은 값은 지정할 수 없습니다1.
- $maxN는 input 배열에서 찾은 null 값을 필터링합니다.
- 지정된 n가 input 배열의 요소 수보다 크거나 같으면는 input 배열의 모든 요소를 $maxN 반환합니다.
- 가 배열이 아닌 값으로 input 확인되면 집계 작업 오류가 발생합니다.

- 에 숫자 및 문자열 요소가 모두 input 포함된 경우 문자열 요소는 BSON 비교 순서에 따라 숫자 요소 앞에 정렬됩니다.

누산기 동작

- 누적기로 사용되는 경우는 상수이거나의 _id 값에 따라 달라지는 양의 정수 표현식이어야 n 합니다$group.
- $maxN는 null 및 누락 값을 필터링합니다.
- 그룹에 개 미만의 n 요소가 포함된 경우는 그룹의 모든 요소를 $maxN 반환합니다.
- $maxN는 BSON 비교 순서에 따라 입력 데이터를 비교하여 적절한 출력 유형을 결정합니다. 입력 데이터에 여러 데이터 형식이 포함된 경우 $maxN 출력 형식은 비교 순서에서 가장 높습니다.

출력 순서 지정

$maxN는 특정 정렬 순서로 값을 반환하지 않습니다. 특정 정렬 순서를 보장해야 하는 경우 $topN 대신를 사용하거나 결과를 로 래핑합니다$sortArray.

예제(MongoDB 쉘)

다음 예제에서는 $maxN 누적기를 사용하여 각 주제에 대해 가장 높은 두 점수를 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.scores.insertMany([
  { subject: "math", score: 85 },
  { subject: "math", score: 72 },
  { subject: "math", score: 93 },
  { subject: "math", score: 68 },
  { subject: "science", score: 90 },
  { subject: "science", score: 78 },
  { subject: "science", score: 65 },
  { subject: "science", score: 88 }
]);
```

쿼리 예제

```
db.scores.aggregate([
```

```
  { $group: { _id: "$subject", highestTwo: { $maxN: { input: "$score", n: 2 } } } }
]);
```

출력

```
[
  { "_id": "math", "highestTwo": [85, 93] },
  { "_id": "science", "highestTwo": [88, 90] }
]
```

표현식 사용 예제(MongoDB Shell)

연$maxN산자를 $project 단계 내의 표현식으로 사용하여 배열 필드에서 가장 큰 N 요소를 반환할 수도 있습니다.

샘플 문서 생성

```
db.readings.insertMany([
  { _id: 1, sensor: "A", values: [45, 12, 78, 3, 56] },
  { _id: 2, sensor: "B", values: [90, 23, 67, 11, 44] }
]);
```

쿼리 예제

```
db.readings.aggregate([
  { $project: {
      highestThree: { $maxN: { input: "$values", n: 3 } }
    }}
]);
```

출력

```
[
  { "_id": 1, "highestThree": [45, 56, 78] },
  { "_id": 2, "highestThree": [44, 90, 67] }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$maxN산자는 $setWindowFields 스테이지에서 창 연산자로도 사용할 수 있습니다. 이 컨텍스트에서는 각 창의 문서에 대해 최대 n 최대 값의 배열을 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

> **Note**
>
> 에서 창 연산자로 사용할 경우 $setWindowFields$maxN는 100MB의 중간 데이터로 제한됩니다. 이 제한을 초과하는 작업은 오류를 반환합니다.

샘플 문서 생성

```
db.stockPrices.insertMany([
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

쿼리 예제

다음 예제에서는 문서를 기준으로 분할하고ticker, 각 파티션을 기준으로 정렬하고hour, 파티션 시작부터 현재 문서까지 가장 높은 가격 2개를 반환합니다.

```
db.stockPrices.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        highestTwoSoFar: {
          $maxN: { input: "$price", n: 2 },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "highestTwoSoFar": [50] },
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "highestTwoSoFar": [50, 45] },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "highestTwoSoFar": [60, 50] },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "highestTwoSoFar": [60, 50] },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "highestTwoSoFar": [30] },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "highestTwoSoFar": [35, 30] },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "highestTwoSoFar": [35, 30] }
]
```

각 문서는 창 내에서 가장 큰 두 price 값highestTwoSoFar인 로 보강됩니다. 출력 순서 섹션에 설명된 대로 반환된 배열 내의 값 순서는 보장되지 않습니다.

코드 예제

$maxN 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누적기 사용량(의 $group), 표현식 사용량(의 $project) 및 창 연산자 사용량(의)을 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');

    // Accumulator usage: N largest values per group
    const scores = db.collection('scores');
    await scores.insertMany([
      { subject: "math", score: 85 },
      { subject: "math", score: 72 },
      { subject: "math", score: 93 },
      { subject: "math", score: 68 },
      { subject: "science", score: 90 },
      { subject: "science", score: 78 },
```

```
      { subject: "science", score: 65 },
      { subject: "science", score: 88 }
    ]);
    const accumulatorResult = await scores.aggregate([
      { $group: { _id: "$subject", highestTwo: { $maxN: { input: "$score", n:
2 } } } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Expression usage: N largest elements from an array field
    const readings = db.collection('readings');
    await readings.insertMany([
      { _id: 1, sensor: "A", values: [45, 12, 78, 3, 56] },
      { _id: 2, sensor: "B", values: [90, 23, 67, 11, 44] }
    ]);
    const expressionResult = await readings.aggregate([
      { $project: { highestThree: { $maxN: { input: "$values", n: 3 } } } }
    ]).toArray();
    console.log('Expression result:', expressionResult);

    // Window operator usage: N largest values over a window in each partition
    const stockPrices = db.collection('stockPrices');
    await stockPrices.insertMany([
      { _id: 1, ticker: "ABC", hour: 1, price: 50 },
      { _id: 2, ticker: "ABC", hour: 2, price: 45 },
      { _id: 3, ticker: "ABC", hour: 3, price: 60 },
      { _id: 4, ticker: "ABC", hour: 4, price: 40 },
      { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
      { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
      { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
    ]);
    const windowResult = await stockPrices.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$ticker",
          sortBy: { hour: 1 },
          output: {
            highestTwoSoFar: {
              $maxN: { input: "$price", n: 2 },
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
```

```
    ]).toArray();
    console.log('Window result:', windowResult);

  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']

        # Accumulator usage: N largest values per group
        scores = db['scores']
        scores.insert_many([
            { 'subject': 'math', 'score': 85 },
            { 'subject': 'math', 'score': 72 },
            { 'subject': 'math', 'score': 93 },
            { 'subject': 'math', 'score': 68 },
            { 'subject': 'science', 'score': 90 },
            { 'subject': 'science', 'score': 78 },
            { 'subject': 'science', 'score': 65 },
            { 'subject': 'science', 'score': 88 }
        ])
        accumulator_result = list(scores.aggregate([
            { '$group': { '_id': '$subject', 'highestTwo': { '$maxN': { 'input':
 '$score', 'n': 2 } } } }
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: N largest elements from an array field
        readings = db['readings']
        readings.insert_many([
```

```
            { '_id': 1, 'sensor': 'A', 'values': [45, 12, 78, 3, 56] },
            { '_id': 2, 'sensor': 'B', 'values': [90, 23, 67, 11, 44] }
        ])
        expression_result = list(readings.aggregate([
            { '$project': { 'highestThree': { '$maxN': { 'input': '$values', 'n':
 3 } } } }
        ]))
        print('Expression result:', expression_result)

        # Window operator usage: N largest values over a window in each partition
        stock_prices = db['stockPrices']
        stock_prices.insert_many([
            { '_id': 1, 'ticker': 'ABC', 'hour': 1, 'price': 50 },
            { '_id': 2, 'ticker': 'ABC', 'hour': 2, 'price': 45 },
            { '_id': 3, 'ticker': 'ABC', 'hour': 3, 'price': 60 },
            { '_id': 4, 'ticker': 'ABC', 'hour': 4, 'price': 40 },
            { '_id': 5, 'ticker': 'XYZ', 'hour': 1, 'price': 30 },
            { '_id': 6, 'ticker': 'XYZ', 'hour': 2, 'price': 35 },
            { '_id': 7, 'ticker': 'XYZ', 'hour': 3, 'price': 25 }
        ])
        window_result = list(stock_prices.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$ticker',
                    'sortBy': { 'hour': 1 },
                    'output': {
                        'highestTwoSoFar': {
                            '$maxN': { 'input': '$price', 'n': 2 },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()

example()
```

$median

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $median 연산자는 숫자 데이터의 중앙값을 계산합니다. 누적기는 집계 파이프라인의 $group 단계에서 그룹 내 문서 전반의 숫자 값 중앙값을 계산합니다. 표현식으로 숫자 배열의 중앙값을 계산합니다.

파라미터

- input: 숫자 값 또는 숫자 값 배열로 확인되는 표현식입니다.
- method: 계산 방법을 지정하는 문자열입니다. 현재"approximate"는 t-digest 알고리즘을 사용하는 만 지원됩니다.

동작

"approximate" 메서드는 t-digest 알고리즘을 사용하여 대략적인 중앙값을 계산합니다. 결과는 값 간의 보간이 아닌 데이터 세트의 기존 값입니다. 데이터 포인트 수가 증가함에 따라 정밀도가 향상됩니다.

예제(MongoDB 쉘)

다음 예제에서는 $median 연산자를 사용하여 클래스당 테스트 점수 중앙값을 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.students.insertMany([
  { class: "A", score: 72 },
  { class: "A", score: 85 },
  { class: "A", score: 90 },
  { class: "A", score: 68 },
  { class: "A", score: 95 },
  { class: "B", score: 80 },
  { class: "B", score: 75 },
  { class: "B", score: 92 },
  { class: "B", score: 88 },
  { class: "B", score: 70 }
]);
```

쿼리 예제

```
db.students.aggregate([
  { $group: {
      _id: "$class",
      medianScore: { $median: { input: "$score", method: "approximate" } }
    }}
]);
```

출력

```
[
  { "_id": "A", "medianScore": 85 },
  { "_id": "B", "medianScore": 80 }
]
```

표현식 사용 예제(MongoDB Shell)

연$median산자를 $project 단계 내의 표현식으로 사용하여 배열 필드의 중앙값을 계산할 수도 있습니다.

샘플 문서 생성

```
db.surveys.insertMany([
  { _id: 1, ratings: [3, 5, 7, 9, 2] },
  { _id: 2, ratings: [10, 20, 30, 40, 50] },
  { _id: 3, ratings: [1, 1, 2, 3, 5] }
]);
```

쿼리 예제

```
db.surveys.aggregate([
  { $project: {
      medianRating: { $median: { input: "$ratings", method: "approximate" } }
    }}
]);
```

출력

```
[
  { "_id": 1, "medianRating": 5 },
  { "_id": 2, "medianRating": 30 },
  { "_id": 3, "medianRating": 2 }
```

```
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$median산자를 $setWindowFields 단계에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 문서에 대한 숫자 값의 중앙값을 계산합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

> **Note**
>
> 에서 창 연산자로 사용할 경우 $setWindowFields$median는 100MB의 중간 데이터로 제한됩니다. 이 제한을 초과하는 작업은 오류를 반환합니다.

샘플 문서 생성

```
db.temperatures.insertMany([
  { _id: 1, city: "A", reading: 60 },
  { _id: 2, city: "A", reading: 65 },
  { _id: 3, city: "A", reading: 70 },
  { _id: 4, city: "B", reading: 80 },
  { _id: 5, city: "B", reading: 85 },
  { _id: 6, city: "B", reading: 90 }
]);
```

쿼리 예제

다음 예시에서는 문서를 기준으로 분할city하고 각 파티션의 모든 문서에 reading 대한 중앙값을 계산합니다.

```
db.temperatures.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$city",
      output: {
        medianReading: {
          $median: { input: "$reading", method: "approximate" },
          window: { documents: ["unbounded", "unbounded"] }
        }
      }
```

```
    }
  }
]);
```

출력

```
[
  { "_id": 1, "city": "A", "reading": 60, "medianReading": 65 },
  { "_id": 2, "city": "A", "reading": 65, "medianReading": 65 },
  { "_id": 3, "city": "A", "reading": 70, "medianReading": 65 },
  { "_id": 4, "city": "B", "reading": 80, "medianReading": 85 },
  { "_id": 5, "city": "B", "reading": 85, "medianReading": 85 },
  { "_id": 6, "city": "B", "reading": 90, "medianReading": 85 }
]
```

각 문서는 파티션에 있는 reading 모든 문서의 대략적인 중앙값medianReading인 로 보강됩니다.

코드 예제

$median 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누적기 사용량(의 $group), 표현식 사용량(의 $project) 및 창 연산자 사용량(의)을 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();
    const db = client.db('test');

    // Accumulator usage: median across grouped documents
    const students = db.collection('students');
    await students.insertMany([
      { class: "A", score: 72 },
      { class: "A", score: 85 },
      { class: "A", score: 90 },
```

```
    { class: "A", score: 68 },
    { class: "A", score: 95 },
    { class: "B", score: 80 },
    { class: "B", score: 75 },
    { class: "B", score: 92 },
    { class: "B", score: 88 },
    { class: "B", score: 70 }
]);
const accumulatorResult = await students.aggregate([
  { $group: {
      _id: "$class",
      medianScore: { $median: { input: "$score", method: "approximate" } }
    }}
]).toArray();
console.log('Accumulator result:', accumulatorResult);

// Expression usage: median of an array field
const surveys = db.collection('surveys');
await surveys.insertMany([
  { _id: 1, ratings: [3, 5, 7, 9, 2] },
  { _id: 2, ratings: [10, 20, 30, 40, 50] },
  { _id: 3, ratings: [1, 1, 2, 3, 5] }
]);
const expressionResult = await surveys.aggregate([
  { $project: {
      medianRating: { $median: { input: "$ratings", method: "approximate" } }
    }}
]).toArray();
console.log('Expression result:', expressionResult);

// Window operator usage: median across each partition
const temperatures = db.collection('temperatures');
await temperatures.insertMany([
  { _id: 1, city: "A", reading: 60 },
  { _id: 2, city: "A", reading: 65 },
  { _id: 3, city: "A", reading: 70 },
  { _id: 4, city: "B", reading: 80 },
  { _id: 5, city: "B", reading: 85 },
  { _id: 6, city: "B", reading: 90 }
]);
const windowResult = await temperatures.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$city",
```

```
          output: {
            medianReading: {
              $median: { input: "$reading", method: "approximate" },
              window: { documents: ["unbounded", "unbounded"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']

        # Accumulator usage: median across grouped documents
        students = db['students']
        students.insert_many([
            { 'class': 'A', 'score': 72 },
            { 'class': 'A', 'score': 85 },
            { 'class': 'A', 'score': 90 },
            { 'class': 'A', 'score': 68 },
            { 'class': 'A', 'score': 95 },
            { 'class': 'B', 'score': 80 },
            { 'class': 'B', 'score': 75 },
            { 'class': 'B', 'score': 92 },
            { 'class': 'B', 'score': 88 },
            { 'class': 'B', 'score': 70 }
```

```
        ])
        accumulator_result = list(students.aggregate([
            { '$group': {
                '_id': '$class',
                'medianScore': { '$median': { 'input': '$score', 'method':
'approximate' } }
            }}
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: median of an array field
        surveys = db['surveys']
        surveys.insert_many([
            { '_id': 1, 'ratings': [3, 5, 7, 9, 2] },
            { '_id': 2, 'ratings': [10, 20, 30, 40, 50] },
            { '_id': 3, 'ratings': [1, 1, 2, 3, 5] }
        ])
        expression_result = list(surveys.aggregate([
            { '$project': {
                'medianRating': { '$median': { 'input': '$ratings', 'method':
'approximate' } }
            }}
        ]))
        print('Expression result:', expression_result)

        # Window operator usage: median across each partition
        temperatures = db['temperatures']
        temperatures.insert_many([
            { '_id': 1, 'city': 'A', 'reading': 60 },
            { '_id': 2, 'city': 'A', 'reading': 65 },
            { '_id': 3, 'city': 'A', 'reading': 70 },
            { '_id': 4, 'city': 'B', 'reading': 80 },
            { '_id': 5, 'city': 'B', 'reading': 85 },
            { '_id': 6, 'city': 'B', 'reading': 90 }
        ])
        window_result = list(temperatures.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$city',
                    'output': {
                        'medianReading': {
                            '$median': { 'input': '$reading', 'method':
'approximate' },
                            'window': { 'documents': ['unbounded', 'unbounded'] }
```

```
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()

example()
```

$meta

$meta 집계 연산자는 집계 파이프라인의 문서와 연결된 메타데이터에 액세스합니다. 일반적으로 텍스트 검색 점수를 검색하고 관련성을 기준으로 결과를 정렬하는 데 사용됩니다.

파라미터

- textScore: 검색 쿼리와 문서 관련성을 나타내는 텍스트 검색 점수를 검색합니다.

예제(MongoDB 쉘)

다음 예제에서는 집계 파이프라인에서 $meta 연산자를 사용하여 텍스트 검색 점수를 검색하고 정렬하는 방법을 보여줍니다.

샘플 문서 생성

```
db.articles.createIndex({ content: "text" });

db.articles.insertMany([
  { _id: 1, title: "Python Programming", content: "Python is a versatile programming
 language used for web development." },
  { _id: 2, title: "Python Guide", content: "Learn Python programming with Python
 tutorials and Python examples." },
  { _id: 3, title: "Java Basics", content: "Java is another popular programming
 language." }
]);
```

쿼리 예제

```
db.articles.aggregate([
  { $match: { $text: { $search: "Python" } } },
  { $addFields: { score: { $meta: "textScore" } } },
  { $sort: { score: -1 } }
]);
```

출력

```
[
  {
    _id: 2,
    title: 'Python Guide',
    content: 'Learn Python programming with Python tutorials and Python examples.',
    score: 1.5
  },
  {
    _id: 1,
    title: 'Python Programming',
    content: 'Python is a versatile programming language used for web development.',
    score: 0.75
  }
]
```

코드 예제

$meta 집계 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('articles');

  const result = await collection.aggregate([
    { $match: { $text: { $search: "Python" } } },
    { $addFields: { score: { $meta: "textScore" } } },
    { $sort: { score: -1 } }
```

```
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['articles']

    result = list(collection.aggregate([
        { '$match': { '$text': { '$search': 'Python' } } },
        { '$addFields': { 'score': { '$meta': 'textScore' } } },
        { '$sort': { 'score': -1 } }
    ]))

    print(result)
    client.close()

example()
```

$merge

8.0에 도입

Amazon DocumentDB의 $merge 집계 단계는 이전 파이프라인 단계의 결과를 대상 컬렉션에 병합하는 데 사용됩니다. 이는 입력 문서의 데이터를 기반으로 대상 컬렉션에 문서를 업데이트하거나 삽입하는 데 유용합니다.

$merge 단계를 사용하면 입력 문서와 대상 컬렉션 간의 일치 조건에 따라 다음과 같은 다양한 작업을 수행할 수 있습니다.

```
- Insert new documents
```

```
- Update existing documents
- Delete documents
- Fail the operation if there are any conflicts
```

파라미터

- into: (필수) 입력 문서를 병합할 대상 컬렉션의 이름입니다.
- on: (필수) 입력 문서와 대상 컬렉션 간의 일치 조건으로 사용할 필드(들)입니다.
- whenMatched: (선택 사항) 입력 문서가 대상 컬렉션의 기존 문서와 일치할 때 수행할 작업입니다. 지원되는 값은 "merge", "replace", "keepExisting" 및 "fail"입니다. Amazon DocumentDB는 집계 파이프라인을 whenMatched 값으로 지정하는 기능을 지원하지 않습니다.
- whenNotMatched: (선택 사항) 입력 문서가 대상 컬렉션의 문서와 일치하지 않을 때 수행할 작업입니다. 지원되는 값은 "insert" 및 입니다"fail".
- bypassDocumentValidation: (선택 사항) 로 설정하면 $merge 단계에 의해 대상 컬렉션에 작성된 true문서는 해당 컬렉션에 정의된 스키마 검증 규칙($jsonSchema)을 우회합니다. 기본값은 대상 컬렉션의 검증 규칙을 false적용하는 입니다. 이는 Amazon DocumentDB 8.0에서 지원됩니다.

예제(MongoDB 쉘)

다음 예제에서는 $merge 스테이지를 사용하여 입력 파이프라인의 새 데이터로 users 컬렉션을 업데이트하는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { _id: 1, name: "John Doe", email: "john@example.com" },
  { _id: 2, name: "Jane Smith", email: "jane@example.com" }
]);

db.inputData.insertMany([
  { _id: 1, name: "John Doe", email: "john@example.com", phone: "123-456-7890" },
  { _id: 3, name: "Bob Johnson", email: "bob@example.com", phone: "987-654-3210" }
]);
```

쿼리 예제

```
db.inputData.aggregate([
  {
```

```
    $merge: {
      into: "users",
      on: "_id",
      whenMatched: "merge",
      whenNotMatched: "insert"
    }
  }
])
```

출력

$merge 파이프라인을 실행한 후 users 컬렉션에는 다음 문서가 포함됩니다.

```
[
  { _id: 1, name: "John Doe", email: "john@example.com", phone: "123-456-7890" },
  { _id: 2, name: "Jane Smith", email: "jane@example.com" },
  { _id: 3, name: "Bob Johnson", email: "bob@example.com", phone: "987-654-3210" }
]
```

코드 예제

$merge 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

다음은 Node.js 애플리케이션에서 $merge 연산자를 사용하는 예입니다.

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');

  await db.collection('inputData').aggregate([
    {
      $merge: {
        into: 'users',
        on: '_id',
        whenMatched: 'merge',
        whenNotMatched: 'insert'
```

```
      }
    }
  ]).toArray();

  const users = await db.collection('users').find({}).toArray();
  console.log(users);

  await client.close();
}

example();
```

Python

다음은 Python 애플리케이션에서 $merge 연산자를 사용하는 예입니다.

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']

    # Assumes collections 'users' and 'inputData' already exist with sample data
    db.inputData.aggregate([
        {
            '$merge': {
                'into': 'users',
                'on': '_id',
                'whenMatched': 'merge',
                'whenNotMatched': 'insert'
            }
        }
    ])

    users = list(db.users.find({}))
    print(users)

    client.close()

example()
```

$mergeObjects

Amazon DocumentDB의 $mergeObjects 연산자는 여러 문서 또는 객체를 단일 문서로 결합하는 데 사용됩니다. 이 연산자는 두 개 이상의 문서 또는 객체의 내용을 병합해야 할 때 특히 유용하며, 한 객체의 값을 다른 객체의 값으로 덮어쓸 수 있습니다.

파라미터

- expression1: 병합할 첫 번째 객체입니다.
- expression2: (선택 사항) 병합할 두 번째 객체입니다.
- expression3: (선택 사항) 병합할 추가 객체입니다.

예제(MongoDB 쉘)

다음 예제에서는 $mergeObjects 연산자를 사용하여 두 객체를 결합하는 방법을 보여줍니다.

샘플 문서 생성

```
db.collection.insertMany([
  { "_id": 1, "name": "John", "address": { "city": "New York", "state": "NY" } },
  { "_id": 2, "name": "Jane", "address": { "city": "Los Angeles", "state": "CA" } }
]);
```

쿼리 예제

```
db.collection.aggregate([
  {
    $project: {
      "combinedAddress": {
        $mergeObjects: ["$address", { "country": "USA" }]
      }
    }
  }
])
```

출력

```
[
  {
    "_id": 1,
    "combinedAddress": {
```

```
      "city": "New York",
      "state": "NY",
      "country": "USA"
    }
  },
  {
    "_id": 2,
    "combinedAddress": {
      "city": "Los Angeles",
      "state": "CA",
      "country": "USA"
    }
  }
]
```

코드 예제

$mergeObjects 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.aggregate([
    {
      $project: {
        "combinedAddress": {
          $mergeObjects: ["$address", { "country": "USA" }]
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}
```

```
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['collection']

    result = list(collection.aggregate([
        {
            '$project': {
                "combinedAddress": {
                    "$mergeObjects": ["$address", { "country": "USA" }]
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$millisecond

Amazon DocumentDB의 `$millisecond` 연산자는 날짜 값의 밀리초 부분을 추출하는 데 사용됩니다.

파라미터

없음

예제(MongoDB 쉘)

이 예제에서는 `$millisecond` 연산자를 사용하여 날짜 값의 밀리초 부분을 추출하는 방법을 보여줍니다.

샘플 문서 생성

```
db.events.insert([
  {
    "name": "Event 1",
    "timestamp": ISODate("2023-04-21T10:30:15.123Z")
  },
  {
    "name": "Event 2",
    "timestamp": ISODate("2023-04-21T10:30:15.456Z")
  },
  {
    "name": "Event 3",
    "timestamp": ISODate("2023-04-21T10:30:15.789Z")
  }
])
```

쿼리 예제

```
db.events.aggregate([
  {
    $project: {
      name: 1,
      milliseconds: { $millisecond: "$timestamp" }
    }
  }
])
```

출력

```
[
  {
    "_id": ObjectId("644332a42054ed1b0d15f0c1"),
    "name": "Event 1",
    "milliseconds": 123
  },
  {
    "_id": ObjectId("644332a42054ed1b0d15f0c2"),
    "name": "Event 2",
    "milliseconds": 456
  },
  {
    "_id": ObjectId("644332a42054ed1b0d15f0c3"),
    "name": "Event 3",
```

```
    "milliseconds": 789
  }
]
```

코드 예제

$millisecond 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const events = db.collection('events');

  const result = await events.aggregate([
    {
      $project: {
        name: 1,
        milliseconds: { $millisecond: '$timestamp' }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
```

```
events = db['events']

result = list(events.aggregate([
    {
        '$project': {
            'name': 1,
            'milliseconds': { '$millisecond': '$timestamp' }
        }
    }
]))

print(result)

client.close()
```

$min

$min 집계 연산자는 문서 세트에서 지정된 필드의 최소값을 반환합니다. 이 연산자는 문서 세트에서 가장 낮은 값을 찾는 데 유용합니다.

파라미터

- expression: 평가할 표현식입니다. 필드 경로, 변수 또는 값으로 확인되는 표현식일 수 있습니다.

예제(MongoDB 쉘)

다음 예제에서는 연$min산자를 사용하여 여러 문서에서 age 필드의 최소값을 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { name: "John", age: 35 },
  { name: "Jane", age: 28 },
  { name: "Bob", age: 42 },
  { name: "Alice", age: 31 }
]);
```

쿼리 예제

```
db.users.aggregate([
```

```
  { $group: { _id: null, minAge: { $min: "$age" } } },
  { $project: { _id: 0, minAge: 1 } }
])
```

출력

```
[ { minAge: 28 } ]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$min산자는 $setWindowFields 스테이지에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 문서에 대한 최소값을 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

> **Note**
>
> 에서 창 연산자로 사용할 경우 $setWindowFields$min는 100MB의 중간 데이터로 제한됩니다. 이 제한을 초과하는 작업은 오류를 반환합니다.

샘플 문서 생성

```
db.stockPrices.insertMany([
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

쿼리 예제

다음 예제에서는 문서를 기준으로 분할하고ticker, 각 파티션을 기준으로 정렬하고hour, 파티션 시작부터 현재 문서를 통해 표시되는 최저 가격을 반환합니다.

```
db.stockPrices.aggregate([
  {
```

```
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        lowestSoFar: {
          $min: "$price",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "lowestSoFar": 50 },
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "lowestSoFar": 45 },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "lowestSoFar": 45 },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "lowestSoFar": 40 },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "lowestSoFar": 30 },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "lowestSoFar": 30 },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "lowestSoFar": 25 }
]
```

각 문서는 현재 문서까지 파티션 price 내에서 실행되는 최소 lowestSoFar인 로 보강됩니다.

코드 예제

$min 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누산기 사용량()과 창 연산자 사용량($group)을 모두 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');

  // Accumulator usage: minimum value across a group
```

```
  const users = db.collection('users');
  await users.insertMany([
    { name: "John", age: 35 },
    { name: "Jane", age: 28 },
    { name: "Bob", age: 42 },
    { name: "Alice", age: 31 }
  ]);
  const accumulatorResult = await users.aggregate([
    { $group: { _id: null, minAge: { $min: "$age" } } }
  ]).toArray();
  console.log('Accumulator result:', accumulatorResult);

  // Window operator usage: running minimum within each partition
  const stockPrices = db.collection('stockPrices');
  await stockPrices.insertMany([
    { _id: 1, ticker: "ABC", hour: 1, price: 50 },
    { _id: 2, ticker: "ABC", hour: 2, price: 45 },
    { _id: 3, ticker: "ABC", hour: 3, price: 60 },
    { _id: 4, ticker: "ABC", hour: 4, price: 40 },
    { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
    { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
    { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
  ]);
  const windowResult = await stockPrices.aggregate([
    {
      $setWindowFields: {
        partitionBy: "$ticker",
        sortBy: { hour: 1 },
        output: {
          lowestSoFar: {
            $min: "$price",
            window: { documents: ["unbounded", "current"] }
          }
        }
      }
    }
  ]).toArray();
  console.log('Window result:', windowResult);

  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']

    # Accumulator usage: minimum value across a group
    users = db.users
    users.insert_many([
        { "name": "John", "age": 35 },
        { "name": "Jane", "age": 28 },
        { "name": "Bob", "age": 42 },
        { "name": "Alice", "age": 31 }
    ])
    accumulator_result = list(users.aggregate([
        { "$group": { "_id": None, "minAge": { "$min": "$age" } } }
    ]))
    print('Accumulator result:', accumulator_result)

    # Window operator usage: running minimum within each partition
    stock_prices = db.stockPrices
    stock_prices.insert_many([
        { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50 },
        { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45 },
        { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60 },
        { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40 },
        { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30 },
        { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35 },
        { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25 }
    ])
    window_result = list(stock_prices.aggregate([
        {
            "$setWindowFields": {
                "partitionBy": "$ticker",
                "sortBy": { "hour": 1 },
                "output": {
                    "lowestSoFar": {
                        "$min": "$price",
                        "window": { "documents": ["unbounded", "current"] }
                    }
```

```
                }
            }
        }
    ]))
    print('Window result:', window_result)

    client.close()

example()
```

$minN

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $minN 연산자는 N개의 가장 작은 값을 반환합니다. $group 단계에서 누적기로 사용할 경우 각 그룹의 N 최소값 배열을 반환합니다. 배열 표현식 연산자로 사용할 경우 배열의 N 최소 요소를 반환합니다.

파라미터

- input: 최소값을 반환할 배열 또는 필드로 확인되는 표현식입니다n.
- n: 반환할 최소값 수를 지정하는 양의 정수로 확인되는 표현식입니다.

동작

배열 표현식 연산자 동작

- 보다 n 작은 값은 지정할 수 없습니다1.
- $minN는 input 배열에서 찾은 null 값을 필터링합니다.
- 지정된 n가 input 배열의 요소 수보다 크거나 같으면는 input 배열의 모든 요소를 $minN 반환합니다.
- 가 배열이 아닌 값으로 input 확인되면 집계 작업 오류가 발생합니다.
- 에 숫자 및 문자열 요소가 모두 input 포함된 경우 숫자 요소는 BSON 비교 순서에 따라 문자열 요소 앞에 정렬됩니다.

누산기 동작

- 누적기로 사용할 경우는 상수이거나의 _id 값에 따라 달라지는 양의 정수 표현식이어야 n 합니다$group.
- $minN는 null 및 누락 값을 필터링합니다.
- 그룹에 개 미만의 n 요소가 포함된 경우는 그룹의 모든 요소를 $minN 반환합니다.
- $minN는 BSON 비교 순서에 따라 입력 데이터를 비교하여 적절한 출력 유형을 결정합니다. 입력 데이터에 여러 데이터 형식이 포함된 경우 $minN 출력 형식은 비교 순서에서 가장 낮습니다.

출력 순서 지정

$minN는 특정 정렬 순서로 값을 반환하지 않습니다. 특정 정렬 순서를 보장해야 하는 경우 $bottomN 대신를 사용하거나 결과를 로 래핑합니다$sortArray.

예제(MongoDB 쉘)

다음 예제에서는 $minN 누적기를 사용하여 각 주제에 대해 가장 낮은 두 점수를 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.scores.insertMany([
  { subject: "math", score: 85 },
  { subject: "math", score: 72 },
  { subject: "math", score: 93 },
  { subject: "math", score: 68 },
  { subject: "science", score: 90 },
  { subject: "science", score: 78 },
  { subject: "science", score: 65 },
  { subject: "science", score: 88 }
]);
```

쿼리 예제

```
db.scores.aggregate([
  { $group: { _id: "$subject", lowestTwo: { $minN: { input: "$score", n: 2 } } } }
]);
```

출력

```
[
```

```
  { "_id": "math", "lowestTwo": [72, 68] },
  { "_id": "science", "lowestTwo": [78, 65] }
]
```

표현식 사용 예제(MongoDB Shell)

연$minN산자를 $project 스테이지 내의 표현식으로 사용하여 배열 필드에서 가장 작은 N 요소를 반환할 수도 있습니다.

샘플 문서 생성

```
db.readings.insertMany([
  { _id: 1, sensor: "A", values: [45, 12, 78, 3, 56] },
  { _id: 2, sensor: "B", values: [90, 23, 67, 11, 44] }
]);
```

쿼리 예제

```
db.readings.aggregate([
  { $project: {
      lowestThree: { $minN: { input: "$values", n: 3 } }
    }}
]);
```

출력

```
[
  { "_id": 1, "lowestThree": [45, 12, 3] },
  { "_id": 2, "lowestThree": [44, 23, 11] }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$minN산자는 $setWindowFields 스테이지에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 문서에 대해 가장 n 작은 값까지 배열을 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

> **Note**
>
> 에서 창 연산자로 사용할 경우 `$setWindowFields` `$minN`는 100MB의 중간 데이터로 제한됩니다. 이 제한을 초과하는 작업은 오류를 반환합니다.

샘플 문서 생성

```
db.stockPrices.insertMany([
  { _id: 1, ticker: "ABC", hour: 1, price: 50 },
  { _id: 2, ticker: "ABC", hour: 2, price: 45 },
  { _id: 3, ticker: "ABC", hour: 3, price: 60 },
  { _id: 4, ticker: "ABC", hour: 4, price: 40 },
  { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
  { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
  { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
]);
```

쿼리 예제

다음 예제에서는를 기준으로 문서를 분할하고`ticker`,를 기준으로 각 파티션을 정렬하고`hour`, 파티션 시작부터 현재 문서를 통해 표시되는 최저 가격 2개를 반환합니다.

```
db.stockPrices.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$ticker",
      sortBy: { hour: 1 },
      output: {
        lowestTwoSoFar: {
          $minN: { input: "$price", n: 2 },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "ticker": "ABC", "hour": 1, "price": 50, "lowestTwoSoFar": [50] },
```

```
  { "_id": 2, "ticker": "ABC", "hour": 2, "price": 45, "lowestTwoSoFar": [45, 50] },
  { "_id": 3, "ticker": "ABC", "hour": 3, "price": 60, "lowestTwoSoFar": [45, 50] },
  { "_id": 4, "ticker": "ABC", "hour": 4, "price": 40, "lowestTwoSoFar": [40, 45] },
  { "_id": 5, "ticker": "XYZ", "hour": 1, "price": 30, "lowestTwoSoFar": [30] },
  { "_id": 6, "ticker": "XYZ", "hour": 2, "price": 35, "lowestTwoSoFar": [30, 35] },
  { "_id": 7, "ticker": "XYZ", "hour": 3, "price": 25, "lowestTwoSoFar": [25, 30] }
]
```

각 문서는 창 내에서 가장 작은 두 price 값lowestTwoSoFar인 로 보강됩니다. 출력 순서 섹션에 설명된 대로 반환된 배열 내의 값 순서는 보장되지 않습니다.

코드 예제

$minN 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누적기 사용량(의 $group), 표현식 사용량(의 $project) 및 창 연산자 사용량(의)을 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');

    // Accumulator usage: N smallest values per group
    const scores = db.collection('scores');
    await scores.insertMany([
      { subject: "math", score: 85 },
      { subject: "math", score: 72 },
      { subject: "math", score: 93 },
      { subject: "math", score: 68 },
      { subject: "science", score: 90 },
      { subject: "science", score: 78 },
      { subject: "science", score: 65 },
      { subject: "science", score: 88 }
    ]);
    const accumulatorResult = await scores.aggregate([
```

```
      { $group: { _id: "$subject", lowestTwo: { $minN: { input: "$score", n:
2 } } } }
   ]).toArray();
   console.log('Accumulator result:', accumulatorResult);

   // Expression usage: N smallest elements from an array field
   const readings = db.collection('readings');
   await readings.insertMany([
     { _id: 1, sensor: "A", values: [45, 12, 78, 3, 56] },
     { _id: 2, sensor: "B", values: [90, 23, 67, 11, 44] }
   ]);
   const expressionResult = await readings.aggregate([
     { $project: { lowestThree: { $minN: { input: "$values", n: 3 } } } }
   ]).toArray();
   console.log('Expression result:', expressionResult);

   // Window operator usage: N smallest values over a window in each partition
   const stockPrices = db.collection('stockPrices');
   await stockPrices.insertMany([
     { _id: 1, ticker: "ABC", hour: 1, price: 50 },
     { _id: 2, ticker: "ABC", hour: 2, price: 45 },
     { _id: 3, ticker: "ABC", hour: 3, price: 60 },
     { _id: 4, ticker: "ABC", hour: 4, price: 40 },
     { _id: 5, ticker: "XYZ", hour: 1, price: 30 },
     { _id: 6, ticker: "XYZ", hour: 2, price: 35 },
     { _id: 7, ticker: "XYZ", hour: 3, price: 25 }
   ]);
   const windowResult = await stockPrices.aggregate([
     {
       $setWindowFields: {
         partitionBy: "$ticker",
         sortBy: { hour: 1 },
         output: {
           lowestTwoSoFar: {
             $minN: { input: "$price", n: 2 },
             window: { documents: ["unbounded", "current"] }
           }
         }
       }
     }
   ]).toArray();
   console.log('Window result:', windowResult);

 } finally {
```

```
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']

        # Accumulator usage: N smallest values per group
        scores = db['scores']
        scores.insert_many([
            { 'subject': 'math', 'score': 85 },
            { 'subject': 'math', 'score': 72 },
            { 'subject': 'math', 'score': 93 },
            { 'subject': 'math', 'score': 68 },
            { 'subject': 'science', 'score': 90 },
            { 'subject': 'science', 'score': 78 },
            { 'subject': 'science', 'score': 65 },
            { 'subject': 'science', 'score': 88 }
        ])
        accumulator_result = list(scores.aggregate([
            { '$group': { '_id': '$subject', 'lowestTwo': { '$minN': { 'input':
 '$score', 'n': 2 } } } }
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: N smallest elements from an array field
        readings = db['readings']
        readings.insert_many([
            { '_id': 1, 'sensor': 'A', 'values': [45, 12, 78, 3, 56] },
            { '_id': 2, 'sensor': 'B', 'values': [90, 23, 67, 11, 44] }
        ])
        expression_result = list(readings.aggregate([
```

```
            { '$project': { 'lowestThree': { '$minN': { 'input': '$values', 'n':
 3 } } } }
        ]))
        print('Expression result:', expression_result)

        # Window operator usage: N smallest values over a window in each partition
        stock_prices = db['stockPrices']
        stock_prices.insert_many([
            { '_id': 1, 'ticker': 'ABC', 'hour': 1, 'price': 50 },
            { '_id': 2, 'ticker': 'ABC', 'hour': 2, 'price': 45 },
            { '_id': 3, 'ticker': 'ABC', 'hour': 3, 'price': 60 },
            { '_id': 4, 'ticker': 'ABC', 'hour': 4, 'price': 40 },
            { '_id': 5, 'ticker': 'XYZ', 'hour': 1, 'price': 30 },
            { '_id': 6, 'ticker': 'XYZ', 'hour': 2, 'price': 35 },
            { '_id': 7, 'ticker': 'XYZ', 'hour': 3, 'price': 25 }
        ])
        window_result = list(stock_prices.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$ticker',
                    'sortBy': { 'hour': 1 },
                    'output': {
                        'lowestTwoSoFar': {
                            '$minN': { 'input': '$price', 'n': 2 },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()

example()
```

$minute

Amazon DocumentDB의 $minute 집계 파이프라인 단계는 날짜 또는 타임스탬프 필드에서 분 값을 추출합니다.

이 연산자는 집계 파이프라인 내에서 날짜 및 시간 기반 계산 또는 그룹화를 수행해야 할 때 유용합니다.

파라미터

- `expression`: 분 값을 추출할 날짜 또는 타임스탬프 필드입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$minute` 연산자를 사용하여 타임스탬프 필드에서 추출한 분 값을 기준으로 문서를 그룹화하고 각 그룹의 문서 수를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.events.insertMany([
  { timestamp: new Date("2023-04-15T10:30:25.000Z") },
  { timestamp: new Date("2023-04-15T10:30:35.000Z") },
  { timestamp: new Date("2023-04-15T10:31:05.000Z") },
  { timestamp: new Date("2023-04-15T10:31:45.000Z") },
  { timestamp: new Date("2023-04-15T10:32:15.000Z") }
]);
```

쿼리 예제

```
db.events.aggregate([
  {
    $group: {
      _id: {
        minute: { $minute: "$timestamp" }
      },
      count: { $count: {} }
    }
  },
  { $sort: { "_id.minute": 1 } }
]);
```

출력

```
[
  { "_id": { "minute": 30 }, "count": 2 },
  { "_id": { "minute": 31 }, "count": 2 },
```

```
  { "_id": { "minute": 32 }, "count": 1 }
]
```

코드 예제

$minute 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('events');

  await collection.insertMany([
    { timestamp: new Date("2023-04-15T10:30:25.000Z") },
    { timestamp: new Date("2023-04-15T10:30:35.000Z") },
    { timestamp: new Date("2023-04-15T10:31:05.000Z") },
    { timestamp: new Date("2023-04-15T10:31:45.000Z") },
    { timestamp: new Date("2023-04-15T10:32:15.000Z") }
  ]);

  const result = await collection.aggregate([
    {
      $group: {
        _id: {
          minute: { $minute: "$timestamp" }
        },
        count: { $count: {} }
      }
    },
    { $sort: { "_id.minute": 1 } }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient
from datetime import datetime

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['events']

    collection.insert_many([
        {'timestamp': datetime(2023, 4, 15, 10, 30, 25)},
        {'timestamp': datetime(2023, 4, 15, 10, 30, 35)},
        {'timestamp': datetime(2023, 4, 15, 10, 31, 5)},
        {'timestamp': datetime(2023, 4, 15, 10, 31, 45)},
        {'timestamp': datetime(2023, 4, 15, 10, 32, 15)}
    ])

    result = list(collection.aggregate([
        {
            '$group': {
                '_id': {
                    'minute': {'$minute': '$timestamp'}
                },
                'count': {'$count': {}}
            }
        },
        {'$sort': {'_id.minute': 1}}
    ]))

    print(result)
    client.close()

example()
```

$mod

연$mod산자는 숫자에 대해 모듈로 연산을 수행하는 산술 연산자입니다. 한 숫자를 다른 숫자로 나눈 나머지 부분을 반환합니다. 이 연산자는 일반적으로 숫자가 홀수인지 짝수인지 확인하거나 항목을 한정된 수의 그룹으로 분산하는 데 사용됩니다.

파라미터

- `expression1`: 배당 표현식입니다.
- `expression2`: 디바이저 표현식입니다.

예제(MongoDB 쉘)

이 예제에서는 연$mod산자를 사용하여 100개 패키지로 배송할 때 남은 위젯 수를 확인하는 방법을 보여줍니다.

샘플 문서 생성

```
db.widgets.insertMany([
  { "_id" : 1, "widget" : "A", "count" : 80372 },
  { "_id" : 2, "widget" : "B", "count" : 409282 },
  { "_id" : 3, "widget" : "C", "count" : 60739 }
])
```

쿼리 예제

```
db.widgets.aggregate([
  { $addFields: { leftOver: { $mod: [ "$count", 100 ] } } }
])
```

출력

```
[
  { "_id" : 1, "widget" : "A", "count" : 80372, "leftOver" : 72 },
  { "_id" : 2, "widget" : "B", "count" : 409282, "leftOver" : 82 },
  { "_id" : 3, "widget" : "C", "count" : 60739, "leftOver" : 39 }
]
```

출력에는의 나머지 부분을 각 문서에 대해 100으로 `count` 나눈 값이 표시되며, 이는 100개 패키지로 배송할 때 남은 위젯 수를 나타냅니다.

코드 예제

$mod 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const widgets = db.collection('widgets');

  await widgets.insertMany([
    { "_id" : 1, "widget" : "A", "count" : 80372 },
    { "_id" : 2, "widget" : "B", "count" : 409282 },
    { "_id" : 3, "widget" : "C", "count" : 60739 }
  ]);

  const result = await widgets.aggregate([
    { $addFields: { leftOver: { $mod: [ "$count", 100 ] } } }
  ]).toArray();

  console.log(result);
  client.close();
}

main();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
widgets = db['widgets']

widgets.insert_many([
    { "_id" : 1, "widget" : "A", "count" : 80372 },
    { "_id" : 2, "widget" : "B", "count" : 409282 },
    { "_id" : 3, "widget" : "C", "count" : 60739 }
])
```

```
result = list(widgets.aggregate([
    { "$addFields": { "leftOver": { "$mod": [ "$count", 100 ] } } }
]))

print(result)
client.close()
```

$month

Amazon DocumentDB의 $month 연산자는 날짜의 월을 1에서 12 사이의 숫자로 반환합니다. 이 연산자는 날짜 필드에서 월 구성 요소를 추출하고 날짜 기반 집계 및 분석을 수행하는 데 유용합니다.

파라미터

- date_expression: 월을 추출하려는 날짜 또는 타임스탬프가 포함된 표현식 또는 필드입니다.

예제(MongoDB 쉘)

다음 예제에서는 $month 연산자를 사용하여 날짜 필드에서 월을 추출하고 데이터를 월별로 그룹화하는 방법을 보여줍니다.

샘플 문서 생성

```
db.sales.insert([
  { product: "abc123", price: 10.99, date: new Date("2022-01-15") },
  { product: "def456", price: 15.50, date: new Date("2022-02-28") },
  { product: "ghi789", price: 8.25, date: new Date("2022-03-10") },
  { product: "jkl012", price: 12.75, date: new Date("2022-04-05") },
  { product: "mno345", price: 18.99, date: new Date("2022-05-20") }
]);
```

쿼리 예제

```
db.sales.aggregate([
  { $group: {
      _id: { month: { $month: "$date" } },
      totalSales: { $sum: "$price" }
    }},
  { $sort: { "_id.month": 1 } }
]);
```

출력

```
[
  { _id: { month: 1 }, totalSales: 10.99 },
  { _id: { month: 2 }, totalSales: 15.5 },
  { _id: { month: 3 }, totalSales: 8.25 },
  { _id: { month: 4 }, totalSales: 12.75 },
  { _id: { month: 5 }, totalSales: 18.99 }
]
```

코드 예제

$month 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function groupSalesByMonth() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('sales');

    const pipeline = [
      {
        $group: {
          _id: { month: { $month: "$date" } },
          totalSales: { $sum: "$price" }
        }
      },
      {
        $sort: { "_id.month": 1 }
      }
    ];

    const results = await collection.aggregate(pipeline).toArray();

    console.dir(results, { depth: null });
```

```
  } finally {
    await client.close();
  }
}

groupSalesByMonth().catch(console.error);
```

Python

```
from pymongo import MongoClient

def group_sales_by_month():

    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client.test
        collection = db.sales

        pipeline = [
            {
                "$group": {
                    "_id": { "$month": "$date" },
                    "totalSales": { "$sum": "$price" }
                }
            },
            {
                "$sort": { "_id": 1 }
            }
        ]

        results = collection.aggregate(pipeline)

        for doc in results:
            print(doc)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
```

```
        client.close()

group_sales_by_month()
```

$multiply

Amazon DocumentDB의 $multiply 연산자는 둘 이상의 필드 또는 표현식 값을 곱하는 데 사용됩니다. 이 연산자는 문서 내의 숫자 필드에 산술 연산을 수행하는 데 특히 유용합니다. $project 및와 같은 집계 파이프라인의 다양한 단계에서 사용하여 새 필드를 $addFields생성하거나 기존 필드를 수정할 수 있습니다.

파라미터

- expression1: 곱할 첫 번째 숫자 표현식입니다.
- expression2: 곱할 두 번째 숫자 표현식입니다.
- [expression3, ...]: (선택 사항) 곱할 추가 숫자 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $multiply를 사용하여를 곱bonus_miles하고 여정base_milesbonus_rate에 신용 카드를 사용한 고객을 계산하는 방법을 보여줍니다.

샘플 문서

```
db.miles.insertMany([
{ "_id": 1, "customer_name": "Arnav Desai", "member_since":
 ISODate("1997-03-01T00:00:00Z"), "base_miles": 2500, "bonus_rate": 1.8, "credit_card":
 true, "trip_cost": 250 },
{ "_id": 2, "customer_name": "Jorge Souza", "member_since":
 ISODate("2004-01-10T00:00:00Z"), "base_miles": 1890, "bonus_rate": 1.4, "credit_card":
 true, "trip_cost": 189 },
{ "_id": 3, "customer_name": "Saanvi Sarkar", "member_since":
 ISODate("1999-11-22T00:00:00Z"), "base_miles": 3250, "bonus_rate": 1.8, "credit_card":
 false, "trip_cost": 325 },
{ "_id": 4, "customer_name": "Paulo Santos", "member_since":
 ISODate("2021-06-19T00:00:00Z"), "base_miles": 2980, "bonus_rate": 1.2, "credit_card":
 true, "trip_cost": 298 },
{ "_id": 5, "customer_name": "Wang Xiulan", "member_since":
 ISODate("1995-12-04T00:00:00Z"), "base_miles": 1350, "bonus_rate": 1.9, "credit_card":
 false, "trip_cost": 135 }
```

```
]);
```

쿼리 예제

```
db.miles.aggregate([
  {
    $match: { credit_card: true }
  },
  {
    $project: {
      customer_name: 1,
      base_miles: 1,
      bonus_rate:1,
      credit_card: 1,
      total_miles: {
        $multiply: ["$base_miles", "$bonus_rate"]
      }
    }
  }
]);
```

출력

```
[
  { _id: 1, customer_name: 'Arnav Desai', base_miles: 12500, bonus_rate: 1.8,
 credit_card: true, total_miles: 22500 },
  { _id: 3, customer_name: 'Saanvi Sarkar',base_miles: 15200, bonus_rate: 1.8,
 credit_card: true, total_miles: 27360 },
  { _id: 4, customer_name: 'Paulo Santos', base_miles: 3400, bonus_rate: 1.1,
 credit_card: true, total_miles: 3740 }
]
```

코드 예제

`$multiply` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function multiplyBonusMiles() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('miles');

  const result = await collection.aggregate([
    { $match: { credit_card: true } },
    {
      $project: {
        customer_name: 1,
        base_miles: 1,
        bonus_rate: 1,
        credit_card: 1,
        total_miles: {
          $multiply: ["$base_miles", "$bonus_rate"]
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

multiplyBonusMiles();
```

Python

```
from pymongo import MongoClient

def multiply_bonus_miles():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['miles']

    result = list(collection.aggregate([
        {'$match': {'credit_card': True}},
        {
            '$project': {
                'customer_name': 1,
```

```
                'base_miles': 1,
                'bonus_rate': 1,
                'credit_card': 1,
                'total_miles': {
                    '$multiply': ['$base_miles', '$bonus_rate']
                }
            }
        }
    ]))

    print(result)
    client.close()

multiply_bonus_miles()
```

$natural

Amazon DocumentDB의 `$natural` 연산자는 문서를 컬렉션에 삽입한 순서인 자연 순서로 정렬하는 데 사용됩니다. 이는 지정된 필드의 값을 기반으로 문서를 정렬하는 기본 정렬 동작과는 대조적입니다.

파라미터

없음

예제(MongoDB 쉘)

다음 예제에서는 `$natural` 연산자를 사용하여 컬렉션의 문서를 자연 순서로 정렬하는 방법을 보여줍니다.

샘플 문서 생성

```
db.people.insertMany([
  { "_id": 1, "name": "María García", "age": 28 },
  { "_id": 2, "name": "Arnav Desai", "age": 32 },
  { "_id": 3, "name": "Li Juan", "age": 25 },
  { "_id": 4, "name": "Carlos Salazar", "age": 41 },
  { "_id": 5, "name": "Sofia Martínez", "age": 35 }
]);
```

쿼리 예제

```
db.people.find({}, { "_id": 1, "name": 1 }).sort({ "$natural": 1 });
```

출력

```
[
  { "_id": 1, "name": "María García" },
  { "_id": 2, "name": "Arnav Desai" },
  { "_id": 3, "name": "Li Juan" },
  { "_id": 4, "name": "Carlos Salazar" },
  { "_id": 5, "name": "Sofia Martínez" }
]
```

쿼리는 모음의 문서를 삽입된 순서인 자연 순서로 정렬합니다.

코드 예제

$natural 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const documents = await collection.find({}, { projection: { _id: 1, name: 1 } })
    .sort({ $natural: 1 })
    .toArray();

  console.log(documents);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['people']

    documents = list(collection.find({}, {'_id': 1, 'name': 1}).sort('$natural', 1))
    print(documents)

    client.close()

example()
```

$ne

$ne 집계 연산자는 두 값을 비교하고 값이 같`true`지 않으면를 반환하고 그렇지 않으면를 반환합니다`false`.

파라미터

- `expression1`: 비교할 첫 번째 값입니다.
- `expression2`: 비교할 두 번째 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 $ne 연산자를 사용하여 상태 변경이 있는 주문을 식별하는 방법을 보여줍니다.

샘플 문서 생성

```
db.orders.insertMany([
  { _id: 1, orderId: "A123", status: "shipped", expectedStatus: "shipped" },
  { _id: 2, orderId: "B456", status: "pending", expectedStatus: "shipped" },
  { _id: 3, orderId: "C789", status: "delivered", expectedStatus: "delivered" }
]);
```

쿼리 예제

```
db.orders.aggregate([
  {
    $project: {
      orderId: 1,
```

```
      status: 1,
      expectedStatus: 1,
      needsAttention: { $ne: ["$status", "$expectedStatus"] }
    }
  }
]);
```

출력

```
[
  { _id: 1, orderId: 'A123', status: 'shipped', expectedStatus: 'shipped',
 needsAttention: false },
  { _id: 2, orderId: 'B456', status: 'pending', expectedStatus: 'shipped',
 needsAttention: true },
  { _id: 3, orderId: 'C789', status: 'delivered', expectedStatus: 'delivered',
 needsAttention: false }
]
```

코드 예제

$ne 집계 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('orders');

  const result = await collection.aggregate([
    {
      $project: {
        orderId: 1,
        status: 1,
        expectedStatus: 1,
        needsAttention: { $ne: ["$status", "$expectedStatus"] }
      }
    }
  ]).toArray();
```

```
  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['orders']

    result = list(collection.aggregate([
        {
            '$project': {
                'orderId': 1,
                'status': 1,
                'expectedStatus': 1,
                'needsAttention': { '$ne': ['$status', '$expectedStatus'] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$not

$not 집계 연산자는 표현식에 대해 논리적 NOT 작업을 수행합니다. 표현식이 로 평가`true`되면 `false`를 반환하고 표현식이 로 평가`false`되면를 반환합니다`true`.

파라미터

- `expression`: 부정할 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $not 연산자를 사용하여 부울 값을 반전하는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { _id: 1, name: "Alice", active: true },
  { _id: 2, name: "Bob", active: false },
  { _id: 3, name: "Charlie", active: true }
]);
```

쿼리 예제

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      active: 1,
      inactive: { $not: ["$active"] }
    }
  }
]);
```

출력

```
[
  { _id: 1, name: 'Alice', active: true, inactive: false },
  { _id: 2, name: 'Bob', active: false, inactive: true },
  { _id: 3, name: 'Charlie', active: true, inactive: false }
]
```

코드 예제

$not 집계 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('users');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
        active: 1,
        inactive: { $not: ["$active"] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['users']

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'active': 1,
                'inactive': { '$not': ['$active'] }
            }
        }
    ]))
```

```
    print(result)
    client.close()

example()
```

$objectToArray

Amazon DocumentDB의 $objectToArray 집계 연산자는 객체(또는 문서)를 배열로 변환합니다. 연산자에 대한 입력은 문서이며 출력은 입력 문서의 각 필드-값 페어에 대한 배열 요소로 구성됩니다. 이 연산자는 특정 필드의 최대값 또는 최소값이 있는 문서를 찾으려는 경우와 같이 문서의 개별 필드를 배열로 사용해야 할 때 유용합니다.

파라미터

- expression: 배열로 변환할 문서 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $objectToArray 운영자를 사용하여 비디오 렌탈 스토어 체인의 최대 인벤토리가 있는 문서를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.videos.insertMany([
  {
    "_id": 1,
    "name": "Live Soft",
    "inventory": {
      "Des Moines": 1000,
      "Ames": 500
    }
  },
  {
    "_id": 2,
    "name": "Top Pilot",
    "inventory": {
      "Mason City": 250,
      "Des Moines": 1000
    }
  },
  {
```

```
    "_id": 3,
    "name": "Romancing the Rock",
    "inventory": {
      "Mason City": 250,
      "Ames": 500
    }
  },
  {
    "_id": 4,
    "name": "Bravemind",
    "inventory": {
      "Mason City": 250,
      "Des Moines": 1000,
      "Ames": 500
    }
  }
]);
```

쿼리 예제

```
db.videos.aggregate([
  {
    $project: {
      name: 1,
      videos: {
        $objectToArray: "$inventory"
      }
    }
  },
  {
    $unwind: "$videos"
  },
  {
    $group: {
      _id: "$name",
      maxInventory: {
        $max: "$videos.v"
      }
    }
  }
]);
```

출력

```
[
  {
    "_id": "Bravemind",
    "maxInventory": 1000
  },
  {
    "_id": "Live Soft",
    "maxInventory": 1000
  },
  {
    "_id": "Romancing the Rock",
    "maxInventory": 500
  },
  {
    "_id": "Top Pilot",
    "maxInventory": 1000
  }
]
```

코드 예제

$objectToArray 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findMaxInventory() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const videos = db.collection('videos');

  const result = await videos.aggregate([
    {
      $project: {
        name: 1,
        videos: {
          $objectToArray: "$inventory"
        }
      }
    },
```

```
    {
      $unwind: "$videos"
    },
    {
      $group: {
        _id: "$name",
        maxInventory: {
          $max: "$videos.v"
        }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

findMaxInventory();
```

Python

```
from pymongo import MongoClient

def find_max_inventory():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    videos = db['videos']

    result = list(videos.aggregate([
        {
            '$project': {
                'name': 1,
                'videos': {
                    '$objectToArray': '$inventory'
                }
            }
        },
        {
            '$unwind': '$videos'
        },
        {
```

```
            '$group': {
                '_id': '$name',
                'maxInventory': {
                    '$max': '$videos.v'
                }
            }
        }
    ]))

    print(result)
    client.close()

find_max_inventory()
```

$or

$or 집계 연산자는 여러 표현식을 평가하고 하나 이상의 표현식이 로 평가true되면를 반환합니다true. 모든 표현식이 인 false 경우에만 반환됩니다false.

파라미터

- expressions: 평가할 표현식의 배열입니다.

예제(MongoDB 쉘)

다음 예제에서는 $or 연산자를 사용하여 제품이 여러 기준 중 하나를 충족하는지 확인하는 방법을 보여줍니다.

샘플 문서 생성

```
db.items.insertMany([
  { _id: 1, name: "Widget", price: 150, onSale: false },
  { _id: 2, name: "Gadget", price: 45, onSale: false },
  { _id: 3, name: "Tool", price: 200, onSale: true }
]);
```

쿼리 예제

```
db.items.aggregate([
  {
    $project: {
```

```
      name: 1,
      price: 1,
      onSale: 1,
      goodDeal: {
        $or: [
          { $lt: ["$price", 50] },
          { $eq: ["$onSale", true] }
        ]
      }
    }
  }
]);
```

출력

```
[
  { _id: 1, name: 'Widget', price: 150, onSale: false, goodDeal: false },
  { _id: 2, name: 'Gadget', price: 45, onSale: false, goodDeal: true },
  { _id: 3, name: 'Tool', price: 200, onSale: true, goodDeal: true }
]
```

코드 예제

$or 집계 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('items');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
        price: 1,
        onSale: 1,
        goodDeal: {
```

```
          $or: [
            { $lt: ["$price", 50] },
            { $eq: ["$onSale", true] }
          ]
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['items']

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'price': 1,
                'onSale': 1,
                'goodDeal': {
                    '$or': [
                        { '$lt': ['$price', 50] },
                        { '$eq': ['$onSale', True] }
                    ]
                }
            }
        }
    ]))

    print(result)
```

```
    client.close()

example()
```

$out

Amazon DocumentDB의 $out 연산자는 집계 파이프라인의 결과를 지정된 컬렉션에 쓰는 데 사용됩니다.

$out는 파이프라인의 마지막 단계여야 합니다.

파라미터

- output_collection: 집계 결과를 쓸 출력 컬렉션의 이름입니다.

참고: 컬렉션이 이미 있는 경우 집계 단계의 결과로 대체됩니다.

예제(MongoDB 쉘)

다음 예제에서는 Amazon DocumentDB의 $out 연산자를 사용하여 집계 파이프라인의 결과를 새 컬렉션에 쓰는 방법을 보여줍니다.

샘플 문서 생성

```
db.products.insertMany([
  { _id: 1, name: "Wireless Headphones", category: "Electronics", price: 100.0 },
  { _id: 2, name: "Smartphone", category: "Electronics", price: 200.0 },
  { _id: 3, name: "JavaScript Guide", category: "Books", price: 50.0 },
  { _id: 4, name: "Database Design Handbook", category: "Books", price: 75.0 }
]);
```

쿼리 예제

```
db.products.aggregate([
  { $group: { _id: "$category", totalPrice: { $sum: "$price" } } },
  { $out: "product_categories" }
])
```

출력

없음(결과가 출력 컬렉션에 기록됨).

집계 파이프라인은 제품을 범주별로 그룹화하고 각 범주에 대한 항목의 총 가격을 계산합니다. 연$out산자는 "product_categories"라는 새 컬렉션에 결과를 기록합니다.

출력 컬렉션의 결과를 보려면:

```
db.product_categories.find()
[
{ "_id" : "Books", "totalPrice" : 125 },
{ "_id" : "Electronics", "totalPrice" : 300 }
]
```

코드 예제

$out 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function demo_out_operator() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const products = db.collection('products');

  // Execute aggregation with $out - results are stored in 'product_categories'
 collection
  await products.aggregate([
    { $group: { _id: "$category", totalPrice: { $sum: "$price" } } },
    { $out: "product_categories" }
  ]).toArray();

  // Retrieve the results from the output collection (limited to 20 records)
  const productCategories = db.collection('product_categories');
  const results = await productCategories.find({}).limit(20).toArray();

  console.log('Results stored in product_categories collection:', results);
  await client.close();
}

demo_out_operator();
```

Python

```
from pymongo import MongoClient

def demo_out_operator():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    products = db['products']

    # Execute aggregation with $out - results are stored in 'product_categories'
 collection
    list(products.aggregate([
        { '$group': { '_id': '$category', 'totalPrice': { '$sum': '$price' } } },
        { '$out': 'product_categories' }
    ]))

    # Retrieve the results from the output collection (limited to 20 records)
    product_categories = db['product_categories']
    results = list(product_categories.find({}).limit(20))

    print('Results stored in product_categories collection:', results)
    client.close()

demo_out_operator()
```

$백분위수

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 `$percentile` 연산자는 숫자 데이터에 대해 지정된 백분위수 값을 계산합니다. 누적기는 집계 파이프라인의 `$group` 단계에서 그룹 내 문서 전반의 백분위수 값을 계산합니다. 표현식으로 숫자 배열의 백분위수를 계산합니다.

파라미터

- `input`: 숫자 값 또는 숫자 값 배열로 확인되는 표현식입니다.
- p: 0과 1 사이의 백분위수 값 배열로, 여기서 각 값은 계산할 백분위수를 나타냅니다. 예를 들어는 25번째, 50번째 및 75번째 백분위수를 `[0.25, 0.5, 0.75]` 계산합니다.
- `method`: 계산 방법을 지정하는 문자열입니다. 현재 `"approximate"`만 지원됩니다.

동작

"approximate" 메서드는 t-digest 알고리즘을 사용하여 대략적인 백분위수 기반 지표를 계산합니다. 데이터 세트가 작으면 고유한 백분위수 값이 동일한 결과로 확인될 수 있습니다. 예를 들어 그룹당 값이 5개뿐인 경우 p90과 p99는 모두 그룹의 최대값을 반환할 수 있습니다. 데이터 포인트 수가 증가함에 따라 정밀도가 향상됩니다.

예제(MongoDB 쉘)

다음 예제에서는 $percentile 연산자를 사용하여 클래스당 점수의 25번째 및 75번째 백분위수를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.students.insertMany([
  { class: "A", score: 72 },
  { class: "A", score: 85 },
  { class: "A", score: 90 },
  { class: "A", score: 68 },
  { class: "A", score: 95 },
  { class: "B", score: 80 },
  { class: "B", score: 75 },
  { class: "B", score: 92 },
  { class: "B", score: 88 },
  { class: "B", score: 70 }
]);
```

쿼리 예제

```
db.students.aggregate([
  { $group: {
      _id: "$class",
      percentiles: { $percentile: { input: "$score", p: [0.25, 0.75], method:
 "approximate" } }
    }}
]);
```

출력

```
[
  { "_id": "A", "percentiles": [72, 90] },
```

```
  { "_id": "B", "percentiles": [75, 88] }
]
```

표현식 사용 예제(MongoDB Shell)

연$percentile산자를 $project 단계 내의 표현식으로 사용하여 배열 필드의 백분위수를 계산할 수도 있습니다.

샘플 문서 생성

```
db.surveys.insertMany([
  { _id: 1, responses: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20] },
  { _id: 2, responses: [1, 3, 5, 7, 9, 11, 13, 15, 17, 19] }
]);
```

쿼리 예제

```
db.surveys.aggregate([
  { $project: {
      quartiles: { $percentile: { input: "$responses", p: [0.25, 0.5, 0.75], method:
 "approximate" } }
    }}
]);
```

출력

```
[
  { "_id": 1, "quartiles": [6, 10, 16] },
  { "_id": 2, "quartiles": [5, 9, 15] }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$percentile산자는 $setWindowFields 스테이지에서 창 연산자로도 사용할 수 있습니다. 이 컨텍스트에서는 각 창의 문서에 대해 지정된 백분위수 값을 계산합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

> **Note**
>
> 에서 창 연산자로 사용할 경우 `$setWindowFields$percentile`는 100MB의 중간 데이터로 제한됩니다. 이 제한을 초과하는 작업은 오류를 반환합니다.

샘플 문서 생성

```
db.sensorData.insertMany([
  { _id: 1, sensor: "A", reading: 72 },
  { _id: 2, sensor: "A", reading: 85 },
  { _id: 3, sensor: "A", reading: 90 },
  { _id: 4, sensor: "A", reading: 68 },
  { _id: 5, sensor: "A", reading: 95 },
  { _id: 6, sensor: "B", reading: 80 },
  { _id: 7, sensor: "B", reading: 75 },
  { _id: 8, sensor: "B", reading: 92 },
  { _id: 9, sensor: "B", reading: 88 },
  { _id: 10, sensor: "B", reading: 70 }
]);
```

쿼리 예제

다음 예제에서는 문서를 기준으로 분할`sensor`하고 각 파티션의 모든 문서에 `reading` 대해의 25번째 및 75번째 백분위수를 계산합니다.

```
db.sensorData.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$sensor",
      output: {
        readingPercentiles: {
          $percentile: { input: "$reading", p: [0.25, 0.75], method: "approximate" },
          window: { documents: ["unbounded", "unbounded"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "sensor": "A", "reading": 72, "readingPercentiles": [72, 90] },
  { "_id": 2, "sensor": "A", "reading": 85, "readingPercentiles": [72, 90] },
  { "_id": 3, "sensor": "A", "reading": 90, "readingPercentiles": [72, 90] },
  { "_id": 4, "sensor": "A", "reading": 68, "readingPercentiles": [72, 90] },
  { "_id": 5, "sensor": "A", "reading": 95, "readingPercentiles": [72, 90] },
  { "_id": 6, "sensor": "B", "reading": 80, "readingPercentiles": [75, 88] },
  { "_id": 7, "sensor": "B", "reading": 75, "readingPercentiles": [75, 88] },
  { "_id": 8, "sensor": "B", "reading": 92, "readingPercentiles": [75, 88] },
  { "_id": 9, "sensor": "B", "reading": 88, "readingPercentiles": [75, 88] },
  { "_id": 10, "sensor": "B", "reading": 70, "readingPercentiles": [75, 88] }
]
```

각 문서는 파티션`readingPercentiles`의 모든 문서에 `reading` 걸쳐의 약 25번째 및 75번째 백분위수인 로 보강됩니다.

코드 예제

`$percentile` 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누적기 사용량(의 `$group`), 표현식 사용량(의 `$project`) 및 창 연산자 사용량(의)을 보여줍니다. `$setWindowFields`

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();
    const db = client.db('test');

    // Accumulator usage: percentiles across grouped documents
    const students = db.collection('students');
    await students.insertMany([
      { class: "A", score: 72 },
      { class: "A", score: 85 },
      { class: "A", score: 90 },
      { class: "A", score: 68 },
```

```
    { class: "A", score: 95 },
    { class: "B", score: 80 },
    { class: "B", score: 75 },
    { class: "B", score: 92 },
    { class: "B", score: 88 },
    { class: "B", score: 70 }
  ]);
  const accumulatorResult = await students.aggregate([
    { $group: {
        _id: "$class",
        percentiles: { $percentile: { input: "$score", p: [0.25, 0.75], method:
"approximate" } }
      }}
  ]).toArray();
  console.log('Accumulator result:', accumulatorResult);

  // Expression usage: percentiles of an array field
  const surveys = db.collection('surveys');
  await surveys.insertMany([
    { _id: 1, responses: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20] },
    { _id: 2, responses: [1, 3, 5, 7, 9, 11, 13, 15, 17, 19] }
  ]);
  const expressionResult = await surveys.aggregate([
    { $project: {
        quartiles: { $percentile: { input: "$responses", p: [0.25, 0.5, 0.75],
method: "approximate" } }
      }}
  ]).toArray();
  console.log('Expression result:', expressionResult);

  // Window operator usage: percentiles across each partition
  const sensorData = db.collection('sensorData');
  await sensorData.insertMany([
    { _id: 1, sensor: "A", reading: 72 },
    { _id: 2, sensor: "A", reading: 85 },
    { _id: 3, sensor: "A", reading: 90 },
    { _id: 4, sensor: "A", reading: 68 },
    { _id: 5, sensor: "A", reading: 95 },
    { _id: 6, sensor: "B", reading: 80 },
    { _id: 7, sensor: "B", reading: 75 },
    { _id: 8, sensor: "B", reading: 92 },
    { _id: 9, sensor: "B", reading: 88 },
    { _id: 10, sensor: "B", reading: 70 }
  ]);
```

```
    const windowResult = await sensorData.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$sensor",
          output: {
            readingPercentiles: {
              $percentile: { input: "$reading", p: [0.25, 0.75], method:
 "approximate" },
              window: { documents: ["unbounded", "unbounded"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']

        # Accumulator usage: percentiles across grouped documents
        students = db['students']
        students.insert_many([
            { 'class': 'A', 'score': 72 },
            { 'class': 'A', 'score': 85 },
            { 'class': 'A', 'score': 90 },
            { 'class': 'A', 'score': 68 },
            { 'class': 'A', 'score': 95 },
```

```
            { 'class': 'B', 'score': 80 },
            { 'class': 'B', 'score': 75 },
            { 'class': 'B', 'score': 92 },
            { 'class': 'B', 'score': 88 },
            { 'class': 'B', 'score': 70 }
        ])
        accumulator_result = list(students.aggregate([
            { '$group': {
                '_id': '$class',
                'percentiles': { '$percentile': { 'input': '$score', 'p': [0.25,
0.75], 'method': 'approximate' } }
            }}
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: percentiles of an array field
        surveys = db['surveys']
        surveys.insert_many([
            { '_id': 1, 'responses': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20] },
            { '_id': 2, 'responses': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19] }
        ])
        expression_result = list(surveys.aggregate([
            { '$project': {
                'quartiles': { '$percentile': { 'input': '$responses', 'p': [0.25,
0.5, 0.75], 'method': 'approximate' } }
            }}
        ]))
        print('Expression result:', expression_result)

        # Window operator usage: percentiles across each partition
        sensor_data = db['sensorData']
        sensor_data.insert_many([
            { '_id': 1, 'sensor': 'A', 'reading': 72 },
            { '_id': 2, 'sensor': 'A', 'reading': 85 },
            { '_id': 3, 'sensor': 'A', 'reading': 90 },
            { '_id': 4, 'sensor': 'A', 'reading': 68 },
            { '_id': 5, 'sensor': 'A', 'reading': 95 },
            { '_id': 6, 'sensor': 'B', 'reading': 80 },
            { '_id': 7, 'sensor': 'B', 'reading': 75 },
            { '_id': 8, 'sensor': 'B', 'reading': 92 },
            { '_id': 9, 'sensor': 'B', 'reading': 88 },
            { '_id': 10, 'sensor': 'B', 'reading': 70 }
        ])
        window_result = list(sensor_data.aggregate([
```

```
            {
                '$setWindowFields': {
                    'partitionBy': '$sensor',
                    'output': {
                        'readingPercentiles': {
                            '$percentile': { 'input': '$reading', 'p': [0.25, 0.75],
 'method': 'approximate' },
                            'window': { 'documents': ['unbounded', 'unbounded'] }
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()

example()
```

$pow

8.0에 도입

Amazon DocumentDB의 $pow 연산자를 사용하면 숫자를 출력으로 높일 수 있습니다. 이는 집계 파이프라인 내에서 지수 계산을 수행하는 데 유용할 수 있습니다.

파라미터

- <number> (필수): 전원으로 높일 숫자입니다.
- <exponent> (필수): 숫자를 늘려야 하는 힘입니다.

예제(MongoDB 쉘)

다음 예제에서는 $pow 연산자를 사용하여 숫자의 제곱을 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.numbers.insertMany([
  { "_id": 1, "value": 2 },
  { "_id": 2, "value": 3 },
  { "_id": 3, "value": 4 }
```

```
]);
```

쿼리 예제

```
db.numbers.aggregate([
  { $addFields: { "square": { $pow: ["$value", 2] } } }
])
```

출력

```
[
  { "_id": 1, "value": 2, "square": 4 },
  { "_id": 2, "value": 3, "square": 9 },
  { "_id": 3, "value": 4, "square": 16 }
]
```

코드 예제

$pow 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

다음은 Node.js 애플리케이션에서 $pow 연산자를 사용하는 예입니다.

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $addFields: { "square": { $pow: ["$value", 2] } } }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

다음은 Python 애플리케이션에서 $pow 연산자를 사용하는 예입니다.

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['numbers']

    result = list(collection.aggregate([
        { "$addFields": { "square": { "$pow": ["$value", 2] } } }
    ]))

    print(result)
    client.close()

example()
```

$push

$push 집계 연산자는 각 그룹의 지정된 표현식에서 모든 값의 배열을 반환합니다. 일반적으로 $group 스테이지 내에서 값을 배열에 누적하는 데 사용됩니다.

파라미터

- expression: 그룹의 각 문서에 대해 평가할 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $push 연산자를 사용하여 각 범주에 대한 모든 제품 이름을 수집하는 방법을 보여줍니다.

샘플 문서 생성

```
db.sales.insertMany([
  { _id: 1, category: "Electronics", product: "Laptop", amount: 1200 },
  { _id: 2, category: "Electronics", product: "Mouse", amount: 25 },
  { _id: 3, category: "Furniture", product: "Desk", amount: 350 },
```

```
  { _id: 4, category: "Furniture", product: "Chair", amount: 150 },
  { _id: 5, category: "Electronics", product: "Keyboard", amount: 75 }
]);
```

쿼리 예제

```
db.sales.aggregate([
  {
    $group: {
      _id: "$category",
      products: { $push: "$product" }
    }
  }
]);
```

출력

```
[
  { _id: 'Furniture', products: [ 'Desk', 'Chair' ] },
  { _id: 'Electronics', products: [ 'Laptop', 'Mouse', 'Keyboard' ] }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$push산자는 $setWindowFields 스테이지에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 문서에 대해 지정된 표현식의 모든 값의 배열을 정렬 순서로 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

> **Note**
>
> 에서 창 연산자로 사용할 경우 $setWindowFields$push는 100MB의 중간 데이터로 제한됩니다. 이 제한을 초과하는 작업은 오류를 반환합니다.

샘플 문서 생성

```
db.purchases.insertMany([
  { _id: 1, category: "Electronics", day: 1, product: "Laptop" },
  { _id: 2, category: "Electronics", day: 2, product: "Mouse" },
```

```
  { _id: 3, category: "Electronics", day: 3, product: "Keyboard" },
  { _id: 4, category: "Furniture", day: 1, product: "Desk" },
  { _id: 5, category: "Furniture", day: 2, product: "Chair" }
]);
```

쿼리 예제

다음 예제에서는를 기준으로 문서를 분할하고category,를 기준으로 각 파티션을 정렬하고day, 파티션 시작부터 현재 문서를 통해 실행 중인 product 값 목록을 반환합니다.

```
db.purchases.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$category",
      sortBy: { day: 1 },
      output: {
        productsSoFar: {
          $push: "$product",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "category": "Electronics", "day": 1, "product": "Laptop",
 "productsSoFar": [ "Laptop" ] },
  { "_id": 2, "category": "Electronics", "day": 2, "product": "Mouse", "productsSoFar":
 [ "Laptop", "Mouse" ] },
  { "_id": 3, "category": "Electronics", "day": 3, "product": "Keyboard",
 "productsSoFar": [ "Laptop", "Mouse", "Keyboard" ] },
  { "_id": 4, "category": "Furniture", "day": 1, "product": "Desk", "productsSoFar":
 [ "Desk" ] },
  { "_id": 5, "category": "Furniture", "day": 2, "product": "Chair", "productsSoFar":
 [ "Desk", "Chair" ] }
]
```

각 문서는 처음부터 현재 문서까지 파티션 내 모든 product 값의 배열productsSoFar인 로 정렬 순서로 보강됩니다.

코드 예제

$push 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누산기 사용량()과 창 연산자 사용량($group)을 모두 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');

  // Accumulator usage: collect all values per group
  const sales = db.collection('sales');
  await sales.insertMany([
    { _id: 1, category: "Electronics", product: "Laptop", amount: 1200 },
    { _id: 2, category: "Electronics", product: "Mouse", amount: 25 },
    { _id: 3, category: "Furniture", product: "Desk", amount: 350 },
    { _id: 4, category: "Furniture", product: "Chair", amount: 150 },
    { _id: 5, category: "Electronics", product: "Keyboard", amount: 75 }
  ]);
  const accumulatorResult = await sales.aggregate([
    {
      $group: {
        _id: "$category",
        products: { $push: "$product" }
      }
    }
  ]).toArray();
  console.log('Accumulator result:', accumulatorResult);

  // Window operator usage: running list of values within each partition
  const purchases = db.collection('purchases');
  await purchases.insertMany([
    { _id: 1, category: "Electronics", day: 1, product: "Laptop" },
    { _id: 2, category: "Electronics", day: 2, product: "Mouse" },
    { _id: 3, category: "Electronics", day: 3, product: "Keyboard" },
    { _id: 4, category: "Furniture", day: 1, product: "Desk" },
    { _id: 5, category: "Furniture", day: 2, product: "Chair" }
  ]);
  const windowResult = await purchases.aggregate([
```

```
    {
      $setWindowFields: {
        partitionBy: "$category",
        sortBy: { day: 1 },
        output: {
          productsSoFar: {
            $push: "$product",
            window: { documents: ["unbounded", "current"] }
          }
        }
      }
    }
  ]).toArray();
  console.log('Window result:', windowResult);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']

    # Accumulator usage: collect all values per group
    sales = db['sales']
    sales.insert_many([
        { '_id': 1, 'category': 'Electronics', 'product': 'Laptop', 'amount':
 1200 },
        { '_id': 2, 'category': 'Electronics', 'product': 'Mouse', 'amount': 25 },
        { '_id': 3, 'category': 'Furniture', 'product': 'Desk', 'amount': 350 },
        { '_id': 4, 'category': 'Furniture', 'product': 'Chair', 'amount': 150 },
        { '_id': 5, 'category': 'Electronics', 'product': 'Keyboard', 'amount': 75 }
    ])
    accumulator_result = list(sales.aggregate([
        {
            '$group': {
```

```
                '_id': '$category',
                'products': { '$push': '$product' }
            }
        }
    ]))
    print('Accumulator result:', accumulator_result)

    # Window operator usage: running list of values within each partition
    purchases = db['purchases']
    purchases.insert_many([
        { '_id': 1, 'category': 'Electronics', 'day': 1, 'product': 'Laptop' },
        { '_id': 2, 'category': 'Electronics', 'day': 2, 'product': 'Mouse' },
        { '_id': 3, 'category': 'Electronics', 'day': 3, 'product': 'Keyboard' },
        { '_id': 4, 'category': 'Furniture', 'day': 1, 'product': 'Desk' },
        { '_id': 5, 'category': 'Furniture', 'day': 2, 'product': 'Chair' }
    ])
    window_result = list(purchases.aggregate([
        {
            '$setWindowFields': {
                'partitionBy': '$category',
                'sortBy': { 'day': 1 },
                'output': {
                    'productsSoFar': {
                        '$push': '$product',
                        'window': { 'documents': ['unbounded', 'current'] }
                    }
                }
            }
        }
    ]))
    print('Window result:', window_result)

    client.close()

example()
```

$project

Amazon DocumentDB의 $project 연산자를 사용하면 출력 문서에서 필드를 선택적으로 포함하거나 제외하고, 값을 다음 파이프라인 단계에 전달하고, 입력 문서 값에서 새 필드를 계산할 수 있습니다.

파라미터

- `field`: 출력 문서에서 포함하거나 제외할 필드로, 필드 경로(예: "a.b.c")일 수 있습니다.
- `1` 또는 `true`: 출력에 필드를 포함합니다.
- `0` 또는 `false`: 출력에서 필드를 제외합니다.

예제(MongoDB 쉘)

다음 예제에서는 학생 컬렉션에서 `$project` 연산자의 사용을 보여줍니다.

샘플 문서 생성

```
db.students.insertMany([
  { "_id": 1, "name": "Alejandro Rosalez", "math": 85, "science": 92, "grade": "A" },
  { "_id": 2, "name": "Carlos Salazar", "math": 78, "science": 84, "grade": "B" },
  { "_id": 3, "name": "Nikhil Jayashankar", "math": 95, "science": 89, "grade": "A" },
  { "_id": 4, "name": "Shirley Rodriguez", "math": 72, "science": 76, "grade": "B" }
  ]);
```

이 쿼리에는 출력에 `name` 및 `math` 필드만 포함됩니다. 명시적으로 제외되지 않는 한 `_id` 필드는 기본적으로 포함됩니다.

```
db.students.aggregate([
  { $project: { "name": 1, "math": 1 } }
])
```

출력

```
{ _id: 1, name: "Alejandro Rosalez", math: 85 }
{ _id: 2, name: "Carlos Salazar", math: 78 }
{ _id: 3, name: "Nikhil Jayashankar", math: 95 }
{ _id: 4, name: "Shirley Rodriguez", math: 72 }
```

이 쿼리는 출력에서 `grade` 및 `_id` 필드를 제외하여 다른 모든 필드(`name`, `math`, `science`)를 표시합니다.

```
db.students.aggregate([
  { $project: { "grade": 0, "_id": 0 } }
])
```

출력

```
{ name: "Alejandro Rosalez", math: 85, science: 92 }
{ name: "Carlos Salazar", math: 78, science: 84 }
{ name: "Nikhil Jayashankar", math: 95, science: 89 }
{ name: "Shirley Rodriguez", math: 72, science: 76 }
```

코드 예제

$project 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('students');

  const result = await collection.aggregate([
    { $project: { "name": 1, "math": 1 } }
  ]).toArray();
  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['students']

    result = list(collection.aggregate([
```

```
        { '$project': { 'name': 1, 'math': 1 } }
    ]))
    print(result)

    client.close()

example()
```

$radiansToDegrees

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 `$radiansToDegrees` 연산자는 값을 라디안에서 도로 변환합니다.

파라미터

- `expression`: 라디안 단위의 숫자 값으로 확인되는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$radiansToDegrees` 연산자를 사용하여 라디안 값을 도로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.angles.insertMany([
  { "_id": 1, "radians": 0 },
  { "_id": 2, "radians": 1.5707963267948966 },
  { "_id": 3, "radians": 3.141592653589793 },
  { "_id": 4, "radians": 6.283185307179586 }
]);
```

쿼리 예제

```
db.angles.aggregate([
  { $project: {
    "degrees": { $radiansToDegrees: "$radians" }
  }}
]);
```

출력

```
[
  { "_id": 1, "degrees": 0 },
  { "_id": 2, "degrees": 90 },
  { "_id": 3, "degrees": 180 },
  { "_id": 4, "degrees": 360 }
]
```

코드 예제

$radiansToDegrees 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('angles');

  const result = await collection.aggregate([
    { $project: {
      "degrees": { $radiansToDegrees: "$radians" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
```

```
    collection = db['angles']

    result = list(collection.aggregate([
        { '$project': {
            'degrees': { '$radiansToDegrees': '$radians' }
        }}
    ]))

    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$rand

버전 8.0의 새로운 기능

Amazon DocumentDB의 $rand 연산자는 0에서 1 사이의 난수를 생성하는 데 사용됩니다.

파라미터

없음

예제(MongoDB 쉘)

다음 예제에서는 $rand 연산자를 사용하여 temp 컬렉션에서 두 문서를 무작위로 선택하는 방법을 보여줍니다.

샘플 문서 생성

```
db.items.insertMany([
  { "name": "pencil", "quantity": 110 },
  { "name": "pen", "quantity": 159 }
])
```

쿼리 예제

```
db.items.aggregate([
  {
    $project: {
      randomValue: { $rand: {} }
```

```
    }
  }
])
```

출력

```
[
  {
    _id: ObjectId('6924a5edd66dcae121d29517'),
    randomValue: 0.8615243955294392
  },
  {
    _id: ObjectId('6924a5edd66dcae121d29518'),
    randomValue: 0.22815483022099903
  }
]
```

코드 예제

$rand 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('items');

  const result = await collection.aggregate([
    {
      $project: {
        randomValue: { $rand: {} }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
```

```
}

example();
```

Python

```
from pymongo import MongoClient

def example():
  client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
  db = client['test']
  collection = db['items']

  result = list(collection.aggregate([
      {
        "$project": {
          "randomValue": { "$rand": {} }
        }
      }
  ]))

  print(result)

  client.close()

example()
```

$range

Amazon DocumentDB의 $range 집계 연산자는 지정된 범위 내에서 연속된 숫자 배열을 생성하는 데 사용됩니다. 이 연산자는 다음 예제와 같이 레이스에서 보조 스테이션의 마일 마커와 같은 숫자 시퀀스를 생성하는 데 특히 유용합니다.

파라미터

- start: 범위의 시작 값입니다.
- end: 범위의 종료 값입니다.
- step: (선택 사항) 범위를 생성할 때 사용할 단계 값입니다. 제공되지 않은 경우 기본 단계 값은 1입니다.

예제(MongoDB 쉘)

이 예제에서는 $range 운영자를 사용하여 자전거 레이스의 수도국에 대한 마일 마커를 생성합니다.

샘플 문서 생성

```
db.races.insertMany([
  { _id: 0, race: "STP", distance: 206 },
  { _id: 1, race: "RSVP", distance: 160 },
  { _id: 2, race: "Chilly Hilly", distance: 33 },
  { _id: 3, race: "Flying Wheels", distance: 100 }
]);
```

쿼리 예제

```
db.races.aggregate([
  {
    $project: {
      race: 1,
      "waterStations": { $range: [20, "$distance", 20] }
    }
  }
]);
```

출력

```
[
  {
    _id: 0,
    race: 'STP',
    waterStations: [
       20,  40,  60,  80,
      100, 120, 140, 160,
      180, 200
    ]
  },
  {
    _id: 1,
    race: 'RSVP',
    waterStations: [
       20,  40,  60, 80,
      100, 120, 140
```

```
    ]
  },
  { _id: 2, race: 'Chilly Hilly', waterStations: [ 20 ] },
  { _id: 3, race: 'Flying Wheels', waterStations: [ 20, 40, 60, 80 ] }
]
```

코드 예제

$range 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('races');

    const pipeline = [
      {
        $project: {
          race: 1,
          waterStations: { $range: [20, "$distance", 20] }
        }
      }
    ];

    const results = await collection.aggregate(pipeline).toArray();

    console.dir(results, { depth: null });

  } finally {
    await client.close();
  }
}

example().catch(console.error);
```

Python

```
from pymongo import MongoClient

def example():

  client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

  try:
      db = client.test
      collection = db.races

      pipeline = [
          {
              "$project": {
                  "race": 1,
                  "waterStations": { "$range": [20, "$distance", 20] }
              }
          }
      ]

      results = collection.aggregate(pipeline)

      for doc in results:
          print(doc)

  except Exception as e:
      print(f"An error occurred: {e}")

  finally:
      client.close()

example()
```

$rank

버전 8.0.2에서 새로 추가되었습니다.

Amazon DocumentDB의 $rank 창 연산자는 $setWindowFields 단계 sortBy 순서에 따라 파티션 내의 각 문서의 위치를 반환합니다. sortBy 값이 동일한 문서는 동일한 순위를 받으며, 다음 고유 값

은 중간 위치를 건너뛰어 갭(예: 1, 2, 2, 4)을 남깁니다. $setWindowFields 스테이지의 output 필드 아래에 지정됩니다.

$setWindowFields 스테이지에서만 사용되는 창 연산자입니다.

파라미터

- $rank는 빈 문서를 {} 값으로 사용합니다.
- window 프레임은 허용되지 않습니다.
- $setWindowFields 스테이지에 정확히 하나의 sortBy 키가 필요합니다.

예제(MongoDB 쉘)

다음 예제에서는를 기준으로 문서를 분할하고game,를 기준으로 각 파티션score을 내림차순으로 정렬하며, 각 플레이어에게 순위를 할당합니다.

샘플 문서 생성

```
db.players.insertMany([
  { _id: 1, game: "A", player: "Alice", score: 90 },
  { _id: 2, game: "A", player: "Bob", score: 85 },
  { _id: 3, game: "A", player: "Carol", score: 85 },
  { _id: 4, game: "A", player: "Dave", score: 70 },
  { _id: 5, game: "B", player: "Erin", score: 95 },
  { _id: 6, game: "B", player: "Frank", score: 80 }
]);
```

쿼리 예제

```
db.players.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$game",
      sortBy: { score: -1 },
      output: {
        rank: { $rank: {} }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "game": "A", "player": "Alice", "score": 90, "rank": 1 },
  { "_id": 2, "game": "A", "player": "Bob", "score": 85, "rank": 2 },
  { "_id": 3, "game": "A", "player": "Carol", "score": 85, "rank": 2 },
  { "_id": 4, "game": "A", "player": "Dave", "score": 70, "rank": 4 },
  { "_id": 5, "game": "B", "player": "Erin", "score": 95, "rank": 1 },
  { "_id": 6, "game": "B", "player": "Frank", "score": 80, "rank": 2 }
]
```

각 game 파티션 내에서 플레이어는 내림차순으로 순위를 매깁니다score. Bob과 Carol은에 묶85여 순위를 공유2하므로 다음 순위는 Dave4의 로 건너뜁니다.

코드 예제

$setWindowFields 단계에서 $rank 연산자를 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const players = db.collection('players');

    await players.insertMany([
      { _id: 1, game: "A", player: "Alice", score: 90 },
      { _id: 2, game: "A", player: "Bob", score: 85 },
      { _id: 3, game: "A", player: "Carol", score: 85 },
      { _id: 4, game: "A", player: "Dave", score: 70 },
      { _id: 5, game: "B", player: "Erin", score: 95 },
      { _id: 6, game: "B", player: "Frank", score: 80 }
    ]);

    const result = await players.aggregate([
```

```
      {
        $setWindowFields: {
          partitionBy: "$game",
          sortBy: { score: -1 },
          output: {
            rank: { $rank: {} }
          }
        }
      }
    ]).toArray();

    console.log(result);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']
        players = db['players']

        players.insert_many([
            { '_id': 1, 'game': 'A', 'player': 'Alice', 'score': 90 },
            { '_id': 2, 'game': 'A', 'player': 'Bob', 'score': 85 },
            { '_id': 3, 'game': 'A', 'player': 'Carol', 'score': 85 },
            { '_id': 4, 'game': 'A', 'player': 'Dave', 'score': 70 },
            { '_id': 5, 'game': 'B', 'player': 'Erin', 'score': 95 },
            { '_id': 6, 'game': 'B', 'player': 'Frank', 'score': 80 }
        ])

        result = list(players.aggregate([
            {
```

```
                '$setWindowFields': {
                    'partitionBy': '$game',
                    'sortBy': { 'score': -1 },
                    'output': {
                        'rank': { '$rank': {} }
                    }
                }
            }
        ]))

        print(result)
    finally:
        client.close()

example()
```

$redact

Amazon DocumentDB의 $redact 집계 단계는 지정된 필드의 값을 기반으로 출력 문서에서 콘텐츠를 선택적으로 포함하거나 제외하는 데 사용됩니다. 이는 액세스 수준 또는 사용자 권한에 따라 민감한 데이터의 가시성을 제어해야 하는 시나리오에서 특히 유용합니다.

파라미터

- $cond: 문서의 각 필드에 $$DESCEND 대해 $$PRUNE, 또는 $$KEEP로 평가되는 표현식입니다.
- $$KEEP: 출력 문서에 현재 필드를 유지합니다.
- $$PRUNE: 출력 문서에서 현재 필드를 제거합니다.
- $$DESCEND: 객체 또는 배열인 현재 필드에 $redact 스테이지를 반복적으로 적용합니다.

예제(MongoDB 쉘)

이 예제에서는 $redact 스테이지를 사용하여 상태를 기준으로 주문을 필터링하고 특정 상태 값이 있는 주문만 표시합니다.

샘플 문서 생성

```
db.orders.insert([
  { "_id": 1, "status": "shipped", "customer": "Carlos Salazar", "total": 150.00,
 "date": "2025-01-15" },
```

```
  { "_id": 2, "status": "processing", "customer": "Saanvi Sarkar", "total": 89.99,
 "date": "2025-01-20" },
  { "_id": 3, "status": "cancelled", "customer": "Zhang Wei", "total": 220.50, "date":
 "2025-01-18" }
])
```

쿼리 예제

```
db.orders.aggregate([
  {
    $redact: {
      $cond: {
        if: { $in: ["$status", ["shipped", "processing"]] },
        then: "$$KEEP",
        else: "$$PRUNE"
      }
    }
  }
])
```

출력

```
[
  { _id: 1, status: 'shipped', customer: 'Carlos Salazar', total: 150, date:
 '2025-01-15' },
  { _id: 2, status: 'processing', customer: 'Saanvi Sarkar', total: 89.99, date:
 '2025-01-20' }
]
```

이 예제에서 $redact 단계는 각 문서의 status 필드 값을 확인합니다. 이 "배송" 또는 "처리"status인 경우 문서는 보관됩니다($$KEEP). 그렇지 않으면 문서가 정리됩니다($$PRUNE).

코드 예제

$redact 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('orders');

  const result = await collection.aggregate([
    {
      $redact: {
        $cond: {
          if: { $in: ['$status', ['shipped', 'processing']] },
          then: '$$KEEP',
          else: '$$PRUNE'
        }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['orders']

    result = list(collection.aggregate([
        {
            '$redact': {
                '$cond': {
                    'if': { '$in': ['$status', ['shipped', 'processing']] },
                    'then': '$$KEEP',
                    'else': '$$PRUNE'
                }
```

```
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$reduce

Amazon DocumentDB의 $reduce 집계 연산자는 배열의 요소에 두 인수의 함수를 누적하여 적용하여 배열을 단일 값으로 줄이는 데 사용됩니다. 이 연산자는 집계 파이프라인 내의 배열 데이터에 대해 복잡한 계산 또는 변환을 수행하는 데 특히 유용합니다.

파라미터

- input: 축소할 배열입니다.
- initialValue: 축소 작업에 사용할 초기 값입니다.
- in: input 배열의 각 요소에 대해 평가할 표현식입니다. 이 표현식은 축소의 다음 반복에 사용할 값을 반환해야 합니다.

예제(MongoDB 쉘)

다음 예제에서는 $reduce 연산자를 사용하여 배열의 모든 요소 합계를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.orders.insertMany([
  { "_id": 1, "items": [1, 2, 3, 4, 5] },
  { "_id": 2, "items": [10, 20, 30] },
  { "_id": 3, "items": [5, 15, 25, 35] },
  { "_id": 4, "items": [100, 200] }
])
```

쿼리 예제

```
db.orders.aggregate([
  {
```

```
    $project: {
      total: {
        $reduce: {
          input: "$items",
          initialValue: 0,
          in: { $add: ["$$value", "$$this"] }
        }
      }
    }
  }
])
```

출력

```
[
  { "_id": 1, "total": 15 },
  { "_id": 2, "total": 60 },
  { "_id": 3, "total": 80 },
  { "_id": 4, "total": 300 }
]
```

$reduce 연산자는 items 배열을 반복하여 각 요소를 0initialValue의에 추가합니다. 결과는 배열에 있는 모든 요소의 합계입니다.

코드 예제

$reduce 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

다음은 Node.js 애플리케이션에서 $reduce 연산자를 사용하는 예입니다.

```
const { MongoClient } = require("mongodb");

async function main() {
  const client = await MongoClient.connect("mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false");
  const db = client.db("test");
  const orders = db.collection("orders");

  const result = await orders.aggregate([
    {
```

```
      $project: {
        total: {
          $reduce: {
            input: "$items",
            initialValue: 0,
            in: { $add: ["$$value", "$$this"] }
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

다음은 Python 애플리케이션에서 $reduce 연산자를 사용하는 예입니다.

```
from pymongo import MongoClient

def main():
    client = MongoClient("mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false")
    db = client["test"]
    orders = db["orders"]

    result = list(orders.aggregate([
        {
            "$project": {
                "total": {
                    "$reduce": {
                        "input": "$items",
                        "initialValue": 0,
                        "in": { "$add": ["$$value", "$$this"] }
                    }
                }
            }
        }
    ]))
```

```
    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$regexFind

버전 5.0의 새 버전입니다.

Elastic 클러스터에서는 지원되지 않습니다.

Amazon DocumentDB의 `$regexFind` 연산자는 문서 내의 문자열 필드에서 정규식 일치를 수행하는 데 사용됩니다. 이를 통해 지정된 정규식 패턴과 일치하는 특정 하위 문자열을 검색하고 추출할 수 있습니다.

파라미터

- `input`: 검색할 문자열 필드 또는 표현식입니다.
- `regex`: 일치시킬 정규식 패턴입니다.
- `options`: (선택 사항) 대소문자 구분 및 여러 줄 일치와 같은 정규 표현식의 선택적 파라미터를 지정하는 객체입니다. 지원되는 옵션은 `i` (대/소문자 비구분) 및 `m` (다중 줄)입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$regexFind` 연산자를 사용하여 `name` 필드가 특정 정규식 패턴과 일치하는 문서를 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { "_id": 1, name: "John Doe", email: "john@example.com" },
  { "_id": 2, name: "Diego Ramirez", email: "diego@example.com" },
  { "_id": 3, name: "Alejandro Rosalez", email: "alejandro@example.com" },
  { "_id": 4, name: "Shirley Rodriguez", email: "shirley@example.com" }
]);
```

쿼리 예제

```
db.users.aggregate([
  {
    $project: {
      names: {
        $regexFind: { input: '$name', regex: 'j', options: 'i' }
      }
    }
  },
  { $match: {names: {$ne: null}}}
])
```

이 쿼리는 name 필드에 문자 "j"(대소문자 비구분)가 포함된 모든 문서를 반환합니다.

출력

```
[
  { _id: 1, names: { match: 'J', idx: 0, captures: [] } }
]
```

참고: 쿼리가 Amazon DocumentDB 플래너 버전 1을 사용하는 경우 힌트를 사용하여 인덱스를 활용해야 합니다. 힌트가 없으면 쿼리가 컬렉션 스캔을 수행할 수 있습니다. 플래너 버전을 확인하고 힌트 사용에 대해 자세히 알아보려면 섹션을 참조하세요쿼리 플래너 v2.

코드 예제

$regexFind 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const results = await users.aggregate([
    { $project: { names: { $regexFind: { input: "$name", regex: "john", options:
 "i" }}}},
```

```
    { $match: {names: {$ne: null}}}
  ]).toArray();


  console.log(results);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
users = db['users']

results = list(users.aggregate([
    { "$project": { "names": { "$regexFind": { "input": "$name", "regex": "john",
 "options": "i" }}}},
    { "$match": {"names": {"$ne": None}}}
]))

print(results)

client.close()
```

$regexFindAll

5.0에 도입

Amazon DocumentDB의 `$regexFindAll` 연산자는 문서 내의 문자열 필드에서 정규식 일치를 수행하는 데 사용됩니다. 이를 통해 지정된 정규식 패턴과 일치하는 특정 하위 문자열을 검색하고 추출하여 정규식의 모든 일치 항목을 반환할 수 있습니다.

파라미터

- `input`: 검색할 문자열 필드 또는 표현식입니다.

- `regex`: 일치시킬 정규식 패턴입니다.
- `options`: (선택 사항) 대소문자 구분 및 여러 줄 일치와 같은 정규 표현식의 선택적 파라미터를 지정하는 객체입니다. 지원되는 옵션은 `i` (대/소문자 비구분) 및 `m` (다중 줄)입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$regexFindAll` 연산자를 사용하여 `email` 필드에서 모든 문자 시퀀스를 추출하는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { _id: 1, name: "John Doe", email: "john@example.com", phone: "555-1234" },
  { _id: 2, name: "Jane Roe", email: "jane@example.com", phone: "555-5678" },
  { _id: 3, name: "Carlos Salazar", email: "carlos@example.com", phone: "555-3456" },
  { _id: 4, name: "Saanvi Sarkar", email: "saanvi@example.com", phone: "555-7890" }

]);
```

쿼리 예제

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      emailMatches: {
        $regexFindAll: { input: '$email', regex: '[a-z]+', options: 'i' }
      }
    }
  }
])
```

출력

```
[
  {
    _id: 1,
    name: 'John Doe',
    emailMatches: [
      { match: 'john', idx: 0, captures: [] },
      { match: 'example', idx: 5, captures: [] },
```

```
      { match: 'com', idx: 13, captures: [] }
    ]
  },
  {
    _id: 2,
    name: 'Jane Roe',
    emailMatches: [
      { match: 'jane', idx: 0, captures: [] },
      { match: 'example', idx: 5, captures: [] },
      { match: 'com', idx: 13, captures: [] }
    ]
  },
  {
    _id: 3,
    name: 'Carlos Salazar',
    emailMatches: [
      { match: 'carlos', idx: 0, captures: [] },
      { match: 'example', idx: 7, captures: [] },
      { match: 'com', idx: 15, captures: [] }
    ]
  },
  {
    _id: 4,
    name: 'Saanvi Sarkar',
    emailMatches: [
      { match: 'saanvi', idx: 0, captures: [] },
      { match: 'example', idx: 7, captures: [] },
      { match: 'com', idx: 15, captures: [] }
    ]
  }
]
```

참고: 쿼리가 Amazon DocumentDB 플래너 버전 1을 사용하는 경우 힌트를 사용하여 인덱스를 활용해야 합니다. 힌트가 없으면 쿼리가 컬렉션 스캔을 수행할 수 있습니다. 플래너 버전을 확인하고 힌트 사용에 대해 자세히 알아보려면 섹션을 참조하세요쿼리 플래너 v2.

코드 예제

`$regexFindAll` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

다음은 Node.js 애플리케이션에서 `$regexFind` 연산자를 사용하는 예입니다.

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const results = await users.aggregate([
    {
      $project: {
        name: 1,
        emailMatches: {
          $regexFindAll: { input: "$email", regex: "[a-z]+", options: "i" }
        }
      }
    }
  ]).toArray();

  console.log(JSON.stringify(results, null, 2));

  await client.close();
}

main();
```

Python

다음은 Python 애플리케이션에서 `$regexFind` 연산자를 사용하는 예입니다.

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
users = db['users']

results = list(users.aggregate([
    {
        "$project": {
            "name": 1,
```

```
            "emailMatches": {
                "$regexFindAll": {
                    "input": "$email",
                    "regex": "[a-z]+",
                    "options": "i"
                }
            }
        }
    }
]))

print(results)

client.close()
```

$regexMatch

버전 5.0의 새 버전입니다. Elastic 클러스터에서는 지원되지 않습니다.

Amazon DocumentDB의 `$regexMatch` 연산자는 문자열 필드에서 정규식 일치를 수행하는 데 사용됩니다. 입력 문자열이 지정된 패턴과 일치하는지 여부를 나타내는 부울 값(`true` 또는 `false`)을 반환합니다.

파라미터

- `input`: 정규식에 대해 테스트할 문자열입니다.
- `regex`: 일치시킬 정규식 패턴입니다.
- `options`: (선택 사항) 대소문자를 구분하지 않는 일치() 또는 여러 줄 일치(`i`와 같은 정규 표현식 동작을 수정하는 플래그입니다`m`.

예제(MongoDB 쉘)

다음 예제에서는 `$regexMatch` 연산자를 사용하여 이름이 'M' 문자로 시작하는지 확인하는 방법을 보여줍니다. 연산자는 각 문서에 `false` 대해 `true` 또는를 반환합니다.

샘플 문서 생성

```
db.users.insertMany([
  { "_id":1, name: "María García", email: "maría@example.com" },
  { "_id":2, name: "Arnav Desai", email: "arnav@example.com" },
```

```
  { "_id":3, name: "Martha Rivera", email: "martha@example.com" },
  { "_id":4, name: "Richard Roe", email: "richard@example.com" },

]);
```

쿼리 예제

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      startsWithM: {
        $regexMatch: {
          input: "$name",
          regex: "^M",
          options: "i"
        }
      }
    }
  }
]);
```

출력

```
{ _id: 1, name: 'María García', startsWithM: true },
{ _id: 2, name: 'Arnav Desai', startsWithM: false },
{ _id: 3, name: 'Martha Rivera', startsWithM: true },
{ _id: 4, name: 'Richard Roe', startsWithM: false }
```

코드 예제

$regexMatch 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function checkNamePattern() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
```

```
  const collection = db.collection('users');

  const result = await collection.aggregate([
    {
      $project: {
        name: 1,
        startsWithM: {
          $regexMatch: {
            input: "$name",
            regex: "^M",
            options: "i"
          }
        }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

checkNamePattern();
```

Python

```
from pymongo import MongoClient

def check_name_pattern():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.users

    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'startsWithM': {
                    '$regexMatch': {
                        'input': '$name',
                        'regex': '^M',
```

```
                        'options': 'i'
                    }
                }
            }
        }
    ]))

    print(result)

    client.close()

check_name_pattern()
```

$replaceAll

5.0에 도입

Amazon DocumentDB의 `$replaceAll` 연산자는 필드 내에서 지정된 문자열 패턴의 모든 발생을 새 문자열로 대체하는 데 사용됩니다. 이 연산자는 데이터 정규화, 텍스트 정리 및 문자열 조작과 같은 작업에 유용할 수 있습니다.

파라미터

- `input`: 대체할 문자열이 포함된 필드 또는 표현식입니다.
- `find`: 검색 및 대체할 문자열 패턴입니다.
- `replacement`: 일치하는 발생을 대체할 문자열입니다.

예제(MongoDB 쉘)

다음 예제에서는 집계 파이프라인에서 `$replaceAll` 연산자를 사용하여 문자열 "Chocolatier"의 모든 발생을 "Chocolate Co."로 대체하는 방법을 보여줍니다. "products" 컬렉션의 "brandName" 필드에

샘플 문서 생성

```
db.products.insertMany([
  {
    "_id": 1,
    "productId": "PROD-0Y9GL0",
    "brandName": "Gordon's Chocolatier",
    "category": "CPG",
    "rating": {
```

```
      "average": 4.8
    }
  },
  {
    "_id": 2,
    "productId": "PROD-1X2YZ3",
    "brandName": "Premium Chocolatier",
    "category": "CPG",
    "rating": {
      "average": 4.5
    }
  },
  {
    "_id": 3,
    "productId": "PROD-Y2E9H5",
    "name": "Nutrition Co. - Original Corn Flakes Cereal",
    "category": "Breakfast Cereals",
    "price": 8.5
  }
]);
```

쿼리 예제

```
db.products.aggregate([
  {
    $addFields: {
      "brandName": {
        $replaceAll: {
          input: "$brandName",
          find: "Chocolatier",
          replacement: "Chocolate Co."
        }
      }
    }
  }
])
```

출력

```
[
  {
    _id: 1,
    productId: 'PROD-0Y9GL0',
```

```
    brandName: "Gordon's Chocolate Co.",
    category: 'CPG',
    rating: { average: 4.8 }
  },
  {
    _id: 2,
    productId: 'PROD-1X2YZ3',
    brandName: 'Premium Chocolate Co.',
    category: 'CPG',
    rating: { average: 4.5 }
  },
  {
    _id: 3,
    productId: 'PROD-Y2E9H5',
    name: 'Nutrition Co. - Original Corn Flakes Cereal',
    category: 'Breakfast Cereals',
    price: 8.5,
    brandName: null
  }
]
```

코드 예제

`$replaceAll` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function replaceAll() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  const results = await collection.aggregate([
    {
      $addFields: {
        "brandName": {
          $replaceAll: {
            input: "$brandName",
            find: "Chocolatier",
            replacement: "Chocolate Co."
```

```
          }
        }
      }
    }
  ]).toArray();

  console.log(results);

  await client.close();
}

replaceAll();
```

Python

```
from pymongo import MongoClient

def replace_all():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.products

    results = list(collection.aggregate([
        {
            "$addFields": {
                "brandName": {
                    "$replaceAll": {
                        "input": "$brandName",
                        "find": "Chocolatier",
                        "replacement": "Chocolate Co."
                    }
                }
            }
        }
    ]))

    print(results)

    client.close()

replace_all()
```

$replaceOne

5.0에 도입

Amazon DocumentDB의 `$replaceOne` 연산자는 집계 파이프라인 내에서 문자열 내에서 지정된 하위 문자열의 첫 번째 발생을 대체 문자열로 대체하는 데 사용되는 문자열 표현식 연산자입니다. 이 연산자는 대/소문자를 구분하며 발견된 첫 번째 일치 항목만 대체합니다.

파라미터

- `input`: 찾기를 수행할 문자열(필드)입니다.
- `find`: 입력 내에서 검색할 문자열입니다.
- `replacement`: 입력(필드)에서 찾기의 첫 번째 발생을 대체하는 문자열입니다.

예제(MongoDB 쉘)

다음 예제에서는 집계 파이프라인 내에서 `$replaceOne` 연산자를 사용하여 제품 이름의 하위 문자열을 대체하는 방법을 보여줍니다.

샘플 문서 생성

```
db.products.insertMany([
  { "_id":1, "productId": "PROD-0Y9GL0", "name": "Gordon's Extra Creamy Milk Chocolate
 - Pack of 4", "category": "Confectionery", "price": 24.99 },
  { "_id":2, "productId": "PROD-Y2E9H5", "name": "Nutrition Co. - Original Corn Flakes
 Cereal", "category": "Breakfast Cereals", "price": 8.50 },
  { "_id":3, "productId": "PROD-Z3F8K2", "name": "Gordon's Dark Chocolate (90% Cocoa)
 Pack - Pack of 4", "category": "Confectionery", "price": 28.99 }
]);
```

집계 예제

```
db.products.aggregate([
  {
    $addFields: {
      standardizedName: {
        $replaceOne: {
          input: "$name",
          find: "Pack",
          replacement: "Package"
        }
```

```
      }
    }
  }
]);
```

출력

출력은 각 제품 이름에서 첫 번째로 발생한 "Pack"만 "Package"로 대체되었음을 보여줍니다.

```
[
  {
    _id: 1,
    productId: 'PROD-0Y9GL0',
    name: "Gordon's Extra Creamy Milk Chocolate - Pack of 4",
    category: 'Confectionery',
    price: 24.99,
    standardizedName: "Gordon's Extra Creamy Milk Chocolate - Package of 4"
  },
  {
    _id: 2,
    productId: 'PROD-Y2E9H5',
    name: 'Nutrition Co. - Original Corn Flakes Cereal',
    category: 'Breakfast Cereals',
    price: 8.5,
    standardizedName: 'Nutrition Co. - Original Corn Flakes Cereal'
  },
  {
    _id: 3,
    productId: 'PROD-Z3F8K2',
    name: "Gordon's Dark Chocolate (90% Cocoa) Pack - Pack of 4",
    category: 'Confectionery',
    price: 28.99,
    standardizedName: "Gordon's Dark Chocolate (90% Cocoa) Package - Pack of 4"
  }
```

코드 예제

`$replaceOne` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function replaceOne() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  const pipeline = [
    {
      $addFields: {
        standardizedName: {
          $replaceOne: {
            input: '$name',
            find: 'Pack',
            replacement: 'Package'
          }
        }
      }
    }
  ];

  const result = await collection.aggregate(pipeline).toArray();
  console.log(result);

  await client.close();
}

replaceOne();
```

Python

```
from pymongo import MongoClient

def replaceOne():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['products']

    pipeline = [
        {
            '$addFields': {
```

```
                'standardizedName': {
                    '$replaceOne': {
                        'input': '$name',
                        'find': 'Pack',
                        'replacement': 'Package'
                    }
                }
            }
        }
    ]

    result = list(collection.aggregate(pipeline))
    print(result)

    client.close()

replaceOne()
```

$replaceRoot

$replaceRoot 연산자는 루트 문서를 지정된 임베디드 문서로 대체하는 데 사용됩니다. 이는 중첩된 문서를 최상위 수준으로 승격하거나 데이터 출력을 재구성하려는 경우에 유용합니다.

파라미터

- newRoot: 기존 루트 문서를 대체할 새 루트 문서입니다.

예제(MongoDB 쉘)

이 예제에서는 배송 레이블 또는 주소 목록을 생성하는 데 유용한 고객 주문에서 배송 주소 정보를 추출하는 방법을 보여줍니다.

샘플 문서 생성

```
db.orders.insertMany([
  {
    "_id":1, "orderId": "ORD-2024-001", "customerId": "CUST-12345", "orderDate":
 "2024-01-15", "shippingAddress": { "name": "María García", "street": "123
 Main St", "city": "Seattle", "state": "WA", "zipCode": "98101", "country":
 "USA" },"totalAmount": 149.99 },
  { "_id":2, "orderId": "ORD-2024-002", "customerId": "CUST-67890", "orderDate":
 "2024-01-16", "shippingAddress": { "name": "Arnav Desai", "street": "456 Oak
```

```
 Ave", "city": "Portland", "state": "OR", "zipCode": "97201", "country": "USA" },
 "totalAmount": 89.50 } ])
```

쿼리 예제

```
db.orders.aggregate([
  {
    $replaceRoot: {
      newRoot: "$shippingAddress"
    }
  }
])
```

출력

```
{
    name: 'María García',
    street: '123 Main St',
    city: 'Seattle',
    state: 'WA',
    zipCode: '98101',
    country: 'USA'
  },
  {
    name: 'Arnav Desai',
    street: '456 Oak Ave',
    city: 'Portland',
    state: 'OR',
    zipCode: '97201',
    country: 'USA'
  }
```

코드 예제

`$replaceRoot` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function extractShippingAddresses() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('orders');

  const result = await collection.aggregate([
    {
      $replaceRoot: {
        newRoot: "$shippingAddress"
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

extractShippingAddresses();
```

Python

```
from pymongo import MongoClient

def extract_shipping_addresses():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['orders']

    result = list(collection.aggregate([
        {
            "$replaceRoot": {
                "newRoot": "$shippingAddress"
            }
        }
    ]))

    print(result)

    client.close()
```

```
extract_shipping_addresses()
```

$replaceWith

버전 8.0의 새로운 기능

Elastic 클러스터에서는 지원되지 않습니다.

Amazon DocumentDB의 `$replaceWith` 집계 단계는 입력 문서를 새 문서로 대체하는 데 사용됩니다. _id 필드를 포함하여 입력 문서의 모든 기존 필드는 새 문서로 대체됩니다. `$replaceWith`는 일반적으로 문서를 평면화하거나 임베디드 문서를 최상위 수준으로 승격하는 데 사용됩니다.

파라미터

- `<replacement>` (필수): 기존 문서를 대체할 새 문서입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$replaceWith` 연산자를 사용하여 Amazon DocumentDB 컬렉션의 기존 문서를 교체하는 방법을 보여줍니다.

샘플 문서 생성

```
db.restaurants.insertMany([
  {
    "restaurantId": "REST-0Y9GL0",
    "name": "Biryani Adda",
    "cuisine": "Indian",
    "ratings": [ 3, 4, 3, 2, 2, 4, 1, 5, 5, 5 ]
  },
  {
    "restaurantId": "REST-8L2PX9",
    "name": "The Burger Spot",
    "cuisine": "American",
    "ratings": [ 2, 3, 4, 5, 3, 1, 1, 2, 4 ]
  }
]);
```

쿼리 예제

```
db.restaurants.aggregate([
  { $replaceWith: {
      name: "$name",
      cuisine: "$cuisine",
      rating: { $avg: "$ratings" }
    }
  }
]);
```

출력

```
[
  {
    name: 'Biryani Adda',
    cuisine: 'Indian',
    rating: 3.4
  },
  {
    name: 'The Burger Spot',
    cuisine: 'American',
    rating: 2.7777777777777777
  }
]
```

코드 예제

$replaceWith 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function replaceDoc() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('restaurants');

  const result = await collection.aggregate([
    { $replaceWith: {
        name: "description_index",
        cuisine: 2,
```

```
        rating: { $avg: "$ratings" }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

replaceDoc();
```

Python

```
from pymongo import MongoClient


def replace_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.restaurants

    result = list(collection.aggregate([
        {
            '$replaceWith': {
                'name': "$name",
                'cuisine': "$cuisine",
                'rating': { '$avg': "$ratings"}
            }
        }
    ]))

    print(result)
    client.close()

replace_document()
```

$reverseArray

Amazon DocumentDB의 `$reverseArray` 연산자는 배열의 요소를 지정된 순서로 되돌리는 데 사용됩니다. 이 연산자는 배열의 요소를 역방향으로 재정렬해야 할 때 유용합니다.

파라미터

- expression: 반전할 배열 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $reverseArray 연산자를 사용하여 배열의 요소 순서를 반전하는 방법을 보여줍니다.

샘플 문서 생성

```
db.miles.insertMany([
  { "_id" : 1, "member_since" : ISODate("1987-01-01T00:00:00Z"), "credit_card" : false,
 "flight_miles" : [ 1205, 2560, 880 ]},
  { "_id" : 2, "member_since" : ISODate("1982-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 1205, 2560, 890, 2780]},
  { "_id" : 3, "member_since" : ISODate("1999-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 1205, 880]}
]);
```

쿼리 예제

```
db.miles.aggregate([
  {
    $project: {
      _id: 1,
      member_since: 1,
      credit_card: 1,
      reversed_flight_miles: { $reverseArray: "$flight_miles" }
    }
  }
]);
```

출력

```
{ "_id" : 1, "member_since" : ISODate("1987-01-01T00:00:00Z"), "credit_card" : false,
 "reversed_flight_miles" : [ 880, 2560, 1205 ] }
{ "_id" : 2, "member_since" : ISODate("1982-01-01T00:00:00Z"), "credit_card" : true,
 "reversed_flight_miles" : [ 2780, 890, 2560, 1205 ] }
{ "_id" : 3, "member_since" : ISODate("1999-01-01T00:00:00Z"), "credit_card" : true,
 "reversed_flight_miles" : [ 880, 1205 ] }
```

이 예제에서는 $reverseArray 연산자를 사용하여 flight_miles 배열 순서를 반대로 지정합니다. 출력의 결과 reversed_flight_miles 필드에는 배열의 요소가 역순으로 표시됩니다.

코드 예제

$reverseArray 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

다음은 Node.js 애플리케이션에서 $reverseArray 연산자를 사용하는 예입니다.

```
const { MongoClient } = require('mongodb');

async function reverseArray() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('miles');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        member_since: 1,
        credit_card: 1,
        reversed_flight_miles: { $reverseArray: '$flight_miles' }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

reverseArray();
```

Python

다음은 Python 애플리케이션에서 $reverseArray 연산자를 사용하는 예입니다.

```
from pymongo import MongoClient
```

```
def reverse_array():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.miles

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'member_since': 1,
                'credit_card': 1,
                'reversed_flight_miles': { '$reverseArray': '$flight_miles' }
            }
        }
    ]))

    print(result)
    client.close()

reverse_array()
```

$round

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $round 연산자는 숫자를 지정된 소수점 자리로 반올림합니다. 값이 해당 위치에서 가장 가까운 두 값 사이의 정확히 중간에 있는 경우는 가장 가까운 짝수로 $round 반올림합니다. 예를 들어 2.5는 2로, 3.5는 4로 반올림합니다.

파라미터

- number: 숫자로 확인되는 표현식입니다.
- place: 선택 사항입니다. 반올림할 소수 자릿수를 지정하는 -20~100 사이의 정수 표현식입니다. 음수 값은 소수점 왼쪽으로 반올림됩니다. 기본값은 0입니다.

예제(MongoDB 쉘)

다음 예제에서는 $round 연산자를 사용하여 숫자 값을 소수점 한 자리로 반올림하는 방법을 보여줍니다.

양수 자리 값으로 반올림

```
db.measurements.insertMany([
  {_id: 1, value: 3.456},
  {_id: 2, value: 7.891},
  {_id: 3, value: 12.345}
]);
```

쿼리 예제

```
db.measurements.aggregate([
  { $project: { rounded: { $round: ["$value", 1] } } }
]);
```

출력

```
[
  {_id: 1, rounded: 3.5},
  {_id: 2, rounded: 7.9},
  {_id: 3, rounded: 12.3}
]
```

반원 값을 가장 가까운 짝수로 반올림

값이 대상 위치에서 가장 가까운 두 값 사이의 정확히 중간에 있는 경우는 항상 $round 반올림하는 대신 가장 가까운 짝수로 반올림합니다. 다음 예제에서는 반원 값을 정수로 반올림합니다.

```
db.halfway.insertMany([
  {_id: 1, value: 10.5},
  {_id: 2, value: 11.5},
  {_id: 3, value: 12.5},
  {_id: 4, value: 13.5}
]);
```

```
db.halfway.aggregate([
  { $project: { rounded: { $round: ["$value", 0] } } }
]);
```

출력

```
[
```

```
  {_id: 1, rounded: 10},
  {_id: 2, rounded: 12},
  {_id: 3, rounded: 12},
  {_id: 4, rounded: 14}
]
```

10.5는 10와의 정확히 중간에 11있기 때문에 가장 가까운 짝수 값 로 반올림됩니다10. 마찬가지로 11.5 및는 12.5 모두 가장 가까운 짝수 값 로 반올림하고 12는 로 13.5 반올림합니다14.

음수 자리 값으로 반올림

place가 음수이면 소수점 왼쪽으로 $round 반올림합니다. 다음 예제에서는 첫 번째 숫자를 소수점 왼쪽으로 사용하여 값을 반올림합니다.

```
db.samples.insertMany([
  {_id: 1, value: 19.25},
  {_id: 2, value: 28.73},
  {_id: 3, value: 34.32}
]);
```

```
db.samples.aggregate([
  { $project: { rounded: { $round: ["$value", -1] } } }
]);
```

출력

```
[
  {_id: 1, rounded: 20},
  {_id: 2, rounded: 30},
  {_id: 3, rounded: 30}
]
```

코드 예제

$round 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('measurements');
    const result = await collection.aggregate([
      { $project: { rounded: { $round: ["$value", 1] } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['measurements']
        result = list(collection.aggregate([
            {'$project': {'rounded': {'$round': ['$value', 1]}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$rtrim

버전 4.0의 새 버전입니다.

Elastic 클러스터에서는 지원되지 않습니다.

Amazon DocumentDB의 $rtrim 연산자는 문자열에서 후행 문자를 제거하는 데 사용됩니다. 기본적으로 후행 공백 문자를 제거하지만 문자 인수를 전달하여 제거할 문자 집합을 지정할 수도 있습니다.

파라미터

- input: 후행 공백 문자를 제거할 입력 문자열입니다.
- chars: (선택 사항) 특정 문자를 제거합니다.

예제(MongoDB 쉘)

다음 예제에서는 $rtrim 연산자를 사용하여 문자열에서 후행 지정 문자(" *")를 제거하는 방법을 보여줍니다.

샘플 문서 생성

```
db.collection.insert([
  { "name": "John Doe*  ", "age": 30 },
  { "name": "Jane Smith  ", "age": 25 },
  { "name": "Bob Johnson", "age": 35 }
]);
```

쿼리 예제

```
db.collection.aggregate([
  {
    $project: {
      _id: 0,
      name: { $rtrim: { input: "$name", chars: " *" } },
      age: 1
    }
  }
]);
```

출력

```
[
  { age: 30, name: 'John Doe' },
  { age: 25, name: 'Jane Smith' },
  { age: 35, name: 'Bob Johnson' }
]
```

코드 예제

`$rtrim` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient, Bson } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {

    const db = client.db('test');
    const collection = db.collection('collection');

    const pipeline = [
      {
        $project: {
          _id: 0,
          name: { $rtrim: { input: "$name", chars: " *" } },
          age: 1
        }
      }
    ];

    const results = await collection.aggregate(pipeline).toArray();
    console.dir(results, { depth: null });

  } catch (err) {
    console.error('Error occurred:', err);
  } finally {
    await client.close();
  }
}

example().catch(console.error);
```

Python

```
from pymongo import MongoClient
```

```
def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:

        db = client.test
        collection = db.collection

        pipeline = [
            {
                "$project": {
                    "_id": 0,
                    "name": { "$rtrim": { "input": "$name", "chars": " *" } },
                    "age": 1
                }
            }
        ]

        results = collection.aggregate(pipeline)

        for doc in results:
            print(doc)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        client.close()

example()
```

$sample

Amazon DocumentDB의 `$sample` 집계 단계는 컬렉션에서 지정된 수의 문서를 무작위로 선택하는 데 사용됩니다. 이는 데이터 분석, 테스트 및 추가 처리를 위한 샘플 생성과 같은 작업에 유용합니다.

파라미터

- `size`: 임의로 선택할 문서 수입니다.

예제(MongoDB 쉘)

다음 예제에서는 $sample 스테이지를 사용하여 temp 컬렉션에서 두 문서를 무작위로 선택하는 방법을 보여줍니다.

샘플 문서 생성

```
db.temp.insertMany([
  { "_id": 1, "temperature": 97.1, "humidity": 0.60, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 2, "temperature": 98.2, "humidity": 0.59, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 3, "temperature": 96.8, "humidity": 0.61, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 4, "temperature": 97.9, "humidity": 0.61, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 5, "temperature": 97.5, "humidity": 0.60, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 6, "temperature": 98.0, "humidity": 0.59, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 7, "temperature": 97.2, "humidity": 0.60, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 8, "temperature": 98.1, "humidity": 0.59, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 9, "temperature": 96.9, "humidity": 0.62, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") },
  { "_id": 10, "temperature": 97.7, "humidity": 0.60, "timestamp":
 ISODate("2019-03-21T21:17:22.425Z") }
]);
```

쿼리 예제

```
db.temp.aggregate([
   { $sample: { size: 2 } }
])
```

출력

```
{ "_id" : 4, "temperature" : 97.9, "humidity" : 0.61, "timestamp" :
 ISODate("2019-03-21T21:17:22.425Z") }
{ "_id" : 9, "temperature" : 96.9, "humidity" : 0.62, "timestamp" :
 ISODate("2019-03-21T21:17:22.425Z") }
```

결과에 따르면 문서 10개 중 2개가 무작위로 샘플링되었습니다. 이제 이러한 문서를 사용하여 평균을 결정하거나 최소/최대 계산을 수행할 수 있습니다.

코드 예제

$sample 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function sampleDocuments() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('temp');

  const result = await collection.aggregate([
    { $sample: { size: 2 } }
  ]).toArray();

  console.log(result);
  await client.close();
}

sampleDocuments();
```

Python

```
from pymongo import MongoClient

def sample_documents():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['temp']

    result = list(collection.aggregate([
        { '$sample': { 'size': 2 } }
    ]))
```

```
    print(result)
    client.close()

sample_documents()
```

$search

버전 5.0의 새 버전입니다.

Amazon DocumentDB의 $search 연산자는 텍스트 검색 기능을 제공하는 데 사용됩니다.

파라미터

없음

예제(MongoDB 쉘)

다음 예제에서는 $search 연산자를 사용하여 텍스트 검색 쿼리를 수행하는 방법을 보여줍니다.

샘플 문서 생성

```
db.textcollection.createIndex({"description": "text"});

db.textcollection.insertMany([
  { _id: 1, name: "John Doe", description: "This is a sample document about John
 Doe." },
  { _id: 2, name: "Jane Smith", description: "This is a sample document about Jane
 Smith." },
  { _id: 3, name: "Bob Johnson", description: "This is a sample document about Bob
 Johnson." },
  { _id: 4, name: "Jon Jeffries", description: "This is a sample document about Jon
 Jeffries." }
]);
```

쿼리 예제

```
db.textcollection.find(
  { $text: { $search: "John" } }
);
```

출력

```
[
  {
    _id: 1,
    name: 'John Doe',
    description: 'This is a sample document about John Doe.'
  }
]
```

코드 예제

$search 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findWithText() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('textcollection');

  const result = await collection.find(
    { $text: { $search: "John" } }
  ).sort({ score: { $meta: "textScore" } }).toArray();

  console.log(result);
  client.close();
}

findWithText();
```

Python

```
from pymongo import MongoClient

def find_with_text():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['textcollection']
```

```
    result = list(collection.find(
        { '$text': { '$search': 'John' } }
    ))

    print(result)
    client.close()

find_with_text()
```

$second

Amazon DocumentDB의 $second 연산자는 날짜 또는 타임스탬프에서 초 단위 구성 요소를 추출합니다. 날짜 또는 타임스탬프 필드에서 초 값을 검색하는 데 사용됩니다.

파라미터

- expression: 초 값을 추출할 날짜 또는 타임스탬프 필드입니다. 이 표현식은 필드 경로이거나 날짜 또는 타임스탬프로 확인되는 유효한 표현식일 수 있습니다.

예제(MongoDB 쉘)

다음 예제에서는 $second 연산자를 사용하여 날짜 필드에서 초 구성 요소를 추출하는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { name: "John", dob: new Date("1990-05-15T12:30:45Z") },
  { name: "Jane", dob: new Date("1985-09-20T23:59:59Z") },
  { name: "Bob", dob: new Date("2000-01-01T00:00:00Z") }
]);
```

쿼리 예제

```
db.users.aggregate([{ $project: { name: 1, dobSeconds: { $second: "$dob" } } }])
```

출력

```
[
```

```
  { "_id" : ObjectId("6089a9c306a829d1f8b456a1"), "name" : "John", "dobSeconds" : 45 },
  { "_id" : ObjectId("6089a9c306a829d1f8b456a2"), "name" : "Jane", "dobSeconds" : 59 },
  { "_id" : ObjectId("6089a9c306a829d1f8b456a3"), "name" : "Bob", "dobSeconds" : 0 }
]
```

코드 예제

$second 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const result = await users.aggregate([{ $project: { name: 1, dobSeconds:
 { $second: '$dob' } } }]).toArray();
  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    users = db['users']

    result = list(users.aggregate([{'$project': {'name': 1, 'dobSeconds':
 {'$second': '$dob'}}}]))
    print(result)
```

```
    client.close()

example ()
```

$set

버전 8.0의 새로운 기능

Elastic 클러스터에서는 지원되지 않습니다.

Amazon DocumentDB의 $set 집계 단계를 사용하면 집계 파이프라인 중에 설명서의 새 필드를 추가하거나 기존 필드 값을 업데이트할 수 있습니다.

파라미터

- expression: 평가할 표현식입니다. 필드 참조 및 산술 연산을 포함하여 유효한 집계 표현식일 수 있습니다.

예제(MongoDB 쉘)

다음 예제에서는 $set 집계 단계를 사용하여 quantity 필드에 price 필드를 곱하여 합계를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.inventory.insertMany([
  { item: "pencil", quantity: 100, price: 0.24},
  { item: "pen", quantity: 204, price: 1.78 }
]);
```

집계 예제

```
db.inventory.aggregate([
  {
    $set: {
      total: { $multiply: ["$quantity", "$price"] }
    }
  }
])
```

출력

```
[
  {
    _id: ObjectId('69248951d66dcae121d2950d'),
    item: 'pencil',
    quantity: 100,
    price: 0.24,
    total: 24
  },
  {
    _id: ObjectId('69248951d66dcae121d2950e'),
    item: 'pen',
    quantity: 204,
    price: 1.78,
    total: 363.12
  }
]
```

코드 예제

$set 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const inventory = db.collection('inventory');

  const result = await inventory.aggregate([
      {
        $set: {
          total: { $multiply: ["$quantity", "$price"] }
        }
      }
  ]).toArray();

  console.log(result);
```

```
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    inventory = db['inventory']

    result = list(inventory.aggregate([
      {
        "$set": {
          "total": { "$multiply": ["$quantity", "$price"] }
        }
      }
    ]))

    print(result)
    client.close()

example()
```

$setDifference

버전 4.0의 새 버전입니다.

Amazon DocumentDB의 `$setDifference` 연산자는 두 세트를 비교하고 첫 번째 세트에는 있지만 두 번째 세트에는 없는 요소를 반환하는 데 사용됩니다. 이 연산자는 두 세트 사이의 고유한 요소를 찾는 데 유용합니다.

파라미터

- `firstSet` : 비교할 첫 번째 세트입니다.
- `secondSet` : 비교할 두 번째 세트입니다.

예제(MongoDB 쉘)

다음 예제에서는 $setDifference 연산자를 사용하여 두 세트 간의 고유한 요소를 찾는 방법을 보여 줍니다.

샘플 문서 생성

```
db.collection.insertMany([
  { _id: 1, fruits: ["apple", "banana", "cherry", "date"] },
  { _id: 2, fruits: ["banana", "cherry", "date", "elderberry"] }
]);
```

쿼리 예제

```
db.collection.aggregate([
  {
    $project: {
      uniqueFruits: { $setDifference: ["$fruits", ["banana", "cherry", "date"]] }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "uniqueFruits": ["apple"] },
  { "_id": 2, "uniqueFruits": ["elderberry"] }
]
```

쿼리는 다음 단계를 수행합니다.

1. $project 스테이지를 사용하여 각 문서에 uniqueFruits 대한 새 필드를 생성합니다.

2. 연$setDifference산자는 fruits 배열을 배열과 비교["banana", "cherry", "date"]하고 fruits 배열의 고유 요소를 반환합니다.

코드 예제

$setDifference 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

다음은 Node.js 애플리케이션에서 $setDifference 연산자를 사용하는 방법의 예입니다.

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('mycollection');

  // Insert sample documents
  await collection.insertMany([
    { _id: 1, fruits: ["apple", "banana", "cherry", "date"] },
    { _id: 2, fruits: ["banana", "cherry", "date", "elderberry"] }
  ]);

 // Query using $setDifference
  const result = await collection.aggregate([
    {
      $project: {
        uniqueFruits: { $setDifference: ["$fruits", ["banana", "cherry", "date"]] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

다음은 Python 애플리케이션에서 `$setDifference` 연산자를 사용하는 방법의 예입니다.

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['mycollection']
```

```
    # Insert sample documents
    collection.insert_many([
        {'_id': 1, 'fruits': ["apple", "banana", "cherry", "date"]},
        {'_id': 2, 'fruits': ["banana", "cherry", "date", "elderberry"]}
    ])

    # Query using $setDifference
    result = list(collection.aggregate([
        {
            '$project': {
                'uniqueFruits': {'$setDifference': ['$fruits', ["banana", "cherry",
 "date"]]}
            }
        }
    ]))

    print(result)
    client.close()

if __name__ == '__main__':
    main()
```

$setEquals

Amazon DocumentDB의 `$setEquals` 연산자는 두 세트가 동일한지 확인하는 데 사용됩니다. 두 배열을 비교하고 순서에 관계없이 동일한 고유 요소를 포함하는 `true` 경우를 반환합니다.

파라미터

- `expression1`: 비교할 첫 번째 배열입니다.
- `expression2`: 비교할 두 번째 배열입니다.

예제(MongoDB 쉘)

다음 예제에서는 연`$setEquals`산자를 사용하여 두 값 세트를 비교하는 방법을 보여줍니다.

샘플 문서 생성

```
db.collection.insertMany([
  { _id: 1, fruits: ["apple", "banana", "cherry"] },
  { _id: 2, fruits: ["banana", "apple", "cherry"] },
```

```
  { _id: 3, fruits: ["apple", "banana", "orange"] }
])
```

쿼리 예제

```
db.collection.find({
  $expr: {
    $setEquals: ["$fruits", ["apple", "banana", "cherry"]]
  }
})
```

출력

```
{ "_id" : 1, "fruits" : [ "apple", "banana", "cherry" ] }
{ "_id" : 2, "fruits" : [ "banana", "apple", "cherry" ] }
```

쿼리는 $setEquals 연산자를 사용하여 각 문서의 fruits 필드를 배열과 비교합니다["apple", "banana", "cherry"]. fruits 필드가 비교 배열과 동일한 문서가 반환됩니다.

코드 예제

$setEquals 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('mycollection');

  // Insert sample documents
  await collection.insertMany([
    { _id: 1, fruits: ["apple", "banana", "cherry"] },
    { _id: 2, fruits: ["banana", "apple", "cherry"] },
    { _id: 3, fruits: ["apple", "banana", "orange"] }
  ]);
```

```
  // Query using $setEquals
  const result = await collection.find({
    $expr: {
      $setEquals: ["$fruits", ["apple", "banana", "cherry"]]
    }
  }).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['mycollection']

    # Insert sample documents
    collection.insert_many([
        {"_id": 1, "fruits": ["apple", "banana", "cherry"]},
        {"_id": 2, "fruits": ["banana", "apple", "cherry"]},
        {"_id": 3, "fruits": ["apple", "banana", "orange"]}
    ])

    # Query using $setEquals
    result = list(collection.find({
        "$expr": {
            "$setEquals": ["$fruits", ["apple", "banana", "cherry"]]
        }
    }))

    print(result)
    client.close()

example()
```

$setIntersection

Amazon DocumentDB의 `$setIntersection` 연산자는 두 개 이상의 배열 간에 공통 요소를 반환하는 데 사용됩니다. 이 연산자는 데이터 세트로 작업할 때 특히 유용하므로 여러 세트의 교차점을 찾을 수 있습니다.

파라미터

- `array1`: 교차할 첫 번째 배열입니다.
- `array2`: 교차할 두 번째 배열입니다.
- `arrayN`: (선택 사항) 교차할 추가 배열입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$setIntersection` 연산자를 사용하여 두 배열 간의 공통 요소를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.collection.insertMany([
  { _id: 1, colors: ["red", "blue", "green"] },
  { _id: 2, colors: ["blue", "yellow", "orange"] },
  { _id: 3, colors: ["red", "green", "purple"] }
])
```

쿼리 예제

```
db.collection.aggregate([
  { $project: {
      _id: 1,
      commonColors: { $setIntersection: ["$colors", ["red", "blue", "green"]] }
    }
  }
])
```

출력

```
[
  { "_id": 1, "commonColors": ["red", "blue", "green"] },
  { "_id": 2, "commonColors": ["blue"] },
```

```
  { "_id": 3, "commonColors": ["red", "green"] }
]
```

코드 예제

$setIntersection 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('mycollection');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        commonColors: { $setIntersection: ["$colors", ["red", "blue", "green"]] }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['mycollection']
```

```
    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'commonColors': { '$setIntersection': ["$colors", ["red", "blue",
 "green"]] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$setIsSubset

Amazon DocumentDB의 $setIsSubset 연산자는 값 집합이 다른 집합의 하위 집합인지 확인하는 데 사용됩니다. 배열 필드에 대해 세트 기반 비교 및 작업을 수행하는 데 유용합니다.

파라미터

- field: $setIsSubset 연산자를 적용할 필드입니다.
- set: 필드를 비교할 집합입니다.

예제(MongoDB 쉘)

다음 예제에서는 연$setIsSubset산자를 사용하여 tags 필드가 지정된 세트의 하위 집합인지 확인하는 방법을 보여줍니다.

샘플 문서 생성

```
db.products.insertMany([
  { _id: 1, name: "Product A", tags: ["tag1", "tag2", "tag3"] },
  { _id: 2, name: "Product B", tags: ["tag1", "tag2"] },
  { _id: 3, name: "Product C", tags: ["tag2", "tag3"] }
]);
```

쿼리 예제

```
db.products.find({
  $expr: { $setIsSubset: [["tag1", "tag2"], "$tags"] }
})
```

참고:$setIsSubset는 집계 연산자이며 find() 쿼리에서 직접 사용할 수 없습니다. 이 예제에서는 $expr를와 함께 사용하여 쿼리 연산자와 집계 표현식 간의 차이를 find( ) 연결합니다.

출력

```
[
  { "_id" : 1, "name" : "Product A", "tags" : [ "tag1", "tag2", "tag3" ] },
  { "_id" : 2, "name" : "Product B", "tags" : [ "tag1", "tag2" ] }
]
```

쿼리는 tags 필드가 집합의 하위 집합인 문서를 반환합니다["tag1", "tag2"].

코드 예제

$setIsSubset 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  const result = await collection.find({
    $expr: { $setIsSubset: [["tag1", "tag2"], "$tags"] }
  }).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['products']

    result = list(collection.find({
        '$expr': {'$setIsSubset': [['tag1', 'tag2'], '$tags']}
    }))

    print(result)

    client.close()

example()
```

$setUnion

Amazon DocumentDB의 `$setUnion` 집계 연산자는 두 개 이상의 값 세트를 결합하고 입력 세트의 모든 고유 요소가 포함된 세트를 반환하는 데 사용됩니다. 이 연산자는 문서의 배열 필드에 대해 설정 기반 작업을 수행해야 할 때 유용합니다.

파라미터

- `expression1`: 배열로 확인되는 표현식입니다.
- `expression2`: 배열로 확인되는 표현식입니다.
- `expressionN`: 배열로 확인되는 추가 표현식(선택 사항).

예제(MongoDB 쉘)

다음 예제에서는 `$setUnion` 연산자를 사용하여 컬렉션에 있는 두 배열 필드의 고유 요소를 결합하는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { _id: 1, name: "Alice", hobbies: ["reading", "swimming"], skills: ["coding",
 "writing"] },
  { _id: 2, name: "Bob", hobbies: ["cooking", "gardening"], skills: ["coding",
 "photography"] },
  { _id: 3, name: "Charlie", hobbies: ["reading", "painting"], skills: ["gardening",
 "music"] }
]);
```

쿼리 예제

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      allInterests: { $setUnion: ["$hobbies", "$skills"] }
    }
  }
]);
```

출력

```
[
  { "_id" : 1, "name" : "Alice", "allInterests" : [ "coding", "reading", "swimming",
 "writing" ] },
  { "_id" : 2, "name" : "Bob", "allInterests" : [ "coding", "cooking", "gardening",
 "photography" ] },
  { "_id" : 3, "name" : "Charlie", "allInterests" : [ "gardening", "music", "painting",
 "reading" ] }
]
```

이 예제에서는 `$setUnion` 연산자를 사용하여 각 사용자 문서에 대한 `hobbies` 및 `skills` 배열 필드의 고유한 요소를 결합합니다. 결과 `allInterests` 필드에는 각 사용자의 모든 고유한 취미와 기술이 결합되어 있습니다.

코드 예제

`$setUnion` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const result = await users.aggregate([
    {
      $project: {
        _id: 1,
        name: 1,
        allInterests: { $setUnion: ["$hobbies", "$skills"] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    users = db['users']

    result = list(users.aggregate([
        {
            '$project': {
                '_id': 1,
                'name': 1,
                'allInterests': { '$setUnion': ["$hobbies", "$skills"] }
            }
        }
```

```
    ]))

    print(result)
    client.close()

example()
```

$setWindowFields

버전 8.0.2에서 새로 추가되었습니다.

Amazon DocumentDB의 `$setWindowFields` 단계는 파티션의 지정된 문서 범위(창)에서 작업을 수행하고 선택한 창 연산자를 기반으로 결과를 반환합니다. 스테이지는 입력 문서를 파티션으로 그룹화하고, 선택적으로 정렬하고, 각 문서에 대해 주변 문서의 창에서 하나 이상의 출력 필드를 계산합니다.

구문

```
{
  $setWindowFields: {
    partitionBy: <expression>,
    sortBy: { <sortField1>: <sortOrder>, ... },
    output: {
      <outputField1>: {
        <windowOperator>: <expression>,
        window: {
          documents: [ <lowerBound>, <upperBound> ],
          range: [ <lowerBound>, <upperBound> ],
          unit: <time unit>
        }
      },
      ...
    }
  }
}
```

파라미터

- `partitionBy`: 선택 사항입니다. 문서를 파티션으로 그룹화하는 데 사용되는 단일 표현식입니다. 필드 경로(예: `"$state"`) 또는 다른 표현식일 수 있습니다. 생략하면 모든 입력 문서가 단일 파티션에 속합니다.

- `sortBy`: 선택 사항입니다. 각 파티션의 문서를 오름`{ <field>: 1 }`차순 또는 `{ <field>: -1 }` 내림차순으로 정렬할 필드를 지정하는 문서입니다. 문서 기반 또는 범위 기반 창 경계를 사용하고 순위 연산자 중 하나를 사용하는 경우가 `sortBy` 필요합니다.
- `output`: 필수 항목입니다. 각 문서에 추가할 필드를 지정하는 문서입니다. 여러 출력 필드를 지정할 수 있지만 각 출력 필드는 정확히 하나의 창 연산자, 인수 및 창 경계를 정의하는 선택적 `window` 문서를 사용합니다`{ <field>: { <windowOperator>: <expression>, window: { ... } } }`.

창 경계

각 `output` 필드 내에서 선택적으로 `window` 문서로 창 경계를 정의할 수 있습니다. 창은 문서 위치 또는 값 범위에 따라 정의할 수 있습니다.

- `documents`: 창을 현재 문서 앞뒤의 여러 문서로 형식으로 지정합니다`[ <lowerBound>, <upperBound> ]`. 각 경계는 정수 오프셋(예: `-1` 또는 2)이거나 키워드 `"unbounded"` 및 중 하나입니다`"current"`.는 하한으로 사용되는 경우 창을 파티션의 시작으로, 상한으로 사용되는 경우 파티션의 끝으로 `"unbounded"` 확장합니다.는 현재 문서를 `"current"` 의미합니다. 두 경계가 모두가 `sortBy` 아니면 `documents` 창에가 필요합니다`"unbounded"`.
- `range`: 창을 현재 문서와 관련된 `sortBy` 필드 값의 범위로 형식으로 지정합니다`[ <lowerBound>, <upperBound> ]`. 각 경계는 현재 문서의 `sortBy` 값 또는 `"unbounded"` 키워드에 추가되는 숫자입니다. `range` 창에는 정확히 하나의 오름차순 `sortBy` 필드가 필요하며 `"current"` 키워드를 수락하지 않습니다.
- `unit`: 선택 사항입니다. `range`와 함께 사용하여 숫자 창 경계를 `sortBy` 필드에 날짜 값이 포함된 시간 범위로 해석합니다. Amazon DocumentDB는 `"year"`, , `"quarter"`, `"month"`, , `"week"`, `"day"``"hour"`, `"second"`및 단위`"minute"`를 지원합니다`"millisecond"`. `unit`를 지정할 때 `range` 경계는 정수여야 합니다.

`window` 문서가 생략되면 기본 기간은 전체 파티션에 걸쳐 `documents: ["unbounded", "unbounded"]`있는 입니다.

> **Note**
>
> `range` 기간 초과 날짜 값을 사용하는 경우 일정에 따라 길이가 모호한 기간 경계를 생성할 수 `"year"`있는 `"month"`, `"quarter"`또는가 아닌 `"day"` (또는 더 작은)`unit`과 같은 고정 기간을 사용하는 것이 좋습니다.

지원되는 창 연산자

Amazon DocumentDB는 `$setWindowFields` 스테이지의 `output` 필드에서 다음 26개의 창 연산자를 지원합니다.

- `$sum`
- `$avg`
- `$count`
- `$push`
- `$addToSet`
- `$stdDevPop`
- `$stdDevSamp`
- `$covariancePop`
- `$covarianceSamp`
- `$first`
- `$last`
- `$firstN`
- `$lastN`
- `$top`
- `$bottom`
- `$topN`
- `$bottomN`
- `$min`
- `$minN`
- `$max`
- `$maxN`
- `$median`
- `$percentile`
- `$rank`
- `$denseRank`
- `$documentNumber`

> **Note**
>
> 연$rand산자는 $setWindowFields 단계의 창 연산자 내에서 지원되지 않습니다.

빈 창

창에 문서가 없는 경우(예: 범위에 값이 없는 range 창) 결과는 연산자에 따라 달라집니다. $count 0, $sum , $push, , $addToSet, $firstN$lastN, $maxN, , $minN$topN, , ,는 빈 배열을 $bottomN 반환하고,는 null 값 배열(요청된 백분위수당 하나)을 $percentile 반환하며, 다른 모든 연산자는를 반환합니다null.

예제(MongoDB 쉘)

다음 예제에서는를 기준으로 문서를 분할하고region,를 기준으로 각 파티션을 정렬하고day, 파티션 시작revenue부터 현재 문서까지 실행 중인 합계를 계산합니다.

샘플 문서 생성

```
db.sales.insertMany([
  { _id: 1, region: "east", day: 1, revenue: 100 },
  { _id: 2, region: "east", day: 2, revenue: 150 },
  { _id: 3, region: "east", day: 3, revenue: 120 },
  { _id: 4, region: "west", day: 1, revenue: 200 },
  { _id: 5, region: "west", day: 2, revenue: 180 }
]);
```

쿼리 예제

```
db.sales.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$region",
      sortBy: { day: 1 },
      output: {
        runningRevenue: {
          $sum: "$revenue",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
```

```
  }
]);
```

출력

```
[
  { "_id": 1, "region": "east", "day": 1, "revenue": 100, "runningRevenue": 100 },
  { "_id": 2, "region": "east", "day": 2, "revenue": 150, "runningRevenue": 250 },
  { "_id": 3, "region": "east", "day": 3, "revenue": 120, "runningRevenue": 370 },
  { "_id": 4, "region": "west", "day": 1, "revenue": 200, "runningRevenue": 200 },
  { "_id": 5, "region": "west", "day": 2, "revenue": 180, "runningRevenue": 380 }
]
```

각 문서는 현재 문서까지 파티션 `revenue` 내의 누적 합계`runningRevenue`인 로 보강됩니다.

코드 예제

`$setWindowFields` 스테이지 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const sales = db.collection('sales');

    await sales.insertMany([
      { _id: 1, region: "east", day: 1, revenue: 100 },
      { _id: 2, region: "east", day: 2, revenue: 150 },
      { _id: 3, region: "east", day: 3, revenue: 120 },
      { _id: 4, region: "west", day: 1, revenue: 200 },
      { _id: 5, region: "west", day: 2, revenue: 180 }
    ]);

    const result = await sales.aggregate([
      {
```

```
        $setWindowFields: {
          partitionBy: "$region",
          sortBy: { day: 1 },
          output: {
            runningRevenue: {
              $sum: "$revenue",
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();

    console.log(result);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']
        sales = db['sales']

        sales.insert_many([
            { '_id': 1, 'region': 'east', 'day': 1, 'revenue': 100 },
            { '_id': 2, 'region': 'east', 'day': 2, 'revenue': 150 },
            { '_id': 3, 'region': 'east', 'day': 3, 'revenue': 120 },
            { '_id': 4, 'region': 'west', 'day': 1, 'revenue': 200 },
            { '_id': 5, 'region': 'west', 'day': 2, 'revenue': 180 }
        ])

        result = list(sales.aggregate([
```

```
            {
                '$setWindowFields': {
                    'partitionBy': '$region',
                    'sortBy': { 'day': 1 },
                    'output': {
                        'runningRevenue': {
                            '$sum': '$revenue',
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))

        print(result)
    finally:
        client.close()

example()
```

$sigmoid

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 `$sigmoid` 연산자는 로지스틱 시그모이드 함수를 숫자 입력`1 / (1 + e^(-x))`에 적용합니다. 함수는 모든 실수를 0에서 1 사이의 값에 매핑하여 S자 곡선을 생성합니다. 원시 점수를 확률 또는 관련성 점수로 변환하는 등 값을 0~1 범위로 정규화해야 하는 경우에 유용합니다.

파라미터

- `expression`: 숫자 값으로 확인되는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$sigmoid` 연산자를 사용하여 로짓 값을 확률로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.predictions.insertMany([
  {_id: 1, logit: -2},
```

```
  {_id: 2, logit: 0},
  {_id: 3, logit: 2}
]);
```

쿼리 예제

```
db.predictions.aggregate([
  { $project: { probability: { $sigmoid: "$logit" } } }
]);
```

출력

```
[
  {_id: 1, probability: 0.11920292202211755},
  {_id: 2, probability: 0.5},
  {_id: 3, probability: 0.8807970779778823}
]
```

코드 예제

$sigmoid 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('predictions');
    const result = await collection.aggregate([
      { $project: { probability: { $sigmoid: "$logit" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
```

```
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['predictions']
        result = list(collection.aggregate([
            {'$project': {'probability': {'$sigmoid': '$logit'}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$sin

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 `$sin` 연산자는 라디안 단위로 측정된 값의 사인을 반환합니다. 집계 파이프라인에서 이를 사용하여 숫자 필드에 대한 삼각 계산을 수행합니다.

파라미터

- `expression`: 라디안 단위의 숫자로 확인되는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$sin` 연산자를 사용하여 각도 값의 사인을 라디안 단위로 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.angles.insertMany([
```

```
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 0.5235987755982988 },
  { "_id": 3, "angle": 1.5707963267948966 }
]);
```

쿼리 예제

```
db.angles.aggregate([
  { $project: {
    "sine": { $sin: "$angle" }
  }}
]);
```

출력

```
[
  { "_id": 1, "sine": 0 },
  { "_id": 2, "sine": 0.49999999999999994 },
  { "_id": 3, "sine": 1 }
]
```

코드 예제

$sin 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('angles');

  const result = await collection.aggregate([
    { $project: {
      "sine": { $sin: "$angle" }
    }}
  ]).toArray();

  console.log(result);
```

```
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['angles']

    result = list(collection.aggregate([
        { '$project': {
            'sine': { '$sin': '$angle' }
        }}
    ]))

    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$sinh

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 `$sinh` 연산자는 값의 하이퍼볼릭 사인을 반환합니다. 집계 파이프라인에서 이를 사용하여 숫자 필드에 대해 하이퍼볼릭 삼각형 계산을 수행합니다.

파라미터

- `expression`: 숫자로 확인되는 표현식입니다.

반환 유형은 `double` 기본적으로 입니다. 입력이 128비트 10진수인 경우 출력도 128비트 10진수입니다.

예제(MongoDB 쉘)

다음 예제에서는 $sinh 연산자를 사용하여 숫자 값의 하이퍼볼릭 사인을 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 1 },
  { "_id": 3, "value": -1 }
]);
```

쿼리 예제

```
db.values.aggregate([
  { $project: {
    "result": { $sinh: "$value" }
  }}
]);
```

출력

```
[
  { "_id": 1, "result": 0 },
  { "_id": 2, "result": 1.1752011936438014 },
  { "_id": 3, "result": -1.1752011936438014 }
]
```

코드 예제

$sinh 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
```

```
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "result": { $sinh: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'result': { '$sinh': '$value' }
        }}
    ]))

    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$skip

Amazon DocumentDB에서는 $skip 연산자를 사용하여 쿼리 결과의 시작점을 오프셋하므로 일치하는 문서의 특정 하위 집합을 검색할 수 있습니다. 이는 결과의 후속 페이지를 검색하려는 페이지 매김 시나리오에서 특히 유용합니다.

파라미터

- skip: 나머지 문서를 반환하기 전에 건너뛸 문서 수입니다.

예제(MongoDB 쉘)

다음 예제에서는 $skip 연산자를 사용하여 컬렉션에서 결과의 두 번째 페이지(문서 11~20)를 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.collection.insert([
  { "name": "Document 1" },
  { "name": "Document 2" },
  { "name": "Document 3" },
  { "name": "Document 4" },
  { "name": "Document 5" },
  { "name": "Document 6" },
  { "name": "Document 7" },
  { "name": "Document 8" },
  { "name": "Document 9" },
  { "name": "Document 10" },
  { "name": "Document 11" },
  { "name": "Document 12" },
  { "name": "Document 13" },
  { "name": "Document 14" },
  { "name": "Document 15" },
  { "name": "Document 16" },
  { "name": "Document 17" },
  { "name": "Document 18" },
  { "name": "Document 19" },
  { "name": "Document 20" }
]);
```

쿼리 예제

```
db.collection.find({}, { "name": 1 })
             .skip(10)
             .limit(10);
```

출력

```
[
  { "_id" : ObjectId("..."), "name" : "Document 11" },
  { "_id" : ObjectId("..."), "name" : "Document 12" },
  { "_id" : ObjectId("..."), "name" : "Document 13" },
  { "_id" : ObjectId("..."), "name" : "Document 14" },
  { "_id" : ObjectId("..."), "name" : "Document 15" },
  { "_id" : ObjectId("..."), "name" : "Document 16" },
  { "_id" : ObjectId("..."), "name" : "Document 17" },
  { "_id" : ObjectId("..."), "name" : "Document 18" },
  { "_id" : ObjectId("..."), "name" : "Document 19" },
  { "_id" : ObjectId("..."), "name" : "Document 20" }
]
```

코드 예제

`$skip` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('collection');

  const results = await collection.find({}, { projection: { name: 1 } })
                                  .skip(10)
                                  .limit(10)
                                  .toArray();

  console.log(results);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient
```

```
def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.mydatabase
    collection = db.collection

    results = list(collection.find({}, {'name': 1})
                   .skip(10)
                   .limit(10))

    print(results)

    client.close()

if __name__ == '__main__':
    main()
```

$slice

$slice 집계 연산자를 사용하면 배열의 시작 또는 끝에서 배열을 통과하여 배열의 하위 집합을 반환할 수 있습니다. 이는 상단 또는 하단 N 항목과 같이 배열 필드의 제한된 수의 항목을 표시하는 데 사용됩니다.

파라미터

- array: 조각화할 배열 필드입니다.
- n: 반환할 요소 수를 지정하는 정수입니다. 양수 값은 배열의 시작 부분부터 시작하고 음수 값은 배열의 끝 부분부터 시작합니다.

예제(MongoDB 쉘)

다음 예제에서는를 사용하여 각 셰프에 대해 처음 두 가지 좋아하는 달콤함을 반환$slice하는 방법을 보여줍니다.

샘플 문서 생성

```
db.sweets.insertMany([
  { "_id" : 1, "name" : "Alvin", "favorites": [ "chocolate", "cake", "toffee",
 "beignets" ] },
```

```
  { "_id" : 2, "name" : "Tom", "favorites": [ "donuts", "pudding", "pie" ] },
  { "_id" : 3, "name" : "Jessica", "favorites": [ "fudge", "smores", "pudding",
 "cupcakes" ] },
  { "_id" : 4, "name" : "Rachel", "favorites": [ "ice cream" ] }
]);
```

쿼리 예제

```
db.sweets.aggregate([
  { $project: { _id: 0, name: 1, topTwoFavorites: { $slice: [ "$favorites", 2 ] } } }
]);
```

출력

```
[
  { name: 'Alvin', topTwoFavorites: [ 'chocolate', 'cake' ] },
  { name: 'Tom', topTwoFavorites: [ 'donuts', 'pudding' ] },
  { name: 'Jessica', topTwoFavorites: [ 'fudge', 'smores' ] },
  { name: 'Rachel', topTwoFavorites: [ 'ice cream' ] }
]
```

이 예제에서는 `$slice` 연산자를 사용하여 각 문서의 `favorites` 배열에서 처음 두 요소를 추출합니다.

코드 예제

`$slice` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('sweets');

  const result = await collection.aggregate([
    { $project: { name: 1, topTwoFavorites: { $slice: ['$favorites', 2] } } }
  ]).toArray();
```

```
  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['sweets']

    result = list(collection.aggregate([
        { '$project': { 'name': 1, 'topTwoFavorites': { '$slice': ['$favorites',
 2] } } }
    ]))

    print(result)
    client.close()

example()
```

$size

`$size` 연산자는 배열 필드 내의 항목 수를 반환하는 데 사용됩니다. 이는 문서에 저장된 배열의 요소 수를 결정하는 데 사용할 수 있습니다.

파라미터

- `field`: 반환하려는 배열 크기의 필드 경로입니다.

예제(MongoDB 쉘)

이 예제에서는 `$size` 연산자를 사용하여 각 사용자가 팔로우하는 팀 수를 반환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.profiles.insertMany([
  { _id: 1, name: "John Doe", teams: ["Acme", "Widgets", "Gadgets"] },
  { _id: 2, name: "Jane Smith", teams: ["Acme", "Gadgets"] },
  { _id: 3, name: "Bob Johnson", teams: ["Acme", "Widgets", "Gadgets"] }
]);
```

쿼리 예제

```
db.profiles.aggregate([
  {
    $project: {
      _id: 0,
      name: 1,
      "numberOfTeams": { $size: "$teams" }
    }
  }
])
```

출력

```
[
  { name: 'John Doe', numberOfTeams: 3 },
  { name: 'Jane Smith', numberOfTeams: 2 },
  { name: 'Bob Johnson', numberOfTeams: 3 }
]
```

코드 예제

`$size` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
```

```
  const profiles = db.collection('profiles');

  const result = await profiles.aggregate([
    {
      $project: {
        item: 1,
        "numberOfTeams": { $size: "$teams" }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    profiles = db.profiles

    result = list(profiles.aggregate([
        {
            '$project': {
                'item': 1,
                'numberOfTeams': { '$size': '$teams' }
            }
        }
    ]))

    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$sort

$sort 집계 단계는 지정된 필드 값을 기반으로 파이프라인의 문서를 정렬합니다. 문서는 제공된 정렬 기준에 따라 오름차순 또는 내림차순으로 정렬됩니다.

파라미터

- field: 정렬 기준으로 사용할 필드 이름입니다.
- order: 오1름차순 또는 내림차순에 -1를 사용합니다.

예제(MongoDB 쉘)

다음 예제에서는 $sort 스테이지를 사용하여 가격을 기준으로 제품을 내림차순으로 주문하는 방법을 보여줍니다.

샘플 문서 생성

```
db.products.insertMany([
  { _id: 1, name: "Laptop", category: "Electronics", price: 1200 },
  { _id: 2, name: "Mouse", category: "Electronics", price: 25 },
  { _id: 3, name: "Desk", category: "Furniture", price: 350 },
  { _id: 4, name: "Chair", category: "Furniture", price: 150 },
  { _id: 5, name: "Monitor", category: "Electronics", price: 400 }
]);
```

쿼리 예제

```
db.products.aggregate([
  { $sort: { price: -1 } }
]);
```

출력

```
[
  { _id: 1, name: 'Laptop', category: 'Electronics', price: 1200 },
  { _id: 5, name: 'Monitor', category: 'Electronics', price: 400 },
  { _id: 3, name: 'Desk', category: 'Furniture', price: 350 },
  { _id: 4, name: 'Chair', category: 'Furniture', price: 150 },
  { _id: 2, name: 'Mouse', category: 'Electronics', price: 25 }
]
```

코드 예제

$sort 집계 단계 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  const result = await collection.aggregate([
    { $sort: { price: -1 } }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['products']

    result = list(collection.aggregate([
        { '$sort': { 'price': -1 } }
    ]))

    print(result)
    client.close()
```

```
example()
```

증분 정렬

증분 정렬은 Amazon DocumentDB가 이미 다중 필드 정렬의 선행 필드를 정렬하는 인덱스를 사용하는 최적화입니다. 그런 다음 Amazon DocumentDB는 동일한 선행 값을 공유하는 각 문서 그룹 내의 나머지 필드만 정렬합니다. Amazon DocumentDB는 메모리의 전체 결과 세트를 하나의 블록(전체 정렬)으로 정렬하는 대신 인덱스 순서로 문서를 읽습니다. 그런 다음 그룹당 하나씩 작고 독립적인 정렬을 수행합니다. 결과는 전체 정렬과 동일합니다. 쿼리 계획과 사용하는 메모리 및 작업량만 변경됩니다.

장점

증분 정렬은 다음과 같은 이점을 제공합니다.

- 메모리 사용량이 적고 디스크 기반 정렬 유출이 적습니다. Amazon DocumentDB는 한 번에 전체 결과 집합 대신 한 번에 하나의 소규모 그룹을 주문합니다.
- 를 사용하여 더 빠른 결과를 얻을 수 있습니다**`limit`**. Amazon DocumentDB는 전체 입력을 읽기 전에 정렬된 문서를 내보낼 수 있습니다. 정렬 후 `limit` (예: 최상위 N 또는 페이지 매김 쿼리)는 충분한 문서를 생성하는 즉시 중지될 수 있습니다.

Amazon DocumentDB가 증분 정렬을 선택하는 경우

예를 들어의 정렬`{ customerId: 1, orderDate: -1 }`과의 인덱스만 주어진 경우 인덱스`{ customerId: 1 }`는 이미에서 정렬한 문서를 반환합니다`customerId`. Amazon DocumentDB는 동일한를 공유하는 각 문서 그룹만 주문`customerId`하면 됩니다`orderDate`. 이미 정렬된 선행 필드(`customerId`)는 미리 정렬된 키입니다. 인덱스가 이미 제공하는 정렬의 양에 따라 Amazon DocumentDB는 다음 세 가지 동작 중 하나를 선택합니다.

- 인덱스는 모든 정렬 키를 정렬합니다. 정렬이 필요하지 않습니다. Amazon DocumentDB는 요청된 순서로 문서를 직접 읽습니다.
- 인덱스는 정렬 키의 선행 접두사를 정렬합니다. Amazon DocumentDB는 증분 정렬을 사용하여 사전 정렬된 각 그룹 내에서 후행 키만 정렬합니다.
- 인덱스는 선행 정렬 키를 정렬하지 않습니다. Amazon DocumentDB는 전체 입력에 대해 전체 정렬을 수행합니다.

요구 사항 및 제한 사항

증분 정렬에는 플래너 버전 3.0(Amazon DocumentDB 8.0.2의 기본 쿼리 플래너)을 실행하는 Amazon DocumentDB 8.0.2 이상이 필요합니다. 플래너 버전에 대한 자세한 내용은 섹션을 참조하세요쿼리 플래너 v3.

Amazon DocumentDB는 부분 인덱스 또는 다중 키(배열) 인덱스를 사용하여 증분 정렬에 대해 접두사를 제공할 수 없습니다. 이러한 경우 Amazon DocumentDB는 전체 정렬로 돌아갑니다. 이는 결과의 정확성에 영향을 미치지 않습니다.

를 사용하여 증분 정렬 확인 **`explain()`**

`explain()`를 사용하여 쿼리가 증분 정렬을 사용하는지 확인합니다. 이기는 계획에서 증분 정렬은 인덱스가 이미 정렬한 선행 정렬 필드를 나열하는 `presortedKeys` 배열이 포함된 `SORT` 단계로 나타납니다. 전체 정렬은 `presortedKeys` 배열이 없는 `SORT` 단계로 나타납니다. `presortedKeys` 필드는 `explain()` 출력에 대한 Amazon DocumentDB별 추가이며 MongoDB 설명 계획에 표시되지 않습니다.

```
db.orders.createIndex({ customerId: 1 })

db.orders.find({}).sort({ customerId: 1, orderDate: -1 }).explain()
```

```
{
    "stage": "SORT",
    "sortPattern": {
        "customerId": 1,
        "orderDate": -1
    },
    "presortedKeys": [
        "customerId"
    ],
    "inputStage": {
        "stage": "IXSCAN",
        "indexName": "customerId_1",
        "direction": "forward"
    }
}
```

$sortArray

버전 8.0의 새로운 기능

Elastic 클러스터에서는 지원되지 않습니다.

Amazon DocumentDB의 $sortArray 연산자는 배열의 요소를 정렬합니다. 스칼라 값 배열의 경우 배열은 요소 값을 기준으로 정렬됩니다. 문서 배열의 경우 배열은 지정된 필드 또는 필드를 기준으로 정렬됩니다. 연산자는 정렬된 배열을 오름차순(1) 또는 내림차순(-1)으로 반환합니다.

> **Note**
>
> $sortArray는 쿼리 플래너 버전 3을 사용하는 집계 파이프라인의 $project 단계에서 지원됩니다.

구문

```
{ $sortArray: { input: <array>, sortBy: <sort specification> } }
```

파라미터

- input: 정렬할 배열입니다.
- sortBy: 정렬 순서입니다. 스칼라 값 배열의 경우 오름차순1에를 지정하고 내림차순-1에를 지정합니다. 문서 배열의 경우 정렬할 각 필드에 { field: -1 } 양식 { field: 1 } 또는의 문서를 지정합니다.

예제(MongoDB 쉘)

다음 예제에서는 $sortArray 연산자를 사용하여 배열의 요소를 오름차순으로 정렬하는 방법을 보여줍니다.

샘플 문서 생성

```
db.miles.insertMany([
  { "_id" : 1, "member_since" : ISODate("1987-01-01T00:00:00Z"), "credit_card" : false,
 "flight_miles" : [ 1205, 2560, 880 ]},
  { "_id" : 2, "member_since" : ISODate("1982-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 1205, 2560, 890, 2780]},
  { "_id" : 3, "member_since" : ISODate("1999-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 1205, 880]}
]);
```

쿼리 예제

```
db.miles.aggregate([
```

```
  {
    $project: {
      _id: 1,
      member_since: 1,
      credit_card: 1,
      sorted_flight_miles: { $sortArray: { input: "$flight_miles", sortBy: 1 } }
    }
  }
]);
```

출력

```
{ "_id" : 1, "member_since" : ISODate("1987-01-01T00:00:00Z"), "credit_card" : false,
 "sorted_flight_miles" : [ 880, 1205, 2560 ] }
{ "_id" : 2, "member_since" : ISODate("1982-01-01T00:00:00Z"), "credit_card" : true,
 "sorted_flight_miles" : [ 890, 1205, 2560, 2780 ] }
{ "_id" : 3, "member_since" : ISODate("1999-01-01T00:00:00Z"), "credit_card" : true,
 "sorted_flight_miles" : [ 880, 1205 ] }
```

이 예제에서 $sortArray 연산자는 flight_miles 배열을 오름차순으로 정렬합니다. 출력의 결과 sorted_flight_miles 필드에는 배열의 요소가 정렬된 순서로 표시됩니다.

코드 예제

$sortArray 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

다음은 Node.js 애플리케이션에서 $sortArray 연산자를 사용하는 예입니다.

```
const { MongoClient } = require('mongodb');

async function sortArray() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('miles');

  const result = await collection.aggregate([
    {
      $project: {
```

```
        _id: 1,
        member_since: 1,
        credit_card: 1,
        sorted_flight_miles: { $sortArray: { input: '$flight_miles', sortBy: 1 } }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

sortArray();
```

Python

다음은 Python 애플리케이션에서 `$sortArray` 연산자를 사용하는 예입니다.

```
from pymongo import MongoClient

def sort_array():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.miles

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'member_since': 1,
                'credit_card': 1,
                'sorted_flight_miles': { '$sortArray': { 'input': '$flight_miles',
 'sortBy': 1 } }
            }
        }
    ]))

    print(result)
    client.close()

sort_array()
```

$sortByCount

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $sortByCount 집계 단계는 지정된 표현식의 값을 기준으로 수신 문서를 그룹화한 다음 각 고유 그룹의 문서 수를 계산하고 내림차순으로 개수를 기준으로 결과를 정렬합니다. 이는 $sum 누적기가 있는 $group 단계와 동일하며 개수 필드에 $sort 단계가 있습니다.

파라미터

- expression: 그룹화 기준의 표현식입니다. 필드 경로(접두사 $) 또는 유효한 집계 표현식일 수 있습니다.

예제(MongoDB 쉘)

다음 예제에서는 $sortByCount 스테이지를 사용하여 범주 필드별로 문서를 계산하고 정렬하는 방법을 보여줍니다.

샘플 문서 생성

```
db.products.insertMany([
  { name: "Widget", category: "electronics" },
  { name: "Gadget", category: "electronics" },
  { name: "Doohickey", category: "electronics" },
  { name: "Apple", category: "food" },
  { name: "Banana", category: "food" },
  { name: "Hammer", category: "tools" }
]);
```

쿼리 예제

```
db.products.aggregate([
  { $sortByCount: "$category" }
]);
```

출력

```
[
  { "_id": "electronics", "count": 3 },
  { "_id": "food", "count": 2 },
  { "_id": "tools", "count": 1 }
```

```
]
```

결과는 내림차순으로 count에 따라 자동으로 정렬됩니다.

코드 예제

$sortByCount 스테이지 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('products');

    const result = await collection.aggregate([
      { $sortByCount: "$category" }
    ]).toArray();

    console.log(result);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
    try:
        db = client['test']
        collection = db['products']

        result = list(collection.aggregate([
            { '$sortByCount': '$category' }
        ]))

        print(result)
    finally:
        client.close()

example()
```

$split

Amazon DocumentDB의 $split 집계 연산자는 지정된 구분 기호를 기반으로 문자열을 하위 문자열 배열로 분할하는 데 사용됩니다. 이는 복잡한 문자열 필드를 구문 분석하고 추가 처리를 위해 개별 구성 요소를 추출하는 데 유용할 수 있습니다.

파라미터

- string: 분할할 문자열입니다.
- delimiter: 입력 문자열을 분할하는 데 사용되는 문자열입니다.

예제(MongoDB 쉘)

이 예제에서는 $split를 사용하여 "데스크" 필드의 구성 요소를 배열로 분리하므로 데이터를 더 쉽게 처리할 수 있습니다.

샘플 문서 생성

```
db.people.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-BVV-021" },
  { "_id": 2, "Desk": "Munich-HGG-32a" },
  { "_id": 3, "Desk": "Cologne-ayu-892.50" },
  { "_id": 4, "Desk": "Dortmund-Hop-78" }
]);
```

쿼리 예제

```
db.people.aggregate([
  { $project: { parts: { $split: ["$Desk", "-"] } } }
]);
```

출력

```
{ "_id" : 1, "parts" : [ "Düsseldorf", "BVV", "021" ] }
{ "_id" : 2, "parts" : [ "Munich", "HGG", "32a" ] }
{ "_id" : 3, "parts" : [ "Cologne", "ayu", "892.50" ] }
{ "_id" : 4, "parts" : [ "Dortmund", "Hop", "78" ] }
```

의 출력은 애플리케이션에서 직원의 정보를 표시하는 데 사용할 수 있는 배열을 $split 생성합니다.

코드 예제

$split 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const result = await db.collection('people').aggregate([
    { $project: { parts: { $split: ['$Desk', '-'] } } }
  ]).toArray();
  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    result = list(db.people.aggregate([
        { '$project': { 'parts': { '$split': ['$Desk', '-'] } } }
    ]))
    print(result)
    client.close()

example()
```

$sqrt

버전 4.0의 새 버전입니다.

Amazon DocumentDB의 `$sqrt` 연산자는 숫자의 제곱근을 계산하는 데 사용됩니다.

파라미터

- `expression`: 인수는 음수가 아닌 숫자로 확인되는 한 유효한 표현식일 수 있습니다.

예제(MongoDB 쉘)

다음 예제에서는 `$sqrt` 연산자를 사용하여 숫자의 제곱근을 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.numbers.insertMany([
  { "_id": 1, "number": 16 },
  { "_id": 2, "number": 36 },
  { "_id": 3, "number": 64 }
]);
```

쿼리 예제

```
db.numbers.aggregate([
  { $project: {
    "_id": 1,
    "square_root": { $sqrt: "$number" }
  }}
```

```
]);
```

출력

```
[
  { _id: 1, square_root: 4 },
  { _id: 2, square_root: 6 },
  { _id: 3, square_root: 8 }
]
```

코드 예제

$sqrt 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
      await client.connect();

      const db = client.db('test');
      const collection = db.collection('numbers');

      const pipeline = [
        {
          $project: {
            _id: 1,
            square_root: { $sqrt: '$number' }
          }
        }
      ];

      const results = await collection.aggregate(pipeline).toArray();

      console.dir(results, { depth: null });

    } finally {
```

```
      await client.close();
    }
}

example().catch(console.error);
```

Python

```
from pymongo import MongoClient

def example():

  client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

  try:
      db = client.test
      collection = db.numbers

      pipeline = [
          {
              "$project": {
                  "_id": 1,
                  "square_root": {
                    "$sqrt": "$number"
                  }
              }
          }
      ]

      results = collection.aggregate(pipeline)

      for doc in results:
          print(doc)

  except Exception as e:
      print(f"An error occurred: {e}")

  finally:
      client.close()

example()
```

$stdDevPop

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $stdDevPop 연산자는 숫자 값의 모집단 표준 편차를 계산합니다. 누적기는 집계 파이프라인의 $group 단계에서 그룹 내 문서 전반의 모집단 표준 편차를 계산합니다. 표현식으로 숫자 배열의 모집단 표준 편차를 계산합니다. 모집단 표준 편차는 N을 제곱(N-1 아님)으로 사용합니다. 숫자가 아닌 값은 무시됩니다. 숫자 값이 없으면를 반환합니다null. 숫자 값이 하나만 있는 경우를 반환합니다0.

파라미터

- expression: 숫자 값 또는 숫자 값 배열로 확인되는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $stdDevPop 연산자를 사용하여 주제당 점수의 모집단 표준 편차를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.scores.insertMany([
  { subject: "math", score: 60 },
  { subject: "math", score: 75 },
  { subject: "math", score: 85 },
  { subject: "math", score: 92 },
  { subject: "math", score: 78 },
  { subject: "science", score: 55 },
  { subject: "science", score: 70 },
  { subject: "science", score: 82 },
  { subject: "science", score: 91 },
  { subject: "science", score: 67 }
]);
```

쿼리 예제

```
db.scores.aggregate([
  { $group: {
      _id: "$subject",
      stdDev: { $stdDevPop: "$score" }
    }}
```

```
]);
```

출력

```
[
  { "_id": "math", "stdDev": 10.75174404457249 },
  { "_id": "science", "stdDev": 12.441864811996632 }
]
```

표현식 사용 예제(MongoDB Shell)

연$stdDevPop산자를 $project 단계 내의 표현식으로 사용하여 배열 필드의 모집단 표준 편차를 계산할 수도 있습니다.

샘플 문서 생성

```
db.experiments.insertMany([
  { _id: 1, measurements: [10, 12, 14, 16, 18] },
  { _id: 2, measurements: [5, 5, 5, 5, 5] },
  { _id: 3, measurements: [2, 4, 6, 8, 10] }
]);
```

쿼리 예제

```
db.experiments.aggregate([
  { $project: {
      stdDev: { $stdDevPop: "$measurements" }
    }}
]);
```

출력

```
[
  { "_id": 1, "stdDev": 2.8284271247461903 },
  { "_id": 2, "stdDev": 0 },
  { "_id": 3, "stdDev": 2.8284271247461903 }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$stdDevPop산자는 $setWindowFields 스테이지에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 문서에 대해 지정된 표현식의 모집단 표준 편차를 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

샘플 문서 생성

```
db.sensorData.insertMany([
  { _id: 1, sensor: "A", time: 1, reading: 4 },
  { _id: 2, sensor: "A", time: 2, reading: 10 },
  { _id: 3, sensor: "A", time: 3, reading: 10 },
  { _id: 4, sensor: "B", time: 1, reading: 10 },
  { _id: 5, sensor: "B", time: 2, reading: 20 }
]);
```

쿼리 예제

다음 예제에서는 문서를 기준으로 분할하고sensor, 각 파티션을 기준으로 정렬하고time, 파티션 시작reading부터 현재 문서까지의 실행 중인 모집단 표준 편차를 반환합니다.

```
db.sensorData.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$sensor",
      sortBy: { time: 1 },
      output: {
        runningStdDev: {
          $stdDevPop: "$reading",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "sensor": "A", "time": 1, "reading": 4, "runningStdDev": 0 },
  { "_id": 2, "sensor": "A", "time": 2, "reading": 10, "runningStdDev": 3 },
  { "_id": 3, "sensor": "A", "time": 3, "reading": 10, "runningStdDev":
 2.8284271247461903 },
  { "_id": 4, "sensor": "B", "time": 1, "reading": 10, "runningStdDev": 0 },
```

```
  { "_id": 5, "sensor": "B", "time": 2, "reading": 20, "runningStdDev": 5 }
]
```

각 문서는 현재 문서까지 파티션 reading 내의 모집단 표준 편차runningStdDev인 로 보강됩니다. 창에 단일 값이 포함된 경우는를 $stdDevPop 반환합니다0.

코드 예제

$stdDevPop 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누적기 사용량(의 $group), 표현식 사용량(의 $project) 및 창 연산자 사용량(의)을 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();
    const db = client.db('test');

    // Accumulator usage: stdDevPop across grouped documents
    const scores = db.collection('scores');
    await scores.insertMany([
      { subject: "math", score: 60 },
      { subject: "math", score: 75 },
      { subject: "math", score: 85 },
      { subject: "math", score: 92 },
      { subject: "math", score: 78 },
      { subject: "science", score: 55 },
      { subject: "science", score: 70 },
      { subject: "science", score: 82 },
      { subject: "science", score: 91 },
      { subject: "science", score: 67 }
    ]);
    const accumulatorResult = await scores.aggregate([
      { $group: {
          _id: "$subject",
```

```
      stdDev: { $stdDevPop: "$score" }
    }}
]).toArray();
console.log('Accumulator result:', accumulatorResult);

// Expression usage: stdDevPop of an array field
const experiments = db.collection('experiments');
await experiments.insertMany([
  { _id: 1, measurements: [10, 12, 14, 16, 18] },
  { _id: 2, measurements: [5, 5, 5, 5, 5] },
  { _id: 3, measurements: [2, 4, 6, 8, 10] }
]);
const expressionResult = await experiments.aggregate([
  { $project: {
      stdDev: { $stdDevPop: "$measurements" }
    }}
]).toArray();
console.log('Expression result:', expressionResult);

// Window operator usage: running stdDevPop within each partition
const sensorData = db.collection('sensorData');
await sensorData.insertMany([
  { _id: 1, sensor: "A", time: 1, reading: 4 },
  { _id: 2, sensor: "A", time: 2, reading: 10 },
  { _id: 3, sensor: "A", time: 3, reading: 10 },
  { _id: 4, sensor: "B", time: 1, reading: 10 },
  { _id: 5, sensor: "B", time: 2, reading: 20 }
]);
const windowResult = await sensorData.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$sensor",
      sortBy: { time: 1 },
      output: {
        runningStdDev: {
          $stdDevPop: "$reading",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]).toArray();
console.log('Window result:', windowResult);
```

```
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']

        # Accumulator usage: stdDevPop across grouped documents
        scores = db['scores']
        scores.insert_many([
            { 'subject': 'math', 'score': 60 },
            { 'subject': 'math', 'score': 75 },
            { 'subject': 'math', 'score': 85 },
            { 'subject': 'math', 'score': 92 },
            { 'subject': 'math', 'score': 78 },
            { 'subject': 'science', 'score': 55 },
            { 'subject': 'science', 'score': 70 },
            { 'subject': 'science', 'score': 82 },
            { 'subject': 'science', 'score': 91 },
            { 'subject': 'science', 'score': 67 }
        ])
        accumulator_result = list(scores.aggregate([
            { '$group': {
                '_id': '$subject',
                'stdDev': { '$stdDevPop': '$score' }
            }}
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: stdDevPop of an array field
        experiments = db['experiments']
```

```
        experiments.insert_many([
            { '_id': 1, 'measurements': [10, 12, 14, 16, 18] },
            { '_id': 2, 'measurements': [5, 5, 5, 5, 5] },
            { '_id': 3, 'measurements': [2, 4, 6, 8, 10] }
        ])
        expression_result = list(experiments.aggregate([
            { '$project': {
                'stdDev': { '$stdDevPop': '$measurements' }
            }}
        ]))
        print('Expression result:', expression_result)

        # Window operator usage: running stdDevPop within each partition
        sensor_data = db['sensorData']
        sensor_data.insert_many([
            { '_id': 1, 'sensor': 'A', 'time': 1, 'reading': 4 },
            { '_id': 2, 'sensor': 'A', 'time': 2, 'reading': 10 },
            { '_id': 3, 'sensor': 'A', 'time': 3, 'reading': 10 },
            { '_id': 4, 'sensor': 'B', 'time': 1, 'reading': 10 },
            { '_id': 5, 'sensor': 'B', 'time': 2, 'reading': 20 }
        ])
        window_result = list(sensor_data.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$sensor',
                    'sortBy': { 'time': 1 },
                    'output': {
                        'runningStdDev': {
                            '$stdDevPop': '$reading',
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()

example()
```

$stdDevSamp

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $stdDevSamp 연산자는 숫자 값의 샘플 표준 편차를 계산합니다. 누적기로서 집계 파이프라인의 $group 단계에서 그룹 내의 문서 간에 샘플 표준 편차를 계산합니다. 표현식으로 숫자 배열의 샘플 표준 편차를 계산합니다. 샘플 표준 편차는 N-1을 제곱으로 사용합니다(Bessel의 수정). 숫자가 아닌 값은 무시됩니다. 숫자 값이 2개 미만인 경우를 반환합니다null.

파라미터

- expression: 숫자 값 또는 숫자 값 배열로 확인되는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $stdDevSamp 연산자를 사용하여 주제당 점수의 샘플 표준 편차를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.scores.insertMany([
  { subject: "math", score: 80 },
  { subject: "math", score: 90 },
  { subject: "math", score: 85 },
  { subject: "math", score: 95 },
  { subject: "science", score: 70 },
  { subject: "science", score: 75 },
  { subject: "science", score: 80 },
  { subject: "science", score: 85 }
]);
```

쿼리 예제

```
db.scores.aggregate([
  { $group: {
      _id: "$subject",
      stdDev: { $stdDevSamp: "$score" }
    }}
]);
```

출력

```
[
  { "_id": "math", "stdDev": 6.454972243679028 },
  { "_id": "science", "stdDev": 6.454972243679028 }
]
```

표현식 사용 예제(MongoDB Shell)

연$stdDevSamp산자를 $project 단계 내의 표현식으로 사용하여 배열 필드의 샘플 표준 편차를 계산할 수도 있습니다.

샘플 문서 생성

```
db.experiments.insertMany([
  { _id: 1, measurements: [10, 12, 14, 16, 18] },
  { _id: 2, measurements: [5, 5, 5, 5, 5] },
  { _id: 3, measurements: [2, 4, 6, 8, 10] }
]);
```

쿼리 예제

```
db.experiments.aggregate([
  { $project: {
      stdDev: { $stdDevSamp: "$measurements" }
    }}
]);
```

출력

```
[
  { "_id": 1, "stdDev": 3.1622776601683795 },
  { "_id": 2, "stdDev": 0 },
  { "_id": 3, "stdDev": 3.1622776601683795 }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$stdDevSamp산자는 $setWindowFields 스테이지에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 문서에 대해 지정된 표현식의 샘플 표준 편차를 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

샘플 문서 생성

```
db.sensorReadings.insertMany([
  { _id: 1, sensor: "A", time: 1, value: 2 },
  { _id: 2, sensor: "A", time: 2, value: 4 },
  { _id: 3, sensor: "A", time: 3, value: 6 },
  { _id: 4, sensor: "B", time: 1, value: 10 },
  { _id: 5, sensor: "B", time: 2, value: 12 }
]);
```

쿼리 예제

다음 예제에서는를 기준으로 문서를 분할하고sensor,를 기준으로 각 파티션을 정렬하고time, 파티션 시작value부터 현재 문서까지의 실행 중인 샘플 표준 편차를 반환합니다.

```
db.sensorReadings.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$sensor",
      sortBy: { time: 1 },
      output: {
        runningStdDev: {
          $stdDevSamp: "$value",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "sensor": "A", "time": 1, "value": 2, "runningStdDev": null },
  { "_id": 2, "sensor": "A", "time": 2, "value": 4, "runningStdDev":
 1.4142135623730951 },
  { "_id": 3, "sensor": "A", "time": 3, "value": 6, "runningStdDev": 2 },
  { "_id": 4, "sensor": "B", "time": 1, "value": 10, "runningStdDev": null },
  { "_id": 5, "sensor": "B", "time": 2, "value": 12, "runningStdDev":
 1.4142135623730951 }
]
```

각 문서는 현재 문서까지 파티션 value 내의 샘플 표준 편차runningStdDev인 로 보강됩니다. 창에 2개 미만의 값(각 파티션의 첫 번째 문서)이 포함된 경우는를 $stdDevSamp 반환합니다null.

코드 예제

$stdDevSamp 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누적기 사용량(의 $group), 표현식 사용량(의 $project) 및 창 연산자 사용량(의)을 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();
    const db = client.db('test');

    // Accumulator usage: stdDevSamp across grouped documents
    const scores = db.collection('scores');
    await scores.insertMany([
      { subject: "math", score: 80 },
      { subject: "math", score: 90 },
      { subject: "math", score: 85 },
      { subject: "math", score: 95 },
      { subject: "science", score: 70 },
      { subject: "science", score: 75 },
      { subject: "science", score: 80 },
      { subject: "science", score: 85 }
    ]);
    const accumulatorResult = await scores.aggregate([
      { $group: {
          _id: "$subject",
          stdDev: { $stdDevSamp: "$score" }
        }}
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);
```

```
    // Expression usage: stdDevSamp of an array field
    const experiments = db.collection('experiments');
    await experiments.insertMany([
      { _id: 1, measurements: [10, 12, 14, 16, 18] },
      { _id: 2, measurements: [5, 5, 5, 5, 5] },
      { _id: 3, measurements: [2, 4, 6, 8, 10] }
    ]);
    const expressionResult = await experiments.aggregate([
      { $project: {
          stdDev: { $stdDevSamp: "$measurements" }
        }}
    ]).toArray();
    console.log('Expression result:', expressionResult);

    // Window operator usage: running stdDevSamp within each partition
    const sensorReadings = db.collection('sensorReadings');
    await sensorReadings.insertMany([
      { _id: 1, sensor: "A", time: 1, value: 2 },
      { _id: 2, sensor: "A", time: 2, value: 4 },
      { _id: 3, sensor: "A", time: 3, value: 6 },
      { _id: 4, sensor: "B", time: 1, value: 10 },
      { _id: 5, sensor: "B", time: 2, value: 12 }
    ]);
    const windowResult = await sensorReadings.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$sensor",
          sortBy: { time: 1 },
          output: {
            runningStdDev: {
              $stdDevSamp: "$value",
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } finally {
    await client.close();
  }
}
```

```
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']

        # Accumulator usage: stdDevSamp across grouped documents
        scores = db['scores']
        scores.insert_many([
            { 'subject': 'math', 'score': 80 },
            { 'subject': 'math', 'score': 90 },
            { 'subject': 'math', 'score': 85 },
            { 'subject': 'math', 'score': 95 },
            { 'subject': 'science', 'score': 70 },
            { 'subject': 'science', 'score': 75 },
            { 'subject': 'science', 'score': 80 },
            { 'subject': 'science', 'score': 85 }
        ])
        accumulator_result = list(scores.aggregate([
            { '$group': {
                '_id': '$subject',
                'stdDev': { '$stdDevSamp': '$score' }
            }}
        ]))
        print('Accumulator result:', accumulator_result)

        # Expression usage: stdDevSamp of an array field
        experiments = db['experiments']
        experiments.insert_many([
            { '_id': 1, 'measurements': [10, 12, 14, 16, 18] },
            { '_id': 2, 'measurements': [5, 5, 5, 5, 5] },
            { '_id': 3, 'measurements': [2, 4, 6, 8, 10] }
        ])
        expression_result = list(experiments.aggregate([
            { '$project': {
```

```
                'stdDev': { '$stdDevSamp': '$measurements' }
            }}
        ]))
        print('Expression result:', expression_result)

        # Window operator usage: running stdDevSamp within each partition
        sensor_readings = db['sensorReadings']
        sensor_readings.insert_many([
            { '_id': 1, 'sensor': 'A', 'time': 1, 'value': 2 },
            { '_id': 2, 'sensor': 'A', 'time': 2, 'value': 4 },
            { '_id': 3, 'sensor': 'A', 'time': 3, 'value': 6 },
            { '_id': 4, 'sensor': 'B', 'time': 1, 'value': 10 },
            { '_id': 5, 'sensor': 'B', 'time': 2, 'value': 12 }
        ])
        window_result = list(sensor_readings.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$sensor',
                    'sortBy': { 'time': 1 },
                    'output': {
                        'runningStdDev': {
                            '$stdDevSamp': '$value',
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        print('Window result:', window_result)

    finally:
        client.close()

example()
```

$strLenBytes

Amazon DocumentDB의 `$strLenBytes` 연산자는 문자열의 길이를 바이트 단위로 결정하는 데 사용됩니다. 이는 문자열 필드의 스토리지 크기를 이해해야 할 때, 특히 문자당 1바이트 이상을 사용할 수 있는 유니코드 문자를 처리할 때 유용합니다.

파라미터

- expression: 길이를 계산할 문자열 표현식입니다.

예제(MongoDB 쉘)

이 예제에서는 $strLenBytes 연산자를 사용하여 문자열 필드의 길이를 바이트 단위로 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.people.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-BVV-021" },
  { "_id": 2, "Desk": "Munich-HGG-32a" },
  { "_id": 3, "Desk": "Cologne-ayu-892.50" },
  { "_id": 4, "Desk": "Dortmund-Hop-78" }
]);
```

쿼리 예제

```
db.people.aggregate([
  {
    $project: {
      "Desk": 1,
      "length": { $strLenBytes: "$Desk" }
    }
  }
])
```

출력

```
{ "_id" : 1, "Desk" : "Düsseldorf-BVV-021", "length" : 19 }
{ "_id" : 2, "Desk" : "Munich-HGG-32a", "length" : 14 }
{ "_id" : 3, "Desk" : "Cologne-ayu-892.50", "length" : 18 }
{ "_id" : 4, "Desk" : "Dortmund-Hop-78", "length" : 15 }
```

"Düsseldorf-BVV-021" 문자열의 길이는 19바이트이며, 이는 2바이트를 차지하는 유니코드 문자 ""로 인한 코드 포인트(18) 수와 다릅니다.

코드 예제

$strLenBytes 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const result = await collection.aggregate([
    {
      $project: {
        "Desk": 1,
        "length": { $strLenBytes: "$Desk" }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.people

    result = list(collection.aggregate([
        {
            '$project': {
                "Desk": 1,
                "length": { "$strLenBytes": "$Desk" }
            }
```

```
        }
    ]))

    print(result)
    client.close()

example()
```

$strLenCP

Amazon DocumentDB의 $strLenCP 연산자는 코드 포인트(유니코드 문자)의 문자열 표현식 길이를 결정하는 데 사용됩니다. 이는 바이트 수가 아닌 문자열의 문자 수를 알아야 할 때 유용합니다.

파라미터

- expression: 코드 포인트의 길이를 반환할 문자열 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $strLenCP 연산자를 사용하여 유니코드 문자가 있는 문자열의 길이를 결정하는 방법을 보여줍니다.

샘플 문서 생성

```
db.people.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-BVV-021" },
  { "_id": 2, "Desk": "Munich-HGG-32a" },
  { "_id": 3, "Desk": "Cologne-ayu-892.50" },
  { "_id": 4, "Desk": "Dortmund-Hop-78" }
])
```

쿼리 예제

```
db.people.aggregate([
  {
    $project: {
      "Desk": 1,
      "length": { $strLenCP: "$Desk" }
    }
  }
```

```
])
```

출력

```
{ "_id" : 1, "Desk" : "Düsseldorf-BVV-021", "length" : 18 }
{ "_id" : 2, "Desk" : "Munich-HGG-32a", "length" : 14 }
{ "_id" : 3, "Desk" : "Cologne-ayu-892.50", "length" : 18 }
{ "_id" : 4, "Desk" : "Dortmund-Hop-78", "length" : 15 }
```

유니코드 문자(")가 포함된 "Düsseldorf-BVV-021" 문자열의 길이 측정 차이를 확인합니다. $strLenCP 연산자는 유니코드 문자 수를 올바르게 계산하고 $strLenBytes 연산자는 바이트 수를 계산합니다.

코드 예제

$strLenCP 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

다음은 MongoDB 드라이버와 함께 Node.js 애플리케이션에서 $strLenCP 연산자를 사용하는 예입니다.

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const result = await collection.aggregate([
    {
      $project: {
        "Desk": 1,
        "length": { $strLenCP: "$Desk" }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
```

```
}

example();
```

Python

다음은 PyMongo 드라이버와 함께 Python 애플리케이션에서 $strLenCP 연산자를 사용하는 예입니다.

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.people

    result = list(collection.aggregate([
        {
            '$project': {
                "Desk": 1,
                "length": { "$strLenCP": "$Desk" }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$strcasecmp

Amazon DocumentDB의 $strcasecmp 연산자는 두 문자열 간에 대/소문자를 구분하지 않는 비교를 수행합니다. 두 입력 문자열의 어휘 비교를 나타내는 정수 값을 반환하며 대소문자 차이를 무시합니다.

파라미터

- string1: 비교할 첫 번째 문자열입니다.
- string2: 비교할 두 번째 문자열입니다.

예제(MongoDB 쉘)

이 예제에서는 연$strcasecmp산자를 사용하여 사례 차이를 무시하고 people 컬렉션의 데스크 위치 문자열을 비교하는 방법을 보여줍니다.

샘플 문서 생성

```
db.people.insertMany([
  { "_id": 1, "Desk": "mke233-wi" },
  { "_id": 2, "Desk": "MKE233-WI" },
  { "_id": 3, "Desk": "mke233-wi" }
]);
```

쿼리 예제

```
db.people.aggregate([
  {
    $project: {
      item: 1,
      compare: { $strcasecmp: ["$Desk", "mke233-wi"] }
    }
  }
]);
```

출력

```
{ "_id" : 1, "compare" : 0 }
{ "_id" : 2, "compare" : 0 }
{ "_id" : 3, "compare" : 0 }
```

출력은 "Desk" 필드와 문자열 간의 비교가 세 문서 모두에 0 대해를 "mke233-wi" 반환한다는 것을 보여줍니다. 이는 대/소문자를 무시할 때 문자열이 동일함을 나타냅니다.

코드 예제

$strcasecmp 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const result = await collection.aggregate([
    {
      $project: {
        item: 1,
        compare: { $strcasecmp: ["$Desk", "mke233-wi"] }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.people

    result = list(collection.aggregate([
        {
            '$project': {
                'item': 1,
                'compare': { '$strcasecmp': ['$Desk', 'mke233-wi'] }
            }
        }
    ]))
```

```
    print(result)

    client.close()

example()
```

$substr

Amazon DocumentDB의 $substr 연산자는 지정된 문자열에서 하위 문자열을 추출하는 데 사용됩니다. 이는 바이트 범위가 아닌 문자 범위를 기반으로 하위 문자열을 정의해야 하는 경우에 특히 유용합니다. 이는 문자를 나타내는 데 사용되는 바이트 수가 다를 수 있는 유니코드 문자열을 처리할 때 특히 중요합니다.

파라미터

- string: 하위 문자열을 추출할 입력 문자열입니다.
- start: 추출할 하위 문자열의 시작 위치(0 기반)입니다. 음수가 아닌 정수 표현식일 수 있습니다.
- length: 추출된 하위 문자열의 문자 수입니다. 음수가 아닌 정수 표현식일 수 있습니다.

예제(MongoDB 쉘)

이 예제에서는를 사용하여 직원의 사무실 위치에서 상태 약어를 추출$substr하는 방법을 보여줍니다.

샘플 문서 생성

```
db.people.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-NRW-021" },
  { "_id": 2, "Desk": "Bremerhaven-HBB-32a" },
  { "_id": 3, "Desk": "Norderstedt-SHH-892.50" },
  { "_id": 4, "Desk": "Brandenburg-BBB-78" }
]);
```

쿼리 예제

```
db.people.aggregate([
  {
    $project: {
      "state": { $substr: ["$Desk", 12, 3] }
    }
```

```
  }
])
```

출력

```
{ "_id": 1, "state": "NRW" },
{ "_id": 2, "state": "HBB" },
{ "_id": 3, "state": "SHH" },
{ "_id": 4, "state": "BBB" }
```

코드 예제

$substr 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require("mongodb");

async function example() {
  const client = await MongoClient.connect("mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false");
  const db = client.db("test");
  const collection = db.collection("people");

  const result = await collection.aggregate([
    {
      $project: {
        "state": { $substrCP: ["$Desk", 12, 3] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient
```

```
def example():
    client = MongoClient("mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false")
    db = client["test"]
    collection = db["people"]

    result = list(collection.aggregate([
        {
            "$project": {
                "state": { "$substrCP": ["$Desk", 12, 3] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$substrBytes

Amazon DocumentDB의 $substrBytes 연산자는 지정된 바이트 범위를 기반으로 문자열에서 하위 문자열을 추출하는 데 사용됩니다. 이 연산자는 문자열에서 하위 문자열을 추출해야 할 때 유용하며 문자열의 각 문자를 나타내는 데 필요한 바이트 수가 중요합니다.

유니코드 코드 포인트 수에서 작동하는와 달리 $substrCP는 문자열의 문자를 나타내는 데 필요한 바이트 수에서 $substrBytes 작동합니다. 비 ASCII 문자가 포함된 문자열을 사용할 때 특히 유용할 수 있습니다. 이러한 문자를 나타내려면 1바이트 이상이 필요할 수 있기 때문입니다.

참고:$substr는 버전 3.4 이후 더 이상 사용되지 않습니다. $substr는 이제의 별칭입니다$substrBytes.

파라미터

- string: 하위 문자열을 추출할 입력 문자열입니다.
- startByte: 추출할 하위 문자열의 0 기반 시작 바이트 위치입니다. 음수 값을 사용하여 문자열 끝에서 위치를 지정할 수 있습니다.
- length: 추출할 하위 문자열의 바이트 수입니다.

예제(MongoDB 쉘)

이 예제에서는 $substrBytes를 사용하여 비 ASCII 문자가 포함된 문자열에서 하위 문자열을 추출합니다.

샘플 문서 생성

```
db.people.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-NRW-021" },
  { "_id": 2, "Desk": "Bremerhaven-HBB-32a" },
  { "_id": 3, "Desk": "Norderstedt-SHH-892.50" },
  { "_id": 4, "Desk": "Brandenburg-BBB-78" }
]);
```

쿼리 예제

```
db.people.aggregate([
  {
    $project: {
      "state": { $substrBytes: [ "$Desk", 12, 3] }
    }
  }
])
```

출력

```
{ "_id": 1, "state": "NRW" },
{ "_id": 2, "state": "HBB" },
{ "_id": 3, "state": "SHH" },
{ "_id": 4, "state": "BBB" }
```

이 예제에서는 Desk 필드의 12바이트부터 시작하여 3바이트 하위 문자열을 추출$substrBytes하는 데를 사용합니다. 이렇게 하면 문자열에 ASCII가 아닌 문자가 포함될 수 있더라도 2자 상태 약어를 추출할 수 있습니다.

코드 예제

$substrBytes 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const people = db.collection('people');

  const result = await people.aggregate([
    {
      $project: {
        "state": { $substrBytes: ["$Desk", 12, 3] }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    people = db.people

    result = list(people.aggregate([
        {
            '$project': {
                "state": { '$substrBytes': ["$Desk", 12, 3] }
            }
        }
    ]))

    print(result)
    client.close()
```

```
example()
```

$substrCP

Amazon DocumentDB의 $substrCP 연산자는 문자열에서 하위 문자열을 추출하는 데 사용되며, 여기서 하위 문자열은 UTF-8 코드 포인트(CP)의 범위로 지정됩니다. 이 연산자는 문자의 기본 바이트 표현에 대해 걱정할 필요 없이 하위 문자열을 추출할 수 있으므로 유니코드 문자열로 작업할 때 특히 유용합니다.

바이트 위치에서 작동하는 $substrBytes 연산자와 달리 연$substrCP산자는 코드 포인트 위치에서 작동합니다. 따라서 코드 포인트 수가 바이트 또는 문자 수와 일치하지 않을 수 있으므로 ASCII가 아닌 문자가 포함된 문자열로 더 쉽게 작업할 수 있습니다.

파라미터

- string: 하위 문자열을 추출할 입력 문자열입니다.
- start: 하위 문자열을 추출할 시작 코드 포인트 위치(0 기반)입니다.
- length: 추출할 코드 포인트 수입니다.

예제(MongoDB 쉘)

이 예제에서는 $substrCP 연산자를 사용하여 직원의 사무실 위치가 포함된 문자열에서 상태 약어를 추출합니다.

샘플 문서 생성

```
db.people.insert([
  { "_id": 1, "first_name": "Jane", "last_name": "Doe", "Desk": "12 Main St,
 Minneapolis, MN 55401" },
  { "_id": 2, "first_name": "John", "last_name": "Doe", "Desk": "456 Oak Rd, New
 Orleans, LA 70032" },
  { "_id": 3, "first_name": "Steve", "last_name": "Smith", "Desk": "789 Elm Ln,
 Bakersfield, CA 93263" }
]);
```

쿼리 예제

```
db.people.aggregate([
```

```
  {
    $project: {
      "state": { $substrCP: ["$Desk", 25, 2] }
    }
  }
]);
```

출력

```
{ "_id" : 1, "state" : "MN" }
{ "_id" : 2, "state" : "LA" }
{ "_id" : 3, "state" : "CA" }
```

이 예제에서는 상태 약어가 Desk 필드의 25번째 코드 포인트에서 시작되고 2개의 코드 포인트 길이임을 알고 있습니다. $substrCP 연산자를 사용하면 문자열의 기본 바이트 표현에 대해 걱정할 필요 없이 상태 약어를 추출할 수 있습니다.

코드 예제

$substrCP 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findStates() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const result = await db.collection('people').aggregate([
    {
      $project: {
        "state": { $substrCP: ["$Desk", 25, 2] }
      }
    }
  ]).toArray();
  console.log(result);
  client.close();
}

findStates();
```

Python

```
from pymongo import MongoClient

def find_states():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    result = list(db.people.aggregate([
        {
            '$project': {
                'state': { '$substrCP': ['$Desk', 25, 2] }
            }
        }
    ]))
    print(result)
    client.close()

find_states()
```

Node.js 및 Python 예제 모두에서 `$substrCP` 연산자를 사용하여 MongoDB Shell 예제와 마찬가지로 `Desk` 필드에서 상태 약어를 추출합니다.

$subtract

Amazon DocumentDB의 `$subtract` 연산자는 값을 빼는 데 사용됩니다. 날짜, 숫자 또는 둘의 조합을 빼는 데 사용할 수 있습니다. 이 연산자는 두 날짜 간의 차이를 계산하거나 숫자에서 값을 빼는 데 유용합니다.

파라미터

- `expression1`: 뺄 첫 번째 값입니다.
- `expression2`:에서 뺄 두 번째 값입니다`<expression1>`.

예제(MongoDB 쉘)

다음 예제에서는 `$subtract` 연산자를 사용하여 두 날짜 간의 차이를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.dates.insert([
  {
  "_id": 1,
  "startDate": ISODate("2023-01-01T00:00:00Z"),
  "endDate": ISODate("2023-01-05T12:00:00Z")
  }
]);
```

쿼리 예제

```
db.dates.aggregate([
  {
    $project: {
      _id: 1,
      durationDays: {
        $divide: [
          { $subtract: ["$endDate", "$startDate"] },
          1000 * 60 * 60 * 24  // milliseconds in a day
        ]
      }
    }
  }
]);
```

출력

```
[ { _id: 1, durationDays: 4.5 } ]
```

이 예제에서는 $subtract 연산자를 사용하여 $endDate와 일 단위의 차이를 계산$startDate합니다.

코드 예제

$subtract 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');

  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('dates');

    const pipeline = [
      {
        $project: {
          _id: 1,
          durationDays: {
            $divide: [
              { $subtract: ["$endDate", "$startDate"] },
              1000 * 60 * 60 * 24  // Convert milliseconds to days
            ]
          }
        }
      }
    ];

    const results = await collection.aggregate(pipeline).toArray();

    console.dir(results, { depth: null });

  } finally {
    await client.close();
  }
}

example().catch(console.error);
```

Python

```
from datetime import datetime, timedelta
from pymongo import MongoClient

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client.test
        collection = db.dates

        pipeline = [
            {
                "$project": {
                    "_id": 1,
                    "durationDays": {
                        "$divide": [
                            { "$subtract": ["$endDate", "$startDate"] },
                            1000 * 60 * 60 * 24  # Convert milliseconds to days
                        ]
                    }
                }
            }
        ]

        results = collection.aggregate(pipeline)

        for doc in results:
            print(doc)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        client.close()

example()
```

$sum

Amazon DocumentDB의 $sum 연산자는 그룹의 각 문서에 대해 지정된 표현식의 합계를 반환합니다. 집계 파이프라인의 $group 단계에서 일반적으로 합계 계산을 수행하는 데 사용되는 그룹 누적기 연산자입니다.

파라미터

- `expression`: 합계할 숫자 표현식입니다. 필드 경로, 표현식 또는 상수일 수 있습니다.

예제(MongoDB 쉘)

다음 예제에서는 `$sum` 운영자를 사용하여 각 제품의 총 매출을 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.sales.insertMany([
  { product: "abc", price: 10, quantity: 2 },
  { product: "abc", price: 10, quantity: 3 },
  { product: "xyz", price: 20, quantity: 1 },
  { product: "xyz", price: 20, quantity: 5 }
]);
```

쿼리 예제

```
db.sales.aggregate([
  { $group: {
      _id: "$product",
      totalSales: { $sum: { $multiply: [ "$price", "$quantity" ] } }
    }}
]);
```

출력

```
[
  { "_id": "abc", "totalSales": 50 },
  { "_id": "xyz", "totalSales": 120 }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연`$sum`산자는 `$setWindowFields` 스테이지에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 문서에 대해 지정된 표현식의 합계를 반환합니다. `output` 필드 아래에 연산자를 지정하고 선택적으로 `window` 문서로 창 경계를 정의합니다.

샘플 문서 생성

```
db.dailySales.insertMany([
```

```
  { _id: 1, product: "abc", day: 1, amount: 10 },
  { _id: 2, product: "abc", day: 2, amount: 20 },
  { _id: 3, product: "abc", day: 3, amount: 15 },
  { _id: 4, product: "xyz", day: 1, amount: 30 },
  { _id: 5, product: "xyz", day: 2, amount: 25 }
]);
```

쿼리 예제

다음 예제에서는를 기준으로 문서를 분할하고product,를 기준으로 각 파티션을 정렬하고day, 파티션 시작amount부터 현재 문서를 통해의 실행 합계를 반환합니다.

```
db.dailySales.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$product",
      sortBy: { day: 1 },
      output: {
        runningTotal: {
          $sum: "$amount",
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "product": "abc", "day": 1, "amount": 10, "runningTotal": 10 },
  { "_id": 2, "product": "abc", "day": 2, "amount": 20, "runningTotal": 30 },
  { "_id": 3, "product": "abc", "day": 3, "amount": 15, "runningTotal": 45 },
  { "_id": 4, "product": "xyz", "day": 1, "amount": 30, "runningTotal": 30 },
  { "_id": 5, "product": "xyz", "day": 2, "amount": 25, "runningTotal": 55 }
]
```

각 문서는 현재 문서까지 파티션 amount 내의 누적 합계runningTotal인 로 보강됩니다.

코드 예제

$sum 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누산기 사용량()과 창 연산자 사용량($group)을 모두 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const db = client.db('test');

    // Accumulator usage: sum per group
    const sales = db.collection('sales');
    await sales.insertMany([
      { product: "abc", price: 10, quantity: 2 },
      { product: "abc", price: 10, quantity: 3 },
      { product: "xyz", price: 20, quantity: 1 },
      { product: "xyz", price: 20, quantity: 5 }
    ]);
    const accumulatorResult = await sales.aggregate([
      { $group: {
          _id: "$product",
          totalSales: { $sum: { $multiply: [ "$price", "$quantity" ] } }
        }}
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Window operator usage: running total within each partition
    const dailySales = db.collection('dailySales');
    await dailySales.insertMany([
      { _id: 1, product: "abc", day: 1, amount: 10 },
      { _id: 2, product: "abc", day: 2, amount: 20 },
      { _id: 3, product: "abc", day: 3, amount: 15 },
      { _id: 4, product: "xyz", day: 1, amount: 30 },
      { _id: 5, product: "xyz", day: 2, amount: 25 }
    ]);
    const windowResult = await dailySales.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$product",
```

```
          sortBy: { day: 1 },
          output: {
            runningTotal: {
              $sum: "$amount",
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client.test

        # Accumulator usage: sum per group
        sales = db.sales
        sales.insert_many([
            { 'product': 'abc', 'price': 10, 'quantity': 2 },
            { 'product': 'abc', 'price': 10, 'quantity': 3 },
            { 'product': 'xyz', 'price': 20, 'quantity': 1 },
            { 'product': 'xyz', 'price': 20, 'quantity': 5 }
        ])
```

```
        accumulator_result = list(sales.aggregate([
            { '$group': {
                '_id': '$product',
                'totalSales': { '$sum': { '$multiply': [ '$price', '$quantity' ] } }
            }}
        ]))
        pprint(accumulator_result)

        # Window operator usage: running total within each partition
        daily_sales = db.dailySales
        daily_sales.insert_many([
            { '_id': 1, 'product': 'abc', 'day': 1, 'amount': 10 },
            { '_id': 2, 'product': 'abc', 'day': 2, 'amount': 20 },
            { '_id': 3, 'product': 'abc', 'day': 3, 'amount': 15 },
            { '_id': 4, 'product': 'xyz', 'day': 1, 'amount': 30 },
            { '_id': 5, 'product': 'xyz', 'day': 2, 'amount': 25 }
        ])
        window_result = list(daily_sales.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$product',
                    'sortBy': { 'day': 1 },
                    'output': {
                        'runningTotal': {
                            '$sum': '$amount',
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        pprint(window_result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$switch

버전 4.0의 새 버전입니다.

Elastic 클러스터에서는 지원되지 않습니다.

연$switch산자는 Amazon DocumentDB의 조건식 연산자로, 사례 표현식 목록을 평가하고 true로 평가되는 첫 번째 사례의 값을 반환하거나 사례 표현식이 true가 아닌 경우 기본값을 반환할 수 있습니다.

파라미터

- branches: 각각 평가할 부울 표현식을 포함하는 대/소문자 필드와 대/소문자 표현식이 true인 경우 반환할 값을 포함하는 대/소문자 필드가 있는 문서 배열입니다.
- default: (선택 사항) true인 대소문자 표현식이 없는 경우 반환할 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 $switch 운영자를 사용하여 주문 합계를 기준으로 주문의 배송 비용을 결정하는 방법을 보여줍니다.

샘플 문서 생성

```
db.orders.insertMany([
  { _id: 1, total: 50 },
  { _id: 2, total: 150 },
  { _id: 3, total: 250 }
]);
```

쿼리 예제

```
db.orders.aggregate([
  {
    $project: {
      _id: 1,
      total: 1,
      shippingCost: {
        $switch: {
          branches: [
            { case: { $lte: ["$total", 100] }, then: 5 },
            { case: { $lte: ["$total", 200] }, then: 10 },
            { case: { $gt: ["$total", 200] }, then: 15 }
```

```
          ],
          default: 0
        }
      }
    }
  }
])
```

출력

```
[
  {
    "_id": 1,
    "total": 50,
    "shippingCost": 5
  },
  {
    "_id": 2,
    "total": 150,
    "shippingCost": 10
  },
  {
    "_id": 3,
    "total": 250,
    "shippingCost": 15
  }
]
```

코드 예제

$switch 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('orders');
```

```
  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        total: 1,
        shippingCost: {
          $switch: {
            branches: [
              { case: { $lte: ['$total', 100] }, then: 5 },
              { case: { $lte: ['$total', 200] }, then: 10 },
              { case: { $gt: ['$total', 200] }, then: 15 }
            ],
            default: 0
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.orders

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'total': 1,
                'shippingCost': {
                    '$switch': {
```

```
                        'branches': [
                            { 'case': { '$lte': ['$total', 100] }, 'then': 5 },
                            { 'case': { '$lte': ['$total', 200] }, 'then': 10 },
                            { 'case': { '$gt': ['$total', 200] }, 'then': 15 }
                        ],
                        'default': 0
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

if __name__ == '__main__':
    main()
```

$tan

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $tan 연산자는 라디안 단위로 측정된 값의 탄젠트를 반환합니다.

입력 표현식은 숫자 값으로 확인되어야 합니다. 값이 도인 경우에 전달하기 $degreesToRadians 전에를 적용합니다$tan.

반환 유형은 double 기본적으로 입니다. 입력이 128비트 10진수인 경우 출력도 128비트 10진수입니다.

파라미터

- expression: 라디안 단위의 숫자로 확인되는 표현식입니다.

동작

null, NaN 및 +/- 무한대

예제	결과
{ $tan: NaN }	NaN

예제	결과
`{ $tan: null }`	`null`
`{ $tan: Infinity }` 또는 `{ $tan: -Infinity }`	오류가 발생합니다.

입력이 `null` 이거나 참조된 필드가 누락된 경우는를 `$tan` 반환합니다`null`. 의 입력은를 `NaN` 생성합니다`NaN`. 양수 또는 음수 무한대는 해당 값에 탄젠트가 정의되지 않았기 때문에 오류가 발생합니다.

예제(MongoDB 쉘)

다음 예제에서는 `$tan` 연산자를 사용하여 각도의 접선을 라디안 단위로 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.angles.insertMany([
  { "_id": 1, "angle": 0 },
  { "_id": 2, "angle": 0.7853981633974483 },
  { "_id": 3, "angle": -0.7853981633974483 }
]);
```

쿼리 예제

```
db.angles.aggregate([
  { $project: {
    "tangent": { $tan: "$angle" }
  }}
]);
```

출력

```
[
  { "_id": 1, "tangent": 0 },
  { "_id": 2, "tangent": 0.9999999999999999 },
  { "_id": 3, "tangent": -0.9999999999999999 }
]
```

코드 예제

`$tan` 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('angles');

  const result = await collection.aggregate([
    { $project: {
      "tangent": { $tan: "$angle" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['angles']

    result = list(collection.aggregate([
        { '$project': {
            'tangent': { '$tan': '$angle' }
        }}
    ]))

    print(result)
    client.close()
```

```
if __name__ == "__main__":
    main()
```

$tanh

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $tanh 연산자는 값의 하이퍼볼릭 탄젠트를 반환합니다.

입력 표현식은 숫자 값으로 확인되어야 합니다.

반환 유형은 double 기본적으로 입니다. 입력이 128비트 10진수인 경우 출력도 128비트 10진수입니다.

파라미터

- expression: 숫자로 확인되는 표현식입니다.

동작

null, NaN 및 +/- 무한대

예제	결과
{ $tanh: NaN }	NaN
{ $tanh: null }	null
{ $tanh: -Infinity }	-1
{ $tanh: Infinity }	1

입력이 null 이거나 참조된 필드가 누락된 경우는를 $tanh 반환합니다null. 의 입력은를 NaN 생성합니다NaN. 양수 무한대의 경우 결과는 이고1, 음수 무한대의 경우 결과는 입니다-1.

예제(MongoDB 쉘)

다음 예제에서는 $tanh 연산자를 사용하여 값의 하이퍼볼릭 탄젠트를 계산하는 방법을 보여줍니다.

샘플 문서 생성

```
db.values.insertMany([
  { "_id": 1, "value": 0 },
  { "_id": 2, "value": 1 },
  { "_id": 3, "value": -1 }
]);
```

쿼리 예제

```
db.values.aggregate([
  { $project: {
    "result": { $tanh: "$value" }
  }}
]);
```

출력

```
[
  { "_id": 1, "result": 0 },
  { "_id": 2, "result": 0.7615941559557649 },
  { "_id": 3, "result": -0.7615941559557649 }
]
```

코드 예제

$tanh 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('values');

  const result = await collection.aggregate([
    { $project: {
      "result": { $tanh: "$value" }
    }}
  ]).toArray();
```

```
  console.log(result);
  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['values']

    result = list(collection.aggregate([
        { '$project': {
            'result': { '$tanh': '$value' }
        }}
    ]))

    print(result)
    client.close()

if __name__ == "__main__":
    main()
```

$상위

버전 8.0.1에서 새로 추가되었습니다.

$group 단계의 $top 누적기를 사용하여 지정된 정렬 순서에 따라 그룹당 가장 높은 순위의 문서를 반환합니다.

파라미터

- sortBy: 정렬 순서를 지정하는 문서입니다. 오름차순1에는를 사용하고 내림차순에는 -1를 사용합니다.
- output: 상위 문서에서 반환할 필드를 지정하는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $top 누적기를 사용하여 판매 컬렉션의 항목당 최고 판매량(최고 수량)을 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.sales.insertMany([
  { item: "abc", quantity: 10, price: 5 },
  { item: "abc", quantity: 5, price: 8 },
  { item: "xyz", quantity: 15, price: 3 },
  { item: "xyz", quantity: 7, price: 6 }
])
```

쿼리 예제

```
db.sales.aggregate([
  { $group: { _id: "$item", topSale: { $top: { sortBy: { quantity: -1 }, output:
 { quantity: "$quantity", price: "$price" } } } } }
])
```

출력

```
[
  { "_id": "xyz", "topSale": { "quantity": 15, "price": 3 } },
  { "_id": "abc", "topSale": { "quantity": 10, "price": 5 } }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$top산자는 $setWindowFields 스테이지에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 문서 중에서 output 최상위 문서(연산자 자신의에 따라sortBy)의를 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

> **Note**
>
> 에서 창 연산자로 사용할 경우 $setWindowFields$top는 100MB의 중간 데이터로 제한됩니다. 이 제한을 초과하는 작업은 오류를 반환합니다.

샘플 문서 생성

```
db.matchScores.insertMany([
  { _id: 1, player: "Alice", round: 1, score: 20 },
  { _id: 2, player: "Alice", round: 2, score: 35 },
  { _id: 3, player: "Alice", round: 3, score: 28 },
  { _id: 4, player: "Bob", round: 1, score: 15 },
  { _id: 5, player: "Bob", round: 2, score: 40 }
]);
```

쿼리 예제

다음 예제에서는 문서를 기준으로 분할하고player, 각 파티션을 기준으로 정렬하고round, 파티션 시작부터 현재 문서를 통해 가장 높은 점수를 받은 라운드를 반환합니다.

```
db.matchScores.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$player",
      sortBy: { round: 1 },
      output: {
        bestRoundSoFar: {
          $top: { sortBy: { score: -1 }, output: { round: "$round", score:
 "$score" } },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "player": "Alice", "round": 1, "score": 20, "bestRoundSoFar": { "round":
 1, "score": 20 } },
  { "_id": 2, "player": "Alice", "round": 2, "score": 35, "bestRoundSoFar": { "round":
 2, "score": 35 } },
  { "_id": 3, "player": "Alice", "round": 3, "score": 28, "bestRoundSoFar": { "round":
 2, "score": 35 } },
  { "_id": 4, "player": "Bob", "round": 1, "score": 15, "bestRoundSoFar": { "round": 1,
 "score": 15 } },
```

```
  { "_id": 5, "player": "Bob", "round": 2, "score": 40, "bestRoundSoFar": { "round": 2,
 "score": 40 } }
]
```

각 문서는 현재 문서를 포함하여 파티션 내의 최상위 점수 라운드bestRoundSoFar인 로 보강됩니다.

코드 예제

$top 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누산기 사용량()과 창 연산자 사용량($group)을 모두 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const db = client.db('test');

    // Accumulator usage: top document per group
    const sales = db.collection('sales');
    await sales.insertMany([
      { item: "abc", quantity: 10, price: 5 },
      { item: "abc", quantity: 5, price: 8 },
      { item: "xyz", quantity: 15, price: 3 },
      { item: "xyz", quantity: 7, price: 6 }
    ]);
    const accumulatorResult = await sales.aggregate([
      { $group: { _id: "$item", topSale: { $top: { sortBy: { quantity: -1 }, output:
 { quantity: "$quantity", price: "$price" } } } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Window operator usage: best round so far within each partition
    const matchScores = db.collection('matchScores');
    await matchScores.insertMany([
```

```
      { _id: 1, player: "Alice", round: 1, score: 20 },
      { _id: 2, player: "Alice", round: 2, score: 35 },
      { _id: 3, player: "Alice", round: 3, score: 28 },
      { _id: 4, player: "Bob", round: 1, score: 15 },
      { _id: 5, player: "Bob", round: 2, score: 40 }
    ]);
    const windowResult = await matchScores.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$player",
          sortBy: { round: 1 },
          output: {
            bestRoundSoFar: {
              $top: { sortBy: { score: -1 }, output: { round: "$round", score:
 "$score" } },
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
        db = client['test']

        # Accumulator usage: top document per group
        sales = db['sales']
        sales.insert_many([
            { 'item': 'abc', 'quantity': 10, 'price': 5 },
            { 'item': 'abc', 'quantity': 5, 'price': 8 },
            { 'item': 'xyz', 'quantity': 15, 'price': 3 },
            { 'item': 'xyz', 'quantity': 7, 'price': 6 }
        ])
        accumulator_result = list(sales.aggregate([
            { '$group': { '_id': '$item', 'topSale': { '$top': { 'sortBy':
{ 'quantity': -1 }, 'output': { 'quantity': '$quantity', 'price':
'$price' } } } } }
        ]))
        pprint(accumulator_result)

        # Window operator usage: best round so far within each partition
        match_scores = db['matchScores']
        match_scores.insert_many([
            { '_id': 1, 'player': 'Alice', 'round': 1, 'score': 20 },
            { '_id': 2, 'player': 'Alice', 'round': 2, 'score': 35 },
            { '_id': 3, 'player': 'Alice', 'round': 3, 'score': 28 },
            { '_id': 4, 'player': 'Bob', 'round': 1, 'score': 15 },
            { '_id': 5, 'player': 'Bob', 'round': 2, 'score': 40 }
        ])
        window_result = list(match_scores.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$player',
                    'sortBy': { 'round': 1 },
                    'output': {
                        'bestRoundSoFar': {
                            '$top': { 'sortBy': { 'score': -1 }, 'output':
{ 'round': '$round', 'score': '$score' } },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        pprint(window_result)
```

```
    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$topN

버전 8.0.1에서 새로 추가되었습니다.

$group 스테이지의 $topN 누적기를 사용하여 지정된 정렬 순서에 따라 그룹 내의 상위 N개 요소를 반환합니다. 그룹에 N개 미만의 요소가 포함된 경우는 그룹의 모든 요소를 $topN 반환합니다.

파라미터

- n: 그룹당 반환할 상위 결과 수를 지정하는 양의 정수 또는 1로 확인되는 표현식입니다.
- sortBy: 정렬 순서를 지정하는 문서입니다. 오름차순1에는를 사용하고 내림차순에는 -1를 사용합니다.
- output: 상위 N개 문서 각각에서 반환할 필드를 지정하는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $topN 누적기를 사용하여 판매 컬렉션의 항목당 상위 2개 판매(최대 수량)를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.sales.insertMany([
  { item: "abc", quantity: 10, price: 5 },
  { item: "abc", quantity: 7, price: 8 },
  { item: "abc", quantity: 5, price: 10 },
  { item: "xyz", quantity: 15, price: 3 },
  { item: "xyz", quantity: 9, price: 6 },
  { item: "xyz", quantity: 3, price: 12 }
])
```

쿼리 예제

```
db.sales.aggregate([
  { $group: { _id: "$item", topTwoSales: { $topN: { n: 2, sortBy: { quantity: -1 },
 output: { quantity: "$quantity", price: "$price" } } } } }
])
```

출력

```
[
  { "_id": "xyz", "topTwoSales": [{ "quantity": 15, "price": 3 }, { "quantity": 9,
 "price": 6 }] },
  { "_id": "abc", "topTwoSales": [{ "quantity": 10, "price": 5 }, { "quantity": 7,
 "price": 8 }] }
]
```

창 연산자 사용 예제(MongoDB Shell)

버전 8.0.2에서 새로 추가되었습니다.

연$topN산자는 $setWindowFields 스테이지에서 창 연산자로 사용할 수도 있습니다. 이 컨텍스트에서는 각 창의 문서 중에서 상위 N개 문서(연산자 자신의에 따라sortBy)output의 배열을 반환합니다. output 필드 아래에 연산자를 지정하고 선택적으로 window 문서로 창 경계를 정의합니다.

> **Note**
>
> 에서 창 연산자로 사용할 경우 $setWindowFields$topN는 100MB의 중간 데이터로 제한됩니다. 이 제한을 초과하는 작업은 오류를 반환합니다.

샘플 문서 생성

```
db.matchScores.insertMany([
  { _id: 1, player: "Alice", round: 1, score: 20 },
  { _id: 2, player: "Alice", round: 2, score: 35 },
  { _id: 3, player: "Alice", round: 3, score: 28 },
  { _id: 4, player: "Bob", round: 1, score: 15 },
  { _id: 5, player: "Bob", round: 2, score: 40 }
]);
```

쿼리 예제

다음 예제에서는를 기준으로 문서를 분할하고player,를 기준으로 각 파티션을 정렬하고round, 파티션 시작부터 현재 문서를 통해 가장 높은 점수를 받은 두 라운드를 반환합니다.

```
db.matchScores.aggregate([
  {
    $setWindowFields: {
      partitionBy: "$player",
      sortBy: { round: 1 },
      output: {
        topTwoRoundsSoFar: {
          $topN: { n: 2, sortBy: { score: -1 }, output: { round: "$round", score:
 "$score" } },
          window: { documents: ["unbounded", "current"] }
        }
      }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "player": "Alice", "round": 1, "score": 20, "topTwoRoundsSoFar":
 [{ "round": 1, "score": 20 }] },
  { "_id": 2, "player": "Alice", "round": 2, "score": 35, "topTwoRoundsSoFar":
 [{ "round": 2, "score": 35 }, { "round": 1, "score": 20 }] },
  { "_id": 3, "player": "Alice", "round": 3, "score": 28, "topTwoRoundsSoFar":
 [{ "round": 2, "score": 35 }, { "round": 3, "score": 28 }] },
  { "_id": 4, "player": "Bob", "round": 1, "score": 15, "topTwoRoundsSoFar":
 [{ "round": 1, "score": 15 }] },
  { "_id": 5, "player": "Bob", "round": 2, "score": 40, "topTwoRoundsSoFar":
 [{ "round": 2, "score": 40 }, { "round": 1, "score": 15 }] }
]
```

각 문서는 현재 문서를 포함하여 파티션 내에서 가장 높은 점수를 받은 topTwoRoundsSoFar두 라운드인 로 보강됩니다. 창에 문서가 2개 미만인 경우 모든 문서가 반환됩니다.

코드 예제

$topN 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다. 다음 예제에서는 누산기 사용량()과 창 연산자 사용량($group)을 모두 보여줍니다. $setWindowFields

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const db = client.db('test');

    // Accumulator usage: top N documents per group
    const sales = db.collection('sales');
    await sales.insertMany([
      { item: "abc", quantity: 10, price: 5 },
      { item: "abc", quantity: 7, price: 8 },
      { item: "abc", quantity: 5, price: 10 },
      { item: "xyz", quantity: 15, price: 3 },
      { item: "xyz", quantity: 9, price: 6 },
      { item: "xyz", quantity: 3, price: 12 }
    ]);
    const accumulatorResult = await sales.aggregate([
      { $group: { _id: "$item", topTwoSales: { $topN: { n: 2, sortBy: { quantity:
 -1 }, output: { quantity: "$quantity", price: "$price" } } } } }
    ]).toArray();
    console.log('Accumulator result:', accumulatorResult);

    // Window operator usage: top two rounds so far within each partition
    const matchScores = db.collection('matchScores');
    await matchScores.insertMany([
      { _id: 1, player: "Alice", round: 1, score: 20 },
      { _id: 2, player: "Alice", round: 2, score: 35 },
      { _id: 3, player: "Alice", round: 3, score: 28 },
      { _id: 4, player: "Bob", round: 1, score: 15 },
      { _id: 5, player: "Bob", round: 2, score: 40 }
    ]);
    const windowResult = await matchScores.aggregate([
      {
        $setWindowFields: {
          partitionBy: "$player",
```

```
          sortBy: { round: 1 },
          output: {
            topTwoRoundsSoFar: {
              $topN: { n: 2, sortBy: { score: -1 }, output: { round: "$round",
 score: "$score" } },
              window: { documents: ["unbounded", "current"] }
            }
          }
        }
      }
    ]).toArray();
    console.log('Window result:', windowResult);

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient
from pprint import pprint

def example():
    client = None
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

        db = client['test']

        # Accumulator usage: top N documents per group
        sales = db['sales']
        sales.insert_many([
            { 'item': 'abc', 'quantity': 10, 'price': 5 },
            { 'item': 'abc', 'quantity': 7, 'price': 8 },
            { 'item': 'abc', 'quantity': 5, 'price': 10 },
            { 'item': 'xyz', 'quantity': 15, 'price': 3 },
```

```
            { 'item': 'xyz', 'quantity': 9, 'price': 6 },
            { 'item': 'xyz', 'quantity': 3, 'price': 12 }
        ])
        accumulator_result = list(sales.aggregate([
            { '$group': { '_id': '$item', 'topTwoSales': { '$topN': { 'n': 2,
'sortBy': { 'quantity': -1 }, 'output': { 'quantity': '$quantity', 'price':
'$price' } } } } }
        ]))
        pprint(accumulator_result)

        # Window operator usage: top two rounds so far within each partition
        match_scores = db['matchScores']
        match_scores.insert_many([
            { '_id': 1, 'player': 'Alice', 'round': 1, 'score': 20 },
            { '_id': 2, 'player': 'Alice', 'round': 2, 'score': 35 },
            { '_id': 3, 'player': 'Alice', 'round': 3, 'score': 28 },
            { '_id': 4, 'player': 'Bob', 'round': 1, 'score': 15 },
            { '_id': 5, 'player': 'Bob', 'round': 2, 'score': 40 }
        ])
        window_result = list(match_scores.aggregate([
            {
                '$setWindowFields': {
                    'partitionBy': '$player',
                    'sortBy': { 'round': 1 },
                    'output': {
                        'topTwoRoundsSoFar': {
                            '$topN': { 'n': 2, 'sortBy': { 'score': -1 }, 'output':
{ 'round': '$round', 'score': '$score' } },
                            'window': { 'documents': ['unbounded', 'current'] }
                        }
                    }
                }
            }
        ]))
        pprint(window_result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()
```

```
example()
```

$trunc

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 $trunc 연산자는 숫자를 지정된 소수점 자리로 잘라내어 반올림 없이 숫자를 제거합니다.

파라미터

- number: 숫자로 확인되는 표현식입니다.
- place: 선택 사항입니다. 잘라낼 소수 자릿수를 지정하는 -20~100 사이의 정수 표현식입니다. 음수 값은 소수점 왼쪽으로 잘립니다. 기본값은 0입니다.

예제(MongoDB 쉘)

다음 예제에서는 $trunc 연산자를 사용하여 숫자 값을 소수점 한 자리로 자르는 방법을 보여줍니다.

양의 자리 값으로 잘라내기

```
db.measurements.insertMany([
  {_id: 1, value: 3.456},
  {_id: 2, value: 7.891},
  {_id: 3, value: 12.345}
]);
```

쿼리 예제

```
db.measurements.aggregate([
  { $project: { truncated: { $trunc: ["$value", 1] } } }
]);
```

출력

```
[
  {_id: 1, truncated: 3.4},
  {_id: 2, truncated: 7.8},
```

```
  {_id: 3, truncated: 12.3}
]
```

음수 자리 값으로 잘라내기

place가 음수인 경우는 소수점 왼쪽의 숫자를 0으로 $trunc 바꿉니다. 다음 예제에서는 소수점 왼쪽의 첫 번째 숫자를 사용하여 값을 잘라냅니다.

```
db.samples.insertMany([
  {_id: 1, value: 19.25},
  {_id: 2, value: 28.73},
  {_id: 3, value: 34.32}
]);
```

```
db.samples.aggregate([
  { $project: { truncated: { $trunc: ["$value", -1] } } }
]);
```

출력

```
[
  {_id: 1, truncated: 10},
  {_id: 2, truncated: 20},
  {_id: 3, truncated: 30}
]
```

코드 예제

$trunc 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
```

```
    const db = client.db('test');
    const collection = db.collection('measurements');
    const result = await collection.aggregate([
      { $project: { truncated: { $trunc: ["$value", 1] } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['measurements']
        result = list(collection.aggregate([
            {'$project': {'truncated': {'$trunc': ['$value', 1]}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$tsIncrement

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 `$tsIncrement` 연산자는 타임스탬프 값에서 증가하는 서수를 긴 정수로 반환합니다.

파라미터

- `expression`: 타임스탬프로 확인되는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $tsIncrement 연산자를 사용하여 타임스탬프 값에서 서수 증분을 추출하는 방법을 보여줍니다.

샘플 문서 생성

```
db.events.insertMany([
  {_id: 1, ts: Timestamp(1678900000, 1)},
  {_id: 2, ts: Timestamp(1678900000, 2)},
  {_id: 3, ts: Timestamp(1678900001, 1)}
]);
```

쿼리 예제

```
db.events.aggregate([
  { $project: { increment: { $tsIncrement: "$ts" } } }
]);
```

출력

```
[
  {_id: 1, increment: Long("1")},
  {_id: 2, increment: Long("2")},
  {_id: 3, increment: Long("1")}
]
```

코드 예제

$tsIncrement 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient, Timestamp } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
```

```
    const db = client.db('test');
    const collection = db.collection('events');
    const result = await collection.aggregate([
      { $project: { increment: { $tsIncrement: "$ts" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['events']
        result = list(collection.aggregate([
            {'$project': {'increment': {'$tsIncrement': '$ts'}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$tsSecond

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 `$tsSecond` 연산자는 타임스탬프 값의 초 부분을 긴 정수(Unix epoch 초)로 반환합니다.

파라미터

- `expression`: 타임스탬프로 확인되는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $tsSecond 연산자를 사용하여 타임스탬프 값에서 초 부분을 추출하는 방법을 보여 줍니다.

샘플 문서 생성

```
db.events.insertMany([
  {_id: 1, ts: Timestamp(1678900000, 1)},
  {_id: 2, ts: Timestamp(1678900000, 2)},
  {_id: 3, ts: Timestamp(1678900001, 1)}
]);
```

쿼리 예제

```
db.events.aggregate([
  { $project: { seconds: { $tsSecond: "$ts" } } }
]);
```

출력

```
[
  {_id: 1, seconds: Long("1678900000")},
  {_id: 2, seconds: Long("1678900000")},
  {_id: 3, seconds: Long("1678900001")}
]
```

코드 예제

$tsSecond 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient, Timestamp } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
```

```
    const db = client.db('test');
    const collection = db.collection('events');
    const result = await collection.aggregate([
      { $project: { seconds: { $tsSecond: "$ts" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['events']
        result = list(collection.aggregate([
            {'$project': {'seconds': {'$tsSecond': '$ts'}}}
        ]))
        print(result)
    finally:
        client.close()

example()
```

$toBool

Amazon DocumentDB의 `$toBool` 연산자는 표현식을 부울 값으로 변환합니다.

파라미터

- `expression`: 부울 값으로 변환할 표현식입니다.

참고: 모든 문자열은 로 변환됩니다`true`.

예제(MongoDB 쉘)

다음 예제에서는 $toBool 연산자를 사용하여 다양한 데이터 형식의 디바이스 상태 값을 정규화하는 방법을 보여줍니다.

샘플 문서 생성

```
db.deviceStates.insertMany([
  { _id: 1, deviceId: "sensor-001", status: true },
  { _id: 2, deviceId: "camera-002", status: 1 },
  { _id: 3, deviceId: "thermostat-003", status: "active" },
  { _id: 4, deviceId: "doorlock-004", status: 0 }
]);
```

쿼리 예제

```
db.deviceStates.aggregate([
  {
    $project: {
      _id: 1,
      deviceId: 1,
      isActive: { $toBool: "$status" }
    }
  }
]);
```

출력

```
[
  { "_id": 1, "deviceId": "sensor-001", "isActive": true },
  { "_id": 2, "deviceId": "camera-002", "isActive": true },
  { "_id": 3, "deviceId": "thermostat-003", "isActive": true },
  { "_id": 4, "deviceId": "doorlock-004", "isActive": false }
]
```

코드 예제

$toBool 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('deviceStates');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        deviceId: 1,
        isActive: { $toBool: '$status' }
      }
    }
  ]).toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['deviceStates']

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'deviceId': 1,
                'isActive': { '$toBool': '$status' }
            }
```

```
        }
    ]))

    print(result)

    client.close()

if __name__ == '__main__':
    main()
```

$toDate

버전 4.0의 새로운 기능

Amazon DocumentDB의 $toDate 집계 연산자는 날짜 또는 날짜 및 시간 문자열을 BSON 날짜 유형으로 변환하는 데 사용됩니다. 연$dateToString산자의 역 작업입니다.

파라미터

- dateString: BSON 날짜 유형으로 변환할 날짜 또는 날짜 및 시간의 문자열 표현입니다.
- format: (선택 사항) 형식을 지정하는 문자열입니다dateString. 제공되지 않은 경우 연산자는 다양한 표준 날짜 및 시간 형식으로 dateString를 구문 분석하려고 시도합니다.
- timezone: (선택 사항) 변환에 사용할 시간대를 나타내는 문자열입니다. 제공되지 않은 경우 현지 시간대가 사용됩니다.

예제(MongoDB 쉘)

다음 예제에서는 $toDate 연산자를 사용하여 날짜 문자열을 BSON 날짜 유형으로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.events.insertMany([
  { _id: 1, eventName: "Mission Start", eventTime: "2023-04-15T10:30:00Z" },
  { _id: 2, eventName: "Checkpoint Reached", eventTime: "2023-04-15T11:15:00Z" },
  { _id: 3, eventName: "Mission End", eventTime: "2023-04-15T12:00:00Z" }
]);
```

쿼리 예제

```
db.events.aggregate([
  {
    $project: {
      eventName: 1,
      eventTimeDate: { $toDate: "$eventTime" }
    }
  }
]);
```

출력

```
[
  {
    "_id": 1,
    "eventName": "Mission Start",
    "eventTimeDate": ISODate("2023-04-15T10:30:00Z")
  },
  {
    "_id": 2,
    "eventName": "Checkpoint Reached",
    "eventTimeDate": ISODate("2023-04-15T11:15:00Z")
  },
  {
    "_id": 3,
    "eventName": "Mission End",
    "eventTimeDate": ISODate("2023-04-15T12:00:00Z")
  }
]
```

코드 예제

$toDate 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
```

```
  const collection = db.collection('events');

  const result = await collection.aggregate([
    {
      $project: {
        eventName: 1,
        eventTimeDate: { $toDate: '$eventTime' }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['events']

    result = list(collection.aggregate([
        {
            '$project': {
                'eventName': 1,
                'eventTimeDate': { '$toDate': '$eventTime' }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$toDecimal

버전 4.0의 새로운 기능

Amazon DocumentDB의 `$toDecimal` 연산자는 값을 Decimal128 데이터 형식으로 변환하는 데 사용됩니다. 이는 정확한 십진수 산술을 수행하거나 Double 데이터 유형을 사용하여 정확하게 표현할 수 없는 큰 십진수 값을 처리해야 하는 경우에 유용합니다.

파라미터

- `expression`: Decimal128 데이터 형식으로 변환할 표현식입니다.

예제(MongoDB 쉘)

이 예제에서는 `$toDecimal` 연산자를 사용하여 문자열 값을 Decimal128 데이터 형식으로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.numbers.insertOne({ _id: 1, value: "3.14" });
db.numbers.insertOne({ _id: 2, value: "2.71" });
```

쿼리 예제

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    decimalValue: { $toDecimal: "$value" }
  }}
])
```

출력

```
[
  { "_id" : 1, "decimalValue" : Decimal128("3.14") },
  { "_id" : 2, "decimalValue" : Decimal128("2.71") }
]
```

코드 예제

`$toDecimal` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $project: {
      _id: 1,
      decimalValue: { $toDecimal: "$value" }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.numbers

    result = list(collection.aggregate([
        {'$project': {
            '_id': 1,
            'decimalValue': {'$toDecimal': '$value'}
        }}
    ]))

    print(result)
```

```
    client.close()

example()
```

$toDouble

버전 4.0의 새로운 기능

Amazon DocumentDB의 $toDouble 연산자는 값을 이중 정밀도 64비트 부동 소수점 숫자로 변환하는 데 사용됩니다. 이는 원래 숫자 형식이 아닌 값에 대해 산술 연산을 수행해야 할 때 유용할 수 있습니다.

파라미터

<expression>: 이중 값으로 변환할 표현식입니다. 숫자, 문자열 또는 부울 값으로 확인되는 유효한 표현식일 수 있습니다.

예제(MongoDB 쉘)

이 예제에서는 수학 계산을 수행할 목적으로 $toDouble 연산자를 사용하여 문자열 값을 숫자 값으로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.numbers.insertMany([
  { _id: 1, value: "10.5" },
  { _id: 2, value: "20.25" },
  { _id: 3, value: "7" }
])
```

쿼리 예제

```
db.numbers.aggregate([
  {
    $project: {
      _id: 1,
      value: 1,
      double_value: { $toDouble: "$value" },
      double_plus_five: { $add: [{ $toDouble: "$value" }, 5] }
    }
```

```
  }
])
```

출력

```
[
  { "_id" : 1, "value" : "10.5", "double_value" : 10.5, "double_plus_five" : 15.5 },
  { "_id" : 2, "value" : "20.25", "double_value" : 20.25, "double_plus_five" : 25.25 },
  { "_id" : 3, "value" : "7", "double_value" : 7.0, "double_plus_five" : 12.0 }
]
```

코드 예제

$toDouble 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    {
      $project: {
        _id: 1,
        value: 1,
        double_value: { $toDouble: "$value" },
        double_plus_five: { $add: [{ $toDouble: "$value" }, 5] }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['numbers']

    result = list(collection.aggregate([
        {
            '$project': {
                '_id': 1,
                'value': 1,
                'double_value': { '$toDouble': '$value' },
                'double_plus_five': { '$add': [{ '$toDouble': '$value' }, 5] }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$toInt

버전 4.0의 새로운 기능

Amazon DocumentDB의 `$toInt` 연산자는 입력 값을 정수 데이터 형식으로 변환하는 데 사용됩니다. 이 연산자는 필드 또는 표현식을 정수로 표현해야 할 때 유용하며, 이는 특정 작업 또는 데이터 처리 작업에 중요할 수 있습니다.

파라미터

- `expression`: 정수로 변환할 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $toInt 연산자를 사용하여 문자열 값을 정수로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.numbers.insertMany([
  { "name": "one", "value": "1" },
  { "name": "hundred", "value": "100" }
]);
```

쿼리 예제

```
db.numbers.aggregate([
  { $project: {
    "_id": 0,
    "name": 1,
    "intValue": { $toInt: "$value" }
  }}
]);
```

출력

```
{ "name": "one", "intValue": 1 }
{ "name": "hundred", "intValue": 100 }
```

코드 예제

$toInt 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');
```

```
  const result = await collection.aggregate([
    { $project: {
      "_id": 0,
      "name": 1,
      "intValue": { $toInt: "$value" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['numbers']

    result = list(collection.aggregate([
        { "$project": {
            "_id": 0,
            "name": 1,
            "intValue": { "$toInt": "$value" }
        }}
    ]))

    print(result)
    client.close()

example()
```

$toLong

버전 4.0의 새로운 기능

Amazon DocumentDB의 $toLong 연산자는 값을 64비트 정수(긴) 데이터 형식으로 변환하는 데 사용됩니다. 이는 문자열 또는 기타 데이터 형식으로 저장될 수 있는 숫자 값에 대한 산술 연산 또는 비교를 수행해야 할 때 유용할 수 있습니다.

파라미터

- expression: 64비트 정수로 변환할 표현식입니다.

예제(MongoDB 쉘)

이 예제에서는 $toLong 연산자를 사용하여 문자열 값을 64비트 정수로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.numbers.insertMany([
  { _id: 1, value: "42" },
  { _id: 3, value: "9223372036854775807" }
]);
```

쿼리 예제

```
db.numbers.aggregate([
  {
    $project: {
      _id: 1,
      longValue: { $toLong: "$value" }
    }
  }
])
```

출력

```
[
  { "_id" : 1, "longValue" : 42 },
  { "_id" : 3, "longValue" : 9223372036854775807 }
]
```

코드 예제

$toLong 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const numbers = db.collection('numbers');

  const result = await numbers.aggregate([
    {
      $project: {
        _id: 1,
        longValue: { $toLong: "$value" }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    numbers = db.numbers

    result = list(numbers.aggregate([
        {
            '$project': {
                '_id': 1,
                'longValue': { '$toLong': '$value' }
            }
```

```
        }
    ]))

    print(result)
    client.close()

example()
```

$toLower

Amazon DocumentDB의 `$toLower` 연산자는 문자열을 소문자로 변환하는 데 사용됩니다.

파라미터

- `expression`: 소문자로 변환할 문자열 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$toLower` 연산자를 사용하여 `Desk` 필드를 소문자로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.locations.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-BVV-021" },
  { "_id": 2, "Desk": "Munich-HGG-32a" }
]);
```

쿼리 예제

```
db.locations.aggregate([
  { $project: { item: { $toLower: "$Desk" } } }
]);
```

출력

```
{ "_id" : 1, "item" : "düsseldorf-bvv-021" }
{ "_id" : 2, "item" : "munich-hgg-32a" }
```

코드 예제

`$toLower` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require("mongodb");

async function main() {
  const client = await MongoClient.connect("mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false");
  const db = client.db("test");
  const collection = db.collection("locations");

  const result = await collection.aggregate([
    { $project: { item: { $toLower: "$Desk" } } }
  ]).toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient("mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false")
    db = client["test"]
    collection = db["locations"]

    result = list(collection.aggregate([
        { "$project": { "item": { "$toLower": "$Desk" } } }
    ]))

    print(result)

    client.close()

if __name__ == "__main__":
    main()
```

$toObjectId

버전 4.0의 새로운 기능

Amazon DocumentDB의 `$toObjectId` 연산자는 ObjectId의 문자열 표현을 실제 ObjectId 데이터 유형으로 변환하는 데 사용됩니다. ObjectIds 데이터 형식이 필요한 작업을 수행할 수 있으므로 ObjectId의 문자열 표현으로 저장된 데이터로 작업할 때 유용할 수 있습니다.

파라미터

- `expression`: 유효한 ObjectId를 나타내는 문자열 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$toObjectId` 연산자를 사용하여 ObjectId의 문자열 표현을 ObjectId 데이터 형식으로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.employees.insertMany([
  { _id: 1, empId:"64e5f8886218c620cf0e8f8a", name: "Carol Smith", employeeId:
 "c720a" },
  { _id: 2, empId:"64e5f94e6218c620cf0e8f8c", name: "Bill Taylor", employeeId:
 "c721a" }
]);
```

쿼리 예제

```
db.employees.aggregate([
  { $project: {
    "empIdAsObjectId": {$toObjectId: "$empId"}}
  }
]);
```

출력

```
[
  { _id: 1, empIdAsObjectId: ObjectId('64e5f8886218c620cf0e8f8a') },
  { _id: 2, empIdAsObjectId: ObjectId('64e5f94e6218c620cf0e8f8c') }
]
```

코드 예제

$toObjectId 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('employees');

  const result = await collection.aggregate([
    { $project: {
      "empIdAsObjectId": {$toObjectId: "$empId"}}
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['employees']

    result = list(collection.aggregate([
      { "$project": {
        "empIdAsObjectId": {"$toObjectId": "$empId"}}
      }
    ]))
```

```
    print(result)
    client.close()

example()
```

$toString

버전 4.0의 새로운 기능

Amazon DocumentDB의 $toString 연산자는 모든 유형의 값(null 제외)을 문자열 표현으로 변환하는 데 사용됩니다. 이는 원래 문자열 형식이 아닌 값에 대해 문자열 작업을 수행해야 할 때 유용할 수 있습니다.

파라미터

- expression: 문자열로 변환할 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $toString 연산자를 사용하여 숫자 값을 문자열로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.numbers.insertMany([
  { "_id": 1, "value": 42 },
  { "_id": 2, "value": 3.14 }
]);
```

쿼리 예제

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    valueAsString: { $toString: "$value" }
  }}
]);
```

출력

```
{ "_id": 1, "valueAsString": "42" }
```

```
{ "_id": 2, "valueAsString": "3.14" }
```

코드 예제

$toString 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  const result = await collection.aggregate([
    { $project: {
      _id: 1,
      valueAsString: { $toString: '$value' }
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['numbers']

    result = list(collection.aggregate([
        { '$project': {
```

```
            '_id': 1,
            'valueAsString': { '$toString': '$value' }
        }}
    ]))

    print(result)
    client.close()

example()
```

$toUpper

Amazon DocumentDB의 $toUpper 연산자는 문자열을 대문자로 변환하는 데 사용됩니다.

파라미터

- expression: 대문자로 변환할 문자열 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 연$toUpper산자를 사용하여 Desk 필드를 대문자로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.locations.insertMany([
  { "_id": 1, "Desk": "düsseldorf-bvv-021" },
  { "_id": 2, "Desk": "munich-hgg-32a" }
]);
```

쿼리 예제

```
db.locations.aggregate([
  { $project: { item: { $toUpper: "$Desk" } } }
]);
```

출력

```
{ "_id" : 1, "item" : "DüSSELDORF-BVV-021" }
{ "_id" : 2, "item" : "MUNICH-HGG-32A" }
```

코드 예제

$toUpper 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('locations');

  const result = await collection.aggregate([
    { $project: { item: { $toUpper: '$Desk' } } }
  ]).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['locations']

    result = list(collection.aggregate([
        { '$project': { 'item': { '$toUpper': '$Desk' } } }
    ]))

    print(result)
```

```
    client.close()

example()
```

$toUUID

버전 8.0.1에서 새로 추가되었습니다.

Amazon DocumentDB의 `$toUUID` 연산자는 문자열 값을 UUID(이진 하위 유형 4)로 변환합니다.

파라미터

- `expression`: UUID 형식의 문자열로 확인되는 표현식입니다(예: "12345678-1234-1234-1234-123456789abc").

예제(MongoDB 쉘)

다음 예제에서는 `$toUUID` 연산자를 사용하여 문자열 값을 UUID 형식으로 변환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.records.insertMany([
  {_id: 1, uuidStr: "550e8400-e29b-41d4-a716-446655440000"},
  {_id: 2, uuidStr: "6ba7b810-9dad-11d1-80b4-00c04fd430c8"}
]);
```

쿼리 예제

```
db.records.aggregate([
  { $project: { uuid: { $toUUID: "$uuidStr" } } }
]);
```

출력

```
[
  {_id: 1, uuid: UUID("550e8400-e29b-41d4-a716-446655440000")},
  {_id: 2, uuid: UUID("6ba7b810-9dad-11d1-80b4-00c04fd430c8")}
]
```

코드 예제

$toUUID 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = new MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('records');
    const result = await collection.aggregate([
      { $project: { uuid: { $toUUID: "$uuidStr" } } }
    ]).toArray();
    console.log(result);
  } finally {
    await client.close();
  }
}
example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    try:
        db = client['test']
        collection = db['records']
        result = list(collection.aggregate([
            {'$project': {'uuid': {'$toUUID': '$uuidStr'}}}
        ]))
        print(result)
    finally:
        client.close()
```

```
example()
```

$trim

버전 4.0의 새로운 기능

Amazon DocumentDB의 `$trim` 연산자는 문자열에서 선행 및/또는 후행 공백 문자를 제거하는 데 사용됩니다.

파라미터

- `input`: 잘라낼 문자열 표현식입니다.
- `chars`: (선택 사항) 입력의 시작과 끝에서 잘라낼 문자를 지정합니다. 기본값은 공백입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$trim` 연산자를 사용하여 문자열에서 선행 및 후행 공백을 제거하는 방법을 보여줍니다.

샘플 문서 생성

```
db.people.insertMany([
  { "name": "   John Doe   " },
  { "name": "   Bob Johnson   " }
])
```

쿼리 예제

```
db.people.aggregate([
  { $project: {
    "name": { $trim: {input: "$name"}}
  }}
])
```

출력

```
[
  { "name": "John Doe" },
```

```
  { "name": "Bob Johnson" }
]
```

코드 예제

$trim 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const result = await collection.aggregate([
    { $project: {
      "name": { $trim: {input: "$name" }}
    }}
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['people']

    result = list(collection.aggregate([
        {"$project": {
```

```
            "name": {"$trim": {"input": "$name"}}
        }}
    ]))

    print(result)
    client.close()

example()
```

$type

$type 집계 연산자는 지정된 필드의 BSON 데이터 형식을 반환합니다. 이는 집계 작업 중에 필드 값의 데이터 유형을 식별하는 데 유용합니다.

파라미터

- expression: 반환할 유형이 있는 필드 또는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $type 연산자를 사용하여 각 제품에 대한 가격 필드의 데이터 유형을 식별하는 방법을 보여줍니다.

샘플 문서 생성

```
db.inventory.insertMany([
  { _id: 1, item: "Notebook", price: 15.99 },
  { _id: 2, item: "Pen", price: "2.50" },
  { _id: 3, item: "Eraser", price: 1 },
  { _id: 4, item: "Ruler", price: null }
]);
```

쿼리 예제

```
db.inventory.aggregate([
  {
    $project: {
      item: 1,
      price: 1,
      priceType: { $type: "$price" }
    }
```

```
  }
]);
```

출력

```
[
  { _id: 1, item: 'Notebook', price: 15.99, priceType: 'double' },
  { _id: 2, item: 'Pen', price: '2.50', priceType: 'string' },
  { _id: 3, item: 'Eraser', price: 1, priceType: 'int' },
  { _id: 4, item: 'Ruler', price: null, priceType: 'null' }
]
```

코드 예제

$type 집계 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('inventory');

  const result = await collection.aggregate([
    {
      $project: {
        item: 1,
        price: 1,
        priceType: { $type: "$price" }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['inventory']

    result = list(collection.aggregate([
        {
            '$project': {
                'item': 1,
                'price': 1,
                'priceType': { '$type': '$price' }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$unset

버전 8.0의 새로운 기능

Elastic 클러스터에서는 지원되지 않습니다.

Amazon DocumentDB의 `$unset` 집계 단계를 사용하면 문서에서 필드를 제거할 수 있습니다.

파라미터

- `expression`: 필드 이름 또는 여러 필드 이름의 목록입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$unset` 집계 단계를 사용하여 `price` 필드를 제거하는 방법을 보여줍니다.

샘플 문서 생성

```
db.inventory.insertMany([
  { item: "pencil", quantity: 100, price: 0.24},
  { item: "pen", quantity: 204, price: 1.78 }
]);
```

집계 예제

```
db.inventory.aggregate([
  {
    $unset: "price"
  }
])
```

출력

```
[
  {
    _id: ObjectId('69248951d66dcae121d2950d'),
    item: 'pencil',
    quantity: 100
  },
  {
    _id: ObjectId('69248951d66dcae121d2950e'),
    item: 'pen',
    quantity: 204
  }
]
```

코드 예제

$unset 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
```

```
  const db = client.db('test');
  const inventory = db.collection('inventory');

  const result = await inventory.aggregate([
      {
        $unset: "price"
      }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    inventory = db['inventory']

    result = list(inventory.aggregate([
      {
        "$unset": "price"
      }
    ]))

    print(result)
    client.close()

example()
```

$unwind

연$unwind산자는 입력 문서에서 배열 필드를 분해하여 각 요소에 대한 문서를 출력하는 데 사용됩니다. 이는 데이터 필터링, 정렬 또는 변환과 같은 배열의 개별 요소에 대한 작업을 수행하려는 경우에 유용할 수 있습니다.

파라미터

- path: 풀릴 배열 필드의 경로입니다.
- includeArrayIndex: (선택 사항) 배열 요소의 인덱스를 포함할 새 필드의 이름을 지정합니다.
- preserveNullAndEmptyArrays: (선택 사항) 배열 필드가 null이거나 빈 배열일 때 작업이 원본 문서를 유지할지 여부를 결정합니다.

예제(MongoDB 쉘)

다음 예제에서는 $unwind 연산자를 사용하여 배열 필드를 분해하고 개별 요소에 대해 추가 작업을 수행하는 방법을 보여줍니다.

샘플 문서 생성

```
db.people.insertMany([
  { _id: 1, name: "jon", hobbies: ["painting", "dancing", "singing"] },
  { _id: 2, name: "jane", hobbies: ["reading", "swimming"] },
  { _id: 3, name: "jack", hobbies: [] }
])
```

쿼리 예제

```
db.people.aggregate([
  { $unwind: "$hobbies" }
])
```

출력

```
[
  { _id: 1, name: 'jon', hobbies: 'painting' },
  { _id: 1, name: 'jon', hobbies: 'dancing' },
  { _id: 1, name: 'jon', hobbies: 'singing' },
  { _id: 2, name: 'jane', hobbies: 'reading' },
  { _id: 2, name: 'jane', hobbies: 'swimming' }
]
```

코드 예제

$unwind 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('people');

  const result = await collection.aggregate([
    { $unwind: '$hobbies' }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['people']

    result = list(collection.aggregate([
        { '$unwind': '$hobbies' }
    ]))

    print(result)
    client.close()

example()
```

$vectorSearch

버전 8.0의 새로운 기능

Elastic 클러스터에서는 지원되지 않습니다.

Amazon DocumentDB의 $vectorSearch 연산자를 사용하면 기계 학습에 사용되는 메서드인 벡터 검색을 수행하여 거리 또는 유사성 지표를 사용하여 벡터 표현을 비교하여 유사한 데이터 포인트를 찾을 수 있습니다. 이 기능은 JSON 기반 문서 데이터베이스의 유연성과 풍부한 쿼리를 벡터 검색의 기능과 결합하여 의미 체계 검색, 제품 권장 사항 등과 같은 기계 학습 및 생성형 AI 사용 사례를 구축할 수 있습니다.

파라미터

- <exact> (선택 사항): 정확히 가장 가까운 이웃(ENN) 또는 대략적인 가장 가까운 이웃(ANN) 검색을 실행할지 여부를 지정하는 플래그입니다. 값은 다음 중 하나일 수 있습니다.
- false - ANN 검색을 실행하려면
- true - ENN 검색을 실행하려면

생략하거나 false로 설정하면 numCandidates가 필요합니다.

```
- `<index>` : Name of the Vector Search index to use.
- `<limit>` : Number of documents to return in the results.
- `<numCandidates>` (optional): This field is required if 'exact' is false or omitted.
 Number of nearest neighbors to use during the search. Value must be less than or equal
 to (<=) 1000. You can't specify a number less than the number of documents to return
 ('limit').
- `<path>` : Indexed vector type field to search.
- `<queryVector>` : Array of numbers that represent the query vector.
```

예제(MongoDB 쉘)

다음 예제에서는 $vectorSearch 연산자를 사용하여 벡터 표현을 기반으로 유사한 제품 설명을 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.products.insertMany([
  {
    _id: 1,
```

```
    name: "Product A",
    description: "A high-quality, eco-friendly product for your home.",
    description_vector: [ 0.2, 0.5, 0.8 ]
  },
  {
    _id: 2,
    name: "Product B",
    description: "An innovative and modern kitchen appliance.",
    description_vector: [0.7, 0.3, 0.9]
  },
  {
    _id: 3,
    name: "Product C",
    description: "A comfortable and stylish piece of furniture.",
    description_vector: [0.1, 0.2, 0.4]
  }
]);
```

벡터 검색 인덱스 생성

```
db.runCommand(
    {
        createIndexes: "products",
        indexes: [{
            key: {
                "description_vector": "vector"
            },
            vectorOptions: {
                type: "hnsw",
                dimensions: 3,
                similarity: "cosine",
                m: 16,
                efConstruction: 64
            },
            name: "description_index"
        }]
    }
);
```

쿼리 예제

```
db.products.aggregate([
  { $vectorSearch: {
```

```
      index: "description_index",
      limit: 2,
      numCandidates: 10,
      path: "description_vector",
      queryVector: [0.1, 0.2, 0.3]
    }
  }
]);
```

출력

```
[
  {
    "_id": 1,
    "name": "Product A",
    "description": "A high-quality, eco-friendly product for your home.",
    "description_vector": [ 0.2, 0.5, 0.8 ]
  },
  {
    "_id": 3,
    "name": "Product C",
    "description": "A comfortable and stylish piece of furniture.",
    "description_vector": [ 0.1, 0.2, 0.4 ]
  }
]
```

코드 예제

$vectorSearch 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findSimilarProducts(queryVector) {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  const result = await collection.aggregate([
    { $vectorSearch: {
```

```
        index: "description_index",
        limit: 2,
        numCandidates: 10,
        path: "description_vector",
        queryVector: queryVector
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

findSimilarProducts([0.1, 0.2, 0.3]);
```

Python

```
from pymongo import MongoClient


def find_similar_products(query_vector):
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.products

    result = list(collection.aggregate([
        {
            '$vectorSearch': {
                'index': "description_index",
                'limit': 2,
                'numCandidates': 10,
                'path': "description_vector",
                'queryVector': query_vector
            }
        }
    ]))

    print(result)
    client.close()

find_similar_products([0.1, 0.2, 0.3])
```

$week

Amazon DocumentDB의 $week 연산자는 ISO 8601 표준에 따라 날짜의 주 번호(0~53)를 반환합니다. 주 번호는 연도와 요일을 기준으로 계산되며, 월요일은 해당 주의 첫 번째 날입니다.

파라미터

없음

예제(MongoDB 쉘)

다음 예제에서는 $week 연산자를 사용하여 지정된 날짜의 주 번호를 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.events.insertMany([
  { _id: 1, date: new Date("2023-01-01") },
  { _id: 2, date: new Date("2023-01-08") },
  { _id: 3, date: new Date("2023-12-31") }
]);
```

쿼리 예제

```
db.events.aggregate([
  { $project: {
    _id: 1,
    week: { $week: "$date" }
  }}
]);
```

출력

```
[
  { "_id": 1, "week": 1 },
  { "_id": 2, "week": 2 },
  { "_id": 3, "week": 53 }
]
```

코드 예제

$week 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const events = db.collection('events');

  const result = await events.aggregate([
    { $project: {
      _id: 1,
      week: { $week: "$date" }
    }}
  ]).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
  client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
  db = client['test']
  events = db['events']

  result = list(events.aggregate([
      {'$project': {
          '_id': 1,
          'week': {'$week': '$date'}
      }}
  ]))

  print(result)
```

```
  client.close()

example()
```

$year

Amazon DocumentDB의 $year 연산자는 날짜 또는 타임스탬프에서 연도 구성 요소를 추출합니다.

파라미터

- expression: 연도 구성 요소를 추출할 날짜 또는 타임스탬프 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $year 연산자를 사용하여 날짜 필드에서 연도 구성 요소를 추출하는 방법을 보여줍니다.

샘플 문서 생성

```
db.events.insertMany([
  { "_id": 1, "date": ISODate("2023-04-15T00:00:00Z") },
  { "_id": 3, "date": ISODate("2021-12-31T00:00:00Z") }
]);
```

쿼리 예제

```
db.events.aggregate([
  { $project: { year: { $year: "$date" } } }
]);
```

출력

```
[
  { "_id": 1, "year": 2023 },
  { "_id": 3, "year": 2021 }
]
```

코드 예제

$year 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('events');

  const result = await collection.aggregate([
    { $project: { year: { $year: "$date" } } }
  ]).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['events']

    result = list(collection.aggregate([
        {'$project': {'year': {'$year': '$date'}}}
    ]))

    print(result)

    client.close()

example()
```

$zip

Amazon DocumentDB의 $zip 연산자를 사용하면 여러 배열을 단일 튜플 배열(키-값 페어)로 결합할 수 있습니다. 이는 문서 내에서 다양한 소스 또는 배열의 데이터를 결합하여 새 문서 또는 객체를 생성해야 할 때 유용합니다.

파라미터

- inputs: 배열로 확인되는 표현식의 배열입니다. 이러한 배열은 단일 튜플 배열로 결합됩니다.
- useLongestLength: (선택 사항) true인 경우 출력 배열은 가장 긴 입력 배열의 길이를 가지며 더 짧은 배열을 null 값으로 패딩합니다. false인 경우 출력 배열은 가장 짧은 입력 배열의 길이를 갖습니다.
- defaults: (선택 사항) 해당 입력 배열이 가장 긴 입력 배열보다 짧고 인 경우 튜플에 사용할 기본값 배열useLongestLength입니다true.

예제(MongoDB 쉘)

다음 예제에서는 $zip 연산자를 사용하여 두 배열을 단일 튜플 배열로 결합하는 방법을 보여줍니다.

샘플 문서 생성

```
db.grades.insert([
  {
    "_id": 1,
    "name": "John",
    "scores": [90, 85, 92],
    "classes": ["Math", "English", "Science"]
  },
  {
    "_id": 2,
    "name": "Jane",
    "scores": [88, 91, 90, 85],
    "classes": ["Math", "English", "Science", "History"]
  }
])
```

쿼리 예제

```
db.grades.aggregate([
```

```
  {
    $project: {
      "name": 1,
      "scoredClasses": {
        $zip: {
          inputs: ["$scores", "$classes"],
          useLongestLength: true,
          defaults: [null, null]
        }
      }
    }
  }
])
```

출력

```
[
  {
    "_id": 1,
    "name": "John",
    "scoredClasses": [
      [90, "Math"],
      [85, "English"],
      [92, "Science"],
      [null, null]
    ]
  },
  {
    "_id": 2,
    "name": "Jane",
    "scoredClasses": [
      [88, "Math"],
      [91, "English"],
      [90, "Science"],
      [85, "History"]
    ]
  }
]
```

코드 예제

$zip 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('grades');

  const result = await collection.aggregate([
    {
      $project: {
        "name": 1,
        "scoredClasses": {
          $zip: {
            inputs: ["$scores", "$classes"],
            useLongestLength: true,
            defaults: [null, null]
          }
        }
      }
    }
  ]).toArray();

  console.log(result);
  client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['grades']
```

```
    result = list(collection.aggregate([
        {
            '$project': {
                'name': 1,
                'scoredClasses': {
                    '$zip': {
                        'inputs': ['$scores', '$classes'],
                        'useLongestLength': True,
                        'defaults': [None, None]
                    }
                }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

Geospatial

이 섹션에서는 Amazon DocumentDB에서 지원하는 지리 공간 연산자에 대한 자세한 정보를 제공합니다.

주제

- $geometry
- $geoIntersects
- $geoWithin
- $maxDistance
- $minDistance
- $near
- $nearSphere

$geometry

Amazon DocumentDB의 $geometry 연산자는 지리 공간 쿼리의 일부로 GeoJSON 지오메트리 객체를 지정하는 데 사용됩니다. 이 연산자는 $geoWithin 및와 같은 다른 지리 공간 쿼리 연산자와 함께 데이터에 대한 공간 쿼리를 수행하는 $geoIntersects 데 사용됩니다.

Amazon DocumentDB에서 $geometry 연산자는 다음 GeoJSON 지오메트리 유형을 지원합니다.

- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon
- GeometryCollection

파라미터

- type: GeoJSON 지오메트리 객체의 유형, 예: , Point Polygon등
- coordinates: 지오메트리를 나타내는 좌표 배열입니다. 좌표 배열의 구조는 지오메트리 유형에 따라 달라집니다.

예제(MongoDB 쉘)

다음 예제에서는 $geometry 연산자를 사용하여 Amazon DocumentDB에서 $geoIntersects 쿼리를 수행하는 방법을 보여줍니다.

샘플 문서 생성

```
db.locations.insertMany([
  {
    "_id": 1,
    "name": "Location 1",
    "location": {
      "type": "Point",
      "coordinates": [-73.983253, 40.753941]
    }
  },
```

```
  {
    "_id": 2,
    "name": "Location 2",
    "location": {
      "type": "Polygon",
      "coordinates": [[
        [-73.998427, 40.730309],
        [-73.954348, 40.730309],
        [-73.954348, 40.780816],
        [-73.998427, 40.780816],
        [-73.998427, 40.730309]
      ]]
    }
  }
]);
```

쿼리 예제

```
db.locations.find({
  "location": {
    "$geoIntersects": {
      "$geometry": {
        "type": "Polygon",
        "coordinates": [[
          [-73.998, 40.730],
          [-73.954, 40.730],
          [-73.954, 40.781],
          [-73.998, 40.781],
          [-73.998, 40.730]
        ]]
      }
    }
  }
})
```

출력

```
[
  {
    "_id": 2,
    "name": "Location 2",
    "location": {
      "type": "Polygon",
```

```
      "coordinates": [
        [
          [-73.998427, 40.730309],
          [-73.954348, 40.730309],
          [-73.954348, 40.780816],
          [-73.998427, 40.780816],
          [-73.998427, 40.730309]
        ]
      ]
    }
  }
]
```

코드 예제

$geometry 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('locations');

  const query = {
    "location": {
      "$geoIntersects": {
        "$geometry": {
          "type": "Polygon",
          "coordinates": [[
            [-73.998, 40.730],
            [-73.954, 40.730],
            [-73.954, 40.781],
            [-73.998, 40.781],
            [-73.998, 40.730]
          ]]
        }
      }
    }
  };
```

```
  const result = await collection.find(query).toArray();
  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['locations']

    query = {
        "location": {
            "$geoIntersects": {
                "$geometry": {
                    "type": "Polygon",
                    "coordinates": [[
                        [-73.998, 40.730],
                        [-73.954, 40.730],
                        [-73.954, 40.781],
                        [-73.998, 40.781],
                        [-73.998, 40.730]
                    ]]
                }
            }
        }
    }

    result = list(collection.find(query))
    print(result)

    client.close()

example()
```

$geoIntersects

Amazon DocumentDB의 `$geoIntersects` 연산자는 지리 공간 데이터가 지정된 GeoJSON 객체와 교차하는 문서를 찾는 데 사용됩니다. 이 연산자는 다각형 또는 다중 다각형과 같은 지정된 지리적 형태와의 공간 관계를 기반으로 문서를 식별해야 하는 애플리케이션에 유용합니다.

파라미터

- `$geometry`: 교차를 확인할 셰이프를 나타내는 GeoJSON 객체입니다. 지원되는 GeoJSON 객체 유형은 `Point`, `LineString`, `Polygon`및 입니다`MultiPolygon`.

예제(MongoDB 쉘)

다음 예제에서는 `$geoIntersects` 연산자를 사용하여 Amazon DocumentDB에서 지정된 좌표 집합의 상태 이름을 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.states.insertMany([
  {
    "name": "New York",
    "loc": {
      "type": "Polygon",
      "coordinates": [[
        [-74.25909423828125, 40.47556838210948],
        [-73.70819091796875, 40.47556838210948],
        [-73.70819091796875, 41.31342607582222],
        [-74.25909423828125, 41.31342607582222],
        [-74.25909423828125, 40.47556838210948]
      ]]
    }
  },
  {
    "name": "California",
    "loc": {
      "type": "Polygon",
      "coordinates": [[
        [-124.4091796875, 32.56456771381587],
        [-114.5458984375, 32.56456771381587],
        [-114.5458984375, 42.00964153424558],
        [-124.4091796875, 42.00964153424558],
        [-124.4091796875, 32.56456771381587]
```

```
      ]]
    }
  }
]);
```

쿼리 예제

```
var location = [-73.965355, 40.782865];

db.states.find({
  "loc": {
    "$geoIntersects": {
      "$geometry": {
        "type": "Point",
        "coordinates": location
      }
    }
  }
}, {
  "name": 1
});
```

출력

```
{ "_id" : ObjectId("536b0a143004b15885c91a2c"), "name" : "New York" }
```

코드 예제

$geoIntersects 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findStateByGeoIntersects(longitude, latitude) {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('states');

  const query = {
```

```
    loc: {
      $geoIntersects: {
        $geometry: {
          type: 'Point',
          coordinates: [longitude, latitude]
        }
      }
    }
  };

  const projection = {
    _id: 0,
    name: 1
  };

  const document = await collection.findOne(query, { projection });

  await client.close();

  if (document) {
    return document.name;
  } else {
    throw new Error('The geo location you entered was not found in the United
 States!');
  }
}
```

Python

```
from pymongo import MongoClient

def find_state_by_geointersects(longitude, latitude):
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
        db = client.test
        collection_states = db.states

        query_geointersects = {
            "loc": {
                "$geoIntersects": {
                    "$geometry": {
```

```
                        "type": "Point",
                        "coordinates": [longitude, latitude]
                    }
                }
            }
        }

        document = collection_states.find_one(query_geointersects,
                                              projection={
                                                  "_id": 0,
                                                  "name": 1
                                              })
        if document is not None:
            state_name = document['name']
            return state_name
        else:
            raise Exception("The geo location you entered was not found in the
United States!")
    except Exception as e:
        print('Exception in geoIntersects: {}'.format(e))
        raise
    finally:
        if client is not None:
            client.close()
```

$geoWithin

Amazon DocumentDB의 `$geoWithin` 연산자는 위치 데이터(GeoJSON 객체로 표시됨)가 다각형 또는 다중 다각형과 같이 지정된 셰이프 내에 완전히 포함된 문서를 찾는 데 사용됩니다. 이는 특정 지리적 리전 내에 있는 객체를 쿼리하는 데 유용합니다.

파라미터

- `$geometry`: 쿼리할 셰이프를 나타내는 GeoJSON 객체입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$geoWithin` 운영자를 사용하여 뉴욕 주 내에 있는 모든 공항을 찾는 방법을 보여줍니다.

샘플 문서 생성

```
// Insert state document
db.states.insert({
    "name": "New York",
    "loc": {
        "type": "Polygon",
        "coordinates": [[
            [-79.76278, 45.0],
            [-73.94, 45.0],
            [-73.94, 40.5],
            [-79.76278, 40.5],
            [-79.76278, 45.0]
        ]]
    }
});

// Insert airport documents
db.airports.insert([
    {
        "name": "John F. Kennedy International Airport",
        "type": "airport",
        "code": "JFK",
        "loc": {
            "type": "Point",
            "coordinates": [-73.7781, 40.6413]
        }
    },
    {
        "name": "LaGuardia Airport",
        "type": "airport",
        "code": "LGA",
        "loc": {
            "type": "Point",
            "coordinates": [-73.8772, 40.7769]
        }
    },
    {
        "name": "Buffalo Niagara International Airport",
        "type": "airport",
        "code": "BUF",
        "loc": {
            "type": "Point",
            "coordinates": [-78.7322, 42.9403]
        }
```

```
    }
]);
```

쿼리 예제

```
var state = db.states.findOne({"name": "New York"});

db.airports.find({
    "loc": {
        "$geoWithin": {
            "$geometry": state.loc
        }
    }
}, {
    "name": 1,
    "type": 1,
    "code": 1,
    "_id": 0
});
```

출력

```
[
  {
    "name": "John F. Kennedy International Airport",
    "type": "airport",
    "code": "JFK"
  },
  {
    "name": "LaGuardia Airport",
    "type": "airport",
    "code": "LGA"
  }
]
```

코드 예제

`$geoWithin` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function findAirportsWithinState(stateName) {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');

  const stateDoc = await db.collection('states').findOne({ name: stateName },
 { projection: { _id: 0, loc: 1 } });
  const airportDocs = await db.collection('airports').find({
    loc: {
      $geoWithin: {
        $geometry: stateDoc.loc
      }
    }
  }, { projection: { name: 1, type: 1, code: 1, _id: 0 } }).toArray();

  console.log(airportDocs);

  await client.close();
}

findAirportsWithinState('New York');
```

Python

```
from pymongo import MongoClient

def find_airports_within_state(state_name):
    try:
        client = MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
        db = client['test']
        state_doc = db.states.find_one({'name': state_name}, {'_id': 0, 'loc': 1})
        airport_docs = db.airports.find({
            'loc': {
                '$geoWithin': {
                    '$geometry': state_doc['loc']
                }
            }
        }, {'name': 1, 'type': 1, 'code': 1, '_id': 0})
```

```
        return list(airport_docs)
    except Exception as e:
        print(f'Error: {e}')
    finally:
        client.close()

airports = find_airports_within_state('New York')
print(airports)
```

$maxDistance

Amazon DocumentDB의 $maxDistance 연산자는 문서가 쿼리 결과에 포함되어야 하는 GeoJSON 지점과의 최대 거리(미터)를 지정하는 데 사용됩니다. 이 연산자는 $nearSphere 연산자와 함께 사용하여 지리 공간 쿼리를 수행합니다.

파라미터

- $maxDistance: 문서가 쿼리 결과에 포함되어야 하는 참조 지점과의 최대 거리(미터)입니다.

예제(MongoDB 쉘)

다음 예제에서는 Amazon DocumentDB에서 $maxDistance 연산자를 사용하여 보스턴에서 100킬로미터 이내에 있는 모든 주 수도를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.capitals.insert([
  { state: "Massachusetts", city: "Boston", location: { type: "Point", coordinates:
 [-71.0589, 42.3601] } },
  { state: "Rhode Island", city: "Providence", location: { type: "Point", coordinates:
 [-71.4128, 41.8239] } },
  { state: "New Hampshire", city: "Concord", location: { type: "Point", coordinates:
 [-71.5383, 43.2067] } },
  { state: "Vermont", city: "Montpelier", location: { type: "Point", coordinates:
 [-72.5751, 44.2604] } }
]);
```

쿼리 예제

```
db.capitals.find(
  {
```

```
    location: {
      $nearSphere: {
        $geometry: { type: "Point", coordinates: [-71.0589, 42.3601] },
        $maxDistance: 100000
      }
    }
  },
  { state: 1, city: 1, _id: 0 }
);
```

출력

```
[
  { "state": "Rhode Island", "city": "Providence" },
  { "state": "New Hampshire", "city": "Concord" }
]
```

코드 예제

$maxDistance 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findCapitalsNearBoston() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const capitals = db.collection('capitals');

  const result = await capitals.find({
    location: {
      $nearSphere: {
        $geometry: { type: "Point", coordinates: [-71.0589, 42.3601] },
        $maxDistance: 100000
      }
    }
  }, {
    projection: { state: 1, city: 1, _id: 0 }
  }).toArray();
```

```
  console.log(result);
  await client.close();
}

findCapitalsNearBoston();
```

Python

```
from pymongo import MongoClient

def find_capitals_near_boston():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    capitals = db.capitals

    result = list(capitals.find(
        {
            "location": {
                "$nearSphere": {
                    "$geometry": { "type": "Point", "coordinates": [-71.0589,
 42.3601] },
                    "$maxDistance": 100000
                }
            }
        },
        {"state": 1, "city": 1, "_id": 0}
    ))

    print(result)
    client.close()

find_capitals_near_boston()
```

$minDistance

$minDistance는 중심점에서 지정된 최소 거리 이상인 문서를 필터링$geoNear하기 위해 $nearSphere 또는와 함께 사용되는 찾기 연산자입니다. 이 연산자는 Amazon DocumentDB에서 지원되며 MongoDB의 해당 연산자와 유사하게 작동합니다.

파라미터

- $minDistance: 결과에 문서를 포함할 중심점으로부터의 최소 거리(미터)입니다.

예제(MongoDB 쉘)

이 예제에서는 워싱턴주 시애틀의 특정 위치에서 반경 2킬로미터 이내에 있는 모든 레스토랑을 찾습니다.

샘플 문서 생성

```
db.usarestaurants.insertMany([
  {
    "state": "Washington",
    "city": "Seattle",
    "name": "Noodle House",
    "rating": 4.8,
    "location": {
      "type": "Point",
      "coordinates": [-122.3517, 47.6159]
    }
  },
  {
    "state": "Washington",
    "city": "Seattle",
    "name": "Pike Place Grill",
    "rating": 4.5,
    "location": {
      "type": "Point",
      "coordinates": [-122.3412, 47.6102]
    }
  },
  {
    "state": "Washington",
    "city": "Bellevue",
    "name": "The Burger Joint",
    "rating": 4.2,
    "location": {
      "type": "Point",
      "coordinates": [-122.2007, 47.6105]
    }
  }
]);
```

쿼리 예제

```
db.usarestaurants.find({
  "location": {
    "$nearSphere": {
      "$geometry": {
        "type": "Point",
        "coordinates": [-122.3516, 47.6156]
      },
      "$minDistance": 1,
      "$maxDistance": 2000
    }
  }
}, {
  "name": 1
});
```

출력

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "name" : "Noodle House" }
{ "_id" : ObjectId("611f3da985009a81ad38e74c"), "name" : "Pike Place Grill" }
```

코드 예제

$minDistance 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findRestaurantsNearby() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('usarestaurants');

  const result = await collection.find({
    "location": {
      "$nearSphere": {
        "$geometry": {
          "type": "Point",
```

```
          "coordinates": [-122.3516, 47.6156]
        },
        "$minDistance": 1,
        "$maxDistance": 2000
      }
    }
  }, {
    "projection": { "name": 1 }
  }).toArray();

  console.log(result);
  client.close();
}

findRestaurantsNearby();
```

Python

```
from pymongo import MongoClient

def find_restaurants_nearby():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.usarestaurants

    result = list(collection.find({
        "location": {
            "$nearSphere": {
                "$geometry": {
                    "type": "Point",
                    "coordinates": [-122.3516, 47.6156]
                },
                "$minDistance": 1,
                "$maxDistance": 2000
            }
        }
    }, {
        "projection": {"name": 1}
    }))

    print(result)
```

```
    client.close()

find_restaurants_nearby()
```

$near

Amazon DocumentDB의 $near 연산자는 지정된 지점에 지리적으로 가까운 문서를 찾는 데 사용됩니다. 거리별로 정렬된 문서를 반환하고 가장 가까운 문서를 먼저 반환합니다. 이 연산자는 2dsphere 지리 공간 인덱스가 필요하며 위치 데이터에 대한 근접 쿼리에 유용합니다.

파라미터

- $geometry: 근거리 쿼리의 중심점을 정의하는 GeoJSON Point 객체입니다.
- $maxDistance: (선택 사항) 문서가 쿼리와 일치할 수 있는 지정된 지점으로부터 미터 단위의 최대 거리입니다.
- $minDistance: (선택 사항) 문서가 쿼리와 일치할 수 있는 지정된 지점으로부터 미터 단위의 최소 거리입니다.

인덱스 요구 사항

- 2dsphere index: GeoJSON Point 데이터에 대한 지리 공간 쿼리에 필요합니다.

예제(MongoDB 쉘)

다음 예제에서는 $near 연산자를 사용하여 워싱턴주 시애틀의 특정 위치에 가장 가까운 레스토랑을 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.usarestaurants.insert([
  {
    "name": "Noodle House",
    "city": "Seattle",
    "state": "Washington",
    "rating": 4.8,
    "location": { "type": "Point", "coordinates": [-122.3517, 47.6159] }
  },
  {
```

```
    "name": "Pike Place Grill",
    "city": "Seattle",
    "state": "Washington",
    "rating": 4.2,
    "location": { "type": "Point", "coordinates": [-122.3403, 47.6062] }
  },
  {
    "name": "Lola",
    "city": "Seattle",
    "state": "Washington",
    "rating": 4.5,
    "location": { "type": "Point", "coordinates": [-122.3407, 47.6107] }
  }
]);
```

2Dsphere 인덱스 생성

```
db.usarestaurants.createIndex({ "location": "2dsphere" });
```

GeoJSON Point를 사용한 쿼리 예제

```
db.usarestaurants.find({
  location: {
    $near: {
      $geometry: {
        type: "Point",
        coordinates: [-122.3516, 47.6156]
      },
      $maxDistance: 100,
      $minDistance: 10
    }
  }
});
```

출력

```
{
  "_id" : ObjectId("69031ec9ea1c2922a1ce5f4a"),
  "name" : "Noodle House",
  "city" : "Seattle",
  "state" : "Washington",
  "rating" : 4.8,
```

```
  "location" : {
    "type" : "Point",
    "coordinates" : [ -122.3517, 47.6159 ]
  }
}
```

코드 예제

$near 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findNearbyRestaurants() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const restaurants = db.collection('usarestaurants');

  // Create 2dsphere index
  await restaurants.createIndex({ "location": "2dsphere" });

  const result = await restaurants.find({
    location: {
      $near: {
        $geometry: {
          type: "Point",
          coordinates: [-122.3516, 47.6156]
        },
        $maxDistance: 100,
        $minDistance: 10
      }
    }
  }).toArray();

  console.log(result);

  client.close();
}

findNearbyRestaurants();
```

Python

```
from pymongo import MongoClient

def find_nearby_restaurants():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    restaurants = db['usarestaurants']

    # Create 2dsphere index
    restaurants.create_index([("location", "2dsphere")])

    result = list(restaurants.find({
        'location': {
            '$near': {
                '$geometry': {
                    'type': 'Point',
                    'coordinates': [-122.3516, 47.6156]
                },
                '$maxDistance': 100,
                '$minDistance': 10
            }
        }
    }))

    print(result)

    client.close()

find_nearby_restaurants()
```

$nearSphere

Amazon DocumentDB의 $nearSphere 연산자는 지리 공간 지점에서 지정된 거리 내에 있는 문서를 찾는 데 사용됩니다. 이 연산자는 특정 위치의 특정 반경 내에서 모든 레스토랑을 찾는 등 지리 공간 쿼리에 특히 유용합니다.

파라미터

- `$geometry`: 참조 지점을 나타내는 GeoJSON 객체입니다. `type` 및 `coordinates` 필드가 있는 `Point` 객체여야 합니다.
- `$minDistance`: (선택 사항) 문서가 있어야 하는 참조 지점과의 최소 거리(미터)입니다.
- `$maxDistance`: (선택 사항) 문서가 있어야 하는 참조 지점과의 최대 거리(미터)입니다.

예제(MongoDB 쉘)

이 예제에서는 워싱턴주 시애틀의 특정 위치에서 2킬로미터(2,000미터) 이내에 있는 모든 레스토랑을 찾습니다.

샘플 문서 생성

```
db.usarestaurants.insert([
  {
    name: "Noodle House",
    location: { type: "Point", coordinates: [-122.3516, 47.6156] }
  },
  {
    name: "Pike Place Grill",
    location: { type: "Point", coordinates: [-122.3403, 47.6101] }
  },
  {
    name: "Seattle Coffee Co.",
    location: { type: "Point", coordinates: [-122.3339, 47.6062] }
  }
]);
```

쿼리 예제

```
db.usarestaurants.find({
  location: {
    $nearSphere: {
      $geometry: {
        type: "Point",
        coordinates: [-122.3516, 47.6156]
      },
      $minDistance: 1,
      $maxDistance: 2000
    }
  }
```

```
}, {
  name: 1
});
```

출력

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "name" : "Noodle House" }
{ "_id" : ObjectId("611f3da985009a81ad38e74c"), "name" : "Pike Place Grill" }
```

코드 예제

$nearSphere 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findNearbyRestaurants() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const restaurants = db.collection('usarestaurants');

  const result = await restaurants.find({
    location: {
      $nearSphere: {
        $geometry: {
          type: "Point",
          coordinates: [-122.3516, 47.6156]
        },
        $minDistance: 1,
        $maxDistance: 2000
      }
    }
  }, {
    projection: { name: 1 }
  }).toArray();

  console.log(result);
  client.close();
}
```

```
findNearbyRestaurants();
```

Python

```
from pymongo import MongoClient

def find_nearby_restaurants():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    restaurants = db.usarestaurants

    result = list(restaurants.find({
        'location': {
            '$nearSphere': {
                '$geometry': {
                    'type': 'Point',
                    'coordinates': [-122.3516, 47.6156]
                },
                '$minDistance': 1,
                '$maxDistance': 2000
            }
        }
    }, {
        'name': 1
    }))

    print(result)
    client.close()

find_nearby_restaurants()
```

쿼리 및 프로젝션 연산자

이 섹션에서는 Amazon DocumentDB에서 지원하는 쿼리 및 프로젝션 연산자에 대한 자세한 정보를 제공합니다.

주제

- $
- $all

- $and
- $bitsAllClear
- $bitsAllSet
- $bitsAnyClear
- $bitsAnySet
- $comment
- $elemMatch
- $eq
- $exists
- $expr
- $gt
- $gte
- $in
- $jsonSchema
- $lt
- $mod
- $lte
- $meta
- $ne
- $nin
- $nor
- $not
- $or
- $regex
- $sampleRate
- $slice
- $size
- $text
- $type

$

$ 프로젝션 연산자는 쿼리 조건과 일치하는 첫 번째 요소만 반환하도록 배열 필드의 내용을 제한합니다. 일치하는 단일 배열 요소를 프로젝션하는 데 사용됩니다.

파라미터

- `field.$`: 첫 번째 일치 요소를 투영하기 위한 위치 연산자가 있는 배열 필드입니다.

예제(MongoDB 쉘)

다음 예제에서는 $ 프로젝션 연산자를 사용하여 일치하는 배열 요소만 반환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.students.insertMany([
  { _id: 1, name: "Alice", grades: [85, 92, 78, 95] },
  { _id: 2, name: "Bob", grades: [70, 88, 92, 65] },
  { _id: 3, name: "Charlie", grades: [95, 89, 91, 88] }
]);
```

쿼리 예제

```
db.students.find(
  { grades: { $gte: 90 } },
  { name: 1, "grades.$": 1 }
);
```

출력

```
{ "_id" : 1, "name" : "Alice", "grades" : [ 92 ] }
{ "_id" : 2, "name" : "Bob", "grades" : [ 92 ] }
{ "_id" : 3, "name" : "Charlie", "grades" : [ 95 ] }
```

이 예제에서는 각 학생에 대해 90 이상의 첫 번째 학년만 반환됩니다.

코드 예제

$ 프로젝션 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('students');

  const result = await collection.find(
    { grades: { $gte: 90 } },
    { projection: { name: 1, "grades.$": 1 } }
  ).toArray();

  console.log(JSON.stringify(result, null, 2));
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['students']

    result = list(collection.find(
        {'grades': {'$gte': 90}},
        {'name': 1, 'grades.$': 1}
    ))

    print(result)
    client.close()

example()
```

$all

Amazon DocumentDB의 $all 연산자는 필드 값이 배열이고 배열의 요소 순서에 관계없이 지정된 모든 요소를 포함하는 문서를 일치시키는 데 사용됩니다.

파라미터

- field: 확인할 필드의 이름입니다.
- [value1, value2, ...]: 배열에서 일치시킬 값의 목록입니다.

$all 표현식 $elemMatch 내에서 사용

$all 표현식 내에서 $elemMatch 연산자를 사용하는 것과 관련된 $all 표현식 내에서 $elemMatch 사용 제한 사항은 섹션을 참조하세요.

필드 이름의 달러($)

중첩된 객체에서의 $ 접두사 필드 쿼리$all와 관련된 필드 이름의 달러($) 및 점(.) 제한 사항은 섹션을 참조하세요.

예제(MongoDB 쉘)

다음 예제에서는 연$all산자를 사용하여 "Colors" 필드가 "Red"와 "Blue"가 모두 포함된 배열인 문서를 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.example.insertMany([
  { "Item": "Pen", "Colors": ["Red", "Blue", "Green"] },
  { "Item": "Notebook", "Colors": ["Blue", "White"] },
  { "Item": "Poster Paint", "Colors": ["Red", "Yellow", "White"] }
])
```

쿼리 예제

```
db.example.find({ "Colors": { $all: ["Red", "Blue"] } }).pretty()
```

출력

```
{
  "_id" : ObjectId("6137d6c5b3a1d35e0b6ee6ad"),
  "Item" : "Pen",
  "Colors" : [
          "Red",
          "Blue",
          "Green"
  ]
}
```

코드 예제

$all 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('example');

  const result = await collection.find({ "Colors": { $all: ["Red",
 "Blue"] } }).toArray();
  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
    db = client['test']
    collection = db['example']

    result = list(collection.find({ "Colors": { "$all": ["Red", "Blue"] } }))
    print(result)

    client.close()

example()
```

$and

Amazon DocumentDB의 $and 연산자는 여러 표현식을 결합하고 단일 조건으로 평가하는 데 사용됩니다. 제공된 모든 표현식이 로 평가`true`되면를 반환하고`true`, `false` 그렇지 않으면를 반환합니다. 이 연산자는 쿼리에 여러 기준을 적용하는 데 유용합니다.

파라미터

- `expression1`: 부울 값으로 평가되는 필수 표현식입니다.
- `expression2`: 부울 값으로 평가되는 필수 표현식입니다.
- `...`: 부울 값으로 평가되는 추가 필수 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $and 연산자를 사용하여 "사용자" 모음에서 "나이" 필드가 18보다 크고 "상태" 필드가 "활성"인 모든 문서를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { name: "John", age: 25, status: "active" },
  { name: "Jane", age: 17, status: "active" },
  { name: "Bob", age: 30, status: "inactive" },
  { name: "Alice", age: 22, status: "active" }
]);
```

쿼리 예제

```
db.users.find({
```

```
  $and: [
    { age: { $gt: 18 } },
    { status: "active" }
  ]
});
```

출력

```
[
  { "_id" : ObjectId("614e3c4b63f5892e7c4e2345"), "name" : "John", "age" : 25,
 "status" : "active" },
  { "_id" : ObjectId("614e3c4b63f5892e7c4e2347"), "name" : "Alice", "age" : 22,
 "status" : "active" }
]
```

코드 예제

$and 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findActiveUsersOlderThan18() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const activeUsersOlderThan18 = await users.find({
    $and: [
      { age: { $gt: 18 } },
      { status: 'active' }
    ]
  }).toArray();

  console.log(activeUsersOlderThan18);
  await client.close();
}

findActiveUsersOlderThan18();
```

Python

```
from pymongo import MongoClient

def find_active_users_older_than_18():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    users = db['users']

    active_users_older_than_18 = list(users.find({
        '$and': [
            {'age': {'$gt': 18}},
            {'status': 'active'}
        ]
    }))

    print(active_users_older_than_18)
    client.close()

find_active_users_older_than_18()
```

$bitsAllClear

Amazon DocumentDB의 `$bitsAllClear` 연산자는 필드의 지정된 모든 비트가 지워지는 문서와 일치시키는 데 사용됩니다(0으로 설정). 이 연산자는 저장된 데이터에 대해 비트 단위 작업을 수행하는 데 유용할 수 있습니다.

파라미터

- `field`: 지워지는 지정된 비트를 확인할 필드입니다.
- `value`: 확인할 비트 또는 확인할 비트 위치 목록을 지정하는 숫자 비트 마스크입니다. 숫자 비트마스크는 바이너리(0b...), 십진수, 16진수(0x...), 8진수(0o...) 또는 바이너리(BinData) 형식일 수 있습니다. 비트 위치 목록에서 가장 중요하지 않은 비트의 위치는 0입니다.

예제(MongoDB 쉘)

다음 예제에서는 Amazon DocumentDB에서 `$bitsAllClear` 연산자의 사용을 보여줍니다.

샘플 문서 생성

```
db.collection.insertMany([
  { _id: 1, bits: 0b1010 },
  { _id: 2, bits: 0b1100 },
  { _id: 3, bits: 0b0101 }
]);
```

쿼리 예제

```
db.collection.find({
  bits: { $bitsAllClear: 0b0011 }
})
```

출력

```
{ "_id" : 2, "bits" : 12 }
```

쿼리는 비트 마스크에 의해 지정된 모든 비트0b0011(가장 중요하지 않은 두 비트)가 bits 필드에서 지워지는지 확인합니다. bits 필드의 비트가 지워져 있으므로 _id 2가 있는 문서는이 조건을 충족합니다.

코드 예제

$bitsAllClear 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.find({
    bits: { $bitsAllClear: 0b0011 }
  }).toArray();

  console.log(result);
```

```
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['collection']

    result = list(collection.find({
        'bits': { '$bitsAllClear': 0b0011 }
    }))

    print(result)

    client.close()

example()
```

$bitsAllSet

Amazon DocumentDB의 `$bitsAllSet` 연산자는 필드에서 특정 비트 집합이 모두 1로 설정된 문서를 쿼리하는 데 사용됩니다. 이 연산자를 사용하면 필드 값에 대해 비트 단위 작업을 수행할 수 있으며 숫자 값 내에서 개별 비트의 상태를 확인해야 할 때 유용할 수 있습니다.

파라미터

- `field`: 비트 단위 작업을 수행할 숫자 필드의 이름입니다.
- `value`: 확인할 비트 또는 확인할 비트 위치 목록을 지정하는 숫자 비트 마스크입니다. 숫자 비트마스크는 바이너리(0b...), 십진수, 16진수(0x...), 8진수(0o...) 또는 바이너리(BinData) 형식일 수 있습니다. 비트 위치 목록에서 가장 중요하지 않은 비트의 위치는 0입니다.

예제(MongoDB 쉘)

다음 예제에서는 $bitsAllSet 연산자를 사용하여 flags 필드에 비트 마스크에 의해 지정된 모든 비트가 설정된 문서를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.collection.insert([
  { _id: 1, flags: 0b1010 },
  { _id: 2, flags: 0b1100 },
  { _id: 3, flags: 0b1110 }
])
```

쿼리 예제

```
db.collection.find({ flags: { $bitsAllSet: 0b1100 } })
```

출력

```
{ "_id": 2, "flags": 12 },
{ "_id": 3, "flags": 14 }
```

이 예제에서 쿼리는 flags 필드에 비트 마스크0b1100(10진수 값 12를 나타냄)에 의해 지정된 모든 비트 세트가 있는 문서를 확인합니다. flags 필드 값에는 필요한 모든 비트가 설정되어 있으므로(3번째 및 4번째 최소 유효 비트) _id2와 3이 있는 문서는이 기준과 일치합니다.

코드 예제

$bitsAllSet 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findWithBitsAllSet() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('collection');
```

```
  const result = await collection.find({ flags: { $bitsAllSet:
 0b1100 } }).toArray();
  console.log(result);

  await client.close();
}

findWithBitsAllSet();
```

Python

```
from pymongo import MongoClient

def find_with_bits_all_set():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.collection

    result = list(collection.find({ 'flags': { '$bitsAllSet': 0b1100 } }))
    print(result)

    client.close()

find_with_bits_all_set()
```

$bitsAnyClear

Amazon DocumentDB의 `$bitsAnyClear` 연산자는 필드의 지정된 비트 중 하나가 지워지는 문서와 일치시키는 데 사용됩니다(0으로 설정). 이는 문서에 저장된 필드 값에 대해 비트 단위 작업을 수행하는 데 유용할 수 있습니다.

파라미터

- `field`: 확인할 필드입니다.
- `value`: 확인할 비트 또는 확인할 비트 위치 목록을 지정하는 숫자 비트 마스크입니다. 숫자 비트마스크는 바이너리(0b...), 십진수, 16진수(0x...), 8진수(0o...) 또는 바이너리(BinData) 형식일 수 있습니다. 비트 위치 목록에서 가장 중요하지 않은 비트의 위치는 0입니다.

예제(MongoDB 쉘)

다음 예제에서는 $bitsAnyClear 연산자를 사용하여 items 컬렉션의 status 필드에 비트가 명확한지 확인하는 방법을 보여줍니다.

샘플 문서 생성

```
db.items.insertMany([
  { "_id": 1, "status": 7 },
  { "_id": 2, "status": 15 },
  { "_id": 3, "status": 31 }
]);
```

쿼리 예제

```
db.items.find({ "status": { $bitsAnyClear: 8 } })
```

출력

```
{ "_id" : 1, "status" : 7 }
```

이 예제에서 쿼리는 status 필드가 비트 마스크(이진)에서 비트 지우기8(0)가 있는 문서를 확인합니다1000. status 값이 7 (이진 111)인 문서는 제공된 비트마스크에서 1비트 이상 지워지기 때문에 쿼리와 일치합니다. 일치하는 클리어 비트는 4번째로 중요하지 않은 비트입니다.

코드 예제

$bitsAnyClear 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('items');
```

```
  const result = await collection.find({ "status": { $bitsAnyClear:
 8 } }).toArray();
  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.items

    result = list(collection.find({ "status": { "$bitsAnyClear": 8 } }))
    print(result)

    client.close()

example()
```

$bitsAnySet

Amazon DocumentDB의 `$bitsAnySet` 연산자는 필드의 지정된 비트에서 최소 1비트가 1로 설정된 문서를 쿼리하는 데 사용됩니다. 이 연산자를 사용하면 필드에 저장된 값에 대해 비트 단위 작업을 수행할 수 있으므로 비트 단위 특성을 가진 데이터를 효율적으로 쿼리하고 분석할 수 있습니다.

파라미터

- `field`: 비트 단위 작업을 적용할 필드 이름입니다.
- `value`: 확인할 비트 또는 확인할 비트 위치 목록을 지정하는 숫자 비트 마스크입니다. 숫자 비트마스크는 바이너리(0b...), 십진수, 16진수(0x...), 8진수(0o...) 또는 바이너리(BinData) 형식일 수 있습니다. 비트 위치 목록에서 가장 중요하지 않은 비트의 위치는 0입니다.

예제(MongoDB 쉘)

다음 예제에서는 연$bitsAnySet산자를 사용하여 flags 필드에 하나 이상의 비트가 설정된 문서를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.collection.insertMany([
  { _id: 1, flags: 0b1010 },
  { _id: 2, flags: 0b1100 },
  { _id: 3, flags: 0b0011 },
  { _id: 4, flags: 0b0100 }
]);
```

쿼리 예제

```
db.collection.find({
  flags: { $bitsAnySet: 0b1010 }
})
```

출력

```
{ "_id" : 1, "flags" : 10 }
{ "_id" : 2, "flags" : 12 }
{ "_id" : 3, "flags" : 3 }
```

쿼리는 비트 마스크에 지정된 비트 중 하나 이상이 flags 필드에 0b1010 설정된 문서를 반환합니다.

코드 예제

$bitsAnySet 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
```

```
  const db = client.db('test');
  const collection = db.collection('collection');

  const result = await collection.find({
    flags: { $bitsAnySet: 0b1010 }
  }).toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['collection']

result = list(collection.find({
    'flags': { '$bitsAnySet': 0b1010 }
}))

print(result)

client.close()
```

$comment

Amazon DocumentDB의 `$comment` 연산자는 쿼리에 설명을 첨부하는 데 사용됩니다. 이는 쿼리에 대한 추가 컨텍스트 또는 정보를 제공하는 데 유용할 수 있으며, 이는 디버깅 또는 문서화 목적에 유용할 수 있습니다. 첨부된 주석은 db.currentOp()와 같은 작업 출력의 일부로 표시됩니다.

파라미터

- `string`: 쿼리에 연결된 주석입니다.

예제(MongoDB 쉘)

다음 예제에서는 Amazon DocumentDB에서 $comment 연산자를 사용하는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { name: "John Doe", age: 30, email: "john.doe@example.com" },
  { name: "Jane Smith", age: 25, email: "jane.smith@example.com" },
  { name: "Bob Johnson", age: 35, email: "bob.johnson@example.com" }
]);
```

쿼리 예제

```
db.users.find({ age: { $gt: 25 } }, { _id: 0, name: 1, age: 1 }).comment("Retrieve
 users older than 25");
```

출력

```
{ "name" : "John Doe", "age" : 30 }
{ "name" : "Bob Johnson", "age" : 35 }
```

코드 예제

$comment 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const result = await users.find({ age: { $gt: 25 } }, { projection: { _id: 0,
 name: 1, age: 1 } })
                            .comment('Retrieve users older than 25')
                            .toArray();
```

```
  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    users = db.users

    result = list(users.find({ 'age': { '$gt': 25 }}, { '_id': 0, 'name': 1, 'age':
 1 })
                .comment('Retrieve users older than 25'))

    print(result)

    client.close()

if __name__ == '__main__':
    main()
```

$elemMatch

Amazon DocumentDB의 `$elemMatch` 연산자는 배열 필드를 쿼리하고 배열의 하나 이상의 요소가 지정된 기준과 일치하는 문서를 반환하는 데 사용됩니다. 이 연산자는 중첩 배열 또는 임베디드 문서가 있는 복잡한 데이터 구조가 있는 경우에 특히 유용합니다.

플래너 버전 2.0에에 대한 인덱스 지원이 추가되었습니다`$elemMatch`.

파라미터

- `field`: 쿼리할 배열 필드입니다.
- `query`: 배열 요소와 일치시킬 기준입니다.

$all 표현식 **$elemMatch** 내에서 사용

$all 표현식 내에서 $elemMatch 연산자를 사용하는 것과 관련된 $all 표현식 내에서 $elemMatch 사용 제한 사항은 섹션을 참조하세요.

예제(MongoDB 쉘)

다음 예제에서는 $elemMatch 연산자를 사용하여 parts 배열에 지정된 기준과 일치하는 요소가 하나 이상 있는 문서를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.col.insertMany([
  { _id: 1, parts: [{ part: "xyz", qty: 10 }, { part: "abc", qty: 20 }] },
  { _id: 2, parts: [{ part: "xyz", qty: 5 }, { part: "abc", qty: 10 }] },
  { _id: 3, parts: [{ part: "xyz", qty: 15 }, { part: "abc", qty: 100 }] },
  { _id: 4, parts: [{ part: "abc", qty: 150 }] }
]);
```

쿼리 예제

```
db.col.find({
  parts: { "$elemMatch": { part: "xyz", qty: { $lt: 11 } } }
})
```

출력

```
{ "_id" : 1, "parts" : [ { "part" : "xyz", "qty" : 10 }, { "part" : "abc", "qty" :
 20 } ] }
{ "_id" : 2, "parts" : [ { "part" : "xyz", "qty" : 5 }, { "part" : "abc", "qty" :
 10 } ] }
```

코드 예제

$elemMatch 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const col = db.collection('col');

  const result = await col.find({
    parts: {
      "$elemMatch": { part: "xyz", qty: { $lt: 11 } }
    }
  }).toArray();

  console.log(JSON.stringify(result, null, 2));
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    col = db['col']

    result = list(col.find({
      'parts': {
        '$elemMatch': {'part': 'xyz', 'qty': {'$lt': 11}}
      }
    }))

    print(result)
    client.close()

example()
```

$eq

Amazon DocumentDB의 $eq 연산자는 필드 값이 지정된 값과 동일한 문서를 일치시키는 데 사용됩니다. 이 연산자는 일반적으로 find() 메서드에서 지정된 기준을 충족하는 문서를 검색하는 데 사용됩니다.

파라미터

- field : 등식 조건을 확인할 필드입니다.
- value : 필드와 비교할 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 연$eq산자를 사용하여 name 필드가와 동일한 모든 문서를 찾는 방법을 보여줍니다"Thai Curry Palace".

샘플 문서 생성

```
db.restaurants.insertMany([
  { name: "Thai Curry Palace", cuisine: "Thai", features: ["Private Dining"] },
  { name: "Italian Bistro", cuisine: "Italian", features: ["Outdoor Seating"] },
  { name: "Mexican Grill", cuisine: "Mexican", features: ["Takeout"] }
]);
```

쿼리 예제

```
db.restaurants.find({ name: { $eq: "Thai Curry Palace" } });
```

출력

```
{ "_id" : ObjectId("68ee586f916df9d39f3d9414"), "name" : "Thai Curry Palace",
 "cuisine" : "Thai", "features" : [ "Private Dining" ] }
```

코드 예제

$eq 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function findByName(name) {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('restaurants');

  const results = await collection.find({ name: { $eq: name } }).toArray();
  console.log(results);

  await client.close();
}

findByName("Thai Curry Palace");
```

Python

```
from pymongo import MongoClient

def find_by_name(name):
    client = MongoClient("mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false")
    db = client["test"]
    collection = db["restaurants"]

    results = list(collection.find({ "name": { "$eq": name } }))
    print(results)

    client.close()

find_by_name("Thai Curry Palace")
```

$exists

`$exists` 연산자는 필드가 문서에 존재하는지 여부를 확인하는 데 사용됩니다. 인덱싱된 필드가 모든 문서에 존재하지 않을 수 있는 Amazon DocumentDB에서 희소 인덱스로 작업할 때 `$exists` 특히 유용합니다.

쿼리에서 생성된 희소 인덱스를 사용하려면 인덱스를 포함하는 필드에서 `$exists` 절을 사용해야 합니다. 를 생략하면 `$exists` Amazon DocumentDB는 쿼리에 희소 인덱스를 사용하지 않습니다.

파라미터

- `field`: 존재 여부를 확인할 필드 이름입니다.
- `value`: 필드가 일치하는 문서에 존재하는지(`true`false) 또는 존재하지 않는지(`true`) 지정하는 부울 값(또는 `false`)입니다.

예제(MongoDB 쉘)

다음 예제에서는 `food` 컬렉션의 `special_diets` 필드에 희소 인덱스가 있는 `$exists` 연산자를 사용하는 방법을 보여줍니다.

샘플 문서 생성

```
db.food.insertMany([
  { _id: 1, name: "Apple", special_diets: ["vegetarian", "gluten-free"] },
  { _id: 2, name: "Broccoli" },
  { _id: 3, name: "Chicken", special_diets: ["dairy-free"] }
]);
```

'special_diets' 필드에 희소 인덱스 생성

```
db.food.createIndex({ "special_diets": 1 }, { sparse: true, name:
 "special_diets_sparse_asc" });
```

쿼리 예제

```
db.food.find({ "special_diets": { $exists: true } });
```

출력

```
[
  { "_id" : 1, "name" : "Apple", "special_diets" : [ "vegetarian", "gluten-free" ] },
  { "_id" : 3, "name" : "Chicken", "special_diets" : [ "dairy-free" ] }
]
```

코드 예제

`$exists` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('food');

  const result = await collection.find({ "special_diets": { $exists:
 true } }).toArray();
  console.log(result);

  await client.close();
}

main();
```

Python

```
import pymongo

client = pymongo.MongoClient('mongodb://<username>:<password>@<cluster-
endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['food']

result = list(collection.find({"special_diets": {"$exists": True}}))
print(result)

client.close()
```

$expr

버전 4.0의 새 버전입니다.

Elastic 클러스터에서는 지원되지 않습니다.

Amazon DocumentDB의 $expr 연산자를 사용하면 쿼리 언어 내에서 집계 표현식을 사용할 수 있습니다. 집계 파이프라인 단계를 사용하는 방식과 마찬가지로 문서 내의 필드에 대해 복잡한 비교 및 계산을 수행할 수 있습니다.

연산자에 배치된 $eq, $lt, ltegt, 및 $gte 비교 $expr 연산자는 $lookup 단계에서 참조되는 from 컬렉션에 대한 인덱스를 포함하여 인덱스를 사용할 수 있습니다.

제한 사항

- 인덱스는 필드와 상수 간의 비교에만 사용할 수 있으므로 피연산자는 상수로 확인되어야 합니다. 예를 들어, $a와 상수 값 간의 비교는 인덱스를 사용할 수 있지만 $a와 간의 비교는 사용할 수 $b 없습니다.
- 멀티키, 부분 또는 스파스 인덱스는 사용되지 않습니다. 이러한 경우 쿼리는 컬렉션 스캔으로 돌아갑니다.
- 피연산자가 let 비어 있거나 누락된 값으로 확인되면 비교는 대상 인덱스 검색을 사용할 수 없습니다. 이 경우 Amazon DocumentDB는 전체 인덱스 스캔을 수행합니다.

파라미터

- expression: 문서 필드에 대해 비교 및 계산을 수행할 수 있도록 부울 값을 반환하는 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $expr 연산자를 사용하여 manufacturingCost 필드가 price 필드보다 큰 모든 문서를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.inventory.insertMany([
  { item: "abc", manufacturingCost: 500, price: 100 },
  { item: "def", manufacturingCost: 300, price: 450 },
  { item: "ghi", manufacturingCost: 400, price: 120 }
]);
```

쿼리 예제

```
db.inventory.find({
  $expr: {
```

```
    $gt: ["$manufacturingCost", "$price"]
  }
})
```

출력

```
{ "_id" : ObjectId("60b9d4d68d2cac581bc5a89a"), "item" : "abc", "manufacturingCost" :
 500, "price" : 100 },
{ "_id" : ObjectId("60b9d4d68d2cac581bc5a89c"), "item" : "ghi", "manufacturingCost" :
 400, "price" : 120 }
```

코드 예제

$expr 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('inventory');

  const result = await collection.find({
    $expr: {
      $gt: ['$manufacturingCost', '$price']
    }
  }).toArray();

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['inventory']

    result = list(collection.find({
        '$expr': {
            '$gt': ['$manufacturingCost', '$price']
        }
    }))

    print(result)
    client.close()

example()
```

$gt

Amazon DocumentDB의 $gt 연산자는 지정된 필드의 값이 지정된 값보다 큰 문서를 선택하는 데 사용됩니다. 이 연산자는 수치 비교를 기반으로 데이터를 필터링하고 쿼리하는 데 유용합니다.

파라미터

- field: 비교할 필드입니다.
- value: 비교할 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 $gt 연산자를 사용하여 age 필드가 30보다 큰 모든 문서를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { name: "John", age: 25 },
  { name: "Jane", age: 32 },
  { name: "Bob", age: 45 },
  { name: "Alice", age: 28 }
]);
```

쿼리 예제

```
db.users.find({ age: { $gt: 30 } });
```

출력

```
{ "_id" : ObjectId("6249e5c22a5d39884a0a0001"), "name" : "Jane", "age" : 32 },
{ "_id" : ObjectId("6249e5c22a5d39884a0a0002"), "name" : "Bob", "age" : 45 }
```

코드 예제

$gt 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findUsersOlderThan30() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = await db.collection('users').find({ age: { $gt: 30 } }).toArray();
  console.log(users);
  await client.close();
}

findUsersOlderThan30();
```

Python

```
from pymongo import MongoClient

def find_users_older_than_30():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    users = list(db.users.find({ 'age': { '$gt': 30 } }))
    print(users)
    client.close()

find_users_older_than_30()
```

$gte

Amazon DocumentDB의 $gte 연산자는 지정된 값보다 크거나 같은 값을 일치시키는 데 사용됩니다. 이 연산자는 수치 비교를 기반으로 데이터를 필터링하고 쿼리하는 데 유용합니다.

파라미터

- `field`: 제공된 값과 비교하여 확인할 필드입니다.
- `value`: 필드와 비교할 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 Amazon DocumentDB에서 $gte 연산자를 사용하여 "나이" 필드가 25보다 크거나 같은 모든 문서를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { _id: 1, name: "John", age: 20 },
  { _id: 2, name: "Jane", age: 25 },
  { _id: 3, name: "Bob", age: 30 },
  { _id: 4, name: "Alice", age: 35 }
]);
```

쿼리 예제

```
db.users.find({ age: { $gte: 25 } }, { _id: 0, name: 1, age: 1 });
```

출력

```
{ "name" : "Jane", "age" : 25 }
{ "name" : "Bob", "age" : 30 }
{ "name" : "Alice", "age" : 35 }
```

코드 예제

$gte 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');
```

```
async function findUsersAboveAge(age) {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const result = await users.find({ age: { $gte: age } }, { projection: { _id: 0,
 name: 1, age: 1 } }).toArray();
  console.log(result);

  await client.close();
}

findUsersAboveAge(25);
```

Python

```
from pymongo import MongoClient

def find_users_above_age(age):
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    users = db.users

    result = list(users.find({ 'age': { '$gte': age } }, { '_id': 0, 'name': 1,
 'age': 1 }))
    print(result)

    client.close()

find_users_above_age(25)
```

$in

Amazon DocumentDB의 $in 연산자는 필드 값이 배열에 지정된 값과 동일한 문서를 찾을 수 있는 논리적 쿼리 연산자입니다.

파라미터

- `field`: 제공된 배열과 비교하여 확인할 필드입니다.
- `[value1, value2, ...]`: 지정된 필드와 일치시킬 값의 배열입니다.

필드 이름의 달러($)

중첩된 객체에서의 $ 접두사 필드 쿼리$in와 관련된 필드 이름의 달러($) 및 점(.) 제한 사항은 섹션을 참조하세요.

예제(MongoDB 쉘)

다음 예제에서는 $in 연산자를 사용하여 color 필드가 제공된 배열의 값 중 하나인 문서를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.colors.insertMany([
  { "_id": 1, "color": "red" },
  { "_id": 2, "color": "green" },
  { "_id": 3, "color": "blue" },
  { "_id": 4, "color": "yellow" },
  { "_id": 5, "color": "purple" }
])
```

쿼리 예제

```
db.colors.find({ "color": { "$in": ["red", "blue", "purple"] } })
```

출력

```
{ "_id": 1, "color": "red" },
{ "_id": 3, "color": "blue" },
{ "_id": 5, "color": "purple" }
```

코드 예제

$in 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findByIn() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('colors');

  const result = await collection.find({ "color": { "$in": ["red", "blue",
 "purple"] } }).toArray();
  console.log(result);

  await client.close();
}

findByIn();
```

Python

```
from pymongo import MongoClient

def find_by_in():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.colors

    result = list(collection.find({ "color": { "$in": ["red", "blue",
 "purple"] } }))
    print(result)

    client.close()

find_by_in()
```

$jsonSchema

버전 4.0의 새 버전입니다.

Elastic 클러스터에서는 지원되지 않습니다.

Amazon DocumentDB의 $jsonSchema 연산자는 지정된 JSON 스키마를 기반으로 문서를 필터링하는 데 사용됩니다. 이 연산자를 사용하면 특정 JSON 스키마와 일치하는 문서를 쿼리하여 검색된 문서가 특정 구조 및 데이터 유형 요구 사항을 준수하는지 확인할 수 있습니다.

$jsonSchema 평가 쿼리 연산자를 컬렉션 생성의 일부로 사용하여 컬렉션에 삽입되는 문서의 스키마를 검증할 수 있습니다. 추가 정보는 JSON 스키마 검증 사용을 참조하세요.

파라미터

- required (배열): 문서의 필수 필드를 지정합니다.
- properties (객체): 문서의 각 필드에 대한 데이터 유형 및 기타 제약 조건을 정의합니다.

예제(MongoDB 쉘)

다음 예제에서는 연$jsonSchema산자를 사용하여 name, employeeId 및 age 필드가 있고 employeeId 필드가 유형인 문서만 검색하도록 employees 컬렉션을 필터링하는 방법을 보여줍니다string.

샘플 문서 생성

```
db.employees.insertMany([
  { "name": { "firstName": "Carol", "lastName": "Smith" }, "employeeId": "1" },
  { "name": { "firstName": "Emily", "lastName": "Brown" }, "employeeId": "2", "age":
 25 },
  { "name": { "firstName": "William", "lastName": "Taylor" }, "employeeId": 3, "age":
 24 },
  { "name": { "firstName": "Jane", "lastName": "Doe" }, "employeeId": "4" }
]);
```

쿼리 예제

```
db.employees.aggregate([
  { $match: {
    $jsonSchema: {
      required: ["name", "employeeId", "age"],
      properties: { "employeeId": { "bsonType": "string" } }
    }
  }}
]);
```

출력

```
{ "_id" : ObjectId("6908e8b61f77fc26b2ecd26f"), "name" : { "firstName" : "Emily",
 "lastName" : "Brown" }, "employeeId" : "2", "age" : 25 }
```

코드 예제

$jsonSchema 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function filterByJsonSchema() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('employees');

  const result = await collection.aggregate([
    {
      $match: {
        $jsonSchema: {
          required: ['name', 'employeeId', 'age'],
          properties: { 'employeeId': { 'bsonType': 'string' } }
        }
      }
    }
  ]).toArray();

  console.log(result);
  await client.close();
}

filterByJsonSchema();
```

Python

```
from pymongo import MongoClient

def filter_by_json_schema():
```

```
  client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
  db = client['test']
  collection = db['employees']

  result = list(collection.aggregate([
    {
      '$match': {
        '$jsonSchema': {
          'required': ['name', 'employeeId', 'age'],
          'properties': {'employeeId': {'bsonType': 'string'}}
        }
      }
    }
  ]))

  print(result)
  client.close()

filter_by_json_schema()
```

$lt

Amazon DocumentDB의 `$lt` 연산자는 지정된 값보다 작은 값을 일치시키는 데 사용됩니다. 이 연산자는 수치 비교를 기반으로 데이터를 필터링하고 쿼리하는 데 유용합니다.

파라미터

- `field`: 확인할 필드 이름입니다.
- `value`: 비교할 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$lt` 연산자를 사용하여 `Inventory.OnHand` 값이 50 미만인 문서를 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.example.insertMany([
  { "_id": 1, "Inventory": { "OnHand": 25, "Sold": 60 }, "Colors": ["Red", "Blue"] },
```

```
  { "_id": 2, "Inventory": { "OnHand": 75, "Sold": 40 }, "Colors": ["Red", "White"] },
  { "_id": 3, "Inventory": { "OnHand": 10, "Sold": 85 }, "Colors": ["Red", "Green"] }
]);
```

쿼리 예제

```
db.example.find({ "Inventory.OnHand": { $lt: 50 } })
```

출력

```
{ "_id" : 1, "Inventory" : { "OnHand" : 25, "Sold" : 60 }, "Colors" : [ "Red",
 "Blue" ] }
{ "_id" : 3, "Inventory" : { "OnHand" : 10, "Sold" : 85 }, "Colors" : [ "Red",
 "Green" ] }
```

코드 예제

`$lt` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('example');

  const result = await collection.find({ "Inventory.OnHand": { $lt:
 50 } }).toArray();
  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient
```

```
def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['example']

    result = list(collection.find({ "Inventory.OnHand": { "$lt": 50 } }))
    print(result)

    client.close()

example()
```

$mod

$mod 쿼리 연산자는 필드 값을 디바이저로 나눈 나머지가 지정된 문서를 선택합니다. 이는 모듈로 산술 조건을 기반으로 문서를 필터링하는 데 유용합니다.

파라미터

- `divisor`: 나눌 숫자입니다.
- `remainder`: 예상되는 나머지 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 $mod 연산자를 사용하여 수량이 홀수인 모든 주문을 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.orders.insertMany([
  { _id: 1, item: "Widget", quantity: 15 },
  { _id: 2, item: "Gadget", quantity: 20 },
  { _id: 3, item: "Tool", quantity: 7 },
  { _id: 4, item: "Device", quantity: 12 },
  { _id: 5, item: "Part", quantity: 9 }
]);
```

쿼리 예제

```
db.orders.find({ quantity: { $mod: [2, 1] } });
```

출력

```
{ "_id" : 1, "item" : "Widget", "quantity" : 15 }
{ "_id" : 3, "item" : "Tool", "quantity" : 7 }
{ "_id" : 5, "item" : "Part", "quantity" : 9 }
```

이 쿼리는 수량을 2로 나눈 나머지 수량이 1인 문서를 반환하여 모든 홀수 수량을 효과적으로 선택합니다.

코드 예제

$mod 쿼리 연산자를 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('orders');

  const result = await collection.find({ quantity: { $mod: [2, 1] } }).toArray();

  console.log(JSON.stringify(result, null, 2));
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
    db = client['test']
    collection = db['orders']

    result = list(collection.find({'quantity': {'$mod': [2, 1]}}))

    print(result)
    client.close()

example()
```

$lte

Amazon DocumentDB의 `$lte` 연산자는 지정된 필드의 값이 지정된 값보다 작거나 같은 문서를 일치시키는 데 사용됩니다. 이 연산자는 수치 비교를 기반으로 데이터를 필터링하고 쿼리하는 데 유용합니다.

파라미터

- `field`: 비교할 필드입니다.
- `value`: 비교할 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 연`$lte`산자를 사용하여 `quantity` 필드가 10 이하인 문서를 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.inventory.insertMany([
  { item: "canvas", qty: 100 },
  { item: "paint", qty: 50 },
  { item: "brush", qty: 10 },
  { item: "paper", qty: 5 }
]);
```

쿼리 예제

```
db.inventory.find({ qty: { $lte: 10 } });
```

출력

```
{ "_id" : ObjectId("..."), "item" : "brush", "qty" : 10 },
{ "_id" : ObjectId("..."), "item" : "paper", "qty" : 5 }
```

코드 예제

$lte 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require("mongodb");

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db("test");
  const collection = db.collection("inventory");

  const result = await collection.find({ qty: { $lte: 10 } }).toArray();
  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client["test"]
    collection = db["inventory"]

    result = list(collection.find({ "qty": { "$lte": 10 } }))
    print(result)

    client.close()
```

```
if __name__ == "__main__":
    main()
```

$meta

$meta 연산자는 현재 쿼리 실행과 연결된 메타데이터에 액세스하는 데 사용됩니다. 이 연산자는 주로 텍스트 검색 작업에 사용되며, 메타데이터는 일치하는 문서의 관련성에 대한 정보를 제공할 수 있습니다.

파라미터

- textScore: 문서의 텍스트 검색 점수를 검색합니다. 이 점수는 텍스트 검색 쿼리와 문서의 관련성을 나타냅니다.

예제(MongoDB 쉘)

다음 예제에서는 $meta 연산자를 사용하여 텍스트 검색 쿼리와 일치하는 문서의 텍스트 검색 점수를 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.documents.insertMany([
  { _id: 1, title: "Coffee Basics", content: "Coffee is a popular beverage made from
 roasted coffee beans." },
  { _id: 2, title: "Coffee Culture", content: "Coffee coffee coffee - the ultimate
 guide to coffee brewing and coffee preparation." },
  { _id: 3, title: "Tea vs Coffee", content: "Many people prefer tea over coffee for
 its health benefits." }
]);
```

텍스트 인덱스 생성

```
db.documents.createIndex({ content: "text" });
```

쿼리 예제

```
db.documents.find(
  { $text: { $search: "coffee" } },
  { _id: 0, title: 1, content: 1, score: { $meta: "textScore" } }
```

```
).sort({ score: { $meta: "textScore" } });
```

출력

```
[
  {
    title: 'Coffee Culture',
    content: 'Coffee coffee coffee - the ultimate guide to coffee brewing and coffee
 preparation.',
    score: 0.8897688388824463
  },
  {
    title: 'Coffee Basics',
    content: 'Coffee is a popular beverage made from roasted coffee beans.',
    score: 0.75990891456604
  },
  {
    title: 'Tea vs Coffee',
    content: 'Many people prefer tea over coffee for its health benefits.',
    score: 0.6079270839691162
  }
]
```

코드 예제

$meta 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findWithTextScore() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('documents');

  const result = await collection.find(
    { $text: { $search: "coffee" } },
    { projection: { _id: 0, title: 1, content: 1, score: { $meta: "textScore" } } }
  ).sort({ score: { $meta: "textScore" } }).toArray();
```

```
  console.log(result);
  client.close();
}

findWithTextScore();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['documents']

for doc in collection.find(
    {'$text': {'$search': 'coffee'}},
    {'_id': 0, 'title': 1, 'content': 1, 'score': {'$meta': 'textScore'}}
).sort([('score', {'$meta': 'textScore'})]):
    print(doc)

client.close()
```

$ne

연$ne산자는 필드 값이 지정된 값과 같지 않은 문서를 일치시키는 데 사용됩니다. 쿼리 조건자에서 문서를 필터링하는 데 사용할 수 있는 비교 연산자입니다.

플래너 버전 2.0에에 대한 인덱스 지원이 추가되었습니다$ne.

파라미터

- `field`: 확인할 필드입니다.
- `value`: 확인할 값입니다.

예제(MongoDB 쉘)

이 예제에서는 `status` 필드가와 같지 않은 `users` 모음의 모든 문서를 찾습니다`"active"`.

샘플 문서 생성

```
db.users.insertMany([
  { name: "John", status: "active" },
  { name: "Jane", status: "inactive" },
  { name: "Bob", status: "suspended" },
  { name: "Alice", status: "active" }
]);
```

쿼리 예제

```
db.users.find({ status: { $ne: "active" } });
```

출력

```
[
  {
    _id: ObjectId('...'),
    name: 'Jane',
    status: 'inactive'
  },
  {
    _id: ObjectId('...'),
    name: 'Bob',
    status: 'suspended'
  }
]
```

코드 예제

$ne 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  const result = await users.find({ status: { $ne: 'active' } }).toArray();
```

```
  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
users = db['users']

result = list(users.find({ 'status': { '$ne': 'active' } }))
print(result)

client.close()
```

$nin

`$nin` 연산자는 지정된 배열에 없는 값을 일치시키는 데 사용됩니다. 연`$in`산자의 역으로, 지정된 배열에 있는 값과 일치합니다.

플래너 버전 2.0에에 대한 인덱스 지원이 추가되었습니다`$nin`.

파라미터

- `field`: 확인할 필드입니다.
- `array`: 확인할 값의 배열입니다.

필드 이름의 달러(**$**)

중첩된 객체에서의 $ 접두사 필드 쿼리`$nin`와 관련된 필드 이름의 달러($) 및 점(.) 제한 사항은 섹션을 참조하세요.

예제(MongoDB 쉘)

다음 예제에서는 $nin 연산자를 사용하여 category 필드가 "Fiction" 또는 "Mystery"와 같지 않은 문서를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.books.insertMany([
  { title: "The Great Gatsby", author: "F. Scott Fitzgerald", category: "Fiction" },
  { title: "To Kill a Mockingbird", author: "Harper Lee", category: "Fiction" },
  { title: "The Girl on the Train", author: "Paula Hawkins", category: "Mystery" },
  { title: "The Martian", author: "Andy Weir", category: "Science Fiction" },
  { title: "The Alchemist", author: "Paulo Coelho", category: "Philosophy" }
])
```

쿼리 예제

```
db.books.find({
  category: {
    $nin: ["Fiction", "Mystery"]
  }
})
```

출력

```
[
  {
    _id: ObjectId('...'),
    title: 'The Martian',
    author: 'Andy Weir',
    category: 'Science Fiction'
  },
  {
    _id: ObjectId('...'),
    title: 'The Alchemist',
    author: 'Paulo Coelho',
    category: 'Philosophy'
  }
]
```

코드 예제

$nin 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findBooksNotInCategories(categories) {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const books = await db.collection('books').find({
    category: {
      $nin: categories
    }
  }).toArray();
  console.log(books);
  client.close();
}

findBooksNotInCategories(['Fiction', 'Mystery']);
```

Python

```
from pymongo import MongoClient

def find_books_not_in_categories(categories):
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    books = list(db.books.find({
        'category': {
            '$nin': categories
        }
    }))
    print(books)
    client.close()

find_books_not_in_categories(['Fiction', 'Mystery'])
```

$nor

연$nor산자는 지정된 쿼리 조건이 true가 아닌 문서를 일치시키는 데 사용됩니다. 논리적 "NOR" 작업과 유사하며, 피연산자가 true가 아니면 결과가 true입니다.

파라미터

- `expression1`: 평가할 첫 번째 표현식입니다.
- `expression2`: 평가할 두 번째 표현식입니다.
- `expressionN`: 평가할 추가 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 `qty` 필드가 20 이상이고 `size` 필드가 "XL"이 아닌 문서를 검색하여 `$nor` 연산자의 사용을 보여줍니다.

샘플 문서 생성

```
db.items.insertMany([
  { qty: 10, size: "M" },
  { qty: 15, size: "XL" },
  { qty: 25, size: "L" },
  { qty: 30, size: "XL" }
])
```

쿼리 예제

```
db.items.find({
  $nor: [
    { qty: { $lt: 20 } },
    { size: "XL" }
  ]
})
```

출력

```
[
  { "_id" : ObjectId("..."), "qty" : 25, "size" : "L" }
]
```

코드 예제

$nor 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('items');

  const result = await collection.find({
    $nor: [
      { qty: { $lt: 20 } },
      { size: "XL" }
    ]
  }).toArray();

  console.log(result);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['items']

    result = list(collection.find({
        '$nor': [
            { 'qty': { '$lt': 20 } },
```

```
            { 'size': 'XL' }
        ]
    }))

    print(result)

    client.close()

example()
```

$not

연$not산자는 지정된 표현식의 결과를 무효화하는 데 사용됩니다. 지정된 조건이 일치하지 않는 문서를 선택할 수 있습니다.

플래너 버전 2.0에 $not {eq} 및에 대한 인덱스 지원이 추가되었습니다$not {in}.

파라미터

- expression: 부정할 표현식입니다.

예제(MongoDB 쉘)

다음 예제에서는 $not 연산자를 사용하여 status 필드가 "활성"과 같지 않은 문서를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { name: "John", status: "active" },
  { name: "Jane", status: "inactive" },
  { name: "Bob", status: "pending" },
  { name: "Alice", status: "active" }
]);
```

쿼리 예제

```
db.users.find({ status: { $not: { $eq: "active" } } });
```

출력

```
[
  {
    _id: ObjectId('...'),
    name: 'Jane',
    status: 'inactive'
  },
  {
    _id: ObjectId('...'),
    name: 'Bob',
    status: 'pending'
  }
]
```

코드 예제

$not 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient, Filters } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('users');

  const result = await collection.find({
    status: { $not: { $eq: "active" } }
  }).toArray();

  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient
from bson.son import SON
```

```
client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
db = client['test']
collection = db['users']

result = collection.find({
    "status": {"$not": {"$eq": "active"}}
})

for doc in result:
    print(doc)

client.close()
```

$or

$or 연산자는 두 개 이상의 표현식 배열에서 논리적 OR 작업을 수행하는 데 사용됩니다. 표현식 중 하나 이상과 일치하는 문서를 반환합니다. 이 연산자는 여러 조건 중 하나를 충족하는 문서를 쿼리해야 할 때 유용합니다.

파라미터

- expression1: 평가할 첫 번째 표현식입니다.
- expression2: 평가할 두 번째 표현식입니다.
- ...: 평가할 추가 표현식(선택 사항).

예제(MongoDB 쉘)

다음 예제에서는 연$or산자를 사용하여 make가 모델 "Heavy H1"의 "TruckForYou" 또는 모델 "Bolid 1"의 "SportForYou"인 문서를 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.cars.insertMany([
  { make: "TruckForYou", model: "Heavy H1", year: 2020 },
  { make: "SportForYou", model: "Bolid 1", year: 2021 },
  { make: "TruckForYou", model: "Cargo 5", year: 2019 },
  { make: "SportForYou", model: "Racer 2", year: 2022 }
```

```
]);
```

쿼리 예제

```
db.cars.find({
  $or: [
    { make: "TruckForYou", model: "Heavy H1" },
    { make: "SportForYou", model: "Bolid 1" }
  ]
});
```

출력

```
[
  {
    _id: ObjectId('...'),
    make: 'TruckForYou',
    model: 'Heavy H1',
    year: 2020
  },
  {
    _id: ObjectId('...'),
    make: 'SportForYou',
    model: 'Bolid 1',
    year: 2021
  }
]
```

코드 예제

$or 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findCarsByMakeModel() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const cars = db.collection('cars');
```

```
  const result = await cars.find({
    $or: [
      { make: "TruckForYou", model: "Heavy H1" },
      { make: "SportForYou", model: "Bolid 1" }
    ]
  }).toArray();

  console.log(result);
  client.close();
}

findCarsByMakeModel();
```

Python

```
from pymongo import MongoClient

def find_cars_by_make_model():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    cars = db.cars

    result = list(cars.find({
        '$or': [
            {'make': 'TruckForYou', 'model': 'Heavy H1'},
            {'make': 'SportForYou', 'model': 'Bolid 1'}
        ]
    }))

    print(result)
    client.close()

find_cars_by_make_model()
```

$regex

$regex 연산자를 사용하면 문자열 필드에서 정규식 일치를 수행할 수 있습니다. 복잡한 패턴을 기반으로 문서를 검색하고 필터링하는 강력한 도구입니다.

파라미터

- `regular expression`: 필드와 일치하는 정규식 패턴입니다.
- `$options`: (선택 사항) 대소문자 구분, 전역 일치 등과 같은 검색 동작을 수정하는 옵션을 제공합니다.

예제(MongoDB 쉘)

다음 예제에서는 연`$regex`산자를 사용하여 "name" 필드가 특정 패턴과 일치하는 문서를 검색하는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { name: "John Doe" },
  { name: "Jane Smith" },
  { name: "Alice Johnson" },
  { name: "Bob Williams" },
  { name: "Charlie Davis" }
]);
```

쿼리 예제

```
db.users.find({ name: { $regex: /^A/ } })
```

출력

```
[
  { "_id" : ObjectId("..."), "name" : "Alice Johnson" }
]
```

이 쿼리는 "이름" 필드가 "A" 문자로 시작하는 모든 문서를 반환합니다.

코드 예제

`$regex` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('users');

  const results = await collection.find({ name: { $regex: /^A/ } }).toArray();
  console.log(results);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['users']

    results = list(collection.find({ 'name': { '$regex': '^A' } }))
    print(results)

    client.close()

if __name__ == "__main__":
    main()
```

$sampleRate

버전 8.0.1에서 새로 추가되었습니다.

Elastic 클러스터에서는 지원되지 않습니다.

Amazon DocumentDB의 `$sampleRate` 연산자는 지정된 속도를 기반으로 입력 문서의 무작위 샘플과 일치합니다. `find()` 쿼리 필터 또는 집계 `$match` 단계에서 사용할 수 있습니다. 선택은 확률적이므로 반환되는 문서 수는 근사치이며 실행마다 다를 수 있습니다.

파라미터

- value: 연산자[0, 1](예:)에게 직접 전달되는 범위(포함{ $sampleRate: 0.3 })의 숫자로, 각 문서를 포함할 확률을 설정합니다. 값이 이면 문서가 0 반환되지 않고 값이 이면 모든 문서가 1 반환됩니다.

예제(MongoDB 쉘)

다음 예제에서는 $sampleRate를 사용하여 컬렉션에 있는 문서의 약 30%를 반환합니다.

샘플 문서 생성

```
db.events.insertMany([
  { _id: 1, type: "click" },
  { _id: 2, type: "view" },
  { _id: 3, type: "click" },
  { _id: 4, type: "view" },
  { _id: 5, type: "purchase" }
]);
```

쿼리 예제

```
db.events.aggregate([
  { $match: { $sampleRate: 0.3 } }
]);
```

출력

작업은 문서의 무작위 하위 집합을 반환합니다. $sampleRate는 확률적이므로 쿼리가 실행될 때마다 반환되는 특정 문서와 해당 수가 달라집니다.

코드 예제

$sampleRate 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const uri = 'mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false';
  const client = new MongoClient(uri);

  try {
    await client.connect();
    const db = client.db('test');
    const collection = db.collection('events');

    // Return approximately 30% of the documents
    const result = await collection.aggregate([
      { $match: { $sampleRate: 0.3 } }
    ]).toArray();

    console.log(result);
  } finally {
    await client.close();
  }
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')

    try:
        db = client['test']
        collection = db['events']

        # Return approximately 30% of the documents
        result = list(collection.aggregate([
            { '$match': { '$sampleRate': 0.3 } }
        ]))

        print(result)
    finally:
```

```
        client.close()

example()
```

$slice

$slice 프로젝션 연산자는 쿼리 결과에 반환되는 배열 요소의 수를 제한합니다. 이를 통해 전체 배열을 로드하지 않고도 배열 필드의 시작 또는 끝에서 특정 수의 요소를 검색할 수 있습니다.

파라미터

- field: 프로젝션할 배열 필드입니다.
- count: 반환할 요소 수입니다. 양수 값은 시작에서 요소를 반환하고, 음수 값은 끝에서 반환합니다.

예제(MongoDB 쉘)

다음 예제에서는 $slice 프로젝션 연산자를 사용하여 배열 필드에서 처음 두 항목만 반환하는 방법을 보여줍니다.

샘플 문서 생성

```
db.inventory.insertMany([
  { _id: 1, item: "notebook", tags: ["office", "school", "supplies", "writing"] },
  { _id: 2, item: "pen", tags: ["office", "writing"] },
  { _id: 3, item: "folder", tags: ["office", "supplies", "storage", "organization"] }
]);
```

쿼리 예제

```
db.inventory.find(
  {},
  { item: 1, tags: { $slice: 2 } }
)
```

출력

```
{ "_id" : 1, "item" : "notebook", "tags" : [ "office", "school" ] }
{ "_id" : 2, "item" : "pen", "tags" : [ "office", "writing" ] }
```

```
{ "_id" : 3, "item" : "folder", "tags" : [ "office", "supplies" ] }
```

코드 예제

`$slice` 프로젝션 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('inventory');

  const result = await collection.find(
    {},
    { projection: { item: 1, tags: { $slice: 2 } } }
  ).toArray();

  console.log(JSON.stringify(result, null, 2));
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['inventory']

    result = list(collection.find(
        {},
        {'item': 1, 'tags': {'$slice': 2}}
    ))
```

```
    print(result)
    client.close()

example()
```

$size

$size 쿼리 연산자는 배열 필드의 요소 수가 정확히 지정된 문서와 일치합니다. 이는 배열 길이를 기준으로 문서를 필터링하는 데 유용합니다.

파라미터

- field: 확인할 배열 필드입니다.
- count: 배열에 포함되어야 하는 정확한 요소 수입니다.

예제(MongoDB 쉘)

다음 예제에서는 $size 연산자를 사용하여 정확히 세 개의 태그가 있는 모든 제품을 찾는 방법을 보여줍니다.

샘플 문서 생성

```
db.products.insertMany([
  { _id: 1, name: "Laptop", tags: ["electronics", "computers", "portable"] },
  { _id: 2, name: "Mouse", tags: ["electronics", "accessories"] },
  { _id: 3, name: "Desk", tags: ["furniture", "office", "workspace"] },
  { _id: 4, name: "Monitor", tags: ["electronics"] }
]);
```

쿼리 예제

```
db.products.find({ tags: { $size: 3 } });
```

출력

```
{ "_id" : 1, "name" : "Laptop", "tags" : [ "electronics", "computers", "portable" ] }
{ "_id" : 3, "name" : "Desk", "tags" : [ "furniture", "office", "workspace" ] }
```

코드 예제

$size 쿼리 연산자를 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  const result = await collection.find({ tags: { $size: 3 } }).toArray();

  console.log(JSON.stringify(result, null, 2));
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['products']

    result = list(collection.find({'tags': {'$size': 3}}))

    print(result)
    client.close()

example()
```

$text

$text 연산자는 문서 모음 내의 텍스트 인덱스 필드에 대해 전체 텍스트 검색을 수행하는 데 사용됩니다. 이 연산자를 사용하면 특정 단어 또는 문구가 포함된 문서를 검색할 수 있으며, 다른 쿼리 연산자와 결합하여 추가 기준에 따라 결과를 필터링할 수 있습니다.

파라미터

- $search: 검색할 텍스트 문자열입니다.

예제(MongoDB 쉘)

다음 예제에서는 $text 연산자를 사용하여 "interest"라는 단어가 포함된 문서를 검색하고 "star_rating" 필드를 기반으로 결과를 필터링하는 방법을 보여줍니다.

샘플 문서 생성

```
db.test.insertMany([
  { "_id": 1, "star_rating": 4, "comments": "apple is red" },
  { "_id": 2, "star_rating": 5, "comments": "pie is delicious" },
  { "_id": 3, "star_rating": 3, "comments": "apples, oranges - healthy fruit" },
  { "_id": 4, "star_rating": 2, "comments": "bake the apple pie in the oven" },
  { "_id": 5, "star_rating": 5, "comments": "interesting couch" },
  { "_id": 6, "star_rating": 5, "comments": "interested in couch for sale, year 2022" }
]);
```

텍스트 인덱스 생성

```
db.test.createIndex({ comments: "text" });
```

쿼리 예제

```
db.test.find({$and: [{star_rating: 5}, {$text: {$search: "interest"}}]})
```

출력

```
{ "_id" : 5, "star_rating" : 5, "comments" : "interesting couch" }
{ "_id" : 6, "star_rating" : 5, "comments" : "interested in couch for sale, year
 2022" }
```

위의 명령은 모든 형태의 "interest"와 "star_rating"이 5인 텍스트 인덱스 필드가 있는 문서를 반환합니다.

코드 예제

$text 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function searchDocuments() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('test');

  const result = await collection.find({
    $and: [
      { star_rating: 5 },
      { $text: { $search: 'interest' } }
    ]
  }).toArray();

  console.log(result);
  client.close();
}

searchDocuments();
```

Python

```
from pymongo import MongoClient

def search_documents():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db.test

    result = list(collection.find({
```

```
        '$and': [
            {'star_rating': 5},
            {'$text': {'$search': 'interest'}}
        ]
    }))

    print(result)
    client.close()

search_documents()
```

$type

연$type산자는 문서에서 필드의 데이터 유형을 확인하는 데 사용됩니다. 유형별 작업 또는 검증이 필요할 때 사용할 수 있습니다. 연$type산자는 평가된 표현식의 BSON 유형을 반환합니다. 반환된 유형은 필드 또는 표현식의 유형에 해당하는 문자열입니다.

플래너 버전 2.0에에 대한 인덱스 지원이 추가되었습니다$type.

파라미터

- `expression` : 평가할 표현식입니다.

예제(MongoDB 쉘)

샘플 문서 생성

```
db.documents.insertMany([
  { _id: 1, name: "John", age: 30, email: "john@example.com" },
  { _id: 2, name: "Jane", age: "25", email: 123456 },
  { _id: 3, name: 123, age: true, email: null }
]);
```

쿼리 예제

```
db.documents.find({
  $or: [
    { age: { $type: "number" } },
    { email: { $type: "string" } },
    { name: { $type: "string" } }
```

```
  ]
})
```

출력

```
[
  { "_id": 1, "name": "John", "age": 30, "email": "john@example.com" },
  { "_id": 2, "name": "Jane", "age": "25", "email": 123456 }
]
```

이 쿼리는 age 필드가 "number" 유형이고, email 필드가 "string" 유형이고, name 필드가 "string" 유형인 문서를 반환합니다.

코드 예제

$type 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function findByType() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('documents');

  const results = await collection.find({
    $or: [
      { age: { $type: 'number' } },
      { email: { $type: 'string' } },
      { name: { $type: 'string' } }
    ]
  }).toArray();

  console.log(results);
  client.close();
}

findByType();
```

Python

```
from pymongo import MongoClient

def find_by_type():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['documents']

    results = list(collection.find({
        '$or': [
            {'age': {'$type': 'number'}},
            {'email': {'$type': 'string'}},
            {'name': {'$type': 'string'}}
        ]
    }))

    print(results)
    client.close()

find_by_type()
```

업데이트 연산자

이 섹션에서는 Amazon DocumentDB에서 지원하는 업데이트 연산자에 대한 자세한 정보를 제공합니다.

주제

- $
- $[]
- $[<identifier>]
- $addToSet
- $bit
- $currentDate
- $each
- $inc

- $max
- $min
- $mul
- $pop
- $position
- $pull
- $pullAll
- $push
- $rename
- $set
- $setOnInsert
- $slice
- $sort
- $unset

$

$ 위치 연산자는 쿼리 조건과 일치하는 첫 번째 배열 요소를 업데이트합니다. 일치하는 배열 요소의 위치에 대한 자리 표시자 역할을 합니다.

파라미터

- `field.$`: 첫 번째 일치 요소를 업데이트하기 위한 위치 연산자가 있는 배열 필드입니다.

예제(MongoDB 쉘)

다음 예제에서는 $ 위치 연산자를 사용하여 특정 배열 요소를 업데이트하는 방법을 보여줍니다.

샘플 문서 생성

```
db.inventory.insertMany([
  { _id: 1, item: "Widget", quantities: [10, 20, 30] },
  { _id: 2, item: "Gadget", quantities: [5, 15, 25] }
]);
```

쿼리 예제

```
db.inventory.updateOne(
  { _id: 1, quantities: 20 },
  { $set: { "quantities.$": 22 } }
);
```

출력

```
{
  "_id" : 1,
  "item" : "Widget",
  "quantities" : [ 10, 22, 30 ]
}
```

코드 예제

$ 위치 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function updateDocument() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('inventory');

  await collection.updateOne(
    { _id: 1, quantities: 20 },
    { $set: { "quantities.$": 22 } }
  );

  const updatedDocument = await collection.findOne({ _id: 1 });
  console.log(updatedDocument);

  await client.close();
}

updateDocument();
```

Python

```
from pymongo import MongoClient

def update_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.inventory

    collection.update_one(
        {'_id': 1, 'quantities': 20},
        {'$set': {'quantities.$': 22}}
    )

    updated_document = collection.find_one({'_id': 1})
    print(updated_document)

    client.close()

update_document()
```

$[]

$[] 모든 위치 연산자는 배열의 모든 요소를 업데이트합니다. 배열 필드의 모든 요소를 수정해야 할 때 사용됩니다.

파라미터

- field.$[]: 모든 요소를 업데이트하기 위한 모든 위치 연산자가 있는 배열 필드입니다.

예제(MongoDB 쉘)

다음 예제에서는 $[] 연산자를 사용하여 모든 배열 요소를 업데이트하는 방법을 보여줍니다.

샘플 문서 생성

```
db.products.insertOne({
  _id: 1,
  name: "Laptop",
```

```
  prices: [1000, 1100, 1200]
});
```

쿼리 예제

```
db.products.updateOne(
  { _id: 1 },
  { $inc: { "prices.$[]": 50 } }
);
```

출력

```
{
  "_id" : 1,
  "name" : "Laptop",
  "prices" : [ 1050, 1150, 1250 ]
}
```

코드 예제

$[] 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function updateDocument() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  await collection.updateOne(
    { _id: 1 },
    { $inc: { "prices.$[]": 50 } }
  );

  const updatedDocument = await collection.findOne({ _id: 1 });
  console.log(updatedDocument);

  await client.close();
```

```
}

updateDocument();
```

Python

```
from pymongo import MongoClient

def update_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.products

    collection.update_one(
        {'_id': 1},
        {'$inc': {'prices.$[]': 50}}
    )

    updated_document = collection.find_one({'_id': 1})
    print(updated_document)

    client.close()

update_document()
```

$[<identifier>]

$[<identifier>] 필터링된 위치 연산자는 지정된 필터 조건과 일치하는 모든 배열 요소를 업데이트합니다. 배열 요소를 선택적으로 업데이트하는 arrayFilters 옵션과 함께 사용됩니다.

파라미터

- field.$[identifier]: 필터링된 위치 연산자가 있는 배열 필드입니다.
- arrayFilters: 업데이트할 요소를 결정하는 필터 조건 배열입니다.

예제(MongoDB 쉘)

다음 예제에서는 $[<identifier>] 연산자를 사용하여 조건을 기반으로 특정 배열 요소를 업데이트하는 방법을 보여줍니다.

샘플 문서 생성

```
db.students.insertOne({
  _id: 1,
  name: "Alice",
  grades: [
    { subject: "Math", score: 85 },
    { subject: "Science", score: 92 },
    { subject: "History", score: 78 }
  ]
});
```

쿼리 예제

```
db.students.updateOne(
  { _id: 1 },
  { $inc: { "grades.$[elem].score": 5 } },
  { arrayFilters: [{ "elem.score": { $gte: 80 } }] }
);
```

출력

```
{
  "_id" : 1,
  "name" : "Alice",
  "grades" : [
    { "subject" : "Math", "score" : 90 },
    { "subject" : "Science", "score" : 97 },
    { "subject" : "History", "score" : 78 }
  ]
}
```

코드 예제

$[<identifier>] 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function updateDocument() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('students');

  await collection.updateOne(
    { _id: 1 },
    { $inc: { "grades.$[elem].score": 5 } },
    { arrayFilters: [{ "elem.score": { $gte: 80 } }] }
  );

  const updatedDocument = await collection.findOne({ _id: 1 });
  console.log(updatedDocument);

  await client.close();
}

updateDocument();
```

Python

```
from pymongo import MongoClient

def update_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.students

    collection.update_one(
        {'_id': 1},
        {'$inc': {'grades.$[elem].score': 5}},
        array_filters=[{'elem.score': {'$gte': 80}}]
    )

    updated_document = collection.find_one({'_id': 1})
    print(updated_document)

    client.close()

update_document()
```

$addToSet

Amazon DocumentDB의 $addToSet 연산자는 값이 배열에 아직 없는 경우에만 배열에 값을 추가하는 데 사용됩니다. 이는 배열에 고유한 요소가 포함되어 있는지 확인하는 데 유용합니다.

파라미터

- field: 업데이트할 필드입니다.
- value: 배열 필드에 추가할 값입니다. 단일 값 또는 표현식일 수 있습니다.

예제(MongoDB 쉘)

다음 예제에서는 $addToSet 연산자를 사용하여 배열에 고유한 요소를 추가하는 방법을 보여줍니다.

샘플 문서 생성

```
db.products.insertMany([
  { "_id": 1, "item": "apple", "tags": ["fruit", "red", "round"] },
  { "_id": 2, "item": "banana", "tags": ["fruit", "yellow"] },
  { "_id": 3, "item": "cherry", "tags": ["fruit", "red"] }
])
```

쿼리 예제

```
db.products.update(
  { "item": "apple" },
  { $addToSet: { "tags": "green" } }
)
```

출력

```
{ "_id": 1, "item": "apple", "tags": ["fruit", "red", "round", "green"] }
```

이 예제에서 $addToSet 연산자는 문서의 "tags" 배열에 "green" 태그를 추가합니다. 여기서 "item" 필드는 "apple"입니다. "그린"이 아직 배열에 없기 때문에 추가되었습니다.

코드 예제

$addToSet 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('products');

  await collection.updateOne(
    { "item": "apple" },
    { $addToSet: { "tags": "green" } }
  );

  const updatedDoc = await collection.findOne({ "item": "apple" });
  console.log(updatedDoc);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.products

    collection.update_one(
        {"item": "apple"},
        {"$addToSet": {"tags": "green"}}
    )

    updated_doc = collection.find_one({"item": "apple"})
    print(updated_doc)
```

```
    client.close()

example()
```

$bit

Amazon DocumentDB의 $bit 연산자를 사용하면 지정된 필드의 비트에 대해 비트 단위 작업을 수행할 수 있습니다. 이는 숫자 내에서 개별 비트의 상태 설정, 지우기 또는 확인과 같은 작업에 유용할 수 있습니다.

파라미터

- field: 비트 단위 작업을 수행할 필드입니다.
- and: 필드에 대해 비트 단위 AND 작업을 수행하는 데 사용되는 정수 값입니다.
- or: 필드에서 비트 단위 OR 작업을 수행하는 데 사용되는 정수 값입니다.
- xor: 필드에서 비트 단위 XOR 작업을 수행하는 데 사용되는 정수 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 $bit 연산자를 사용하여 숫자 필드에 대해 비트 단위 작업을 수행하는 방법을 보여줍니다.

샘플 문서 생성

```
db.numbers.insert([
  { "_id": 1, "number": 5 },
  { "_id": 2, "number": 12 }
])
```

쿼리 예제

```
db.numbers.update(
  { "_id": 1 },
  { "$bit": { "number": { "and": 3 } } }
)
```

출력

```
{
```

```
  "_id": 1,
  "number": 1
}
```

이 예제에서는 연$bit산자를 사용하여가 _id 1인 문서의 "숫자" 필드에 대해 비트 단위 AND 작업을 수행합니다. 그 결과 "숫자" 필드의 값이 1로 설정되는데, 이는 원래 값 5와 값 3 사이의 비트 단위 AND 작업의 결과입니다.

코드 예제

$bit 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('numbers');

  await collection.updateOne(
    { "_id": 1 },
    { "$bit": { "number": { "and": 3 } } }
  );

  const result = await collection.findOne({ "_id": 1 });
  console.log(result);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
db = client['test']
collection = db['numbers']

collection.update_one(
    {"_id": 1},
    {"$bit": {"number": {"and": 3}}}
)

result = collection.find_one({"_id": 1})
print(result)

client.close()
```

$currentDate

$currentDate 연산자는 필드 값을 현재 날짜 및 시간으로 설정하는 데 사용됩니다. 이 연산자는 문서가 삽입되거나 업데이트될 때 현재 타임스탬프로 필드를 자동으로 업데이트하는 데 유용합니다.

파라미터

- field: 현재 날짜 및 시간으로 업데이트할 필드입니다.
- type: (선택 사항) 현재 날짜에 사용할 BSON 유형을 지정합니다. date 또는 timestamp입니다.

예제(MongoDB 쉘)

다음 예제에서는 $currentDate 연산자를 사용하여 필드를 새 문서가 삽입된 lastModified 현재 날짜 및 시간으로 설정하는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insert({
  name: "John Doe",
  email: "john.doe@example.com"
})
```

쿼리 예제

```
db.users.updateOne(
  { name: "John Doe" },
```

```
  { $currentDate: { lastModified: true } }
)
```

업데이트된 문서 보기

```
db.users.findOne({ name: "John Doe" })
```

출력

```
{
  _id: ObjectId('...'),
  name: 'John Doe',
  email: 'john.doe@example.com',
  lastModified: ISODate('2025-10-25T22:50:29.963Z')
}
```

코드 예제

$currentDate 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function updateUserWithCurrentDate() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');

  await users.updateOne(
    { name: 'John Doe' },
    { $currentDate: { lastModified: true } }
  );

  console.log('User updated with current date');
  client.close();
}

updateUserWithCurrentDate();
```

Python

```
from pymongo import MongoClient

def update_user_with_current_date():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    users = db.users

    result = users.update_one(
        {'name': 'John Doe'},
        {'$currentDate': {'lastModified': True}}
    )

    print('User updated with current date')
    client.close()

update_user_with_current_date()
```

$each

연$each산자는 $push 및와 같은 다른 업데이트 연산자와 함께 사용하여 배열 필드에 여러 값을 $addToSet추가합니다. 여러 업데이트 작업을 실행할 필요 없이 단일 작업으로 배열에 여러 요소를 추가할 수 있습니다.

파라미터

- value: 배열 필드에 추가할 값의 배열입니다.

예제(MongoDB 쉘)

다음 예제에서는 $each 연산자와 함께 $push 연산자를 사용하여 배열 필드에 여러 요소를 추가하는 방법을 보여줍니다.

샘플 문서 생성

```
db.fruits.insertOne({
  _id: 1,
```

```
  fruits: ["apple", "banana"]
})
```

쿼리 예제

```
db.fruits.updateOne(
  { _id: 1 },
  { $push: { fruits: { $each: ["cherry", "durian", "elderberry"] } } }
)
```

업데이트된 문서 보기

```
db.fruits.findOne({ _id: 1 })
```

출력

```
{
  _id: 1,
  fruits: [ 'apple', 'banana', 'cherry', 'durian', 'elderberry' ]
}
```

코드 예제

$each 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('fruits');

  await collection.updateOne(
    { _id: 1 },
    { $push: { fruits: { $each: ["cherry", "durian", "elderberry"] } } }
  );
```

```
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['fruits']

    collection.update_one(
        {'_id': 1},
        {'$push': {'fruits': {'$each': ['cherry', 'durian', 'elderberry']}}}
    )

    client.close()

example()
```

$inc

$inc 연산자는 필드 값을 지정된 양만큼 늘리는 데 사용됩니다. 현재 값을 검색하고 새 값을 계산한 다음 필드를 업데이트할 필요 없이 카운터 또는 등급과 같은 숫자 필드를 업데이트하는 데 사용됩니다.

파라미터

- field: 늘릴 필드의 이름입니다.
- amount: 필드를 늘릴 단위입니다. 양수 또는 음수 값일 수 있습니다.

예제(MongoDB 쉘)

다음 예제에서는 $inc 연산자를 사용하여 문서의 age 필드를 늘리는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertOne({_id: 123, name: "John Doe", age: 30})
```

쿼리 예제

```
db.users.updateOne({_id: 123}, {$inc: {age: 1}})
```

업데이트된 문서 보기

```
db.users.findOne({_id: 123})
```

출력

```
{ "_id" : 123, "name" : "John Doe", "age" : 31 }
```

코드 예제

$inc 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function updateWithInc() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('users');

  const result = await collection.updateOne(
    { _id: 123 },
    { $inc: { age: 1 } }
  );

  console.log(result);

  await client.close();
}

updateWithInc();
```

Python

```
from pymongo import MongoClient

def update_with_inc():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['users']

    result = collection.update_one(
        {'_id': 123},
        {'$inc': {'age': 1}}
    )

    print(result.modified_count)

    client.close()

update_with_inc()
```

$max

$max 업데이트 연산자는 지정된 값이 현재 필드 값보다 큰 경우에만 필드 값을 업데이트합니다. 이 연산자는 업데이트 전반에서 최대값을 유지하는 데 유용합니다.

파라미터

- field: 업데이트할 필드입니다.
- value: 현재 필드 값과 비교할 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 $max 연산자를 사용하여 플레이어의 기록된 최고 점수를 업데이트하는 방법을 보여 줍니다.

샘플 문서 생성

```
db.scores.insertMany([
  { _id: 1, player: "Alice", highScore: 85 },
```

```
  { _id: 2, player: "Bob", highScore: 92 },
  { _id: 3, player: "Charlie", highScore: 78 }
])
```

업데이트 예제

```
db.scores.updateOne(
  { _id: 1 },
  { $max: { highScore: 95 } }
)
```

결과

Alice의 highScore 필드는 95가 현재 값인 85보다 크므로 95로 업데이트됩니다.

```
{ "_id": 1, "player": "Alice", "highScore": 95 }
```

코드 예제

$max 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('scores');

  const result = await collection.updateOne(
    { _id: 1 },
    { $max: { highScore: 95 } }
  );

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['scores']

    result = collection.update_one(
        { '_id': 1 },
        { '$max': { 'highScore': 95 } }
    )

    print(result)
    client.close()

example()
```

$min

$min 업데이트 연산자는 지정된 값이 현재 필드 값보다 작은 경우에만 필드 값을 업데이트합니다. 이 연산자는 업데이트 전반에 걸쳐 최소값을 유지하는 데 유용합니다.

파라미터

- field: 업데이트할 필드입니다.
- value: 현재 필드 값과 비교할 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 $min 연산자를 사용하여 기상 관측소의 기록된 최저 온도를 업데이트하는 방법을 보여줍니다.

샘플 문서 생성

```
db.weather.insertMany([
  { _id: 1, station: "Station A", lowestTemp: 15 },
```

```
  { _id: 2, station: "Station B", lowestTemp: 20 },
  { _id: 3, station: "Station C", lowestTemp: 18 }
])
```

업데이트 예제

```
db.weather.updateOne(
  { _id: 1 },
  { $min: { lowestTemp: 12 } }
)
```

결과

12가 현재 값인 15보다 작기 때문에 스테이션 A의 `lowestTemp` 필드가 12로 업데이트됩니다.

```
{ "_id": 1, "station": "Station A", "lowestTemp": 12 }
```

코드 예제

`$min` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('weather');

  const result = await collection.updateOne(
    { _id: 1 },
    { $min: { lowestTemp: 12 } }
  );

  console.log(result);
  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['weather']

    result = collection.update_one(
        { '_id': 1 },
        { '$min': { 'lowestTemp': 12 } }
    )

    print(result)
    client.close()

example()
```

$mul

Amazon DocumentDB의 `$mul` 연산자는 필드 값에 지정된 숫자를 곱하는 데 사용됩니다. 이는 신용 카드 상태에 따라 비행 거리를 업데이트하는 등 여러 문서를 원자적이고 일관되게 업데이트하는 데 유용할 수 있습니다.

파라미터

- `field`: 곱할 필드입니다.
- `multiplier`: 필드 값에 곱할 숫자입니다.

예제(MongoDB 쉘)

이 예제에서는 `$mul` 연산자를 사용하여 `credit_card` 필드가 인 모든 문서의 `flight_miles` 값을 두 배로 늘리는 방법을 보여줍니다`true`.

샘플 문서 생성

```
db.miles.insertMany([
```

```
  { "_id": 1, "member_since": new Date("1987-01-01"), "credit_card": false,
 "flight_miles": [1205, 2560, 880] },
  { "_id": 2, "member_since": new Date("1982-01-01"), "credit_card": true,
 "flight_miles": [2410, 5120, 1780, 5560] },
  { "_id": 3, "member_since": new Date("1999-01-01"), "credit_card": true,
 "flight_miles": [2410, 1760] }
]);
```

쿼리 예제

```
db.miles.update(
  { "credit_card": { "$eq": true } },
  { "$mul": { "flight_miles.$[]": NumberInt(2) } },
  { "multi": true }
);
```

출력

```
{ "_id" : 1, "member_since" : ISODate("1987-01-01T00:00:00Z"), "credit_card" : false,
 "flight_miles" : [ 1205, 2560, 880 ] }
{ "_id" : 2, "member_since" : ISODate("1982-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 4820, 10240, 3560, 11120 ] }
{ "_id" : 3, "member_since" : ISODate("1999-01-01T00:00:00Z"), "credit_card" : true,
 "flight_miles" : [ 4820, 3520 ] }
```

신용 카드가 있는 고객의 경우 비행 마일이 두 배로 늘어났습니다.

`$[]` 위치 배열 연산자는 `flight_miles` 배열의 각 요소에 `$mul` 작업을 적용하는 데 사용됩니다.

코드 예제

`$mul` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function updateFlightMiles() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('miles');
```

```
  await collection.updateMany(
    { credit_card: true },
    { $mul: { 'flight_miles.$[]': 2 } }
  );

  const documents = await collection.find().toArray();
  console.log(documents);

  await client.close();
}

updateFlightMiles();
```

Python

```
from pymongo import MongoClient

def update_flight_miles():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.miles

    collection.update_many(
        {'credit_card': True},
        {'$mul': {'flight_miles.$[]': 2}}
    )

    documents = list(collection.find())
    print(documents)

    client.close()

update_flight_miles()
```

$pop

Amazon DocumentDB의 $pop 연산자는 배열 필드에서 첫 번째 또는 마지막 요소를 제거하는 데 사용됩니다. 고정 크기 배열을 유지하거나 문서 내에서 대기열과 같은 데이터 구조를 구현해야 할 때 특히 유용합니다.

파라미터

- `field`: 요소를 제거할 배열 필드의 이름입니다.
- `value`: 제거할 요소의 위치를 결정하는 정수 값입니다. 값이 이면 마지막 요소가 1 제거되고 값이 이면 첫 번째 요소가 -1 제거됩니다.

예제(MongoDB 쉘)

이 예제에서는 $pop 연산자를 사용하여 배열 필드에서 첫 번째 및 마지막 요소를 제거하는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insertMany([
  { "_id": 1, "name": "John Doe", "hobbies": ["reading", "swimming", "hiking"] },
  { "_id": 2, "name": "Jane Smith", "hobbies": ["cooking", "gardening", "painting"] }
])
```

쿼리 예제

```
// Remove the first element from the "hobbies" array
db.users.update({ "_id": 1 }, { $pop: { "hobbies": -1 } })

// Remove the last element from the "hobbies" array
db.users.update({ "_id": 2 }, { $pop: { "hobbies": 1 } })
```

출력

```
{ "_id" : 1, "name" : "John Doe", "hobbies" : [ "swimming", "hiking" ] }
{ "_id" : 2, "name" : "Jane Smith", "hobbies" : [ "cooking", "gardening" ] }
```

코드 예제

$pop 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('users');

  // Remove the first element from the "hobbies" array
  await collection.updateOne({ "_id": 1 }, { $pop: { "hobbies": -1 } });

  // Remove the last element from the "hobbies" array
  await collection.updateOne({ "_id": 2 }, { $pop: { "hobbies": 1 } });

  const users = await collection.find().toArray();
  console.log(users);

  await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['users']

    # Remove the first element from the "hobbies" array
    collection.update_one({"_id": 1}, {"$pop": {"hobbies": -1}})

    # Remove the last element from the "hobbies" array
    collection.update_one({"_id": 2}, {"$pop": {"hobbies": 1}})

    users = list(collection.find())
    print(users)

    client.close()

example()
```

$position

Amazon DocumentDB의 $position 한정자는 $push 연산자가 요소를 삽입하는 배열의 위치를 지정합니다. $position 한정자가 없으면 $push 연산자가 배열 끝에 요소를 삽입합니다.

파라미터

- field: 업데이트할 배열 필드입니다.
- num: 제로 기반 인덱싱을 기반으로 요소를 삽입해야 하는 배열의 위치입니다.

참고: $position 한정자를 사용하려면 $each 한정자와 함께 표시되어야 합니다.

예제(MongoDB 쉘)

다음 예제에서는 $position 연산자를 사용하여 프로젝트 관리 시스템의 특정 위치에 작업을 삽입하는 방법을 보여줍니다.

샘플 문서 생성

```
db.projects.insertOne({ "_id": 1, "name": "Website Redesign", "tasks": ["Design
 mockups"] })
```

쿼리 예제 1 - 시작 부분에 긴급 작업 추가

```
db.projects.updateOne(
   { _id: 1 },
   {
     $push: {
        tasks: {
           $each: ["Security audit", "Performance review"],
           $position: 0
        }
     }
   }
)
```

출력 1

```
{ "_id": 1, "name": "Website Redesign", "tasks": ["Security audit", "Performance
 review", "Design mockups"] }
```

쿼리 예제 2 - 특정 위치에 작업 추가

```
db.projects.insertOne({ "_id": 2, "name": "Mobile App", "tasks": ["Setup project",
 "Create wireframes", "Deploy to store"] })

db.projects.updateOne(
   { _id: 2 },
   {
     $push: {
        tasks: {
           $each: ["Code review", "Testing phase"],
           $position: 2
        }
     }
   }
)
```

출력 2

```
{ "_id": 2, "name": "Mobile App", "tasks": ["Setup project", "Create wireframes", "Code
 review", "Testing phase", "Deploy to store"] }
```

코드 예제

$position 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function insertTasksAtPosition() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('projects');

  await collection.updateOne(
    { _id: 1 },
    {
      $push: {
        tasks: {
          $each: ["Security audit", "Performance review"],
```

```
          $position: 0
        }
      }
    }
  );

  const updatedProject = await collection.findOne({ _id: 1 });
  console.log(updatedProject);

  await client.close();
}

insertTasksAtPosition();
```

Python

```
from pymongo import MongoClient

def insert_tasks_at_position():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['projects']

    result = collection.update_one(
        {'_id': 1},
        {
            '$push': {
                'tasks': {
                    '$each': ['Security audit', 'Performance review'],
                    '$position': 0
                }
            }
        }
    )

    updated_project = collection.find_one({'_id': 1})
    print(updated_project)

    client.close()

insert_tasks_at_position()
```

$pull

$pull 연산자는 지정된 조건과 일치하는 값 또는 값의 모든 인스턴스를 배열에서 제거하는 데 사용됩니다. 이 연산자는 문서 내의 배열 필드에서 특정 요소를 제거해야 할 때 유용합니다.

파라미터

- field: 값을 제거할 배열 필드의 이름입니다().
- value: 배열에서 제거할 요소(들)를 결정하는 값 또는 조건입니다.

예제(MongoDB 쉘)

다음 예제에서는 $pull 연산자를 사용하여 배열 필드에서 요소를 제거하는 방법을 보여줍니다.

샘플 문서 생성

```
db.restaurants.insertMany([
  {
    name: "Pizza Hut",
    cuisine: "Italian",
    features: ["Delivery", "Takeout", "Dine-in"]
  },
  {
    name: "Sushi Saito",
    cuisine: "Japanese",
    features: ["Dine-in", "Private Dining"]
  },
  {
    name: "Taco Bell",
    cuisine: "Mexican",
    features: ["Delivery", "Takeout", "Drive-thru"]
  }
])
```

쿼리 예제

```
db.restaurants.updateMany(
  { cuisine: "Italian" },
  { $pull: { features: "Takeout" } }
)
```

출력

```
{
  "acknowledged" : true,
  "matchedCount" : 1,
  "modifiedCount" : 1
}
```

위의 쿼리는 cuisine 필드가 "이탈리아어"인 모든 문서에서 "Takeout" 기능을 제거합니다.

코드 예제

$pull 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function updateRestaurants() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const restaurants = db.collection('restaurants');

  await restaurants.updateMany(
    { cuisine: 'Italian' },
    { $pull: { features: 'Takeout' } }
  );

  const updatedRestaurants = await restaurants.find({}).toArray();
  console.log(updatedRestaurants);

  await client.close();
}

updateRestaurants();
```

Python

```
from pymongo import MongoClient
```

```
def update_restaurants():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    restaurants = db.restaurants

    result = restaurants.update_many(
        { 'cuisine': 'Italian' },
        { '$pull': { 'features': 'Takeout' } }
    )

    updated_restaurants = list(restaurants.find({}))
    print(updated_restaurants)

    client.close()

update_restaurants()
```

$pullAll

Amazon DocumentDB의 `$pullAll` 연산자는 배열 필드에서 지정된 값의 모든 인스턴스를 제거하는 데 사용됩니다. 이는 단일 작업으로 배열에서 여러 요소를 제거해야 하는 경우에 특히 유용합니다.

파라미터

- `field`: 요소를 제거할 배열 필드의 이름입니다.
- `value`: 배열 필드에서 제거할 값의 배열입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$pullAll` 연산자를 사용하여 배열 필드에서 여러 요소를 제거하는 방법을 보여줍니다.

샘플 문서 생성

```
db.restaurants.insert([
  {
    "name": "Taj Mahal",
    "cuisine": "Indian",
    "features": ["Private Dining", "Live Music"]
```

```
  },
  {
    "name": "Golden Palace",
    "cuisine": "Chinese",
    "features": ["Private Dining", "Takeout"]
  },
  {
    "name": "Olive Garden",
    "cuisine": "Italian",
    "features": ["Private Dining", "Outdoor Seating"]
  }
])
```

쿼리 예제

```
db.restaurants.update(
  { "name": "Taj Mahal" },
  { $pullAll: { "features": ["Private Dining", "Live Music"] } }
)
```

출력

```
{
  "name": "Taj Mahal",
  "cuisine": "Indian",
  "features": []
}
```

코드 예제

`$pullAll` 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function main() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('restaurants');
```

```
  await collection.updateMany(
    { "name": "Taj Mahal" },
    { $pullAll: { "features": ["Private Dining", "Live Music"] } }
  );

  const updatedDocument = await collection.findOne({ "name": "Taj Mahal" });
  console.log(updatedDocument);

  await client.close();
}

main();
```

Python

```
from pymongo import MongoClient

def main():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['restaurants']

    collection.update_many(
        {"name": "Taj Mahal"},
        {"$pullAll": {"features": ["Private Dining", "Live Music"]}}
    )

    updated_document = collection.find_one({"name": "Taj Mahal"})
    print(updated_document)

    client.close()

if __name__ == '__main__':
    main()
```

$push

Amazon DocumentDB의 $push 연산자는 문서의 배열 필드에 항목을 추가하는 데 사용됩니다. 이 연산자는 전체 배열을 덮어쓰지 않고 기존 배열에 새 데이터를 추가해야 하는 경우에 특히 유용합니다.

파라미터

- `field`: 새 요소를 추가해야 하는 배열 필드의 이름입니다.
- `value`: 배열에 추가할 값입니다.
- `position`: (선택 사항) 새 요소를 추가해야 하는 배열의 위치를 지정하는 한정자입니다. 지원되는 한정자는 $ (배열 끝에 추가) 및 $[] (배열 끝에 추가, 배열 필터 무시)입니다.

예제(MongoDB 쉘)

다음 예제에서는 $push 연산자를 사용하여 문서의 배열 필드에 새 요소를 추가하는 방법을 보여줍니다.

샘플 문서 생성

```
db.users.insert([
  { _id: 1, name: "John Doe", hobbies: ["reading", "swimming"] },
  { _id: 2, name: "Jane Smith", hobbies: ["gardening", "cooking"] }
])
```

쿼리 예제

```
db.users.updateOne(
  { _id: 1 },
  { $push: { hobbies: "hiking" } }
)
```

출력

```
{
  "acknowledged" : true,
  "matchedCount" : 1,
  "modifiedCount" : 1
}
```

업데이트를 실행한 후의 문서에 hobbies 배열_id: 1이 로 업데이트됩니다["reading", "swimming", "hiking"].

코드 예제

$push 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function updateDocument() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('users');

  const result = await collection.updateOne(
    { _id: 1 },
    { $push: { hobbies: "hiking" } }
  );

  console.log(result);
  client.close();
}

updateDocument();
```

Python

```
from pymongo import MongoClient

def update_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['users']

    result = collection.update_one(
        {'_id': 1},
        {'$push': {'hobbies': 'hiking'}}
    )

    print(result.raw_result)
    client.close()

update_document()
```

$rename

Amazon DocumentDB의 $rename 연산자는 문서의 필드 이름을 바꾸는 데 사용됩니다. 이 연산자는 문서의 구조를 업데이트하거나 새 데이터 모델에 맞게 조정해야 할 때 특히 유용할 수 있습니다.

파라미터

- field: 이름을 바꿀 필드입니다.
- newName: 필드의 새 이름입니다.

예제(MongoDB 쉘)

다음 예제에서는 $rename 연산자를 사용하여 "Date.DoW" 필드가 로 설정된 문서에서 "DocName" 필드의 이름을 로 바꾸"Date.DayOfWeek"는 방법을 보여줍니다"Document 1".

샘플 문서 생성

```
db.example.insertOne({
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DoW": "Saturday"
    },
    "Words": 2482
})
```

쿼리 예제

```
db.example.update(
    { "DocName": "Document 1" },
    { $rename: { "Date.DoW": "Date.DayOfWeek" } }
)
```

출력

```
{
```

```
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DayOfWeek": "Saturday"
    },
    "Words": 2482
}
```

코드 예제

$rename 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function example() {
    const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
    const db = client.db('test');
    const collection = db.collection('example');

    await collection.updateOne(
        { "DocName": "Document 1" },
        { $rename: { "Date.DoW": "Date.DayOfWeek" } }
    );

    const updatedDoc = await collection.findOne({ "DocName": "Document 1" });
    console.log(updatedDoc);

    await client.close();
}

example();
```

Python

```
from pymongo import MongoClient

def example():
```

```
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['example']

    collection.update_one(
        {"DocName": "Document 1"},
        {"$rename": {"Date.DoW": "Date.DayOfWeek"}}
    )

    updated_doc = collection.find_one({"DocName": "Document 1"})
    print(updated_doc)

    client.close()

example()
```

$set

Amazon DocumentDB의 `$set` 연산자는 문서에서 지정된 필드의 값을 업데이트하는 데 사용됩니다. 이 연산자를 사용하면 문서 내에서 새 필드를 추가하거나 기존 필드를 수정할 수 있습니다. Amazon DocumentDB와 호환되는 MongoDB Java 드라이버의 기본 업데이트 연산자입니다.

파라미터

- `field`: 업데이트할 필드입니다.
- `value`: 필드의 새 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 `$set` 연산자를 사용하여 문서의 `Item` 필드를 업데이트하는 방법을 보여줍니다.

샘플 문서 생성

```
db.example.insert([
  {
    "Item": "Pen",
    "Colors": ["Red", "Green", "Blue", "Black"],
    "Inventory": {
      "OnHand": 244,
```

```
      "MinOnHand": 72
    }
  },
  {
    "Item": "Poster Paint",
    "Colors": ["Red", "Green", "Blue", "White"],
    "Inventory": {
      "OnHand": 120,
      "MinOnHand": 36
    }
  }
])
```

쿼리 예제

```
db.example.update(
  { "Item": "Pen" },
  { $set: { "Item": "Gel Pen" } }
)
```

출력

```
{
  "Item": "Gel Pen",
  "Colors": ["Red", "Green", "Blue", "Black"],
  "Inventory": {
    "OnHand": 244,
    "MinOnHand": 72
  }
}
```

코드 예제

$set 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function updateDocument() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
```

```
  const db = client.db('test');
  const collection = db.collection('example');

  await collection.updateOne(
    { "Item": "Pen" },
    { $set: { "Item": "Gel Pen" } }
  );

  const updatedDocument = await collection.findOne({ "Item": "Gel Pen" });
  console.log(updatedDocument);

  await client.close();
}

updateDocument();
```

Python

```
from pymongo import MongoClient

def update_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.example

    collection.update_one(
        {"Item": "Pen"},
        {"$set": {"Item": "Gel Pen"}}
    )

    updated_document = collection.find_one({"Item": "Gel Pen"})
    print(updated_document)

    client.close()

update_document()
```

$setOnInsert

Amazon DocumentDB의 `$setOnInsert` 연산자는 문서가 삽입되는 경우 필드의 값을 설정하는 데 사용되지만 문서가 업데이트되는 경우에는 영향을 주지 않습니다.

파라미터

- `field`: 설정할 필드입니다.
- `value`: 필드에 할당할 값입니다.

예제(MongoDB 쉘)

다음 예제에서는 Amazon DocumentDB에서 `$setOnInsert` 연산자의 사용을 보여줍니다. 문서가 아직 없는 경우 새 문서를 생성하지만 문서가 업데이트되는 경우에는 영향을 미치지 않습니다.

샘플 문서 생성

```
db.users.insertOne({
  _id: 1,
  name: "John Doe",
  age: 30
})
```

쿼리 예제 1 - 기존 문서 업데이트

```
db.users.update(
  { _id: 1 },
  {
    $set: { age: 31 },
    $setOnInsert: { createdAt: new Date() }
  },
  { upsert: true }
)
```

출력 1

```
{
  _id: 1,
  name: "John Doe",
  age: 31
```

```
}
```

출력은 문서가 업데이트되었지만 문서가 이미 존재했기 때문에 createdAt 필드가 추가되지 않았음을 보여줍니다. 연$setOnInsert산자는 새 문서를 삽입할 때만 적용됩니다.

쿼리 예제 2 - 새 문서 삽입(업서트)

```
db.users.update(
  { _id: 2 },
  {
    $set: { name: "Jane Smith", age: 25 },
    $setOnInsert: { createdAt: new Date() }
  },
  { upsert: true }
)
```

출력 2

```
{
  _id: 2,
  name: "Jane Smith",
  age: 25,
  createdAt: ISODate("2025-10-31T09:57:52.459Z")
}
}
```

코드 예제

$setOnInsert 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function updateWithSetOnInsert() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const users = db.collection('users');
```

```
  await users.updateOne(
    { _id: 1 },
    {
      $set: { age: 31 },
      $setOnInsert: { createdAt: new Date() }
    },
    { upsert: true }
  );

  const updatedUser = await users.findOne({ _id: 1 });
  console.log(updatedUser);

  await client.close();
}

updateWithSetOnInsert();
```

Python

```
from pymongo import MongoClient

def update_with_set_on_insert():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    users = db['users']

    result = users.update_one(
        {'_id': 1},
        {
            '$set': {'age': 31},
            '$setOnInsert': {'createdAt': datetime.datetime.now()}
        },
        upsert=True
    )

    updated_user = users.find_one({'_id': 1})
    print(updated_user)

    client.close()

update_with_set_on_insert()
```

$slice

$slice 업데이트 연산자는 크기를 제한하여 배열을 수정합니다. $push 연산자와 함께 사용하면 배열의 요소 수를 제한하여 지정된 수의 가장 최근 또는 가장 오래된 요소만 유지합니다.

파라미터

- field: 수정할 배열 필드입니다.
- count: 유지할 최대 요소 수입니다. 양수 값은 첫 번째 N 요소를 유지하고 음수 값은 마지막 N 요소를 유지합니다.

예제(MongoDB 쉘)

다음 예제에서는에서 $slice 업데이트 연산자를 사용하여 최근 점수의 고정 크기 배열을 $push 유지하는 방법을 보여줍니다.

샘플 문서 생성

```
db.students.insertOne({
  _id: 1,
  name: "Alice",
  scores: [85, 90, 78]
});
```

쿼리 예제

```
db.students.updateOne(
  { _id: 1 },
  {
    $push: {
      scores: {
        $each: [92, 88],
        $slice: -3
      }
    }
  }
)
```

출력

```
{
```

```
  "_id" : 1,
  "name" : "Alice",
  "scores" : [ 78, 92, 88 ]
}
```

이 예제에서 `$slice: -3` 한정자는 배열에 새 값을 푸시한 후 마지막 세 요소만 유지합니다.

코드 예제

`$slice` 업데이트 연산자 사용에 대한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function updateDocument() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('students');

  await collection.updateOne(
    { _id: 1 },
    {
      $push: {
        scores: {
          $each: [92, 88],
          $slice: -3
        }
      }
    }
  );

  const updatedDocument = await collection.findOne({ _id: 1 });
  console.log(updatedDocument);

  await client.close();
}

updateDocument();
```

Python

```
from pymongo import MongoClient

def update_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client.test
    collection = db.students

    collection.update_one(
        {'_id': 1},
        {
            '$push': {
                'scores': {
                    '$each': [92, 88],
                    '$slice': -3
                }
            }
        }
    )

    updated_document = collection.find_one({'_id': 1})
    print(updated_document)

    client.close()

update_document()
```

$sort

$sort 업데이트 한정자는 $push 연산자와 함께 사용할 때 배열 요소를 정렬합니다. 지정된 필드 값 또는 요소 자체를 기준으로 배열 요소를 오름차순 또는 내림차순으로 정렬합니다.

파라미터

- field: 수정할 배열 필드입니다.
- order: 오1름차순 또는 내림차순에 -1 사용합니다.

예제(MongoDB 쉘)

다음 예제에서는 $sort에서 수정자를 사용하여 새 퀴즈 점수를 $push 추가하고 내림차순으로 정렬된 상태를 유지하는 방법을 보여줍니다.

샘플 문서 생성

```
db.students.insertOne({
  _id: 1,
  name: "Bob",
  quizzes: [
    { score: 85, date: "2024-01-15" },
    { score: 92, date: "2024-02-10" }
  ]
});
```

쿼리 예제

```
db.students.updateOne(
  { _id: 1 },
  {
    $push: {
      quizzes: {
        $each: [{ score: 78, date: "2024-03-05" }],
        $sort: { score: -1 }
      }
    }
  }
)
```

출력

```
{
  "_id" : 1,
  "name" : "Bob",
  "quizzes" : [
    { "score" : 92, "date" : "2024-02-10" },
    { "score" : 85, "date" : "2024-01-15" },
    { "score" : 78, "date" : "2024-03-05" }
  ]
}
```

코드 예제

$sort 업데이트 한정자를 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function updateDocument() {
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('students');

  await collection.updateOne(
    { _id: 1 },
    {
      $push: {
        quizzes: {
          $each: [{ score: 78, date: "2024-03-05" }],
          $sort: { score: -1 }
        }
      }
    }
  );

  const updatedDocument = await collection.findOne({ _id: 1 });
  console.log(updatedDocument);

  await client.close();
}

updateDocument();
```

Python

```
from pymongo import MongoClient

def update_document():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
```

```
    db = client.test
    collection = db.students

    collection.update_one(
        {'_id': 1},
        {
            '$push': {
                'quizzes': {
                    '$each': [{'score': 78, 'date': '2024-03-05'}],
                    '$sort': {'score': -1}
                }
            }
        }
    )

    updated_document = collection.find_one({'_id': 1})
    print(updated_document)

    client.close()

update_document()
```

$unset

Amazon DocumentDB의 $unset 연산자는 문서에서 지정된 필드를 제거하는 데 사용됩니다. 를 사용하여 필드를 제거하면 문서에서 $unset필드가 삭제되고 그에 따라 문서 크기가 줄어듭니다. 이는 문서에서 불필요한 데이터를 제거하려는 경우에 유용할 수 있습니다.

파라미터

- field: 문서에서 제거할 필드입니다. 이는 단일 필드이거나 중첩된 필드의 점선 경로일 수 있습니다.

예제(MongoDB 쉘)

다음 예제에서는 $unset 연산자를 사용하여 example 컬렉션의 문서에서 Words 필드를 제거하는 방법을 보여줍니다.

샘플 문서 생성

```
db.example.insert({
```

```
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DoW": "Saturday"
    },
    "Words": 2482
})
```

쿼리 예제

```
db.example.update(
    { "DocName" : "Document 1" },
    { $unset: { Words:1 } }
)
```

출력

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DoW": "Saturday"
    }
}
```

이 예제에서는 $unset 연산자를 사용하여 "문서 1"과 DocName 동일한 로 문서에서 Words 필드를 제거합니다. 결과 문서에는 더 이상 Words 필드가 포함되지 않습니다.

코드 예제

$unset 명령을 사용하기 위한 코드 예제를 보려면 사용하려는 언어의 탭을 선택합니다.

Node.js

```
const { MongoClient } = require('mongodb');

async function removeField() {
```

```
  const client = await MongoClient.connect('mongodb://
<username>:<password>@<cluster-endpoint>:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false');
  const db = client.db('test');
  const collection = db.collection('example');

  const result = await collection.updateOne(
    { "DocName": "Document 1" },
    { $unset: { "Words": 1 } }
  );

  console.log(`Modified ${result.modifiedCount} document(s)`);
  client.close();
}

removeField();
```

Python

```
from pymongo import MongoClient

def remove_field():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    collection = db['example']

    result = collection.update_one(
        {"DocName": "Document 1"},
        {"$unset": {"Words": 1}}
    )

    print(f"Modified {result.modified_count} document(s)")
    client.close()

remove_field()
```

Amazon DocumentDB 클러스터 및 인스턴스 관리

이 단원에서는 Amazon DocumentDB(MongoDB 호환) 구현을 관리하기 위한 여러 가지 구성 요소와 관련 작업을 설명합니다.

주제

- Amazon DocumentDB 운영 작업 개요
- Amazon DocumentDB 글로벌 클러스터의 개요
- Amazon DocumentDB 클러스터 관리
- Amazon DocumentDB 인스턴스 관리
- Amazon DocumentDB 서브넷 그룹 관리
- Amazon DocumentDB 고가용성 및 복제
- Amazon DocumentDB 이벤트 관리
- Amazon DocumentDB 클러스터 파라미터 그룹 관리
- Amazon DocumentDB 엔드포인트에 대한 이해
- Amazon DocumentDB Amazon 리소스 이름(ARN) 이해
- Amazon DocumentDB 리소스 태그 지정
- Amazon DocumentDB 유지 관리
- 서비스 연결 역할 이해

Amazon DocumentDB 운영 작업 개요

이 섹션에서는 Amazon DocumentDB 클러스터의 운영 작업 개요 및 AWS CLI를 사용하여 이러한 작업을 수행하는 방법을 소개합니다.

주제

- Amazon DocumentDB 클러스터에 복제본 추가
- 클러스터 및 인스턴스 설명
- 클러스터 스냅샷 생성
- 스냅샷에서 복원
- 클러스터에서 인스턴스 제거

- 클러스터 삭제

Amazon DocumentDB 클러스터에 복제본 추가

Amazon DocumentDB 클러스터에 대한 기본 인스턴스를 만든 후에 1개 이상의 복제본을 추가할 수 있습니다. 복제본은 두 가지 용도로 사용되는 읽기 전용 인스턴스입니다.

- 확장성 — 동시 액세스가 필요한 클라이언트가 많은 경우 읽기 규모 조정을 위해 복제본을 더 추가할 수 있습니다.
- 고가용성 — 기본 인스턴스에 장애가 발생하면 Amazon DocumentDB에서는 복제본 인스턴스로 자동으로 장애 조치하고 이를 새로운 기본 인스턴스로 지정합니다. 복제본이 실패하면 실패한 노드가 복구될 때까지 클러스터의 다른 인스턴스가 계속 요청을 수행할 수 있습니다.

각 Amazon DocumentDB 클러스터에서 최대 15개의 복제본을 지원할 수 있습니다.

> **Note**
>
> 내결함성을 최대화하려면 별도의 가용 영역에 복제본을 배포해야 합니다. 이렇게 하면 전체 가용 영역을 사용할 수 없게 되는 경우에도 Amazon DocumentDB 클러스터가 계속 작동할 수 있습니다.

다음 AWS CLI 예제에서는 새 복제본을 추가하는 방법을 보여줍니다. `--availability-zone` 파라미터는 복제본을 지정된 가용 영역에 배치합니다.

```
aws docdb create-db-instance \
    --db-instance-identifier sample-instance \
    --db-cluster-identifier sample-cluster \
    --engine docdb \
    --db-instance-class db.r5.large \
    --availability-zone us-east-1a
```

클러스터 및 인스턴스 설명

다음 AWS CLI 예시에서는 리전의 모든 Amazon DocumentDB 클러스터를 나열합니다. 클러스터 및 인스턴스 수명주기 관리와 같은 특정 관리 기능의 경우 Amazon DocumentDB는 Amazon RDS와 공

유하는 운영 기술을 활용합니다. `filterName=engine,Values=docdb` 필터 파라미터는 Amazon DocumentDB 클러스터만 반환합니다.

클러스터 설명 및 수정에 대한 자세한 내용은 Amazon DocumentDB 클러스터 수명 주기 섹션을 참조하십시오.

```
aws docdb describe-db-clusters --filter Name=engine,Values=docdb
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBClusters": [
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-1",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        },
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-2",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        },
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
```

```
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-3",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        }
    ]
}
```

다음 AWS CLI 예시에서는 Amazon DocumentDB 클러스터의 인스턴스를 나열합니다. 클러스터 설명 및 수정에 대한 자세한 내용은 Amazon DocumentDB 인스턴스 수명 주기 섹션을 참조하십시오.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterMembers]'
```

출력은 아래와 같습니다. 이 출력에는 두 개의 인스턴스가 있습니다. 기본 인스턴스는 `sample-instance-1`(`"IsClusterWriter": true`)입니다. 또한 복제본 인스턴스 `sample-instance2`(`"IsClusterWriter: false"`)도 있습니다.

```
[
    [
        [
            {
                "DBInstanceIdentifier": "sample-instance-1",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            },
            {
                "DBInstanceIdentifier": "sample-cluster-2",
                "IsClusterWriter": false,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            }
        ]
    ]
]
```

클러스터 스냅샷 생성

클러스터 스냅샷은Amazon DocumentDB 클러스터의 데이터를 완벽하게 백업합니다. 스냅샷을 만들 때 Amazon DocumentDB는 클러스터 볼륨에서 직접 데이터를 읽습니다. 따라서 클러스터에서 해당 인스턴스가 실행 중이 아니어도 스냅샷을 생성할 수 있습니다. 클러스터 스냅샷을 생성하는 데 걸리는 시간은 클러스터 볼륨의 크기에 따라 다릅니다.

Amazon DocumentDB는 자동 백업을 지원합니다. 자동 백업은 선호하는 백업 윈도우(하루 중 30분) 동안 매일 수행됩니다. 다음 AWS CLI 예제에서는 클러스터의 백업 기간을 보는 방법을 보여줍니다.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].PreferredBackupWindow'
```

출력에는 백업 창(UTC)이 표시됩니다.

```
[
    "00:18-00:48"
]
```

Amazon DocumentDB 클러스터를 생성할 때 백업 기간을 정의할 수 있습니다. 다음 예제와 같이 백업 기간을 변경할 수도 있습니다. 백업 기간을 정의하지 않으면 Amazon DocumentDB에서 자동으로 클러스터에 할당합니다.

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --preferred-backup-window "02:00-02:30"
```

자동 백업 외에도 언제든지 수동으로 클러스터 스냅샷을 만들 수 있습니다. 이때 백업할 클러스터를 지정한 다음 나중에 복원할 수 있도록 스냅샷에 대해 고유한 이름을 지정합니다.

다음 AWS CLI 예제에서는 데이터의 스냅샷을 생성하는 방법을 보여줍니다.

```
aws docdb create-db-cluster-snapshot \
    --db-cluster-identifier sample-cluster \
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

스냅샷에서 복원

클러스터 스냅샷을 새 Amazon DocumentDB 클러스터로 복원할 수 있습니다. 이렇게 하려면 스냅샷의 이름과 새 클러스터의 이름을 제공합니다. 스냅샷에서 기존 클러스터로 복원할 수 없습니다. 대신 복원할 때 Amazon DocumentDB에서 새 클러스터를 만든 다음 스냅샷 데이터로 채웁니다.

다음 예제에서는 `sample-cluster` 클러스터의 모든 스냅샷을 보여 줍니다.

```
aws docdb describe-db-cluster-snapshots \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusterSnapshots[*].[DBClusterSnapshotIdentifier,SnapshotType,Status]'
```

출력은 다음과 같습니다. 수동 스냅샷은 수동으로 생성한 스냅샷이지만 자동 스냅샷은 클러스터 백업 기간 중 Amazon DocumentDB에 의해 생성됩니다.

```
[
        "sample-cluster-snapshot",
        "manual",
        "available"
    ],
    [
        "rds:sample-cluster",
        "automated",
        "available"
    ]
]
```

다음 예제에서는 스냅샷에서 Amazon DocumentDB 클러스터를 복원하는 방법을 보여 줍니다.

```
aws docdb restore-db-cluster-from-snapshot \
    --engine docdb \
    --db-cluster-identifier new-sample-cluster \
    --snapshot-identifier sample-cluster-snapshot
```

새 클러스터에 연결된 인스턴스가 없으므로 클러스터와 상호 작용하려면 인스턴스를 클러스터에 추가해야 합니다.

```
aws docdb create-db-instance \
    --db-instance-identifier new-sample-instance \
    --db-instance-class db.r5.large \
```

```
    --engine docdb \
    --db-cluster-identifier new-sample-cluster
```

다음 AWS CLI 작업을 사용하여 클러스터 및 인스턴스 생성 진행 상황을 모니터링할 수 있습니다. 클러스터 및 인스턴스 상태를 사용할 수 있으면 새 클러스터의 엔드포인트에 연결하여 데이터에 액세스할 수 있습니다.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier new-sample-cluster  \
    --query 'DBClusters[*].[Status,Endpoint]'
```

```
aws docdb describe-db-instances \
    --db-instance-identifier new-sample-instance \
    --query 'DBInstances[*].[DBInstanceStatus]'
```

클러스터에서 인스턴스 제거

Amazon DocumentDB에서는 모든 데이터를 클러스터 볼륨에 저장합니다. 모든 인스턴스를 클러스터에서 제거하는 경우에도 데이터가 클러스터 볼륨에 유지됩니다. 데이터에 다시 액세스해야 하는 경우 언제든지 클러스터에 인스턴스를 추가하고 중단한 지점을 선택할 수 있습니다.

다음 예제에서는 Amazon DocumentDB 클러스터에서 인스턴스를 제거하는 방법을 보여줍니다.

```
aws docdb delete-db-instance \
    --db-instance-identifier sample-instance
```

클러스터 삭제

Amazon DocumentDB 클러스터를 삭제하려면 먼저 해당 인스턴스를 모두 제거해야 합니다. 다음 AWS CLI 예제에서는 클러스터의 인스턴스에 대한 정보를 반환합니다. 이 작업으로 인스턴스 식별자가 반환되면 각 인스턴스를 삭제해야 합니다. 자세한 내용은 클러스터에서 인스턴스 제거 단원을 참조하십시오.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterMembers[*].DBInstanceIdentifier'
```

더 이상 인스턴스가 남아 있지 않으면 클러스터를 삭제할 수 있습니다. 이때 다음 옵션 중 하나를 선택해야 합니다.

- 최종 스냅샷 생성 — 나중에 해당 데이터로 새 인스턴스를 다시 만들 수 있도록 스냅샷의 모든 클러스터 데이터를 캡처합니다. 다음 예에서는 이 작업을 수행하는 방법을 보여줍니다.

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --final-db-snapshot-identifier sample-cluster-snapshot
```

- 최종 스냅샷 건너뛰기 — 모든 클러스터 데이터를 영구적으로 삭제합니다. 이 작업은 되돌릴 수 없습니다. 다음 예에서는 이 작업을 수행하는 방법을 보여줍니다.

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --skip-final-snapshot
```

Amazon DocumentDB 글로벌 클러스터의 개요

글로벌 클러스터란 무엇입니까?

글로벌 클러스터는 하나의 기본 리전과 최대 10개의 읽기 전용 보조 리전으로 구성됩니다. 기본 리전의 기본 클러스터에 직접 쓰기 작업을 실행하면 Amazon DocumentDB가 전용 인프라를 사용하여 보조 리전에 데이터를 자동으로 복제합니다. 지연 시간은 일반적으로 1초 미만입니다.

글로벌 클러스터는 어떻게 유용할까요?

- 리전 전체의 중단으로부터 복구 - 리전 전체의 중단이 발생하는 경우 일반적인 Recovery Time Objective(RTO)가 1분 미만인 몇 분 내에 보조 클러스터 중 하나를 기본 클러스터로 승격할 수 있습니다. Recovery Point Objective(RPO)는 일반적으로 초 단위로 측정되지만, 장애 발생 시 네트워크를 통한 스토리지 복제 지연에 따라 달라집니다.
- 로컬 지연 시간이 있는 글로벌 읽기 - 전 세계에 지사가 있는 경우 글로벌 클러스터를 사용하여 기본 리전에서 주요 정보 소스를 최신 상태로 유지할 수 있습니다. 다른 리전의 사무실은 로컬 지연 시간을 사용하여 해당 리전의 정보에 액세스할 수 있습니다.
- 확장 가능한 보조 클러스터 - 보조 리전에 읽기 전용 인스턴스를 더 추가하여 보조 클러스터를 확장할 수 있습니다. 보조 클러스터는 읽기 전용이므로 단일 클러스터에 대한 일반적인 제한인 15개가 아닌 최대 16개의 읽기 전용 복제본 인스턴스를 지원할 수 있습니다.
- 일차적 클러스터에서 이차 클러스터로 빠른 복제 - 글로벌 클러스터에서 수행하는 복제는 기본 데이터베이스 클러스터에 대한 성능 영향이 거의 없습니다. DB 인스턴스의 리소스는 애플리케이션 읽기/쓰기 워크로드 처리 전용입니다.

- 스트림 변경 - 글로벌 클러스터의 보조 클러스터에서 변경 스트림을 읽을 수 있으므로 기본 클러스터의 부하를 줄이고 보조 리전에서 읽기 워크로드를 확장할 수 있습니다.

글로벌 클러스터의 현재 한계는 무엇입니까?

- Amazon DocumentDB v3.6에서는 글로벌 클러스터가 지원되지 않습니다.
- 글로벌 클러스터는 db.t3, db.t4g 및 db.r4를 제외한 모든 인스턴스 유형에서 지원됩니다.
- 글로벌 클러스터는 남아메리카(상파울루), 유럽(밀라노), 중국(베이징), 중국(닝샤) 리전에서 사용할 수 없습니다.
- 리전이 다른 엔진 버전에 있는 경우에는 전환 및 글로벌 장애 조치가 지원되지 않습니다. 엔진 버전 불일치가 있는 경우 수동 장애 조치가 지원됩니다.
- 기본 클러스터만 쓰기 작업을 수행합니다. 쓰기 작업을 수행하는 클라이언트는 기본 클러스터의 DB 클러스터 엔드포인트에 연결합니다.
- 클러스터에는 최대 10개의 보조 리전과 1개의 기본 리전이 있을 수 있습니다.
- 보조 클러스터는 중지할 수 없습니다. 기본 클러스터에 보조 클러스터가 연결되어 있는 경우 기본 클러스터를 중지할 수 없습니다. 보조 클러스터가 없는 리전 클러스터만 중지할 수 있습니다.
- 보조 클러스터에 연결된 복제본은 특정 상황에서 다시 시작될 수 있습니다. 기본 리전의 인스턴스가 다시 시작되거나 장애 조치되면 보조 리전의 복제본도 다시 시작됩니다. 그러면 모든 복제본이 주 데이터베이스 클러스터의 작성자 인스턴스와 다시 동기화될 때까지 클러스터를 사용할 수 없습니다. 이는 예상된 동작입니다. 기본 클러스터를 변경하기 전에 글로벌 클러스터에 미치는 영향을 이해해야 합니다.

주제

- 빠른 시작 설명서: 글로벌 클러스터
- Amazon DocumentDB 글로벌 클러스터 관리
- Amazon DocumentDB 글로벌 클러스터에 연결
- Amazon DocumentDB 글로벌 클러스터 모니터링
- 재해 복구 및 Amazon DocumentDB 글로벌 클러스터

빠른 시작 설명서: 글로벌 클러스터

주제

- 구성

- Amazon DocumentDB 글로벌 클러스터 생성
- Amazon DocumentDB 글로벌 클러스터 AWS 리전 에 추가
- Amazon DocumentDB 글로벌 클러스터에 스냅샷 사용

구성

Amazon DocumentDB 글로벌 클러스터는 2개 이상에 걸쳐 있습니다 AWS 리전. 기본 리전은 기본(라이터) 인스턴스 1개와 복제본 인스턴스 최대 15개로 구성된 클러스터를 지원하는 반면, 보조 리전은 최대 16개의 복제본 인스턴스로 구성된 읽기 전용 클러스터를 실행합니다. 글로벌 클러스터에는 최대 10개의 보조 리전이 있을 수 있습니다. 이 표에는 글로벌 클러스터에서 허용되는 최대 클러스터, 인스턴스 및 복제본이 나열됩니다.

설명	기본 AWS 리전	보조 AWS 리전
클러스터	1	10개(최대)
라이터 인스턴스	1	0
클러스터당 읽기 전용 인스턴스(Amazon DocumentDB 복제본)	15(최대)	16(최대)
읽기 전용 인스턴스(최대 허용, 지정된 보조 리전 수)	15 - s	s = 총 보조 수 AWS 리전

클러스터의 특정 요구 사항은 다음과 같습니다.

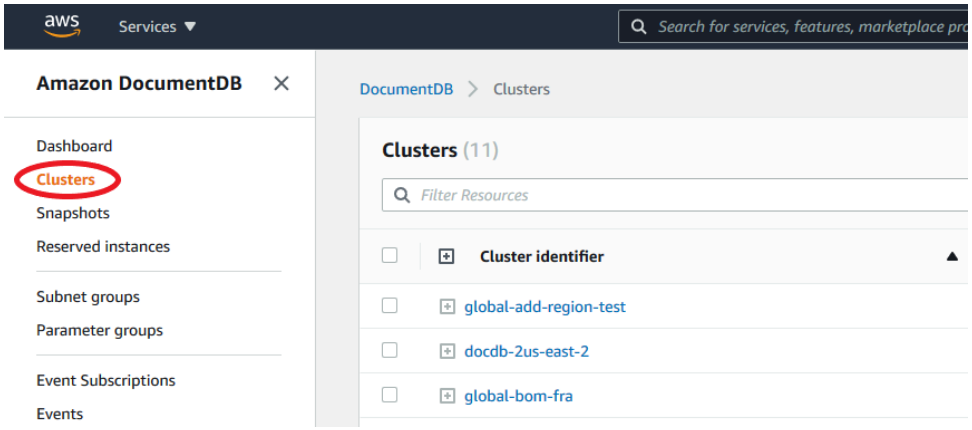
- 데이터베이스 인스턴스 클래스 요구 사항 - `db.r5` 및 `db.r6g` 인스턴스 클래스만 사용할 수 있습니다.
- AWS 리전 요구 사항 - 기본 클러스터는 한 리전에 있어야 하며, 하나 이상의 보조 클러스터는 동일한 계정의 다른 리전에 있어야 합니다. 최대 10개의 보조(읽기 전용) 클러스터를 생성할 수 있으며 각 클러스터는 다른 리전에 있어야 합니다. 즉, 두 클러스터가 동일한 리전에 있을 수 없습니다.
- 이름 설정 요구 사항 - 개별 클러스터에 선택하는 이름은 모든 리전에서 고유해야 합니다. 다른 리전에 있더라도 다른 클러스터에는 동일한 이름을 사용할 수 없습니다.

Amazon DocumentDB 글로벌 클러스터 생성

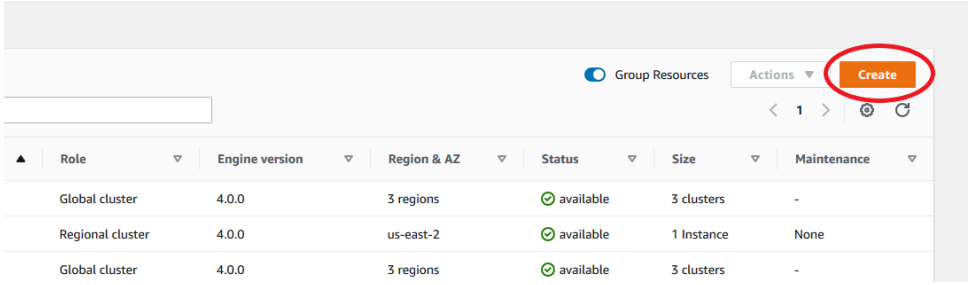
첫 번째 글로벌 클러스터를 구축할 준비가 되셨나요? 이 섹션에서는 다음 지침에 따라 AWS Management Console 또는를 사용하여 새 데이터베이스 클러스터 및 인스턴스로 새 글로벌 클러스터 AWS CLI 를 생성하는 방법을 설명합니다.

사용 AWS Management Console

1. 에서 Amazon DocumentDB로 AWS Management Console이동합니다.
2. Amazon DocumentDB 콘솔로 이동하면 클러스터를 선택합니다.

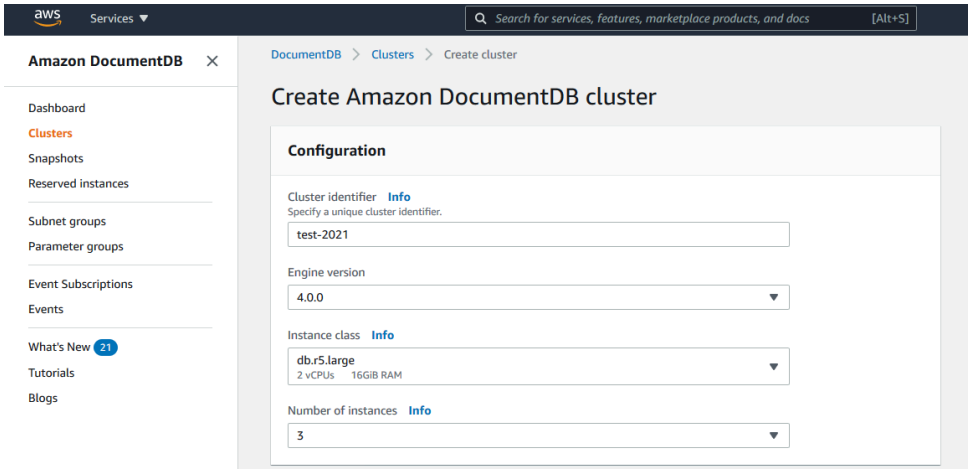


3. 생성(Create)을 선택합니다.

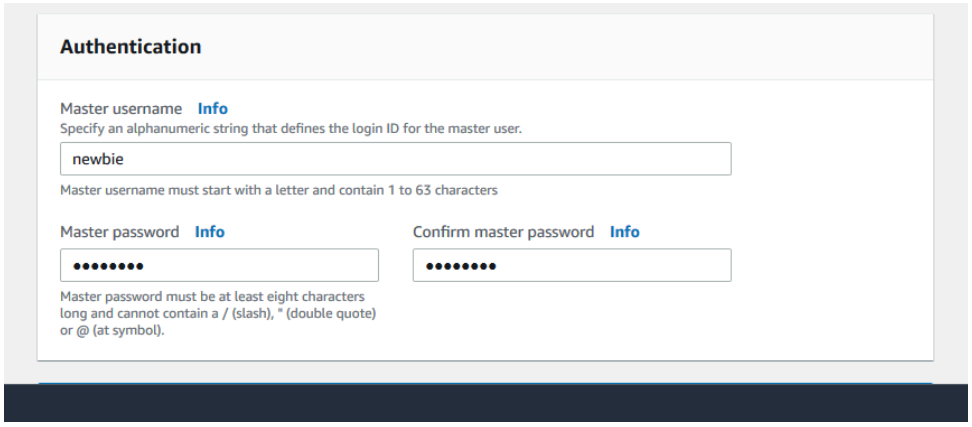


4. Amazon DocumentDB 클러스터 생성 양식의 구성 섹션을 적절히 작성하세요.

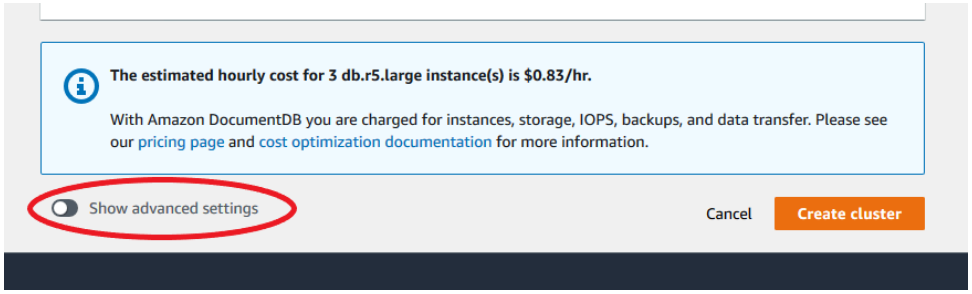
   - 클러스터 식별자 - 이 인스턴스의 고유 식별자를 입력하거나 Amazon DocumentDB가 클러스터 식별자를 기반으로 인스턴스 식별자를 제공하도록 허용할 수 있습니다.
   - 엔진 버전: 4.0.0 선택
   - 인스턴스 클래스: db.r5.large를 선택합니다.
   - 인스턴스 수에 3을 선택합니다.



5. 인증 섹션에서 마스터 사용자 이름과 마스터 비밀번호를 입력합니다.

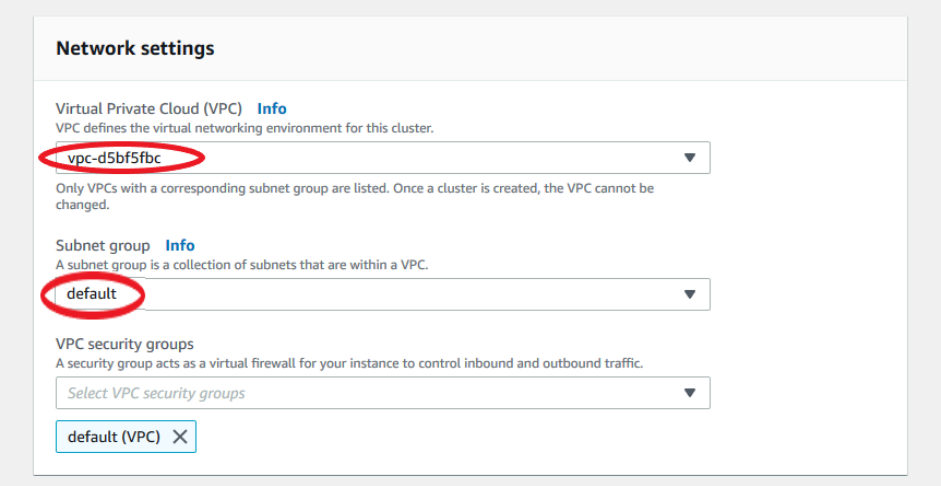


6. 고급 설정 표시를 선택합니다.

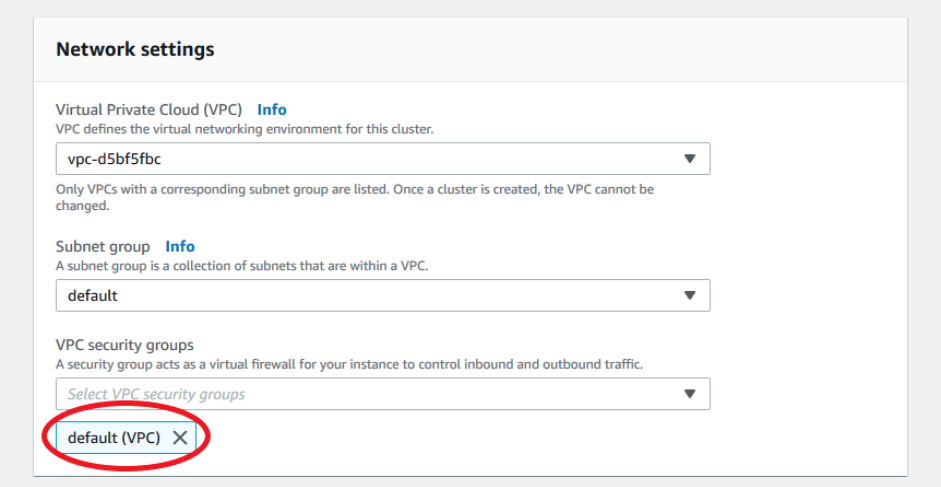


7. 네트워크 설정 섹션에서:

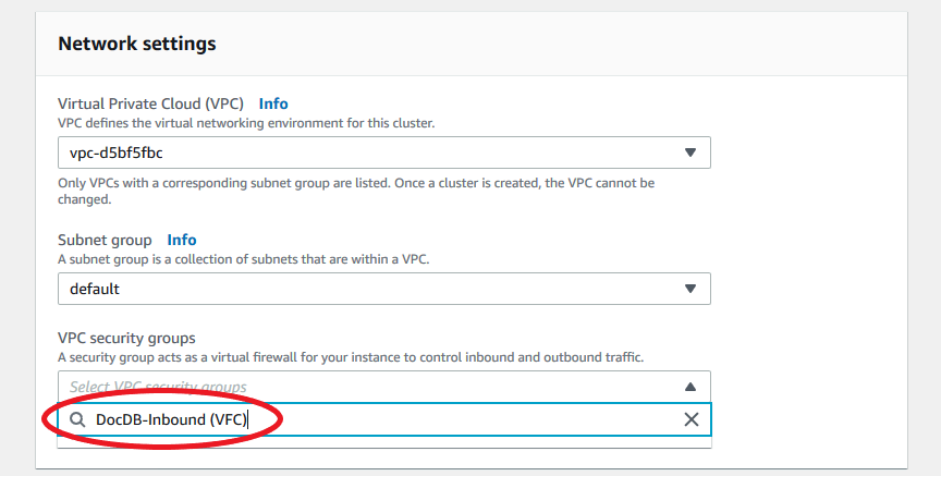
   - 가상 프라이빗 클라우드(VPC) 및 서브넷 그룹의 기본 옵션을 유지합니다.



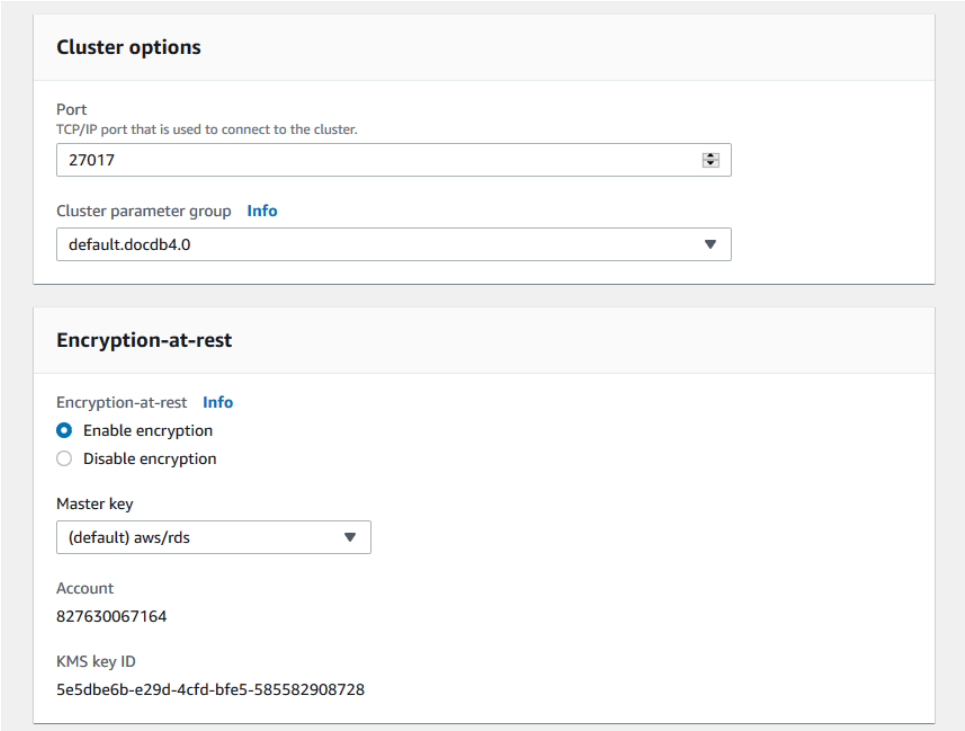
- VPC 보안 그룹의 경우 기본 VPC가 이미 추가되어 있어야 합니다.



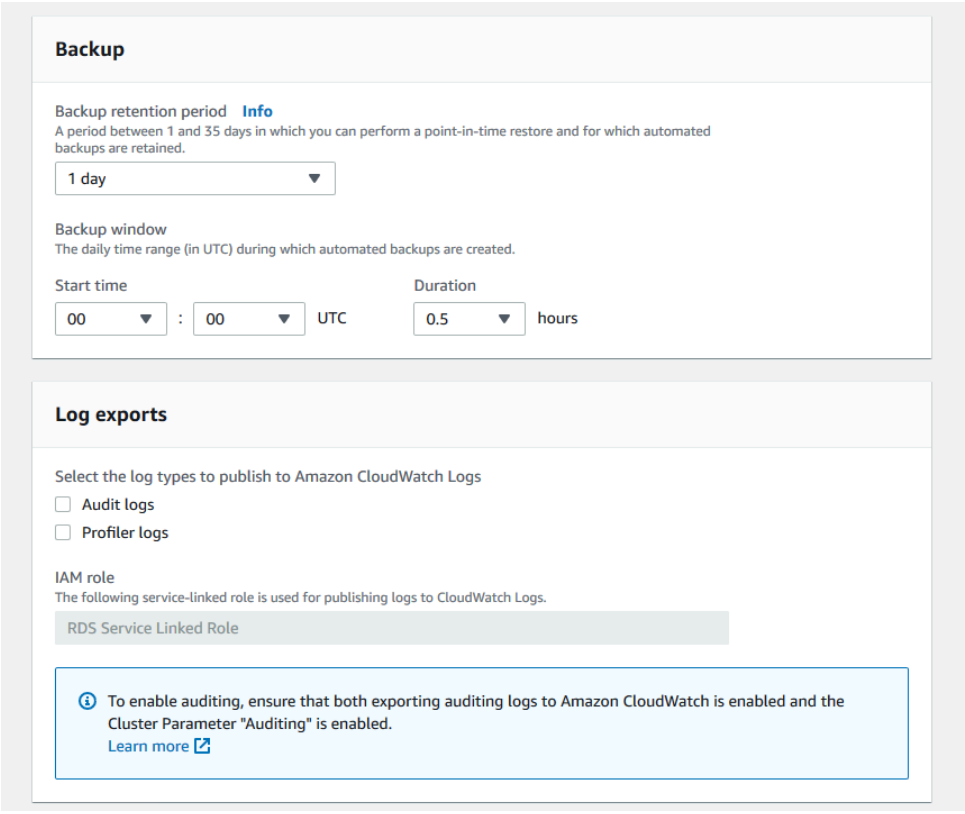
- DocDB를 VPC 보안 그룹 필드에 입력하고 DocDB-인바운드(VFC)를 선택합니다.



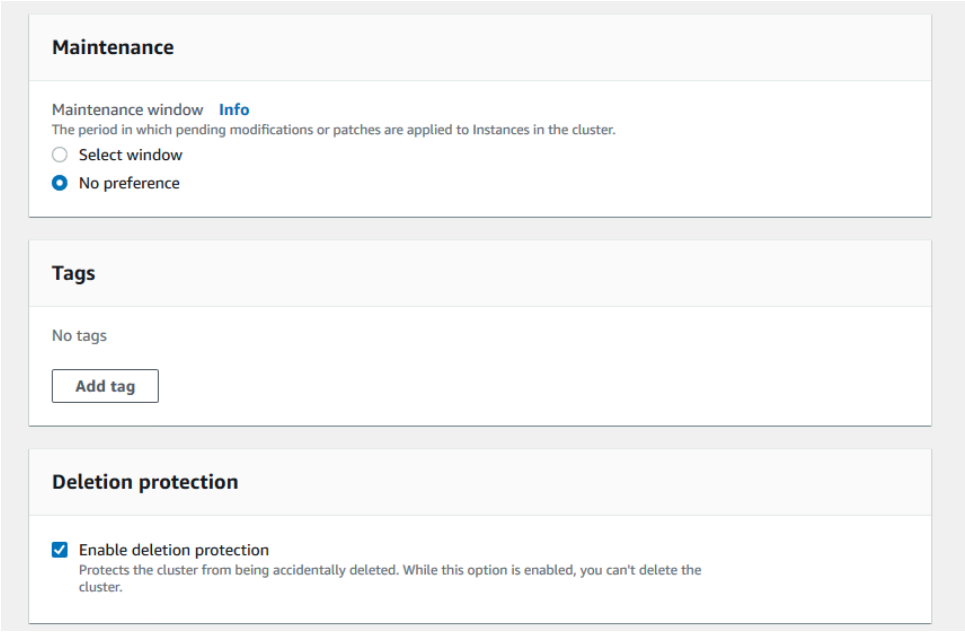
8. 클러스터 옵션 및 저장 중 암호화의 경우 기본 선택 사항을 그대로 유지합니다.



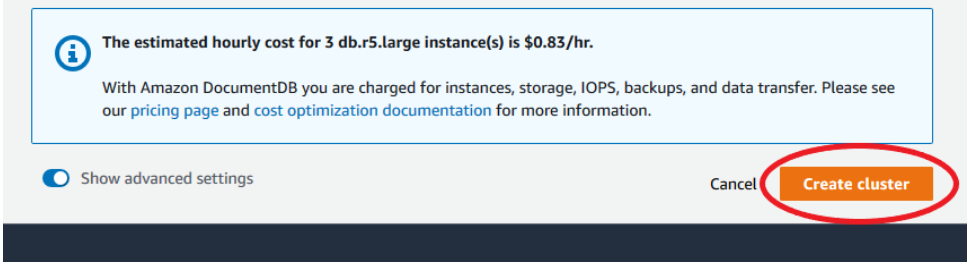
9. 백업 및 로그 내보내기의 경우 기본 선택 사항을 그대로 유지합니다.



10. 유지 관리, 태그 및 삭제 보호의 경우 기본 선택 사항을 그대로 유지합니다.



11. 이제 클러스터 생성을 선택합니다.



사용 AWS CLI

Amazon DocumentDB 리전 클러스터를 생성하려면 create-global-cluster AWS CLI를 호출합니다. 다음 AWS CLI 명령은 라는 Amazon DocumentDB 클러스터를 생성합니다`global-cluster-id`. 삭제 방지에 대한 자세한 내용은 Amazon DocumentDB 클러스터 삭제을(를) 참조하세요.

또한 `--engine-version`은 기본적으로 최신 주요 엔진 버전으로 설정되는 선택적 매개변수입니다. 현재 기본 엔진 버전은 입니다`5.0.0`(참고: Amazon DocumentDB 8.0을 사용할 수 있지만 로 명시적으로 지정해야 함`8.0.0`). 새 메이저 엔진 버전이 릴리스되면 마지막 메이저 엔진 버전을 반영하도록의 기본 엔진 버전이 `--engine-version` 업데이트됩니다. 따라서 프로덕션 워크로드, 특히 스크립팅, 자동화 또는 CloudFormation 템플릿에 의존하는 워크로드의 경우를 의도한 메이저 버전`--engine-version`에 명시적으로 지정합니다.

`db-subnet-group-name` 또는 `vpc-security-group-id`이(가) 지정되지 않은 경우 Amazon DocumentDB는 해당 리전에 대해 기본 서브넷 그룹과 Amazon VPC 보안 그룹을 사용합니다.

다음 예제에서는 자신의 정보로 각각의 *### ## ## ###*를 바꿉니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb create-db-cluster \
      --global-cluster-identifier global-cluster-id \
      --source-db-cluster-identifier arn:aws:rds:us-east-1:111122223333:cluster-id
```

Windows의 경우:

```
aws docdb create-db-cluster ^
      --global-cluster-identifier global-cluster-id ^
      --source-db-cluster-identifier arn:aws:rds:us-east-1:111122223333:cluster-id
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "DBCluster": {
        "StorageEncrypted": false,
        "DBClusterMembers": [],
        "Engine": "docdb",
        "DeletionProtection" : "enabled",
        "ClusterCreateTime": "2018-11-26T17:15:19.885Z",
        "DBSubnetGroup": "default",
        "EngineVersion": "4.0.0",
        "MasterUsername": "masteruser",
        "BackupRetentionPeriod": 1,
        "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:cluster-id",
        "DBClusterIdentifier": "cluster-id",
        "MultiAZ": false,
        "DBClusterParameterGroup": "default.docdb4.0",
        "PreferredBackupWindow": "09:12-09:42",
        "DbClusterResourceId": "cluster-KQSGI4MHU4NTDDRVNLNTU7XVAY",
        "PreferredMaintenanceWindow": "tue:04:17-tue:04:47",
        "Port": 27017,
        "Status": "creating",
        "ReaderEndpoint": "cluster-id.cluster-ro-sfcrlcjcoroz.us-
east-1.docdb.amazonaws.com",
        "AssociatedRoles": [],
        "HostedZoneId": "ZNKXTT8WH85VW",
```

```
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-77186e0d",
                "Status": "active"
            }
        ],
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1c",
            "us-east-1e"
        ],
        "Endpoint": "cluster-id.cluster-sfcrlcjcoroz.us-east-1.docdb.amazonaws.com"
    }
}
```

클러스터를 생성하는 데 몇 분 정도 걸립니다. AWS Management Console 또는를 사용하여 클러스터의 상태를 AWS CLI 모니터링할 수 있습니다. 자세한 내용은 Amazon DocumentDB 클러스터 상태 모니터링 단원을 참조하십시오.

> **Important**
>
> AWS CLI 를 사용하여 Amazon DocumentDB 리전 클러스터를 생성하면 인스턴스가 생성되지 않습니다. 따라서 기본 인스턴스와 필요한 복제 인스턴스를 명시적으로 생성해야 합니다. 콘솔 또는를 사용하여 인스턴스 AWS CLI 를 생성할 수 있습니다. 자세한 내용은 Amazon DocumentDB API 참조의 클러스터에 Amazon DocumentDB 인스턴스 추가 및 CreateDBCluster를 참조하세요. Amazon DocumentDB

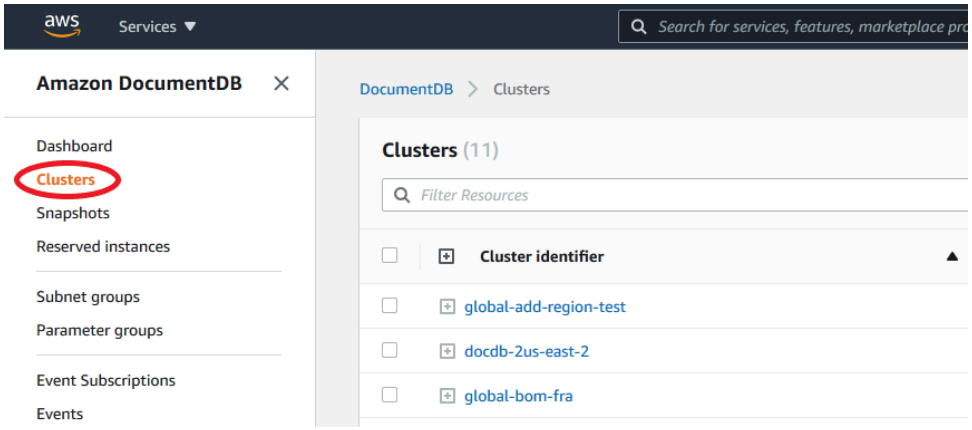
리전 클러스터를 사용할 수 있게 되면 Amazon DocumentDB 글로벌 클러스터 AWS 리전 에 추가 지침에 따라 다른 리전에 보조 클러스터를 추가할 수 있습니다. 리전을 추가하면 리전 클러스터가 기본 클러스터가 되고 선택한 리전에 새 보조 클러스터가 생깁니다.

Amazon DocumentDB 글로벌 클러스터 AWS 리전 에 추가

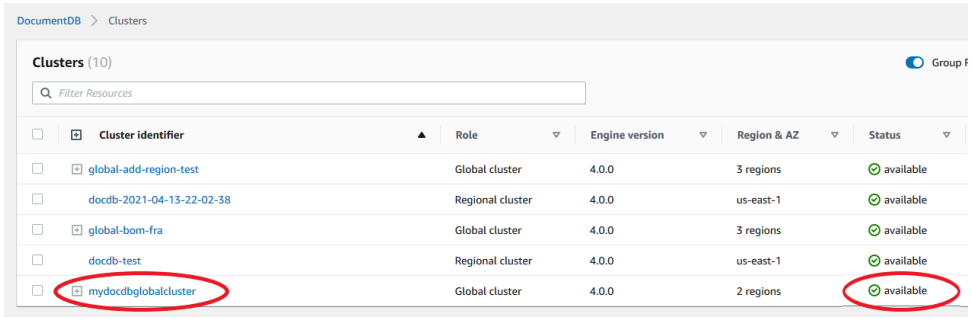
글로벌 클러스터에는 기본 클러스터와 다른 리전에 하나 이상의 보조 클러스터가 필요하며 최대 10개의 보조 클러스터를 추가할 수 있습니다. 추가하는 각 보조 클러스터에 대해 기본 클러스터에 허용되는 복제본 수를 하나씩 줄여야 한다는 점에 유의하세요. 예를 들어 글로벌 클러스터에 보조 리전이 10개 있는 경우 기본 클러스터에는 (15개가 아닌) 복제본이 5개만 있을 수 있습니다. 자세한 내용은 구성 단원을 참조하십시오.

사용 AWS Management Console

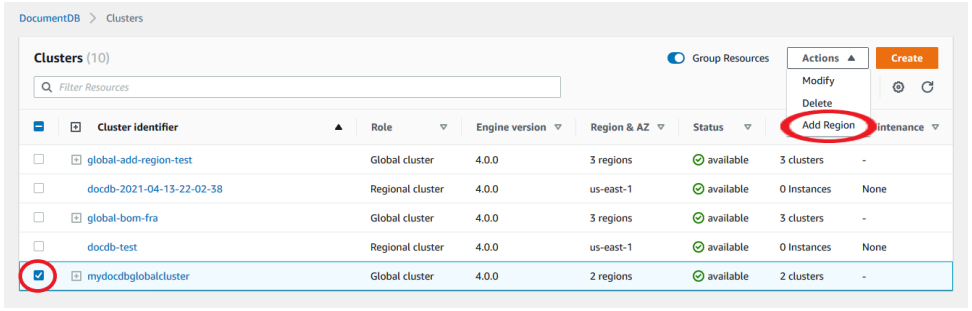
1. 에 로그인 AWS Management Console 하고 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.



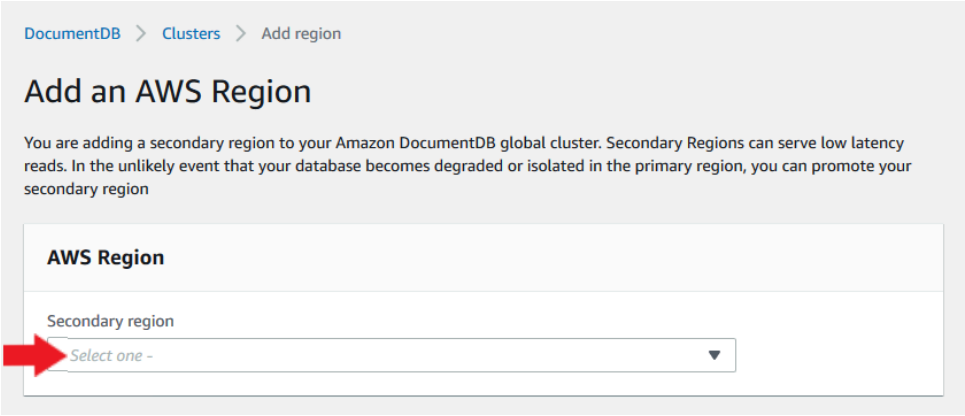
3. 보조 클러스터를 추가할 클러스터를 선택합니다. 클러스터가 `Available`인지 확인합니다.



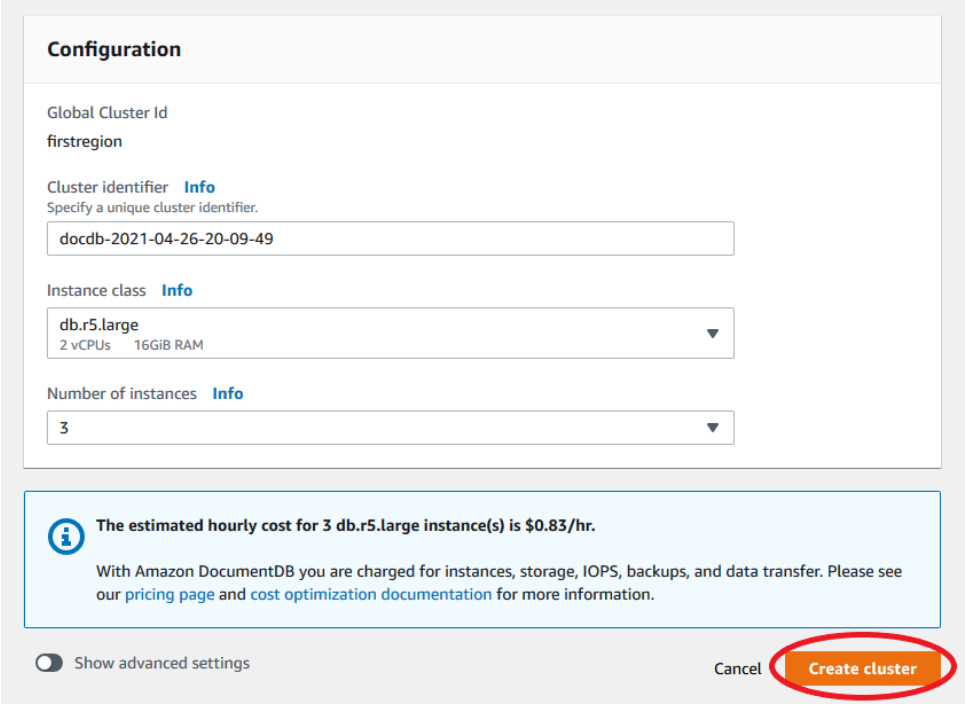
4. 작업의 드롭다운 목록을 선택한 다음 리전 추가를 선택합니다.



5. AWS 리전추가 페이지에서 보조 리전을 선택합니다. 동일한 글로벌 데이터베이스에 대해 보조 DB 클러스터가 이미 있는 리전을 선택할 수 없다는 점에 유의하세요. 또한 이는 기본 클러스터와 동일한 리전이 될 수 없습니다. 이 리전을 처음 추가하는 경우 선택한 글로벌 클러스터 식별자도 지정해야 합니다.



6. 새 리전에서 보조 클러스터에 대한 나머지 필드를 완료합니다. 이후 클러스터 생성을 선택합니다. 리전 추가를 완료하면 AWS Management Console의 클러스터 목록에서 볼 수 있습니다.



사용 AWS CLI

- `create-db-cluster` CLI 명령을 글로벌 클러스터의 이름 (`--global-cluster-identifier`)과 함께 사용합니다. 기타 명령 파라미터에서 다음을 수행합니다.
  - 에서 기본 리전 AWS 리전 과 다른을 `--region`선택합니다.
  - `--engine` 및 `--engine-version` 파라미터의 구체적인 값을 선택합니다.
  - 암호화된 클러스터의 경우 기본를 암호화를 `--source-region` 위한 AWS 리전 로 지정합니다.

다음 예제에서는 새 Amazon DocumentDB 클러스터를 생성하여 글로벌 클러스터에 읽기 전용 보조 클러스터로 연결합니다. 마지막 단계에서 인스턴스가 새 클러스터에 추가됩니다.

다음 예제에서는 자신의 정보로 각각의 *### ## ## ###*를 바꿉니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb --region secondary-region-id \
  create-db-cluster \
    --db-cluster-identifier cluster-id \
    --global-cluster-identifier global-cluster-id \
    --engine-version version \
    --engine docdb

aws docdb --region secondary-region-id \
  create-db-instance \
    --db-cluster-identifier cluster-id \
    --global-cluster-identifier global-cluster-id \
    --engine-version version \
    --engine docdb
```

Windows의 경우:

```
aws docdb --region secondary-region-id ^
  create-db-cluster ^
    --db-cluster-identifier cluster-id ^
    --global-cluster-identifier global-cluster-id ^
    --engine-version version ^
    --engine docdb

aws docdb --region secondary-region-id ^
  create-db-instance ^
    --db-cluster-identifier cluster-id ^
    --global-cluster-identifier global-cluster-id ^
    --engine-version version ^
    --engine docdb
```

Amazon DocumentDB 글로벌 클러스터에 스냅샷 사용

Amazon DocumentDB 클러스터의 스냅샷을 복원하여 글로벌 클러스터의 시작점으로 사용할 수 있습니다. 이렇게 하려면 스냅샷을 복원하고 새 클러스터를 생성해야 합니다. 이는 글로벌 클러스터의 기본

클러스터 역할을 합니다. 그런 다음 복원된 클러스터에 다른 리전을 추가하여 글로벌 클러스터로 변환할 수 있습니다.

Amazon DocumentDB 글로벌 클러스터 관리

글로벌 클러스터를 구성하는 개별 클러스터에서 대부분의 관리 작업을 수행할 수 있습니다. 콘솔의 데이터베이스 페이지에서 관련 리소스 그룹화를 선택하면, 연결된 글로벌 클러스터 아래에 그룹화된 기본 클러스터와 보조 클러스터를 볼 수 있습니다.

글로벌 클러스터의 구성 탭에는 클러스터가 실행 중인 AWS 리전 , 버전 및 글로벌 클러스터 식별자가 표시됩니다.

주제

- Amazon DocumentDB 글로벌 클러스터 수정
- 파라미터 Amazon DocumentDB 글로벌 클러스터 수정
- Amazon DocumentDB 글로벌 클러스터에서 클러스터 분리
- Amazon DocumentDB 글로벌 클러스터에서 클러스터 제거
- 보조 리전에서 헤드리스 Amazon DocumentDB 클러스터 생성

Amazon DocumentDB 글로벌 클러스터 수정

의 클러스터 페이지에는 모든 글로벌 클러스터가 AWS Management Console 나열되며 각 클러스터의 기본 클러스터와 보조 클러스터가 표시됩니다. 글로벌 클러스터에는 고유한 구성 설정이 있습니다. 특히 기본 및 보조 클러스터와 연결된 리전이 있습니다.

글로벌 클러스터를 변경하면 변경 사항을 취소 할 수 있는 기회가 제공됩니다.

계속을 선택하면 변경 사항이 승인됩니다.

파라미터 Amazon DocumentDB 글로벌 클러스터 수정

글로벌 클러스터 내에서 각 클러스터에 대해 클러스터 파라미터 그룹을 별도로 구성할 수 있습니다. 대부분의 파라미터는 다른 종류의 Amazon DocumentDB 클러스터와 동일하게 작동합니다. 보조 클러스터를 기본 클러스터로 승격하는 경우 예기치 않은 동작 변경을 방지하려면 글로벌 데이터베이스의 모든 클러스터 간에 설정을 일관되게 유지합니다.

예를 들면 다른 클러스터가 기본 클러스터로 대신 사용되는 경우 일관되지 않은 동작을 방지하려면 시간대와 문자 세트에 대해 동일한 설정을 사용합니다.

Amazon DocumentDB 글로벌 클러스터에서 클러스터 분리

글로벌 클러스터에서 클러스터를 제거해야 하는 상황은 여러 가지가 있습니다. 예를 들어, 기본 클러스터가 성능이 저하되거나 격리된 경우 글로벌 클러스터에서 클러스터를 제거할 수 있습니다. 그런 다음, 프로비저닝된 독립형 클러스터가 되어, 새로운 글로벌 클러스터를 생성하는 데 사용할 수 있습니다. 자세한 내용은 Amazon DocumentDB 글로벌 클러스터에 대한 수동 장애 조치 수행를 참조하세요.

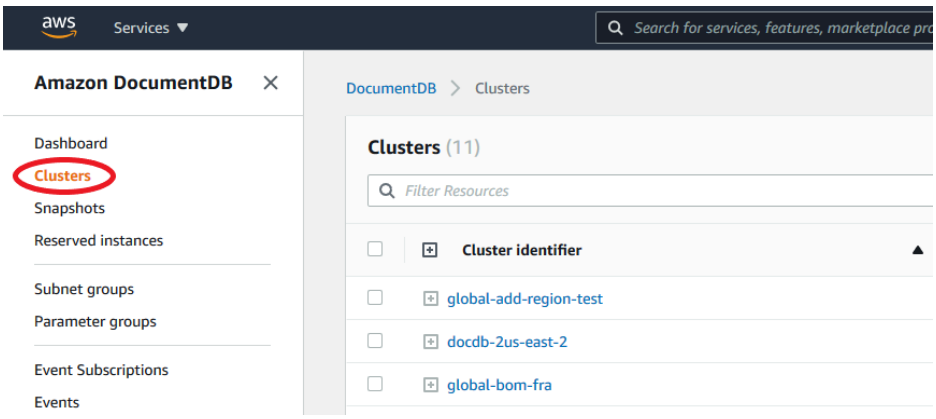
더 이상 필요하지 않은 글로벌 클러스터를 삭제하고자 하기 때문에 클러스터를 제거하고 싶을 수도 있습니다. 연결된 모든 클러스터를 분리하고 기본 클러스터를 마지막으로 남겨둘 때까지는 글로벌 클러스터를 삭제할 수 없습니다. 자세한 내용은 Amazon DocumentDB 글로벌 클러스터에서 클러스터 제거 단원을 참조하십시오.

> **Note**
>
> 클러스터가 글로벌 클러스터에서 분리되면, 더 이상 기본 클러스터와 동기화되지 않습니다. 완전한 읽기/쓰기 기능을 갖춘 프로비저닝된 독립형 클러스터가 됩니다. 또한 Amazon DocumentDB 콘솔에 더이상 표시되지 않습니다. 클러스터가 위치한 콘솔에서 리전을 선택한 경우에만 표시됩니다.

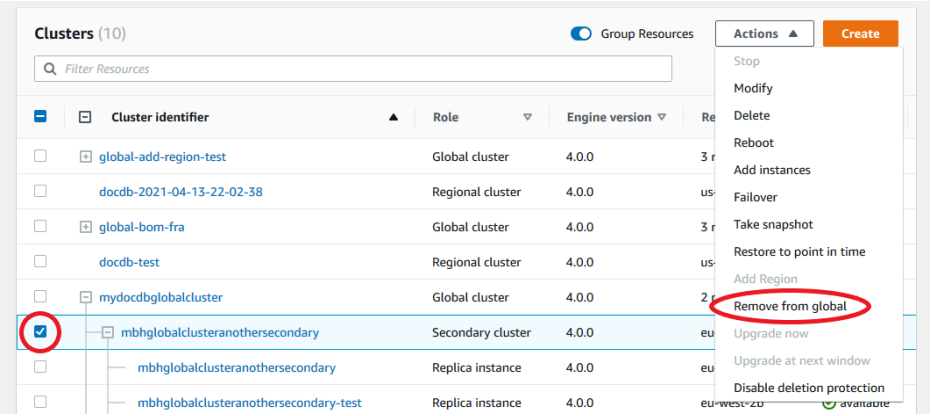
AWS Management Console AWS CLI, 또는 RDS API를 사용하여 글로벌 클러스터에서 클러스터를 제거할 수 있습니다.

Using the AWS Management Console

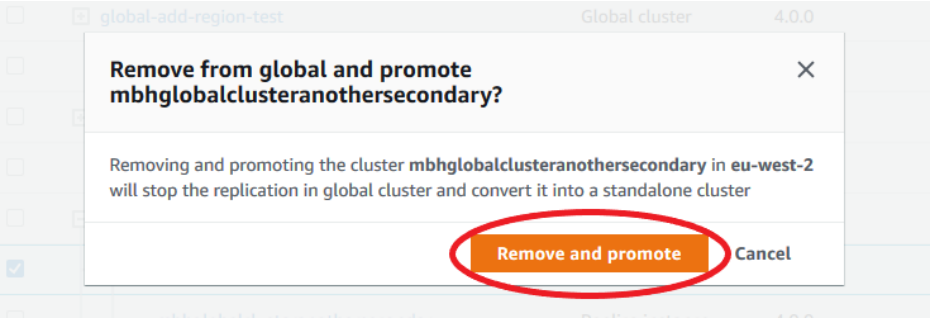
1. 에 로그인 AWS Management Console 하고 Amazon DocumentDB 콘솔로 이동합니다.
2. 왼쪽 측면의 탐색 창에서 클러스터를 선택합니다.



3. 모든 보조 클러스터를 볼 수 있도록 글로벌 클러스터를 확장하세요. 제거하려는 보조 클러스터를 선택합니다. 작업을 선택하고 드롭다운 메뉴에서 글로벌에서 제거를 선택합니다.



4. 보조 클러스터를 글로벌 클러스터에서 분리할지 확인하는 메시지가 표시됩니다. 글로벌 클러스터에서 클러스터를 제거하려면 제거 및 승격을 선택합니다.



클러스터는 더 이상 보조 클러스터로 사용되지 않으며 더 이상 기본 클러스터와 동기화되지 않습니다. 완전한 읽기/쓰기 기능을 갖춘 독립형 클러스터입니다.

보조 클러스터를 모두 제거하거나 삭제한 후 동일한 방식으로 기본 클러스터를 제거할 수 있습니다. 모든 보조 클러스터를 제거해야 기본 클러스터를 글로벌 클러스터에서 분리 또는 제거할 수 있습니다. 글로벌 클러스터는 리전과 AZs. 이 글로벌 클러스터를 더 이상 사용하지 않을 경우 삭제할 수 있습니다.

Using the AWS CLI

글로벌 클러스터에서 클러스터를 제거하려면 `remove-from-global-cluster` CLI 명령 및 다음 파라미터를 실행합니다.

- `--global-cluster-identifier` - 글로벌 클러스터의 이름(식별자)입니다.
- `--db-cluster-identifier` - 글로벌 클러스터에서 제거할 각 클러스터의 이름입니다.

다음 명령은 글로벌 클러스터에서 보조 클러스터를 제거한 후 기본 클러스터를 제거합니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb --region secondary_region \
  remove-from-global-cluster \
    --db-cluster-identifier secondary_cluster_ARN \
    --global-cluster-identifier global_cluster_id

aws docdb --region primary_region \
  remove-from-global-cluster \
    --db-cluster-identifier primary_cluster_ARN \
    --global-cluster-identifier global_cluster_id
```

글로벌 클러스터의 각 보조 리전에 대해 `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN` 명령을 반복합니다.

Windows의 경우:

```
aws docdb --region secondary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier secondary_cluster_ARN ^
    --global-cluster-identifier global_cluster_id

aws docdb --region primary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier primary_cluster_ARN ^
    --global-cluster-identifier global_cluster_id
```

글로벌 클러스터의 각 보조 리전에 대해 `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN` 명령을 반복합니다.

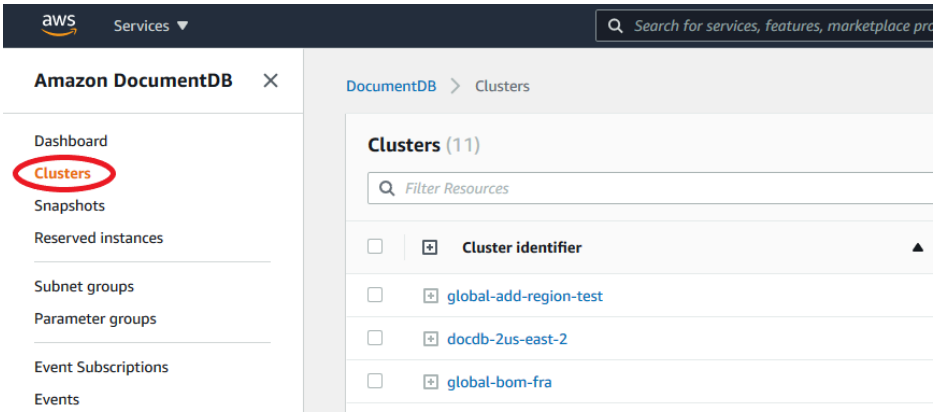
Amazon DocumentDB 글로벌 클러스터에서 클러스터 제거

글로벌 클러스터를 삭제하려면 다음을 수행합니다.

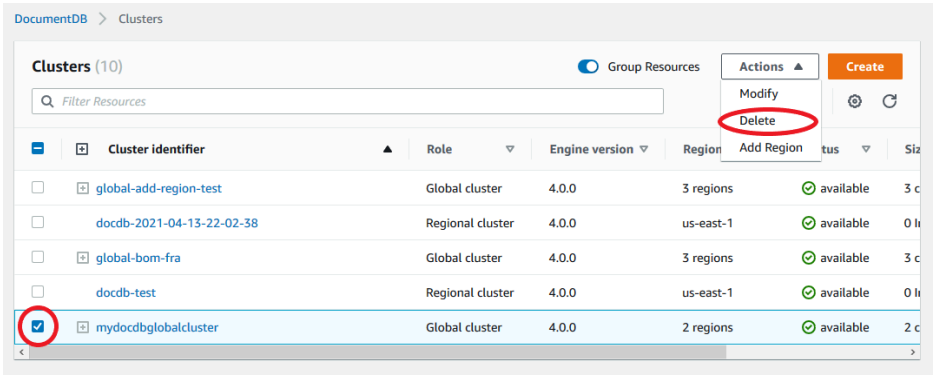
- 글로벌 클러스터에서 모든 보조 클러스터를 제거합니다. 각 클러스터는 독립형 클러스터가 됩니다. 이전 Amazon DocumentDB 글로벌 클러스터에서 클러스터 분리 섹션을 참조하세요.
- 각 독립형 클러스터에서 모든 복제본을 삭제합니다.
- 글로벌 클러스터에서 기본 클러스터를 제거합니다. 이 클러스터는 독립형 클러스터가 됩니다.
- 기본 클러스터에서 먼저 모든 복제본을 삭제한 다음 기본 인스턴스를 삭제합니다. 새로 독립 실행형 클러스터에서 기본 인스턴스를 삭제하면 일반적으로 클러스터와 글로벌 클러스터 모두가 제거됩니다.

Using the AWS Management Console

1. 에 로그인 AWS Management Console 하고 Amazon DocumentDB 콘솔로 이동합니다.
2. 클러스터를 선택하고 삭제할 글로벌 클러스터를 찾습니다.



3. 글로벌 클러스터를 선택한 상태에서 작업 메뉴에서 삭제를 선택합니다.



글로벌 클러스터에서 다른 모든 클러스터가 제거되었는지 확인합니다. 글로벌 클러스터에는 0개의 리전 및 AZs와 0개의 클러스터 크기가 표시되어야 합니다. 글로벌 클러스터에 클러스터가 포함되어 있으면 삭제할 수 없습니다. 먼저 이전 단계인 Amazon DocumentDB 글로벌 클러스터에서 클러스터 분리의 지침을 따라야 합니다.

Using the AWS CLI

글로벌 클러스터를 삭제하려면 다음 예제와 같이 AWS 리전 의 이름과 글로벌 클러스터 식별자를 사용하여 `delete-global-cluster` CLI 명령을 실행합니다.

Linux, macOS, Unix의 경우:

```
aws docdb --region primary_region delete-global-cluster \
   --global-cluster-identifier global_cluster_id
```

Windows의 경우:

```
aws docdb --region primary_region delete-global-cluster ^
   --global-cluster-identifier global_cluster_id
```

보조 리전에서 헤드리스 Amazon DocumentDB 클러스터 생성

Amazon DocumentDB 글로벌 클러스터에는 기본 클러스터 AWS 리전 와 다른에 하나 이상의 보조 클러스터가 필요하지만 보조 클러스터에 헤드리스 구성을 사용할 수 있습니다. 헤드리스 세컨더리 Amazon DocumentDB 클러스터는 인스턴스가 없는 클러스터입니다. 이 유형의 구성은 글로벌 클러스터에 대한 비용을 줄일 수 있습니다. Amazon DocumentDB 클러스터에서는 컴퓨팅과 스토리지가 분리되어 있습니다. 인스턴스가 없으면 컴퓨팅 요금이 청구되지 않고 스토리지 요금만 청구됩니다. 올바르게 설정되면, 헤드리스 보조 스토리지 볼륨이 기본 클러스터와 동기화 상태를 유지합니다.

Amazon DocumentDB 글로벌 클러스터를 생성할 때 보통 때처럼 보조 클러스터를 추가합니다. 그러나 프라이머리 클러스터가 보조 클러스터로 복제되기 시작한 후에는 보조 클러스터에서 읽기 전용 DB 인스턴스를 삭제합니다. 이제이 보조 클러스터에는 더 이상 인스턴스가 없으므로 "헤드리스"로 간주됩니다. 그러나 스토리지 볼륨은 기본 Amazon DocumentDB 클러스터와 동기화 상태를 유지합니다.

> **Important**
>
> 15분 이상 리전 전체의 장애를 허용할 수 있는 경우에만 헤드리스 클러스터를 사용합니다. 헤드리스 보조 클러스터를 사용하여 리전 전체 장애에서 복구하려면 장애 조치 후 새 인스턴스를 생성해야 합니다. 새 인스턴스를 사용할 수 있게 되려면 10~15분이 걸릴 수 있습니다.

글로벌 클러스터에 헤드리스 보조 클러스터를 추가하는 방법

1. 에 로그인 AWS Management Console 하고 Amazon DocumentDB 콘솔을 엽니다.
2. 왼쪽 측면의 탐색 창에서 클러스터를 선택합니다.
3. 보조 클러스터가 필요한 글로벌 클러스터를 선택합니다. 기본 클러스터가 `Available`인지 확인합니다.
4. 작업에서 리전 추가를 선택합니다.
5. 리전 추가 페이지에서 보조 리전을 선택합니다.

> **Note**
>
> 동일한 글로벌 클러스터에 대한 보조 클러스터가 이미 있는 리전은 선택할 수 없습니다. 또한 이는 기본 클러스터와 동일한 리전이 될 수 없습니다.

6. 새 리전에서 보조 클러스터에 대한 나머지 필드를 완료합니다. 이 옵션은 클러스터 인스턴스와 동일한 구성 옵션입니다.
7. 리전을 추가합니다. 글로벌 클러스터에 리전 추가를 완료하면의 목록에 리전`Clusters`이 나타납니다 AWS Management Console.
8. AWS Management Console 또는를 사용하여 계속하기 전에 보조 클러스터와 해당 리더 인스턴스의 상태를 확인합니다 AWS CLI. 다음은 AWS CLI을 사용하는 경우의 샘플 명령입니다.

```
$ aws docdb describe-db-clusters --db-cluster-identifier secondary-cluster-id --
query '*[].[Status]' --output text
```

   새로 추가된 보조 클러스터의 상태가 생성에서 사용 가능으로 변경되려면 몇 분 정도 걸릴 수 있습니다. 클러스터를 사용할 수 있게 되면 리더 인스턴스를 삭제할 수 있습니다.
9. 보조 클러스터에서 리더 인스턴스를 선택한 후 삭제를 선택합니다.
10. 리더 인스턴스를 삭제한 후 보조 클러스터는 글로벌 클러스터의 일부로 남습니다. 데이터와 연결되지 않아야 합니다.

> **Note**
>
> 이러한 중단이 발생하는 경우이 헤드리스 보조 Amazon DocumentDB 클러스터를 사용하여 기본 리전의 예상치 못한 중단으로부터 Amazon DocumentDB 글로벌 클러스터를 수동으로 복구할 수 있습니다.

Amazon DocumentDB 글로벌 클러스터에 연결

글로벌 클러스터에 연결하는 방법은 클러스터에 쓸 것인지 클러스터에서 읽을 것인지에 따라 달라집니다:

- 읽기 전용 요청 또는 쿼리의 경우 자신의 AWS 리전에 있는 Aurora 클러스터 리더 엔드포인트에 연결합니다.

- 데이터 조작 언어(DML) 또는 데이터 정의 언어(DDL) 설명문을 실행하려면 기본 클러스터용 클러스터 엔드포인트에 연결합니다. 이 엔드포인트는 애플리케이션 AWS 리전 과 다른에 있을 수 있습니다.

콘솔에서 글로벌 클러스터를 볼 때 해당 모든 클러스터와 연결된 모든 범용 엔드포인트를 볼 수 있습니다.

글로벌 클러스터에 연결하는 방법은 데이터베이스에 쓸 것인지 데이터베이스에서 읽을 것인지에 따라 달라집니다. 기본 리전에서 제공하려는 DDL, DML 및 읽기 작업의 경우 읽기 기본 설정이 인 복제본 세트 모드에서 클러스터 엔드포인트를 사용하여 기본 클러스터에 연결합니다`secondaryPreferred=true`. 이렇게 하면 쓰기 트래픽이 기본 클러스터의 라이터 인스턴스로 라우팅되고 읽기 트래픽이 기본 클러스터의 복제본 인스턴스로 라우팅됩니다.

리전 간 읽기 전용 트래픽의 경우 복제본 세트 모드에서 클러스터 엔드포인트를 사용하여 보조 클러스터 중 하나에 연결합니다. 모든 인스턴스는 읽기 전용 복제본 인스턴스이므로 이외의 읽기 기본 설정을 지정합니다`primary`(예: , `secondary` `secondaryPreferred`또는 `nearest`). 지연 시간을 최소화하려면 자신의 리전이나 가장 가까운 리전의 리더 엔드포인트를 선택합니다.

Amazon DocumentDB 글로벌 클러스터 모니터링

Amazon DocumentDB(MongoDB 호환)는 CloudWatch와 통합되므로 클러스터에 대한 운영 지표를 수집하고 분석할 수 있습니다. CloudWatch 콘솔, Amazon DocumentDB 콘솔, AWS Command Line Interface (AWS CLI) 또는 CloudWatch API를 사용하여 이러한 지표를 모니터링할 수 있습니다.

글로벌 클러스터를 모니터링하려면 다음 CloudWatch 지표를 사용합니다.

지표	설명
`GlobalClusterReplicatedWriteIO`	5분 간격으로 AWS 리전보고된 기본의 클러스터 볼륨에서 보조 AWS 리전 의 클러스터 볼륨으로 복제된 청구된 쓰기 I/O 작업의 평균 수입니다. 각 보조 리전`ReplicatedWriteIOs` 에 복제된 수는 기본 리전에서 `VolumeWriteIOPs` 수행한 리전 내 수와 동일합니다.
`GlobalClusterDataTransferBytes`	기본 클러스터에서 AWS 리전 보조 클러스터로 전송되는 데이터의 양은 바이트 단위로 AWS 리전측정됩니다.

지표	설명
GlobalClusterReplicationLag	기본 클러스터의에서 보조 클러스터의 로 변경 이벤트를 복제 AWS 리전 할 때 밀리초 단위의 지연 시간 AWS 리전

이러한 지표를 보는 방법에 대한 자세한 내용은 CloudWatch 데이터 보기를 참조하세요.

재해 복구 및 Amazon DocumentDB 글로벌 클러스터

주제

- Amazon DocumentDB 글로벌 클러스터에 대한 관리형 장애 조치 수행
- Amazon DocumentDB 글로벌 클러스터에 대한 수동 장애 조치 수행
- Amazon DocumentDB 글로벌 클러스터에 대한 전환 수행
- 글로벌 클러스터 전환 또는 장애 조치 차단 해제
- Amazon DocumentDB 글로벌 클러스터에 대한 RPOs 관리

글로벌 클러스터를 사용하면 리전 장애와 같은 재해로부터 신속하게 복구할 수 있습니다. 재해 복구는 일반적으로 RTO 및 RPO 값을 사용하여 측정됩니다.

- (Recovery Time Objective(RTO) – 재해 발생 후 시스템이 정상 작동 상태로 돌아가는 데 걸리는 시간입니다. 즉 RTO는 가동 중지 시간을 측정합니다. 글로벌 클러스터의 경우 몇 분 안에 RTO를 수행합니다.
- Recovery Point Objective(RPO) – 손실될 수 있는 데이터의 양입니다(시간으로 측정). 글로벌 클러스터의 경우 일반적으로 RPO는 초 단위로 측정됩니다.
- 계획되지 않은 중단으로부터 복구하기 위해 글로벌 클러스터의 보조 중 하나로 리전 간 장애 조치를 수행할 수 있습니다. 글로벌 클러스터에 여러 보조 리전이 있는 경우 기본으로 승격하려는 모든 보조 리전을 분리해야 합니다. 그런 다음 이러한 보조 리전 중 하나를 새 기본 리전으로 승격합니다 AWS 리전. 마지막으로 다른 각 보조 리전에서 새 클러스터를 생성하고 해당 클러스터를 글로벌 클러스터에 연결합니다.

Amazon DocumentDB 글로벌 클러스터에 대한 관리형 장애 조치 수행

이 접근 방식은 실제 리전별 재해 이벤트 또는 전체 서비스 수준 중단이 발생하는 경우 비즈니스 연속성을 유지하기 위해 개발되었습니다.

관리형 장애 조치 중에, Amazon DocumentDB 글로벌 클러스터의 기존 복제 토폴로지가 유지되는 동안 기본 클러스터는 선택한 보조 리전으로 장애 조치됩니다. 선택한 보조 클러스터에서는 읽기 전용 노드 중 하나가 전체 라이터 상태로 승격됩니다. 이 단계를 통해 클러스터는 기본 클러스터의 역할을 맡게 됩니다. 클러스터가 새 역할을 맡는 동안에는 데이터베이스를 일시적으로 사용할 수 없습니다. 이 보조 클러스터가 새 기본 클러스터가 되면 기존 기본 클러스터에서 선택한 보조 클러스터로 복제되지 않은 데이터는 누락될 수 있습니다. 이전 기본 볼륨은 스냅샷에 복제되지 않은 데이터가 보존되도록 새 기본 볼륨과 동기화하기 전에 스냅샷을 찍으려고 최선의 노력을 기울입니다.

> **Note**
>
> 기본 및 모든 보조 클러스터에 있는 엔진 버전이 동일한 경우 Amazon DocumentDB 글로벌 클러스터에서 관리형 리전 간 클러스터 장애 조치만 수행할 수 있습니다. 엔진 버전이 호환되지 않는 경우 Amazon DocumentDB 글로벌 클러스터에 대한 수동 장애 조치 수행에 나온 단계에 따라 수동으로 장애 조치를 수행할 수 있습니다.
>
> 리전의 엔진 버전이 일치하지 않으면 장애 조치가 차단됩니다. 보류 중인 업그레이드가 있는지 확인하고 이를 적용하여 모든 리전의 엔진 버전이 일치하고 글로벌 클러스터 장애 조치가 차단 해제되었는지 확인합니다. 자세한 내용은 글로벌 클러스터 전환 또는 장애 조치 차단 해제 단원을 참조하십시오.

데이터 손실을 최소화하려면이 기능을 사용하기 전에 다음을 수행합니다.

- 애플리케이션을 오프라인으로 전환하여 쓰기가 Amazon DocumentDB 글로벌 클러스터의 기본 클러스터로 전송되는 경우를 방지합니다.
- 모든 Amazon DocumentDB 보조 클러스터의 지연 시간을 확인합니다. 복제 지연이 가장 적게 소요되는 보조 리전을 선택하면 현재 장애가 발생한 기본 리전에서 데이터 손실을 최소화할 수 있습니다. Amazon CloudWatch에서 `GlobalClusterReplicationLag` 지표를 확인하여 글로벌 클러스터의 모든 Amazon DocumentDB 보조 클러스터에 대한 지연 시간을 확인합니다. 이 지표를 통해 보조 클러스터가 기본 DB 클러스터에 비해 얼마나 뒤처져 있는지(밀리초 단위) 알 수 있습니다.

  Amazon DocumentDB용 CloudWatch 지표에 대한 자세한 내용은 Amazon DocumentDB 지표 섹션을 참조하세요.

관리형 장애 조치 중에, 선택한 보조 DB 클러스터가 기본 클러스터처럼 새 역할로 승격됩니다. 하지만 기본 클러스터의 다양한 구성 옵션은 상속되지 않습니다. 구성이 일치하지 않으면 성능 문제, 워크로드 비호환성, 기타 비정상적인 동작이 발생할 수 있습니다. 이러한 문제를 방지하려면 다음에 대한 Amazon DocumentDB 글로벌 클러스터 간의 차이를 해결하세요.

- 필요한 경우 새 기본 클러스터에 대한 Amazon DocumentDB 클러스터 파라미터 그룹 구성 - Amazon DocumentDB 글로벌 클러스터의 각 클러스터에 대해 Amazon DocumentDB 클러스터 파라미터 그룹을 독립적으로 구성할 수 있습니다. 즉, 보조 클러스터가 기본 클러스터 역할을 맡도록 승격되면 보조 클러스터의 파라미터 그룹이 기본 클러스터와 다르게 구성될 수 있습니다. 그런 경우, 승격된 보조 클러스터의 파라미터 그룹을 기본 클러스터의 설정에 맞게 수정합니다. 자세한 방법은 Amazon DocumentDB 클러스터 파라미터 그룹 수정을 참조하세요.
- Amazon CloudWatch Events 및 경보 등의 모니터링 도구 및 옵션 구성 — 승격된 클러스터를 글로벌 클러스터에 필요한 것과 동일한 로깅 기능, 경보 등으로 구성합니다. 파라미터 그룹과 마찬가지로, 이러한 기능에 대한 구성은 장애 조치 프로세스 중에 기본 클러스터에서 상속되지 않습니다. 복제 지연과 같은 일부 CloudWatch 지표는 보조 리전에서만 사용할 수 있습니다. 따라서 장애 조치로 인해 해당 지표를 보고 경보를 설정하는 방법이 달라짐에 따라 사전 정의된 대시보드를 변경해야 할 수도 있습니다. Amazon DocumentDB 클러스터 및 모니터링에 대한 자세한 내용은 Amazon DocumentDB에서 모니터링 및 로깅 섹션을 참조하세요.

일반적으로 선택한 보조 클러스터는 1분 이내에 기본 역할을 맡게 됩니다. 새 기본 리전의 라이터 노드를 사용할 수 있게 되면 애플리케이션을 노드에 연결하고 워크로드를 재개할 수 있습니다. Amazon DocumentDB에서 새 기본 클러스터가 승격되고 난 후, 모든 추가 보조 리전 클러스터가 자동으로 재구축됩니다.

Amazon DocumentDB 글로벌 클러스터는 비동기식 복제를 사용하므로 보조 리전마다 복제 지연 시간이 다를 수 있습니다. Amazon DocumentDB는 이러한 보조 리전을 재구축하여 새로운 기본 리전 클러스터와 정확히 동일한 시점의 데이터를 확보하도록 지원합니다. 전체 재구축 작업에 소요되는 시간은 스토리지 볼륨의 크기와 리전 간 거리에 따라 몇 분에서 몇 시간이 걸릴 수 있습니다. 보조 리전 클러스터의 재구축이 새 기본 리전에서 완료되면 읽기 권한으로 액세스할 수 있습니다. 새 기본 라이터가 승격되어 사용 가능해지면 새 기본 리전의 클러스터에서 Amazon DocumentDB 글로벌 클러스터의 읽기 및 쓰기 작업을 처리할 수 있습니다.

글로벌 클러스터의 기존 토폴로지를 복원하기 위해 Amazon DocumentDB는 기존 기본 리전의 가용성을 모니터링합니다. 해당 리전이 정상이고 다시 사용 가능한 상태가 되면 Amazon DocumentDB는 자동으로 글로벌 클러스터에 보조 리전으로 다시 추가합니다. Amazon DocumentDB는 장애가 발생하는 시점에 기존 스토리지 볼륨의 스냅샷을 만들고자 시도한 후, 기존 기본 리전에 새 스토리지 볼륨을 생성합니다. 이렇게 하면 누락된 데이터를 복구하는 데 사용할 수 있습니다. 이 작업이 성공하면 Amazon DocumentDB는 AWS Management Console의 스냅샷 섹션에 'rds:docdb-unplanned-global-failover-name-of-old-primary-DB-cluster-timestamp'라는 이름으로 이 스냅샷을 배치합니다. 또한 이 스냅샷은 `DescribeDBClusterSnapshots` API 작업에서 반환된 정보에 나열되어 있으니 해당 위치에서 확인할 수 있습니다.

> **Note**
>
> 기존 스토리지 볼륨의 스냅샷은 시스템 스냅샷으로, 기존 기본 클러스터에 구성된 백업 보존 기간이 적용됩니다. 보존 기간 이후에도 이 스냅샷을 보존하려면 스냅샷을 복사하여 수동 스냅샷으로 저장할 수 있습니다. 요금 등 스냅샷 복사에 대해 자세히 알아보려면 클러스터 스냅샷 복사 단원을 참조하세요.

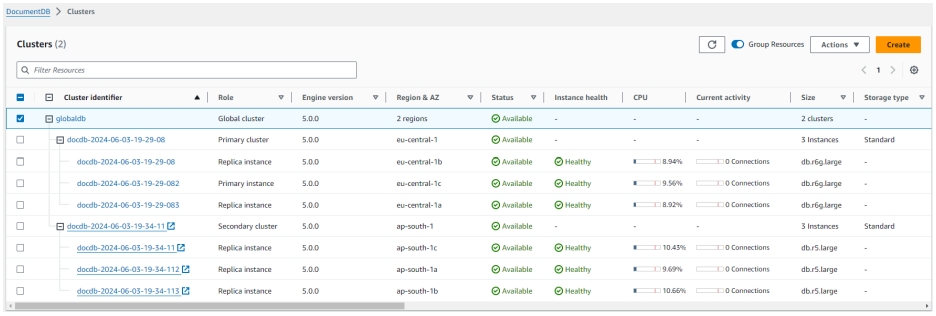
기존 토폴로지가 복원된 후 비즈니스 및 워크로드에 가장 적합한 시점에 전환 작업을 수행하여 글로벌 클러스터를 기존의 기본 리전으로 장애 복구할 수 있습니다. 이렇게 하려면 Amazon DocumentDB 글로벌 클러스터에 대한 전환 수행 단원의 절차를 따르세요.

AWS CLI, 또는 Amazon DocumentDB API를 사용하여 Amazon DocumentDB 글로벌 클러스터 AWS Management Console를 장애 조치할 수 있습니다.

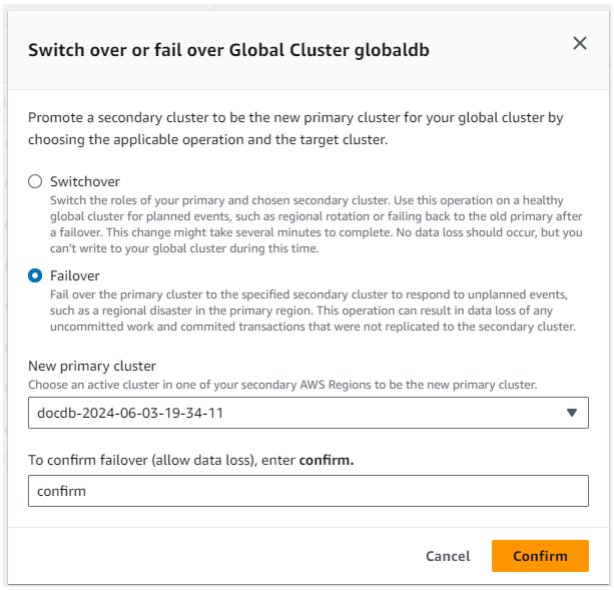
Using the AWS Management Console

Amazon DocumentDB 글로벌 클러스터에서 관리형 장애 조치 수행

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 장애 조치하려는 Amazon DocumentDB 글로벌 클러스터를 찾아 선택합니다.



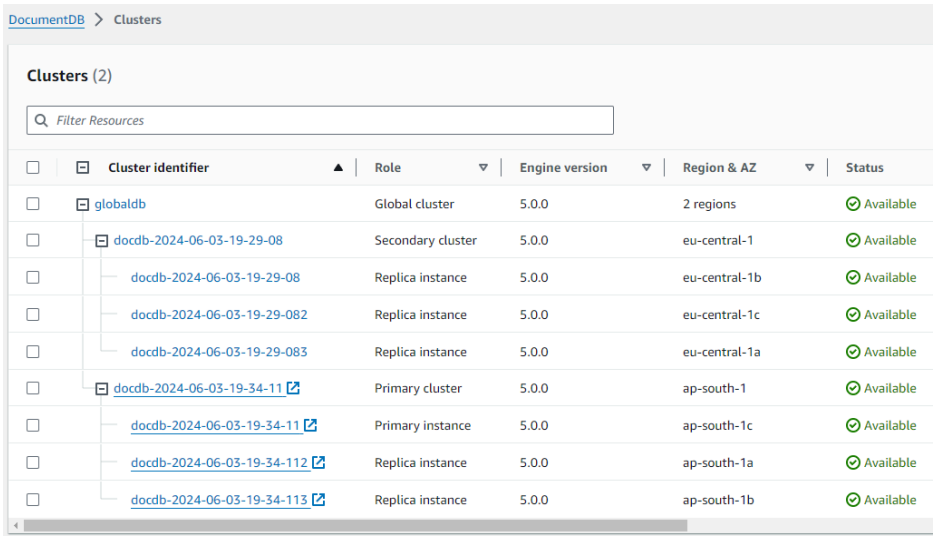
4. 작업 메뉴에서 전환 또는 장애 조치를 선택합니다.
5. 나타나는 대화 상자에서 장애 조치를 선택한 다음 새 기본 클러스터 필드 드롭다운 목록에서 보조 클러스터를 선택합니다.



6. 마지막 필드에 '확인'을 입력합니다. 그 다음 확인을 선택합니다.

   기본 클러스터의 상태가 '장애 조치 중'으로 변경됩니다. 이 조건은 약 1분이 소요됩니다. 이 시간 동안 새 기본 클러스터의 상태는 '수정 중...'으로 표시됩니다. 새 기본이 승격되면 '사용 가능'이 표시되고 읽기 및 쓰기 트랜잭션을 제공할 수 있습니다. 이전 기본 리전을 포함한 보조 리전에는 새 기본 리전과 다시 동기화되는 동안 "Resyncing..."이 표시됩니다. 새 기본과 마찬가지로 상태가 '사용 가능'으로 변경된 후에만 트랜잭션을 제공할 수 있습니다.

7. 완료되면 원래 기본 클러스터가 보조 클러스터가 됩니다. 선택한 보조 클러스터가 기본 클러스터가 됩니다.



Using the AWS CLI

Amazon DocumentDB 글로벌 클러스터에서 관리형 장애 조치 수행

`failover-global-cluster` CLI 명령을 실행하여 Amazon DocumentDB 글로벌 클러스터를 장애 조치합니다. 명령을 사용하여 다음 옵션에 대한 값을 전달합니다.

- `--region`
- `--global-cluster-identifier`
- `--target-db-cluster-identifier`
- `--allow-data-loss`

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

Linux, macOS, Unix의 경우:

```
aws docdb failover-global-cluster \
   --region region_of_selected_secondary \
   --global-cluster-identifier global_cluster_id \
   --target-db-cluster-identifier arn_of_secondary_to_promote \
   --allow-data-loss
```

Windows의 경우:

```
aws docdb failover-global-cluster ^
   --region region_of_selected_secondary ^
   --global-cluster-identifier global_cluster_id ^
   --target-db-cluster-identifier arn_of_secondary_to_promote ^
   --allow-data-loss
```

Amazon DocumentDB 글로벌 클러스터에 대한 수동 장애 조치 수행

한의 전체 클러스터를 사용할 수 없게 AWS 리전 되면 글로벌 클러스터의 다른 클러스터를 읽기/쓰기 기능으로 승격할 수 있습니다.

다른 AWS 리전 의 클러스터가 기본 클러스터로 더 적합한 경우 글로벌 클러스터 장애 조치 메커니즘을 수동으로 활성화할 수 있습니다. 예를 들어 보조 클러스터 중 하나의 용량을 늘린 후 이 클러스터를 기본 클러스터로 승격할 수 있습니다. 또는 간의 활동 균형이 변경될 AWS 리전 수 있으므로 기본 클러스터를 다른 로 전환하면 쓰기 작업의 지연 시간이 줄어들 AWS 리전 수 있습니다.

다음 절차에서는 Amazon DocumentDB 글로벌 클러스터의 보조 클러스터 중 하나를 승격하기 위해 수행할 작업을 설명합니다.

보조 클러스터를 승격하려면:

1. 중단 AWS 리전 시의 기본 클러스터에 대한 DML 문 및 기타 쓰기 작업 실행을 중지합니다.
2. 보조 클러스터에서 새 기본 클러스터로 사용할 클러스터를 식별 AWS 리전 합니다. 글로벌 클러스터 AWS 리전 에 보조 클러스터가 두 개(또는 그 이상) 있는 경우 지연 시간이 가장 짧은 보조 클러스터를 선택합니다.
3. 선택한 보조 클러스터를 글로벌 클러스터에서 분리합니다.

   글로벌 클러스터에서 보조 클러스터를 제거하면 기본 클러스터에서 이 보조 클러스터로의 복제가 즉시 중지되고 전체 읽기/쓰기 기능을 갖춘 독립 실행형 프로비저닝 클러스터로 승격됩니다. 중단이 발생한 리전의 기본 클러스터와 연결된 다른 보조 클러스터는 계속 사용할 수 있으며 애플리케이션의 호출을 수락할 수 있습니다. 클러스터는 리소스도 소비합니다. 글로벌 클러스터를 다시 생성하는 중이므로 분할 브레인 및 기타 문제를 방지하려면 다음 단계에서 새 글로벌 클러스터를 생성하기 전에 다른 보조 클러스터를 제거하십시오.

   분리 단계에 대한 자세한 내용은 Amazon DocumentDB 글로벌 클러스터에서 클러스터 분리 섹션을 참조하십시오.
4. 이 클러스터는 다음 단계에서 영역을 추가하기 시작하면 새 글로벌 클러스터의 기본 클러스터가 됩니다.
5. 클러스터에 AWS 리전 를 추가합니다. 이렇게 하면 기본 클러스터에서 보조 클러스터로의 복제 프로세스가 시작됩니다.
6. 필요에 AWS 리전 따라를 추가하여 애플리케이션을 지원하는 데 필요한 토폴로지를 다시 생성합니다. 글로벌 클러스터에 있는 클러스터 간의 데이터 불일치(분할 뇌 문제)를 방지하기 위해 이러한 변경 전, 변경 중 및 변경 후에 애플리케이션 쓰기가 올바른 클러스터로 전송되는지 확인합니다.
7. 운영 중단이 해결되고 원본 AWS 리전 을 다시 기본 클러스터로 할당할 준비가 되면 동일한 단계를 반대로 수행합니다.
8. 글로벌 데이터베이스에서 보조 클러스터 중 하나를 제거합니다. 이렇게 하면 읽기/쓰기 트래픽을 처리할 수 있습니다.
9. 모든 쓰기 트래픽을 원래 AWS 리전의 기본 클러스터로 리디렉션합니다.
10. AWS 리전 를 추가하여 이전 AWS 리전 과 동일한에 하나 이상의 보조 클러스터를 설정합니다.

Amazon DocumentDB 글로벌 클러스터는 AWS SDKs를 사용하여 관리할 수 있으므로 재해 복구 및 비즈니스 연속성 계획 사용 사례에 대한 글로벌 클러스터 장애 조치 프로세스를 자동화하는 솔루션을 생성할 수 있습니다. 이러한 솔루션 중 하나는 Apache 2.0 라이센스에 따라 고객이 사용할 수 있도록 제공되며 여기에 있는 툴 저장소에서 액세스할 수 있습니다. 이 솔루션은 엔드포인트 관리를 위해 Amazon Route 53를 활용하고 적절한 이벤트를 기반으로 트리거할 수 있는 AWS Lambda 함수를 제공합니다.

Amazon DocumentDB 글로벌 클러스터에 대한 전환 수행

전환을 사용하면 기본 클러스터의 리전을 정기적으로 변경할 수 있습니다. 이 접근 방식은 운영 유지 관리 및 기타 계획된 운영 절차 등 제어된 시나리오를 대상으로 개발되었습니다.

전환은 일반적으로 3가지의 경우에 사용됩니다.

- 특정 산업에서 필요로 하는 '리전별 순환' 요구 사항을 살펴봅니다. 예를 들어 금융 서비스 규정에 따라 재해 복구 절차가 정기적으로 실행되도록 보장하려면 Tier-0 시스템을 몇 개월 동안 다른 리전으로 전환해야 할 수 있습니다.
- 여러 리전의 'follow-the-sun' 애플리케이션을 살펴봅니다. 각기 다른 시간대 전반에서 업무 시간을 기준으로 여러 리전별로 지연 시간이 짧은 쓰기 기능을 제공하고자 하는 기업을 예로 들 수 있습니다.
- 데이터 손실 제로의 방법이며, 장애 조치 후 기존의 기본 리전으로 페일백하는 데 유용합니다.

> **Note**
>
> 전환은 정상 Amazon DocumentDB 글로벌 클러스터에서 사용하도록 설계되었습니다. 예상치 못한 중단이 발생한 상태에서 복구하려면 Amazon DocumentDB 글로벌 클러스터에 대한 수동 장애 조치 수행 단원에 나온 적절한 절차를 따르세요.
> 전환을 수행하려면 모든 보조 리전이 기본 리전과 정확히 동일한 엔진 버전을 실행해야 합니다. 리전의 엔진 버전이 일치하지 않으면 전환이 차단됩니다. 보류 중인 업그레이드가 있는지 확인하고 이를 적용하여 모든 리전의 엔진 버전이 일치하고 글로벌 클러스터 전환이 차단 해제되었는지 확인합니다. 자세한 내용은 글로벌 클러스터 전환 또는 장애 조치 차단 해제 단원을 참조하십시오.

전환 중에, Amazon DocumentDB에서는 글로벌 클러스터의 기존 복제 토폴로지가 유지되는 동안 기본 클러스터를 선택한 보조 리전으로 전환합니다. Amazon DocumentDB는 모든 보조 리전 클러스터가 기본 리전 클러스터와 완전히 동기화될 때까지 기다린 다음, 전환 프로세스를 시작합니다. 그러면 기본 리전의 DB 클러스터가 읽기 전용 상태가 되고, 선택한 보조 클러스터는 읽기 전용 노드 중 하나를 전체

라이터 상태로 승격시킵니다. 이 노드를 라이터로 승격시키면 보조 클러스터가 기본 클러스터의 역할을 맡을 수 있습니다. 프로세스 시작 시 모든 보조 클러스터가 기본 클러스터와 동기화되었으므로, 새로운 기본 클러스터는 데이터 손실 없이 Amazon DocumentDB 글로벌 클러스터에 대한 작업을 계속합니다. 기본 클러스터와 선택한 보조 클러스터가 새 역할을 맡으므로 데이터베이스를 잠시 사용할 수 없습니다.

애플리케이션 가용성을 최적화하려면이 기능을 사용하기 전에 다음을 수행합니다.

- 사용량이 적은 시간이나 기본 클러스터에 대한 쓰기가 최소인 시간에 이 작업을 수행합니다.
- 애플리케이션을 오프라인으로 전환하여 쓰기가 Amazon DocumentDB 글로벌 클러스터의 기본 클러스터로 전송되는 경우를 방지합니다.
- Amazon CloudWatch에서 `GlobalClusterReplicationLag` 지표를 확인하여 글로벌 클러스터의 모든 Amazon DocumentDB 보조 클러스터에 대한 지연 시간을 확인합니다. 이 지표는 보조 DB 클러스터가 기본 DB 클러스터에 비해 얼마나 뒤처져 있는지(밀리초 단위)를 보여 줍니다. 이 값은 Amazon DocumentDB가 전환을 완료하는 데 걸리는 시간에 직접적으로 비례합니다. 따라서 지연 값이 클수록 전환 시간이 더 오래 걸립니다.

  Amazon DocumentDB용 CloudWatch 지표에 대한 자세한 내용은 Amazon DocumentDB 지표 섹션을 참조하세요.

전환 중에, 선택한 보조 DB 클러스터가 기본 DB 클러스터처럼 새 역할로 승격됩니다. 하지만 기본 DB 클러스터의 다양한 구성 옵션은 상속되지 않습니다. 구성이 일치하지 않으면 성능 문제, 워크로드 비호환성, 기타 비정상적인 동작이 발생할 수 있습니다. 이러한 문제를 방지하려면 다음에 대한 Amazon DocumentDB 글로벌 클러스터 간의 차이를 해결하세요.

- 필요한 경우 새 기본에 대한 Amazon DocumentDB 클러스터 파라미터 그룹 구성 — Amazon DocumentDB 글로벌 클러스터의 각 클러스터에 대해 Amazon DocumentDB 클러스터 파라미터 그룹을 개별적으로 구성할 수 있습니다. 즉, 보조 DB 클러스터가 기본 클러스터 역할을 맡도록 승격되면 보조 클러스터의 파라미터 그룹이 기본 클러스터와 다르게 구성될 수 있습니다. 그런 경우, 승격된 보조 DB 클러스터의 파라미터 그룹을 기본 클러스터의 설정에 맞게 수정합니다. 자세한 방법은 Amazon DocumentDB 클러스터 파라미터 그룹 관리 단원을 참조하십시오.
- Amazon CloudWatch Events 및 경보 등의 모니터링 도구 및 옵션 구성 — 승격된 클러스터를 글로벌 클러스터에 필요한 것과 동일한 로깅 기능, 경보 등으로 구성합니다. 파라미터 그룹과 마찬가지로, 이러한 기능에 대한 구성은 전환 프로세스 중에 기본 클러스터에서 상속되지 않습니다. 복제 지연과 같은 일부 CloudWatch 지표는 기본 리전에서만 사용할 수 있습니다. 따라서 전환으로 인해 해당 지표를 보고 경보를 설정하는 방법이 달라짐에 따라 사전 정의된 대시보드를 변경해야 할 수도 있습니다. 자세한 내용은 Amazon DocumentDB에서 모니터링 및 로깅 단원을 참조하십시오.

> **Note**
>
> 일반적으로 역할 전환에는 최대 몇 분이 걸릴 수 있습니다.

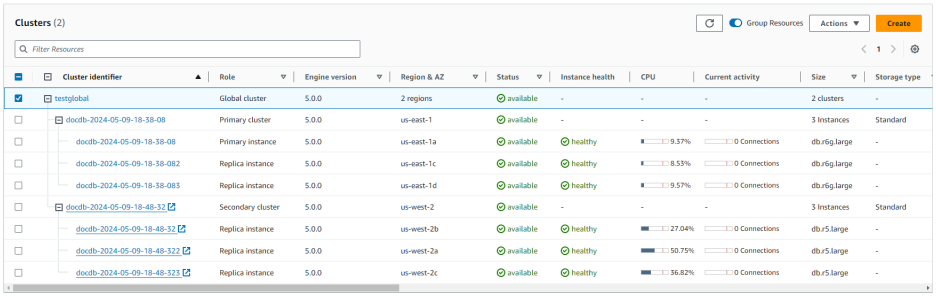
전환 프로세스가 완료되면 승격된 Amazon DocumentDB 클러스터에서 글로벌 클러스터에 대한 쓰기 작업을 처리할 수 있습니다.

AWS Management Console 또는 AWS CLI를 사용하여 Amazon DocumentDB 글로벌 클러스터를 전환할 수 있습니다.

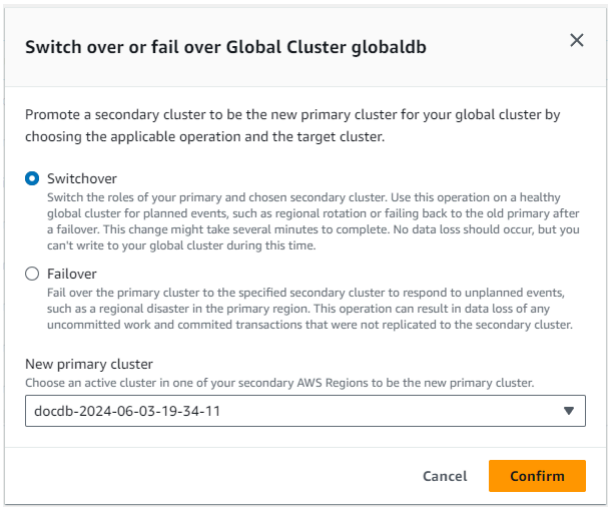
Using the AWS Management Console

Amazon DocumentDB 글로벌 클러스터에서 전환 수행

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 전환하려는 Amazon DocumentDB 글로벌 클러스터를 찾아 선택합니다.



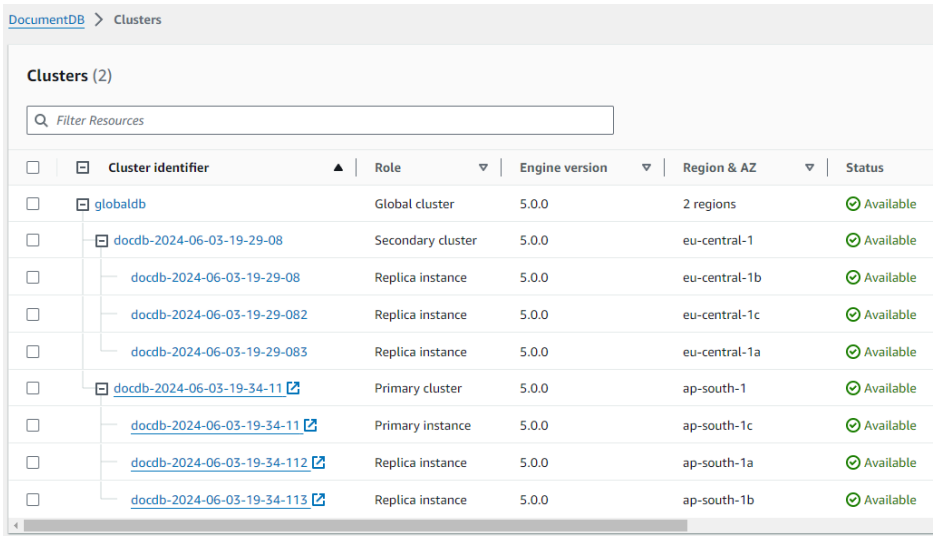
4. 작업 메뉴에서 전환 또는 장애 조치를 선택합니다.
5. 나타나는 대화 상자에서 전환을 선택한 다음 새 기본 클러스터 필드 드롭다운 목록에서 보조 클러스터를 선택합니다.



6. 확인을 선택합니다.

   기본 클러스터의 상태가 '전환'으로 변경됩니다. 이 조건은 약 3분이 소요됩니다. 이 시간 동안 모든 리전 클러스터의 상태는 '수정 중...'으로 표시됩니다. 리전이 동기화되고 새 기본 필드가 승격되면 모든 상태 필드에 대해 "사용 가능"이 표시되고 트랜잭션을 제공할 수 있습니다.

7. 완료되면 원래 기본 클러스터가 보조 클러스터가 됩니다. 선택한 보조 클러스터가 기본 클러스터가 됩니다.



Using the AWS CLI

Amazon DocumentDB 글로벌 클러스터에서 전환 수행

`switchover-global-cluster` CLI 명령을 실행하여 Amazon DocumentDB 글로벌 클러스터를 전환합니다. 명령을 사용하여 다음 옵션에 대한 값을 전달합니다.

- `--region`
- `--global-cluster-identifier`
- `--target-db-cluster-identifier`

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

Linux, macOS, Unix의 경우:

```
aws docdb switchover-global-cluster \
   --region region_of_primary \
   --global-cluster-identifier global_cluster_id \
   --target-db-cluster-identifier arn_of_secondary_to_promote
```

Windows의 경우:

```
aws docdb switchover-global-cluster ^
   --region region_of_primary ^
   --global-cluster-identifier global_cluster_id ^
   --target-db-cluster-identifier arn_of_secondary_to_promote
```

글로벌 클러스터 전환 또는 장애 조치 차단 해제

다른 엔진 버전의 리전

글로벌 클러스터에 있는 모든 리전 클러스터의 엔진 버전이 동일하지 않은 경우 글로벌 클러스터 전환 및 장애 조치가 차단됩니다. 버전이 일치하지 않으면 전환 또는 장애 조치를 호출할 때이 오류가 표시될 수 있습니다. 지정된 대상 DB 클러스터가 소스 DB 클러스터와 다른 패치 수준의 엔진 버전을 실행하고 있습니다. 정기적으로 최신 엔진 버전을 적용하여 글로벌 클러스터를 정상 상태로 유지합니다.

이 오류를 해결하려면 먼저 모든 보조 리전을 업데이트한 다음 보류 중인 유지 관리 작업 항목을 적용하여 기본 리전을 동일한 엔진 버전으로 업데이트합니다. 보류 중인 유지 관리 작업 항목을 보고 필요한 변경 사항을 적용하여 문제를 해결하려면 다음 탭 중 하나의 지침을 수행합니다.

Using the AWS Management Console

글로벌 클러스터 전환 또는 장애 조치를 차단 해제하려면 클러스터에 대해 보류 중인 유지 관리 작업이 있는지 확인하고 해당 작업을 적용해야 합니다. 다음 단계에 따라 유지 관리 작업을 보고 적용합니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터 테이블의 클러스터 식별자 열에서 글로벌 클러스터를 찾습니다. 글로벌 클러스터에서 지정된 글로벌 클러스터의 각 보조 클러스터와 기본 클러스터를 기록하고 각각에 대해 다음 단계를 수행합니다.
4. 각 보조 클러스터의 경우:
   a. 클러스터에 업데이트가 있는 경우 유지 관리 열에 사용 가능, 필수 또는 다음 창의 단어로 표시됩니다.
   b. 조치를 취하려면 클러스터를 선택하여 세부 정보를 표시한 후 유지 관리 및 백업을 선택하십시오. 그러면 대기 중인 유지 관리 항목이 표시됩니다.
   c. 설명에서 '새 유지 관리 업데이트를 사용할 수 있음'을 나타내는 경우 해당 업데이트를 선택한 다음 지금 적용을 선택합니다.
5. 기본 클러스터의 경우:
   a. 클러스터에 업데이트가 있는 경우 유지 관리 열에 사용 가능, 필수 또는 다음 창의 단어로 표시됩니다.
   b. 조치를 취하려면 클러스터를 선택하여 세부 정보를 표시한 후 유지 관리 및 백업을 선택하십시오. 그러면 대기 중인 유지 관리 항목이 표시됩니다.
   c. 설명에서 '새 유지 관리 업데이트를 사용할 수 있음'을 나타내는 경우 해당 업데이트를 선택한 다음 지금 적용을 선택합니다.

Using the AWS CLI

글로벌 클러스터 전환 또는 장애 조치를 차단 해제하려면 클러스터에 대해 보류 중인 유지 관리 작업이 있는지 확인하고 해당 작업을 적용해야 합니다. 다음 단계에 따라 보조 클러스터에서 먼저 유지 관리 작업을 보고 적용한 다음 글로벌 클러스터의 기본 클러스터에서 유지 관리 작업을 보고 적용합니다.

1. 각 보조 리전의 리전 클러스터에서 먼저 다음을 실행한 다음 기본 리전 리전 클러스터에서 다음을 실행합니다.
2. `--resource-identifier` 옵션과 함께 `describe-pending-maintenance-actions` CLI 명령을 실행하여 Amazon DocumentDB 리전 클러스터에 사용할 수 있는 유지 관리 작업이 있는지 확인합니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

Linux, macOS, Unix의 경우:

```
aws docdb describe-pending-maintenance-action \
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15
```

Windows의 경우:

```
aws docdb describe-pending-maintenance-action ^
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15
```

다음과 비슷한 결과가 반환됩니다.

```
{
    "PendingMaintenanceActions": [
        {
            "ResourceIdentifier": "arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "system-update",
                    "CurrentApplyDate": "2025-04-11T03:01:00Z",
                    "Description": "db-version-upgrade",
                    "ForcedApplyDate": "2025-06-18T03:01:00Z",
                    "AutoAppliedAfterDate": "2025-05-11T03:01:00Z"
                    "OptInStatus": "pending"
                }
            ]
        }
    ]
}
```

3. 유지 관리 작업이 필요한 경우 다음 옵션을 사용하여 apply-pending-maintenance-action CLI 명령을 실행합니다.

- `--resource-identifier`
- `--apply-action`

- `--opt-in-type`
- `--region`

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

Linux, macOS, Unix의 경우:

```
aws docdb apply-pending-maintenance-action \
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15 \
   --apply-action system-update \
   --opt-in-type immediate \
   --region us-east-1
```

Windows의 경우:

```
aws docdb apply-pending-maintenance-action ^
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15 ^
   --apply-action system-update ^
   --opt-in-type immediate ^
   --region us-east-1
```

4. 유지 관리 작업이 완료되면 describe-pending-maintenance-actions 명령을 다시 실행하여 클러스터에 보류 중인 다른 작업이 없는지 확인합니다.

   원하는 결과는 다음과 같습니다.

```
{
    "PendingMaintenanceActions": []
}
```

Using the Amazon DocumentDB API

글로벌 클러스터 전환 또는 장애 조치를 차단 해제하려면 클러스터에 대해 보류 중인 유지 관리 작업이 있는지 확인하고 해당 작업을 적용해야 합니다. 다음 API를 사용하여 유지 관리 작업을 보고 적용합니다.

1. 각 보조 리전의 리전 클러스터에서 먼저 다음을 실행한 다음 기본 리전 리전 클러스터에서 다음을 실행합니다.
2. PendingMaintenanceAction API를 호출하여 Amazon DocumentDB 글로벌 클러스터에 사용할 수 있는 유지 관리 작업이 있는지 확인합니다.
3. ApplyPendingMaintenanceAction API를 호출하여 변경 사항을 적용합니다.

대상 클러스터에서 보류 중인 변경 사항

대상 보조 클러스터에 다음 유지 관리 기간에 요청한 수정 또는 유지 관리 작업과 같이 아직 적용되지 않은 예약된 변경이 있는 경우에도 전환 및 장애 조치가 차단됩니다. 이 경우 전환 또는 장애 조치를 호출할 때이 오류가 표시될 수 있습니다. ARN *arn:aws:rds:us-east-1:001234567890:cluster:docdb-2025-03-27-19-21-15*를 사용하여 클러스터로 장애 조치할 수 없습니다. 수정 중이거나 수정이 보류 중이기 때문입니다. 클러스터를 사용할 수 있게 되면 다시 시도하세요. 작업의 차단을 해제하려면 대상 클러스터에서 예약된 변경 사항을 제거합니다. 단계는 변경 유형에 따라 다릅니다. 예약된 변경 사항이 제거되고 클러스터 상태가 로 돌아가면 전환 또는 장애 조치를 `available`다시 시도합니다.

예약된 유지 관리 작업 취소

에서는 예약된 유지 관리 작업(옵트인 실행 취소)을 취소할 수 AWS Management Console없으므로를 사용합니다 AWS CLI.

Using the AWS CLI

1. `--resource-identifier` 옵션을 사용하여 describe-pending-maintenance-actions CLI 명령을 실행하고 `OptInStatus`가 인 유지 관리 작업을 찾습니다`next-maintenance`.

   다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

   Linux, macOS, Unix의 경우:

   ```
   aws docdb describe-pending-maintenance-actions \
      --resource-identifier arn:aws:rds:us-
   east-1:001234567890:cluster:docdb-2025-03-27-19-21-15 \
      --region us-east-1
   ```

   Windows의 경우:

   ```
   aws docdb describe-pending-maintenance-actions ^
   ```

```
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15 ^
   --region us-east-1
```

결과는 다음과 비슷합니다. 다음 단계에서 사용하는 Action 값(os-upgrade이 예제에서는)을 기록해 둡니다.

```
{
    "PendingMaintenanceActions": [
        {
            "ResourceIdentifier": "arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "os-upgrade",
                    "OptInStatus": "next-maintenance",
                    "CurrentApplyDate": "2026-09-02T03:02:00Z",
                    "Description": "New Operating System update is available"
                }
            ]
        }
    ]
}
```

2. 를 사용하여 apply-pending-maintenance-action CLI 명령을 실행하고 이전 단계의 Action 값을에 --opt-in-type undo-opt-in전달하여 예약된 작업을 취소합니다--apply-action.

   Linux, macOS, Unix의 경우:

```
aws docdb apply-pending-maintenance-action \
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15 \
   --apply-action os-upgrade \
   --opt-in-type undo-opt-in \
   --region us-east-1
```

   Windows의 경우:

```
aws docdb apply-pending-maintenance-action ^
```

```
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15 ^
   --apply-action os-upgrade ^
   --opt-in-type undo-opt-in ^
   --region us-east-1
```

응답에서 유지 관리 작업에는 예약된 옵트인이 취소되었음을 확인하는 `OptInStatus`이 더 이상 없습니다.

```
{
    "ResourcePendingMaintenanceActions": {
        "ResourceIdentifier": "arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15",
        "PendingMaintenanceActionDetails": [
            {
                "Action": "os-upgrade",
                "Description": "New Operating System update is available"
            }
        ]
    }
}
```

예약된 클러스터 수정 되돌리기

Using the AWS CLI

1. describe-db-clusters CLI 명령을 실행하고 `PendingModifiedValues` 출력 필드를 검사하여 예약된 수정 사항을 확인합니다.

> **Note**
>
> Amazon DocumentDB `describe-db-clusters` API는 클러스터 수준 `PendingModifiedValues` 필드를 반환하지 않으므로 Amazon DocumentDB CLI(`aws rds`)가 아닌이 명령에 Amazon RDS CLI(`aws docdb`)를 사용합니다. Amazon DocumentDB

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

Linux, macOS, Unix의 경우:

```
aws rds describe-db-clusters \
   --db-cluster-identifier docdb-2025-03-27-19-21-15 \
   --query 'DBClusters[0].PendingModifiedValues' \
   --region us-east-1
```

Windows의 경우:

```
aws rds describe-db-clusters ^
   --db-cluster-identifier docdb-2025-03-27-19-21-15 ^
   --query "DBClusters[0].PendingModifiedValues" ^
   --region us-east-1
```

결과에 예약된 변경 사항이 표시됩니다. 이 예제에서는 클러스터 이름(DBClusterIdentifier)에 대한 변경 사항이에서 docdb-2025-03-27-19-21-15로 보류 중입니다docdb-new-name.

```
{
    "DBClusterIdentifier": "docdb-new-name"
}
```

2. 를 사용하여 modify-db-cluster CLI 명령을 실행하고 수정된 값을 원래 값으로 다시 --apply-immediately설정하여 변경 사항을 되돌립니다. 이 예제에서는를 클러스터의 현재 이름으로 --new-db-cluster-identifier 다시 설정하여 보류 중인 이름 변경을 되돌립니다.

   Linux, macOS, Unix의 경우:

   ```
   aws docdb modify-db-cluster \
      --db-cluster-identifier docdb-2025-03-27-19-21-15 \
      --new-db-cluster-identifier docdb-2025-03-27-19-21-15 \
      --apply-immediately \
      --region us-east-1
   ```

   Windows의 경우:

   ```
   aws docdb modify-db-cluster ^
      --db-cluster-identifier docdb-2025-03-27-19-21-15 ^
      --new-db-cluster-identifier docdb-2025-03-27-19-21-15 ^
   ```

```
    --apply-immediately ^
    --region us-east-1
```

Amazon DocumentDB 글로벌 클러스터에 대한 RPOs 관리

Amazon DocumentDB 글로벌 클러스터를 사용하면 `global_db_rpo` 파라미터를 사용하여 복구 시점 목표(RPO)를 관리할 수 있습니다. RPO는 중단 이벤트로 인해 손실될 수 있는 최대 데이터 양을 나타냅니다.

Amazon DocumentDB 글로벌 클러스터에 대한 RPO를 설정하면 Amazon DocumentDB는 모든 보조 클러스터의 RPO 지연 시간을 모니터링합니다. 이 모니터링을 통해 하나 이상의 보조 클러스터가 대상 RPO 기간 내에 유지됩니다.

RPO 설정은 장애 조치가 발생할 경우 잠재적인 데이터 손실을 제한하기 위해 Amazon DocumentDB가 기본 클러스터에서 쓰기 트랜잭션을 관리하는 방법을 제어합니다. Amazon DocumentDB는 다음과 같이 프라이머리에서 트랜잭션을 커밋(또는 차단)하기 위한 RPO 및 RPO 지연 시간을 평가합니다.

- 하나 이상의 보조 DB 클러스터의 RPO 지연 시간이 이 RPO보다 적은 경우 트랜잭션을 커밋합니다.
- 모든 보조 DB 클러스터의 RPO 지연 시간이 이 RPO보다 큰 경우 트랜잭션을 차단합니다.

즉, 모든 보조 클러스터가 대상 RPO 뒤에 있는 경우 Amazon DocumentDB는 기본 클러스터에서 트랜잭션을 일시 중지합니다. Amazon DocumentDB는 하나 이상의 보조 DB 클러스터의 지연 시간이 RPO 아래로 떨어지는 즉시 일시 중지된 트랜잭션을 재개하고 커밋합니다. 결과적으로 RPO가 충족될 때까지 트랜잭션이 커밋될 수 없습니다.

`global_db_rpo` 형식은 동적 파라미터입니다. 지연 시간이 충분히 감소할 때까지 모든 쓰기 트랜잭션이 중단되지 않도록 하려는 경우, 신속하게 재설정할 수 있습니다. 이 경우 Amazon DocumentDB는 짧은 지연 후 변경 사항을 적용합니다.

> **Important**
>
> AWS 리전이 두 개뿐인 글로벌 데이터베이스에서는 `global_db_rpo` 파라미터의 기본값을 보조 리전의 파라미터 그룹에 유지하는 것이 좋습니다. 그렇지 않으면 기본 AWS 리전의 손실로 인해 장애 조치를 수행하면 Amazon DocumentDB가 트랜잭션을 일시 중지할 수 있습니다. 대신 Amazon DocumentDB가 이전에 실패한 AWS 리전에서 클러스터 재구축을 완료할 때까지 기다린 후이 파라미터를 변경하여 최대 RPO를 적용합니다.

주제

- 복구 시점 목표 설정
- 복구 시점 목표 보기
- 복구 시점 목표 비활성화

복구 시점 목표 설정

`global_db_rpo` 파라미터는 Amazon DocumentDB 데이터베이스의 RPO 설정을 제어합니다. 유효한 값의 범위는 20초~2,147,483,647초(68년)입니다. 비즈니스 요구 사항에 맞는 현실적인 값을 선택합니다. 예를 들어, RPO에 대해 최대 10분 허용할 수 있습니다. 이 경우, 값을 600으로 설정합니다.

AWS Management Console AWS CLI, 또는 Amazon DocumentDB API를 사용하여 Amazon DocumentDB 글로벌 클러스터에 대해이 값을 설정할 수 있습니다.

Using the AWS Management Console

RPO를 설정하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. Amazon DocumentDB 글로벌 클러스터의 기본 클러스터를 선택하고 구성 탭을 열어 DB 클러스터 파라미터 그룹을 찾습니다.

   파라미터 그룹은 직접 편집할 수 없습니다. 대신, 다음 작업을 수행합니다.

   - 적절한 기본 파라미터 그룹을 시작 지점으로 사용하여 사용자 지정 DB 클러스터 파라미터 그룹을 생성합니다.
   - 사용자 지정 DB 클러스터 파라미터 그룹에서 사용 사례에 맞게 global_db_rpo 파라미터의 값을 설정합니다. 유효한 값은 20초에서 최대 정수 값인 2,147,483,647(68년)까지입니다.
   - 수정된 DB 클러스터 파라미터 그룹을 Amazon DocumentDB DB 클러스터에 적용합니다.

DB 클러스터 파라미터 그룹 수정에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB 클러스터 파라미터 그룹 수정.

Using the AWS CLI

`global_db_rpo` 파라미터를 설정하려면 modify-db-cluster-parameter-group CLI 명령을 사용합니다. 명령에서 기본 클러스터의 파라미터 그룹 이름과 RPO 파라미터 값을 지정합니다.

다음 예제에서는 `my_custom_global_parameter_group`이라는 기본 DB 클러스터 파라미터 그룹의 RPO를 600초(10분)로 설정합니다.

Linux, macOS 또는 Unix의 경우는 다음과 같습니다.

```
aws docdb modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my_custom_global_parameter_group \
    --parameters
 "ParameterName=global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

Windows의 경우:

```
aws docdb modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my_custom_global_parameter_group ^
    --parameters
 "ParameterName=global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

Using the Amazon DocumentDB API

`global_db_rpo` 파라미터를 수정하려면 ModifyDBClusterParameterGroup API 작업을 사용합니다.

복구 시점 목표 보기

글로벌 클러스터의 복구 시점 목표(RPO)는 각 DB 클러스터의 `global_db_rpo` 파라미터에 저장됩니다.

CLI를 사용하여 Amazon DocumentDB DB 클러스터의 `global_db_rpo` 파라미터를 볼 수 있습니다. `--query` 옵션을 사용하여 `global_db_rpo` 파라미터 그룹의 파라미터만 반환합니다.

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name my_custom_global_parameter_group \
    --query "Parameters[?ParameterName=='global_db_rpo']"
```

Windows의 경우:

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name my_custom_global_parameter_group ^
```

```
    --query "Parameters[?ParameterName=='global_db_rpo']"
```

이 명령은 다음과 비슷한 출력을 반환합니다.

```
[
    {
        "ParameterName": "global_db_rpo",
        "Description": "(s) Recovery point objective threshold, in seconds, that blocks
 user commits when it is violated.",
        "Source": "engine-default",
        "ApplyType": "dynamic",
        "DataType": "integer",
        "AllowedValues": "20-2147483647",
        "IsModifiable": true,
        "ApplyMethod": "immediate"
    }
]
```

클러스터 파라미터 그룹의 파라미터 보기에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB 클러스터 파라미터 그룹 관리.

복구 시점 목표 비활성화

RPO를 비활성화하려면 `global_db_rpo` 파라미터를 재설정합니다. AWS CLI, 또는 Amazon DocumentDB API AWS Management Console를 사용하여 파라미터를 재설정할 수 있습니다.

Using the AWS Management Console

RPO를 비활성화하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.
3. 목록에서 기본 DB 클러스터 파라미터 그룹을 선택합니다.
4. global_db_rpo 파라미터 옆에 있는 라디오 버튼을 선택합니다.
5. 기본값으로 재설정을 선택하고 확인합니다.

콘솔을 사용하여 파라미터를 재설정하는 방법에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB 클러스터 파라미터 그룹 수정.

Using the AWS CLI

`global_db_rpo` 파라미터를 재설정하려면 reset-db-cluster-parameter-group 명령을 사용합니다.

Linux, macOS 또는 Unix의 경우는 다음과 같습니다.

```
aws docdb reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my_custom_global_parameter_group \
    --parameters "ParameterName=global_db_rpo,ApplyMethod=immediate"
```

Windows의 경우:

```
aws docdb reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my_custom_global_parameter_group ^
    --parameters "ParameterName=global_db_rpo,ApplyMethod=immediate"
```

Using the Amazon DocumentDB API

`global_db_rpo` 파라미터를 재설정하려면 ResetDBClusterParameterGroup API 작업을 사용합니다.

Amazon DocumentDB 클러스터 관리

Amazon DocumentDB 클러스터를 관리하려면 적절한 Amazon DocumentDB 컨트롤 플레인 권한을 가진 IAM 정책이 있어야 합니다. 이러한 권한은 클러스터와 인스턴스를 생성, 수정 및 삭제하도록 허용합니다. `AmazonDocDBFullAccess` 정책은 Amazon DocumentDB 클러스터를 관리하는 데 필요한 모든 권한을 제공합니다.

다음 주제에서는 Amazon DocumentDB 클러스터로 작업할 때 클러스터 생성, 삭제, 수정, 연결, 보기 등과 같은 다양한 작업을 수행하는 방법을 보여줍니다.

주제

- 클러스터 이해
- Amazon DocumentDB 클러스터 설정
- Amazon DocumentDB 클러스터 스토리지 구성
- 클러스터 상태 확인
- Amazon DocumentDB 클러스터 수명 주기
- Amazon DocumentDB 클러스터 규모 조정

- Amazon DocumentDB 클러스터에 대한 볼륨 복제
- Amazon DocumentDB 클러스터 내결함성에 대한 이해

클러스터 이해

Amazon DocumentDB에서는 컴퓨팅과 스토리지를 구분하고, 데이터 복제 및 백업을 클러스터 볼륨으로 오프로드합니다. 클러스터 볼륨은 가용 영역 3곳에 6가지 방식으로 데이터를 복제하는 내구성이 뛰어나고 안정적인 고가용성 스토리지 계층을 제공합니다. 복제본은 높은 데이터 가용성을 확보하고 읽기 확장이 가능합니다. 각 클러스터를 최대 15개의 복제본으로 스케일 업할 수 있습니다.

명사	설명	API 작업(동사)

Cluster	하나 이상의 인스턴스와 이 인스턴스의 데이터를 관리하는 클러스터 스토리지 볼륨으로 구성됩니다.	`create-db-cluster`
		`delete-db-cluster`
		`describe-db-clusters`
		`modify-db-cluster`

Instance	클러스터 스토리지 볼륨에 대한 데이터 읽기 및 쓰기는 인스턴스를 통해 수행됩니다. 지정된 클러스터에는 기본 인스턴스와 복제본의 2가지 인스턴스가 있습니다. 클러스터는 항상 기본 인스턴스 하나와 0~15개의 복제본을 가질 수 있습니다.	`create-db-instance`
		`delete-db-instance`
		`describe-db-instances`
		`modify-db-instance`
		`describe-orderable-db-instance-options`
		`reboot-db-instance`

클러스터 볼륨	3개의 가용 영역을 모두 아우르는 가상 데이터베이스 스토리지 볼륨으로서, 각 가용 영역에는 클러스터 데이터의 사본이 2개 있습니다.	해당 사항 없음

명사	설명	API 작업(동사)
기본 인스턴스	읽기 및 쓰기 작업을 모두 지원하고, 클러스터 볼륨의 모든 데이터 수정을 실행합니다. 클러스터마다 기본 인스턴스가 하나씩 있습니다.	해당 사항 없음
복제본 인스턴스	읽기 연산만 지원합니다. 각 Amazon DocumentDB 클러스터는 기본 인스턴스에 더해 최대 15개의 복제 인스턴스를 포함할 수 있습니다. 복제본이 여러 개 있으면 읽기 워크로드가 분산됩니다. 복제본을 별도의 가용 영역에 두어 데이터베이스 가용성을 높일 수도 있습니다.	해당 사항 없음
클러스터 엔드포인트	클러스터의 현재 기본 인스턴스에 연결되는 Amazon DocumentDB 클러스터 엔드포인트입니다. 각 Amazon DocumentDB 클러스터에는 클러스터 엔드포인트 하나와 기본 인스턴스 하나가 있습니다.	해당 사항 없음

명사	설명	API 작업(동사)
리더 엔드포인트	클러스터에서 사용할 수 있는 복제본 중 하나에 연결되는 Amazon DocumentDB 클러스터 엔드포인트입니다. 각 Amazon DocumentDB 클러스터에는 리더 엔드포인트가 1개씩 있습니다. 복제본이 하나 이상이면 리더 엔드포인트는 각 연결 요청을 Amazon DocumentDB 복제본 중 하나로 전달합니다.	해당 사항 없음
인스턴스 엔드포인트	특정 인스턴스에 연결되는 Amazon DocumentDB 클러스터의 인스턴스 엔드포인트입니다. 클러스터의 인스턴스에는 인스턴스 유형에 상관없이 각각 고유한 인스턴스 엔드포인트가 있습니다.	해당 사항 없음

Amazon DocumentDB 클러스터 설정

클러스터를 생성 또는 수정할 때는 생성 후 수정할 수 없는 파라미터와 수정 가능한 파라미터를 이해하는 것이 중요합니다. 다음 표에는 한 클러스터에 적용되는 모든 설정 또는 파라미터가 나와 있습니다. 표에 나와 있듯이 일부 파라미터는 수정할 수 있고 다른 파라미터는 수정할 수 없습니다.

> **Note**
>
> 이러한 설정을 Amazon DocumentDB 클러스터 파라미터 그룹 및 해당 파라미터와 혼동해서는 안 됩니다. 클러스터 파라미터 그룹에 대한 자세한 내용은 Amazon DocumentDB 클러스터 파라미터 그룹 관리 섹션을 참조하세요.

파라미터	수정 가능	참고
DBClusterIdentifier	예	명명 제약 조건: • 길이는 [1~63] 글자, 숫자 또는 하이픈입니다. • 첫 번째 문자는 글자이어야 합니다. • 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다. • 리전별로 Amazon RDS, Amazon Neptune 및 Amazon DocumentDB의 모든 클러스터 AWS 계정에 대해 고유해야 합니다.
Engine	아니요	docdb여야 합니다.
BackupRetentionPeriod	예	1~35일 사이여야 합니다.
DBClusterParameterGroupName	예	명명 제약 조건: • [1-255]자 길이의 영숫자 문자입니다. • 첫 번째 문자는 글자이어야 합니다. • 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.
DBSubnetGroupName	아니요	클러스터가 생성된 이후에는 클러스터의 서브넷을 수정할 수 없습니다.
EngineVersion	아니요	값은 `5.0.0` (기본값)`4.0.0`, 또는 `3.6.0`일 수 있습니다.
KmsKeyId	아니요	클러스터를 암호화하기로 선택한 경우 클러스터를 암호화하는 데 사용한 AWS KMS 키를 변경할 수 없습니다.
MasterUsername	아니요	클러스터가 생성된 이후에는 `MasterUsername` 을 수정할 수 없습니다. 명명 제약 조건:

파라미터	수정 가능	참고
		• 길이는 [1-63]자의 영숫자 문자입니다. • 첫 번째 문자는 글자이어야 합니다. • 데이터베이스 엔진에 포함된 단어는 사용할 수 없습니다.

MasterUserPassword	예	제약 조건:
		• 길이는 [8-100]자의 인쇄 가능한 ASCII 문자입니다.
		• 다음을 제외한 인쇄 가능한 ASCII 문자를 사용할 수 있습니다.
		  • **/**(슬래시)
		  • **"**(큰 따옴표)
		  • **@**(at 기호)

Port	예	포트 번호는 클러스터의 모든 인스턴스에 적용됩니다.
PreferredBackupWindow	예	
PreferredMaintenanceWindow	예	
StorageEncrypted	아니요	클러스터를 암호화하도록 선택한 경우 암호를 해제할 수 없습니다.

파라미터	수정 가능	참고
StorageType	예	DB 클러스터의 스토리지 유형: 표준 (standard) 또는 I/O 최적화(iopt1). 기본값: standard 이 파라미터는 CreateDBCluster 및 ModifyDBCluster 항목을 사용하여 구성할 수 있습니다. 자세한 내용은 Amazon DocumentDB 클러스터 스토리지 구성 섹션을 참조하세요.
Tags	예	
VpcSecurityGroupIds	아니요	클러스터를 만든 후에는 클러스터가 상주하는 VPC를 수정할 수 없습니다.

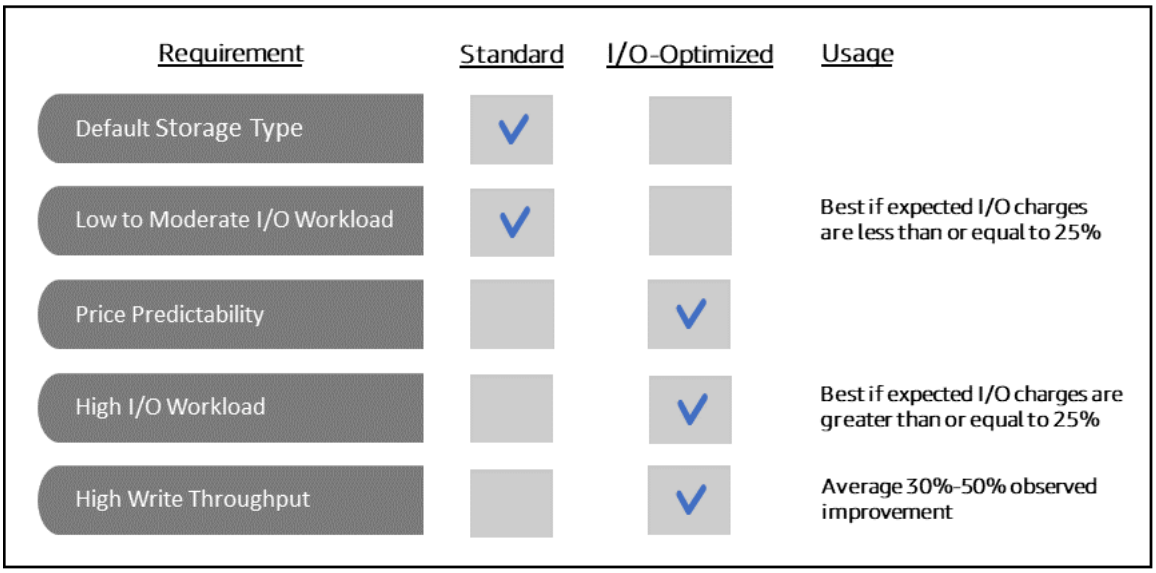
Amazon DocumentDB 클러스터 스토리지 구성

Amazon DocumentDB 5.0부터 인스턴스 기반 클러스터는 다음의 두 가지 스토리지 구성 유형을 지원합니다.

- Amazon DocumentDB 표준 스토리지: I/O 소비가 낮거나 중간 수준인 고객을 위해 설계되었습니다. I/O 비용이 총 Amazon DocumentDB 클러스터의 25% 미만일 것으로 예상되는 경우 이러한 선택이 적합할 수 있습니다. Amazon DocumentDB 표준 스토리지 구성을 사용하면 인스턴스 및 스토리지 요금 외에도 요청당 지불 I/O 기준으로 요금이 청구됩니다. 즉, 사용량에 따라 주기별로 청구가 달라질 수 있습니다. 이 구성은 애플리케이션의 변화하는 I/O 수요에 맞게 조정됩니다.
- Amazon DocumentDB I/O 최적화 스토리지: 가격 예측 가능성을 우선시하거나 I/O 집약적 애플리케이션을 사용하는 고객을 위해 설계되었습니다. I/O 최적화 구성은 I/O 집약적 워크로드를 사용하는 고객에게 향상된 성능, 향상된 처리량 및 감소된 지연 시간을 제공합니다. I/O 비용이 총 Amazon DocumentDB 클러스터 비용의 25%를 초과할 것으로 예상되는 경우 이 옵션은 향상된 가격 성능을 제공합니다. Amazon DocumentDB I/O 최적화 스토리지 구성을 사용하면 I/O 작업에 따른 요금이 부과되지 않으므로 각 결제 주기마다 비용을 예측 가능하도록 보장해 줍니다. 구성은 성능을 개선하면서 비용을 안정화합니다.

30일마다 한 번씩 기존 데이터베이스 클러스터를 Amazon DocumentDB I/O 최적화 스토리지로 전환할 수 있습니다. 언제든지 Amazon DocumentDB 표준 스토리지로 다시 전환할 수 있습니다. 스토리지 구성을 I/O 최적화로 수정하는 다음 날짜는 `describe-db-clusters` 명령을 사용하여 추적 AWS CLI 하거나 클러스터의 구성 페이지에서 AWS Management Console 를 통해 추적할 수 있습니다.

Amazon DocumentDB I/O 최적화 구성을 포함한 새 데이터베이스 클러스터를 생성하거나 AWS Management Console에서 몇 번의 클릭, AWS Command Line Interface (AWS CLI)에서 단일 파라미터 변경 또는 AWS SDK를 통해 기존 데이터베이스 클러스터를 변환할 수 있습니다. 스토리지 구성을 수정하는 동안 또는 수정한 후에 필요한 인스턴스의 가동 중지 시간이나 재부팅은 없습니다.

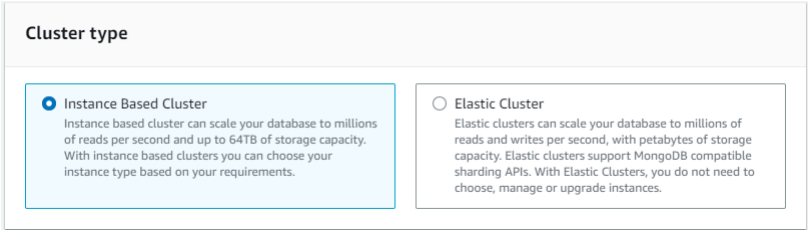


I/O 최적화 클러스터 생성

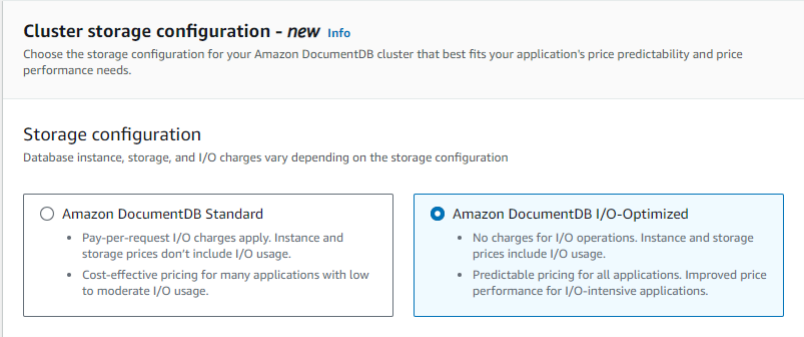
Using the AWS Management Console

AWS Management Console을 사용하여 I/O 최적화 클러스터를 생성하거나 수정합니다.

1. Amazon DocumentDB 관리 콘솔의 클러스터 에서 클러스터 생성 또는 선택을 선택하고 작업을 선택한 다음 수정을 선택합니다.
2. 새 클러스터를 생성하는 경우 클러스터 유형 섹션에서 인스턴스 기반 클러스터를 선택해야 합니다(기본 옵션).



3. 구성 섹션의 클러스터 스토리지 구성에서 Amazon DocumentDB I/O 최적화를 선택합니다.



4. 클러스터 생성 또는 수정을 완료하고 클러스터 생성 또는 클러스터 수정을 선택합니다.

   전체 클러스터 생성 프로세스는 를 사용하여 클러스터 및 기본 인스턴스 생성 AWS Management Console 항목을 참조하세요.

   전체 클러스터 수정 프로세스는 Amazon DocumentDB 클러스터 수정 항목을 참조하세요.

Using the AWS CLI

AWS CLI를 사용하여 I/O 최적화 클러스터를 생성합니다.

다음 예에서는 자신의 정보로 각각의 *### ## ## ###*를 바꿉니다.

Linux, macOS, Unix의 경우:

```
aws docdb create-db-cluster \
      --db-cluster-identifier sample-cluster \
      --engine docdb \
      --engine-version 5.0.0 \
      --storage-type iopt1 \
      --deletion-protection \
      --master-username username \
      --master-user-password password
```

Windows의 경우:

```
aws docdb create-db-cluster ^
      --db-cluster-identifier sample-cluster ^
      --engine docdb ^
      --engine-version 5.0.0 ^
      --storage-type iopt1 ^
      --deletion-protection ^
```

```
    --master-username username ^
    --master-user-password password
```

스토리지 구성을 결정하기 위한 비용 분석

Amazon DocumentDB를 사용하면 보유한 모든 데이터베이스 클러스터에 대한 스토리지 구성을 유연하게 선택할 수 있습니다. 표준 및 I/O 최적화 간에 클러스터를 올바르게 할당하기 위해 Amazon DocumentDB 비용을 클러스터별로 추적할 수 있습니다. 이렇게 하려면 기존 클러스터에 태그를 추가하고, AWS 결제 및 비용 관리 대시보드에서 비용 할당 태그 지정을 활성화하고, AWS Cost Explorer Service에서 지정된 클러스터에 대한 비용을 분석할 수 있습니다. 비용 분석에 대한 자세한 내용은 블로그 비용 할당 태그 사용을 참조하세요.

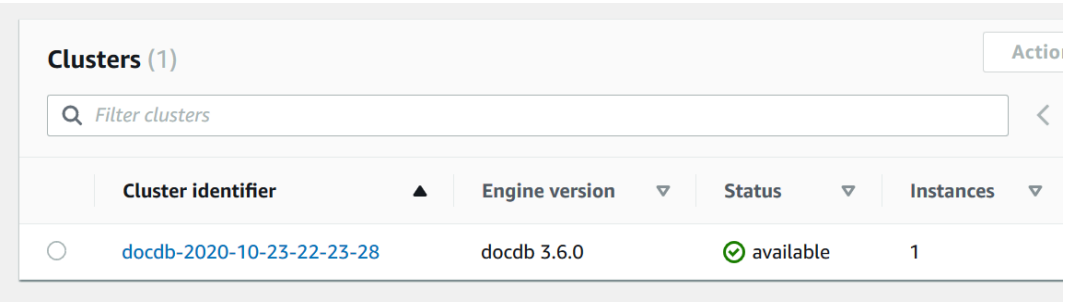
클러스터 상태 확인

AWS Management Console 또는를 사용하여 클러스터의 상태를 확인할 수 있습니다 AWS CLI.

Using the AWS Management Console

다음 절차에 따라를 사용하여 Amazon DocumentDB 클러스터의 상태를 확인합니다. AWS Management Console

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터 식별자 열에서 원하는 클러스터의 이름을 찾습니다. 그런 다음 클러스터의 상태를 찾으려면 아래와 같이 상태 열에 대한 해당 행을 읽습니다.



Using the AWS CLI

`describe-db-clusters` 작업에 따라 AWS CLI를 사용하여 Amazon DocumentDB 클러스터의 상태를 확인합니다.

다음 코드는 `sample-cluster` 클러스터의 상태를 찾습니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster  \
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

Windows의 경우:

```
aws docdb describe-db-clusters ^
    --db-cluster-identifier sample-cluster  ^
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
[
    [
        "sample-cluster",
        "available"
    ]
]
```

Amazon DocumentDB 클러스터 수명 주기

Amazon DocumentDB 클러스터의 수명 주기는 클러스터 생성, 설명, 수정, 삭제로 구성되어 있습니다. 이 섹션에서는 이러한 프로세스를 완료하는 방법에 관해 설명합니다.

주제

- Amazon DocumentDB 클러스터 생성
- Amazon DocumentDB 클러스터에 대한 설명
- Amazon DocumentDB 클러스터 수정
- 대기 중인 유지 관리 확인
- 클러스터의 엔진 버전에 대한 패치 업데이트 수행
- 클러스터의 운영 체제 업데이트
- Amazon DocumentDB 클러스터 중지 및 시작

- Amazon DocumentDB 클러스터 삭제

Amazon DocumentDB 클러스터 생성

Amazon DocumentDB 클러스터는 인스턴스와 해당 클러스터의 데이터를 나타내는 클러스터 볼륨으로 구성됩니다. 클러스터 볼륨은 3개의 가용 영역에서 단일 가상 볼륨으로 6개의 방법으로 복제됩니다. 클러스터에는 기본 인스턴스와 옵션으로 최대 15개의 복제 인스턴스가 포함됩니다.

다음 섹션에서는 AWS Management Console 또는를 사용하여 Amazon DocumentDB 클러스터를 생성하는 방법을 보여줍니다 AWS CLI. 그런 다음 해당 클러스터에 대한 복제 인스턴스를 추가할 수 있습니다. 콘솔을 사용하여 Amazon DocumentDB 클러스터를 생성하면 기본 인스턴스가 자동으로 동시에 생성됩니다. AWS CLI 를 사용하여 Amazon DocumentDB 클러스터를 생성하는 경우 클러스터의 상태를 사용할 수 있게 되면 해당 클러스터의 기본 인스턴스를 생성해야 합니다.

사전 조건

다음은 Amazon DocumentDB 클러스터를 생성할 때 필요한 사전 조건입니다.

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://signin.aws.amazon.com/signup?request_type=register를 엽니다.
2. 온라인 지시 사항을 따르세요.

   등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

VPC 사전 조건

Amazon DocumentDB 클러스터는 Amazon Virtual Private Cloud(Amazon VPC)에서만 생성할 수 있습니다. Amazon DocumentDB 클러스터에서 Amazon VPC를 사용하려면 2개 이상의 가용 영역마다 VPC에 서브넷이 1개 이상 있어야 합니다. 가용 영역에 클러스터 인스턴스를 배포하면 어쩌다가 가용 영역에 장애가 발생해도 클러스터에서 인스턴스를 사용할 수 있습니다.

서브넷 사전 조건

Amazon DocumentDB 클러스터를 생성하는 경우 VPC와 VPC 내 해당 서브넷 그룹을 선택해야 클러스터를 시작할 수 있습니다. 서브넷은 인스턴스를 시작하기 위해 사용할 가용 영역과 해당 가용 영역 내 IP 범위를 결정합니다. 설명을 위해 서브넷과 가용 영역이라는 용어가 혼용됩니다. 서브넷 그룹은 이름이 지정된 서브넷의 집합(또는 가용 영역)입니다. 서브넷 그룹을 통해 Amazon DocumentDB 인스턴스를 시작하는 데 사용할 가용 영역을 지정할 수 있습니다. 예를 들어, 인스턴스가 3개인 클러스터에서 고가용성을 위해 각 인스턴스를 별도의 가용 영역에 프로비저닝하는 것이 좋습니다. 이렇게 하면 가용 영역 하나가 작동이 안 되는 경우 인스턴스 하나만 영향을 받게 됩니다.

Amazon DocumentDB 인스턴스는 현재 최대 3개의 가용 영역에 프로비저닝할 수 있습니다. 서브넷 그룹에 서브넷이 3개 이상 있는 경우에도 Amazon DocumentDB 클러스터 생성에는 그 중 3개만 사용할 수 있습니다. 따라서 서브넷 그룹을 생성할 때 인스턴스 배포에 사용하려는 서브넷을 3개만 선택하는 것이 좋습니다. 미국 동부(버지니아 북부)에서 서브넷 그룹에는 6개의 서브넷(또는 가용 영역)이 있을 수 있습니다. 그러나 Amazon DocumentDB 클러스터가 프로비저닝되면 Amazon DocumentDB에서는 가용 영역 중 3개를 선택하여 인스턴스 프로비저닝에 사용합니다.

예를 들어, 클러스터를 생성할 때 Amazon DocumentDB에서 가용 영역{1A, 1B 및 1C}를 선택한다고 가정하겠습니다. 가용 영역{1D}에 인스턴스를 생성하려고 하면 API 직접 호출이 실패합니다. 하지만 특정 가용 영역을 지정하지 않고 인스턴스를 생성하기로 선택하면 Amazon DocumentDB가 사용자를 대신하여 가용 영역을 선택합니다. Amazon DocumentDB는 알고리즘을 사용하여 가용 영역 전체에 걸쳐 인스턴스의 부하를 분산하므로 고가용성을 확보할 수 있습니다. 예를 들어, 인스턴스 3개가 프로비저닝되면 이러한 인스턴스가 기본적으로 가용 영역 1개에 모두 프로비저닝되지 않고 가용 영역 3개에 프로비저닝됩니다.

권장 사항:

- 특별한 사유가 없으면 항상 서브넷 그룹 서브넷을 3개 생성합니다. 이렇게 하면 인스턴스가 3개 이상인 클러스터의 경우 인스턴스가 가용 영역 3개에 프로비저닝되므로 더 높은 가용성을 달성할 수 있습니다.
- 고가용성을 달성하려면 항상 여러 가용 영역에 인스턴스를 분산시킵니다. 가용 영역 1개에 클러스터의 모든 인스턴스를 배치하지 마세요.
- 장애 조치 이벤트는 언제든지 발생할 수 있으므로 기본 인스턴스나 복제본 인스턴스가 항상 특정 가용 영역에 있다고 가정해서는 안 됩니다.

추가 사전 조건

다음은 Amazon DocumentDB 클러스터를 생성할 때 필요한 기타 사전 요구 사항입니다.

- AWS Identity and Access Management (IAM) 자격 증명을 AWS 사용하여에 연결하는 경우 IAM 계정에 Amazon DocumentDB 작업을 수행하는 데 필요한 권한을 부여하는 IAM 정책이 있어야 합니다.

  IAM 계정을 사용하여 Amazon DocumentDB 콘솔에 액세스하는 경우 먼저 IAM 계정으로 AWS Management Console 에 로그인해야 합니다. https://console.aws.amazon.com/docdb에서 Amazon DocumentDB 콘솔로 이동합니다.
- 클러스터의 구성 파라미터를 사용자 지정하려면 클러스터 파라미터 그룹과 파라미터 그룹을 필요한 파라미터 설정으로 지정해야 합니다. 클러스터 파라미터 그룹 또는 파라미터 그룹의 생성 또는 수정에 대한 자세한 내용은 Amazon DocumentDB 클러스터 파라미터 그룹 관리 섹션을 참조하세요.
- 클러스터에 지정할 TCP/IP 포트 번호를 결정해야 합니다. 일부 기업에서는 방화벽이 Amazon DocumentDB에 대한 기본 포트 연결을 차단하는 경우도 있습니다. 이처럼 기업 방화벽이 기본 포트를 차단할 경우 클러스터에 다른 포트를 선택해야 합니다. 클러스터의 인스턴스는 모두 동일한 포트를 사용합니다.

를 사용하여 클러스터 및 기본 인스턴스 생성 AWS Management Console

다음 절차에서는 콘솔을 사용하여 인스턴스가 1개 이상인 Amazon DocumentDB 클러스터를 시작하는 방법을 설명합니다.

클러스터 생성: 기본 설정 사용

를 사용하여 기본 설정을 사용하여 인스턴스가 있는 클러스터를 생성하려면 AWS Management Console

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 미국 동부(버지니아 북부) 리전 AWS 리전 이외의에서 클러스터를 생성하려면 콘솔의 오른쪽 상단 섹션에 있는 목록에서 리전을 선택합니다.
3. 탐색 창에서 클러스터를 선택한 다음 생성을 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택합니다.

4. Amazon DocumentDB 클러스터 생성 페이지에서 구성 창을 작성합니다.

a. 클러스터 식별자 - Amazon DocumentDB 제공 이름을 수락하거나 **`sample-cluster`**와 같은 클러스터 이름을 입력합니다.

   클러스터 명명 제약 조건:

   - 길이는 [1-63]개의 글자, 숫자 또는 하이픈입니다.
   - 첫 번째 문자는 글자이어야 합니다.
   - 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.
   - 리전별로 Amazon RDS, Neptune 및 Amazon DocumentDB의 모든 클러스터 AWS 계정에 대해 고유해야 합니다.

b. 엔진 버전 - 기본 엔진 버전인 5.0.0을 수락하거나 선택적으로 8.0.0, 4.0.0 또는 3.6.0을 선택합니다.

c. 인스턴스 클래스 - 기본값 `db.r5.large`를 수락하거나 목록에서 원하는 인스턴스 클래스를 선택합니다.

d. 인스턴스 수 - 목록에서 이 클러스터로 생성하려는 인스턴스 수를 선택합니다. 첫 번째 인스턴스는 기본 인스턴스이며, 그 외 다른 모든 인스턴스는 읽기 전용 복제본 인스턴스입니다. 나중에 필요하면 인스턴트를 추가하거나 삭제할 수 있습니다. 기본적으로 Amazon DocumentDB 클러스터는 세 개의 인스턴트를 시작합니다(기본 한 개, 복제본 두 개).

5. 클러스터 스토리지 구성 섹션을 완료합니다.

   Amazon DocumentDB Standard(기본값) 또는 Amazon DocumentDB I/O 최적화를 선택합니다. 자세한 내용은 Amazon DocumentDB 클러스터 스토리지 구성 섹션을 참조하세요.

6. 인증 창을 작성합니다.

a. 사용자 이름 - 기본 사용자의 이름을 입력합니다. 클러스터에 로그인하려면 기본 사용자 이름을 사용해야 합니다.

   기본 사용자 이름 지정 시 제약 조건:

   - 길이는 [1-63]자의 영숫자 문자입니다.
   - 첫 번째 문자는 글자이어야 합니다.
   - 데이터베이스 엔진에 포함된 단어는 사용할 수 없습니다.

b. 다음 암호 옵션 중 하나를 선택합니다.

   - 관리형 AWS Secrets Manager- 기본 사용자 암호를 자동으로 관리 AWS Secrets Manager 하려면이 옵션을 선택합니다.

이 옵션을 선택하는 경우 자체 키를 생성하거나 Secrets Manager가 생성하는 키를 사용하여 KMS 키를 구성합니다.

- 자체 관리 - 기본 사용자 암호를 자체 관리하려면 이 옵션을 선택합니다. 이 옵션을 선택하는 경우 기본 사용자의 암호를 입력한 다음 확인합니다. 클러스터에 로그인하려면 기본 사용자의 암호를 사용해야 합니다.

  암호 제약:

  - 길이는 [8-100]자의 인쇄 가능한 ASCII 문자입니다.
  - 다음을 제외한 인쇄 가능한 ASCII 문자를 사용할 수 있습니다.
    - **/**(슬래시)
    - **"**(큰 따옴표)
    - **@**(at 기호)

7. 화면 하단에서 다음 중 하나를 선택합니다.
   - 지금 클러스터를 생성하려면 클러스터 생성을 선택합니다.
   - 클러스터를 생성하지 않으려면 않으려면 취소를 선택합니다.
   - 생성 전 클러스터를 추가로 구성하려면 추가 구성 표시를 선택한 후 클러스터 생성: 추가 구성에서 계속 진행합니다.

     추가 구성 섹션에 포함된 구성은 다음과 같습니다.

     - 네트워크 설정 - 기본값은 `default` VPC 보안 그룹의 사용입니다.
     - 클러스터 옵션 - 기본값은 포트 27017과 기본 파라미터 그룹을 사용하는 것입니다.
     - 암호화 - 기본값은 `(default) aws/rds` 키를 사용한 암호화 활성화입니다.

       > **Important**
       > 클러스터가 암호화되면 암호화를 해제할 수 없습니다.

     - 백업 - 기본값은 백업을 1일 동안 유지하며 Amazon DocumentDB에서 백업 기간을 선택하는 것입니다.
     - 로그 내보내기 - 기본값은 감사 로그를 CloudWatch 로그로 내보내지 않는 것입니다.
     - 유지 관리 - 기본값은 Amazon DocumentDB에서 유지 관리 기간을 선택하는 것입니다.
     - 삭제 방지 - 실수로 인한 삭제로부터 클러스터를 보호합니다. 콘솔을 사용해 생성된 클러스터의 기본값이 활성화됩니다.

지금 기본 설정을 수락해도 나중에 클러스터를 수정하여 대부분을 변경할 수 있습니다.

8. 클러스터의 보안 그룹에 대해 인바운드 연결을 활성화합니다.

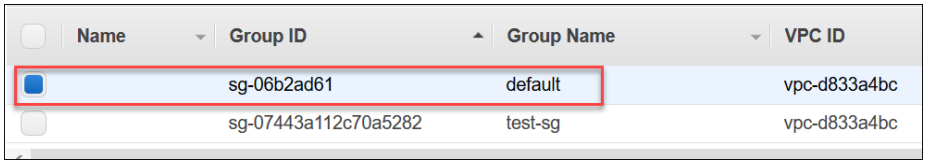
   클러스터의 기본 설정을 변경하지 않은 경우 지정된 리전의 기본 VPC에 대한 기본 보안 그룹을 사용하여 클러스터를 생성했습니다. Amazon DocumentDB에 연결하려면 클러스터의 보안 그룹에 대해 포트 27017(또는 사용자가 선택한 포트)에서 인바운드 연결을 활성화해야 합니다.

   클러스터의 보안 그룹에 대해 인바운드 연결을 추가하려면

   a. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/ec2/ Amazon EC2 콘솔을 엽니다.

   b. 기본 창의 리소스 섹션에서 보안 그룹을 선택합니다.

   c. 보안 그룹 목록에서 클러스터 생성 시 사용한 보안 그룹을 찾고(기본값 보안 그룹일 확률이 높음) 보안 그룹 이름 왼쪽의 상자를 선택합니다.



   d. 작업 메뉴에서 인바운드 규칙 편집을 선택한 다음 규칙 제약을 선택 또는 입력합니다.

      i. 유형 - 목록에서 네트워크 트래픽에 개방할 프로토콜을 선택합니다.

      ii. 프로토콜 - 목록에서 프로토콜 유형을 선택합니다.

      iii. 포트 범위 - 사용자 지정 규칙에 대해 포트 번호나 포트 범위를 입력합니다. 포트 번호 또는 범위에는 클러스터를 생성할 때 지정한 포트가 포함됩니다(기본값: 27017).

      iv. 소스 - 인스턴스에 도달 가능한 트래픽을 지정합니다. 목록에서 트래픽 소스를 선택합니다. 사용자 지정을 선택하는 경우, 단일 IP 주소나 IP 주소 범위를 CIDR 표기법으로 지정합니다(예: 203.0.113.5/32).

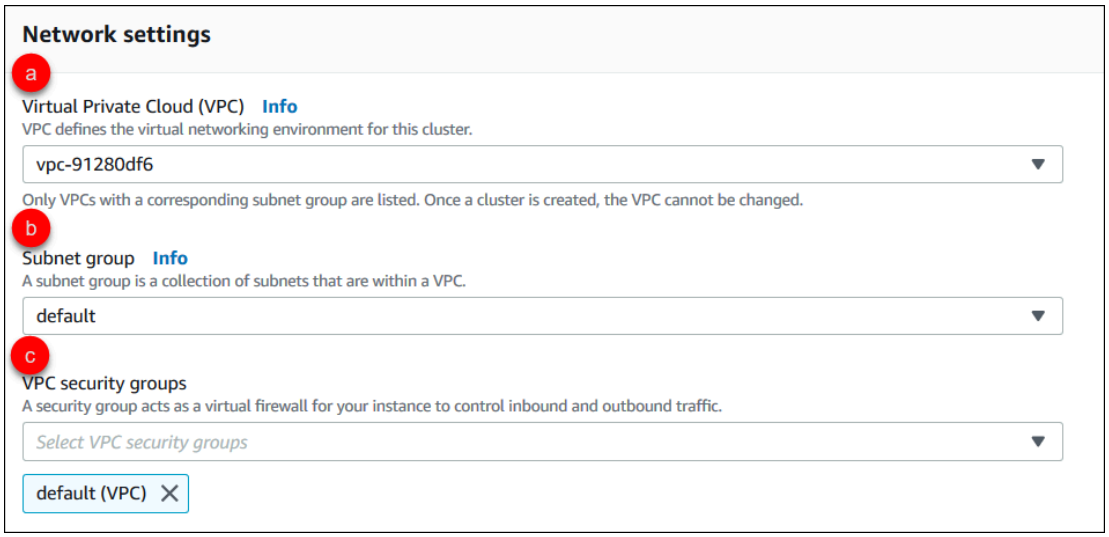
      v. 설명 - 이 규칙에 대한 설명을 입력합니다.

      vi. 규칙 만들기가 끝나면 저장을 선택합니다.

클러스터 생성: 추가 구성

클러스터에 대한 기본 설정을 수락하면 다음 단계를 건너뛰고 클러스터 생성을 선택할 수 있습니다.

1. 네트워크 설정 창을 작성합니다.



   a. 가상 프라이빗 클라우드(VPC) - 목록에서 이 클러스터를 시작할 Amazon VPC를 선택합니다.

   b. 서브넷 그룹 - 목록에서 이 클러스터에 사용할 서브넷 그룹을 선택합니다.

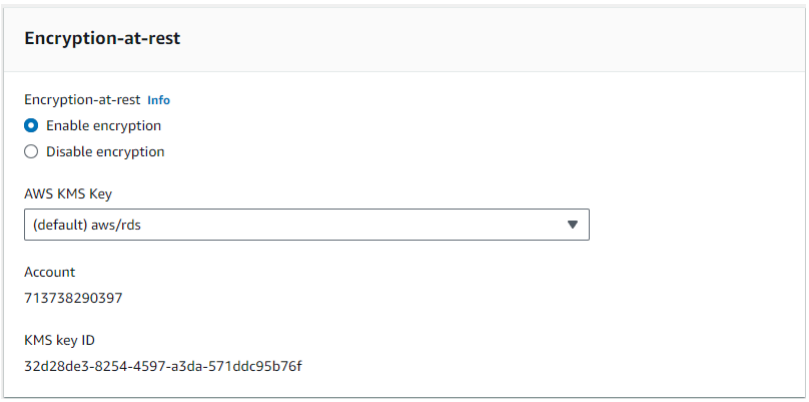
   c. VPC 보안 그룹 - 목록에서 이 클러스터에 사용할 VPC 보안 그룹을 선택합니다.

2. 클러스터 옵션 창을 작성합니다.

   a. 데이터베이스 포트 - 위, 아래 화살표를 사용해 애플리케이션과 인스턴스 연결에 사용할 TCP/IP 포트를 설정합니다.

   b. 클러스터 파라미터 그룹 - 파라미터 그룹 목록에서 이 클러스터에 사용할 클러스터 파라미터 그룹을 선택합니다.

3. 암호화 창을 작성합니다.



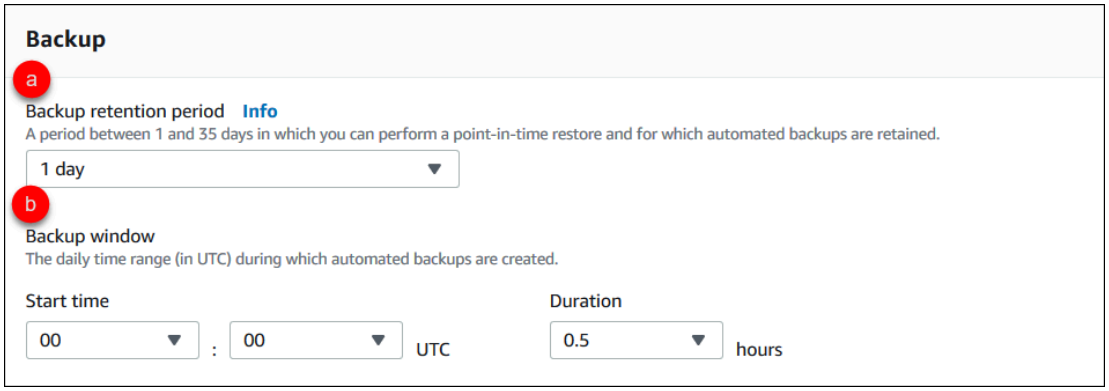
a. 유휴 암호화 - 다음 중 하나를 선택합니다.

- 암호화 활성 - 기본 설정입니다. 모든 저장 데이터는 암호화됩니다. 데이터 암호화를 선택하면 해당 작업을 실행 취소할 수 없습니다.
- 암호화 비활성 - 데이터가 암호화되지 않습니다.

b. AWS KMS 키 - 데이터를 암호화하는 경우에만 사용할 수 있습니다. 목록에서 이 클러스터에 있는 데이터 암호화에 사용할 키를 선택합니다. 기본값은 `(default) aws/rds`입니다.

키 ARN 입력을 선택하면 해당 키의 Amazon 리소스 이름(ARN)을 입력해야 합니다.

4. 백업 창을 작성합니다.



a. 백업 보존 기간 - 목록에서 이 클러스터의 자동 백업을 삭제하지 않고 유지하는 일 수를 선택합니다.

b. 백업 기간 - Amazon DocumentDB에서 이 클러스터를 백업하는 일별 시간과 기간을 설정합니다.

i. 시작 시간 - 첫 번째 목록에서 자동 백업 시작 시간(UTC)을 선택합니다. 두 번째 목록에서 자동 백업 시작 시간의 분을 선택합니다.

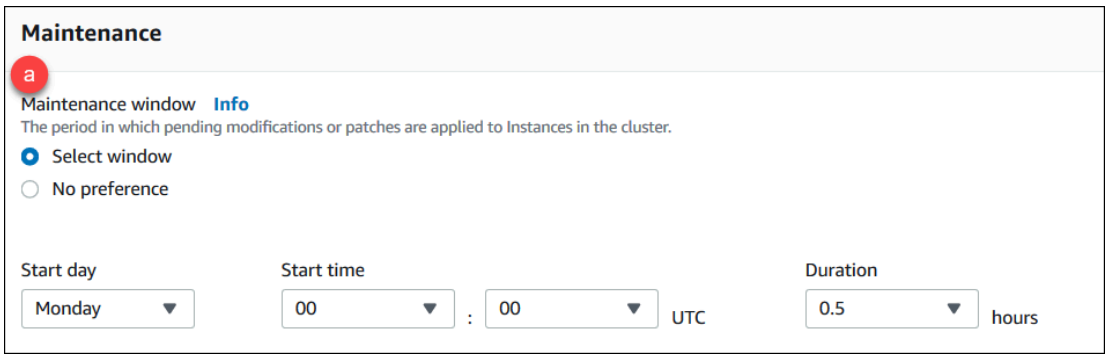
ii. 기간 - 목록에서 자동 백업 생성에 할당된 시간 수를 선택합니다.

5. CloudWatch Logs로 내보내려는 로그 유형을 선택하여 로그 내보내기 창을 완성하세요.

- 감사 로그 - Amazon CloudWatch Logs에 감사 로그 내보내기를 활성화하려면 이 옵션을 선택합니다. 감사 로그를 선택하는 경우 클러스터의 사용자 지정 파라미터 그룹에서 `audit_logs`를 활성화해야 합니다. 자세한 내용은 Amazon DocumentDB 이벤트 감사 섹션을 참조하십시오.
- 프로파일러 로그 - 작업 프로파일러 로그를 Amazon CloudWatch Logs로 내보내려면 이 옵션을 선택합니다. 프로파일러 로그를 선택하는 경우 클러스터의 사용자 지정 파라미터 그룹에서 다음 파라미터도 수정해야 합니다.
  - `profiler` - `enabled`로 설정합니다.
  - `profiler_threshold_ms` - 작업 프로파일링에 대한 임계값을 설정하려면 값 `[0-INT_MAX]`로 설정합니다.
  - `profiler_sampling_rate` - 프로파일링할 느린 작업 비율을 설정하려면 값 `[0.0-1.0]`으로 설정합니다.

  자세한 내용은 Amazon DocumentDB 작업 프로파일링 섹션을 참조하십시오.

6. 유지 관리 창을 작성합니다.



- 다음 중 하나를 선택합니다.
  - 선택 창 - 클러스터에 대한 유지 관리를 수행할 Amazon DocumentDB에 대한 요일, UTC 시작 시간 및 기간을 지정할 수 있습니다.

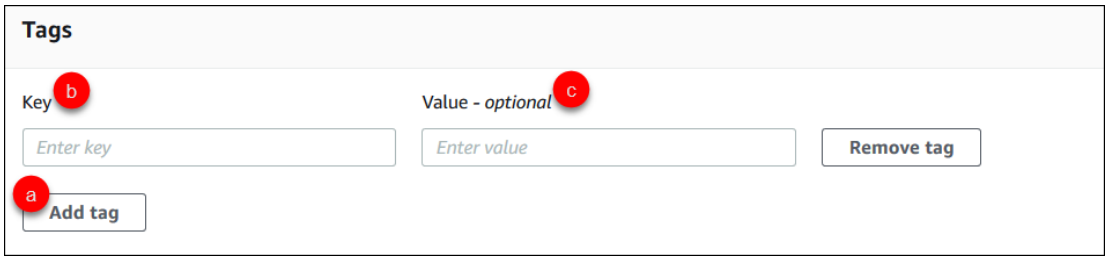
    a. 시작일 - 목록에서 클러스터 유지 관리를 시작할 요일을 선택합니다.

b. 시작 시간 - 목록에서 유지 관리를 시작할 시간과 분(UTC)을 선택합니다.

c. 기간 - 목록에서 클러스터 유지 관리에 할당할 시간의 양을 선택합니다. 유지 관리 작업을 특정 시간에 끝내지 못하면 특정 시간이 지나도 완료될 때까지 유지 관리 프로세스가 계속 진행됩니다.

- 기본 설정 없음 - Amazon DocumentDB에서 유지 관리를 수행할 요일, 시작 시간 및 기간을 선택합니다.

7. 이 클러스터에 하나 이상의 태그를 추가하려면 태그 창을 작성합니다.



클러스터에 추가하려는 각 태그에 대해 다음 단계를 반복합니다. 클러스터 하나에 최대 10개까지 추가할 수 있습니다.

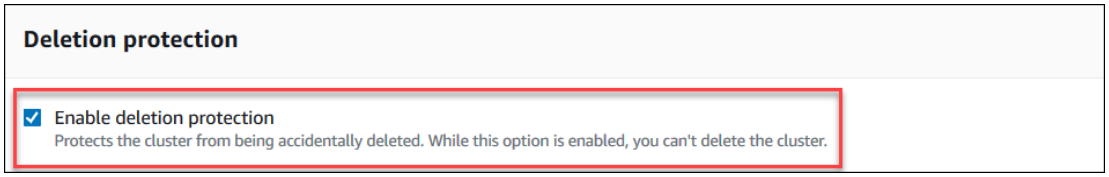
a. 태그 추가를 선택합니다.

b. 태그의 키를 입력합니다.

c. 선택적으로 태그의 값을 입력합니다.

태그를 제거하려면 태그 제거를 선택합니다.

8. 콘솔을 사용하여 클러스터를 생성할 때 기본적으로 삭제 방지가 활성화됩니다. 삭제 방지를 비활성화하려면 삭제 방지 활성화를 선택 취소합니다. 삭제 방지를 활성화하면 클러스터가 삭제되지 않도록 방지합니다. 삭제 방지된 클러스터를 삭제하려면 먼저 클러스터를 수정하여 삭제 방지를 비활성화해야 합니다.



삭제 방지에 대한 자세한 내용은 Amazon DocumentDB 클러스터 삭제 섹션을 참조하세요.

9. 클러스터를 생성하려면 클러스터 생성을 선택합니다. 그렇지 않은 경우 취소를 선택합니다.

를 사용하여 클러스터 생성 AWS CLI

다음 절차에서는를 사용하여 Amazon DocumentDB 클러스터를 AWS CLI 시작하고 Amazon DocumentDB 복제본을 생성하는 방법을 설명합니다.

파라미터

- **`--db-cluster-identifier`** - 필수입니다. 이 클러스터를 식별하는 소문자 문자열입니다.

  클러스터 명명 제약 조건:

  - 길이는 [1-63] 글자, 숫자 또는 하이픈입니다.
  - 첫 번째 문자는 글자이어야 합니다.
  - 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.
  - 리전별로 AWS 계정당 모든 클러스터(Amazon RDS, Amazon Neptune 및 Amazon DocumentDB)에 대해 고유해야 합니다.

- **`--engine`** - 필수입니다. **`docdb`**여야 합니다.
- **`--deletion-protection | --no-deletion-protection`** - 선택 사항. 삭제 방지를 활성화하면 클러스터가 삭제되지 않도록 방지합니다. 를 사용할 때 기본 설정은 삭제 방지 AWS CLI기능을 비활성화하는 것입니다.

  삭제 방지에 대한 자세한 내용은 Amazon DocumentDB 클러스터 삭제 섹션을 참조하세요.

- **`--storage-type standard | iopt1`** - 선택 사항. 기본값: **`standard`**. 클러스터 스토리지 구성입니다. 유효한 값은 `standard` (표준) 또는 `iopt1` (I/O 최적화)입니다.
- **`--master-username`** - 필수입니다. 사용자 인증에 사용되는 사용자 이름입니다.

  마스터 사용자 명명 제약:

  - 길이는 [1-63]자의 영숫자 문자입니다.
  - 첫 번째 문자는 글자이어야 합니다.
  - 데이터베이스 엔진에 포함된 단어는 사용할 수 없습니다.

- **`--master-user-password`** - 선택 사항. 사용자 인증에 사용되는 사용자 암호입니다.

  마스터 암호 제약

  - 길이는 [8-100]자의 인쇄 가능한 ASCII 문자입니다.
  - 다음을 제외한 인쇄 가능한 ASCII 문자를 사용할 수 있습니다.
    - **/**(슬래시)
    - **"**(큰 따옴표)
    - **@**(at 기호)

- **`--manage-master-user-password`** - 선택 사항. Amazon DocumentDB가 Secrets Manager에서 마스터 사용자 암호를 생성하고 해당 수명 주기 동안 이를 관리합니다.

추가 파라미터는 CreateDBCluster를 참조하세요.

를 사용하여 Amazon DocumentDB 클러스터를 시작하려면 AWS CLI

Amazon DocumentDB 클러스터를 생성하려면를 호출합니다`create-db-cluster` AWS CLI. 다음 AWS CLI 명령은 삭제 방지가 활성화된 라는 Amazon DocumentDB 클러스터`sample-cluster`를 생성합니다. 삭제 방지에 대한 자세한 내용은 Amazon DocumentDB 클러스터 삭제을(를) 참조하세요.

또한 `--engine-version`은 기본적으로 최신 주요 엔진 버전으로 설정되는 선택적 매개변수입니다. 현재 기본 엔진 버전은 5.0.0입니다(참고: Amazon DocumentDB 8.0을 사용할 수 있지만 명시적으로 지정해야 함). 새 주요 엔진 버전이 출시되면 최신 주요 엔진 버전을 반영하도록 `--engine-version`의 주요 엔진 버전이 업데이트됩니다. 따라서 프로덕션 워크로드, 특히 스크립팅, 자동화 또는 CloudFormation 템플릿에 의존하는 워크로드의 경우를 의도한 메이저 버전`--engine-version`에 명시적으로 지정합니다.

> **Note**
>
> `db-subnet-group-name` 또는 `vpc-security-group-id`이(가) 지정되지 않은 경우 Amazon DocumentDB는 해당 리전에 대해 기본 서브넷 그룹과 Amazon VPC 보안 그룹을 사용합니다.

Linux, macOS, Unix의 경우:

```
aws docdb create-db-cluster \
      --db-cluster-identifier sample-cluster \
      --engine docdb \
      --engine-version 5.0.0 \
```

```
      --deletion-protection \
      --master-username masteruser \
      --master-user-password password
```

Windows의 경우:

```
aws docdb create-db-cluster ^
      --db-cluster-identifier sample-cluster ^
      --engine docdb ^
      --engine-version 5.0.0 ^
      --deletion-protection ^
      --master-username masteruser ^
      --master-user-password password
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "DBCluster": {
        "StorageEncrypted": false,
        "DBClusterMembers": [],
        "Engine": "docdb",
        "DeletionProtection" : "enabled",
        "ClusterCreateTime": "2018-11-26T17:15:19.885Z",
        "DBSubnetGroup": "default",
        "EngineVersion": "5.0.0",
        "MasterUsername": "masteruser",
        "BackupRetentionPeriod": 1,
        "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster",
        "DBClusterIdentifier": "sample-cluster",
        "MultiAZ": false,
        "DBClusterParameterGroup": "default.docdb5.0",
        "PreferredBackupWindow": "09:12-09:42",
        "DbClusterResourceId": "cluster-KQSGI4MHU4NTDDRVNLNTU7XVAY",
        "PreferredMaintenanceWindow": "tue:04:17-tue:04:47",
        "Port": 27017,
        "Status": "creating",
        "ReaderEndpoint": "sample-cluster.cluster-ro-sfcrlcjcoroz.us-
east-1.docdb.amazonaws.com",
        "AssociatedRoles": [],
        "HostedZoneId": "ZNKXTT8WH85VW",
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-77186e0d",
```

```
                "Status": "active"
            }
        ],
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1c",
            "us-east-1e"
        ],
        "Endpoint": "sample-cluster.cluster-sfcrlcjcoroz.us-east-1.docdb.amazonaws.com"
    }
}
```

클러스터를 생성하는 데 몇 분 정도 걸립니다. AWS Management Console 또는를 사용하여 클러스터의 상태를 AWS CLI 모니터링할 수 있습니다. 자세한 내용은 Amazon DocumentDB 클러스터 상태 모니터링 섹션을 참조하세요.

> **Important**
>
> AWS CLI 를 사용하여 Amazon DocumentDB 클러스터를 생성하면 인스턴스가 생성되지 않습니다. 따라서 기본 인스턴스와 필요한 복제 인스턴스를 명시적으로 생성해야 합니다. 콘솔 또는를 사용하여 인스턴스 AWS CLI 를 생성할 수 있습니다. 자세한 내용은 클러스터에 Amazon DocumentDB 인스턴스 추가 섹션을 참조하세요.

자세한 내용은 Amazon DocumentDB API 참조의 CreateDBCluster를 참조하세요. Amazon DocumentDB

Amazon DocumentDB 클러스터에 대한 설명

Amazon DocumentDB Management Console 또는 AWS CLI 를 사용하여 Amazon DocumentDB 클러스터와 관련된 연결 엔드포인트, 보안 그룹, VPCs 및 파라미터 그룹과 같은 세부 정보를 볼 수 있습니다.

자세한 내용은 다음을 참조하세요.

- Amazon DocumentDB 클러스터 상태 모니터링
- 클러스터 엔드포인트 찾기

Using the AWS Management Console

다음 절차에 따라 콘솔을 사용하여 지정된 Amazon DocumentDB 클러스터의 세부 정보를 확인합니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

3. 클러스터 목록에서 세부 정보를 보려는 클러스터의 이름을 선택합니다. 클러스터에 대한 정보는 다음과 같은 그룹으로 구성됩니다.

   - 요약 - 엔진 버전, 클러스터 상태, 보류 중인 유지 관리 및 파라미터 그룹의 상태를 포함하여 클러스터에 대한 일반 정보입니다.
   - 연결 및 보안 - 연결 섹션에는 mongo 쉘 또는 애플리케이션을 사용하여 이 클러스터에 연결할 연결 엔드포인트가 나열됩니다. 보안 그룹 섹션에는 이 클러스터와 연결된 보안 그룹과 해당 VPC ID 및 설명이 나열됩니다.
   - 구성 - 클러스터 세부 정보 섹션에는 클러스터의 Amazon 리소스 이름(ARN), 엔드포인트 및 파라미터 그룹을 포함하여 클러스터에 대한 세부 정보가 나열됩니다. 또한 클러스터의 백업 정보, 유지 관리 세부 정보, 보안 및 네트워크 설정이 나열됩니다. 클러스터 인스턴스 섹션에는 각 인스턴스의 역할 및 클러스터 파라미터 그룹 상태와 함께 이 클러스터에 속하는 인스턴스가 나열됩니다.
   - 모니터링 - 이 클러스터의 Amazon CloudWatch Logs 측정치. 자세한 내용은 CloudWatch를 사용하여 Amazon DocumentDB 모니터링 섹션을 참조하십시오.
   - 이벤트 및 태그 - 최근 이벤트 섹션에는 이 클러스터의 최근 이벤트가 나열됩니다. Amazon DocumentDB에서는 클러스터, 인스턴스, 스냅샷, 보안 그룹 및 클러스터 파라미터 그룹과 관련된 이벤트 레코드를 유지합니다. 이 정보에는 각 이벤트와 연결된 날짜, 시간 및 메시지가 포함됩니다. 태그 섹션에는 이 클러스터에 연결된 태그가 나열됩니다.

Using the AWS CLI

를 사용하여 Amazon DocumentDB 클러스터의 세부 정보를 보려면 다음 예제와 같이 `describe-db-clusters` 명령을 AWS CLI사용합니다. 자세한 내용은 Amazon DocumentDB 리소스 관리 API 참조의 DescribeDBClusters를 참조하세요. Amazon DocumentDB

> **Note**
>
> 클러스터 및 인스턴스 수명주기 관리와 같은 특정 관리 기능의 경우 Amazon DocumentDB는 Amazon RDS와 공유하는 운영 기술을 활용합니다. `filterName=engine,Values=docdb` 필터 파라미터는 Amazon DocumentDB 클러스터만 반환합니다.

Example

예 1: 모든 Amazon DocumentDB 클러스터를 나열합니다.

다음 AWS CLI 코드는 리전의 모든 Amazon DocumentDB 클러스터에 대한 세부 정보를 나열합니다.

```
aws docdb describe-db-clusters --filter Name=engine,Values=docdb
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBClusters": [
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-1",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        },
        {
```

```
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-2",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        },
        {
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1b",
                "us-east-1a"
            ],
            "BackupRetentionPeriod": 1,
            "DBClusterIdentifier": "sample-cluster-3",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            ...
        }
    ]
}
```

Example

예 2: 지정된 Amazon DocumentDB 클러스터에 대한 모든 세부 정보 나열

다음 AWS CLI 코드는 클러스터에 대한 세부 정보를 나열합니다`sample-cluster`.

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-clusters \
   --filter Name=engine,Values=docdb \
   --db-cluster-identifier sample-cluster
```

Windows의 경우:

```
aws docdb describe-db-clusters ^
```

```
    --filter Name=engine,Values=docdb ^
    --db-cluster-identifier sample-cluster
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBClusters": [
        {
            "AllocatedStorage": 1,
            "AvailabilityZones": [
                "us-east-1c",
                "us-east-1a",
                "us-east-1d"
            ],
            "BackupRetentionPeriod": 2,
            "DBClusterIdentifier": "sample-cluster",
            "DBClusterParameterGroup": "sample-parameter-group",
            "DBSubnetGroup": "default",
            "Status": "available",
            "EarliestRestorableTime": "2023-11-07T22:34:08.148000+00:00",
            "Endpoint": "sample-cluster.node.us-east-1.amazon.com",
            "ReaderEndpoint": "sample-cluster.node.us-east-1.amazon.com",
            "MultiAZ": false,
            "Engine": "docdb",
            "EngineVersion": "5.0.0",
            "LatestRestorableTime": "2023-11-10T07:21:16.772000+00:00",
            "Port": 27017,
            "MasterUsername": "chimeraAdmin",
            "PreferredBackupWindow": "22:22-22:52",
            "PreferredMaintenanceWindow": "sun:03:01-sun:03:31",
            "ReadReplicaIdentifiers": [],
            "DBClusterMembers": [
                {
                    "DBInstanceIdentifier": "sample-instance-1",
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "PromotionTier": 1
                },
                {
                    "DBInstanceIdentifier": "sample-instance-2",
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "PromotionTier": 1
```

```
                },

            ],
            "VpcSecurityGroups": [
                {
                    "VpcSecurityGroupId": "sg-9084c2ec",
                    "Status": "active"
                }
            ],
            "HostedZoneId": "Z06853723JYKYBXTJ49RB",
            "StorageEncrypted": false,
            "DbClusterResourceId": "cluster-T4LGLANHVAPGQYYULWUDKLVQL4",
            "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:sample-
cluster",
            "AssociatedRoles": [],
            "IAMDatabaseAuthenticationEnabled": false,
            "ClusterCreateTime": "2023-11-06T18:05:41.568000+00:00",
            "EngineMode": "provisioned",
            "DeletionProtection": false,
            "HttpEndpointEnabled": false,
            "CopyTagsToSnapshot": false,
            "CrossAccountClone": false,
            "DomainMemberships": [],
            "TagList": [],
            "StorageType": "iopt1",
            "AutoMinorVersionUpgrade": false,
            "NetworkType": "IPV4",
            "IOOptimizedNextAllowedModificationTime":
 "2023-12-07T18:05:41.580000+00:00"
        }
    ]
}
```

Example

예 3: Amazon DocumentDB 클러스터에 대한 특정 세부 정보 나열

를 사용하여 클러스터 세부 정보의 하위 집합을 나열하려면 `describe-db-clusters` 작업에서 나열할 클러스터 멤버를 `--query` 지정하는를 AWS CLI추가합니다. `--db-cluster-identifier` 파라미터는 세부 정보를 표시할 특정 클러스터의 식별자입니다. 쿼리에 대한 자세한 내용은 AWS Command Line Interface 사용 설명서의 [--query 옵션을 사용하여 출력을 필터링하는 방법]을 참조하세요.

다음 예제에서는 Amazon DocumentDB 클러스터의 인스턴스를 나열합니다.

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-clusters \
    --filter Name=engine,Values=docdb \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterMembers]'
```

Windows의 경우:

```
aws docdb describe-db-clusters ^
    --filter Name=engine,Values=docdb ^
    --db-cluster-identifier sample-cluster ^
    --query 'DBClusters[*].[DBClusterMembers]'
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
[
    [
        [
            {
                "DBInstanceIdentifier": "sample-instance-1",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            },
            {
                "DBInstanceIdentifier": "sample-instance-2",
                "IsClusterWriter": false,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            }
        ]
    ]
]
```

Amazon DocumentDB 클러스터 수정

클러스터를 수정하려면 클러스터가 사용 가능 상태에 있어야 합니다. 중지된 클러스터는 수정할 수 없습니다. 클러스터가 중지된 경우, 먼저 클러스터를 시작하고 클러스터가 사용 가능하게 될 때까지 기다린 다음 원하는 대로 수정합니다. 자세한 내용은 Amazon DocumentDB 클러스터 중지 및 시작 섹션을 참조하십시오.

Using the AWS Management Console

다음 절차에 따라 콘솔을 사용하여 특정 Amazon DocumentDB 클러스터를 수정합니다.

Amazon DocumentDB 클러스터를 수정하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택합니다.

3. 클러스터 이름 왼쪽에 있는 버튼을 선택하여 수정할 클러스터를 지정합니다.
4. 작업을 선택한 후 수정을 선택합니다.
5. 클러스터 수정: <cluster-name> 창에서 원하는 항목을 변경합니다. 다음 영역에서 변경할 수 있습니다.

   - 클러스터 사양 - 클러스터의 이름, 보안 그룹 및 자격 증명 관리입니다.
   - 클러스터 스토리지 구성 - 클러스터의 데이터 스토리지 모드입니다. 표준 구성과 I/O 최적화 구성 중에서 선택합니다.
   - 클러스터 옵션 - 클러스터의 포트 및 파라미터 그룹.
   - 백업 - 클러스터의 백업 보존 기간 및 백업 기간.
   - 로그 내보내기 - 감사 또는 프로파일러 로그 내보내기 활성화 또는 비활성화.
   - 유지 관리 - 클러스터의 유지 관리 기간 설정.
   - 삭제 방지 - 클러스터에서 삭제 방지 활성화 또는 비활성화. 삭제 방지 기능은 기본적으로 활성화됩니다.

6. 작업을 마쳤으면 계속을 선택하여 변경 사항의 요약을 확인합니다.
7. 변경 사항이 만족스러우면 클러스터 수정을 선택하여 클러스터를 수정할 수 있습니다. 또는 뒤로 또는 취소를 선택하여 각각 변경 사항을 편집하거나 취소할 수 있습니다.

변경 사항을 적용하는 데 몇 분 정도 걸립니다. 사용 가능 상태인 경우에만 클러스터를 사용할 수 있습니다. 콘솔 또는 AWS CLI를 사용하여 클러스터의 상태를 모니터링할 수 있습니다. 자세한 내용은 Amazon DocumentDB 클러스터 상태 모니터링 섹션을 참조하세요.

Using the AWS CLI

`modify-db-cluster` 작업에 따라 AWS CLI를 사용하여 지정된 클러스터를 수정합니다. 자세한 내용은 Amazon DocumentDB API 참조의 ModifyDBCluster를 참조하세요. Amazon DocumentDB

파라미터

- **`--db-cluster-identifier`** - 필수입니다. 수정하려는 Amazon DocumentDB 클러스터의 식별자입니다.
- **`--backup-retention-period`** - 선택 사항. 자동 백업이 보존되는 일수입니다. 유효한 값은 1–35입니다.
- **`--storage-type`** - 선택 사항. 클러스터 스토리지 구성입니다. 유효한 값은 `standard` (표준) 또는 `iopt1` (I/O 최적화)입니다.
- **`--db-cluster-parameter-group-name`** - 선택 사항. 클러스터에 사용할 클러스터 파라미터 그룹의 이름입니다.
- **`--manage-master-user-password`** - 선택 사항. Amazon DocumentDB가 Secrets Manager에서 마스터 사용자 암호를 생성하고 해당 수명 주기 동안 이를 관리합니다.
- **`--rotate-master-user-password`** - 선택 사항. Secrets Manager는 기존 보안 암호에 대한 새 보안 암호 버전을 생성합니다. 새 버전의 보안 암호에는 새 기본 사용자 암호가 포함됩니다. Amazon RDSAurora는 새 보안 암호 버전의 암호와 일치하도록 클러스터의 기본 사용자 암호를 변경합니다.

  기본 암호를 교체할 때 **`--apply-immediately`** 옵션을 지정해야 합니다.
- **`--master-user-password`** - 선택 사항. 기본 데이터베이스 사용자의 새 암호입니다.

  암호 제약:
  - 길이는 [8-100]자의 인쇄 가능한 ASCII 문자입니다.
  - 다음을 제외한 인쇄 가능한 ASCII 문자를 사용할 수 있습니다.
    - **/**(슬래시)

  - **"**(큰 따옴표)
  - **@**(at 기호)
- **`--new-db-cluster-identifier`** - 선택 사항. 클러스터의 이름을 변경할 때 클러스터의 새 클러스터 식별자입니다. 이 값은 소문자 문자열로 저장됩니다.

  명명 제약 조건:
  - 길이는 [1~63] 글자, 숫자 또는 하이픈입니다.
  - 첫 번째 문자는 글자이어야 합니다.
  - 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.
  - 리전별로 Amazon RDS, Amazon Neptune 및 Amazon DocumentDB의 모든 클러스터 AWS 계정에 대해 고유해야 합니다.
- **`--preferred-backup-window`** - 선택 사항. 자동 백업이 생성되는 일별 시간 범위를 UTC(협정 세계시)로 표시합니다.
  - 형식: `hh24:mm-hh24:mm`
- **`--preferred-maintenance-window`** - 선택 사항. 시스템 유지 관리를 실행할 수 있는 주 단위 시간 범위(UTC)입니다.
  - 형식: `ddd:hh24:mm-ddd:hh24:mm`
  - 유효한 요일: `Sun`, `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`
- **`--deletion-protection`** 또는 **`--no-deletion-protection`** - 선택 사항. 이 클러스터에서 삭제 방지를 활성화해야 하는지 여부. 삭제 방지는 삭제 방지를 비활성화하도록 클러스터를 수정할 때까지 실수로 클러스터가 삭제되는 것을 방지합니다. 자세한 내용은 Amazon DocumentDB 클러스터 삭제 섹션을 참조하십시오.
- **`--apply-immediately`** 또는 **`--no-apply-immediately`** - `--apply-immediately`를 사용하여 즉시 변경합니다. `--no-apply-immediately`를 사용하여 클러스터의 다음 유지 관리 기간에 변경합니다.

Example

다음 코드는 클러스터 `sample-cluster`에 대한 백업 보존 기간을 변경합니다.

Linux, macOS, Unix의 경우:

```
aws docdb modify-db-cluster \
       --db-cluster-identifier sample-cluster \
       --apply-immediately \
```

```
        --backup-retention-period 7
```

Windows의 경우:

```
aws docdb modify-db-cluster ^
        --db-cluster-identifier sample-cluster ^
        --apply-immediately ^
        --backup-retention-period 7
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBCluster": {
        "BackupRetentionPeriod": 7,
        "DbClusterResourceId": "cluster-VDP53QEWST7YHM36TTXOPJT5YE",
        "Status": "available",
        "DBClusterMembers": [
            {
                "PromotionTier": 1,
                "DBClusterParameterGroupStatus": "in-sync",
                "DBInstanceIdentifier": "sample-cluster-instance",
                "IsClusterWriter": true
            }
        ],
        "ReadReplicaIdentifiers": [],
        "AvailabilityZones": [
            "us-east-1b",
            "us-east-1c",
            "us-east-1a"
        ],
        "ReaderEndpoint": "sample-cluster.cluster-ro-ctevjxdlur57.us-
east-1.rds.amazonaws.com",
        "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster",
        "PreferredMaintenanceWindow": "sat:09:51-sat:10:21",
        "EarliestRestorableTime": "2018-06-17T00:06:19.374Z",
        "StorageEncrypted": false,
        "MultiAZ": false,
        "AssociatedRoles": [],
        "MasterUsername": "<your-master-user-name>",
        "DBClusterIdentifier": "sample-cluster",
        "VpcSecurityGroups": [
            {
                "Status": "active",
```

```
                "VpcSecurityGroupId": "sg-77186e0d"
            }
        ],
        "HostedZoneId": "Z2SUY0A1719RZT",
        "LatestRestorableTime": "2018-06-18T21:17:05.737Z",
        "AllocatedStorage": 1,
        "Port": 27017,
        "Engine": "docdb",
        "DBClusterParameterGroup": "default.docdb3.4",
        "Endpoint": "sample-cluster.cluster-ctevjxdlur57.us-
east-1.rds.amazonaws.com",
        "DBSubnetGroup": "default",
        "PreferredBackupWindow": "00:00-00:30",
        "EngineVersion": "3.4",
        "ClusterCreateTime": "2018-06-06T19:25:47.991Z",
        "IAMDatabaseAuthenticationEnabled": false
    }
}
```

변경 사항을 적용하는 데 몇 분 정도 걸립니다. 사용 가능 상태인 경우에만 클러스터를 사용할 수 있습니다. 콘솔 또는 AWS CLI를 사용하여 클러스터의 상태를 모니터링할 수 있습니다. 자세한 내용은 Amazon DocumentDB 클러스터 상태 모니터링 섹션을 참조하세요.

대기 중인 유지 관리 확인

유지 관리에는 대부분 클러스터의 다음 리소스에 대한 업데이트가 포함됩니다.

- 클러스터에 있는 인스턴스의 운영 체제(OS)
- 클러스터의 엔진 버전

Amazon DocumentDB 유지 관리에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB 유지 관리.

Using the AWS Management Console

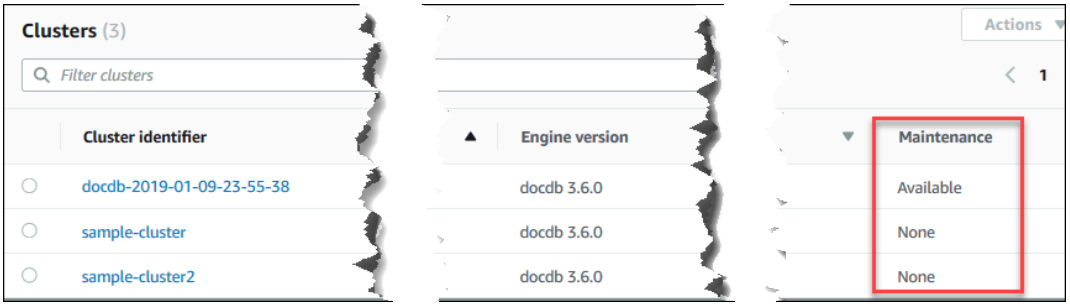
AWS Management Console 를 사용하여 클러스터에 보류 중인 유지 관리가 있는지 확인할 수 있습니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.

2. 탐색 창에서 클러스터를 선택합니다.

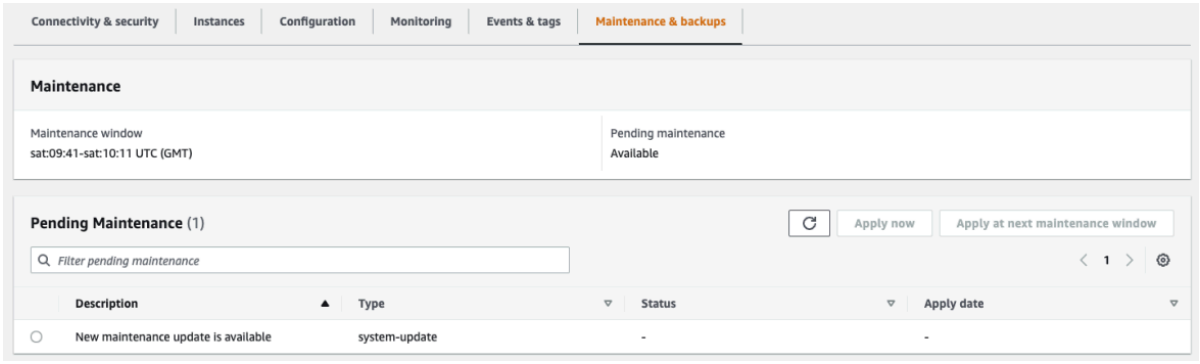
> **Tip**
>
> 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택합니다.

3. 유지 관리 열을 찾아 클러스터에 대기 중인 유지 관리가 있는지 확인합니다.



없음은 클러스터가 최신 엔진 패치 버전을 실행 중이고 인스턴스에 최신 OS 업데이트가 있음을 나타냅니다. 사용 가능은 클러스터에 보류 중인 유지 관리가 있음을 나타내며, 이는 엔진 업그레이드 또는 OS 업데이트가 필요함을 의미할 수 있습니다.

4. 클러스터에 보류 중인 유지 관리가 있는 경우 클러스터 이름을 선택하고 유지 관리 및 백업 탭을 선택하여 보류 중인 유지 관리 작업을 볼 수 있습니다. 보류 중인 유지 관리 섹션에는 보류 중인 유지 관리 작업이 나열됩니다.



Using the AWS CLI

AWS CLI 를 사용하여 다음 파라미터와 함께 `describe-pending-maintenance-actions` 작업을 사용하여 클러스터에 최신 엔진 버전 및 OS 업데이트가 있는지 확인할 수 있습니다.

파라미터

- **--resource-identifier** - 선택 사항. 리소스(클러스터)의 ARN입니다. 이 파라미터를 생략하면 모든 클러스터의 대기 중인 유지 관리 작업이 나열됩니다.
- **--region** - 선택 사항. 이 작업을 실행하려는 AWS 리전입니다. 예: `us-east-1`.

Example

Linux, macOS, Unix의 경우:

```
aws docdb describe-pending-maintenance-actions \
   --resource-identifier arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster \
   --region aa-example-1
```

Windows의 경우:

```
aws docdb describe-pending-maintenance-actions ^
   --resource-identifier arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster ^
   --region aa-example-1
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "PendingMaintenanceActions": [
        {
            "ResourceIdentifier": "arn:aws:rds:aa-
example-1:111122223333:cluster:sample-cluster",
            "PendingMaintenanceActionDetails": [
                {
                    "Description": "New feature",
                    "Action": "system-update",
                    "ForcedApplyDate": "2019-02-25T21:46:00Z",
                    "AutoAppliedAfterDate": "2019-02-25T07:41:00Z",
                    "CurrentApplyDate": "2019-02-25T07:41:00Z"
                }
            ]
```

```
        }
    ]
}
```

Amazon DocumentDB 클러스터에는 다음과 같은 유지 관리 작업 유형이 적용됩니다.

- `system-update` - Amazon DocumentDB 클러스터의 엔진 패치를 업그레이드합니다.

  클러스터에 사용 가능한 대기 중인 엔진 패치 업그레이드가 있는 경우의 단계를 계속 진행합니다클러스터의 엔진 버전에 대한 패치 업데이트 수행.
- `os-upgrade` - 롤링 업그레이드를 사용하여 Amazon DocumentDB 클러스터에 있는 모든 DB 인스턴스의 운영 체제를 업데이트합니다.

  자세한 내용은 Amazon DocumentDB 운영 체제 업데이트 섹션을 참조하세요.

클러스터의 엔진 버전에 대한 패치 업데이트 수행

이 섹션에서는 AWS Management Console 또는를 사용하여 패치 업데이트를 배포하는 방법을 설명합니다 AWS CLI. 패치 업데이트는 동일한 엔진 버전 내의 업데이트입니다(예: 3.6 엔진 버전을 최신 3.6 엔진 버전으로 업데이트). 즉시 업데이트하거나 클러스터의 다음 유지 관리 기간에 업그레이드할 수 있습니다. 엔진 업데이트가 필요한지 확인하려면 대기 중인 유지 관리 확인 섹션을 참조하세요. 업데이트를 적용하면 클러스터에 약간의 가동 중지가 발생합니다.

> **Note**
>
> 메이저 엔진 버전에서 3.6에서 5.0으로 또는 5.0에서 8.0으로 업그레이드하려는 경우 Amazon DocumentDB 인플레이스 주요 버전 업그레이드 또는 단원을 참조하십시오를 사용하여 Amazon DocumentDB 클러스터 업그레이드 AWS Database Migration Service.

클러스터의 엔진 버전에 대한 최신 패치 업데이트를 받으려면 다음과 같은 두 가지 구성 요구 사항이 있습니다.

- 클러스터의 상태가 사용 가능이어야 합니다.
- 클러스터가 이전 엔진 버전을 실행하고 있어야 합니다.

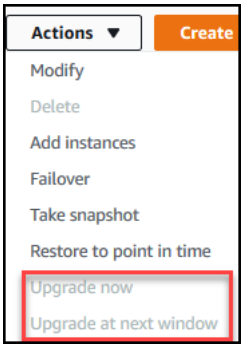
Using the AWS Management Console

다음 절차는 콘솔을 사용하여 클러스터의 엔진 버전에 패치 업데이트를 적용하는 것입니다. 즉시 업데이트하거나 클러스터의 다음 유지 관리 기간 중 업그레이드할 수 있습니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다. 클러스터 목록에서 업그레이드할 클러스터 왼쪽에 있는 버튼을 선택합니다. 클러스터의 상태가 사용 가능이어야 합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

3. 작업 메뉴에서 다음 옵션 중 하나를 선택합니다. 이러한 메뉴 옵션은 선택한 클러스터가 최신 엔진 버전을 실행하고 있지 않은 경우에만 선택할 수 있습니다.



   - 지금 업그레이드 - 업그레이드 프로세스를 즉시 시작합니다. 클러스터가 최신 엔진 버전으로 업그레이드하는 동안에는 클러스터가 잠시 오프라인 상태가 됩니다.
   - 다음에 업그레이드 - 클러스터의 다음 유지 관리 기간 중에 업그레이드 프로세스를 시작합니다. 클러스터가 최신 엔진 버전으로 업그레이드되는 동안에는 클러스터가 잠시 오프라인 상태가 됩니다.
4. 확인 창이 열리면 다음 중 하나를 선택합니다.
   - 업그레이드 - 이전 단계에서 선택한 일정에 따라 클러스터를 최신 엔진 버전으로 업그레이드하려면 선택합니다.
   - 취소 - 클러스터의 엔진 업그레이드를 취소하고 클러스터의 현재 엔진 버전으로 계속하려면 선택합니다.

Using the AWS CLI

다음 파라미터와 함께 AWS CLI 및 `apply-pending-maintenance-action` 작업을 사용하여 클러스터에 패치 업데이트를 적용할 수 있습니다.

파라미터

- **`--resource-identifier`** - 필수입니다. 업그레이드할 Amazon DocumentDB 클러스터의 ARN입니다.
- **`--apply-action`** - 필수입니다. `system-update`를 사용하여 클러스터에 엔진 패치 업데이트를 적용합니다.
- **`--opt-in-type`** - 필수입니다. 다음과 같은 값이 허용됩니다.
  - `immediate` - 유지 관리 작업을 즉시 적용합니다.
  - `next-maintenance` - 다음 유지 관리 기간 중에 유지 관리 작업을 적용합니다.
  - `undo-opt-in` - 기존 `next-maintenance` 옵트인 요청을 취소합니다.

Example

다음 예제에서는 엔진 패치 업데이트를에 적용합니다`sample-cluster`.

Linux, macOS, Unix의 경우:

```
aws docdb apply-pending-maintenance-action \
   --resource-identifier arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster \
   --apply-action system-update \
   --opt-in-type immediate
```

Windows의 경우:

```
aws docdb apply-pending-maintenance-action ^
   --resource-identifier arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster ^
   --apply-action system-update ^
   --opt-in-type immediate
```

이 작업의 출력은 다음과 같습니다.

```
{
```

```
    "ResourcePendingMaintenanceActions": {
        "ResourceIdentifier": "arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster",
        "PendingMaintenanceActionDetails": [
            {
                "CurrentApplyDate": "2019-02-20T20:57:06.904Z",
                "Description": "Bug fixes",
                "ForcedApplyDate": "2019-02-25T21:46:00Z",
                "OptInStatus": "immediate",
                "Action": "system-update",
                "AutoAppliedAfterDate": "2019-02-25T07:41:00Z"
            }
        ]
    }
}
```

클러스터의 운영 체제 업데이트

클러스터 수준 `os-upgrade` 유지 관리 작업을 사용하여 클러스터의 모든 인스턴스에 운영 체제(OS) 업데이트를 적용할 수 있습니다. Amazon DocumentDB는 인스턴스를 한 번에 몇 개씩 롤링 방식으로 업데이트하고 기본 인스턴스를 마지막으로 업데이트하여 장애 조치를 최소화합니다. 업데이트는 클러스터 유지 관리 기간 동안 실행됩니다. 인스턴스가 OS 업데이트를 수신하면 버퍼 캐시가 비워지기 시작하고 해당 인스턴스의 쿼리는 작업 세트가 스토리지 볼륨에서 다시 채워질 때까지 지연 시간이 길어질 수 있습니다.

클러스터에 보류 중인 OS 업데이트가 있는지 확인하려면 섹션을 참조하세요대기 중인 유지 관리 확인. 콘솔 및 CLI 절차를 포함한 OS 업데이트에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB 운영 체제 업데이트.

Amazon DocumentDB 클러스터 중지 및 시작

Amazon DocumentDB 클러스터를 중지하고 시작하면 개발 및 테스트 환경 비용을 관리하는 데 도움이 됩니다. 필요하지 않을 경우 Amazon DocumentDB를 사용할 때마다 클러스터 및 인스턴스를 생성하고 삭제하는 대신 클러스터의 모든 인스턴스를 일시적으로 중지할 수 있습니다. 그런 다음 테스트를 재개할 때 다시 시작할 수 있습니다.

주제

- 클러스터의 중지 및 시작 개요
- 중지된 클러스터에서 수행할 수 있는 작업

클러스터의 중지 및 시작 개요

Amazon DocumentDB 클러스터가 필요하지 않은 기간에는 이 클러스터의 모든 인스턴스를 한번에 중지할 수 있습니다. 그런 다음 사용해야 할 때는 언제든지 클러스터를 다시 시작할 수 있습니다. 시작 및 중지를 사용하면 개발, 테스트 또는 연속 가용성을 필요로 하지 않는 유사한 활동에 사용되는 클러스터의 설정 및 해제 프로세스가 간소화됩니다. 클러스터에 있는 인스턴스 수에 관계없이 단일 작업 AWS CLI 으로 AWS Management Console 또는를 사용하여 클러스터를 중지하고 시작할 수 있습니다.

클러스터가 중지되는 동안 클러스터 스토리지 볼륨은 변경되지 않습니다. 지정된 보존 기간 내에는 스토리지, 수동 스냅샷 및 자동 백업 스토리지에 대한 비용만 청구됩니다. 인스턴스 시간에 대해서는 요금이 부과되지 않습니다 Amazon DocumentDB는 필요한 유지 관리 업데이트에 뒤처지지 않도록 7일 후에 클러스터를 자동으로 시작합니다. 7일 후에 클러스터가 시작되면 클러스터의 인스턴스 비용이 다시 청구됩니다. 클러스터가 중지된 동안에는 쿼리하려면 인스턴스가 사용 가능한 상태여야 하므로 스토리지 볼륨을 쿼리할 수 없습니다.

Amazon DocumentDB 클러스터가 중지되면 클러스터도 인스턴스도 어떤 식으로든 수정할 수 없습니다. 여기에는 인스턴스 추가 또는 제거, 클러스터 삭제도 포함됩니다.

Using the AWS Management Console

다음 절차에서는 사용 가능한 상태인 하나 이상의 인스턴스가 있는 클러스터를 중지하거나, 중지된 클러스터를 시작하는 방법을 보여 줍니다.

Amazon DocumentDB 클러스터 중지 및 시작 방법

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

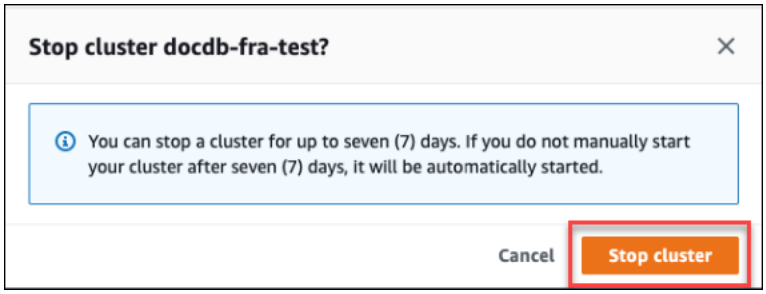
   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택합니다.

3. 클러스터 목록에서 중지하거나 시작하려는 클러스터 이름의 왼쪽에 있는 버튼을 선택합니다.
4. 작업을 선택한 다음 클러스터에서 수행하려는 작업을 선택합니다.
   - 클러스터를 중지하고 다음 작업을 수행합니다.

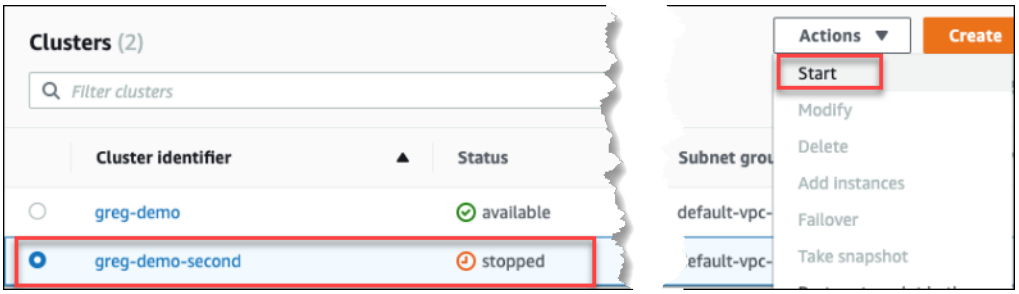
     a. 중지를 선택합니다.

중지 작업은 장애 조치 메커니즘 활성화를 피하기 위해 복제본 인스턴스를 먼저 중지한 후 기본 인스턴스를 중지합니다.

b. 확인 대화 상자에서 클러스터 중지를 선택하여 클러스터 중지를 확인하거나, 클러스터를 계속 실행하려면 취소를 선택합니다.



- 클러스터를 시작하려면 클러스터가 중지된 상태에서 시작을 선택합니다.



5. 클러스터 및 인스턴스 상태를 모니터링합니다. 클러스터를 시작한 경우 클러스터 및 해당 인스턴스를 사용 가능할 때 클러스터 사용을 다시 시작할 수 있습니다. 자세한 내용은 클러스터 상태 확인 섹션을 참조하십시오.

Using the AWS CLI

다음 코드 예제에서는 사용 가능한 상태인 하나 이상의 인스턴스가 있는 클러스터를 중지하거나, 중지된 클러스터를 시작하는 방법을 보여 줍니다.

를 사용하여 사용 가능한 인스턴스가 하나 이상인 클러스터를 중지하려면 `stop-db-cluster` 작업을 AWS CLI사용합니다. 중지된 클러스터를 시작하려면 `start-db-cluster` 작업을 사용합니다. 두 작업은 `--db-cluster-identifier` 파라미터를 사용합니다.

파라미터:

- **`--db-cluster-identifier`** - 필수입니다. 중지하거나 시작할 클러스터의 이름.

Example-를 사용하여 클러스터를 중지하려면 AWS CLI

다음 코드는 클러스터 `sample-cluster`를 중지합니다. 클러스터에는 사용 가능한 상태인 인스턴스가 하나 이상 있어야 합니다.

Linux, macOS, Unix의 경우:

```
aws docdb stop-db-cluster \
   --db-cluster-identifier sample-cluster
```

Windows의 경우:

```
aws docdb stop-db-cluster ^
   --db-cluster-identifier sample-cluster
```

Example-를 사용하여 클러스터를 시작하려면 AWS CLI

다음 코드는 클러스터 `sample-cluster`를 시작합니다. 클러스터는 현재 중지된 상태여야 합니다.

Linux, macOS, Unix의 경우:

```
aws docdb start-db-cluster \
   --db-cluster-identifier sample-cluster
```

Windows의 경우:

```
aws docdb start-db-cluster ^
   --db-cluster-identifier sample-cluster
```

중지된 클러스터에서 수행할 수 있는 작업

Amazon DocumentDB 클러스터가 중지되어 있는 동안, 지정된 자동 백업 보존 기간 내에서 원하는 시점으로 특정 시점 복원을 수행할 수 있습니다. 특정 시점으로 복원에 대한 세부 정보는 특정 시점으로 복원 섹션을 참조하세요.

클러스터가 중지되는 동안 Amazon DocumentDB 클러스터의 구성 또는 해당 인스턴스를 수정할 수 없습니다. 또한 클러스터에서 인스턴스를 추가 또는 제거할 수 없으며, 연결된 인스턴스가 있는 경우에도 클러스터를 삭제할 수 없습니다. 이러한 관리 작업을 수행하기 전에 클러스터를 시작해야 합니다.

Amazon DocumentDB는 다시 시작한 후에만 중지된 클러스터에 예약 유지 관리를 적용합니다. 7일 후에 Amazon DocumentDB는 중지된 클러스터를 자동으로 시작하므로 유지 관리 상태에서 너무 늦어지지 않습니다. 클러스터가 다시 시작되면 클러스터의 인스턴스 비용이 다시 청구됩니다.

클러스터가 중지되는 동안 Amazon DocumentDB는 자동화된 백업을 수행하지 않으며 백업 보존 기간도 연장하지 않습니다.

Amazon DocumentDB 클러스터 삭제

AWS Management Console 또는를 사용하여 Amazon DocumentDB 클러스터를 삭제할 수 있습니다 AWS CLI. 클러스터를 삭제하려면 클러스터가 사용 가능한 상태여야 하며 클러스터와 연결된 인스턴스가 없어야 합니다. 클러스터가 중지된 경우, 먼저 클러스터를 시작하고 클러스터가 사용 가능하게 될 때까지 기다린 다음 클러스터를 삭제합니다. 자세한 내용은 Amazon DocumentDB 클러스터 중지 및 시작 섹션을 참조하십시오.

삭제 방지

실수로 인한 삭제로부터 클러스터를 보호하려면 삭제 방지를 활성화할 수 있습니다. 콘솔을 사용하여 클러스터를 생성할 때 기본적으로 삭제 방지가 활성화됩니다. 하지만 AWS CLI를 사용하여 클러스터를 생성하는 경우 삭제 방지가 기본적으로 비활성화됩니다.

Amazon DocumentDB는 콘솔 또는 AWS CLI중 어느 곳에서 삭제 작업을 수행하든 간에 클러스터에 대한 삭제 방지를 강제 시행합니다. 삭제 방지가 활성화되어 있으면 클러스터를 삭제할 수 없습니다. 삭제 방지가 활성화된 클러스터를 삭제하려면 먼저 클러스터를 수정하고 삭제 방지를 비활성화해야 합니다.

삭제 방지가 활성화된 클러스터가 있는 콘솔을 사용할 경우 클러스터의 마지막 인스턴스를 삭제하면 클러스터도 삭제되므로 해당 인스턴스를 삭제할 수 없습니다. AWS CLI를 사용하여 삭제 방지된 클러스터의 마지막 인스턴스를 삭제할 수 있습니다. 하지만 클러스터 자체는 여전히 존재하고 데이터는 보

존됩니다. 클러스터의 새 인스턴스를 생성하여 데이터에 액세스할 수 있습니다. 삭제 방지 활성화 및 비활성화에 대한 자세한 내용은 다음을 참조하세요.

- Amazon DocumentDB 클러스터 생성
- Amazon DocumentDB 클러스터 수정

Using the AWS Management Console

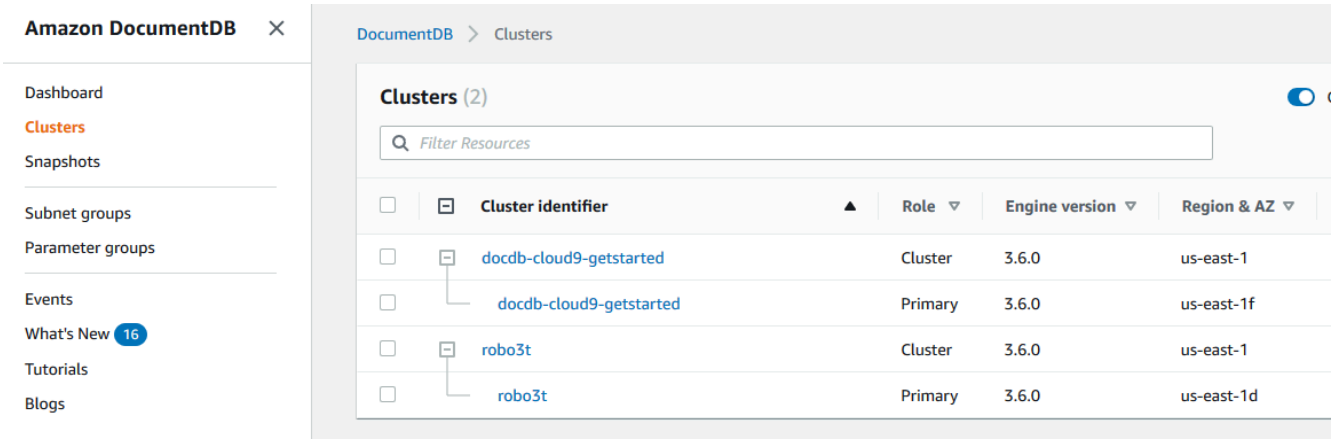
를 사용하여 클러스터를 삭제하려면 AWS Management Console삭제 방지 기능을 비활성화해야 합니다.

클러스터에 대해 삭제 방지를 활성화했는지 여부를 확인하려면 다음과 같이 하세요.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택합니다.

3. 참고로 클러스터 탐색 상자의 클러스터 식별자 열에는 클러스터와 인스턴스가 모두 표시됩니다. 인스턴스는 다음 스크린샷과 마찬가지로 클러스터 아래에 나열됩니다.



4. 클러스터의 이름을 선택하고 구성 탭을 선택합니다. 클러스터 세부 정보 섹션에서 삭제 보호를 찾습니다. 삭제 방지가 활성화된 경우 클러스터를 수정하여 삭제 방지를 비활성화합니다. 클러스터 수정에 대한 자세한 내용은 Amazon DocumentDB 클러스터 수정 섹션을 참조하세요.

삭제 방지가 비활성화된 후에는 클러스터를 삭제할 수 있습니다.

클러스터를 삭제하려면 다음과 같이 하세요.

1. 탐색 창에서 클러스터를 선택합니다.
2. 클러스터 식별자 열 아래에 나열된 인스턴스를 확인하여 클러스터에 인스턴스가 있는지 확인합니다. 클러스터를 삭제하려면 모든 인스턴스를 삭제해야 합니다. 자세한 내용은 Amazon DocumentDB 인스턴스 삭제 섹션을 참조하십시오.
3. 클러스터에 인스턴스가 있는지 여부에 따라 다음 단계 중 하나를 수행합니다.
   - 클러스터에 인스턴스가 없는 경우 클러스터 이름 왼쪽에 있는 단추를 선택하고 작업을 선택합니다. 드롭다운 메뉴에서 삭제를 선택합니다. <cluster-name> 삭제 대화 상자를 완료한 다음 삭제를 선택합니다.
   - 클러스터에 인스턴스가 하나 이상 있는 경우 다음을 수행합니다.
     a. 탐색 창에서 클러스터를 선택합니다.
     b. 클러스터 이름 왼쪽에 있는 확인란을 선택하여 클러스터의 각 인스턴스를 삭제합니다. 작업을 선택한 후 삭제를 선택합니다. <cluster-name> 삭제 대화 상자를 완료한 다음 삭제를 선택합니다.

        마지막 인스턴스를 삭제할 경우 클러스터도 삭제됩니다. 인스턴스 삭제에 대한 자세한 내용은 Amazon DocumentDB 인스턴스 삭제 섹션을 참조하세요.

클러스터를 삭제하는 데 몇 분 정도 걸립니다. 클러스터 상태를 모니터링하려면 Amazon DocumentDB 클러스터 상태 모니터링 섹션을 참조하세요.

Using the AWS CLI

인스턴스가 연결되어 있는 클러스터를 삭제할 수 없습니다. 클러스터와 연결된 인스턴스를 확인하려면 `describe-db-clusters` 명령을 실행하고 클러스터의 모든 인스턴스를 삭제합니다. 그런 다음 필요한 경우 클러스터에서 삭제 방지를 비활성화하고 마지막으로 클러스터를 삭제합니다.

1. 먼저 클러스터의 모든 인스턴스를 삭제합니다.

   삭제해야 할 인스턴스를 결정하려면 다음 명령을 실행합니다.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
```

```
    --query 'DBClusters[*].
[DBClusterIdentifier,DBClusterMembers[*].DBInstanceIdentifier]'
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
[
    [
        "sample-cluster",
        [
            "sample-instance-1",
            "sample-instance-2"
        ]
    ]
]
```

삭제하려는 클러스터에 인스턴스가 있는 경우 아래와 같이 인스턴스를 삭제합니다.

```
aws docdb delete-db-instance \
    --db-instance-identifier sample-instance
```

2. 다음으로 삭제 방지를 비활성화합니다.

AWS CLI 를 사용하여 클러스터의 모든 인스턴스를 삭제해도 클러스터는 삭제되지 않습니다. 클러스터를 삭제해야 하더라도 삭제 방지가 비활성화된 경우에만 삭제할 수 있습니다.

클러스터에 삭제 방지가 활성화되었는지 확인하려면 다음 명령을 실행합니다.

> **Tip**
>
> 모든 Amazon DocumentDB 클러스터의 삭제 방지 상태를 보려면 `--db-cluster-identifier` 파라미터를 생략합니다.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterIdentifier,DeletionProtection]'
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
[
    [
        "sample-cluster",
        "true"
    ]
]
```

클러스터에 삭제 방지가 활성화된 경우 클러스터를 수정하여 삭제 방지를 비활성화합니다. 클러스터에서 삭제 방지를 비활성화하려면 다음 명령을 실행합니다.

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --no-deletion-protection \
    --apply-immediately
```

3. 마지막으로 클러스터를 삭제합니다.

   삭제 방지가 비활성화된 후에는 클러스터를 삭제할 수 있습니다. 클러스터를 삭제하려면 다음 파라미터와 함께 `delete-db-cluster` 작업을 사용합니다.

   - **`--db-cluster-identifier`** - 필수입니다. 삭제하려는 클러스터의 식별자입니다.
   - **`--final-db-snapshot-identifier`** - 선택 사항. 최종 스냅샷을 원하는 경우 최종 스냅샷의 이름과 함께 이 파라미터를 포함해야 합니다. `--final-db-snapshot-identifier` 또는 `--skip-final-snapshot`를 포함시켜야 합니다.

     명명 제약 조건:

     - 길이는 [1~63] 글자, 숫자 또는 하이픈입니다.
     - 첫 번째 문자는 글자이어야 합니다.
     - 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.
     - 리전별로 Amazon RDS, Amazon Neptune 및 Amazon DocumentDB의 모든 클러스터 AWS 계정에 대해 고유해야 합니다.
   - **`--skip-final-snapshot`** - 선택 사항. 클러스터를 삭제하기 전에 최종 스냅샷을 생성하지 않으려는 경우에만 이 파라미터를 사용하세요. 기본 설정은 최종 스냅샷을 생성하는 것입니다. `--final-db-snapshot-identifier` 또는 `--skip-final-snapshot`를 포함시켜야 합니다.

다음 AWS CLI 코드는 최종 스냅샷이 `sample-cluster` 있는 클러스터를 삭제합니다. 클러스터와 연결된 인스턴스가 있거나 삭제 방지가 활성화된 경우 작업이 실패합니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --final-db-snapshot-identifier sample-cluster-final-snapshot
```

Windows의 경우:

```
aws docdb delete-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --final-db-snapshot-identifier sample-cluster-final-snapshot
```

Example

다음 AWS CLI 코드는 최종 스냅샷을 생성`sample-cluster`하지 않고 클러스터를 삭제합니다.

Linux, macOS, Unix의 경우:

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --skip-final-snapshot
```

Windows의 경우:

```
aws docdb delete-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --skip-final-snapshot
```

`delete-db-cluster` 연산은 삭제하려는 클러스터를 출력합니다.

클러스터를 삭제하는 데 몇 분 정도 걸립니다. 클러스터 상태를 모니터링하려면 클러스터 상태 모니터링 섹션을 참조하세요.

Amazon DocumentDB 클러스터 규모 조정

Amazon DocumentDB를 사용하면 필요에 따라 클러스터의 스토리지와 컴퓨팅을 조정할 수 있습니다. 이 섹션에서는 스토리지 조정, 인스턴스 조정 및 읽기 조정을 사용하여 Amazon DocumentDB 클러스터 및 인스턴스의 성능 및 확장을 관리하는 방법을 설명합니다.

주제

- 스토리지 조정
- 인스턴스 조정
- 읽기 확장
- 쓰기 조정

스토리지 조정

Amazon DocumentDB 스토리지는 클러스터 볼륨에 저장된 데이터에 따라 자동 조정됩니다. 데이터가 증가함에 따라 클러스터 볼륨 스토리지는 10GiB 단위로 증가하며 엔진 버전 8.0 이상의 경우 최대 256TiB(이전 엔진 버전의 경우 128TiB)까지 증가합니다.

인스턴스 조정

필요에 따라 클러스터의 인스턴스마다 인스턴스 클래스를 수정하여 Amazon DocumentDB 클러스터의 크기를 조정할 수 있습니다. Amazon DocumentDB는 Amazon DocumentDB에 최적화된 여러 인스턴스 클래스를 지원합니다.

자세한 내용은 Amazon DocumentDB 인스턴스 수정 섹션을 참조하십시오.

읽기 확장

클러스터에서 Amazon DocumentDB 복제본을 최대 15개까지 생성하여 Amazon DocumentDB 클러스터에 대한 읽기 확장이 가능합니다. 각 Amazon DocumentDB 복제본은 복제본 지연을 최소화하여 클러스터 볼륨에서 동일한 데이터를 반환합니다. 일반적으로 이 지연 시간은 기본 인스턴스가 업데이트를 적용한 후 100밀리초 미만입니다. 읽기 트래픽이 증가하면 Amazon DocumentDB 복제본을 추가 생성하여 직접 연결함으로써 클러스터의 읽기 부하를 분산시키는 것도 가능합니다. Amazon DocumentDB 복제본의 인스턴스 클래스가 기본 인스턴스의 DB 인스턴스 클래스와 같을 필요는 없습니다.

자세한 내용은 클러스터에 Amazon DocumentDB 인스턴스 추가 섹션을 참조하세요.

Amazon DocumentDB로 스케일을 읽으려면 클러스터에 복제본 세트로 연결하고 드라이버의 기본 제공 읽기 기본 설정 기능을 사용하여 복제본 인스턴스에 읽기를 배포합니다. 자세한 내용은 Amazon DocumentDB에 복제본 세트로 연결 섹션을 참조하세요.

쓰기 조정

클러스터의 기본 인스턴스 크기를 늘려 Amazon DocumentDB 클러스터의 쓰기 용량을 확장할 수 있습니다. 이 섹션에서는 필요에 따라 클러스터의 기본 인스턴스를 확장하는 두 가지 방법을 설명합니다. 첫 번째 옵션은 애플리케이션에 미치는 영향을 최소화하도록 노력하지만 더 많은 단계를 완료해야 합니다. 두 번째 옵션은 단계가 적어 더 간단하지만 애플리케이션에 더 많은 잠재적 영향을 미칠 수 있다는 단점이 있습니다.

애플리케이션에 따라 둘 중에서 더 적합한 방법을 선택할 수 있습니다. 사용 가능한 인스턴스 크기 및 비용에 대한 자세한 내용은 Amazon DocumentDB 요금 페이지를 참조하세요.

1. 고가용성 및 성능 최적화 - 복제본 집합 모드로 클러스터에 연결하는 경우(권장) 기본 인스턴스를 확장할 때 다음 프로세스를 사용하여 애플리케이션에 미치는 영향을 최소화할 수 있습니다. 이 방법을 사용하면 클러스터가 필요한 고가용성 수준 이상으로 유지되며 읽기 조정 대상이 현재 위치에서 업데이트되지 않고 인스턴스로 클러스터에 추가되므로 영향이 최소화됩니다.

   a. 클러스터에 더 큰 인스턴스 유형의 복제본을 하나 이상 추가합니다(??? 참조). 모든 복제본이 기본 인스턴스 유형과 동일하거나 더 큰 인스턴스 유형인지 확인합니다. 이렇게 하면 더 작은 인스턴스 유형으로 장애 조치되어 쓰기 성능이 의도치 않게 저하되는 것을 방지할 수 있습니다. 대부분의 경우 이는 클러스터의 인스턴스 수를 일시적으로 두 배로 늘리고 조정이 완료된 후 더 작은 복제본을 제거하는 것을 의미합니다.

   b. 모든 새 복제본의 장애 조치 계층을 우선 순위 0으로 설정하여 더 작은 인스턴스 유형의 복제본이 가장 높은 장애 조치 우선 순위를 갖도록 합니다. 자세한 내용은 ??? 섹션을 참조하십시오.

   c. 수동 장애 조치를 시작합니다. 그러면 새 복제본 중 하나가 기본 인스턴스로 승격됩니다. 자세한 내용은 ??? 섹션을 참조하십시오.

      > **Note**
      >
      > 이렇게 하면 클러스터에 약 30초의 가동 중지 시간이 발생합니다. 이에 따라 적절한 계획을 수립해야 합니다.

   d. 새 기본 인스턴스보다 작은 인스턴스 유형의 모든 복제본을 클러스터에서 제거합니다.

e. 모든 인스턴스의 장애 조치 계층을 동일한 우선 순위로 다시 설정합니다(일반적으로 다시 1로 설정됨).

예를 들어 현재 3개의 `r5.large` 인스턴스(하나의 기본 인스턴스와 두 개의 복제본)가 포함된 클러스터가 있고 `r5.xlarge` 인스턴스 유형으로 확장하려는 경우, 먼저 클러스터에 `r5.xlarge` 복제본 인스턴스 3개를 추가하고 새 `r5.xlarge` 복제본의 장애 조치 계층을 0으로 설정한 다음 수동 장애 조치를 시작합니다(애플리케이션에 30초 정도의 가동 중지 시간이 발생함). 장애 조치가 완료된 후 클러스터에서 3개의 `r5.large` 인스턴스를 모두 제거하면 클러스터가 `r5.xlarge` 인스턴스로 확장됩니다.

비용을 최적화할 수 있도록 Amazon DocumentDB 인스턴스 요금은 인스턴스 생성, 수정 또는 삭제 같은 청구 가능한 상태 변경에 따라 1초 단위로 청구되며 최소 10분의 요금이 부과됩니다. 자세한 내용은 모범 사례 설명서의 비용 최적화 섹션을 참조하세요.

2. 단순성을 위한 최적화 - 이 접근 방식은 단순성을 위해 최적화됩니다. 클러스터를 확장하거나 축소하지는 않지만 일시적으로 읽기 용량을 줄일 수 있습니다.

   복제본의 인스턴스 클래스를 변경하면 해당 인스턴스가 몇 초에서 30초 미만으로 짧은 기간 동안 요청을 처리하지 못할 수 있습니다. 복제본 세트 모드(권장)로 클러스터에 연결하는 경우 조정 작업 중에 읽기 용량이 복제본 1개만큼 줄어듭니다(예: 3노드 클러스터의 경우 66%, 4노드 클러스터의 경우 75% 용량 등).

   a. 클러스터의 복제본 인스턴스 중 하나를 확장합니다. 자세한 내용은 인스턴스 클래스 관리 섹션을 참조하십시오.

   b. 인스턴스를 사용할 수 있을 때까지 기다리세요(Amazon DocumentDB 인스턴스 상태 모니터링 참조).

      > **Note**
      >
      > 이렇게 하면 클러스터에 약 30초의 가동 중지 시간이 발생합니다. 이에 따라 적절한 계획을 수립해야 합니다.

   c. 모든 복제본 인스턴스가 하나씩 확장될 때까지 1단계와 2단계를 계속 실행합니다.

   d. 수동 장애 조치 시작. 이렇게 하면 복제본 중 한 개가 기본 인스턴스로 승격됩니다. 자세한 내용은 Amazon DocumentDB 장애 조치 섹션을 참조하십시오.

> **Note**
>
> 이로 인해 클러스터에 최대 30초의 가동 중지 시간이 발생하지만 대개 그보다 시간이 덜 걸립니다. 이에 따라 적절한 계획을 수립해야 합니다.

e. 이전의 기본(지금은 복제본) 인스턴스를 확장하세요.

Amazon DocumentDB 클러스터에 대한 볼륨 복제

Amazon DocumentDB 복제를 사용하면 동일한 Amazon DocumentDB 클러스터 볼륨을 사용하고 원본과 동일한 데이터를 갖는 새 클러스터를 생성할 수 있습니다. 이 프로세스는 빠르고 비용 효율적으로 진행되도록 설계되었습니다. 연결된 데이터 볼륨이 있는 새 클러스터를 복제본이라고 합니다. 복제본 생성은 스냅샷 복원과 같은 다른 기술을 사용하여 데이터를 물리적으로 복사하는 것보다 빠르고 공간 효율적입니다.

Amazon DocumentDB는 프로비저닝된 Amazon DocumentDB 클러스터에서 Amazon DocumentDB 프로비저닝된 복제본을 생성할 수 있도록 지원합니다. 소스와 다른 배포 구성을 사용하여 복제를 생성하면 소스 Amazon DocumentDB 엔진의 최신 버전을 사용하여 복제본이 생성됩니다.

Amazon DocumentDB 클러스터에서 복제본을 생성하면 소스 Amazon DocumentDB 클러스터를 소유한 것과 동일한 계정인 AWS 계정에 복제본이 생성됩니다.

주제

- Amazon DocumentDB 복제 개요
- Amazon DocumentDB 복제 사용 시 참고 사항
- Amazon DocumentDB 복제의 작동 방식
- Amazon DocumentDB 복제본 생성

Amazon DocumentDB 복제 개요

Amazon DocumentDB는 기록 중 복사(Copy-on-Write) 프로토콜을 사용하여 복제본을 생성합니다. 이 메커니즘은 최소한의 추가 공간을 사용하여 초기 복제를 만듭니다. 복제가 처음 생성되면 Amazon DocumentDB는 소스 DB 클러스터와 새로운(복제된) Amazon DocumentDB 클러스터에서 사용하는 데이터의 단일 복사본을 유지합니다. 소스 Amazon DocumentDB 클러스터 또는 Amazon DocumentDB 클러스터 복제가 Amazon DocumentDB 스토리지 볼륨의 데이터를 변경한 경우에만

추가 스토리지가 할당됩니다. 기록 중 복사(Copy-on-Write) 프로토콜에 대한 자세한 내용은 Amazon DocumentDB 복제의 작동 방식 섹션을 참조하세요.

Amazon DocumentDB 복제 작업은 데이터 손상 위험 없이 프로덕션 데이터를 사용하여 테스트 환경을 신속하게 설정하는 데 특히 유용합니다. 다음과 같은 여러 유형의 애플리케이션에 복제본을 사용할 수 있습니다.

- 잠재적 변경 사항(예: 스키마 변경 및 파라미터 그룹 변경)을 실험하여 모든 영향을 평가합니다.
- 데이터 내보내기 또는 복제본에서 분석 쿼리 실행과 같은 워크로드 집약적인 작업을 수행하는 경우
- 개발, 테스트 또는 기타 용도로 프로덕션 DB 클러스터의 복사본을 생성합니다.

동일한 Amazon DocumentDB 클러스터에서 둘 이상의 복제본을 생성할 수 있습니다. 다른 복제본에서 여러 복제본을 생성할 수도 있습니다.

Amazon DocumentDB 복제본을 생성한 후 Amazon DocumentDB 인스턴스를 원본 Amazon DocumentDB 클러스터와 다르게 구성할 수 있습니다. 예를 들어 소스 프로덕션 Amazon DocumentDB 클러스터와 동일한 고가용성 요구 사항을 충족하기 위해 개발 목적으로 복제본이 필요하지 않을 수 있습니다. 이 경우 Amazon DocumentDB 클러스터에서 사용하는 여러 DB 인스턴스가 아닌 단일 Amazon DocumentDB 인스턴스로 복제본을 구성할 수 있습니다.

복제본을 테스트, 개발 또는 다른 용도로 사용한 후 삭제할 수 있습니다.

Amazon DocumentDB 복제 사용 시 참고 사항

Amazon DocumentDB 복제는 다음과 같은 제한 사항이 있습니다.

- AWS 리전에서 허용하는 최대 DB 클러스터 개수까지 원하는 만큼 복제본을 생성할 수 있습니다. 그러나 15개의 복제본을 생성한 후에는 다음 복제가 전체 복제본이 됩니다. 시점 복구와 같은 복제 작업이 수행됩니다.
- 소스 Amazon DocumentDB 클러스터와 다른 AWS 리전에서는 복제본을 생성할 수 없습니다.
- DB 인스턴스가 없는 Amazon DocumentDB 클러스터에서 복제본을 생성할 수 없습니다. 하나 이상의 DB 인스턴스가 있는 Amazon DocumentDB 클러스터만 복제할 수 있습니다.
- 복제본을 Amazon DocumentDB 클러스터의 가상 프라이빗 클라우드(VPC)와 다른 가상 프라이빗 클라우드(VPC)에 생성할 수 있습니다. 이렇게 하면 VPC의 서브넷을 동일한 가용 영역에 매핑해야 합니다.

Amazon DocumentDB 복제의 작동 방식

Amazon DocumentDB 복제는 Amazon DocumentDB 클러스터의 스토리지 계층에서 작동합니다. 이는 Amazon DocumentDB 스토리지 볼륨을 지원하는 기본 내구 미디어 측면에서 빠르고 공간 효율적인 기록 중 복사(copy-on-write) 프로토콜을 사용합니다. Amazon DocumentDB 클러스터 볼륨에 대한 자세한 내용은 Amazon DocumentDB 클러스터 관리에서 확인할 수 있습니다.

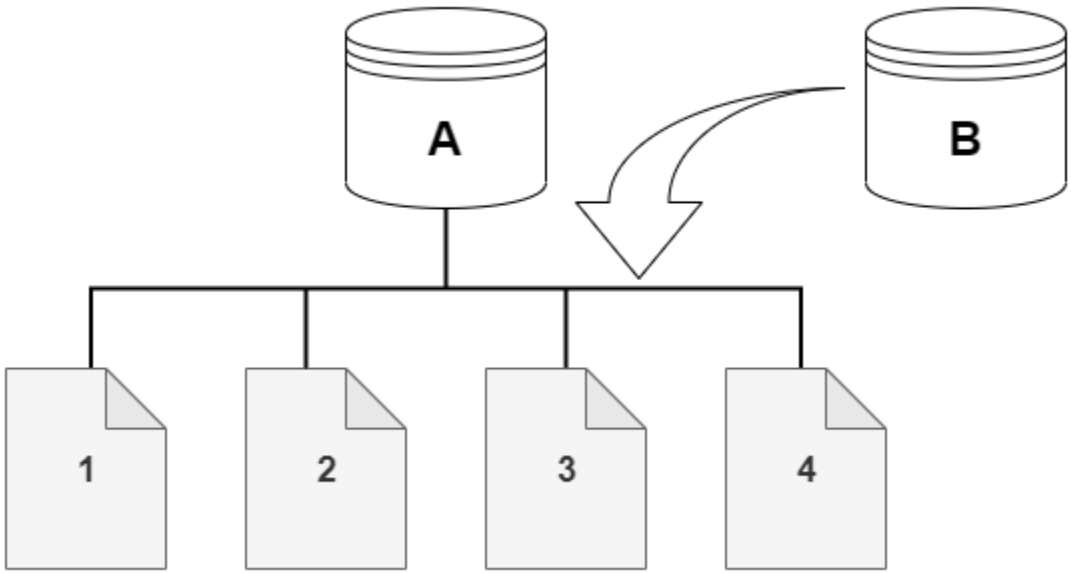
주제

- 기록 중 복사(copy-on-write) 이해
- 원본 클러스터 볼륨 삭제

기록 중 복사(copy-on-write) 이해

Amazon DocumentDB 클러스터는 기본 Amazon DocumentDB 스토리지 볼륨의 페이지에 데이터를 저장합니다.

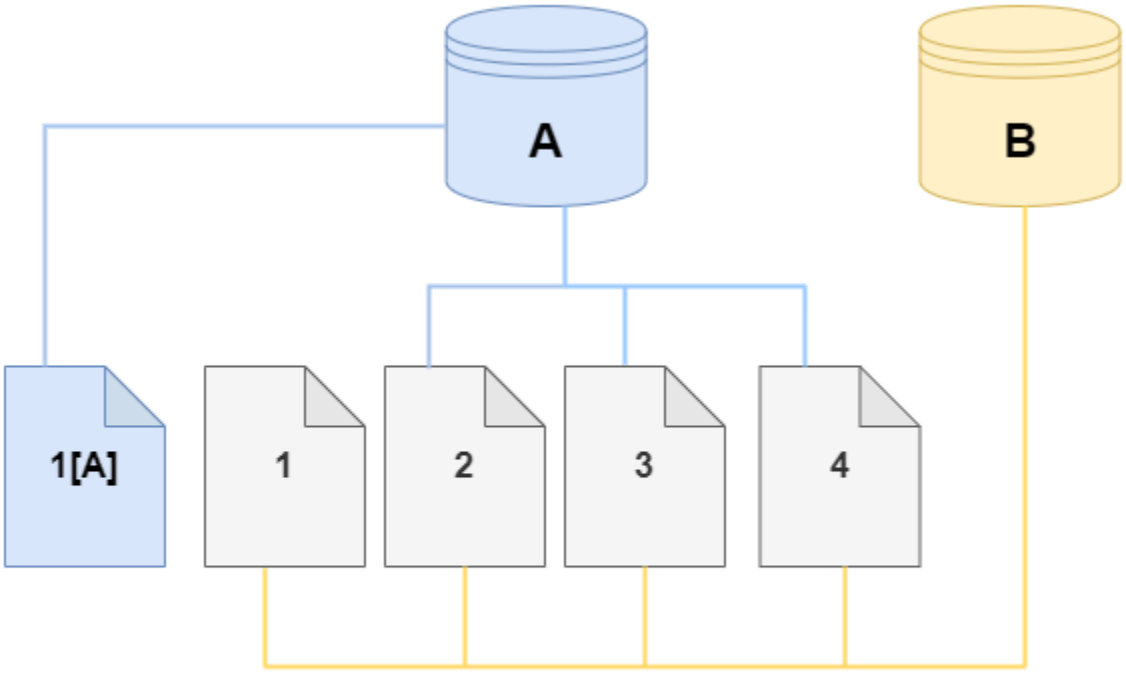
예를 들어, 다음 다이어그램에서 데이터 페이지(1, 2, 3, 4)가 네 개인 Amazon DocumentDB 클러스터(A)를 찾을 수 있습니다. 복제본 B가 Amazon DocumentDB 클러스터에서 생성된다고 가정해 보겠습니다. 복제본이 생성되면 데이터가 복사되지 않습니다. 대신 복제본은 소스 Amazon DocumentDB 클러스터와 동일한 페이지 집합을 가리킵니다.



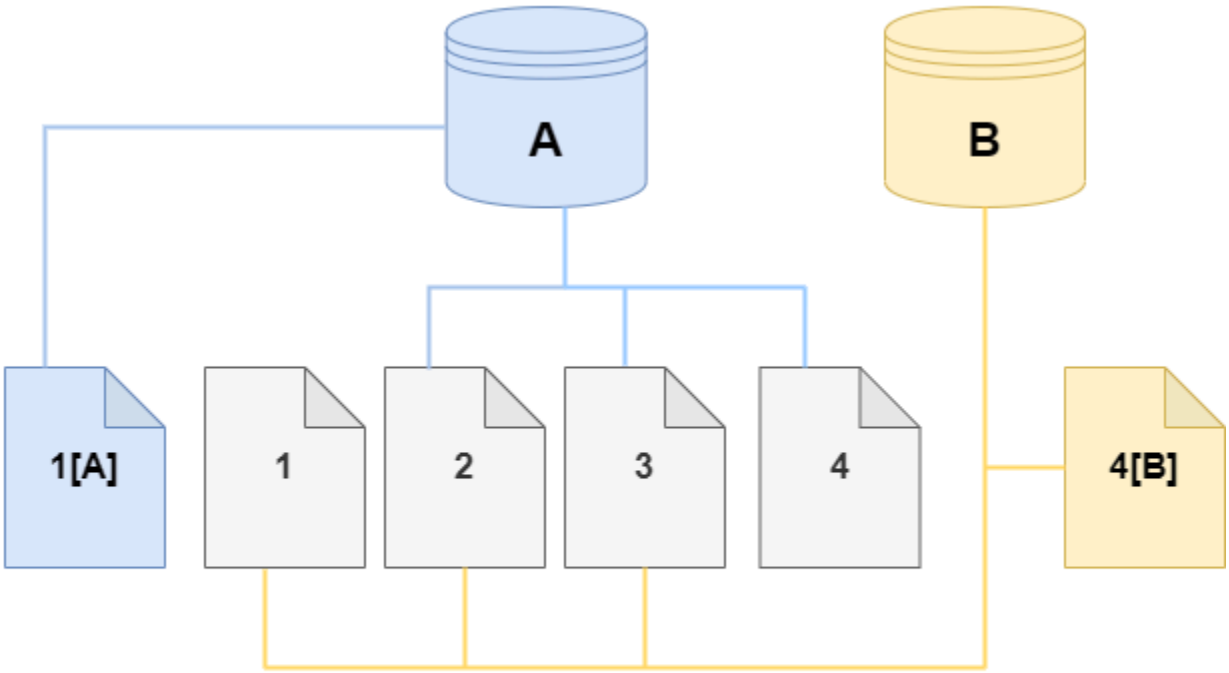
복제본이 생성되면 일반적으로 추가 스토리지가 필요하지 않습니다. 기록 중 복사(copy-on-write) 프로토콜은 물리적 스토리지 미디어에서 소스 세그먼트와 동일한 세그먼트를 사용합니다. 소스 세그먼트

의 용량이 전체 복제본 세그먼트에 충분하지 않은 경우에만 추가 스토리지가 필요합니다. 이 경우 소스 세그먼트는 다른 물리적 디바이스로 복사됩니다.

다음 다이어그램에서는 앞에서와 같이 동일한 클러스터 A와 해당 복제본 B를 사용하여 작동하는 기록 중 복사(copy-on-write) 프로토콜의 예를 찾을 수 있습니다. Amazon DocumentDB 클러스터(A)를 변경하여 페이지 1에 보관된 데이터가 변경된다고 가정해 보겠습니다. Amazon DocumentDB는 원본 페이지 1에 기록하는 대신 새 페이지 1[A]을 생성합니다. 클러스터(A)에 대한 Amazon DocumentDB 클러스터 볼륨은 이제 페이지 1[A], 2, 3, 4를 가리키고 복제본(B)은 여전히 원본 페이지를 참조합니다.



복제본에서 스토리지 볼륨의 페이지 4가 변경됩니다. Amazon DocumentDB는 원본 페이지 4에 기록하는 대신 새 페이지 4[B]를 생성합니다. 이제 복제본은 페이지 1, 2, 3 및 페이지 4[B]를 가리키고 클러스터(A)는 계속해서 1[A], 2, 3 및 4를 가리킵니다.



시간이 경과하여 소스 Amazon DocumentDB 클러스터 볼륨과 복제본 모두가 변경될 경우 변경 사항을 캡처하고 저장하기 위해 점점 더 많은 스토리지가 필요합니다.

원본 클러스터 볼륨 삭제

하나 이상의 복제본이 연결된 소스 클러스터 볼륨을 삭제해도 복제본은 영향을 받지 않습니다. 복제본은 이전에 원본 클러스터 볼륨이 소유하던 페이지를 계속 가리킵니다.

Amazon DocumentDB 복제본 생성

소스 Amazon DocumentDB 클러스터와 동일한 AWS 계정에 복제본을 생성할 수 있습니다. 이렇게 하려면 AWS Management Console 또는 AWS CLI 및 다음 절차를 사용할 수 있습니다.

Amazon DocumentDB 복제를 사용하면 프로비저닝된 Amazon DocumentDB 클러스터에서 프로비저닝된 Amazon DocumentDB 클러스터 복제본을 생성할 수 있습니다.

Using the AWS Management Console

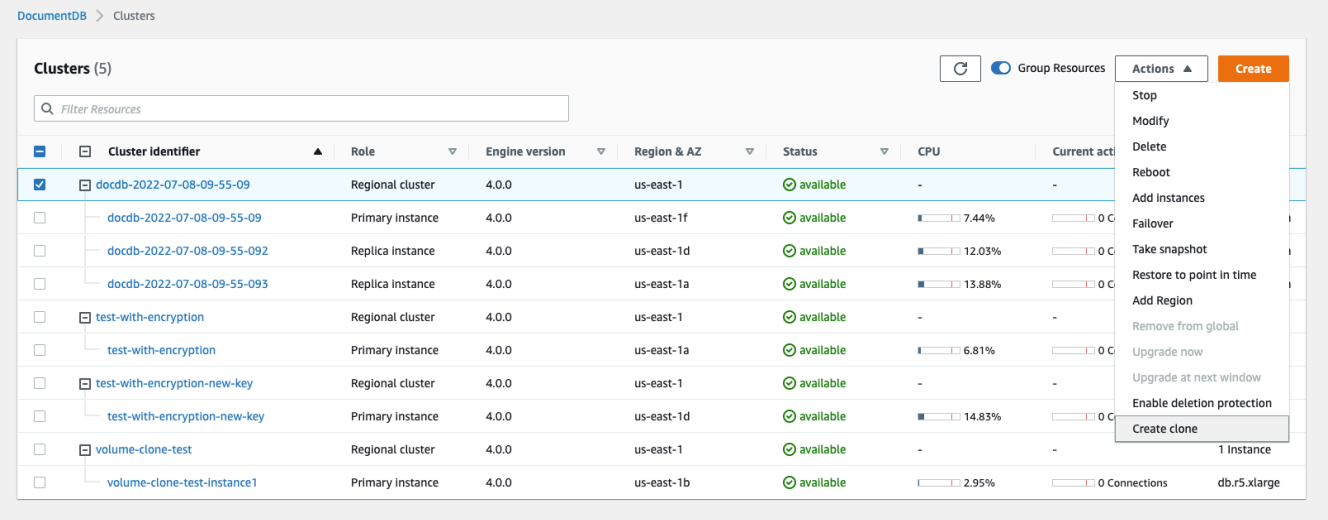
다음 절차에서는 AWS Management Console을 사용하여 Amazon DocumentDB 클러스터를 복제하는 방법에 대해 설명합니다.

Amazon DocumentDB 인스턴스 하나가 있는 Amazon DocumentDB 클러스터에서 AWS Management Console 결과를 사용하여 복제본을 생성합니다.

이 지침은 복제본을 생성하는 계정과 동일한 AWS 계정이 소유한 DB 클러스터에 적용됩니다. DB 클러스터는 Amazon DocumentDB에서 교차 AWS 계정 복제가 지원되지 않으므로 동일한 계정이 소유해야 합니다.

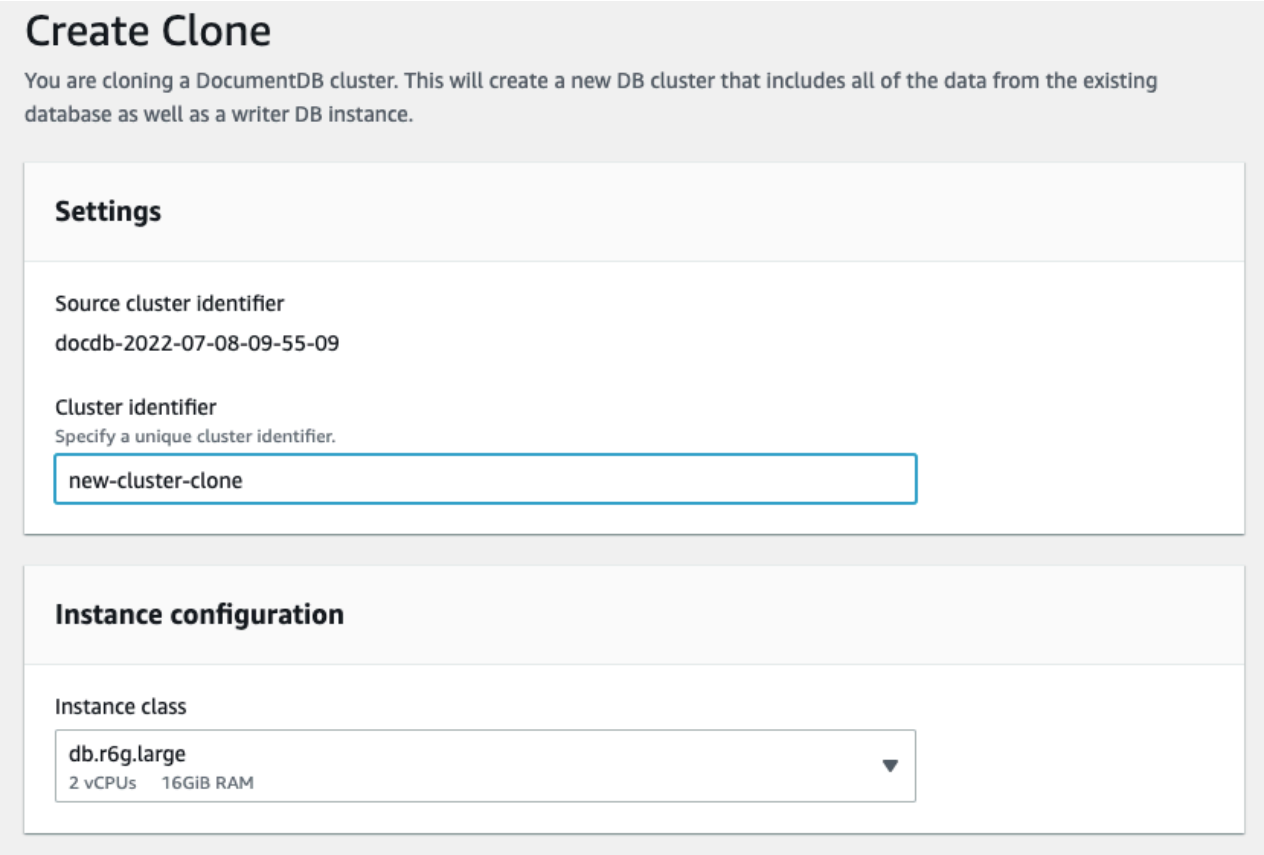
를 사용하여 AWS 계정이 소유한 DB 클러스터의 복제본을 생성하려면 AWS Management Console

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 목록에서 Amazon DocumentDB 클러스터를 선택하고 작업에서 복제본 생성을 선택합니다.



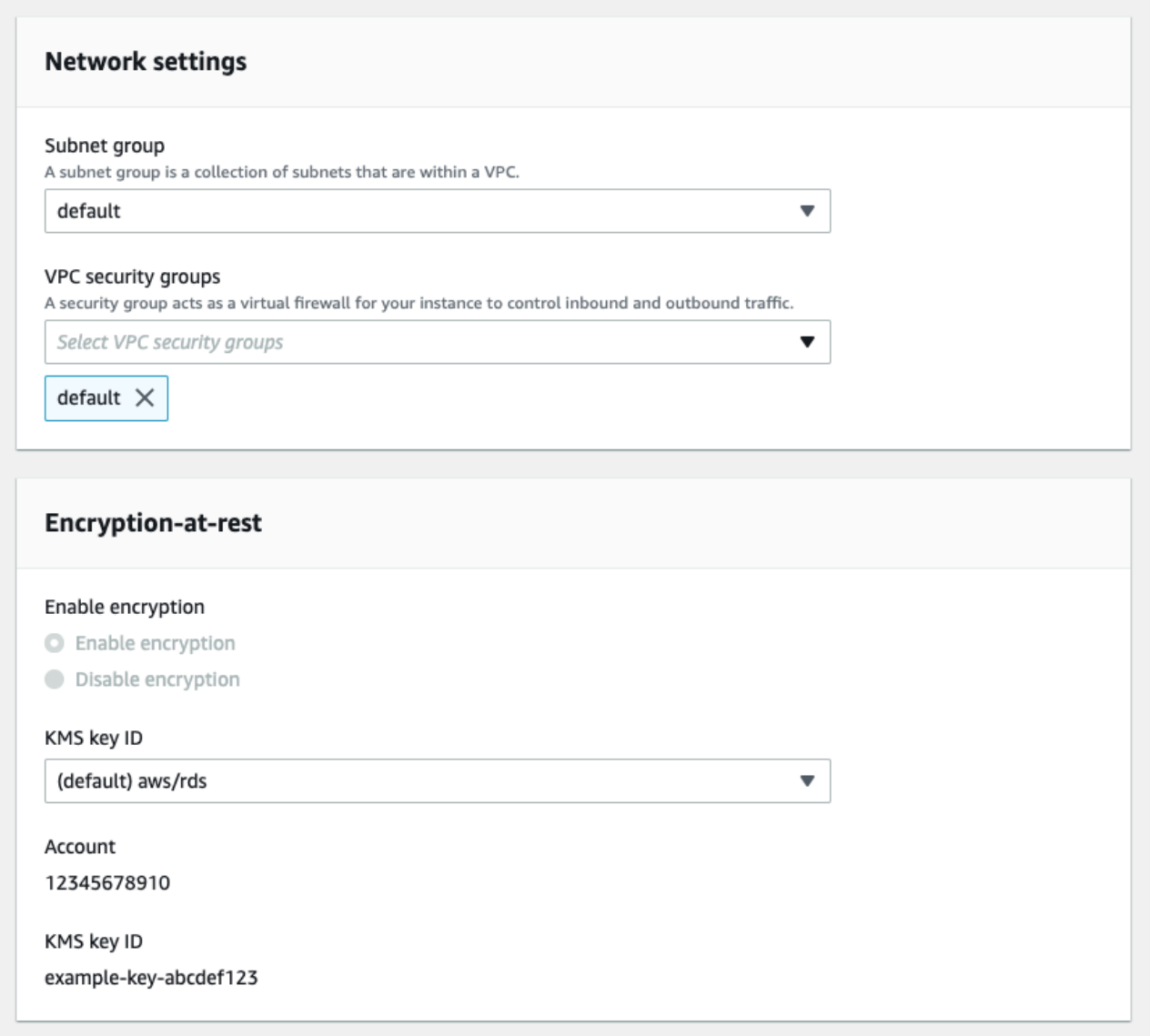
클러스터 식별자, 인스턴스 클래스, 그리고 Amazon DocumentDB 클러스터 복제본에 대한 기타 옵션을 구성할 수 있는 복제본 생성 페이지가 열립니다.

4. 설정 섹션에서 다음을 수행합니다.
   a. 클러스터 식별자에 복제된 Amazon DocumentDB 클러스터에 부여할 이름을 입력합니다.
   b. 인스턴스 구성의 경우 복제된 Amazon DocumentDB 클러스터에 적합한 인스턴스 클래스를 선택합니다.

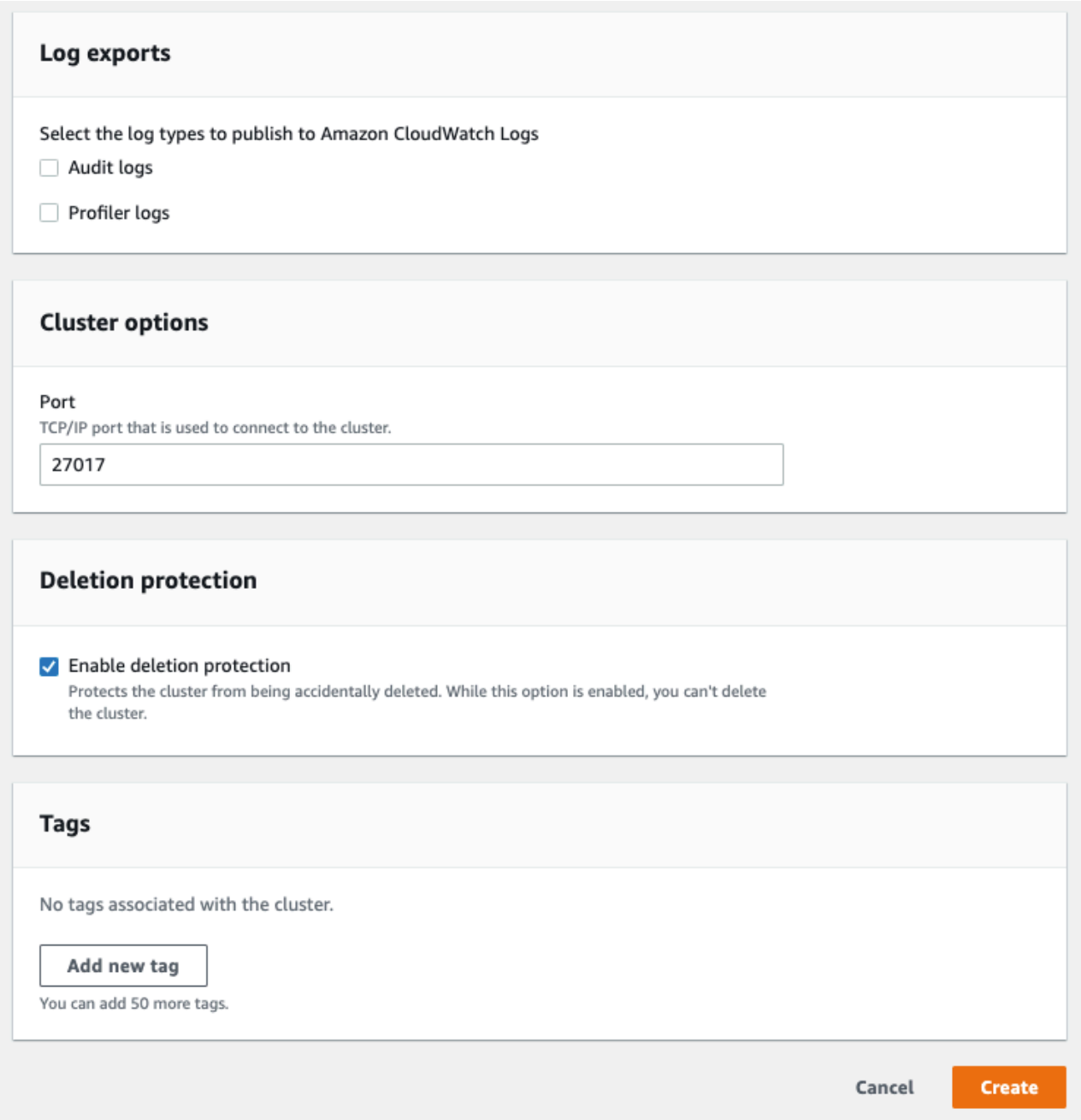


c. 네트워크 설정에서 사용 사례에 맞는 서브넷 그룹과 관련 VPC 보안 그룹을 선택합니다.

d. 저장 시 암호화의 경우 소스 클러스터(복제 중인 클러스터) 에 암호화가 활성화되어 있는 경우 복제된 클러스터에도 암호화가 활성화되어 있어야 합니다. 이 시나리오가 사실인 경우 암호화 사용 옵션은 회색으로 표시(사용 안 함)되지만 암호화 사용 옵션은 선택됩니다. 반대로 소스 클러스터에 암호화가 활성화되어 있지 않은 경우 암호화 활성화 옵션을 사용할 수 있으며 암호화를 활성화하거나 비활성화하도록 선택할 수 있습니다.



e. 내보낼 로그 유형(선택 사항)을 선택하고, 클러스터에 연결하는 데 사용되는 특정 포트를 입력하고, 클러스터를 실수로 삭제하지 않도록 보호(기본적으로 활성화됨)하여 새 클러스터 복제본 구성을 완료합니다.



f. Amazon DocumentDB 클러스터 복제본에 대한 모든 설정을 입력합니다. Amazon DocumentDB 클러스터 및 인스턴스 설정에 대한 자세한 내용은 Amazon DocumentDB 클러스터 관리 섹션에서 참조하세요.

5. 복제본 생성을 선택하여 선택한 Amazon DocumentDB 클러스터의 Amazon DocumentDB 복제본을 시작합니다.

복제본이 생성되면 복제본은 콘솔의 데이터베이스 섹션에 다른 Amazon DocumentDB 클러스터와 함께 나열되고 현재 상태가 표시됩니다. 상태가 사용 가능이면 복제본을 사용할 준비가 된 것입니다.

Using the AWS CLI

AWS CLI 를 사용하여 Amazon DocumentDB 클러스터를 복제하려면 몇 가지 단계가 필요합니다.

사용하는 `restore-db-cluster-to-point-in-time` AWS CLI 명령은 Amazon DocumentDB 인스턴스가 0개인 빈 Amazon DocumentDB 클러스터를 생성합니다. 즉, 명령은 Amazon DocumentDB 클러스터만 복원하고, 해당 클러스터의 DB 인스턴스는 복원하지 않습니다. 복제본이 사용 가능한 후에 별도로 이 작업을 수행합니다. 프로세스의 두 단계는 다음과 같습니다.

1. restore-db-cluster-to-point-in-time CLI 명령을 사용하여 복제본을 생성합니다. 이 명령과 함께 사용하는 파라미터는 생성 중인 빈 Amazon DocumentDB 클러스터(복제본)의 용량 유형 및 기타 세부 정보를 제어합니다.
2. create-db-instance CLI 명령을 사용하여 복제본에 대한 Amazon DocumentDB 인스턴스를 생성하여 복원된 Amazon DocumentDB 클러스터에서 Amazon DocumentDB 인스턴스를 다시 생성합니다.

다음 명령은 AWS CLI 가 AWS 리전을 기본값으로 설정한다고 가정합니다. 이 방식을 사용하면 각 명령에서 `--region` 이름을 전달하지 않아도 됩니다. 자세한 내용은 AWS CLI구성 섹션을 참조하세요. 다음 각 CLI 명령에서 `--region`을 지정할 수도 있습니다.

복제본 생성

`restore-db-cluster-to-point-in-time` CLI 명령으로 전달하는 특정 파라미터는 다양합니다. 전달하는 항목은 생성하려는 복제본 유형에 따라 다릅니다.

다음 절차를 사용하여 프로비저닝된 Amazon DocumentDB 클러스터에서 프로비저닝된 Amazon DocumentDB 복제본을 생성할 수 있습니다.

소스 Amazon DocumentDB 클러스터와 동일한 엔진 모드의 복제본을 생성하려면

- `restore-db-cluster-to-point-in-time` CLI 명령을 사용하여 다음 파라미터에 대한 값을 지정합니다.
  - `--db-cluster-identifier` - 복제본에 대해 의미 있는 이름을 선택합니다. 복제본의 이름을 지정할 때 restore-db-cluster-to-point-in-time CLI 명령을 사용합니다.

- `--restore-type` – `copy-on-write`을(를) 사용하여 소스 DB 클러스터의 복제본을 생성합니다. 이 파라미터가 없으면 `restore-db-cluster-to-point-in-time`은 복제본을 생성하는 대신 Amazon DocumentDB 클러스터를 복원합니다. `restore-type` 기본값은 `full-copy`입니다.
- `--source-db-cluster-identifier` - 복제할 소스 Amazon DocumentDB 클러스터의 이름을 사용합니다.
- `--use-latest-restorable-time` - 이 값은 복제본에 대해 복원 가능한 최신 볼륨 데이터를 가리킵니다. 이 매개변수는 필수 `restore-type copy-on-write` 매개변수이지만 `restore-to-time parameter`를 함께 사용할 수는 없습니다.

다음 예제에서는 `my-source-cluster`라는 이름의 클러스터에서 `my-clone`라는 이름의 복제본을 생성합니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

Windows의 경우:

```
aws docdb restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

이 명령은 복제본의 세부 사항을 포함하는 JSON 객체를 반환합니다. 복제본에 대한 DB 인스턴스를 생성하기 전에 복제된 DB 클러스터를 사용할 수 있는지 확인합니다. 자세한 내용은 아래의 상태 확인 및 복제본 세부 정보 가져오기를 참조하세요.

상태 확인 및 복제본 세부 정보 가져오기

다음 명령을 사용하여 새로 생성된 빈 DB 클러스터의 상태를 확인할 수 있습니다.

```
$ aws docdb describe-db-clusters --db-cluster-identifier my-clone --query '*[].
[Status]' --output text
```

또는 다음 AWS CLI 쿼리를 사용하여 복제본의 DB 인스턴스를 생성하는 데 필요한 상태 및 기타 값을 얻을 수 있습니다.

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-clusters --db-cluster-identifier my-clone \
  --query '*[].{Status:Status,Engine:Engine,EngineVersion:EngineVersion}'
```

Windows의 경우:

```
aws docdb describe-db-clusters --db-cluster-identifier my-clone ^
  --query "*[].{Status:Status,Engine:Engine,EngineVersion:EngineVersion}"
```

이 쿼리는 다음과 비슷한 출력을 반환합니다.

```
[
  {
        "Status": "available",
        "Engine": "docdb",
        "EngineVersion": "4.0.0",
    }
]
```

복제본을 위한 Amazon DocumentDB 인스턴스 생성

create-db-instance CLI 명령을 사용하여 복제본에 대한 DB 인스턴스를 생성하세요.

`--db-instance-class` 파라미터는 프로비저닝된 Amazon DocumentDB 클러스터에만 사용됩니다.

Linux, macOS, Unix의 경우:

```
aws docdb create-db-instance \
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
    --db-instance-class  db.r5.4xlarge \
    --engine docdb
```

Windows의 경우:

```
aws docdb create-db-instance ^
```

```
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class  db.r5.4xlarge ^
    --engine docdb
```

복제본에 사용할 파라미터

다음 표에는 `restore-db-cluster-to-point-in-time`에서 Amazon DocumentDB 클러스터를 복제하는 데 사용되는 다양한 파라미터가 요약되어 있습니다.

파라미터	설명
---source-db-cluster-identifier	복제할 소스 Amazon DocumentDB 클러스터의 이름을 사용합니다.
--db-cluster-identifier	복제본에 대해 의미 있는 이름을 선택합니다. `restore-db-cluster-to-point-in-time` 명령으로 복제본의 이름을 지정합니다. 그런 다음 이 이름을 `create-db-instance` 명령으로 전달합니다.
--restore-type	`copy-on-write` 를 `--restore-type` 로 지정하여 소스 Amazon DocumentDB 클러스터를 복원하는 대신 소스 DB 클러스터의 복제본을 생성합니다.
--use-latest-restorable-time	이 값은 복제본에 대해 복원 가능한 최신 볼륨 데이터를 가리킵니다.

Amazon DocumentDB 클러스터 내결함성에 대한 이해

Amazon DocumentDB 클러스터는 내결함성을 고려하여 설계되었습니다. 각 클러스터의 볼륨은 단일의 여러 가용 영역에 걸쳐 있으며 AWS 리전각 가용 영역에는 클러스터의 볼륨 데이터 사본이 포함됩니다. 이 기능은 가용 영역 한 곳에서 결함이 발생하더라도 클러스터가 잠시 서비스가 중단될 뿐 전혀 데이터 손실 없이 결함을 견딜 수 있음을 의미합니다.

클러스터의 기본 인스턴스에 결함이 발생하면 Amazon DocumentDB가 다음 두 가지 방법 중 하나를 사용하여 자동으로 새 기본 인스턴스로 장애 조치를 수행합니다.

- 기존 Amazon DocumentDB 복제본을 각 복제본의 프로모션 티어 설정에 따라 선택된 새 기본 인스턴스로 승격한 다음 이전 기본 인스턴스의 대체 인스턴스를 생성합니다. 복제본 인스턴스로의

페일오버는 보통 30초도 걸리지 않습니다. 이 기간 동안에는 읽기 및 쓰기 작업이 잠시 중단될 수 있습니다. 클러스터의 가용성을 높이려면 둘 이상의 서로 다른 가용 영역에 하나 이상의 Amazon DocumentDB 복제본을 생성합니다.

- 새로운 기본 인스턴스를 생성합니다. 이는 클러스터에 복제 인스턴스가 없는 경우에만 발생하며 완료하는 데 몇 분 정도 걸릴 수 있습니다.

클러스터에 Amazon DocumentDB 복제본이 하나 이상인 경우에는 장애가 발생하더라도 Amazon DocumentDB 복제본이 기본 인스턴스로 승격됩니다. 이 실패 이벤트로 인해 예외적으로 실패하는 읽기 및 쓰기 작업 동안 짧은 중단이 발생합니다. 하지만, 일반적인 서비스 복구 시간은 120초 미만이지만 대부분 60초 미만에 복원됩니다. 클러스터의 가용성을 높이려면 둘 이상의 서로 다른 가용 영역에 하나 이상의 Amazon DocumentDB 복제본을 생성합니다.

각 복제본에 우선 순위를 지정하여 장애 이후 기본 인스턴스로 승격할 Amazon DocumentDB 복제본 순서를 사용자 지정할 수 있습니다. 우선 순위 범위는 가장 높은 값인 0부터 가장 낮은 값인 15까지입니다. 기본 인스턴스에 결함이 발생하면 우선 순위가 가장 높은 Amazon DocumentDB 복제본을 새로운 기본 인스턴스로 승격시킵니다. Amazon DocumentDB 복제본의 우선 순위는 언제든지 수정할 수 있습니다. 우선 순위 수정으로 인해 장애 조치가 트리거되지는 않습니다. `modify-db-instance` 연산을 `--promotion-tier` 파라미터와 함께 사용할 수 있습니다. 인스턴스의 장애 조치 우선 순위를 사용자 지정하는 방법에 대한 자세한 내용은 Amazon DocumentDB 장애 조치 섹션을 참조하세요.

둘 이상의 Amazon DocumentDB 복제본이 동일한 우선 순위를 공유하여 승격 계층을 만들 수도 있습니다. 둘 이상의 Amazon DocumentDB 복제본이 동일한 우선 순위를 공유하면 크기가 가장 큰 복제본이 기본 복제본으로 승격됩니다. 둘 이상의 Amazon DocumentDB 복제본이 동일한 우선 순위와 크기를 공유하면 동일한 승격 티어에서 임의의 복제본이 승격됩니다.

클러스터에 Amazon DocumentDB 복제본이 포함되어 있지 않으면 기본 인스턴스가 실패 이벤트 중에 다시 생성됩니다. 이 실패 이벤트로 인해 예외적으로 실패하는 읽기 및 쓰기 작업 동안 중단이 발생합니다. 새로운 기본 인스턴스가 생성되면 서비스가 복구되며, 보통 10분 미만의 시간이 걸립니다. Amazon DocumentDB 복제본을 기본 인스턴스로 승격시키는 것이 기본 인스턴스를 새로 생성하는 것보다 훨씬 빠릅니다.

Amazon DocumentDB 인스턴스 관리

다음 항목에서는 Amazon DocumentDB 인스턴스를 관리하는 데 도움이 되는 정보를 제공합니다. 인스턴스 클래스 및 상태, 인스턴스를 생성, 삭제, 수정하는 방법에 대한 세부 정보가 포함됩니다.

주제

- 인스턴스 상태 확인
- Amazon DocumentDB 인스턴스 수명 주기
- 인스턴스 클래스 관리
- NVMe 지원 인스턴스

인스턴스 상태 확인

유효한 인스턴스 상태, 해당 의미, 인스턴스 상태 확인 방법을 알아보려면 Amazon DocumentDB 인스턴스 상태 모니터링 섹션을 참조하세요.

Amazon DocumentDB 인스턴스 수명 주기

Amazon DocumentDB 인스턴스의 수명 주기에는 인스턴스 생성, 수정, 유지 및 업그레이드, 백업 및 복원 수행, 재부팅 및 삭제가 포함됩니다. 이 섹션에서는 이러한 프로세스를 완료하는 방법에 관해 설명합니다.

주제

- 클러스터에 Amazon DocumentDB 인스턴스 추가
- Amazon DocumentDB 인스턴스 설명
- Amazon DocumentDB 인스턴스 수정
- Amazon DocumentDB 인스턴스 재부팅
- Amazon DocumentDB 인스턴스 삭제

AWS Management Console 또는를 사용하여 새 Amazon DocumentDB 인스턴스를 생성할 수 있습니다 AWS CLI. 인스턴스를 클러스터에 추가하려면 클러스터가 사용 가능 상태에 있어야 합니다. 중지된 클러스터에는 인스턴스를 추가할 수 없습니다. 클러스터가 중지된 경우, 먼저 클러스터를 시작하고 클러스터가 사용 가능하게 될 때까지 기다린 다음 인스턴스를 추가합니다. 자세한 내용은 Amazon DocumentDB 클러스터 중지 및 시작 섹션을 참조하십시오.

> **Note**
>
> 콘솔을 사용하여 Amazon DocumentDB 클러스터를 작성하면 인스턴스가 동시에 자동으로 생성됩니다. 추가 인스턴스를 생성하려면 다음 절차 중 하나를 사용하세요.

클러스터에 Amazon DocumentDB 인스턴스 추가

Using the AWS Management Console

다음 절차에 따라 콘솔을 사용하여 클러스터에 대한 Amazon DocumentDB 인스턴스를 생성할 수 있습니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택합니다.

3. 인스턴스를 추가할 클러스터를 선택하려면 클러스터의 이름 왼쪽에 있는 버튼을 선택합니다.
4. 작업을 선택한 다음 인스턴스 추가를 선택합니다.
5. 다음 항목에 인스턴스 추가: <cluster-name> 페이지에서 클러스터에 추가하려는 각 인스턴스에 대해 다음 단계를 반복 실행합니다. 최대 15개까지 가능합니다.

   a. 인스턴스 식별자 - 이 인스턴스의 고유 식별자를 입력하거나 Amazon DocumentDB가 클러스터 식별자를 기반으로 인스턴스 식별자를 제공하도록 허용할 수 있습니다.

      인스턴스 명명 제약 조건:

      - 길이는 [1-63] 문자, 숫자 또는 하이픈입니다.
      - 첫 번째 문자는 글자이어야 합니다.
      - 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.
      - 리전별로 Amazon RDS, Neptune 및 Amazon DocumentDB의 모든 인스턴스 AWS 계정에 대해 고유해야 합니다.

   b. 인스턴스 클래스 - 드롭다운 목록에서 이 인스턴스에 대해 원하는 인스턴스 유형을 선택합니다.

   c. 승격 계층 - 드롭다운 목록에서 인스턴스의 승격 계층을 선택하거나 선호 없음을 선택하여 Amazon DocumentDB가 인스턴스의 승격 계층을 설정할 수 있도록 합니다. 번호가 낮을수록 우선 순위가 높습니다. 자세한 내용은 장애 조치 대상 제어 섹션을 참조하십시오.

d. 더 많은 인스턴스를 추가하려면 추가 인스턴스 추가를 선택하고 a, b, c 단계를 반복합니다.

6. 작업을 완료합니다.

   - 클러스터에 인스턴스를 추가하려면 생성을 선택합니다.
   - 작업을 취소하려면 취소를 선택합니다.

인스턴스를 생성하는 데 몇 분 정도 걸립니다. 콘솔 또는 AWS CLI 를 사용하여 인스턴스의 상태를 볼 수 있습니다. 자세한 내용은 인스턴스 상태 모니터링 단원을 참조하십시오.

Using the AWS CLI

다음 파라미터와 함께 `create-db-instance` AWS CLI 작업을 사용하여 클러스터의 기본 인스턴스를 생성합니다.

- **`--db-instance-class`** - 필수입니다. 인스턴스의 컴퓨팅 및 메모리 용량(예: `db.m4.large`)입니다. 모든 AWS 리전에서 모든 인스턴스 클래스를 사용할 수 있는 것은 아닙니다.
- **`--db-instance-identifier`** - 필수입니다. 인스턴스를 식별하는 문자열입니다.

  인스턴스 명명 제약:

  - 길이는 [1-63] 문자, 숫자 또는 하이픈입니다.
  - 첫 번째 문자는 글자이어야 합니다.
  - 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.
  - 리전별로 Amazon RDS, Neptune 및 Amazon DocumentDB의 모든 인스턴스 AWS 계정에 대해 고유해야 합니다.
- **`--engine`** — 필수입니다. `docdb`여야 합니다.
- **`--availability-zone`** - 선택 사항. 이 인스턴스를 생성할 가용 영역입니다. 다른 가용 영역에서 인스턴스를 찾아서 내결함성을 높이려면 이 파라미터를 사용합니다. 자세한 내용은 Amazon DocumentDB 고가용성 및 복제 섹션을 참조하십시오.
- **`--promotion-tier`** - 선택 사항. 이 인스턴스의 장애 조치 우선 순위 계층입니다. 0~15 사이여야 하며, 숫자가 작을수록 우선 순위가 높습니다. 자세한 내용은 장애 조치 대상 제어 섹션을 참조하십시오.

1. 먼저 인스턴스를 생성할 수 있는 가용 영역을 결정합니다.

인스턴스를 생성하기 전에 사용 가능 영역을 지정하려면 다음 명령을 실행하여 Amazon DocumentDB 클러스터에 사용할 수 있는 가능 영역을 결정합니다.

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-clusters \
       --query 'DBClusters[*].[DBClusterIdentifier,AvailabilityZones[*]]'
```

Windows의 경우:

```
aws docdb describe-db-clusters ^
       --query 'DBClusters[*].[DBClusterIdentifier,AvailabilityZones[*]]'
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
[
    [
        "sample-cluster",
        [
            "us-east-1c",
            "us-east-1b",
            "us-east-1a"
        ]
    ]
]
```

2. 둘째, 리전에서 생성할 수 있는 인스턴스 클래스를 확인합니다.

   리전에서 사용 가능한 인스턴스 클래스를 확인하려면 다음 명령을 실행합니다. 출력에서 Amazon DocumentDB 클러스터에 추가할 인스턴스의 인스턴스 클래스를 선택합니다.

   Linux, macOS, Unix의 경우:

   ```
   aws docdb describe-orderable-db-instance-options \
           --engine docdb \
           --query 'OrderableDBInstanceOptions[*].DBInstanceClass'
   ```

   Windows의 경우:

```
aws docdb describe-orderable-db-instance-options ^
        --engine docdb ^
        --query 'OrderableDBInstanceOptions[*].DBInstanceClass'
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
[
    "db.r5.16xlarge",
    "db.r5.2xlarge",
    "db.r5.4xlarge",
    "db.r5.8xlarge",
    "db.r5.large",
    "db.r5.xlarge"
]
```

3. 마지막으로, Amazon DocumentDB 클러스터에 인스턴스를 추가합니다.

   인스턴스를 Amazon DocumentDB 클러스터에 추가하려면 다음 명령을 실행합니다..

   Linux, macOS, Unix의 경우:

```
aws docdb create-db-instance \
       --db-cluster-identifier sample-cluster \
       --db-instance-identifier sample-instance-2 \
       --availability-zone us-east-1b \
       --promotion-tier 2 \
       --db-instance-class db.r5.xlarge \
       --engine docdb
```

   Windows의 경우:

```
aws docdb create-db-instance ^
       --db-cluster-identifier sample-cluster ^
       --db-instance-identifier sample-instance-2 ^
       --availability-zone us-east-1b ^
       --promotion-tier 2 ^
       --db-instance-class db.r5.xlarge ^
       --engine docdb
```

   이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "sample-instance-2",
        "DBInstanceClass": "db.r5.xlarge",
        "Engine": "docdb",
        "DBInstanceStatus": "creating",
        "PreferredBackupWindow": "02:00-02:30",
        "BackupRetentionPeriod": 1,
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcd0123",
                "Status": "active"
            }
        ],
        "AvailabilityZone": "us-east-1b",
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-6242c31a",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-abcd0123",
                    "SubnetAvailabilityZone": {
                        "Name": "us-west-2a"
                    },
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-wxyz0123",
                    "SubnetAvailabilityZone": {
                        "Name": "us-west-2b"
                    },
                    "SubnetStatus": "Active"
                }
            ]
        },
        "PreferredMaintenanceWindow": "sun:11:35-sun:12:05",
        "PendingModifiedValues": {},
        "EngineVersion": "3.6.0",
        "AutoMinorVersionUpgrade": true,
        "PubliclyAccessible": false,
        "DBClusterIdentifier": "sample-cluster",
```

```
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "CACertificateIdentifier": "rds-ca-2019",
        "PromotionTier": 2,
        "DBInstanceArn": "arn:aws:rds:us-east-1:<accountID>:db:sample-instance-2"
    }
}
```

인스턴스를 생성하는 데 몇 분 정도 걸립니다. 콘솔 또는 AWS CLI 를 사용하여 인스턴스의 상태를 볼 수 있습니다. 자세한 내용은 Amazon DocumentDB 인스턴스 상태 모니터링 단원을 참조하십시오.

Amazon DocumentDB 인스턴스 설명

Amazon DocumentDB Management Console 또는 AWS CLI 을 사용하여 연결 엔드포인트, 보안 그룹 VPC, 인증 기관 및 Amazon DocumentDB 인스턴스와 관련된 매개 변수 그룹과 같은 세부 정보를 볼 수 있습니다.

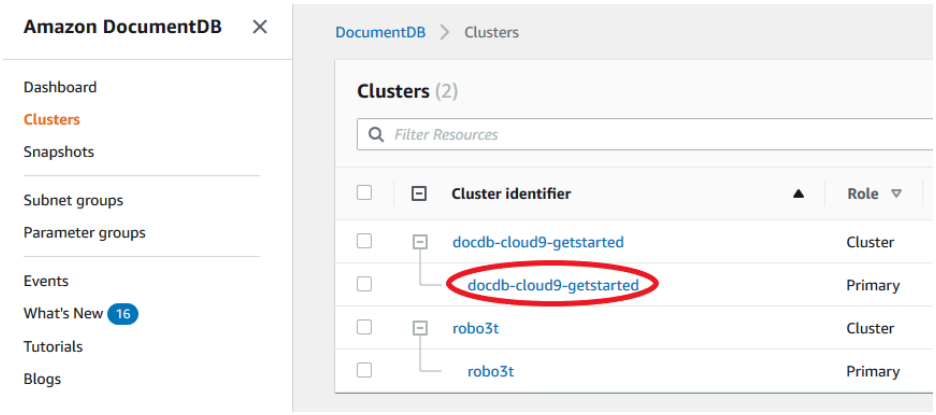
Using the AWS Management Console

AWS Management Console를 사용하여 인스턴스의 세부 정보를 보려면 아래 단계를 따르세요.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택합니다.

3. 클러스터 탐색 상자에 클러스터 식별자 열이 표시됩니다. 인스턴스는 다음 스크린샷과 마찬가지로 클러스터 아래에 나열됩니다.



4. 인스턴스 목록에서 세부 정보를 볼 인스턴스의 이름을 선택합니다. 인스턴스에 대한 정보는 다음과 같은 그룹으로 구성됩니다.

   - 요약 - 엔진 버전, 클래스, 상태 및 보류 중인 유지 보수를 포함한 인스턴스에 대한 일반 정보입니다.
   - 연결 및 보안 - 연결 섹션에는 mongo 쉘 또는 애플리케이션을 사용하여 이 인스턴스에 연결할 연결 엔드포인트가 나열됩니다. 보안 그룹 섹션에는 이 인스턴스와 연결된 보안 그룹과 해당 VPC ID 및 설명이 나열됩니다.
   - 구성 - 세부 정보 섹션에는 인스턴스의 Amazon 리소스 이름(ARN), 엔드포인트, 역할, 클래스 및 인증 기관을 포함하여 인스턴스의 구성 및 상태가 나열됩니다. 또한 인스턴스의 보안 및 네트워크 설정과 백업 정보가 나열됩니다. 클러스터 세부 정보 섹션에는 이 인스턴스가 속한 클러스터의 세부 정보가 나열됩니다. 클러스터 인스턴스 섹션에는 클러스터에 속한 모든 인스턴스가 각 인스턴스의 역할 및 클러스터 파라미터 그룹 상태와 함께 나열됩니다.

     > **Note**
     >
     > 클러스터 디테일 헤더 옆에 있는 수정을 선택하여 인스턴스와 연결된 클러스터를 수정할 수 있습니다. 자세한 내용은 Amazon DocumentDB 클러스터 수정 섹션을 참조하십시오.

   - 모니터링 - 이 인스턴스에 대한 CloudWatch Logs 지표. 자세한 내용은 CloudWatch를 사용하여 Amazon DocumentDB 모니터링 섹션을 참조하십시오.
   - 이벤트 및 태그 — 최근 이벤트 섹션에는 이 인스턴스의 최근 이벤트가 나열됩니다. Amazon DocumentDB는 클러스터, 인스턴스, 스냅샷, 보안 그룹 및 클러스터 매개 변수 그룹과 관련된 이벤트를 기록합니다. 이 정보에는 각 이벤트와 연결된 날짜, 시간 및 메시지가 포함됩니다. 태그 섹션에는 이 클러스터에 연결된 태그가 나열됩니다. 자세한 내용은 Amazon DocumentDB 리소스 태그 지정 단원을 참조하십시오.

Using the AWS CLI

를 사용하여 Amazon DocumentDB 인스턴스의 세부 정보를 보려면 다음 예제와 같이 `describe-db-clusters` 명령을 AWS CLI사용합니다. 자세한 내용은 Amazon DocumentDB 리소스 관리 API 참조에서 DescribeDBInstances 섹션을 참조하세요.

> **Note**
>
> 클러스터 및 인스턴스 수명주기 관리와 같은 특정 관리 기능의 경우 Amazon DocumentDB는 Amazon RDS와 공유하는 운영 기술을 활용합니다. `filterName=engine,Values=docdb` 필터 파라미터는 Amazon DocumentDB 클러스터만 반환합니다.

1. 모든 Amazon DocumentDB 인스턴스를 나열합니다.

   다음 AWS CLI 코드는 리전의 모든 Amazon DocumentDB 인스턴스에 대한 세부 정보를 나열합니다.

   Linux, macOS, Unix의 경우:

   ```
   aws docdb describe-db-instances \
       --filter Name=engine,Values=docdb
   ```

   Windows의 경우:

   ```
   aws docdb describe-db-instances \
       --filter Name=engine,Values=docdb
   ```

2. 지정된 Amazon DocumentDB 인스턴스에 대한 모든 세부 정보 나열

   다음 코드는 `sample-cluster-instance`의 세부 정보를 나열합니다. 인스턴스 이름과 함께 `--db-instance-identifier` 파라미터를 포함하면 해당 특정 인스턴스에 대한 정보로 출력이 제한됩니다.

   Linux, macOS, Unix의 경우:

   ```
   aws docdb describe-db-instances \
       --db-instance-identifier sample-cluster-instance
   ```

Windows의 경우:

```
aws docdb describe-db-instances \
    --db-instance-identifier sample-cluster-instance
```

이 작업의 출력은 다음과 같습니다.

```
{
    "DBInstances": [
        {
            "DbiResourceId": "db-BJKKB54PIDV5QFKGVRX5T3S6GM",
            "DBInstanceArn": "arn:aws:rds:us-east-1:012345678901:db:sample-
cluster-instance-00",
            "VpcSecurityGroups": [
                {
                    "VpcSecurityGroupId": "sg-77186e0d",
                    "Status": "active"
                }
            ],
            "DBInstanceClass": "db.r5.large",
            "DBInstanceStatus": "creating",
            "AutoMinorVersionUpgrade": true,
            "PreferredMaintenanceWindow": "fri:09:32-fri:10:02",
            "BackupRetentionPeriod": 1,
            "StorageEncrypted": true,
            "DBClusterIdentifier": "sample-cluster",
            "EngineVersion": "3.6.0",
            "AvailabilityZone": "us-east-1a",
            "Engine": "docdb",
            "PromotionTier": 2,
            "DBInstanceIdentifier": "sample-cluster-instance",
            "PreferredBackupWindow": "00:00-00:30",
            "PubliclyAccessible": false,
            "DBSubnetGroup": {
                "DBSubnetGroupName": "default",
                "Subnets": [
                    {
                        "SubnetIdentifier": "subnet-4e26d263",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1a"
                        },
                        "SubnetStatus": "Active"
```

```
                    },
                    {
                        "SubnetIdentifier": "subnet-afc329f4",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1c"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-b3806e8f",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1e"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-53ab3636",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1d"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-991cb8d0",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1b"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-29ab1025",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1f"
                        },
                        "SubnetStatus": "Active"
                    }
                ],
                "VpcId": "vpc-91280df6",
                "DBSubnetGroupDescription": "default",
                "SubnetGroupStatus": "Complete"
            },
            "PendingModifiedValues": {},
            "KmsKeyId": "arn:aws:kms:us-east-1:012345678901:key/0961325d-
a50b-44d4-b6a0-a177d5ff730b"
```

```
        }
    ]
}
```

Amazon DocumentDB 인스턴스 수정

AWS Management Console 또는를 사용하여 Amazon DocumentDB 인스턴스를 수정할 수 있습니다 AWS CLI. 인스턴스를 수정하려면 인스턴스가 사용 가능 상태에 있어야 합니다. 중지된 인스턴스는 수정할 수 없습니다. 클러스터가 중지된 경우, 먼저 클러스터를 시작하고 인스턴스가 사용 가능하게 될 때까지 기다린 다음 원하는 대로 수정합니다. 자세한 내용은 Amazon DocumentDB 클러스터 중지 및 시작 섹션을 참조하십시오.

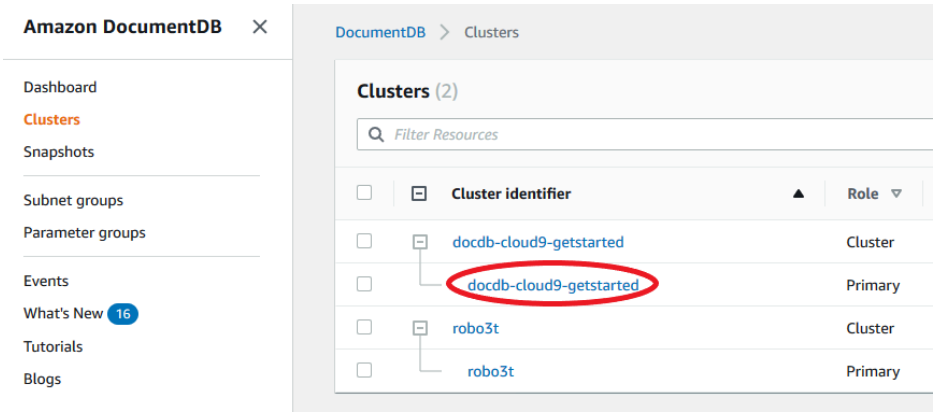
Using the AWS Management Console

콘솔을 사용하여 특정 Amazon DocumentDB 인스턴스를 수정하려면 다음 단계를 완료합니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택합니다.

3. 클러스터 탐색 상자에 클러스터 식별자 열이 표시됩니다. 인스턴스는 다음 스크린샷과 마찬가지로 클러스터 아래에 나열됩니다.



4. 수정할 인스턴스의 왼쪽에 있는 상자를 선택합니다.
5. 작업을 선택한 후 수정을 선택합니다.
6. 인스턴스 수정: <instance-name> 창에서 원하는 항목을 변경합니다. 다음과 같이 변경할 수 있습니다:

   - 인스턴스 규격 - 인스턴스 식별자 및 클래스. 인스턴스 식별자 명명 제약 조건:
     - 인스턴스 식별자 - 현재 리전 AWS 계정 에서가 소유한 모든 인스턴스에 대해 고유한 이름을 입력합니다. 인스턴스 식별자는 [1-63]개의 영숫자 또는 하이픈을 포함해야 하며 첫 번째 문자로 문자가 있어야 하며 하이픈으로 끝나거나 연속적으로 두 개의 하이픈을 포함할 수 없습니다.
     - 인스턴스 클래스 — 드롭다운 메뉴에서 Amazon DocumentDB 인스턴스의 인스턴스 클래스를 선택합니다. 자세한 내용은 인스턴스 클래스 관리 섹션을 참조하십시오.
   - 인증서 권한 - 인스턴스의 서버 인증서입니다. 자세한 내용은 Amazon DocumentDB TLS 인증서 업데이트 섹션을 참조하십시오.
   - 장애 조치 - 장애 조치 중에 가장 높은 승격 계층을 가진 인스턴스가 기본으로 승격됩니다. 자세한 내용은 Amazon DocumentDB 장애 조치 섹션을 참조하십시오.
   - 관리 - 보류 중인 수정 또는 패치가 클러스터의 인스턴스에 적용되는 유지 관리 창입니다.
7. 작업을 마쳤으면 계속을 선택하여 변경 사항의 요약을 확인합니다.
8. 변경 사항을 확인한 후 즉시 적용하거나 수정 일정 아래의 다음 유지 관리 기간 중에 적용할 수 있습니다. 인스턴스 수정를 선택하여 변경 사항을 저장합니다. 또는 취소를 선택하여 변경 사항을 취소할 수 있습니다.

변경 사항을 적용하는 데 몇 분 정도 걸립니다. 사용 가능 상태인 경우에만 인스턴스를 사용할 수 있습니다. 콘솔 또는 AWS CLI를 사용하여 인스턴스의 상태를 모니터링할 수 있습니다. 자세한 내용은 Amazon DocumentDB 인스턴스 상태 모니터링 단원을 참조하십시오.

Using the AWS CLI

를 사용하여 특정 Amazon DocumentDB 인스턴스를 수정하려면 다음 파라미터`modify-db-instance`와 함께를 AWS CLI사용합니다. 자세한 내용은 ModifyDBInstance을 참조하세요. 다음 코드는 `sample-instance` 인스턴스에 대한 인스턴스 클래스를 `db.r5.large`로 수정합니다.

파라미터

- **`--db-instance-identifier`** - 필수입니다. 수정할 인스턴스의 식별자입니다.

- **--db-instance-class** - 선택 사항. 인스턴스의 새 컴퓨팅 및 메모리 용량입니다 (예: db.r5.large). 모든 AWS 리전에서 모든 인스턴스 클래스를 사용할 수 있는 것은 아닙니다. 인스턴스 클래스를 수정하면 변경 중에 중단이 발생합니다. 이 요청에 대해 ApplyImmediately이 true로 지정되지 않은 경우 다음 유지 관리 기간 동안 변경 사항이 적용됩니다.
- **--apply-immediately** 또는 **--no-apply-immediately** - 선택 사항. 이 수정을 즉시 적용할지 다음 유지 관리 기간까지 대기할지 여부를 지정합니다. 이 파라미터를 생략하면 다음 유지 관리 기간 중에 수정이 수행됩니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb modify-db-instance \
       --db-instance-identifier sample-instance \
       --db-instance-class db.r5.large \
       --apply-immediately
```

Windows의 경우:

```
aws docdb modify-db-instance ^
       --db-instance-identifier sample-instance ^
       --db-instance-class db.r5.large ^
       --apply-immediately
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBInstances": [
        {
            "DBInstanceIdentifier": "sample-instance-1",
            "DBInstanceClass": "db.r5.large",
            "Engine": "docdb",
            "DBInstanceStatus": "modifying",
            "Endpoint": {
                "Address": "sample-instance-1.node.us-east-1.docdb.amazonaws.com",
                "Port": 27017,
                "HostedZoneId": "ABCDEFGHIJKLM"
            },
            "InstanceCreateTime": "2020-01-10T22:18:55.921Z",
```

```
            "PreferredBackupWindow": "02:00-02:30",
            "BackupRetentionPeriod": 1,
            "VpcSecurityGroups": [
                {
                    "VpcSecurityGroupId": "sg-abcd0123",
                    "Status": "active"
                }
            ],
            "AvailabilityZone": "us-east-1a",
            "DBSubnetGroup": {
                "DBSubnetGroupName": "default",
                "DBSubnetGroupDescription": "default",
                "VpcId": "vpc-abcd0123",
                "SubnetGroupStatus": "Complete",
                "Subnets": [
                    {
                        "SubnetIdentifier": "subnet-abcd0123",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1a"
                        },
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-abcd0123",
                        "SubnetAvailabilityZone": {
                            "Name": "us-east-1b"
                        },
                        "SubnetStatus": "Active"
                    }
                ]
            },
            "PreferredMaintenanceWindow": "sun:10:57-sun:11:27",
            "PendingModifiedValues": {
                "DBInstanceClass": "db.r5.large"
            },
            "EngineVersion": "3.6.0",
            "AutoMinorVersionUpgrade": true,
            "PubliclyAccessible": false,
            "DBClusterIdentifier": "sample-cluster",
            "StorageEncrypted": true,
            "KmsKeyId": "arn:aws:kms:us-east-1:123456789012:key/wJalrXUtnFEMI/
K7MDENG/bPxRfiCYEXAMPLEKEY",
            "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
            "CACertificateIdentifier": "rds-ca-2019",
```

```
            "PromotionTier": 1,
            "DBInstanceArn": "arn:aws:rds:us-east-1:123456789012:db:sample-
instance-1",
            "EnabledCloudwatchLogsExports": [
                "profiler"
            ]
        }
    ]
}
```

수정 사항을 적용하는 데 몇 분 정도 걸립니다. 사용 가능 상태인 경우에만 인스턴스를 사용할 수 있습니다. AWS Management Console 또는를 사용하여 인스턴스의 상태를 모니터링할 수 있습니다 AWS CLI. 자세한 내용은 Amazon DocumentDB 인스턴스 상태 모니터링 단원을 참조하십시오.

Amazon DocumentDB 인스턴스 재부팅

보통 유지보수의 이유로 Amazon DocumentDB 인스턴스를 재부팅해야 하는 경우가 있습니다. 특정 항목을 변경한 경우(예: 클러스터와 연결된 클러스터 파라미터 그룹 변경) 클러스터의 인스턴스를 재부팅해야 변경 내용이 적용됩니다. AWS Management Console 또는를 사용하여 지정된 인스턴스를 재부팅할 수 있습니다 AWS CLI.

인스턴스를 재부팅하면 데이터베이스 엔진 서비스가 재시작됩니다. 재부팅하면 DB 인스턴스 상태가 `rebooting`으로 설정되면서 잠시 중단됩니다. 재부팅이 완료되면 Amazon DocumentDB 이벤트가 생성됩니다.

인스턴스를 재부팅해도 장애 조치가 발생하지 않습니다. Amazon DocumentDB 클러스터를 장애 조치하려면 AWS Management Console 또는 AWS CLI 작업을 사용합니다`failover-db-cluster`. 자세한 내용은 Amazon DocumentDB 장애 조치 단원을 참조하십시오.

사용 가능 상태가 아닌 경우 인스턴스를 재부팅할 수 없습니다. 이전에 수정을 요청했거나 유지 관리 기간 작업 등 여러 가지 이유로 데이터베이스를 사용할 수 없는 경우가 있습니다. 인스턴스 상태에 대한 자세한 내용은 Amazon DocumentDB 인스턴스 상태 모니터링 섹션을 참조하세요.

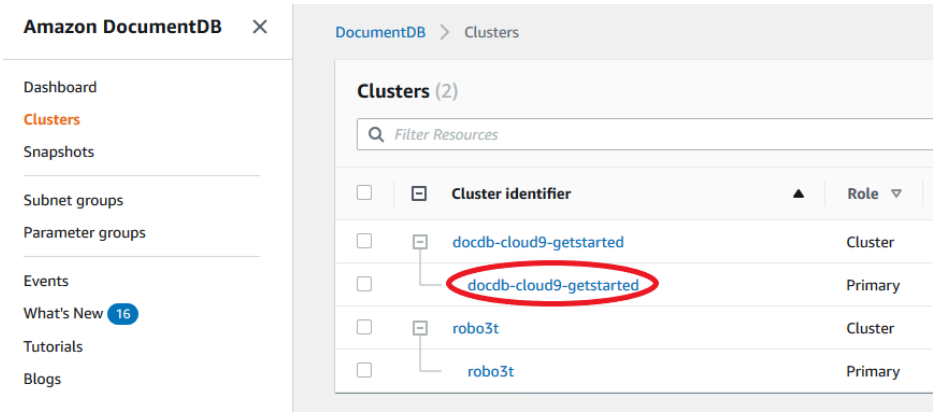
Using the AWS Management Console

다음 절차는 콘솔을 사용하여 지정한 인스턴스를 재부팅합니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

3. 클러스터 탐색 상자에 클러스터 식별자 열이 표시됩니다. 인스턴스는 다음 스크린샷과 마찬가지로 클러스터 아래에 나열됩니다.



4. 재부팅할 인스턴스의 왼쪽에 있는 확인란을 선택합니다.
5. 작업을 선택하고 재부팅을 선택한 다음 재부팅을 선택하여 재부팅을 확인합니다.

인스턴스가 재부팅되는 데 몇 분 정도 걸릴 수 있습니다. 사용 가능 상태인 경우에만 인스턴스를 사용할 수 있습니다. 콘솔 또는 AWS CLI를 사용하여 인스턴스의 상태를 모니터링할 수 있습니다. 자세한 내용은 Amazon DocumentDB 인스턴스 상태 모니터링 단원을 참조하십시오.

Using the AWS CLI

Amazon DocumentDB 인스턴스를 재부팅하려면 `--db-instance-identifier` 파라미터와 함께 `reboot-db-instance` 작업을 사용합니다. 이 파라미터는 재부팅할 인스턴스의 식별자를 지정합니다.

다음 코드는 `sample-instance` 인스턴스를 재부팅합니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb reboot-db-instance \
       --db-instance-identifier sample-instance
```

Windows의 경우:

```
aws docdb reboot-db-instance ^
       --db-instance-identifier sample-instance
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "sample-instance",
        "DBInstanceClass": "db.r5.large",
        "Engine": "docdb",
        "DBInstanceStatus": "rebooting",
        "Endpoint": {
            "Address": "sample-instance.node.us-east-1.docdb.amazonaws.com",
            "Port": 27017,
            "HostedZoneId": "ABCDEFGHIJKLM"
        },
        "InstanceCreateTime": "2020-03-27T08:05:56.314Z",
        "PreferredBackupWindow": "02:00-02:30",
        "BackupRetentionPeriod": 1,
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcd0123",
                "Status": "active"
            }
        ],
        "AvailabilityZone": "us-east-1c",
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-abcd0123",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-abcd0123",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1a"
                    },
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-wxyz0123",
```

```
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1b"
                    },
                    "SubnetStatus": "Active"
                }
            ]
        },
        "PreferredMaintenanceWindow": "sun:06:53-sun:07:23",
        "PendingModifiedValues": {},
        "EngineVersion": "3.6.0",
        "AutoMinorVersionUpgrade": true,
        "PubliclyAccessible": false,
        "DBClusterIdentifier": "sample-cluster",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "CACertificateIdentifier": "rds-ca-2019",
        "PromotionTier": 1,
        "DBInstanceArn": "arn:aws:rds:us-east-1:<accountID>:db:sample-instance",
        "EnabledCloudwatchLogsExports": [
            "profiler"
        ]
    }
}
```

인스턴스가 재부팅되는 데 몇 분 정도 걸릴 수 있습니다. 사용 가능 상태인 경우에만 인스턴스를 사용할 수 있습니다. 콘솔 또는 AWS CLI를 사용하여 인스턴스의 상태를 모니터링할 수 있습니다. 자세한 내용은 Amazon DocumentDB 인스턴스 상태 모니터링 단원을 참조하십시오.

Amazon DocumentDB 인스턴스 삭제

AWS Management Console 또는를 사용하여 Amazon DocumentDB 인스턴스를 삭제할 수 있습니다 AWS CLI. 인스턴스를 삭제하려면 인스턴스가 사용 가능 상태에 있어야 합니다. 중지된 인스턴스는 삭제할 수 없습니다. 인스턴스가 포함된 Amazon DocumentDB 클러스터가 중지된 경우 먼저 클러스터를 시작하고 인스턴스가 사용 가능해질 때까지 기다렸다가 인스턴스를 삭제합니다. 자세한 내용은 Amazon DocumentDB 클러스터 중지 및 시작 섹션을 참조하십시오.

> **Note**
>
> Amazon DocumentDB에서는 모든 데이터를 클러스터 볼륨에 저장합니다. 모든 인스턴스를 클러스터에서 제거하는 경우에도 데이터가 클러스터 볼륨에 유지됩니다. 데이터에 다시 액세스해야 하는 경우 언제든지 클러스터에 인스턴스를 추가하고 중단한 지점을 선택할 수 있습니다.

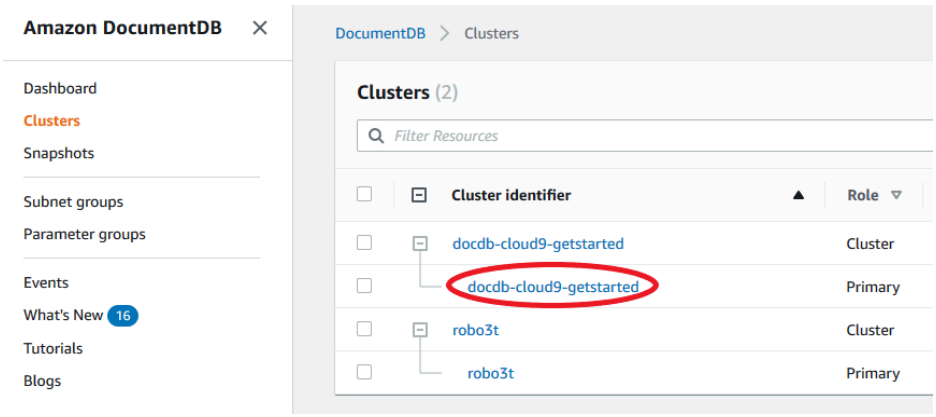
Using the AWS Management Console

다음 절차에서는 콘솔을 사용하여 지정된 Amazon DocumentDB 인스턴스를 삭제합니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

3. 클러스터 탐색 상자에 클러스터 식별자 열이 표시됩니다. 인스턴스는 다음 스크린샷과 마찬가지로 클러스터 아래에 나열됩니다.



4. 삭제할 인스턴스의 왼쪽에 있는 확인란을 선택합니다.
5. 작업을 선택한 후 삭제를 선택합니다.

   1. 클러스터의 마지막 인스턴스를 삭제하는 경우:

- 최종 클러스터 스냅샷을 생성하시겠습니까? — 클러스터를 삭제하기 전에 최종 스냅샷을 생성하려면 예를 선택합니다. 그렇지 않은 경우 아니요를 선택합니다.
- 최종 스냅샷 이름 - 최종 스냅샷을 생성하도록 선택한 경우 생성된 새 클러스터 스냅샷의 클러스터 스냅샷 식별자를 입력합니다.
- <instance-name> 인스턴스를 삭제하시겠습니까? — 삭제 확인을 위해 전체 클러스터 삭제라는 문구를 필드에 입력합니다.

2. 클러스터의 마지막 인스턴스를 삭제하지 않는 경우:

- <instance-name> 인스턴스를 삭제하시겠습니까? — 삭제 라는 문구를 필드에 입력하여 삭제를 확인합니다.

6. 삭제를 선택하여 인스턴스를 삭제합니다.

인스턴스를 삭제하는 데 몇 분 정도 걸립니다. 인스턴스 상태를 모니터링하려면 Amazon DocumentDB 인스턴스 상태 모니터링 섹션을 참조하십시오.

Using the AWS CLI

다음 절차는 AWS CLI을 사용하여 Amazon DocumentDB 인스턴스를 삭제합니다.

1. 먼저 Amazon DocumentDB 클러스터에 몇 개의 인스턴스가 있는지 확인합니다:

   클러스터에 있는 인스턴스 수를 확인하려면 다음과 같이 `describe-db-clusters` 명령을 실행합니다.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].
[DBClusterIdentifier,DBClusterMembers[*].DBInstanceIdentifier]'
```

   이 작업의 출력은 다음과 같이 표시됩니다.

```
[
    [
        "sample-cluster",
        [
            "sample-instance-1",
            "sample-instance-2"
        ]
    ]
```

```
]
```

2. Amazon DocumentDB 클러스터에 인스턴스가 두 개 이상 있는 경우:

   지정된 Amazon DocumentDB 인스턴스를 삭제하려면 아래와 같이 `delete-db-instance` 명령을 `--db-instance-identifier` 파라미터와 함께 사용합니다. 인스턴스를 삭제하는 데 몇 분 정도 걸립니다. 인스턴스 상태를 모니터링하려면 Amazon DocumentDB 인스턴스 상태 모니터링 섹션을 참조하십시오.

   ```
   aws docdb delete-db-instance \
         --db-instance-identifier sample-instance-2
   ```

   이 작업의 출력은 다음과 같이 표시됩니다.

   ```
   {
       "DBInstance": {
           "DBInstanceIdentifier": "sample-instance-2",
           "DBInstanceClass": "db.r5.large",
           "Engine": "docdb",
           "DBInstanceStatus": "deleting",
           "Endpoint": {
               "Address": "sample-instance-2.node.us-east-1.docdb.amazonaws.com",
               "Port": 27017,
               "HostedZoneId": "ABCDEFGHIJKLM"
           },
           "InstanceCreateTime": "2020-03-27T08:05:56.314Z",
           "PreferredBackupWindow": "02:00-02:30",
           "BackupRetentionPeriod": 1,
           "VpcSecurityGroups": [
               {
                   "VpcSecurityGroupId": "sg-abcd0123",
                   "Status": "active"
               }
           ],
           "AvailabilityZone": "us-east-1c",
           "DBSubnetGroup": {
               "DBSubnetGroupName": "default",
               "DBSubnetGroupDescription": "default",
               "VpcId": "vpc-6242c31a",
               "SubnetGroupStatus": "Complete",
               "Subnets": [
                   {
   ```

```
                    "SubnetIdentifier": "subnet-abcd0123",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1a"
                    },
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-wxyz0123",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1b"
                    },
                    "SubnetStatus": "Active"
                }
            ]
        },
        "PreferredMaintenanceWindow": "sun:06:53-sun:07:23",
        "PendingModifiedValues": {},
        "EngineVersion": "3.6.0",
        "AutoMinorVersionUpgrade": true,
        "PubliclyAccessible": false,
        "DBClusterIdentifier": "sample-cluster",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "CACertificateIdentifier": "rds-ca-2019",
        "PromotionTier": 1,
        "DBInstanceArn": "arn:aws:rds:us-east-1:<accountID>:db:sample-instance-2",
        "EnabledCloudwatchLogsExports": [
            "profiler"
        ]
    }
}
```

3. 삭제할 인스턴스가 Amazon DocumentDB 클러스터의 마지막 인스턴스인 경우:

   Amazon DocumentDB 클러스터의 마지막 인스턴스를 삭제하는 경우 해당 클러스터와 해당 클러스터에 연결된 자동 스냅샷 및 연속 백업도 삭제합니다.

   클러스터의 마지막 인스턴스를 삭제하려면 클러스터를 삭제하고 선택적으로 최종 스냅샷을 생성할 수 있습니다. 자세한 내용은 Amazon DocumentDB 클러스터 삭제 섹션을 참조하십시오.

삭제 방지

또한 Amazon DocumentDB 클러스터의 마지막 인스턴스를 삭제하면 해당 클러스터와 관련된 자동 스냅샷 및 연속 백업뿐만 아니라 클러스터도 삭제됩니다. Amazon DocumentDB는 AWS Management Console 또는를 사용하여 삭제 작업을 수행하든 관계없이 클러스터에 대한 삭제 보호를 적용합니다 AWS CLI. 삭제 방지가 활성화되어 있으면 클러스터를 삭제할 수 없습니다.

삭제 방지가 활성화된 클러스터를 삭제하려면 먼저 클러스터를 수정하고 삭제 방지를 비활성화해야 합니다. 자세한 내용은 Amazon DocumentDB 클러스터 삭제 섹션을 참조하십시오.

인스턴스 클래스 관리

인스턴스 클래스는 Amazon DocumentDB(MongoDB 호환) 인스턴스의 컴퓨팅 및 메모리 용량을 결정합니다. 필요한 인스턴스 클래스는 DB 인스턴스의 처리력 및 메모리 요구 사항에 따라 다릅니다.

Amazon DocumentDB는 R4, R5, R6G, R8G, T3 및 T4G 인스턴스 클래스 제품군을 지원합니다. 이들 클래스는 메모리 집약적 애플리케이션에 최적화된 최신 세대 인스턴스 클래스입니다. 이들 클래스의 사양은 인스턴스 클래스 사양 섹션을 참조하세요.

주제

- 인스턴스 클래스 결정
- 인스턴스의 클래스 변경
- 리전별 지원되는 인스턴스 클래스
- 인스턴스 클래스 사양

인스턴스 클래스 결정

인스턴스의 클래스를 확인하려면 AWS Management Console 또는 `describe-db-instances` AWS CLI 작업을 사용할 수 있습니다.

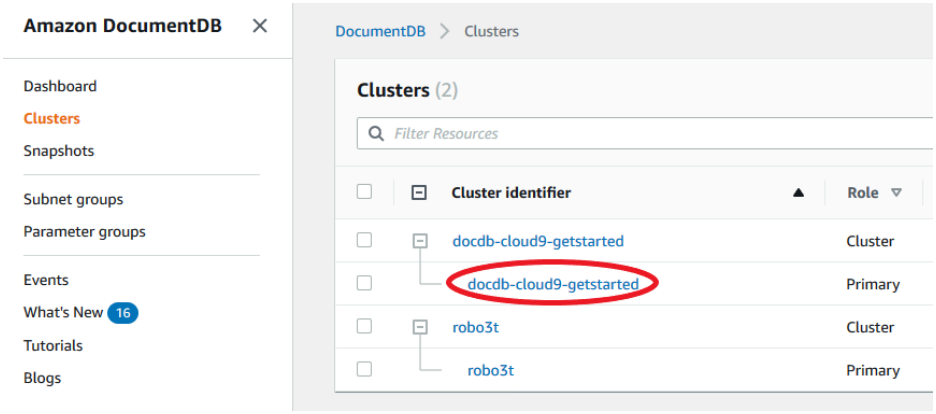
Using the AWS Management Console

클러스터 인스턴스의 인스턴스 클래스를 결정하려면 콘솔에서 다음 단계를 완료하세요.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택하여 관심 있는 인스턴스를 찾습니다.

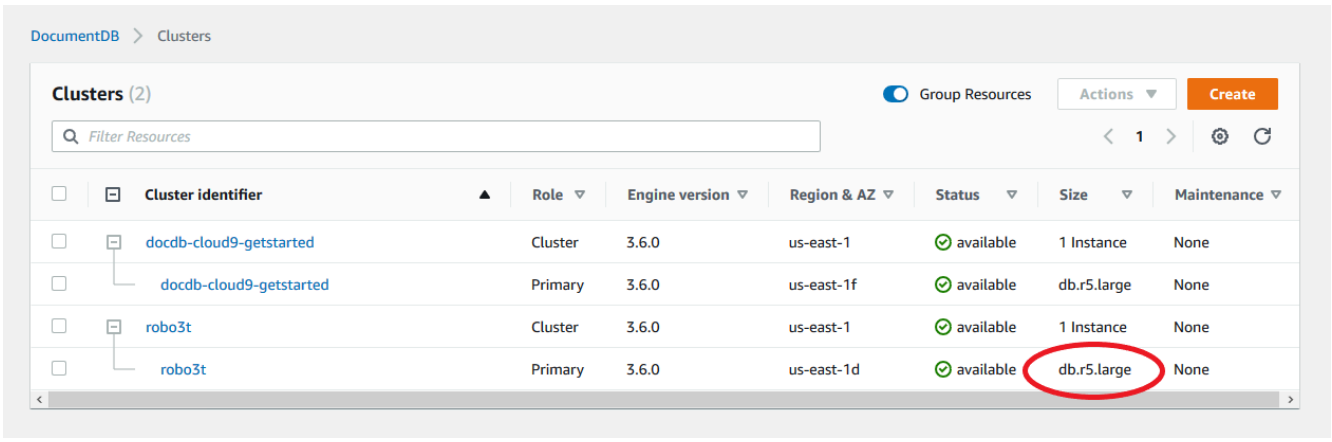
> **Tip**
>
> 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

3. 클러스터 탐색 상자에 클러스터 식별자 열이 표시됩니다. 인스턴스는 다음 스크린샷과 마찬가지로 클러스터 아래에 나열됩니다.



4. 인스턴스 목록에서 클러스터를 확장하여 원하는 인스턴스를 찾으세요. 원하는 인스턴스를 찾습니다. 그런 다음 인스턴스 행의 크기 열을 확인하여 인스턴스 클래스를 확인합니다.

   다음 이미지에서는 인스턴스 `robo3t`의 인스턴스 클래스가 `db.r5.4xlarge`입니다.



Using the AWS CLI

를 사용하여 인스턴스의 클래스를 확인하려면 다음 파라미터와 함께 `describe-db-instances` 작업을 AWS CLI사용합니다.

- **--db-instance-identifier** - 선택 사항입니다. 인스턴스 클래스를 찾을 인스턴스를 지정합니다. 이 매개 변수를 생략한 경우 describe-db-instances는 최대 100개의 인스턴스에 대한 설명을 반환합니다.
- **--query** – 선택 사항. 결과에 포함할 인스턴스 멤버를 지정합니다. 이 매개 변수를 생략하면 모든 인스턴스 멤버가 반환됩니다.

Example

다음 예에서는 모든 인스턴스에 대한 인스턴스 sample-instance-1 이름과 클래스를 찾습니다.

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-instances \
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' \
    --db-instance-identifier sample-instance-1
```

Windows의 경우:

```
aws docdb describe-db-instances ^
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' ^
    --db-instance-identifier sample-instance-1
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
[
    [
        "sample-instance-1",
        "db.r5.large"
    ]
```

Example

다음 예에서는 최대 100개의 Amazon DocumentDB 인스턴스에 대한 인스턴스 이름과 클래스를 찾습니다.

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-instances \
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' \
    --filter Name=engine,Values=docdb
```

Windows의 경우:

```
aws docdb describe-db-instances ^
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' ^
    --filter Name=engine,Values=docdb
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
[
    [
        "sample-instance-1",
        "db.r5.large"
    ],
    [
        "sample-instance-2",
        "db.r5.large"
    ],
    [
        "sample-instance-3",
        "db.r5.4xlarge"
    ],
    [
        "sample-instance-4",
        "db.r5.4xlarge"
    ]
]
```

자세한 내용은 Amazon DocumentDB 인스턴스 설명 단원을 참조하십시오.

인스턴스의 클래스 변경

AWS Management Console 또는를 사용하여 인스턴스의 인스턴스 클래스를 변경할 수 있습니다 AWS CLI. 자세한 내용은 Amazon DocumentDB 인스턴스 수정 단원을 참조하십시오.

리전별 지원되는 인스턴스 클래스

Amazon DocumentDB는 다음 인스턴스 클래스를 지원합니다.

- R8G- R6G 인스턴스보다 최대 30% 더 나은 성능을 제공하는 Arm-based AWS Graviton4 프로세서로 구동되는 메모리 최적화 인스턴스의 최신 세대입니다.

- R6G- Arm-based AWS Graviton2 프로세서로 구동되는 메모리 최적화 인스턴스는 5% 더 저렴한 비용으로 R5 인스턴스보다 최대 30% 더 나은 성능을 제공합니다.
- R6GD - 임시 데이터를 위한 로컬 비휘발성 메모리 익스프레스(NVMe) 기반 솔리드 스테이트 드라이브(SSD) 스토리지가 있는 메모리 최적화 R6G 인스턴스입니다.
- R5 - 동일한 인스턴스 비용으로 R4 인스턴스보다 최대 100% 더 나은 성능을 제공하는 메모리 최적화 인스턴스입니다.
- R4 - 이전 세대의 메모리 최적화 인스턴스입니다.
- T4G- 기준 수준의 CPU 성능을 제공하는 Arm-based AWS Graviton2 프로세서로 구동되는 최신 세대 저비용 버스트 가능 범용 인스턴스 유형으로, T3 인스턴스보다 최대 35% 더 나은 가격 성능을 제공하며 일시적인 사용량 급증이 발생하는 중간 정도의 CPU 사용량이 있는 애플리케이션을 실행하는 데 적합합니다.
- T3 - 필요한 기간 동안 언제든지 CPU 사용량을 버스트할 수 있는 기능과 함께 기본 수준의 CPU 성능을 제공하는 저비용 버스터블 범용 인스턴스 유형입니다.

각 인스턴스 클래스의 세부 사양은 인스턴스 클래스 사양 섹션을 참조하세요.

> **Note**
>
> I/O 최적화 스토리지는 엔진 버전 5.0 및 8.0(인스턴스 기반 클러스터)에서만 사용할 수 있습니다.

특정 인스턴스 클래스가 특정 리전에서 지원되거나 지원되지 않을 수 있습니다. 다음 표에는 각 리전에서 Amazon DocumentDB가 지원하는 인스턴스 클래스가 나와 있습니다.

리전별 지원되는 인스턴스 클래스

리전	R8G	R6GD	R6G	R5	R4	T4G	T3	서버리스
미국 동부(오하이오)	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨
미국 동부(버지니아 북부)	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨

리전	R8G	R6GD	R6G	R5	R4	T4G	T3	서버리스
미국 서부(오리건)	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨
아프리카(케이프타운)			지원됨	지원됨		지원됨	지원됨	지원됨
남아메리카(상파울루)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(홍콩)	지원됨		지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(하이데라바드)			지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(말레이시아)			지원됨			지원됨	지원됨	지원됨
아시아 태평양(뭄바이)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(오사카)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(서울)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(시드니)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(자카르타)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(멜버른)			지원됨	지원됨		지원됨	지원됨	지원됨

리전	R8G	R6GD	R6G	R5	R4	T4G	T3	서버리스
아시아 태평양(싱가포르)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
아시아 태평양(태국)			지원됨			지원됨	지원됨	지원됨
아시아 태평양(도쿄)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
캐나다(중부)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
캐나다 서부(캘거리)			지원됨	지원됨		지원됨	지원됨	지원됨
유럽(프랑크푸르트)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
유럽(취리히)		지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
유럽(아일랜드)	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨	지원됨
유럽(런던)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
유럽(밀라노)			지원됨	지원됨		지원됨	지원됨	지원됨
유럽(파리)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
유럽(스페인)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
유럽(스톡홀름)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
멕시코(중부)			지원됨			지원됨	지원됨	지원됨
중동(UAE)			지원됨	지원됨		지원됨	지원됨	지원됨
중국(베이징)	지원됨	지원됨	지원됨	지원됨		지원됨	지원됨	지원됨
중국(닝샤)	지원됨		지원됨	지원됨		지원됨	지원됨	지원됨

리전	R8G	R6GD	R6G	R5	R4	T4G	T3	서버리스
이스라엘(텔아비브)			지원됨	지원됨		지원됨	지원됨	지원됨
AWS GovCloud(미국 서부)	지원됨	지원됨	지원됨	지원됨			지원됨	지원됨
AWS GovCloud(미국 동부)		지원됨	지원됨	지원됨		지원됨	지원됨	지원됨

인스턴스 클래스 사양

다음 표에는 각 클래스에서 지원되는 인스턴스 유형을 포함하여 Amazon DocumentDB 인스턴스 클래스에 대한 세부 정보가 나와 있습니다. 테이블 아래에서 각 테이블 열에 대한 설명을 찾아볼 수 있습니다.

인스턴스 클래스	vCPU[1]	메모리(GiB)[2]	NVMe SSD 계층형 캐시(GiB)[3]	최대 온도 스토리지(GiB)[4]	기준/버스트 대역폭(Gbps)[5]	지원 엔진[6]
R8G - Graviton4 기반 현재 세대 메모리 최적화 인스턴스 클래스 R8G는 엔진 버전 5.0 및 8.0에서만 사용할 수 있는 최신 인스턴스 패밀리(Graviton4)입니다.						
`db.r8g.large`	2	16	-	30	0.937/12.5	5.0.0 및 8.0.0
`db.r8g.xlarge`	4	32	-	60	1.875/12.5	5.0.0 및 8.0.0
`db.r8g.2xlarge`	8	64	-	121	3.75/15.0	5.0.0 및 8.0.0
`db.r8g.4xlarge`	16	128	-	243	7.5/15.0	5.0.0 및 8.0.0

인스턴스 클래스	vCPU[1]	메모리 (GiB)[2]	NVMe SSD 계층형 캐시(GiB)[3]	최대 온도 스토리지 (GiB)[4]	기준/버스트 대역폭 (Gbps)[5]	지원 엔진[6]
db.r8g.8xlarge	32	256	-	488	15	5.0.0 및 8.0.0
db.r8g.12xlarge	48	384	-	732	22	5.0.0 및 8.0.0
db.r8g.16xlarge	64	512	-	987	30	5.0.0 및 8.0.0
db.r8g.24xlarge	96	768	-	1484	40	5.0.0 및 8.0.0
db.r8g.48xlarge	192	1536	-	2967	50	5.0.0 및 8.0.0
R6G — Graviton2 기반의 현재 세대 메모리 최적화 인스턴스 클래스						
db.r6g.large	2	16	-	32	0.75/10	4.0.0, 5.0.0 및 8.0.0
db.r6g.xlarge	4	32	-	63	1.25/10	4.0.0, 5.0.0 및 8.0.0
db.r6g.2xlarge	8	64	-	126	2.5/10	4.0.0, 5.0.0 및 8.0.0
db.r6g.4xlarge	16	128	-	252	5.0/10	4.0.0, 5.0.0 및 8.0.0
db.r6g.8xlarge	32	256	-	504	12	4.0.0, 5.0.0 및 8.0.0
db.r6g.12xlarge	48	384	-	756	20	4.0.0, 5.0.0 및 8.0.0

인스턴스 클래스	vCPU[1]	메모리 (GiB)[2]	NVMe SSD 계층형 캐시(GiB)[3]	최대 온도 스토리지 (GiB)[4]	기준/버스트 대역폭 (Gbps)[5]	지원 엔진[6]
db.r6g.16xlarge	64	512	-	1008	25	4.0.0, 5.0.0 및 8.0.0
R6G - Graviton2 기반의 현재 세대 NVMe 지원 인스턴스 클래스						
db.r6gd.xlarge	4	32	173	64	1.25/10	5.0.0 및 8.0.0
db.r6gd.2xlarge	8	64	346	128	2.5/10	5.0.0 및 8.0.0
db.r6gd.4xlarge	16	128	694	256	5.0/10	5.0.0 및 8.0.0
db.r6gd.8xlarge	32	256	1388	512	12	5.0.0 및 8.0.0
db.r6gd.12xlarge	48	384	2082	768	20	5.0.0 및 8.0.0
db.r6gd.16xlarge	64	512	2776	1024	25	5.0.0 및 8.0.0
R5 – 이전 세대 메모리 최적화 인스턴스 클래스						
db.r5.large	2	16	-	31	0.75/10	3.6.0, 4.0.0, 5.0.0 및 8.0.0
db.r5.xlarge	4	32	-	62	1.25/10	3.6.0, 4.0.0, 5.0.0 및 8.0.0

인스턴스 클래스	vCPU[1]	메모리 (GiB)[2]	NVMe SSD 계층형 캐시(GiB)[3]	최대 온도 스토리지 (GiB)[4]	기준/버스트 대역폭 (Gbps)[5]	지원 엔진[6]
db.r5.2xlarge	8	64	-	124	2.5/10	3.6.0, 4.0.0, 5.0.0 및 8.0.0
db.r5.4xlarge	16	128	-	249	5.0/10	3.6.0, 4.0.0, 5.0.0 및 8.0.0
db.r5.8xlarge	32	256	-	504	10	3.6.0, 4.0.0, 5.0.0 및 8.0.0
db.r5.12xlarge	48	384	-	748	12	3.6.0, 4.0.0, 5.0.0 및 8.0.0
db.r5.16xlarge	64	512	-	1008	20	3.6.0, 4.0.0, 5.0.0 및 8.0.0
db.r5.24xlarge	96	768	-	1500	25	3.6.0, 4.0.0, 5.0.0 및 8.0.0
R4 – 이전 세대 메모리 최적화 인스턴스 클래스 R4 인스턴스는 엔진 버전 3.6에서만 지원됩니다. Amazon DocumentDB 3.6은 2026년 3월 30일에 표준 지원이 종료됩니다. 추가 지원(유료)은 2029년 3월까지 사용할 수 있습니다.						
db.r4.large	2	15.25	-	30	0.75/10	3.6.0만
db.r4.xlarge	4	30.5	-	60	1.25/10	3.6.0만

인스턴스 클래스	vCPU[1]	메모리 (GiB)[2]	NVMe SSD 계층형 캐시(GiB)[3]	최대 온도 스토리지 (GiB)[4]	기준/버스트 대역폭 (Gbps)[5]	지원 엔진[6]
db.r4.2xlarge	8	61	-	120	2.5/10	3.6.0만
db.r4.4xlarge	16	122	-	240	5.0/10	3.6.0만
db.r4.8xlarge	32	244	-	480	10	3.6.0만
db.r4.16xlarge	64	488	-	960	25	3.6.0만
T4G – Graviton2 기반 최신 세대 성능 버스트 기능이 있는 인스턴스 클래스 T 시리즈 인스턴스는 무제한 CPU 버스트 모드에서 실행됩니다. 기준 이후의 버스트 사용량은 추가 요금이 청구됩니다. 글로벌 클러스터에는 지원되지 않습니다.						
db.t4g.medium	2	4	-	8.13	0.256/5	4.0.0, 5.0.0 및 8.0.0
T3 – 이전 세대 성능 버스트 기능이 있는 인스턴스 클래스						
db.t3.medium	2	4	-	7.5	0.256/5	3.6.0, 4.0.0, 5.0.0 및 8.0.0

인스턴스 클래스	vCPU[1]	메모리 (GiB)[2]	NVMe SSD 계층형 캐시(GiB)[3]	최대 온도 스토리지 (GiB)[4]	기준/버스트 대역폭 (Gbps)[5]	지원 엔진[6]

1. vCPU - 가상 중앙 처리 장치(CPU)의 수입니다. 가상 CPU는 인스턴스 클래스를 비교하는 데 사용할 수 있는 용량을 가진 디바이스입니다. 특정 프로세서를 구매하거나 임차해 몇 개월 또는 몇 년간 사용하는 것이 아니라, 시간 단위로 용량을 임대합니다. 기본 하드웨어 용량에 관계없이 일정 CPU 용량을 제공하는 것이 목표입니다.
2. 메모리(GiB) - 인스턴스에 할당되는 RAM(기가바이트)입니다. 메모리와 vCPU 간 일정한 비율이 존재하는 경우가 많다는 점에 유의하세요.
3. NVMe SSD 계층형 캐시 - 임시 데이터를 저장하기 위한 확장 캐시로 할당된 SSD 볼륨의 공간(기가바이트)입니다. 이 캐시는 NVMe 지원 인스턴스에서만 사용할 수 있습니다.
4. 최대 임시 스토리지(GiB) - 비영구 임시 파일 스토리지용으로 인스턴스에 할당된 RAM(기가바이트)입니다. NVMe 지원 인스턴스의 경우 이 스토리지는 NVMe 기반 SSD 볼륨에서 호스팅됩니다. 다른 모든 인스턴스에서는 Amazon Elastic Block Store(EBS)에서 호스팅됩니다.
5. 기준/버스트 대역폭(Gbps) - 버스트 대역폭은 초당 기가비트 단위의 최대 대역폭을 나타냅니다. 이 값을 8로 나누면 초당 메가바이트 단위로 예상되는 처리량을 구할 수 있습니다. 크기가 4xlarge 이하인 인스턴스에는 기준 대역폭이 있습니다. 추가 요구 사항을 충족하려면 네트워크 I/O 크레딧 메커니즘을 통해 기준 대역폭을 초과하여 확장할 수 있습니다. 인스턴스는 인스턴스 크기에 따라 일반적으로 5분에서 60분까지 제한된 시간 동안 버스트 대역폭을 사용할 수 있습니다.
6. 지원 엔진 - 인스턴스 클래스를 지원하는 Amazon DocumentDB 엔진.

NVMe 지원 인스턴스

NVMe 지원 인스턴스는 일반 인스턴스의 메모리를 초과하는 대규모 데이터세트가 있는 애플리케이션에 대해 최대 7배 더 빠른 쿼리 성능을 제공합니다. 이러한 인스턴스는 r6gd 인스턴스에서 사용할 수 있는 로컬 비휘발성 메모리 익스프레스(NVMe) 기반 솔리드 스테이트 드라이브(SSD) 스토리지를 활용하여 임시 데이터를 저장하고, 네트워크 기반 스토리지 액세스를 줄이며, 읽기 지연 시간과 처리량을 개선합니다.

로컬 SSD 공간은 다음 두 섹션으로 나뉩니다.

- 계층형 캐시 - 로컬 SSD의 약 73%가 데이터베이스 캐시로 할당되므로 시스템이 기본 메모리만 저장할 때보다 최대 5배 더 많은 데이터베이스 페이지를 저장할 수 있습니다. 로컬 SSD는 2계층 캐시 역할을 하는 반면 기존의 인 메모리 버퍼 캐시는 1계층 캐시로 유지됩니다. 쿼리는 버퍼 캐시와 SSD 캐시 모두에 누락이 발생한 경우에만 외부 스토리지에 액세스합니다.

- 임시 스토리지 - 나머지 27%는 비영구 임시 파일 스토리지용으로 예약되어 있으며, 정렬 또는 인덱스 빌드와 같은 리소스 집약적인 작업과 관련된 복잡한 쿼리에 사용됩니다. 일반 인스턴스에서는 임시 공간이 Amazon Elastic Block Store(EBS) 볼륨에 상주합니다. SSD의 로컬 호스팅 임시 스토리지는 정렬과 관련된 쿼리 지연 시간을 최대 2배 줄이고 인덱스 빌드와 같은 리소스 집약적인 작업을 가속화합니다.

지원되는 NVMe 지원 인스턴스 유형 및 사양에 대한 세부 정보는 인스턴스 클래스 사양 섹션에서 확인할 수 있습니다.

주제

- NVMe 지원 인스턴스의 권장 사용 사례
- Amazon DocumentDB에서 NVMe 지원 인스턴스 사용
- NVMe 지원 인스턴스 모니터링

NVMe 지원 인스턴스의 권장 사용 사례

다음 시나리오에서 NVMe 지원 인스턴스를 사용합니다.

- 읽기 중심 워크로드 - 워크로드가 읽기 집약적이고 데이터세트가 버퍼 캐시보다 크며 낮은 `BufferCacheHitRatio` 및 높은 `ReadIOPS` 지표로 표시되는 경우 NVMe 지원 인스턴스는 성능 이점을 제공할 수 있습니다.
- 업데이트가 많은 워크로드 - 워크로드가 업데이트 집약적이고 네트워크 스토리지의 읽기 지연 시간으로 인해 폐영역 회수를 따라잡을 수 없는 경우 NVMe 지원 인스턴스가 문제를 완화하는 데 도움이 될 수 있습니다.

NVMe 지원 인스턴스는 다음과 같은 다양한 사용 사례에 도움이 될 수 있습니다.

- 인터넷 규모의 애플리케이션 - 엄격한 성능의 서비스 수준 계약(SLA)을 사용하는 결제 처리, 결제 및 전자 상거래와 같은 애플리케이션은 NVMe 지원 인스턴스의 성능 이점을 활용할 수 있습니다.
- 실시간 보고 대시보드 - 지표/데이터 수집을 위해 수백 개의 쿼리를 실행하는 대시보드는 NVMe 지원 인스턴스의 짧은 지연 시간과 높은 처리량을 활용할 수 있습니다.
- 생성형 AI 애플리케이션 - 벡터 검색을 사용하여 수백만 개의 벡터 임베딩에서 정확하거나 가장 가까운 인접 항목을 찾는 애플리케이션은 NVMe 지원 인스턴스의 고성능을 활용할 수 있습니다.

Amazon DocumentDB에서 NVMe 지원 인스턴스 사용

Amazon DocumentDB의 NVMe 지원 인스턴스를 사용하려면 다음을 수행합니다.

- Amazon DocumentDB 클러스터를 생성하고 NVMe 지원 인스턴스 클래스 중 하나를 추가합니다. 자세한 내용은 Amazon DocumentDB 클러스터 생성 단원을 참조하십시오.
- 또는 NVMe 지원 인스턴스 클래스 중 하나를 사용하도록 기존 Amazon DocumentDB 클러스터를 수정합니다. 자세한 내용은 Amazon DocumentDB 클러스터 수정 단원을 참조하십시오.

여러 AWS 리전에서 NVMe 지원 인스턴스의 가용성을 확인하려면 섹션을 참조하세요리전별 지원되는 인스턴스 클래스.

NVMe 지원 인스턴스에서 일반 인스턴스로 다시 전환하려면 Amazon DocumentDB 인스턴스의 데이터베이스 인스턴스 클래스를 NVMe 스토리지가 없는 유사한 인스턴스 클래스로 수정합니다. 예를 들어, 현재 인스턴스 클래스가 'db.r6gd.4xlarge'인 경우 'db.r6g.4xlarge'를 선택하여 다시 전환합니다. 자세한 내용은 Amazon DocumentDB 클러스터 수정 단원을 참조하십시오.

NVMe 지원 인스턴스 모니터링

Amazon CloudWatch에서 사용할 수 있는 일반 인스턴스 지표 외에도 NVMe 지원 인스턴스는 NVMe 기반 SSD 스토리지, IOPS 및 처리량과 관련된 추가 지표를 내보냅니다.

```
NVMeStorageCacheHitRatio
FreeNVMeStorage
ReadIOPSNVMeStorage
ReadLatencyNVMeStorage
ReadThroughputNVMeStorage
WriteIOPSNVMeStorage
WriteLatencyNVMeStorage
WriteThroughputNVMeStorage
```

이러한 지표에 대한 자세한 내용은 NVMe 지원 인스턴스 지표 섹션을 참조하세요.

Amazon DocumentDB 서브넷 그룹 관리

Virtual Private Cloud(VPC)는 전용 가상 네트워크입니다 AWS 계정. AWS 클라우드의 다른 가상 네트워크와 논리적으로 격리되어 있습니다. Amazon DocumentDB 클러스터와 같은 AWS 리소스를 Amazon VPC로 시작할 수 있습니다. IP 주소 범위와 VPC 범위를 설정하고 서브넷을 추가하고 보안 그룹을 연결한 다음 라우팅 테이블을 구성합니다.

서브넷은 사용자의 Amazon VPC의 IP 주소 범위입니다. 지정된 서브넷에서 AWS 리소스를 시작할 수 있습니다. 인터넷에 연결해야 할 리소스에 대한 퍼블릭 서브넷을 사용합니다. 인터넷에 연결하지 않을 리소스에 대한 프라이빗 서브넷을 사용합니다. 퍼블릭 및 프라이빗 서브넷에 대한 자세한 내용은 Amazon Virtual 프라이빗 클라우드 사용 설명서에서 VPC 및 서브넷 기본 사항을 참조하세요.

DB 서브넷 그룹은 사용자가 VPC에서 만든 다음 클러스터에 대해 지정하는 서브넷의 모음입니다. 서브넷 그룹에서는 클러스터를 생성할 때 특정 VPC를 지정할 수 있습니다. `default` 서브넷 그룹을 사용하면 해당 그룹은 VPC 내의 모든 서브넷을 포괄합니다.

각 DB 서브넷 그룹은 지정된 리전에서 두 개 이상의 가용 영역에 서브넷이 있어야 합니다. VPC에서 DB 클러스터를 만들 때, DB 서브넷 그룹을 선택해야 합니다. Amazon DocumentDB는 해당 DB 서브넷 그룹과 사용자가 선호하는 가용 영역을 사용하여 해당 서브넷 내에서 클러스터와 연결할 서브넷과 IP 주소를 선택합니다. 기본 인스턴스가 실패할 경우 Amazon DocumentDB가 해당 복제본 인스턴스를 새 기본 인스턴스로 승격할 수 있습니다. 이때 이전의 기본 인스턴스가 위치했던 서브넷의 IP 주소를 사용하여 새 복제본 인스턴스를 생성할 수 있습니다.

Amazon DocumentDB가 VPC에 인스턴스를 생성할 때는 DB 서브넷 그룹에서 선택한 IP 주소를 사용하여 클러스터에 네트워크 인터페이스를 할당합니다. 장애 조치 중에 기본 IP 주소가 변경될 수 있으므로 항상 DNS 이름을 사용합니다. 자세한 내용은 Amazon DocumentDB 엔드포인트 단원을 참조하십시오.

VPC 및 서브넷 생성에 대한 자세한 내용은 Amazon Virtual 프라이빗 클라우드 사용 설명서의 VPC 및 서브넷으로 작업하기를 참조하세요.

주제

- Amazon DocumentDB 서브넷 그룹 생성
- Amazon DocumentDB 서브넷 그룹 설명
- Amazon DocumentDB 서브넷 그룹 수정
- Amazon DocumentDB 서브넷 그룹 삭제

Amazon DocumentDB 서브넷 그룹 생성

Amazon DocumentDB 클러스터를 생성하는 경우 Amazon VPC와 Amazon VPC 내 해당 서브넷 그룹을 선택해야 클러스터를 시작할 수 있습니다. 서브넷은 인스턴스를 시작하기 위해 사용할 가용 영역과 해당 가용 영역 내 IP 범위를 결정합니다.

서브넷 그룹은 Amazon DocumentDB 인스턴스를 시작하는 데 사용하고자 하는 가용 영역을 지정할 수 있는 명명된 서브넷(AZ) 집합입니다. 예를 들어, 인스턴스가 3개인 클러스터에서 각 인스턴스를 별도

의 AZ에 프로비저닝하여 고가용성을 최적화하는 것이 좋습니다. 따라서 단일 AZ가 실패하면 단일 인스턴스에만 영향을 미칩니다.

Amazon DocumentDB 인스턴스는 최대 3개의 가용 영역에 프로비저닝할 수 있습니다. 서브넷 그룹에 서브넷이 3개 이상 있는 경우에도 Amazon DocumentDB 클러스터 생성에는 그 중 3개만 사용할 수 있습니다. 서브넷 그룹을 생성할 때 인스턴스를 배포할 서브넷 3개만 선택합니다.

예: 클러스터가 생성되고 Amazon DocumentDB가 AZ{1A, 1B 및 1C}를 선택합니다. AZ{1D}에서 인스턴스를 생성하려고 하면 API 직접 호출이 실패합니다. 하지만 특정 AZ를 지정하지 않고 인스턴스를 생성하기로 선택하면 Amazon DocumentDB가 사용자를 대신하여 AZ를 선택합니다. Amazon DocumentDB는 알고리즘을 사용하여 여러 AZ에 걸쳐 인스턴스의 부하를 분산하므로 고가용성을 확보할 수 있습니다. 기본적으로 세 개의 인스턴스가 프로비저닝되면 세 개의 AZ에 프로비저닝되고 모든 인스턴스가 단일 AZ에 프로비저닝되지 않습니다.

모범 사례

- 특별한 사유가 없으면 항상 서브넷 그룹 서브넷을 3개 생성합니다. 이렇게 하면 인스턴스가 세 AZ 간에 프로비저닝되므로 인스턴스가 세 개 이상인 클러스터가 더 높은 가용성을 획득할 수 있습니다.
- 항상 여러 AZ에 걸쳐 인스턴스를 분산하여 고가용성을 실현합니다. 단일 AZ에 클러스터의 모든 인스턴스를 배치하지 마십시오.
- 장애 조치 이벤트는 언제든지 발생할 수 있으므로 기본 인스턴스 또는 복제본 인스턴스가 항상 특정 AZ에 있다고 가정해서는 안 됩니다.

서브넷 그룹 생성 방법

AWS Management Console 또는 AWS CLI 를 사용하여 Amazon DocumentDB 서브넷 그룹을 생성할 수 있습니다.

Using the AWS Management Console

다음 단계에 따라 Amazon DocumentDB 서브넷 그룹을 만듭니다.

Amazon DocumentDB 서브넷 그룹 생성

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 서브넷 그룹을 선택한 다음 생성을 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

3. 서브넷 그룹 생성 페이지에서:

   a. 서브넷 그룹 세부 정보 섹션에서:

      i. 이름 — 서브넷 그룹에 대해 의미 있는 이름을 입력합니다.

      ii. 설명 - 서브넷 그룹에 대한 설명을 입력합니다:

   b. 서브넷 추가 섹션에서:

      i. VPC — 목록에서 이 서브넷 그룹에 대한 VPC를 선택합니다.

      ii. 다음 중 하나를 수행하세요.

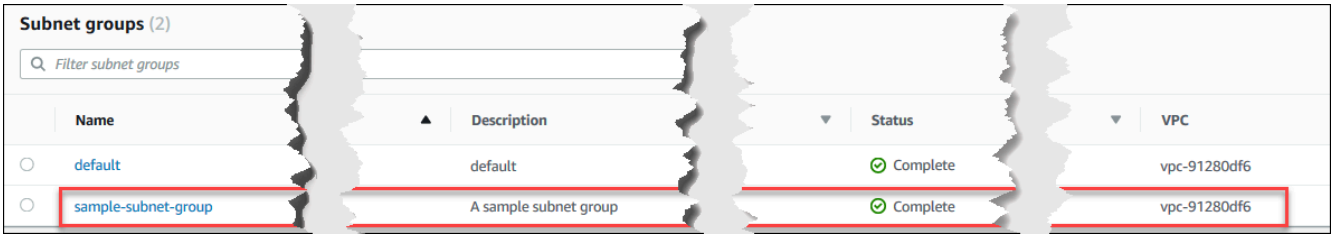
      - 선택된 VPC에 서브넷을 모두 포함하려면 이 VPC와 관련된 모든 서브넷 추가를 선택합니다.
      - 이 서브넷 그룹에 대한 서브넷을 지정하려면, 서브넷을 포함하려는 각 가용 영역에 대해 다음을 수행합니다. 2개 이상의 가용 영역을 포함해야 합니다.

        A. 가용 영역 - 목록에서 가용 영역을 선택합니다.

        B. 서브넷 — 목록에서 이 서브넷 그룹에 대해 선택된 가용 영역의 서브넷을 선택합니다.

        C. 서브넷 추가를 선택합니다.

4. 생성(Create)을 선택합니다. 새로 생성된 서브넷 그룹은 다른 서브넷 그룹과 함께 나열됩니다.



Using the AWS CLI

를 사용하여 서브넷 그룹을 생성하려면 먼저 사용 가능한 서브넷을 결정 AWS CLI해야 합니다. 다음 AWS CLI 작업을 실행하여 가용 영역과 해당 서브넷을 나열합니다.

파라미터:

- **--db-subnet-group** - 선택 사항. 특정 서브넷 그룹을 지정하면 가용 영역과 해당 그룹의 서브넷이 나열됩니다. 이 파라미터를 생략하면 모든 서브넷 그룹의 가용 영역과 서브넷이 나열됩니다. `default` 서브넷 그룹을 지정하면 모든 VPC의 서브넷이 나열됩니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-subnet-groups \
    --db-subnet-group-name default \
    --query 'DBSubnetGroups[*].[DBSubnetGroupName,Subnets[*].
[SubnetAvailabilityZone.Name,SubnetIdentifier]]'
```

Windows의 경우:

```
aws docdb describe-db-subnet-groups ^
    --db-subnet-group-name default ^
    --query 'DBSubnetGroups[*].[DBSubnetGroupName,Subnets[*].
[SubnetAvailabilityZone.Name,SubnetIdentifier]]'
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
[
    [
        "default",
        [
            [
                "us-east-1a",
                "subnet-4e26d263"
            ],
            [
                "us-east-1c",
                "subnet-afc329f4"
            ],
            [
                "us-east-1e",
                "subnet-b3806e8f"
            ],
            [
                "us-east-1d",
```

```
                "subnet-53ab3636"
            ],
            [
                "us-east-1b",
                "subnet-991cb8d0"
            ],
            [
                "us-east-1f",
                "subnet-29ab1025"
            ]
        ]
    ]
]
```

이전 작업의 출력을 사용하여 새 서브넷 그룹을 만들 수 있습니다. 새 서브넷 그룹에는 최소 2개 가용 영역의 서브넷이 포함되어야 합니다.

파라미터:

- **--db-subnet-group-name** - 필수입니다. 이 서브넷 그룹의 이름입니다.
- **--db-subnet-group-description** - 필수입니다. 이 서브넷 그룹에 대한 설명입니다.
- **--subnet-ids** - 필수입니다. 이 서브넷 그룹에 포함하는 서브넷의 목록입니다. 예시: subnet-53ab3636.
- --태그 - 선택사항. 이 서브넷 그룹에 연결할 태그(키/값 페어) 목록입니다.

다음 코드는 subnet-4e26d263, subnet-afc329f4, subnet-b3806e8f의 3개 서브넷으로 구성된 서브넷 그룹 sample-subnet-group을 만듭니다.

Linux, macOS, Unix의 경우:

```
aws docdb create-db-subnet-group \
    --db-subnet-group-name sample-subnet-group \
    --db-subnet-group-description "A sample subnet group" \
    --subnet-ids subnet-4e26d263 subnet-afc329f4 subnet-b3806e8f \
    --tags Key=tag1,Value=One Key=tag2,Value=2
```

Windows의 경우:

```
aws docdb create-db-subnet-group ^
    --db-subnet-group-name sample-subnet-group ^
```

```
    --db-subnet-group-description "A sample subnet group" ^
    --subnet-ids subnet-4e26d263 subnet-afc329f4 subnet-b3806e8f ^
    --tags Key=tag1,Value=One Key=tag2,Value=2
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "DBSubnetGroup": {
        "DBSubnetGroupDescription": "A sample subnet group",
        "DBSubnetGroupName": "sample-subnet-group",
        "Subnets": [
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1a"
                },
                "SubnetIdentifier": "subnet-4e26d263",
                "SubnetStatus": "Active"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1c"
                },
                "SubnetIdentifier": "subnet-afc329f4",
                "SubnetStatus": "Active"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1e"
                },
                "SubnetIdentifier": "subnet-b3806e8f",
                "SubnetStatus": "Active"
            }
        ],
        "VpcId": "vpc-91280df6",
        "DBSubnetGroupArn": "arn:aws:rds:us-east-1:123SAMPLE012:subgrp:sample-
subnet-group",
        "SubnetGroupStatus": "Complete"
    }
}
```

Amazon DocumentDB 서브넷 그룹 설명

AWS Management Console 또는를 사용하여 Amazon DocumentDB 서브넷 그룹의 세부 정보를 AWS CLI 가져올 수 있습니다.

Using the AWS Management Console

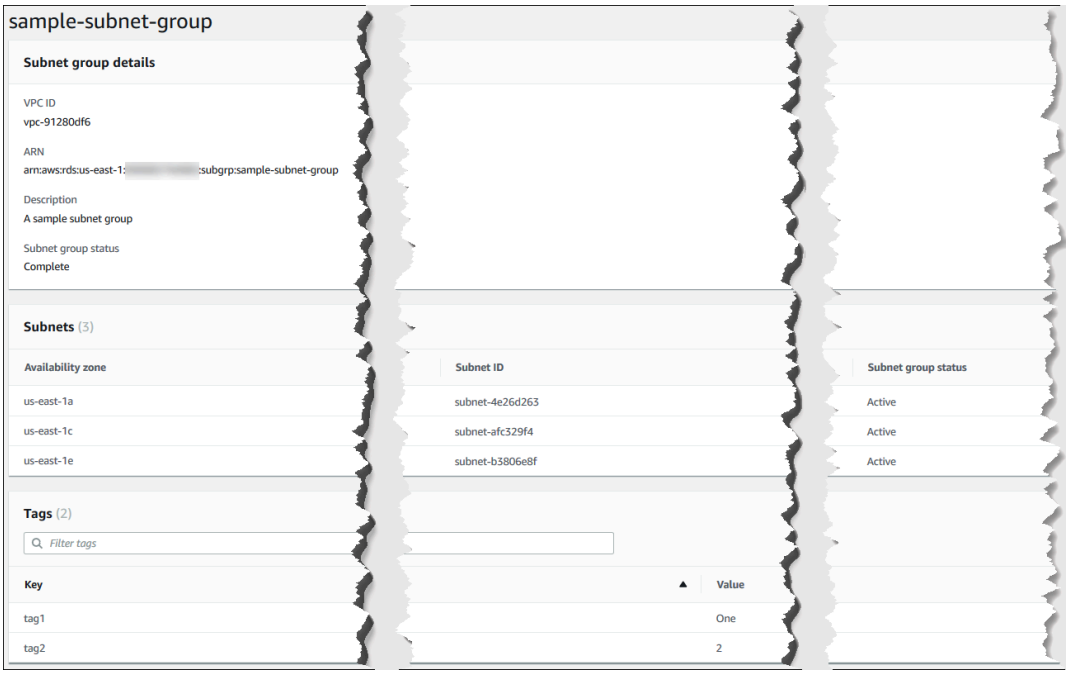
다음 절차에서는 Amazon DocumentDB 서브넷 그룹의 세부 정보 확인 방법을 볼 수 있습니다.

서브넷 그룹의 세부 정보를 찾으려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 서브넷 그룹을 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택합니다.

3. 서브넷 그룹의 세부 정보를 확인하려면 해당 서브넷 그룹의 이름을 선택합니다.



Using the AWS CLI

Amazon DocumentDB 서브넷 그룹의 세부 정보를 찾으려면 다음 파라미터와 함께 `describe-db-subnet-groups` 작업을 사용합니다.

파라미터

- `--db-subnet=group-name` - 선택 사항. 포함된 경우, 명명된 서브넷 그룹의 세부 정보가 나열됩니다. 생략된 경우, 최대 100개 서브넷 그룹의 세부 정보가 나열됩니다.

Example

다음 코드는 Amazon DocumentDB 서브넷 그룹 생성 섹션에 생성한 `sample-subnet-group` 서브넷 그룹의 세부 정보를 나열합니다.

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-subnet-groups \
    --db-subnet-group-name sample-subnet-group
```

Windows의 경우:

```
aws docdb describe-db-subnet-groups ^
    --db-subnet-group-name sample-subnet-group
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "DBSubnetGroup": {
        "DBSubnetGroupArn": "arn:aws:rds:us-east-1:123SAMPLE012:subgrp:sample-
subnet-group",
        "VpcId": "vpc-91280df6",
        "SubnetGroupStatus": "Complete",
        "DBSubnetGroupName": "sample-subnet-group",
        "Subnets": [
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1a"
                },
                "SubnetStatus": "Active",
```

```
                "SubnetIdentifier": "subnet-4e26d263"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1c"
                },
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-afc329f4"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1e"
                },
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-b3806e8f"
            }
        ],
        "DBSubnetGroupDescription": "A sample subnet group"
    }
}
```

Amazon DocumentDB 서브넷 그룹 수정

AWS Management Console 또는를 사용하여 서브넷 그룹의 설명을 AWS CLI 수정하거나 Amazon DocumentDB 서브넷 그룹에서 서브넷을 추가하거나 제거할 수 있습니다. 그러나 `default` 서브넷 그룹은 수정할 수 없습니다.

Using the AWS Management Console

AWS Management Console 를 사용하여 서브넷 그룹의 설명을 변경하거나 서브넷을 추가 및 제거할 수 있습니다. 완료 시 서브넷 그룹과 연결된 가용 영역은 2개 이상이어야 합니다.

서브넷 그룹을 수정하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 서브넷 그룹을 선택합니다. 그런 다음 해당 서브넷 그룹 이름의 왼쪽에 있는 버튼을 선택합니다. `default` 서브넷 그룹은 수정할 수 없습니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

3. 작업을 선택한 후 수정을 선택합니다.
4. 설명- 서브넷 그룹의 설명을 변경하려면 새 설명을 입력합니다.
5. 서브넷 그룹과 연결된 서브넷을 변경하려면 서브넷 추가 섹션에서 다음 중 하나 이상을 수행합니다.
   - 이 서브넷 그룹에서 서브넷을 모두 제거하려면 모두 제거를 선택합니다.
   - 이 서브넷 그룹에서 특정 서브넷을 제거하려면 제거하려는 각 서브넷에 대해 제거를 선택합니다.
   - 이 VPC에 연결된 서브넷을 모두 추가하려면 이 VPC와 관련된 모든 서브넷 추가를 선택합니다.
   - 이 서브넷 그룹에 특정 서브넷을 추가하려면 서브넷을 추가하려는 각 가용 영역에 대해 다음을 수행합니다.
     a. 가용 영역 - 목록에서 새 가용 영역을 선택합니다.
     b. 서브넷 — 목록에서 이 서브넷 그룹에 대해 선택된 가용 영역의 서브넷을 선택합니다.
     c. 서브넷 추가를 선택합니다.
6. 확인 대화 상자에서:
   - 서브넷 그룹에 해당 수정 사항을 적용하려면 수정을 선택합니다.
   - 서브넷 그룹을 변경 없이 유지하려면 취소를 선택합니다.

Using the AWS CLI

AWS CLI 를 사용하여 서브넷 그룹의 설명을 변경하거나 서브넷을 추가 및 제거할 수 있습니다. 완료 시 서브넷 그룹과 연결된 가용 영역은 2개 이상이어야 합니다. `default` 서브넷 그룹은 수정할 수 없습니다.

파라미터:

- `--db-subnet-group-name` - 필수입니다. 수정하는 Amazon DocumentDB 서브넷 그룹의 이름입니다.

- `--subnet-ids` - 필수입니다. 이 수정 사항의 적용이 완료된 후 서브넷 그룹에 포함하려는 모든 서브넷의 목록입니다.

  > **Important**
  >
  > 현재 서브넷 그룹에 속해 있지만 이 목록에는 포함되지 않은 모든 서브넷은 서브넷 그룹에서 제거됩니다. 현재 서브넷 그룹에 포함된 모든 서브넷을 유지하려면 이 목록에 포함시켜야 합니다.

- `--db-subnet-group-description` - 선택 사항. 서브넷 그룹에 대한 설명입니다.

Example

다음 코드는 설명을 수정하고 기존 서브넷을 `subnet-991cb8d0`, `subnet-53ab3636`, `subnet-29ab1025` 서브넷으로 대체합니다.

Linux, macOS, Unix의 경우:

```
aws docdb modify-db-subnet-group \
    --db-subnet-group-name sample-subnet-group \
    --subnet-ids subnet-991cb8d0 subnet-53ab3636 subnet-29ab1025 \
    --db-subnet-group-description "Modified subnet group"
```

Windows의 경우:

```
aws docdb modify-db-subnet-group ^
    --db-subnet-group-name sample-subnet-group ^
    --subnet-ids subnet-991cb8d0 subnet-53ab3636 subnet-29ab1025 ^
    --db-subnet-group-description "Modified subnet group"
```

이 작업의 출력은 다음과 같습니다(JSON 형식). Amazon DocumentDB 서브넷 그룹 생성 섹션에 생성된 것과 동일한 서브넷 그룹입니다. 그러나 서브넷 그룹의 서브넷은 `modify-db-subnet-group` 작업에 나열된 서브넷으로 대체됩니다.

```
{
    "DBSubnetGroup": {
        "DBSubnetGroupArn": "arn:aws:rds:us-east-1:123SAMPLE012:subgrp:sample-
subnet-group",
        "DBSubnetGroupDescription": "Modified subnet group",
        "SubnetGroupStatus": "Complete",
```

```
        "Subnets": [
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1d"
                },
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-53ab3636"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1b"
                },
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-991cb8d0"
            },
            {
                "SubnetAvailabilityZone": {
                    "Name": "us-east-1f"
                },
                "SubnetStatus": "Active",
                "SubnetIdentifier": "subnet-29ab1025"
            }
        ],
        "VpcId": "vpc-91280df6",
        "DBSubnetGroupName": "sample-subnet-group"
    }
}
```

Amazon DocumentDB 서브넷 그룹 삭제

AWS Management Console 또는 AWS CLI 를 사용하여 Amazon DocumentDB 서브넷 그룹을 삭제할 수 있습니다. 그러나 `default` 서브넷 그룹은 삭제할 수 없습니다.

Using the AWS Management Console

AWS Management Console 를 사용하여 서브넷 그룹을 삭제할 수 있습니다. `default` 서브넷 그룹은 삭제할 수 없습니다.

서브넷 그룹을 삭제하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.

2. 탐색 창에서 서브넷 그룹을 선택합니다. 그런 다음 해당 서브넷 그룹 이름의 왼쪽에 있는 버튼을 선택합니다. `default` 서브넷 그룹은 삭제할 수 없습니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

3. 작업을 선택한 후 삭제를 선택합니다.
4. 확인 대화 상자에서:
   - 서브넷 그룹을 삭제하려면 삭제를 선택합니다.
   - 서브넷 그룹을 유지하려면 취소를 선택합니다.

Using the AWS CLI

를 사용하여 Amazon DocumentDB 서브넷 그룹을 삭제하려면 다음 파라미터와 함께 `delete-db-subnet-group` 작업을 AWS CLI사용합니다.

파라미터

- `--db-subnet-group-name` - 필수입니다. 삭제할 Amazon DocumentDB 서브넷 그룹의 이름입니다. `default` 서브넷 그룹은 삭제할 수 없습니다.

Example

다음 코드를 사용하여 `sample-subnet-group`을 삭제할 수 있습니다.

Linux, macOS, Unix의 경우:

```
aws docdb delete-db-subnet-group \
    --db-subnet-group-name sample-subnet-group
```

Windows의 경우:

```
aws docdb delete-db-subnet-group ^
    --db-subnet-group-name sample-subnet-group
```

이 작업은 출력을 생성하지 않습니다.

Amazon DocumentDB 고가용성 및 복제

복제본 인스턴스를 사용하여 Amazon DocumentDB(MongoDB 호환)에서 고가용성 및 읽기 규모 조정을 구현할 수 있습니다. 단일 Amazon DocumentDB 클러스터는 1개의 기본 인스턴스와 최대 15개의 복제본 인스턴스를 지원합니다. 이러한 인스턴스는 해당 클러스터의 리전 내의 가용 영역에 두루 분산할 수 있습니다. 기본 인스턴스는 읽기 및 쓰기 트래픽을 허용하며, 복제본 인스턴스는 읽기 요청만 허용합니다.

클러스터 볼륨은 클러스터에 대한 여러 개의 데이터 사본으로 구성됩니다. 그러나 Amazon DocumentDB 클러스터의 기본 인스턴스 및 복제본에는 클러스터 볼륨 데이터가 단 하나의 논리 볼륨으로 표시됩니다. 복제본 인스턴스는 최종적으로 일관됩니다. 복제본 인스턴스 복제본 지연 시간을 최소화하여 쿼리 결과를 반환합니다. 이 지연 시간은 일반적으로 기본 인스턴스가 업데이트를 적용한 후 100밀리초 미만입니다. 데이터베이스 변경률에 따라 달라집니다. 즉, 데이터베이스의 쓰기 연산이 많은 기간에는 복제 지연 시간이 증가할 수 있습니다.

읽기 확장

Amazon DocumentDB 복제본은 클러스터 볼륨의 읽기 작업에 전적으로 사용되므로 읽기 규모 조정에 유용합니다. 쓰기 연산은 기본 인스턴스에서 관리합니다. 클러스터 볼륨은 클러스터에 있는 모든 인스턴스에 공유됩니다. 따라서 각 Amazon DocumentDB 복제본에 대한 데이터 사본을 복제하여 보관할 필요가 없습니다.

높은 가용성

Amazon DocumentDB 클러스터를 만들면 서브넷 그룹 내 가용 영역의 수에 따라(2개 이상이어야 함) Amazon DocumentDB가 가용 영역에서 인스턴스를 프로비저닝합니다. 클러스터에서 인스턴스를 생성할 때 Amazon DocumentDB가 서브넷 그룹의 가용 영역에 인스턴스를 자동으로 배포하여 클러스터의 균형을 잡습니다. 또한 이 작업을 통해 모든 인스턴스가 동일한 가용 영역에 위치하는 경우를 방지할 수 있습니다.

예제

이해를 돕기 위해 세 개의 가용 영역(AZ1, AZ2 및 AZ3)으로 구성된 서브넷 그룹이 있는 클러스터를 만든다고 가정해 보겠습니다.

클러스터의 첫 번째 인스턴스가 만들어지면 이는 기본 인스턴스로서 세 가용 영역 중 하나에 위치합니다. 이 예에서는 AZ1입니다. 두 번째로 만드는 인스턴스는 복제 인스턴스로서 나머지 두 가용 영역 중

하나, 즉 AZ2에 위치합니다. 세 번째 인스턴스는 복제 인스턴스이며 나머지 한 가용 영역, AZ3에 위치합니다. 추가로 인스턴스를 만들 경우 가용 영역에 배포하여 클러스터의 균형을 달성할 수 있습니다.

기본 인스턴스(AZ1)에 장애가 발생하면 장애 조치가 트리거되고 기존의 복제 인스턴스 중 하나가 기본 인스턴스로 승격됩니다. 기존의 기본 인스턴스가 복구되면 이는 프로비저닝되었던 동일 가용 영역(AZ1)에서 복제 인스턴스로 지정됩니다. 3개 인스턴스 클러스터를 프로비저닝하면 Amazon DocumentDB는 해당 3개 인스턴스 클러스터를 계속 보존합니다. Amazon DocumentDB는 수동 개입 없이 인스턴스 장애 감지, 장애 조치 및 복구를 자동으로 처리합니다.

Amazon DocumentDB의 장애 조치로 인스턴스를 복구하는 경우, 복구된 인스턴스는 처음 프로비저닝된 가용 영역에 그대로 위치합니다. 그러나 인스턴스의 역할은 기본에서 복제본으로 변경될 수 있습니다. 이렇게 함으로써 여러 번의 장애 조치로 모든 인스턴스가 동일한 가용 영역에 모이는 경우를 방지할 수 있습니다.

Amazon DocumentDB 복제본을 장애 조치 대상으로 지정할 수 있습니다. 즉 기본 인스턴스에 장애가 발생하면 지정된 Amazon DocumentDB 복제본이나 티어의 복제본이 기본 인스턴스로 승격됩니다. 기본 인스턴스에 대한 읽기/쓰기 요청이 예외로 인해 장애가 발생하는 동안에는 시스템이 짧게 중단됩니다. Amazon DocumentDB 클러스터에 Amazon DocumentDB 복제본이 없는 경우에는 기본 인스턴스 장애 시 해당 인스턴스가 다시 생성됩니다. Amazon DocumentDB 복제본을 승격시키는 것이 기본 인스턴스를 다시 생성하는 것보다 훨씬 빠릅니다.

고가용성을 위해 서로 다른 가용 영역의 기본 인스턴스와 동일한 인스턴스 클래스의 Amazon DocumentDB 복제본을 하나 이상 생성합니다.

자세한 내용은 다음을 참조하세요.

- Amazon DocumentDB 클러스터 내결함성에 대한 이해
- Amazon DocumentDB 장애 조치
  - 장애 조치 대상 제어

Amazon DocumentDB 버전 5.0 및 8.0에서는 기본 인스턴스가 다시 시작될 때 복제본 인스턴스가 다시 시작되지 않습니다. 기본 인스턴스를 다시 시작하는 동안 읽기 요청을 계속 처리하여 가용성을 개선합니다.

글로벌 클러스터를 통한 고가용성

여러에서 고가용성을 위해 Amazon DocumentDB 글로벌 클러스터를 설정할 AWS 리전수 있습니다. 각 글로벌 클러스터는 여러 리전에 걸쳐 있으므로 지연 시간이 짧은 글로벌 읽기와 리전 간 중단으로부

터 재해 복구가 가능합니다. Amazon DocumentDB는 기본 리전에서 각 보조 리전으로의 모든 데이터 및 업데이트 복제를 자동으로 처리합니다.

복제본 추가

클러스터에 추가된 첫 번째 인스턴스는 기본 인스턴스입니다. 첫 번째 인스턴스 이후 추가된 모든 인스턴스는 복제본 인스턴스입니다. 클러스터에는 기본 인스턴스 외에 최대 15개의 복제본 인스턴스가 있을 수 있습니다.

를 사용하여 클러스터를 생성하면 AWS Management Console기본 인스턴스가 동시에 자동으로 생성됩니다. 클러스터 및 기본 인스턴스를 생성하는 것과 동시에 복제본을 생성하려면 Create replica in different zone(다른 영역에 복제본 만들기)을 선택합니다. 자세한 내용은 Amazon DocumentDB 클러스터 생성의 4.d 단계를 참조하십시오. Amazon DocumentDB 클러스터에 복제본을 추가하려면 클러스터에 Amazon DocumentDB 인스턴스 추가 단원을 참조하십시오.

를 사용하여 클러스터 AWS CLI 를 생성할 때는 기본 및 복제본 인스턴스를 명시적으로 생성해야 합니다. 자세한 내용은 다음 주제의 "사용 AWS CLI" 섹션을 참조하세요.

- Amazon DocumentDB 클러스터 생성
- 클러스터에 Amazon DocumentDB 인스턴스 추가

Amazon DocumentDB 장애 조치

특정 유형의 계획된 유지 관리 또는 예상치 못한 기본 노드 또는 가용 영역 장애의 경우와 같은 특정 경우에 Amazon DocumentDB(MongoDB 호환)는 장애를 감지하고 기본 노드를 대체합니다. 장애 조치를 실시하는 동안 쓰기 시간은 최소화됩니다. 이러한 경우는 기본 노드의 역할이 새 기본 노드를 생성하여 프로비저닝하는 대신 읽기 전용 복제본 중 하나에 장애 조치를 실행하기 때문에 발생합니다. 이 장애 탐지 및 복제본 승격을 통해 승격이 완료되는 즉시 새 기본 노드에 작성을 재개할 수 있습니다.

장애 조치가 작동하려면 클러스터에 최소 두 개의 인스턴스(기본 인스턴스 및 적어도 한 개의 복제본 인스턴스)가 있어야 합니다.

> **Note**
>
> 이 주제는 원본 Amazon DocumentDB 인스턴스 기반 클러스터에만 적용됩니다. 탄력적 또는 글로벌 클러스터에는 적용되지 않습니다.

장애 조치 대상 제어

Amazon DocumentDB에서는 장애 조치가 발생할 때 기본 인스턴스로 승격시킬 복제본 인스턴스를 제어하기 위한 수단으로 장애 조치 티어를 제공합니다.

장애 조치 티어

각 복제본 인스턴스는 장애 조치 티어(0–15)와 연결됩니다. 유지 관리 또는 예상치 못한 하드웨어 장애로 인해 장애 조치가 발생하면 기본 인스턴스는 우선 순위가 가장 높은 복제본(가장 낮은 번호의 티어)으로 장애 조치됩니다. 여러 복제본의 우선 순위 티어가 동일한 경우 이전의 기본 인스턴스의 크기와 가장 가까운 티어의 복제본에 기본 인스턴스가 장애 조치됩니다.

선택한 복제본 그룹에 대한 장애 조치 티어를 0(가장 높은 우선 순위)으로 설정하여 장애 조치에서 해당 그룹의 복제본 중 하나를 승격시키도록 할 수 있습니다. 우선 순위가 낮은 티어(높은 번호)를 이러한 복제본에 할당하여 장애 조치 시 특정 복제본이 기본 인스턴스로 승격되지 않도록 효과적으로 차단할 수 있습니다. 이는 애플리케이션에서 특정 복제본의 사용량이 높고 복제본 중 하나에 장애 조치하면 중요 애플리케이션에 부정적인 영향을 미치는 경우에 유용합니다.

잠재적 장애 조치 대상인 복제본은 기본 인스턴스와 동일한 인스턴스 클래스를 사용하는 것이 좋습니다. 이렇게 하면 장애 조치가 OS 업데이트 또는 기타 유지 관리 중에 복제본을 승격하는 경우 쓰기 성능 저하를 방지할 수 있습니다.

인스턴스를 생성할 때 또는 나중에 수정하여 인스턴스의 장애 조치 티어를 설정할 수 있습니다. 인스턴스를 수정하여 인스턴스 장애 조치를 설정하면 장애 조치가 트리거되지 않습니다. 자세한 정보는 다음 주제를 참조하세요.

- 클러스터에 Amazon DocumentDB 인스턴스 추가
- Amazon DocumentDB 인스턴스 수정

장애 조치를 수동으로 시작할 때 두 가지 방법, 즉 앞에서 설명한 장애 조치 티어와 `--target-db-instance-identifier` 파라미터를 사용하여 기본 인스턴스로 승격되는 복제본 인스턴스를 제어할 수 있습니다.

`--target-db-instance-identifier`

테스트를 위해 `failover-db-cluster` 연산을 사용하여 장애 조치 이벤트를 강제할 수 있습니다. `--target-db-instance-identifier` 파라미터를 사용하여 기본 인스턴스로 승격할 복제본을 지정합니다. `--target-db-instance-identifier` 파라미터 사용이 장애 조치 우선 순위 티어에 우선합니다. `--target-db-instance-identifier` 파라미터를 지정하지 않은 경우 기본 장애 조치가 장애 조치 우선 순위 티어를 따릅니다.

장애 조치 중에 발생하는 일

장애 조치는 Amazon DocumentDB가 자동적으로 처리하므로 관리자의 개입 없이 애플리케이션에서 최대한 신속하게 데이터베이스 작업을 재개할 수 있습니다.

- 장애 조치 시 동일하거나 다른 가용 영역에 Amazon DocumentDB 복제본 인스턴스가 있는 경우 Amazon DocumentDB는 인스턴스의 정식 이름 레코드(CNAME)를 플립하여 정상 복제본을 가리키고, 이는 다시 새 기본 복제본으로 승격됩니다. 일반적으로 장애 조치는 처음부터 끝까지 30초 이내에 완료됩니다.
- Amazon DocumentDB 복제본 인스턴스(예: 단일 인스턴스 클러스터)가 없는 경우 Amazon DocumentDB는 원래 인스턴스와 동일한 가용 영역에 새 인스턴스 생성을 시도합니다. 이와 같은 원래 인스턴스 대체가 최선의 방법이며, 가용 영역에 크게 영향을 주는 문제가 발생하는 경우 등에는 성공하지 못할 수도 있습니다.

데이터베이스 연결이 끊어지는 경우 애플리케이션에서 연결을 다시 시도해야 합니다.

장애 조치 테스트

클러스터에 대한 장애 조치는 클러스터에 있는 Amazon DocumentDB 복제본(읽기 전용 인스턴스) 중 하나를 기본 인스턴스(클러스터 라이터)로 승격시킵니다.

기본 인스턴스가 실패하면 Amazon DocumentDB 복제본이 존재하는 경우 Amazon DocumentDB 복제본으로 Amazon DocumentDB가 자동적으로 장애 조치됩니다. 테스트를 위해 기본 인스턴스의 실패를 시뮬레이션하려는 경우 장애 조치를 강제할 수 있습니다. 클러스터의 각 인스턴스에는 자체적인 엔드포인트 주소가 있습니다. 장애 조치가 완료되면 엔드포인트 주소를 사용하도록 기존 연결을 정리한 후 다시 설정해야 합니다.

장애 조치를 강제하려면 `failover-db-cluster` 연산을 해당 파라미터와 함께 사용합니다.

- `--db-cluster-identifier` - 필수입니다. 장애 조치할 클러스터의 이름입니다.
- `--target-db-instance-identifier`—선택 사항. 기본 인스턴스로 승격시킬 인스턴스의 이름입니다.

Example

다음 작업은 `sample-cluster` 클러스터의 장애 조치를 강제로 실행합니다. 새 기본 인스턴스를 만들 인스턴스를 지정하지 않아 Amazon DocumentDB는 장애 조치 티어 우선 순위에 따라 인스턴스를 선택합니다.

Linux, macOS, Unix의 경우:

```
aws docdb failover-db-cluster \
   --db-cluster-identifier sample-cluster
```

Windows의 경우:

```
aws docdb failover-db-cluster ^
   --db-cluster-identifier sample-cluster
```

다음 작업은 sample-cluster-instance를 기본 역할로 승격하도록 지정하여 sample-cluster 클러스터의 장애 조치를 강제로 실행합니다. 출력의 "IsClusterWriter": true를 참고하세요.

Linux, macOS, Unix의 경우:

```
aws docdb failover-db-cluster \
   --db-cluster-identifier sample-cluster \
   --target-db-instance-identifier sample-cluster-instance
```

Windows의 경우:

```
aws docdb failover-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --target-db-instance-identifier sample-cluster-instance
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "DBCluster": {
        "HostedZoneId": "Z2SUY0A1719RZT",
        "Port": 27017,
        "EngineVersion": "3.6.0",
        "PreferredMaintenanceWindow": "thu:04:05-thu:04:35",
        "BackupRetentionPeriod": 1,
        "ClusterCreateTime": "2018-06-28T18:53:29.455Z",
        "AssociatedRoles": [],
        "DBSubnetGroup": "default",
        "MasterUsername": "master-user",
        "Engine": "docdb",
        "ReadReplicaIdentifiers": [],
        "EarliestRestorableTime": "2018-08-21T00:04:10.546Z",
        "DBClusterIdentifier": "sample-cluster",
```

```
        "ReaderEndpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "DBClusterMembers": [
            {
                "DBInstanceIdentifier": "sample-cluster-instance",
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1,
                "IsClusterWriter": true
            },
            {
                "DBInstanceIdentifier": "sample-cluster-instance-00",
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1,
                "IsClusterWriter": false
            },
            {
                "DBInstanceIdentifier": "sample-cluster-instance-01",
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1,
                "IsClusterWriter": false
            }
        ],
        "AvailabilityZones": [
            "us-east-1b",
            "us-east-1c",
            "us-east-1a"
        ],
        "DBClusterParameterGroup": "default.docdb3.6",
        "Endpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "IAMDatabaseAuthenticationEnabled": false,
        "AllocatedStorage": 1,
        "LatestRestorableTime": "2018-08-22T21:57:33.904Z",
        "PreferredBackupWindow": "00:00-00:30",
        "StorageEncrypted": false,
        "MultiAZ": true,
        "Status": "available",
        "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster",
        "VpcSecurityGroups": [
            {
                "Status": "active",
                "VpcSecurityGroupId": "sg-12345678"
            }
        ],
        "DbClusterResourceId": "cluster-ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    }
```

```
}
```

복제 지연

복제 지연은 일반적으로 50밀리초 이하입니다. 복제 지연이 증가하는 가장 일반적인 이유는 다음과 같습니다.

- 기본 복제본의 쓰기 속도가 높으면 읽기 전용 복제본이 기본 복제본보다 뒤쳐집니다.
- 장기 실행 쿼리(예: 대규모 순차 스캔, 집계 쿼리)와 수신되는 쓰기 복제 간의 읽기 전용 복제본 경합.
- 읽기 전용 복제본에 대한 동시 쿼리 수가 매우 많습니다.

복제 지연을 최소화하려면 다음 문제 해결 방법을 시도해 보십시오.

- 쓰기 속도가 높거나 CPU 사용률이 높은 경우 클러스터의 인스턴스를 확장합니다.
- 읽기 전용 복제본에서 쿼리가 오래 실행되고 쿼리 대상 문서가 매우 자주 업데이트되는 경우, 장기 실행 쿼리를 변경하거나 기본/쓰기 복제본에 대해 실행하여 읽기 전용 복제본에서 충돌이 발생하지 않도록 하는 것이 좋습니다.
- 동시 쿼리 수가 매우 많거나 읽기 전용 복제본에서만 CPU 사용률이 높은 경우, 읽기 전용 복제본의 수를 스케일 아웃하여 워크로드를 분산하는 것도 또 다른 방법입니다.
- 높은 쓰기 처리량과 장기 실행 쿼리로 인해 복제 지연이 발생하므로 `DBClusterReplicaLagMaximum` CloudWatch 지표를 느린 쿼리 로거 및 `WriteThroughput`/`WriteIOPS` 지표와 함께 사용하여 복제 지연 문제를 해결합니다.

클러스터 장애 조치로 인해 성능이 저하되지 않도록 모든 복제본의 인스턴스 유형이 동일한지 확인합니다.

스케일 업과 스케일 아웃 중 하나를 선택하는 경우(예: 작은 인스턴스 6개 vs. 대형 인스턴스 3개), DB 인스턴스당 버퍼 캐시가 더 커지므로 일반적으로 스케일 아웃 전에 먼저 스케일 업(대형 인스턴스) 하는 것이 좋습니다.

애플리케이션 기능에 영향을 미치기 전에, 복제본 인스턴스의 데이터가 얼마나 뒤처질 수 있는지(또는 "부실"하게 될 수 있는지) 사전에 복제 지연 경보를 설정하고 해당 임계값의 상한선을 설정해야 합니다. 일반적으로 일시적인 워크로드로 인해 경보가 울리기 전에 여러 데이터 포인트에 대해 복제 지연 임계값을 초과하는 것이 좋습니다.

> **Note**
>
> 10초를 초과하는 복제 지연에 대해 또 다른 경보를 설정합니다. 여러 데이터 포인트에 대해이 임계값을 초과하는 경우 인스턴스를 확장하거나 기본 인스턴스의 쓰기 처리량을 줄입니다.

Amazon DocumentDB 이벤트 관리

Amazon DocumentDB(MongoDB 호환)는 클러스터, 인스턴스, 스냅샷, 보안 그룹 및 클러스터 매개 변수 그룹과 관련된 이벤트 기록을 보관합니다. 여기에는 이벤트 날짜 및 시간, 이벤트의 원본 이름 및 유형, 이벤트 관련 메시지 등의 정보가 포함됩니다.

> **Important**
>
> 특정 관리 기능의 경우, Amazon DocumentDB는 Amazon RDS 및 Amazon Neptune과 공유하는 운영 기술을 사용한다. 영역 제한, 즉 영역 수준에서 관리되는 제한은 Amazon DocumentDB, Amazon RDS 및 Amazon Neptune 간에 공유됩니다. 자세한 내용은 리전별 할당량 단원을 참조하십시오.

주제

- Amazon DocumentDB 이벤트 범주 보기
- Amazon DocumentDB 이벤트 보기

Amazon DocumentDB 이벤트 범주 보기

각 Amazon DocumentDB 리소스 유형에는 관련될 수 있는 특정 유형의 이벤트가 있습니다. 작업을 사용하여 AWS CLI `describe-event-categories` 이벤트 유형과 Amazon DocumentDB 리소스 유형 간의 매핑을 볼 수 있습니다.

파라미터

- **`--source-type`**—선택 사항. `--source-type` 파라미터를 사용하여 특정 소스 유형의 이벤트 범주를 봅니다. 다음은 허용되는 값입니다:
  - `db-cluster`
  - `db-instance`
  - `db-parameter-group`

  - `db-security-group`
  - `db-cluster-snapshot`
- **`--filters`**—선택 사항. Amazon DocumentDB에 대한 이벤트 범주를 보려면 필터`--filter Name=engine,Values=docdb`를 사용합니다.

Example

다음 코드는 클러스터와 연결된 이벤트 범주를 나열합니다.

Linux, macOS, Unix의 경우:

```
aws docdb describe-event-categories \
    --filter Name=engine,Values=docdb \
    --source-type db-cluster
```

Windows의 경우:

```
aws docdb describe-event-categories ^
    --filter Name=engine,Values=docdb ^
    --source-type db-cluster
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "EventCategoriesMapList": [
        {
            "EventCategories": [
                "notification",
                "failure",
                "maintenance",
                "failover"
            ],
            "SourceType": "db-cluster"
        }
    ]
}
```

다음 코드는 각 Amazon DocumentDB 원본 유형과 관련된 이벤트 범주를 나열합니다.

```
aws docdb describe-event-categories
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "EventCategoriesMapList": [
        {
            "SourceType": "db-instance",
            "EventCategories": [
                "notification",
                "failure",
                "creation",
                "maintenance",
                "deletion",
                "recovery",
                "restoration",
                "configuration change",
                "read replica",
                "backtrack",
                "low storage",
                "backup",
                "availability",
                "failover"
            ]
        },
        {
            "SourceType": "db-security-group",
            "EventCategories": [
                "configuration change",
                "failure"
            ]
        },
        {
            "SourceType": "db-parameter-group",
            "EventCategories": [
                "configuration change"
            ]
        },
        {
            "SourceType": "db-cluster",
            "EventCategories": [
                "notification",
                "failure",
                "maintenance",
                "failover"
            ]
```

```
        },
        {
            "SourceType": "db-cluster-snapshot",
            "EventCategories": [
                "backup"
            ]
        }
    ]
}
```

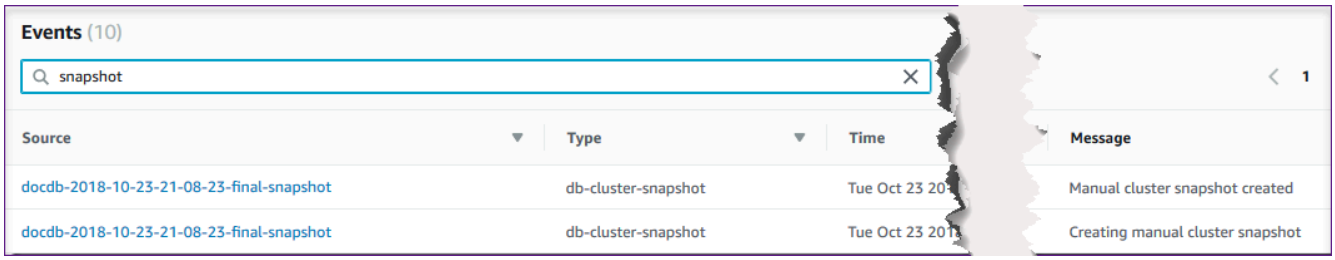
Amazon DocumentDB 이벤트 보기

지난 24시간 동안의 이벤트를 보여주는 Amazon DocumentDB 콘솔을 통해 Amazon DocumentDB 리소스에 대한 이벤트를 검색할 수 있습니다. 설명이벤트 AWS CLI 명령 또는 설명 이벤트 Amazon DocumentDB API 작업을 사용하여 Amazon DocumentDB 리소스에 대한 이벤트를 검색할 수도 있습니다. AWS CLI 또는 Amazon DocumentDB API를 사용하여 이벤트를 보는 경우 최대 지난 14일 동안의 이벤트를 검색할 수 있습니다.

Using the AWS Management Console

지난 24시간 동안 발생한 모든 Amazon DocumentDB 인스턴스 이벤트를 보려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 이벤트를 선택합니다. 사용 가능한 이벤트가 목록에 표시됩니다.
3. 필터 목록을 사용하여 유형별로 이벤트를 필터링합니다. 텍스트 상자에 단어를 입력하여 결과를 추가로 필터링할 수 있습니다. 예를 들어, 다음 스크린샷은 스냅샷 이벤트에 대한 모든 Amazon DocumentDB 이벤트를 필터링하는 것을 보여줍니다.



Using the AWS CLI

지난 7일 동안의 모든 Amazon DocumentDB 인스턴스 이벤트를 보려면 다음과 같이 하십시오

`--duration` 파라미터가 `10080` (10,080분)로 설정된 상태에서 describe-events AWS CLI 작업을 실행하여 지난 7일 동안의 모든 Amazon DocumentDB 인스턴스 이벤트를 볼 수 있습니다.

```
aws docdb describe-events --duration 10080
```

Amazon DocumentDB 이벤트 필터링

특정 Amazon DocumentDB 이벤트를 보려면 다음 매개변수를 사용하여 `describe-events` 작업을 수행합니다.

파라미터

- **`--filter`**—반환된 값을 Amazon DocumentDB 이벤트로 제한하는 데 필요합니다. Amazon DocumentDB의 모든 이벤트를 **`Name=engine,Values=docdb`** 필터링하는 데만 사용합니다.
- **`--source-identifier`**—선택 사항. 반환되는 이벤트에 대한 이벤트 소스의 식별자입니다. 생략하면 모든 소스의 이벤트가 결과에 포함됩니다.
- **`--source-type`**—선택 사항, 다만 `--source-identifier`이 제공되지 않을 경우, 필수 사항. `--source-identifier`를 제공한 경우 `--source-type`은 `--source-identifier` 유형에 동의해야 합니다. 다음은 허용되는 값입니다:
  - `db-cluster`
  - `db-instance`
  - `db-parameter-group`
  - `db-security-group`
  - `db-cluster-snapshot`

다음 예제에서는 모든 Amazon DocumentDB 이벤트를 나열합니다.

```
aws docdb describe-events --filters Name=engine,Values=docdb
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "Events": [
        {
            "SourceArn": "arn:aws:rds:us-east-1:123SAMPLE012:db:sample-cluster-
instance3",
```

```
            "Message": "instance created",
            "SourceType": "db-instance",
            "Date": "2018-12-11T21:17:40.023Z",
            "SourceIdentifier": "sample-cluster-instance3",
            "EventCategories": [
                "creation"
            ]
        },
        {
            "SourceArn": "arn:aws:rds:us-
east-1:123SAMPLE012:db:docdb-2018-12-11-21-08-23",
            "Message": "instance shutdown",
            "SourceType": "db-instance",
            "Date": "2018-12-11T21:25:01.245Z",
            "SourceIdentifier": "docdb-2018-12-11-21-08-23",
            "EventCategories": [
                "availability"
            ]
        },
        {
            "SourceArn": "arn:aws:rds:us-
east-1:123SAMPLE012:db:docdb-2018-12-11-21-08-23",
            "Message": "instance restarted",
            "SourceType": "db-instance",
            "Date": "2018-12-11T21:25:11.441Z",
            "SourceIdentifier": "docdb-2018-12-11-21-08-23",
            "EventCategories": [
                "availability"
            ]
        }
    ]
}
```

자세한 내용은 Amazon DocumentDB 이벤트 감사 단원을 참조하십시오.

Amazon DocumentDB 클러스터 파라미터 그룹 관리

클러스터 파라미터 그룹에서 파라미터를 사용하여 Amazon DocumentDB 엔진 구성을 관리할 수 있습니다. 클러스터 파라미터 그룹은 Amazon DocumentDB 클러스터의 파라미터를 보다 쉽게 관리할 수 있는 Amazon DocumentDB 구성 값 모음입니다. 클러스터 파라미터 그룹은 클러스터의 모든 인스턴스에 적용되는 엔진 구성 값의 컨테이너 역할을 합니다.

이 단원에서는 클러스터 파라미터 그룹을 생성, 확인 및 수정하는 방법에 대해 설명합니다. 또한 지정된 클러스터와 연결된 클러스터 파라미터 그룹을 확인하는 방법을 보여줍니다.

주제

- Amazon DocumentDB 클러스터 파라미터 그룹 설명
- Amazon DocumentDB 클러스터 파라미터 그룹 생성
- Amazon DocumentDB 클러스터 파라미터 그룹 수정
- 사용자 지정 클러스터 파라미터 그룹을 사용하도록 Amazon DocumentDB 클러스터 수정
- Amazon DocumentDB 클러스터 파라미터 그룹 복사
- Amazon DocumentDB 클러스터 파라미터 그룹 재설정
- Amazon DocumentDB 클러스터 파라미터 그룹 삭제
- Amazon DocumentDB 클러스터 파라미터 참조

Amazon DocumentDB 클러스터 파라미터 그룹 설명

`default` 클러스터 파라미터 그룹은 새에서 첫 번째 Amazon DocumentDB 클러스터를 생성 AWS 리전 하거나 새 엔진을 사용할 때 자동으로 생성됩니다. 동일한 리전에서 생성되고 엔진 버전이 동일한 후속 클러스터는 `default` 클러스터 파라미터 그룹으로 생성됩니다.

주제

- Amazon DocumentDB 클러스터 파라미터 그룹의 세부 정보 설명
- Amazon DocumentDB 클러스터의 파라미터 그룹 확인
- Amazon DocumentDB 클러스터 파라미터 그룹과 연결된 클러스터 및 인스턴스 확인

Amazon DocumentDB 클러스터 파라미터 그룹의 세부 정보 설명

지정된 클러스터 파라미터 그룹의 세부 정보를 설명하려면 AWS Management Console 또는 AWS Command Line Interface ()를 사용하여 다음 단계를 완료합니다AWS CLI.

Using the AWS Management Console

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택하십시오.

3. 클러스터 파라미터 그룹 창에서 세부 정보를 보려는 파라미터 그룹의 이름을 선택합니다.
4. 결과 페이지에는 파라미터 그룹의 파라미터, 최근 활동 및 태그가 표시됩니다.

   - 클러스터 파라미터에서 파라미터의 이름, 현재 값, 허용되는 값, 파라미터의 수정 가능 여부, 적용 유형, 데이터 유형 및 설명을 볼 수 있습니다. 파라미터를 선택한 다음 클러스터 파라미터 섹션에서 편집을 선택하여 개별 파라미터를 수정할 수 있습니다. 자세한 내용은 Amazon DocumentDB 클러스터 파라미터 수정(을)를 참조하세요.
   - 최근 이벤트에서 이 파라미터 그룹에 대한 가장 최근 이벤트를 볼 수 있습니다. 이 섹션의 검색 창을 사용하여 이러한 이벤트를 필터링할 수 있습니다. 자세한 내용은 Amazon DocumentDB 이벤트 관리(을)를 참조하세요.
   - 태그에서 이 클러스터 파라미터 그룹에 있는 태그를 볼 수 있습니다. 태그 섹션에서 편집을 선택하여 태그를 추가하거나 제거할 수 있습니다. 자세한 내용은 Amazon DocumentDB 리소스 태그 지정(을)를 참조하세요.

Using the AWS CLI

`describe-db-cluster-parameter-groups` AWS CLI 명령을 사용하여 단일 클러스터 파라미터 그룹 또는 Amazon DocumentDB에 대해 보유한 모든 클러스터 파라미터 그룹의 Amazon 리소스 이름(ARN), 패밀리, 설명 및 이름을 볼 수 있습니다. `describe-db-cluster-parameters` AWS CLI 명령을 사용하여 단일 클러스터 파라미터 그룹 내에서 파라미터와 해당 세부 정보를 볼 수도 있습니다.

- **`--describe-db-cluster-parameter-groups`** - 모든 클러스터 파라미터 그룹 및 세부 정보 목록을 확인합니다.
  - **`--db-cluster-parameter-group-name`** - 선택 사항. 설명하려는 클러스터 파라미터 그룹의 이름입니다. 이 파라미터를 생략할 경우 모든 클러스터 파라미터 그룹을 설명합니다.
- **`--describe-db-cluster-parameters`** - 파라미터 그룹 내의 모든 파라미터와 해당 값을 나열합니다.

- **--db-cluster-parameter-group name** - 필수입니다. 설명하려는 클러스터 파라미터 그룹의 이름입니다.

Example

다음 코드에는 최대 100개의 클러스터 파라미터 그룹과 해당 ARN, 패밀리, 설명 및 이름이 나열되어 있습니다.

```
aws docdb describe-db-cluster-parameter-groups
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
          "DBClusterParameterGroups": [
              {
                  "DBClusterParameterGroupArn": "arn:aws:rds:us-
east-1:012345678912:cluster-pg:default.docdb4.0",
                  "DBParameterGroupFamily": "docdb4.0",
                  "Description": "Default cluster parameter group for docdb4.0",
                  "DBClusterParameterGroupName": "default.docdb4.0"
              },
              {
                  "DBClusterParameterGroupArn": "arn:aws:rds:us-
east-1:012345678912:cluster-pg:sample-parameter-group",
                  "DBParameterGroupFamily": "docdb4.0",
                  "Description": "Custom docdb4.0 parameter group",
                  "DBClusterParameterGroupName": "sample-parameter-group"
              }
          ]
}
```

Example

다음 코드에는 `sample-parameter-group`에 대한 ARN, 패밀리, 설명 및 이름이 나열되어 있습니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb describe-db-cluster-parameter-groups \
          --db-cluster-parameter-group-name sample-parameter-group
```

Windows의 경우:

```
aws docdb describe-db-cluster-parameter-groups ^
          --db-cluster-parameter-group-name sample-parameter-group
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
          "DBClusterParameterGroups": [
              {
                  "DBClusterParameterGroupArn": "arn:aws:rds:us-
east-1:123456789012:cluster-pg:sample-parameter-group",
                  "Description": "Custom docdb4.0 parameter group",
                  "DBParameterGroupFamily": "docdb4.0",
                  "DBClusterParameterGroupName": "sample-parameter-group"
              }
          ]
}
```

Example

다음 코드에는 `sample-parameter-group`의 파라미터 값이 나열되어 있습니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

Windows의 경우:

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name sample-parameter-group
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
   "Parameters": [
```

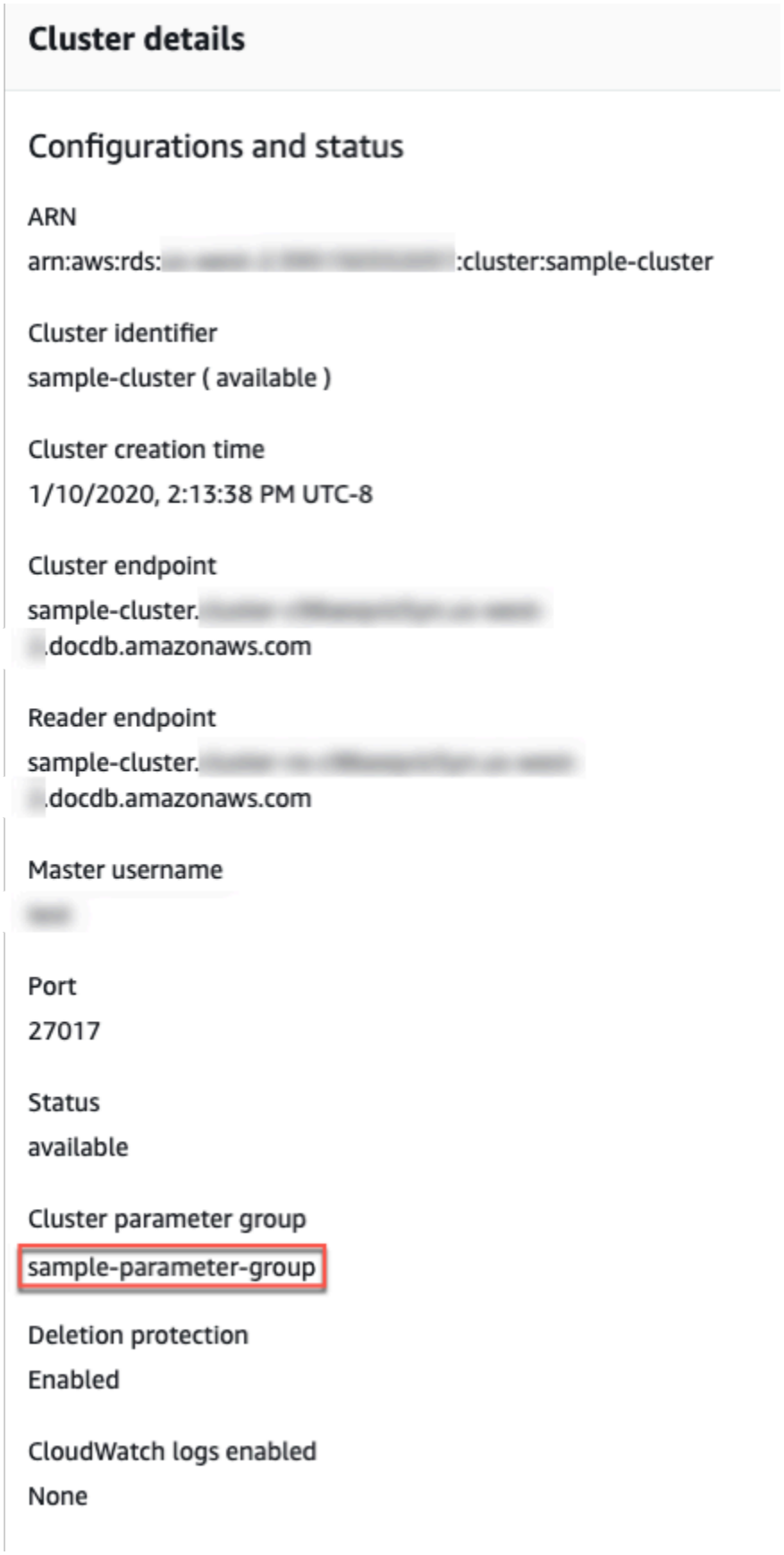
```
        {
           "ParameterName": "audit_logs",
           "ParameterValue": "disabled",
           "Description": "Enables auditing on cluster.",
           "Source": "system",
           "ApplyType": "dynamic",
           "DataType": "string",
           "AllowedValues": "enabled,disabled",
           "IsModifiable": true,
           "ApplyMethod": "pending-reboot"
        },
        {
           "ParameterName": "change_stream_log_retention_duration",
           "ParameterValue": "17777",
           "Description": "Duration of time in seconds that the change stream log
 is retained and can be consumed.",
           "Source": "user",
           "ApplyType": "dynamic",
           "DataType": "integer",
           "AllowedValues": "3600-86400",
           "IsModifiable": true,
           "ApplyMethod": "pending-reboot"
        }
   ]
}
```

Amazon DocumentDB 클러스터의 파라미터 그룹 확인

특정 클러스터와 연결된 파라미터 그룹을 확인하려면 AWS Management Console 또는를 사용하여 다음 단계를 완료합니다 AWS CLI.

Using the AWS Management Console

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 좌측 탐색 창에서 클러스터를 선택합니다.
3. 클러스터 목록에서 원하는 클러스터 이름을 선택합니다.
4. 결과 페이지에는 선택한 클러스터의 세부 정보가 표시됩니다. 클러스터 세부 정보까지 아래로 스크롤합니다. 해당 섹션의 하단에 있는 클러스터 파라미터 그룹 아래에서 파라미터 그룹의 이름을 찾습니다.


Cluster details
Configurations and status
ARN
arn:aws:rds: :cluster:sample-cluster
Cluster identifier
sample-cluster (available)
Cluster creation time
1/10/2020, 2:13:38 PM UTC-8
Cluster endpoint
sample-cluster.
.docdb.amazonaws.com
Reader endpoint
sample-cluster.
.docdb.amazonaws.com
Master username
Port
27017
Status
available
Cluster parameter group
sample-parameter-group
Deletion protection
Enabled
CloudWatch logs enabled
None

Using the AWS CLI

다음 AWS CLI 코드는 클러스터를 관리하는 파라미터 그룹을 결정합니다`sample-cluster`.

```
aws docdb describe-db-clusters \
      --db-cluster-identifier sample-cluster \
      --query 'DBClusters[*].[DBClusterIdentifier,DBClusterParameterGroup]'
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
[
       [
           "sample-cluster",
           "sample-parameter-group"
       ]
]
```

Amazon DocumentDB 클러스터 파라미터 그룹과 연결된 클러스터 및 인스턴스 확인

각 파라미터 그룹과 연결된 클러스터 및 인스턴스 수를 확인하려면를 사용하여 다음 단계를 완료합니다 AWS CLI.

Using the AWS CLI

다음 AWS CLI 코드는와 연결된 클러스터 및 인스턴스 수를 결정합니다`sample-parameter-group`.

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-clusters \
      --query 'DBClusters[*].
[DBClusterParameterGroup,DBClusterIdentifier,DBClusterMembers[*].DBInstanceIdentifier]'
 \
      --output json | \
      jq -r 'group_by(.[0]) | map({
         parameter_group_name: .[0][0],
         total_clusters: length,
         total_instances: map(.[2] | length) | add // 0
      }) | .[]'
```

Windows의 경우:

```
aws docdb describe-db-clusters ^
      --query 'DBClusters[*].
[DBClusterParameterGroup,DBClusterIdentifier,DBClusterMembers[*].DBInstanceIdentifier]'
 ^
      --output json | ^
      jq -r 'group_by(.[0]) | map({
         parameter_group_name: .[0][0],
         total_clusters: length,
         total_instances: map(.[2] | length) | add // 0
      }) | .[]'
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
   "parameter_group_name": "sample-parameter-group",
   "total_clusters": 50,
   "total_instances": 150
}
```

Amazon DocumentDB 클러스터 파라미터 그룹 생성

`default.docdb5.0`, `default.docdb4.0`또는와 같은 기본 클러스터 파라미터 그룹은 새 엔진 버전 및 로 클러스터를 생성할 때 `default.docdb3.6`생성됩니다 AWS 리전. 동일한 리전 및 동일한 엔진 버전으로 생성된 후속 클러스터는 `default` 클러스터 파라미터 그룹을 상속합니다. 한 번 생성된 `default` 파라미터 그룹은 삭제하거나 이름을 바꿀 수 없습니다. 선호하는 파라미터 값을 사용하여 사용자 지정 파라미터 그룹을 생성하고 Amazon DocumentDB 클러스터에 연결하여 클러스터 인스턴스의 엔진 동작을 수정할 수 있습니다.

다음 절차에서는 사용자 지정 클러스터 파라미터 그룹을 생성하는 방법을 안내합니다. 그런 다음 해당 파라미터 그룹 내에서 파라미터를 수정할 수 있습니다.

> **Note**
>
> 클러스터 파라미터 그룹을 생성한 후 해당 특정 파라미터 그룹을 사용하려면 5분 이상 기다려야 합니다. 그러면 클러스터 파라미터 그룹이 새 클러스터에 사용되기 전에 Amazon DocumentDB에서 `create` 작업을 완전히 마칠 수 있습니다. AWS Management Console 또는 `describe-db-cluster-parameter-groups` AWS CLI 작업을 사용하여 클러스터 파라미

터 그룹이 생성되었는지 확인할 수 있습니다. 자세한 내용은 Amazon DocumentDB 클러스터 파라미터 그룹 설명 단원을 참조하십시오.

Using the AWS Management Console

클러스터 파라미터 그룹을 생성하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.

   > **Tip**
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택하십시오.

3. 클러스터 파라미터 그룹 창에서 생성을 선택합니다.
4. 클러스터 파라미터 그룹 만들기 창에서 다음을 입력합니다.

   a. 새 클러스터 파라미터 그룹 이름 - 클러스터 파라미터 그룹의 이름을 입력합니다. 예를 들어 `sample-parameter-group`입니다. 클러스터 파라미터 그룹에는 다음과 같은 명명 제약 조건이 있습니다.

   - [1-255]자 길이의 영숫자 문자입니다.
   - 첫 번째 문자는 글자이어야 합니다.
   - 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.

   b. 패밀리 - 클러스터에 사용할 DocumentDB 버전을 선택합니다.

   c. 설명 - 이 클러스터 파라미터 그룹에 대한 설명을 제공합니다.

5. 클러스터 파라미터 그룹을 생성하려면 생성을 선택합니다. 작업을 취소하려면 취소를 선택합니다.
6. 생성을 선택하면 클러스터 파라미터 그룹이 성공적으로 생성되었는지 확인하기 위해 페이지 상단에 다음 텍스트가 나타납니다.

```
Successfully created cluster parameter group 'sample-parameter-group'.
```

Using the AWS CLI

Amazon DocumentDB 4.0 클러스터에 AWS CLI `create-db-cluster-parameter-group` 대한 새 클러스터 파라미터 그룹을 생성하려면 다음 파라미터와 함께 작업을 사용합니다.

- **`--db-cluster-parameter-group-name`** - 사용자 지정 클러스터 파라미터 그룹의 이름입니다. 예를 들어 `sample-parameter-group`입니다.
- **`--db-cluster-parameter-group-family`** - 사용자 지정 클러스터 파라미터 그룹의 템플릿으로 사용되는 클러스터 파라미터 그룹 패밀리입니다.
- **`--description`** - 이 클러스터 파라미터 그룹에 대해 사용자가 제공한 설명입니다. 다음 예에는 "`Custom docdb4.0 parameter group`"이 사용됩니다.

Linux, macOS 또는 Unix의 경우:

Example

```
aws docdb create-db-cluster-parameter-group \
 --db-cluster-parameter-group-name sample-parameter-group \
 --db-parameter-group-family docdb5.0 \
 --description "Custom docdb5.0 parameter group"
```

Windows의 경우:

```
aws docdb create-db-cluster-parameter-group ^
 --db-cluster-parameter-group-name sample-parameter-group ^
 --db-parameter-group-family docdb5.0 ^
 --description "Custom docdb5.0 parameter group"
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "sample-parameter-group",
        "DBParameterGroupFamily": "docdb5.0",
        "Description": "Custom docdb4.0 parameter group",
        "DBClusterParameterGroupArn": "sample-parameter-group-arn"
    }
}
```

Amazon DocumentDB 클러스터 파라미터 그룹 수정

이 단원에서는 사용자 지정 Amazon DocumentDB 파라미터 그룹을 수정하는 방법에 대해 설명합니다. Amazon DocumentDB에서는 새 리전에서 새 엔진 버전으로 `default` 클러스터를 처음 생성할 때 생성되는 클러스터 파라미터 그룹을 수정할 수 없습니다. Amazon DocumentDB 클러스터가 기본 클러스터 파라미터 그룹을 사용하고 있는 경우, 이 그룹의 값을 수정하려면 먼저 새 파라미터 그룹 생성을 하거나 기존 파라미터 그룹 복사를 하여 수정한 다음 수정된 파라미터 그룹을 클러스터에 적용해야 합니다.

사용자 지정 클러스터 파라미터 그룹을 수정하려면 다음 단계를 완료합니다. 수정 작업이 전파되는 데 시간이 걸릴 수 있습니다. 수정된 클러스터 파라미터 그룹을 클러스터에 연결하기 전에 사용할 수 있을 때까지 기다립니다. AWS Management Console 또는 AWS CLI `describe-db-cluster-parameters` 작업을 사용하여 클러스터 파라미터 그룹이 수정되었는지 확인할 수 있습니다. 자세한 내용은 클러스터 파라미터 그룹 설명 단원을 참조하십시오.

Using the AWS Management Console

사용자 지정 Amazon DocumentDB 파라미터 그룹을 수정하려면 다음 단계를 수행합니다. `default` 파라미터 그룹은 수정할 수 없습니다. `default` 파라미터 그룹의 값을 수정하려면 기본 클러스터 파라미터 그룹을 복사하여 수정한 다음 수정된 파라미터 그룹을 클러스터에 적용하면 됩니다. 클러스터에 파라미터 그룹을 적용하는 방법에 대한 자세한 내용은 Amazon DocumentDB 클러스터 수정(을)를 참조하세요.

사용자 지정 클러스터 파라미터 그룹을 수정하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 파라미터 그룹을 선택합니다. 파라미터 그룹 목록에서 수정하고자 하는 파라미터 그룹의 이름을 선택합니다.

   > **Tip**
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택합니다.

3. 수정하고자 하는 파라미터 그룹의 각 파라미터에 대해 다음을 수행합니다.

a. 수정하려는 파라미터를 찾은 다음, 수정 가능 열에 `true`로 나열되어 있는지 확인하여 해당 파라미터가 수정 가능한지 확인합니다.

b. 수정할 수 있는 경우 해당 파라미터를 선택하고 콘솔 페이지의 오른쪽 상단에서 편집을 선택합니다.

c. 수정 **<parameter-name>** 대화 상자에서 원하는 대로 변경합니다. 그런 다음 Modify cluster parameter(클러스터 파라미터 수정)를 선택하거나 취소를 선택하여 변경 사항을 무시합니다.

Using the AWS CLI

AWS CLI를 사용하여 사용자 지정 Amazon DocumentDB 클러스터 파라미터 그룹에서 수정 가능한 파라미터의 `ParameterValue`, `Description` 또는 `ApplyMethod`를 수정할 수 있습니다. 기본 클러스터 파라미터 그룹을 직접 수정할 수는 없습니다.

사용자 지정 클러스터 파라미터 그룹의 파라미터를 수정하려면 다음 파라미터와 함께 `modify-db-cluster-parameter-group` 작업을 사용합니다.

- **`--db-cluster-parameter-group-name`** - 필수입니다. 수정하려는 클러스터 파라미터 그룹의 이름입니다.
- **`--parameters`** - 필수입니다. 수정 중인 파라미터입니다. Amazon DocumentDB 클러스터의 모든 인스턴스에 적용되는 파라미터 목록은 Amazon DocumentDB 클러스터 파라미터 참조(을)를 참조하세요. 각 파라미터 요소는 다음을 포함해야 합니다.
  - **`ParameterName`** - 수정 중인 파라미터의 이름입니다.
  - **`ParameterValue`** - 이 파라미터의 새 값입니다.
  - **`ApplyMethod`** - 파라미터에 대한 변경 사항을 적용할 방법입니다. 허용된 값은 `immediate` 및 `pending-reboot`입니다.

> **Note**
>
> `static`의 `ApplyType` 파라미터에는 `pending-reboot`의 `ApplyMethod`이 있어야 합니다.

Example- 파라미터 값 수정

이 예에서는 `sample-parameter-group`의 파라미터 값을 나열하고 `tls` 파라미터를 수정합니다. 그런 다음 5분 후에 다시 `sample-parameter-group`의 파라미터 값을 나열하여 변경된 파라미터 값을 확인합니다.

1. `sample-parameter-group`의 파라미터 및 해당 값을 나열합니다.

   Linux, macOS 또는 Unix의 경우:

   ```
   aws docdb describe-db-cluster-parameters \
          --db-cluster-parameter-group-name sample-parameter-group
   ```

   Windows의 경우:

   ```
   aws docdb describe-db-cluster-parameters ^
          --db-cluster-parameter-group-name sample-parameter-group
   ```

   이 작업의 출력은 다음과 같습니다(JSON 형식).

   ```
   {
          "Parameters": [
              {
                  "Source": "system",
                  "ApplyType": "static",
                  "AllowedValues": "disabled,enabled",
                  "ParameterValue": "enabled",
                  "ApplyMethod": "pending-reboot",
                  "DataType": "string",
                  "ParameterName": "tls",
                  "IsModifiable": true,
                  "Description": "Config to enable/disable TLS"
              },
              {
                  "Source": "user",
                  "ApplyType": "dynamic",
                  "AllowedValues": "disabled,enabled",
                  "ParameterValue": "enabled",
                  "ApplyMethod": "pending-reboot",
                  "DataType": "string",
                  "ParameterName": "ttl_monitor",
                  "IsModifiable": true,
   ```

```
                "Description": "Enables TTL Monitoring"
            }
        ]
}
```

2. 해당 값이 `disabled`가 되도록 `tls` 파라미터를 수정합니다.

   `ApplyType`은 `static`이므로 `ApplyMethod`를 수정할 수 없습니다.

   Linux, macOS 또는 Unix의 경우:

```
aws docdb modify-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group \
       --parameters
 "ParameterName"=tls,"ParameterValue"=disabled,"ApplyMethod"=pending-reboot
```

   Windows의 경우:

```
aws docdb modify-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group ^
       --parameters
 "ParameterName"=tls,"ParameterValue"=disabled,"ApplyMethod"=pending-reboot
```

   이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
       "DBClusterParameterGroupName": "sample-parameter-group"
   }
```

3. 적어도 5분을 기다립니다.

4. `sample-parameter-group`의 파라미터 값을 나열하여 `tls` 파라미터가 수정되었는지 확인합니다.

   Linux, macOS 또는 Unix의 경우:

```
aws docdb describe-db-cluster-parameters \
       --db-cluster-parameter-group-name sample-parameter-group
```

   Windows의 경우:

```
aws docdb describe-db-cluster-parameters ^
       --db-cluster-parameter-group-name sample-parameter-group
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
       "Parameters": [
           {
               "ParameterValue": "false",
               "ParameterName": "enable_audit_logs",
               "ApplyType": "dynamic",
               "DataType": "string",
               "Description": "Enables auditing on cluster.",
               "AllowedValues": "true,false",
               "Source": "system",
               "IsModifiable": true,
               "ApplyMethod": "pending-reboot"
           },
           {
               "ParameterValue": "disabled",
               "ParameterName": "tls",
               "ApplyType": "static",
               "DataType": "string",
               "Description": "Config to enable/disable TLS",
               "AllowedValues": "disabled,enabled",
               "Source": "system",
               "IsModifiable": true,
               "ApplyMethod": "pending-reboot"
           }
       ]
}
```

사용자 지정 클러스터 파라미터 그룹을 사용하도록 Amazon DocumentDB 클러스터 수정

Amazon DocumentDB 클러스터를 생성하면 해당 클러스터에 대해 `default.docdb4.0` 파라미터 그룹이 자동으로 생성됩니다. `default` 클러스터 파라미터 그룹은 수정할 수 없습니다. 대신 Amazon DocumentDB 클러스터를 수정하여 새로 사용자 지정된 파라미터 그룹을 연결할 수 있습니다.

이 섹션에서는 AWS Management Console 및 AWS Command Line Interface ()를 사용하여 사용자 지정 클러스터 파라미터 그룹을 사용하도록 기존 Amazon DocumentDB 클러스터를 수정하는 방법을 설명합니다AWS CLI.

Using the AWS Management Console

기본이 아닌 새로운 클러스터 파라미터 그룹을 사용하도록 Amazon DocumentDB 클러스터를 수정하려면

1. 시작하기 전에 Amazon DocumentDB 클러스터와 클러스터 파라미터 그룹을 생성했는지 확인합니다. 자세한 내용은 Amazon DocumentDB 클러스터 생성 및 Amazon DocumentDB 클러스터 파라미터 그룹 생성(을)를 참조하세요.
2. 클러스터 파라미터 그룹을 생성한 후 https://console.aws.amazon.com/docdb에서 Amazon DocumentDB 콘솔을 엽니다. 탐색 창에서 클러스터를 선택하여 새 파라미터 그룹을 클러스터에 추가합니다.
3. 파라미터 그룹을 연결할 클러스터를 선택합니다. 작업을 선택한 다음 수정을 선택하여 클러스터를 수정합니다.
4. 클러스터 옵션에서 클러스터를 연결할 새 파라미터 그룹을 선택합니다.
5. 수정 사항의 요약을 보려면 계속을 선택합니다.
6. 변경 사항을 확인한 후 즉시 적용하거나 수정 일정 아래의 다음 유지 관리 기간 중에 적용할 수 있습니다.
7. 클러스터를 새 파라미터 그룹으로 업데이트하려면 클러스터 수정을 선택합니다.

Using the AWS CLI

시작하기 전에 Amazon DocumentDB 클러스터와 클러스터 파라미터 그룹을 생성했는지 확인합니다. 작업을 사용하여 Amazon DocumentDB 클러스터를 AWS CLI `create-db-cluster` 생성할 수 있습니다. `create-db-cluster-parameter-group` 작업을 사용하여 클러스터 파라미터 그룹을 생성할 수 있습니다 AWS CLI .

새 클러스터 파라미터 그룹을 클러스터에 추가하려면 다음 파라미터와 함께 `modify-db-cluster` 작업을 사용합니다 AWS CLI .

- --db-cluster-identifier - 클러스터의 이름(예: `sample-cluster`)입니다.
- --db-cluster-parameter-group-name - 클러스터를 연결할 파라미터 그룹의 이름(예: `sample-parameter-group`)입니다.

Example

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --db-cluster-parameter-group-name sample-parameter-group
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
"DBCluster": {
    "AvailabilityZones": [
            "us-west-2c",
            "us-west-2b",
            "us-west-2a"
    ],
    "BackupRetentionPeriod": 1,
    "DBClusterIdentifier": "sample-cluster",
    "DBClusterParameterGroup": "sample-parameter-group",
    "DBSubnetGroup": "default",
   ...
}
```

Amazon DocumentDB 클러스터 파라미터 그룹 복사

AWS Management Console 또는 AWS Command Line Interface ()를 사용하여 Amazon DocumentDB에서 클러스터 파라미터 그룹의 복사본을 만들 수 있습니다AWS CLI.

Using the AWS Management Console

다음 절차에서는 기존 클러스터 파라미터 그룹의 복사본을 만들어 새 클러스터 파라미터 그룹을 만드는 방법을 안내합니다.

클러스터 파라미터 그룹을 복사하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.
3. 클러스터 파라미터 그룹 창에서 복사하려는 클러스터 파라미터 그룹의 이름을 선택합니다.
4. 작업을 선택한 후 복사를 선택하여 해당 파라미터 그룹을 복사합니다.

5. 복사 옵션에서 새 클러스터 파라미터 그룹의 이름과 설명을 입력합니다. 그런 다음 복사를 선택하여 변경 사항을 저장합니다.

Using the AWS CLI

클러스터 파라미터 그룹을 복사하려면 다음 파라미터와 함께 `copy-db-cluster-parameter-group` 작업을 사용합니다.

- **`--source-db-cluster-parameter-group-identifier`** - 필수입니다. 복사하려는 클러스터 파라미터 그룹의 의 이름 또는 Amazon 리소스 이름(ARN)입니다.

  소스 및 대상 클러스터 파라미터 그룹이 동일한 경우 AWS 리전식별자는 이름 또는 ARN일 수 있습니다.

  소스 및 대상 클러스터 파라미터 그룹이 다른 경우 AWS 리전식별자는 ARN이어야 합니다.

- **`--target-db-cluster-parameter-group-identifier`** - 필수입니다. 클러스터 파라미터 그룹 복사본의 이름 또는 ARN입니다.

  제약 조건:

  - null이거나, 비워 두거나, 공백을 입력할 수 없습니다.
  - 1~255자의 문자, 숫자 또는 하이픈을 포함해야 합니다.
  - 첫 번째 문자는 글자이어야 합니다.
  - 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다.

- **`--target-db-cluster-parameter-group-description`** - 필수입니다. 사용자가 제공한 클러스터 파라미터 그룹 복사본에 대한 설명입니다.

Example

다음은 `sample-parameter-group`의 복사본을 만들어 이름을 `sample-parameter-group-copy`로 지정하는 코드입니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb copy-db-cluster-parameter-group \
    --source-db-cluster-parameter-group-identifier sample-parameter-group \
    --target-db-cluster-parameter-group-identifier sample-parameter-group-copy \
    --target-db-cluster-parameter-group-description "Copy of sample-parameter-group"
```

Windows의 경우:

```
aws docdb copy-db-cluster-parameter-group ^
    --source-db-cluster-parameter-group-identifier sample-parameter-group ^
    --target-db-cluster-parameter-group-identifier sample-parameter-group-copy ^
    --target-db-cluster-parameter-group-description "Copy of sample-parameter-group"
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:sample-parameter-group-copy",
        "DBClusterParameterGroupName": "sample-parameter-group-copy",
        "DBParameterGroupFamily": "docdb4.0",
        "Description": "Copy of sample-parameter-group"
    }
}
```

Amazon DocumentDB 클러스터 파라미터 그룹 재설정

AWS Management Console 또는 AWS Command Line Interface (AWS CLI)를 사용하여 클러스터 파라미터 그룹을 재설정하여 Amazon DocumentDB 클러스터 파라미터 그룹의 파라미터 값 중 일부 또는 전부를 기본값으로 재설정할 수 있습니다.

Using the AWS Management Console

클러스터 파라미터 그룹의 파라미터 값 중 일부 또는 전부를 기본값으로 재설정하려면 다음 단계를 수행하십시오.

클러스터 파라미터 그룹의 파라미터 값을 재설정하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 콘솔 왼쪽의 탐색 창에서 파라미터 그룹을 선택합니다.
3. 클러스터 파라미터 그룹 창에서 재설정하려는 클러스터 파라미터 그룹의 이름을 선택합니다.
4. 작업을 선택한 후 재설정을 선택하여 해당 파라미터 그룹을 재설정합니다.

5. 결과 클러스터 파라미터 그룹 재설정 확인 페이지에서 해당 파라미터 그룹에 대한 모든 클러스터 파라미터를 기본값으로 재설정할지 확인합니다. 그런 다음 재설정을 선택하여 파라미터 그룹을 재설정합니다. 취소를 선택하여 변경 사항을 취소할 수도 있습니다.

Using the AWS CLI

클러스터 파라미터 그룹의 파라미터 값 중 일부 또는 전부를 기본값으로 재설정하려면 다음 파라미터와 함께 `reset-db-cluster-parameter-group` 작업을 사용합니다.

- **`--db-cluster-parameter-group-name`** - 필수입니다. 재설정할 클러스터 파라미터 그룹의 이름입니다.
- **`--parameters`** - 선택 사항. 기본값으로 재설정하려는 클러스터 파라미터 그룹의 `ParameterName` 및 `ApplyMethod` 목록입니다. 다음 인스턴스 재시작 또는 `reboot-db-instance` 요청 시 적용하려면 정적 파라미터를 `pending-reboot`로 설정해야 합니다. 업데이트된 정적 파라미터를 적용할 클러스터의 모든 인스턴스에 대해 `reboot-db-instance`를 호출해야 합니다.

  이 파라미터와 `--reset-all-parameters`는 상호 배타적이므로 둘 중 하나를 선택할 수 있지만 둘 다 선택할 수는 없습니다.
- **`--reset-all-parameters`** 또는 **`--no-reset-all-parameters`** - 선택 사항. 모든 파라미터(`--reset-all-parameters`)를 기본값으로 재설정할지 또는 일부 파라미터(`--no-reset-all-parameters`)를 기본값으로 재설정할지를 지정합니다. `--reset-all-parameters` 파라미터와 `--parameters`는 상호 배타적이므로 둘 중 하나를 선택할 수 있지만 둘 다 선택할 수는 없습니다.

  전체 그룹을 재설정하면 동적 파라미터가 즉시 업데이트됩니다. 다음 인스턴스 재시작 시 또는 `reboot-db-instance` 요청 시 적용하려면 정적 파라미터를 `pending-reboot`로 설정합니다. 업데이트된 정적 파라미터를 적용할 클러스터의 모든 인스턴스에 대해 `reboot-db-instance`를 호출해야 합니다.

Example

예제 1: 모든 파라미터를 기본값으로 재설정합니다.

다음 코드는 클러스터 파라미터 그룹 `sample-parameter-group`의 모든 파라미터를 기본값으로 재설정합니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb reset-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group \
       --reset-all-parameters
```

Windows의 경우:

```
aws docdb reset-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group ^
       --reset-all-parameters
```

예제 2: 지정된 파라미터를 기본값으로 재설정합니다.

다음 코드는 클러스터 파라미터 그룹 `sample-parameter-group`의 `tls` 파라미터를 기본값으로 재설정합니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb reset-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group \
       --no-reset-all-parameters \
       --parameters ParameterName=tls,ApplyMethod=pending-reboot
```

Windows의 경우:

```
aws docdb reset-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group ^
       --no-reset-all-parameters ^
       --parameters ParameterName=tls,ApplyMethod=pending-reboot
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
       "DBClusterParameterGroupName": "sample-parameter-group"
   }
```

클러스터 인스턴스 재부팅

정적 파라미터 값을 변경하기 전에 클러스터 인스턴스를 재부팅해야 합니다. 업데이트된 정적 파라미터를 적용할 클러스터의 각 인스턴스를 재부팅합니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb reboot-db-instance \
       --db-instance-identifier sample-cluster-instance
```

Windows의 경우:

```
aws docdb reboot-db-instance ^
       --db-instance-identifier sample-cluster-instance
```

Amazon DocumentDB 클러스터 파라미터 그룹 삭제

AWS Management Console 또는 AWS Command Line Interface ()를 사용하여 사용자 지정 Amazon DocumentDB 클러스터 파라미터 그룹을 삭제할 수 있습니다AWS CLI. `default.docdb4.0` 클러스터 파라미터 그룹은 삭제할 수 없습니다.

Using the AWS Management Console

클러스터 파라미터 그룹을 삭제하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택하십시오.

3. 파라미터 그룹 창에서 삭제하려는 클러스터 파라미터 그룹 왼쪽에 있는 라디오 버튼을 선택합니다.
4. 작업을 선택한 후 삭제를 선택합니다.
5. 삭제 확인 창에서 삭제를 선택하여 클러스터 파라미터 그룹을 삭제합니다. 클러스터 파라미터 그룹을 유지하려면 취소를 선택합니다.

Using the AWS CLI

클러스터 파라미터 그룹을 삭제하려면 다음 파라미터와 함께 `delete-db-cluster-parameter-group` 작업을 사용합니다.

- **`--db-cluster-parameter-group-name`** - 필수입니다. 삭제할 클러스터 파라미터 그룹의 이름입니다. 기존 클러스터 파라미터 그룹이어야 합니다. *`default.docdb4.0`* 클러스터 파라미터 그룹은 삭제할 수 없습니다.

Example- 클러스터 파라미터 그룹 삭제

다음 예제에서는 클러스터 파라미터 그룹을 삭제하기 위한 3단계를 안내합니다.

1. 삭제하려는 클러스터 파라미터 그룹의 이름 찾기
2. 지정된 클러스터 파라미터 그룹 삭제
3. 클러스터 파라미터 그룹이 삭제되었는지 확인

1. 삭제하려는 클러스터 파라미터 그룹의 이름을 찾습니다.

다음 코드는 모든 클러스터 파라미터 그룹의 이름을 나열합니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb describe-db-cluster-parameter-groups \
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

Windows의 경우:

```
aws docdb describe-db-cluster-parameter-groups ^
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

이전 작업의 출력에는 다음과 유사한 클러스터 파라미터 그룹의 이름이 나열됩니다(JSON 형식).

```
[
       [
           "default.docdb4.0"
       ],
       [
           "sample-parameter-group"
       ],
```

```
        [
            "sample-parameter-group-copy"
        ]
    ]
```

2. 클러스터 파라미터 그룹을 삭제합니다.

다음 코드는 sample-parameter-group-copy 클러스터 파라미터 그룹을 삭제합니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb delete-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group-copy
```

Windows의 경우:

```
aws docdb delete-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group-copy
```

이 작업에는 출력이 없습니다.

3. 지정된 클러스터 파라미터 그룹이 삭제되었는지 확인합니다.

다음 코드는 이름이 변경된 모든 클러스터 파라미터 그룹의 이름을 나열합니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb describe-db-cluster-parameter-groups \
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

Windows의 경우:

```
aws docdb describe-db-cluster-parameter-groups ^
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

이전 작업의 출력에는 다음과 유사한 클러스터 파라미터 그룹이 나열됩니다(JSON 형식). 방금 삭제한 클러스터 파라미터 그룹이 목록에 없어야 합니다.

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
[
```

```
    [
        "default.docdb4.0"
    ],
    [
        "sample-parameter-group"
    ]
]
```

Amazon DocumentDB 클러스터 파라미터 참조

동적 파라미터를 변경하고 클러스터 파라미터 그룹을 저장하면 즉시 적용 설정과 관계없이 변경 사항이 바로 적용됩니다. 고정 파라미터를 변경하고 클러스터 파라미터 그룹을 저장하면 인스턴스를 수동으로 재부팅한 후에 파라미터 변경 사항이 적용됩니다. Amazon DocumentDB 콘솔을 사용하거나 명시적으로 `reboot-db-instance`를 호출하여 인스턴스를 재부팅할 수 있습니다.

다음 표에는 Amazon DocumentDB 클러스터의 모든 인스턴스에 적용되는 파라미터가 나와 있습니다.

Amazon DocumentDB 클러스터 수준 파라미터

파라미터	기본값	유효값	수정 가능	적용 유형	데이터 형식	설명
`audit_logs`	disabled	enabled, disabled, ddl, dml_read, dml_write, all, none	예	동적	문자열	Amazon CloudWatch 감사 로그 활성화 여부를 정의합니다. • **enabled** — CloudWatch 감사 로그가 활성화되었습니다. • **disabled** — CloudWatc

파라미터	기본값	유효값	수정 가능	적용 유형	데이터 형식	설명
						h 감사 로그가 비활성화되었습니다. • **ddl** — DDL 이벤트 감사가 활성화되었습니다. • **dml_read** — DML 읽기 이벤트 감사가 활성화되었습니다. • **dml_write** — DML 쓰기 이벤트 감사가 활성화되었습니다. • **all** — 모든 데이터베이스 이벤트에 대한 감사가 활성

파라미터	기본값	유효값	수정 가능	적용 유형	데이터 형식	설명
						화되었습니다. • **none** — 감사가 비활성화되었습니다.
change_stream_log_retention_duration	10800	3600-604800	예	동적	Integer	변경 스트림 로그가 보관되고 소비될 수 있는 시간(초)을 정의합니다.

파라미터	기본값	유효값	수정 가능	적용 유형	데이터 형식	설명
`default_collection_compression`	disabled	활성화됨, 비활성화됨 (Amazon DocumentDB 5.0)/zstd, lz4, 없음 (Amazon DocumentDB8.0)	예	동적	문자열	클러스터의 새 컬렉션에 대한 기본 압축 설정을 정의합니다. • **enabled** - Lz4 압축은 Amazon DocumentDB 5.0에서 기본적으로 활성화됩니다. zstd 압축은 Amazon DocumentDB 8.0에서 기본적으로 활성화됩니다. • **disabled, none** - 압축은 기본적으로 비활성화되어 있습니다 (Amazon

파라미터	기본값	유효값	수정 가능	적용 유형	데이터 형식	설명
						DocumentDB 5.0에서는 비활성화되고 Amazon DocumentDB 8.0에서는 비활성화되지 않음).

파라미터	기본값	유효값	수정 가능	적용 유형	데이터 형식	설명
`profiler`	disabled	enabled, disabled	예	동적	문자열	느린 작업에 대해 프로파일링을 활성화합니다. • **`enabled`** — 고객이 정의한 임계값(예: 100ms)보다 오래 걸리는 작업은 Amazon CloudWatch Logs에 기록됩니다. • **`disabled`** — 느린 작업은 CloudWatch Logs에 기록되지 않습니다.

파라미터	기본값	유효값	수정 가능	적용 유형	데이터 형식	설명
profiler_sampling_rate	1.0	0.0-1.0	예	동적	Float	로깅된 작업의 샘플링 비율을 정의합니다.
profiler_threshold_ms	100	50-2147483646	예	동적	Integer	profiler에 대한 임계값을 정의합니다. • profiler_threshold_ms 보다 큰 모든 작업은 CloudWatch Logs에 기록됩니다.

파라미터	기본값	유효값	수정 가능	적용 유형	데이터 형식	설명
planner_version	3.0	1.0, 2.0, 3.0	예	동적	Float	쿼리에 사용할 쿼리 플래너 버전을 정의합니다. • **1.0** - Amazon DocumentDB 5.0에서 지원됩니다. • **2.0** - 연산자를 찾고 업데이트하는 기능이 개선되었습니다. Amazon DocumentDB 5.0에서 지원됩니다. • **3.0** - 검색, 업데이트 및 집계 연산자를 개선했습니다. Amazon

파라미터	기본값	유효값	수정 가능	적용 유형	데이터 형식	설명
						DocumentDB 8.0의 기본 쿼리 플래너입니다.

파라미터	기본값	유효값	수정 가능	적용 유형	데이터 형식	설명
`tls`	enabled	enabled, disabled, fips-140-3, tls1.2+, tls1.3+	예	정적	문자열	TLS(전송 계층 보안) 연결이 필요한지 여부를 정의합니다. • **`enabled`** — 연결하려면 TLS 연결이 필요합니다. Amazon DocumentDB 5.0 이하의 경우 TLS 1.0~1.3을 활성화하지만 TLS 1.2 이상을 사용합니다. 마이너 버전 5.0.1부터 Amazon DocumentDB 5.0, Amazon DocumentDB 8.0 이

파라미터	기본값	유효값	수정 가능	적용 유형	데이터 형식	설명
						상에서는 TLS 1.2 이상만 지원됩니다. • **disabled** — 연결하는 데 TLS 연결을 사용할 수 없습니다. • **fips-140-3** — 연결하려면 FIPS(연방 정보 처리 표준) 특성을 사용한 TLS 연결이 필요합니다. FIPS 간행물 140-3에 따라 클러스터는 보안 연결만 허용합니다. 이는 ca-

파라미터	기본값	유효값	수정 가능	적용 유형	데이터 형식	설명
						central-1, us-west-2, us-east-1, us-east-2, us-gov-east-1, us-gov-west-1 리전의 Amazon DocumentDB 5.0(엔진 버전 3.0.3727) 클러스터부터만 지원됩니다. • **tls1.2+** — 연결하려면 TLS 버전 1.2 이상을 사용하는 TLS 연결이 필요합니다. Amazon DocumentDB 4.0(엔

파라미터	기본값	유효값	수정 가능	적용 유형	데이터 형식	설명
						진 버전 2.0.10980) 및 Amazon DocumentDB 5.0(엔진 버전 3.0.11051)부터만 지원됩니다. • **`tls1.3+`** — 연결하려면 TLS 버전 1.3 이상을 사용하는 TLS 연결이 필요합니다. Amazon DocumentDB 4.0(엔진 버전 2.0.10980) 및 Amazon DocumentDB 5.0(엔진 버전 3.0.11051)부터만

파라미터	기본값	유효값	수정 가능	적용 유형	데이터 형식	설명
						지원됩니다.
`ttl_monitor`	enabled	enabled, disabled	예	동적	문자열	Time To Live(TTL) 모니터링이 클러스터에 대해 활성화되었는지 여부를 정의합니다. • **`enabled`** — TTL 모니터링이 활성화되었습니다. • **`disabled`** — TTL 모니터링이 비활성화되었습니다.

Amazon DocumentDB 클러스터 파라미터 수정

Amazon DocumentDB에서 클러스터 파라미터 그룹은 클러스터에서 만드는 모든 인스턴스에 적용되는 파라미터로 구성됩니다. 사용자 지정 클러스터 파라미터 그룹의 경우 언제든지 파라미터 값을 수정하거나 모든 파라미터 값을 사용자가 생성하는 파라미터 그룹에 대한 기본값으로 재설정할 수 있습니다. 이 단원에서는 Amazon DocumentDB 클러스터 파라미터 그룹을 구성하는 파라미터 및 해당 값을 확인하는 방법과 이러한 값을 변경하거나 업데이트하는 방법에 대해 설명합니다.

파라미터는 동적이거나 정적일 수 있습니다. 동적 파라미터를 변경하고 클러스터 파라미터 그룹을 저장하면 `Apply Immediately` 설정과 관계없이 변경 사항이 바로 적용됩니다. 고정 파라미터를 변경하고 클러스터 파라미터 그룹을 저장하면 인스턴스를 수동으로 재부팅한 후에만 파라미터 변경 사항이 적용됩니다.

Amazon DocumentDB 클러스터 파라미터 그룹의 파라미터 보기

AWS Management Console 또는를 사용하여 Amazon DocumentDB 클러스터의 파라미터와 해당 값을 볼 수 있습니다 AWS CLI.

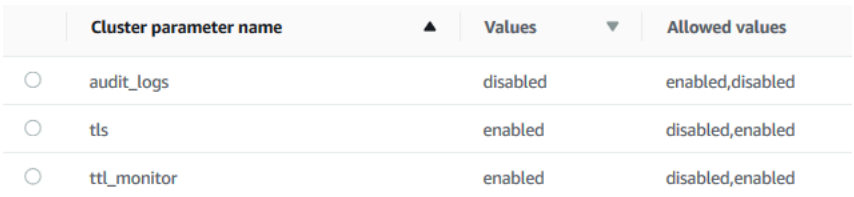
Using the AWS Management Console

클러스터 파라미터 그룹의 세부 정보를 보려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택하십시오.

3. 파라미터 그룹 창에서 세부 정보를 보려는 클러스터 파라미터 그룹의 이름을 선택합니다.
4. 결과 페이지에는 각 파라미터에 대한 값(파라미터 이름, 현재 값, 허용된 값, 파라미터 수정 가능 여부, 적용 유형, 데이터 형식 및 설명)이 표시됩니다.



Cluster parameter name	Values	Allowed values
audit_logs	disabled	enabled,disabled
tls	enabled	disabled,enabled
ttl_monitor	enabled	disabled,enabled

Using the AWS CLI

클러스터 파라미터 그룹의 파라미터 및 해당 값을 보려면 다음 파라미터와 함께 `describe-db-cluster-parameters` 작업을 사용합니다.

- **--db-cluster-parameter-group-name** - 필수입니다. 자세한 파라미터 목록을 원하는 클러스터 파라미터 그룹의 이름입니다.
- **--source** - 선택 사항. 제공된 경우 특정 소스에 대한 파라미터만 반환합니다. 파라미터 소스는 engine-default, system 또는 user가 될 수 있습니다.

Example

다음 코드에는 custom3-6-param-grp 파라미터 그룹의 모든 파라미터 및 해당 값이 나열되어 있습니다. 파라미터 그룹에 대한 세부 정보를 보려면 --query 줄을 생략합니다. 모든 파라미터 그룹에 대한 정보를 보려면 --db-cluster-parameter-group-name 줄을 생략합니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb describe-db-cluster-parameters \
   --db-cluster-parameter-group-name custom3-6-param-grp \
   --query 'Parameters[*].[ParameterName,ParameterValue]'
```

Windows의 경우:

```
aws docdb describe-db-cluster-parameters ^
   --db-cluster-parameter-group-name custom3-6-param-grp ^
   --query 'Parameters[*].[ParameterName,ParameterValue]'
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
[
    [
        "audit_logs",
        "disabled"
    ],
    [
        "tls",
        "enabled"
    ],
    [
        "ttl_monitor",
        "enabled"
    ]
]
```

Amazon DocumentDB 클러스터 파라미터 그룹의 파라미터 수정

AWS Management Console 또는를 사용하여 파라미터 그룹의 파라미터를 수정할 수 있습니다 AWS CLI.

Using the AWS Management Console

클러스터 파라미터 그룹의 파라미터를 업데이트하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택하십시오.

3. 파라미터 그룹 창에서 파라미터를 업데이트하려는 클러스터 파라미터 그룹을 선택합니다.
4. 결과 페이지에는 이 클러스터 파라미터 그룹에 대한 파라미터 및 해당 세부 정보가 표시됩니다. 업데이트할 파라미터를 선택합니다.
5. 페이지 오른쪽 상단에서 편집을 선택하여 파라미터의 값을 변경합니다. 클러스터 파라미터 유형에 대한 자세한 내용은 Amazon DocumentDB 클러스터 파라미터 참조(을)를 참조하세요.
6. 변경한 다음 Modify cluster parameter(클러스터 파라미터 수정)를 선택하여 변경 사항을 저장합니다. 변경 사항을 취소하려면 취소를 선택합니다.

Using the AWS CLI

클러스터 파라미터 그룹의 파라미터를 수정하려면 다음 파라미터와 함께 `modify-db-cluster-parameter-group` 작업을 사용합니다.

- **`--db-cluster-parameter-group-name`** - 필수입니다. 수정하려는 클러스터 파라미터 그룹의 이름입니다.
- **`--parameters`** - 필수입니다. 수정할 파라미터입니다. 각 파라미터 요소는 다음을 포함해야 합니다.
  - **`ParameterName`** - 수정 중인 파라미터의 이름입니다.

- **ParameterValue** - 이 파라미터의 새 값입니다.
- **ApplyMethod** - 파라미터에 대한 변경 사항을 적용할 방법입니다. 허용된 값은 `immediate` 및 `pending-reboot`입니다.

> **Note**
>
> `static`의 `ApplyType` 파라미터에는 `pending-reboot`의 `ApplyMethod`이 있어야 합니다.

클러스터 파라미터 그룹의 파라미터 값을 변경하려면(AWS CLI)

다음 예에서는 `tls` 파라미터를 변경합니다.

1. **sample-parameter-group**의 파라미터 및 해당 값을 나열합니다.

   Linux, macOS 또는 Unix의 경우:

   ```
   aws docdb describe-db-cluster-parameters \
       --db-cluster-parameter-group-name sample-parameter-group
   ```

   Windows의 경우:

   ```
   aws docdb describe-db-cluster-parameters ^
       --db-cluster-parameter-group-name sample-parameter-group
   ```

   이 작업의 출력은 다음과 같습니다(JSON 형식).

   ```
   {
       "Parameters": [
           {
               "Source": "system",
               "ApplyType": "static",
               "AllowedValues": "disabled,enabled",
               "ParameterValue": "enabled",
               "ApplyMethod": "pending-reboot",
               "DataType": "string",
               "ParameterName": "tls",
               "IsModifiable": true,
               "Description": "Config to enable/disable TLS"
   ```

```
        },
        {
            "Source": "user",
            "ApplyType": "dynamic",
            "AllowedValues": "disabled,enabled",
            "ParameterValue": "enabled",
            "ApplyMethod": "pending-reboot",
            "DataType": "string",
            "ParameterName": "ttl_monitor",
            "IsModifiable": true,
            "Description": "Enables TTL Monitoring"
        }
    ]
}
```

2. 값이 **disabled**이 되도록 **tls** 파라미터를 수정합니다. ApplyType은 static이므로 ApplyMethod를 수정할 수 없습니다.

   Linux, macOS 또는 Unix의 경우:

```
aws docdb modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name sample-parameter-group \
    --parameters
 "ParameterName"=tls,ParameterValue=disabled,ApplyMethod=pending-reboot"
```

   Windows의 경우:

```
aws docdb modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name sample-parameter-group ^
    --parameters "ParameterName=tls,ParameterValue=disabled,ApplyMethod=pending-
reboot"
```

   이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "DBClusterParameterGroupName": "sample-parameter-group"
}
```

3. 적어도 5분을 기다립니다.
4. **sample-parameter-group**의 파라미터 값을 나열합니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

Windows의 경우:

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name sample-parameter-group
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "Parameters": [
        {
            "ParameterName": "audit_logs",
            "ParameterValue": "disabled",
            "Description": "Enables auditing on cluster.",
            "Source": "system",
            "ApplyType": "dynamic",
            "DataType": "string",
            "AllowedValues": "enabled,disabled",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot"
        },
        {
            "ParameterName": "tls",
            "ParameterValue": "disabled",
            "Description": "Config to enable/disable TLS",
            "Source": "user",
            "ApplyType": "static",
            "DataType": "string",
            "AllowedValues": "disabled,enabled",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot"
        }
    ]
}
```

Amazon DocumentDB 엔드포인트에 대한 이해

Amazon DocumentDB(MongoDB와 호환됨) 엔드포인트를 사용하여 클러스터 또는 인스턴스에 연결할 수 있습니다. Amazon DocumentDB에는 각각 고유한 용도가 있는 세 가지 유형의 엔드포인트가 있습니다.

주제

- 클러스터 엔드포인트 찾기
- 인스턴스의 엔드포인트 찾기
- 엔드포인트에 연결

클러스터 엔드포인트

클러스터 엔드포인트는 클러스터의 현재 기본 인스턴스에 연결되는 Amazon DocumentDB 클러스터의 엔드포인트입니다. 각 Amazon DocumentDB 클러스터 엔드포인트 하나와 기본 인스턴스 하나가 있습니다. 장애 조치의 경우 클러스터 엔드포인트가 새로운 기본 인스턴스에 다시 매핑됩니다.

리더 엔드포인트

리더 엔드포인트는 해당 클러스터에서 사용할 수 있는 복제본 중 하나에 연결되는 Amazon DocumentDB 클러스터의 엔드포인트입니다. 각 Amazon DocumentDB 클러스터에는 리더 엔드포인트가 1개씩 있습니다. 복제본이 둘 이상이면 리더 엔드포인트는 각 연결 요청을 Amazon DocumentDB 복제본 중 하나로 전달합니다.

인스턴스 엔드포인트

인스턴스 엔드포인트는 특정 인스턴스에 연결되는 엔드포인트입니다. 기본 인스턴스인지 복제본 인스턴스인지 여부에 상관없이 클러스터의 각 인스턴스에는 고유한 인스턴스 엔드포인트가 있습니다. 애플리케이션에서 인스턴스 엔드포인트를 사용하지 않는 것이 좋습니다. 왜냐하면 장애 조치를 실행하는 경우 역할을 변경할 수 있으므로 애플리케이션에서 코드 변경을 요구할 수 있기 때문입니다.

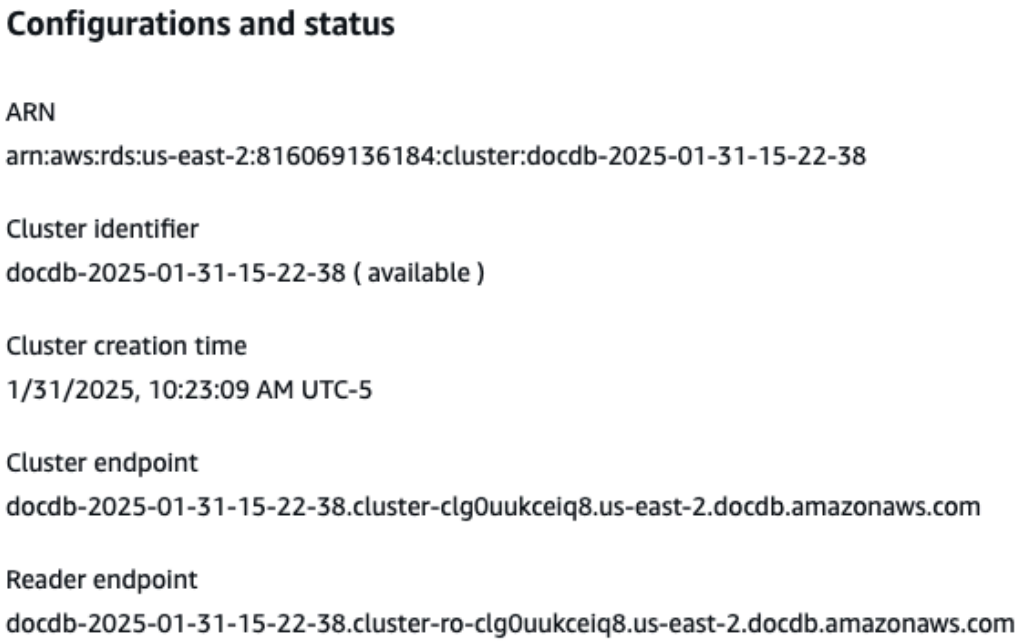
클러스터 엔드포인트 찾기

Amazon DocumentDB 콘솔 또는 AWS CLI를 사용하여 클러스터의 클러스터 엔드포인트와 리더 엔드포인트를 찾을 수 있습니다.

Using the AWS Management Console

콘솔을 사용하여 클러스터의 엔드포인트 찾기:

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터 목록에서 원하는 클러스터 이름을 선택합니다.
4. 클러스터 세부 정보 페이지에서 구성 탭을 선택합니다. 구성 및 상태 섹션에서 클러스터 엔드포인트와 리더 엔드포인트를 찾을 수 있습니다.



5. 이 클러스터에 연결하려면 연결 및 보안 탭을 선택합니다. mongo 쉘에 대한 연결 문자열과 애플리케이션 코드에서 클러스터에 연결하는 데 사용할 수 있는 연결 문자열을 찾습니다.

Using the AWS CLI

를 사용하여 클러스터의 클러스터 및 리더 엔드포인트를 찾으려면 다음 파라미터와 함께 `describe-db-clusters` 명령을 AWS CLI실행합니다.

파라미터

- **`--db-cluster-identifier`**—선택 사항. 엔드포인트를 반환하려는 클러스터를 지정합니다. 생략한 경우 최대 100개의 클러스터에 대한 엔드포인트를 반환합니다.
- **`--query`**—선택 사항. 표시할 필드를 지정합니다. 엔드포인트를 찾기 위해 볼 데이터의 양을 줄여서 도와줍니다. 생략하면 클러스터에 대한 모든 정보가 반환됩니다.
- **`--region`**—선택 사항. `--region` 파라미터를 사용하여 명령을 적용할 리전을 지정합니다. 생략하면 기본 리전이 사용됩니다.

Example

다음 예에서는 `sample-cluster`에 대한 `DBClusterIdentifier`, 엔드포인트(클러스터 엔드포인트) 및 `ReaderEndpoint`를 반환합니다.

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-clusters \
   --region us-east-1 \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[DBClusterIdentifier,Port,Endpoint,ReaderEndpoint]'
```

Windows의 경우:

```
aws docdb describe-db-clusters ^
   --region us-east-1 ^
   --db-cluster-identifier sample-cluster ^
   --query 'DBClusters[*].[DBClusterIdentifier,Port,Endpoint,ReaderEndpoint]'
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
[
  [
     "sample-cluster",
     27017,
```

```
    "sample-cluster.cluster-corlsfccjozr.us-east-1.docdb.amazonaws.com",
    "sample-cluster.cluster-ro-corlsfccjozr.us-east-1.docdb.amazonaws.com"
  ]
]
```

클러스터 엔드포인트가 있으므로 이제 mongo 또는 mongodb를 사용하여 클러스터에 연결할 수 있습니다. 자세한 내용은 엔드포인트에 연결 단원을 참조하십시오.

인스턴스의 엔드포인트 찾기

Amazon DocumentDB 콘솔 또는 AWS CLI을 사용하여 인스턴스의 엔드포인트를 찾을 수 있습니다.

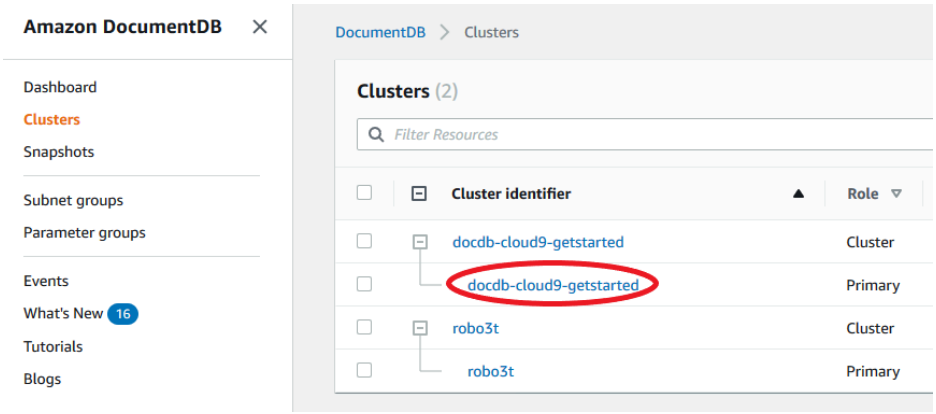
Using the AWS Management Console

콘솔을 사용하여 인스턴스의 엔드포인트를 찾으려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

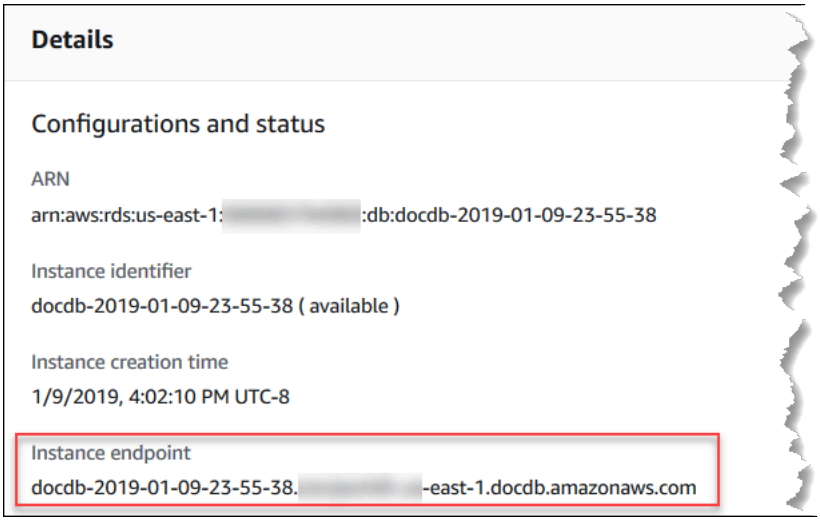
   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택합니다.

3. 클러스터 탐색 상자에 클러스터 식별자 열이 표시됩니다. 인스턴스는 다음 스크린샷과 마찬가지로 클러스터 아래에 나열됩니다.



4. 관심 있는 인스턴스의 왼쪽에 있는 확인란을 체크합니다.

5. 아래로 스크롤하여 세부 정보 섹션으로 이동한 다음 인스턴스 엔드포인트를 찾습니다.



6. 이 인스턴스에 연결하려면 위로 스크롤하여 연결 섹션으로 이동합니다. mongo 쉘에 대한 연결 문자열과 애플리케이션 코드에서 인스턴스에 연결하는 데 사용할 수 있는 연결 문자열을 찾습니다.

Using the AWS CLI

를 사용하여 인스턴스 엔드포인트를 찾으려면 다음 인수와 함께 다음 명령을 AWS CLI실행합니다.

인수

- **`--db-instance-identifier`**—선택 사항. 엔드포인트를 반환하려는 인스턴스를 지정합니다. 생략한 경우 최대 100개의 인스턴스에 대한 엔드포인트를 반환합니다.
- **`--query`**—선택 사항. 표시할 필드를 지정합니다. 엔드포인트를 찾기 위해 볼 데이터의 양을 줄여서 도와줍니다. 생략하면 인스턴스에 대한 모든 정보가 반환됩니다. `Endpoint` 필드에는 세 개의 멤버가 있습니다. 따라서 다음 예제와 같이 쿼리에 나열하면 세 멤버가 모두 반환됩니다. `Endpoint` 멤버 중 일부에만 관심이 있는 경우 두 번째 예제와 같이 쿼리의 `Endpoint`를 원하는 멤버로 교체하세요.
- **`--region`**—선택 사항. `--region` 파라미터를 사용하여 명령을 적용할 리전을 지정합니다. 생략하면 기본 리전이 사용됩니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-instances \
    --region us-east-1 \
    --db-instance-identifier sample-cluster-instance \
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint]'
```

Windows의 경우:

```
aws docdb describe-db-instances ^
    --region us-east-1 ^
    --db-instance-identifier sample-cluster-instance ^
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint]'
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
[
    [
        "sample-cluster-instance",
        {
            "Port": 27017,
            "Address": "sample-cluster-instance.corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
            "HostedZoneId": "Z2R2ITUGPM61AM"
        }
    ]
]
```

엔드포인트의 `HostedZoneId`를 제거하기 위해 출력을 줄이고 `Endpoint.Port` 및 `Endpoint.Address`를 지정하여 쿼리를 수정할 수 있습니다.

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-instances \
    --region us-east-1 \
    --db-instance-identifier sample-cluster-instance \
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint.Port,Endpoint.Address]'
```

Windows의 경우:

```
aws docdb describe-db-instances ^
    --region us-east-1 ^
    --db-instance-identifier sample-cluster-instance ^
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint.Port,Endpoint.Address]'
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
[
    [
        "sample-cluster-instance",
        27017,
        "sample-cluster-instance.corcjozrlsfc.us-east-1.docdb.amazonaws.com"
    ]
]
```

인스턴스 엔드포인트가 있으므로 이제 mongo 또는 mongodb를 사용하여 인스턴스에 연결할 수 있습니다. 자세한 내용은 엔드포인트에 연결 단원을 참조하십시오.

엔드포인트에 연결

엔드포인트(클러스터 또는 인스턴스)가 있는 경우 mongo 쉘 또는 연결 문자열을 사용하여 엔드포인트에 연결할 수 있습니다.

mongo 쉘을 사용하여 연결

다음 구조를 사용하면 mongo 쉘을 사용하여 클러스터 또는 인스턴스에 연결하는 데 필요한 문자열을 작성할 수 있습니다.

```
mongo \
    --ssl \
    --host Endpoint:Port \
    --sslCAFile global-bundle.pem \
    --username UserName \
    --password Password
```

mongo 쉘 예

클러스터에 연결:

```
mongo \
```

```
    --ssl \
    --host sample-cluster.corcjozrlsfc.us-east-1.docdb.amazonaws.com:27017 \
    --sslCAFile global-bundle.pem \
    --username UserName \
    --password Password
```

인스턴스에 연결:

```
mongo \
    --ssl \
    --host sample-cluster-instance.corcjozrlsfc.us-east-1.docdb.amazonaws.com:27017 \
    --sslCAFile global-bundle.pem \
    --username UserName \
    --password Password
```

연결 문자열을 사용하여 연결

다음 구조를 사용하면 클러스터 또는 인스턴스에 연결하는 데 필요한 연결 문자열을 작성할 수 있습니다.

```
mongodb://UserName:Password@endpoint:port?replicaSet=rs0&ssl_ca_certs=global-
bundle.pem
```

연결 문자열 예

클러스터에 연결:

```
mongodb://UserName:Password@sample-cluster.cluster-corlsfccjozr.us-
east-1.docdb.amazonaws.com:27017?replicaSet=rs0&ssl_ca_certs=global-bundle.pem
```

인스턴스에 연결:

```
mongodb://UserName:Password@sample-cluster-instance.cluster-corlsfccjozr.us-
east-1.docdb.amazonaws.com:27017?replicaSet=rs0&ssl_ca_certs=global-bundle.pem
```

Amazon DocumentDB Amazon 리소스 이름(ARN) 이해

에서 생성하는 리소스 AWS 는 각각 Amazon 리소스 이름(ARN)으로 고유하게 식별됩니다. 특정 Amazon DocumentDB(MongoDB 호환) 작업의 경우 ARN을 지정하여 Amazon DocumentDB 리소스를

고유하게 식별해야 합니다. 예를 들어, 리소스에 태그를 추가할 경우 리소스의 ARN을 제공해야 합니다.

주제

- Amazon DocumentDB 리소스에 대한 ARN 구성
- Amazon DocumentDB 리소스 ARN 찾기

Amazon DocumentDB 리소스에 대한 ARN 구성

다음 구문을 사용하여 Amazon DocumentDB 리소스에 대한 ARN을 구성할 수 있습니다. Amazon DocumentDB는 Amazon Relational Database Service(Amazon RDS) ARN의 형식을 공유합니다. Amazon DocumentDB ARN에는 `rds`가 포함되지만 `docdb`는 포함되지 않습니다.

`arn:aws:rds:`*`region`*`:`*`account_number`*`:`*`resource_type`*`:`*`resource_id`*

리전 이름	리전	가용 영역(컴퓨팅)
미국 동부(오하이오)	us-east-2	3
미국 동부(버지니아 북부)	us-east-1	6
미국 서부(오리건)	us-west-2	4
아프리카(케이프타운)	af-south-1	3
남아메리카(상파울루)	sa-east-1	3
아시아 태평양(홍콩)	ap-east-1	3
아시아 태평양(하이데라바드)	ap-south-2	3
아시아 태평양(말레이시아)	ap-southeast-5	3
아시아 태평양(뭄바이)	ap-south-1	3
아시아 태평양(오사카)	ap-northeast-3	3

리전 이름	리전	가용 영역(컴퓨팅)
아시아 태평양(서울)	ap-northeast-2	4
아시아 태평양(싱가포르)	ap-southeast-1	3
아시아 태평양(시드니)	ap-southeast-2	3
아시아 태평양(자카르타)	ap-southeast-3	3
아시아 태평양(멜버른)	ap-southeast-4	3
아시아 태평양(태국)	ap-southeast-7	3
아시아 태평양(도쿄)	ap-northeast-1	3
캐나다(중부)	ca-central-1	3
캐나다 서부(캘거리)	ca-west-1	3
중국(베이징) 리전	cn-north-1	3
중국(닝샤)	cn-northwest-1	3
유럽(프랑크푸르트)	eu-central-1	3
유럽(취리히)	eu-central-2	3
유럽(아일랜드)	eu-west-1	3
유럽(런던)	eu-west-2	3
유럽(밀라노)	eu-south-1	3
유럽(파리)	eu-west-3	3
유럽(스페인)	eu-south-2	3
유럽(스톡홀름)	eu-north-1	3

리전 이름	리전	가용 영역(컴퓨팅)
멕시코(중부)	mx-central-1	3
중동(UAE)	me-central-1	3
이스라엘(텔아비브)	il-central-1	3
AWS GovCloud(미국 서부)	us-gov-west-1	3
AWS GovCloud(미국 동부)	us-gov-east-1	3

> **Note**
>
> Amazon DocumentDB 아키텍처는 스토리지와 컴퓨팅을 분리합니다. 스토리지 계층의 경우 Amazon DocumentDB는 3개의 AWS 가용 영역(AZs. 위 표에 나열된 AZs는 지정된 리전에서 컴퓨팅 인스턴스를 프로비저닝하는 데 사용할 수 있는 AZs 수입니다. 예를 들어 ap-northeast-1에서 Amazon DocumentDB 클러스터를 시작하는 경우 스토리지는 3개의 AZ에 걸쳐 6가지 방식으로 복제되지만 컴퓨팅 인스턴스는 2개의 AZ에서만 사용할 수 있습니다.

다음 표는 특정 Amazon DocumentDB 리소스에 대한 ARN을 구성할 때 사용해야 하는 형식을 보여줍니다. Amazon DocumentDB는 Amazon RDS ARN의 형식을 공유합니다. Amazon DocumentDB ARN에는 `rds`가 포함되지만 `docdb`는 포함되지 않습니다.

리소스 유형	ARN 형식/예
인스턴스(db)	`arn:aws:rds:` *`region`*`:`*`account_number`* `:db:`*`resource_id`* `arn:aws:rds:us-east-1:` *`1234567890`* `:db:`*`sample-db-instance`*
클러스터(cluster)	`arn:aws:rds:` *`region`*`:`*`account_number`* `:cluster:` *`resource_id`*

리소스 유형	ARN 형식/예
	`arn:aws:rds:us-east-1:` *1234567890* `:cluster:` *sample-db-cluster*
클러스터 파라미터 그룹 (cluster-pg)	`arn:aws:rds:` *region*`:`*account_number* `:cluster-pg:` *resource_id* `arn:aws:rds:us-east-1:` *1234567890* `:cluster-pg:` *sample-db-cluster-parameter-group*
보안 그룹(secgrp)	`arn:aws:rds:` *region*`:`*account_number* `:secgrp:`*resource_id* `arn:aws:rds:us-east-1:` *1234567890* `:secgrp:`*sample-public-secgrp*
클러스터 스냅샷(cluster-snapshot)	`arn:aws:rds:` *region*`:`*account_number* `:cluster-snapshot:` *resource_id* `arn:aws:rds:us-east-1:` *1234567890* `:cluster-snapshot:` *sample-db-cluster-snapshot*
서브넷 그룹(subgrp)	`arn:aws:rds:` *region*`:`*account_number* `:subgrp:`*resource_id* `arn:aws:rds:us-east-1:` *1234567890* `:subgrp:`*sample-subnet-10*

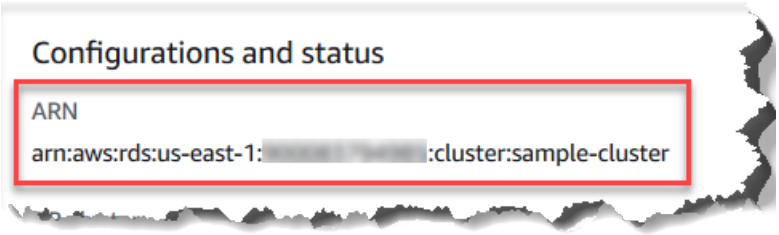
Amazon DocumentDB 리소스 ARN 찾기

AWS Management Console 또는를 사용하여 Amazon DocumentDB 리소스의 ARN을 찾을 수 있습니다 AWS CLI.

Using the AWS Management Console

콘솔을 사용해 ARN을 찾으려면 ARN을 보려는 리소스를 탐색하여 해당 리소스의 세부 정보를 확인합니다.

예를 들어, 클러스터 세부 정보 페이지의 구성 탭을 선택하여 클러스터의 ARN을 가져올 수 있습니다. 다음 스크린샷과 같이 구성 및 상태 섹션에서 ARN을 찾을 수 있습니다.



Using the AWS CLI

특정 Amazon DocumentDB 리소스에 AWS CLI 대해를 사용하여 ARN을 가져오려면 해당 리소스에 대한 `describe` 작업을 사용합니다. 다음 표에는 각 AWS CLI 작업과 작업과 함께 사용하여 ARN을 가져오는 ARN 속성이 나와 있습니다.

AWS CLI 명령	ARN 속성
`describe-db-instances`	`DBInstanceArn`
`describe-db-clusters`	`DBClusterArn`
`describe-db-parameter-groups`	`DBParameterGroupArn`
`describe-db-cluster-parameter-groups`	`DBClusterParameterGroupArn`
`describe-db-security-groups`	`DBSecurityGroupArn`
`describe-db-snapshots`	`DBSnapshotArn`
`describe-db-cluster-snapshots`	`DBClusterSnapshotArn`
`describe-db-subnet-groups`	`DBSubnetGroupArn`

Example- 클러스터의 ARN 찾기

다음 AWS CLI 작업은 클러스터의 ARN을 찾습니다`sample-cluster`.

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterArn'
```

Windows의 경우:

```
aws docdb describe-db-clusters ^
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterArn'
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
[
    "arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster"
]
```

Example- 여러 파라미터 그룹의 ARN 찾기

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-cluster-parameter-groups \
   --query 'DBClusterParameterGroups[*].DBClusterParameterGroupArn'
```

Windows의 경우:

```
aws docdb describe-db-cluster-parameter-groups ^
   --query 'DBClusterParameterGroups[*].DBClusterParameterGroupArn'
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
[
    "arn:aws:rds:us-east-1:123456789012:cluster-pg:custom3-6-param-grp",
    "arn:aws:rds:us-east-1:123456789012:cluster-pg:default.aurora5.6",
    "arn:aws:rds:us-east-1:123456789012:cluster-pg:default.docdb3.6"
```

```
]
```

Amazon DocumentDB 리소스 태그 지정

Amazon DocumentDB(MongoDB 호환) 태그를 사용하여 Amazon DocumentDB 리소스에 메타데이터를 추가할 수 있습니다. 이러한 태그는 AWS Identity and Access Management (IAM) 정책과 함께 사용하여 Amazon DocumentDB 리소스에 대한 액세스를 관리하고 리소스에 적용할 수 있는 작업을 제어할 수 있습니다. 또한 비슷하게 태그가 지정된 리소스에 대한 비용을 그룹화하여 이러한 태그로 비용을 추적할 수 있습니다.

다음 Amazon DocumentDB 리소스에 태그를 지정할 수 있습니다.

- 클러스터
- 인스턴스
- 스냅샷
- 클러스터 스냅샷
- 파라미터 그룹
- 클러스터 파라미터 그룹
- 보안 그룹
- 서브넷 그룹

Amazon DocumentDB 리소스 태그 개요

Amazon DocumentDB 태그는 Amazon DocumentDB 리소스를 정의하고 연결하는 이름-값 페어입니다. 이 이름을 키라고 합니다. 키 값을 제공하는 것은 선택 사항입니다. 태그를 사용하여 임의의 정보를 Amazon DocumentDB 리소스에 할당할 수 있습니다. 범주 정의 등에 태그 키를 사용할 수 있으며 태그 값은 해당 범주의 항목일 수 있습니다. 예를 들어, 태그 키를 `project`로 정의하고 태그 값을 `Salix`로 정의할 수 있는데, 이는 Amazon DocumentDB 리소스가 Salix 프로젝트에 할당되었음을 나타냅니다. 태그를 사용하여 `environment=test` 또는 `environment=production`와 같은 키를 사용해 Amazon DocumentDB 리소스를 테스트나 생산에 사용되도록 지정할 수도 있습니다. 일관된 태그 키 세트를 사용하여 Amazon DocumentDB 리소스와 연관된 메타데이터를 보다 쉽게 추적하는 것이 좋습니다.

태그를 사용하여 자체 비용 구조를 반영하도록 AWS 청구서를 구성할 수 있습니다. 이렇게 하려면 가입하여 태그 키 값이 포함된 AWS 계정 청구서를 가져옵니다. 그런 다음 같은 태그 키 값을 가진 리소스

에 따라 결제 정보를 구성하여 리소스 비용의 합을 볼 수 있습니다. 예를 들어, 특정 애플리케이션 이름으로 여러 리소스에 태그를 지정한 다음 결제 정보를 구성하여 여러 서비스에 걸친 해당 애플리케이션의 총 비용을 볼 수 있습니다. 자세한 내용은 AWS 청구 및 비용 관리 사용 설명서의 비용 할당 태그 사용을 참조하십시오.

각 Amazon DocumentDB 리소스에는 해당 리소스에 할당된 모든 태그가 포함된 태그 세트가 있습니다. 태그 세트는 최대 10개의 태그를 포함하거나 비어 있을 수 있습니다. 리소스의 기존 태그와 동일한 키를 가진 Amazon DocumentDB 리소스에 태그를 추가하면 새 값이 이전 값을 덮어씁니다.

AWS 는 태그에 의미론적 의미를 적용하지 않습니다. 태그는 엄격하게 문자열로 해석됩니다. Amazon DocumentDB는 리소스를 생성할 때 사용하는 설정에 따라 인스턴스 또는 다른 Amazon DocumentDB 리소스에 태그를 설정할 수 있습니다. 예를 들어, Amazon DocumentDB는 인스턴스가 생산용 또는 테스트용임을 나타내는 태그를 추가할 수 있습니다.

스냅샷에 태그를 추가할 수 있지만 이 그룹화는 청구서에 반영되지 않습니다.

AWS Management Console 또는를 사용하여 Amazon DocumentDB 리소스에 태그를 AWS CLI 추가, 나열 및 삭제할 수 있습니다. 를 사용할 때는 작업하려는 리소스의 Amazon 리소스 이름(ARN)을 제공해야 AWS CLI합니다. Amazon DocumentDB ARN에 대한 자세한 내용은 Amazon DocumentDB Amazon 리소스 이름(ARN) 이해 섹션을 참조하십시오.

태그 제약 조건

Amazon DocumentDB 태그에는 다음 제약 조건이 적용됩니다.

- 리소스당 최대 태그 수 - 10개
- 최대 키 길이 - 유니코드 128자
- 최대 값 길이 - 유니코드 256자
- 키 및 값에 유효한 문자 - UTF-8 문자 세트의 대문자와 소문자, 숫자, 공백 및 `_ . : / = + -`와 `@` 문자(Java regex: `"^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$"`)
- 태그 키와 값은 대소문자를 구분합니다.
- 태그 키 또는 값에 `aws:` 접두사는 사용할 수 없습니다. 이 접두사는 AWS전용입니다.

Amazon DocumentDB 리소스의 태그 추가 및 업데이트

AWS Management Console 또는를 사용하여 리소스에 최대 10개의 태그를 추가할 수 있습니다 AWS CLI.

Using the AWS Management Console

리소스에 태그를 추가하는 프로세스는 태그를 추가하는 리소스에 관계없이 비슷합니다. 이 예에서는 클러스터에 태그를 추가합니다.

콘솔을 사용하여 클러스터에 태그를 추가하거나 업데이트하려면 다음을 수행합니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 태그를 추가할 클러스터 이름을 선택합니다.
4. 태그 섹션으로 스크롤을 내린 다음 편집을 선택합니다.
5. 이 리소스에 추가할 각 태그에 대해 다음을 수행합니다.
   a. 새 태그를 추가하려면 키 상자의 태그 이름에 입력합니다. 태그 값을 변경하려면 키 열에서 태그 이름을 찾습니다.
   b. 태그에 새 값 또는 업데이트된 값을 제공하려면 값 상자에 태그에 대한 값을 입력합니다.
   c. 태그를 더 추가하려는 경우 추가를 선택합니다. 추가하지 않고 완료하려면 저장을 선택합니다.

Using the AWS CLI

리소스에 태그를 추가하는 프로세스는 태그를 추가하는 리소스에 관계없이 비슷합니다. 이 예에서는 클러스터에 세 개의 태그를 추가합니다. 두 번째 태그인 `key2`에는 값이 없습니다.

이러한 파라미터와 `add-tags-to-resource` 함께 AWS CLI 작업을 사용합니다.

파라미터

- **`--resource-name`** - 태그를 추가할 Amazon DocumentDB 리소스의 ARN입니다.
- **`--tags`** - 이 리소스에 추가할 태그(키-값 페어)를 `Key=`*`key-name`*`,Value=`*`tag-value`* 형식으로 나열합니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb add-tags-to-resource \
```

```
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster \
    --tags Key=key1,Value=value1 Key=key2 Key=key3,Value=value3
```

Windows의 경우:

```
aws docdb add-tags-to-resource ^
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster \
    --tags Key=key1,Value=value1 Key=key2 Key=key3,Value=value3
```

`add-tags-to-resource` 작업은 출력을 생성하지 않습니다. 작업 결과를 보려면 `list-tags-for-resource` 작업을 사용합니다.

Amazon DocumentDB 리소스에 태그 나열

AWS Management Console 또는를 사용하여 Amazon DocumentDB 리소스에 대한 태그 목록을 AWS CLI 가져올 수 있습니다.

Using the AWS Management Console

리소스에서 태그를 나열하는 프로세스는 태그를 추가하는 리소스에 관계없이 비슷합니다. 이 예에서는 클러스터에 태그를 나열합니다.

콘솔을 사용하여 클러스터에서 태그를 나열하려면

1. https://console.aws.amazon.com/docdb에서 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 태그를 나열할 클러스터 이름을 선택합니다.
4. 이 리소스의 태그 목록을 보려면 아래로 스크롤하여 태그 섹션으로 이동합니다.

Using the AWS CLI

리소스에서 태그를 나열하는 프로세스는 태그를 나열하는 리소스에 관계없이 비슷합니다. 이 예에서는 클러스터에 태그를 나열합니다.

이러한 파라미터와 `list-tags-for-resource` 함께 AWS CLI 작업을 사용합니다.

파라미터

- **`--resource-name`** - 필수입니다. 태그를 나열할 Amazon DocumentDB 리소스의 ARN입니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb list-tags-for-resource \
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster
```

Windows의 경우:

```
aws docdb list-tags-for-resource ^
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "TagList": [
        {
            "Key": "key1",
            "Value": "value1"
        },
        {
            "Key": "key2",
            "Value": ""
        },
        {
            "Key": "key3",
            "Value": "value3"
        }
    ]
}
```

Amazon DocumentDB 리소스에서 태그 제거

AWS Management Console 또는를 사용하여 Amazon DocumentDB 리소스에서 태그를 AWS CLI 제거할 수 있습니다.

Using the AWS Management Console

리소스에서 태그를 제거하는 프로세스는 태그를 추가하는 리소스에 관계없이 비슷합니다. 이 예에서는 클러스터에서 태그를 제거합니다.

콘솔을 사용하여 클러스터에서 태그를 제거하려면

1. https://console.aws.amazon.com/docdb에서 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 태그를 제거할 클러스터 이름을 선택합니다.
4. 태그 섹션으로 스크롤을 내린 다음 편집을 선택합니다.
5. 이 리소스에서 모든 태그를 제거하려면 모두 제거를 선택합니다. 그렇지 않으면, 이 리소스에서 제거할 각 태그에 대해 다음을 수행합니다.
   a. 키 열에서 태그 이름을 찾습니다.
   b. 태그 키와 동일한 행에서 제거를 선택합니다.
   c. 완료하였으면 저장을 선택합니다.

Using the AWS CLI

리소스에서 태그를 제거하는 프로세스는 태그를 제거하는 리소스에 관계없이 비슷합니다. 이 예에서는 클러스터에서 태그를 제거합니다.

이러한 파라미터와 `remove-tags-from-resource` 함께 AWS CLI 작업을 사용합니다.

- **`--resource-name`** - 필수입니다. 태그를 제거할 Amazon DocumentDB 리소스의 ARN입니다.
- **`--tag-keys`** - 필수입니다. 이 리소스에서 제거할 태그 키의 목록입니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb remove-tags-from-resource \
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster \
    --tag-keys key1 key3
```

Windows의 경우:

```
aws docdb remove-tags-from-resource ^
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster \
    --tag-keys key1 key3
```

`removed-tags-from-resource` 작업은 출력을 생성하지 않습니다. 작업 결과를 보려면 `list-tags-for-resource` 작업을 사용합니다.

Amazon DocumentDB 유지 관리

Amazon DocumentDB는 다음 두 가지 종류의 유지 관리를 주기적으로 수행합니다.

- 클러스터 유지 관리는 데이터베이스 엔진을 업데이트합니다. 엔진 업데이트에는 보안 수정, 버그 수정, 새로운 기능 및 기타 엔진 개선 사항이 포함됩니다.
- 인스턴스 유지 관리는 인스턴스의 운영 체제(OS)를 업데이트합니다.

엔진 패치 및 OS 업데이트는 선택 사항, 필수 및 강제라는 동일한 세 가지 수명 주기 범주를 사용하며, 각 범주에 동일한 알림 및 적용 동작을 사용합니다. 엔진 릴리스에는 수동으로 업그레이드하는 마이너 버전이라는 네 번째 범주도 있습니다. 범주는 다음과 같습니다.

- 선택 사항 - 중요하지 않은 개선 사항이 포함되어 있습니다. 자동 적용 날짜 및 AHD 알림이 없습니다. 사용자에게 적합할 때 적용됩니다. (OS 업데이트의 경우를 구독`RDS-EVENT-0230`하여 사용할 수 있게 되면 알림을 받을 수 있습니다.)
- 필수 - 보안 및 기타 중요한 수정 사항이 포함되어 있습니다. Health Dashboard (AHD) 및 이메일을 통해 알림을 받습니다. 필요한 작업은 클러스터 또는 인스턴스 유지 관리 기간 동안 이후에 자동으로 적용됩니다`AutoAppliedAfterDate`. 해당 날짜 이전에 유지 관리 기간을 변경하여 연기할 수 있습니다.
- 강제 - 드물지만 매우 중요한 수정입니다. 유지 관리 기간 이후에 자동으로 적용됩니다`ForcedApplyDate`. Amazon DocumentDB는 다른 옵션을 사용할 수 없는 경우에만 강제 작업을 지정합니다.
- 마이너 버전(엔진 릴리스만 해당) - 메이저 버전(예:)을 기반으로 번호가 매겨진 엔진 릴리스입니다`5.0.1`. 사용자 기반: 클러스터의 엔진 버전을 수정하여 업그레이드합니다. 자동 적용 안 함, AHD 알림 안 함. 마이너 버전은 5.0 이전의 메이저 버전에는 게시되지 않습니다.

엔진 패치는 단일 범주(선택 사항, 필수 또는 강제)로 릴리스되어 그대로 유지됩니다. OS 업데이트 진행 상황: 대부분은 선택 사항으로 시작되며, 적용되지 않으면 필수로 전환되고 결국 강제로 전환됩니다. 정확한 타이밍은 패치에 따라 달라지며 AHD 알림 및에서 반환한 날짜 필드에 게시됩니다`describe-pending-maintenance-actions`(참조적용 날짜). Amazon DocumentDB 릴리스 정보는 엔진 변경을 발표할 때 이러한 범주 이름을 사용합니다.

엔진 패치를 적용하면 클러스터가 잠시 오프라인 상태가 됩니다. 이 주제의 나머지 부분에서는 유지 관리 기간이 작동하는 방식, 보류 중인 작업을 찾는 방법, 엔진 패치 및 마이너 버전을 적용하는 방법, OS 업데이트 작동 방식, 글로벌 클러스터에 대한 특별 처리 방법을 안내합니다.

주제

- Amazon DocumentDB에 대한 유지 관리 작업
- 엔진 버전 번호 지정
- Amazon DocumentDB 유지 관리 기간 관리
- Amazon DocumentDB 엔진 패치에 대한 알림
- 보류 중인 Amazon DocumentDB 유지 관리 작업 보기
- Amazon DocumentDB 엔진 업데이트
- 마이너 버전 업그레이드
- Amazon DocumentDB 운영 체제 업데이트
- 사용자 시작 업데이트
- 글로벌 클러스터 패치 적용

Amazon DocumentDB에 대한 유지 관리 작업

Amazon DocumentDB 클러스터에는 다음과 같은 유지 관리 작업이 적용됩니다.

- `system-update` - Amazon DocumentDB 클러스터의 엔진 패치를 업그레이드합니다. 자세한 내용은 Amazon DocumentDB 엔진 업데이트 섹션을 참조하세요.
- `os-upgrade` - 롤링 업그레이드를 사용하여 Amazon DocumentDB 클러스터에 있는 모든 DB 인스턴스의 운영 체제를 업데이트합니다. 자세한 내용은 Amazon DocumentDB 운영 체제 업데이트 섹션을 참조하세요.

Amazon DocumentDB 인스턴스에는 다음과 같은 유지 관리 작업이 적용됩니다.

- `system-update` - Amazon DocumentDB 인스턴스의 운영 체제를 업그레이드합니다. 대신 클러스터 수준 `os-upgrade` 유지 관리 작업을 사용하는 것이 좋습니다. 자세한 내용은 Amazon DocumentDB 운영 체제 업데이트 섹션을 참조하세요.

엔진 버전 번호 지정

Amazon DocumentDB는 두 개의 개별 버전 식별자를 사용합니다.

- 엔진 버전 - 양식의 세 부분으로 구성된 번호*major*.*major*.*minor*입니다(예: 5.0.0 또는 5.0.1). 처음 두 부분(5.0)은 MongoDB 호환성 버전이고, 세 번째 부분은 마이너 버전이며, Amazon DocumentDB가 버그 수정 및 중단되지 않는 개선 사항이 포함된 마이너 릴리스를 게시할 때 증가합니다. 클러스터를 생성하거나 업그레이드할 때 지정하는 버전입니다.
- 엔진 패치 버전 - 클러스터에 적용된 패치 수준을 식별하는 양식의 별도의 세 부분으로 구성된 번호*major*.0.*patch*(예: 3.0.17983)입니다. 중간 숫자는 항상 입니다0. 패치 버전에는 중요한 보안 및 안정성 수정 사항이 포함되어 있습니다.

엔진 패치 버전이 자체적으로 나타나면 다음 표와 같이 접두사에서 엔진 버전을 확인할 수 있습니다.

엔진 패치 버전 접두사	Amazon DocumentDB 엔진 버전
1.0.*x*	3.6
2.0.*x*	4.0
3.0.*x*	5.0
4.0.*x*	8.0

클러스터가 실행 중인 패치 버전을 확인하려면를 연결하고 실행합니다db.runCommand({getEngineVersion: 1}).

> **Note**
>
> Amazon DocumentDB 5.0.2 및 8.0.2부터는 엔진 버전과 엔진 패치 버전을 형식의 단일 문자열로 db.runCommand({getEngineVersion: 1}) 반환합니다*engine-version*+*engine-patch-version*. 예를 들어 Amazon DocumentDB 5.0.2의 클러스터는를 반환하고 5.0.2+3.0.123458.0.2의 클러스터는를 반환합니다8.0.2+4.0.12345. 이전 버전의 클러스터는와 같이 엔진 패치 버전을 자체적으로 반환합니다3.0.12345.
> 의 출력을 구문 분석하는 스크립트 또는 모니터링이 있는 경우 결합된 형식을 처리하도록 getEngineVersion업데이트합니다.

릴리스된 엔진 패치 버전 목록과 각 버전에 포함된 내용은 섹션을 참조하세요릴리스 노트.

Amazon DocumentDB 유지 관리 기간 관리

모든 클러스터와 모든 인스턴스에는 예약된 수정 및 소프트웨어 패치 적용이 실행되는 주별 30분의 유지 관리 기간이 있습니다. 대부분의 이벤트는 30분 이내에 완료되며, 더 큰 이벤트는 더 오래 실행될 수 있습니다.

리소스를 생성할 때 기간을 선택하지 않으면 Amazon DocumentDB는 임의로 선택한 날짜에 리전에 정의된 8시간 일일 블록 내에서 무작위로 하나를 할당합니다. 예를 들어 저녁이나 주말 등 애플리케이션에 미치는 영향을 최소화하는 기간을 선택합니다.

데이터베이스 엔진 업그레이드의 경우 Amazon DocumentDB는 개별 인스턴스의 창이 아닌 클러스터의 창을 사용합니다.

다음 표에는 리전당 기본 시간 블록이 나와 있습니다.

리전 이름	리전	UTC 시간 블록
미국 동부(오하이오)	us-east-2	03:00~11:00
미국 동부(버지니아 북부)	us-east-1	03:00~11:00
미국 서부(오리곤)	us-west-2	06:00~14:00
Africa (Cape Town)	af-south-1	03:00~11:00
Asia Pacific (Hong Kong)	ap-east-1	06:00~14:00
아시아 태평양(하이데라바드)	ap-south-2	06:30-14:30
아시아 태평양(말레이시아)	ap-southeast-5	13:00~21:00
아시아 태평양(뭄바이)	ap-south-1	06:00~14:00
아시아 태평양(오사카)	ap-northeast-3	12:00–20:00
아시아 태평양(서울)	ap-northeast-2	13:00~21:00
아시아 태평양(싱가포르)	ap-southeast-1	14:00–22:00
아시아 태평양(시드니)	ap-southeast-2	12:00–20:00

리전 이름	리전	UTC 시간 블록
아시아 태평양(자카르타)	ap-southeast-3	08:00~16:00
아시아 태평양(멜버른)	ap-southeast-4	11:00~19:00
아시아 태평양(태국)	ap-southeast-7	15:00~23:00
아시아 태평양(도쿄)	ap-northeast-1	13:00~21:00
캐나다(중부)	ca-central-1	03:00~11:00
캐나다 서부(캘거리)	ca-west-1	18:00-02:00
중국(베이징)	cn-north-1	06:00~14:00
중국(닝샤)	cn-northwest-1	06:00~14:00
유럽(프랑크푸르트)	eu-central-1	21:00~05:00
유럽(취리히)	eu-central-2	02:00~10:00
유럽(아일랜드)	eu-west-1	22:00~06:00
Europe (London)	eu-west-2	22:00~06:00
Europe (Milan)	eu-south-1	02:00~10:00
유럽(파리)	eu-west-3	23:59~07:29
유럽(스페인)	eu-south-2	02:00~10:00
Europe (Stockholm)	eu-north-1	04:00~12:00
멕시코(중부)	mx-central-1	03:00~11:00
중동(UAE)	me-central-1	05:00~13:00
남아메리카(상파울루)	sa-east-1	00:00~08:00
이스라엘(텔아비브)	il-central-1	04:00-12:00

리전 이름	리전	UTC 시간 블록
AWS GovCloud(미국 동부)	us-gov-east-1	17:00–01:00
AWS GovCloud(미국 서부)	us-gov-west-1	06:00~14:00

Amazon DocumentDB 유지 관리 윈도우 변경

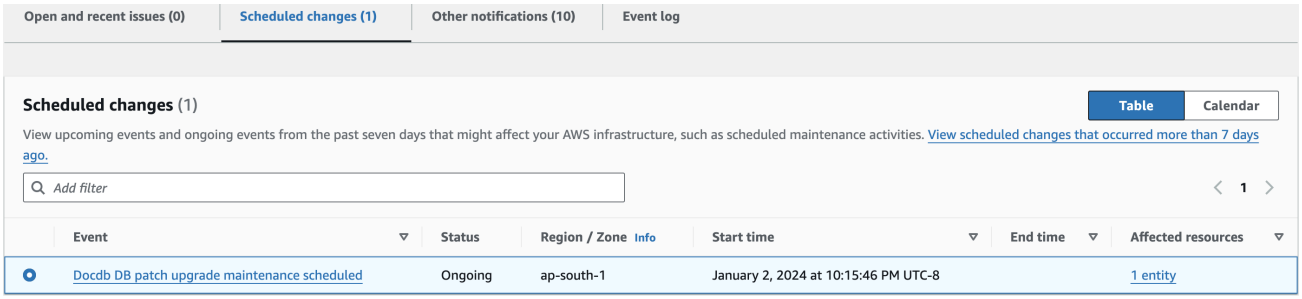
가능한 가장 낮은 트래픽 기간을 선택하고 트래픽 패턴이 이동함에 따라 시간 경과에 따라 조정합니다. 예를 들어 스케일 스토리지 작업 또는 인스턴스 클래스 변경과 같은 시스템 변경에 중단이 필요한 경우에만 기간 동안 클러스터 또는 인스턴스를 사용할 수 없으며, 해당 변경에 실제로 필요한 기간 동안만 사용할 수 있습니다.

유지 관리 기간 변경하기

- 클러스터: Amazon DocumentDB 클러스터 수정 단원을 참조하십시오.
- 인스턴스: Amazon DocumentDB 인스턴스 수정 단원을 참조하십시오.

Amazon DocumentDB 엔진 패치에 대한 알림

리전에서 필요한 엔진 패치를 사용할 수 있게 되면 해당 AWS 리전에 영향을 받는 Amazon DocumentDB 클러스터가 있는 모든 AWS 계정은 Health Dashboard (AHD) 및 이메일(AWS 계정의 루트 사용자 주소로 전송됨)을 통해 알림을 받습니다. 영향을 받는 Amazon DocumentDB 엔진 버전당 하나의 알림이 전송됩니다. AHD의 예약된 변경 사항에서 찾을 수 있습니다. 각 알림에는 패치 가용성 타이밍, 자동 적용 일정, 영향을 받는 클러스터 및 릴리스 정보가 나열됩니다.



필요한 엔진 패치는 약 30일의 단일 리드 타임을 따릅니다. 리전에서 패치를 사용할 수 있게 되면 Amazon DocumentDB는 위에 설명된 알림을 보냅니다. 이때 패치`AutoAppliedAfterDate`는 약 30일 후에 로 설정됩니다. 이 날짜까지 패치는 보류 상태로 유지됩니다. 언제든지 패치를 적용하거나 클러스터의 유지 관리 기간을 이후 날짜로 이동하여 패치를 연기할 수 있습니다. 또는 그 이후에 패치`AutoAppliedAfterDate`는 클러스터의 다음 유지 관리 기간 동안 자동으로 적용됩니다.

예를 들어, 2026년 6월 1일에 사용할 수 있게 되는 필수 패치AutoAppliedAfterDate의는 약 2026년 7월 1일입니다. 2026년 6월 1일에 알림을 받게 되며, 조치를 취하지 않으면 2026년 7월 1일 또는 그 이후에 클러스터의 첫 번째 유지 관리 기간에 패치가 자동으로 적용됩니다.

알림을 받은 후 두 가지 옵션이 있습니다. 자동 적용 날짜 이전에 패치를 자체 적용하거나 예정된 유지 관리 기간(기본값) 동안 패치가 자동으로 적용될 때까지 기다립니다. 자체 적용하려면 클러스터의 유지 관리 및 백업 탭을 열고 유형의 항목을 찾습니다system-update.

> **Note**
>
> Amazon DocumentDB가 새 패치 버전으로 다른 엔진 패치를 릴리스할 때까지 AHD의 알림 상태는 진행 중으로 유지됩니다.
> 패치가 적용되면 클러스터의 엔진 패치 버전이 알림의 버전과 일치하도록 업데이트됩니다. 를 실행하여 새 버전을 확인합니다db.runCommand({getEngineVersion: 1}).

선택적 패치와 새 마이너 버전은 AHD 또는 이메일 알림을 생성하지 않습니다. 이를 추적하려면 Amazon DocumentDB 릴리스 정보를 확인하세요.

강제 패치(가장 드문 범주로, 가장 중요한 보안 수정용으로 예약됨)도 AHD 및 이메일을 통해 발표됩니다. 필수 패치와 달리 유지 관리 기간 외부에서 적용되므로 위의 자동 적용 타이밍 예제는 적용되지 않습니다.

프로그래밍 방식으로 패치 알림에 대응

AWS Health 는 Amazon EventBridge와 통합되어 AWS Lambda 및 Amazon Simple Queue Service(SQS)를 포함하여 20개 이상의 대상에서 이벤트 기반 애플리케이션을 구축할 수 있습니다. 엔진 패치 가용성에 프로그래밍 방식으로 대응하려면 AWS_DOCDB_DB_PATCH_UPGRADE_MAINTENANCE_SCHEDULED 이벤트에 대해 EventBridge를 구성합니다. 여기에서 이벤트 데이터를 캡처하거나, 추가 이벤트를 발생시키거나,를 통해 푸시 알림을 보내거나 AWS Console Mobile Application, 필요한 다른 조치를 취할 수 있습니다.

Amazon DocumentDB가 패치(드문 경우)를 취소하면 AHD 알림과 취소에 대한 이메일을 받게 됩니다. Amazon EventBridge와 함께 AWS_DOCDB_DB_PATCH_UPGRADE_MAINTENANCE_CANCELLED 이벤트 코드를 사용하여이 사례를 처리합니다. 규칙 작성에 대한 자세한 내용은 Amazon EventBridge 사용 설명서를 참조하세요.

보류 중인 Amazon DocumentDB 유지 관리 작업 보기

AWS Management Console 또는를 사용하여 클러스터 또는 인스턴스에 대해 보류 중인 유지 관리를 AWS CLI 확인합니다.

보류 중인 업데이트는 엔진 패치와 OS 업데이트를 모두 `system-update`다루는 작업 유형와 함께 나타납니다.

업데이트가 보류 중인 경우 다음을 수행할 수 있습니다.

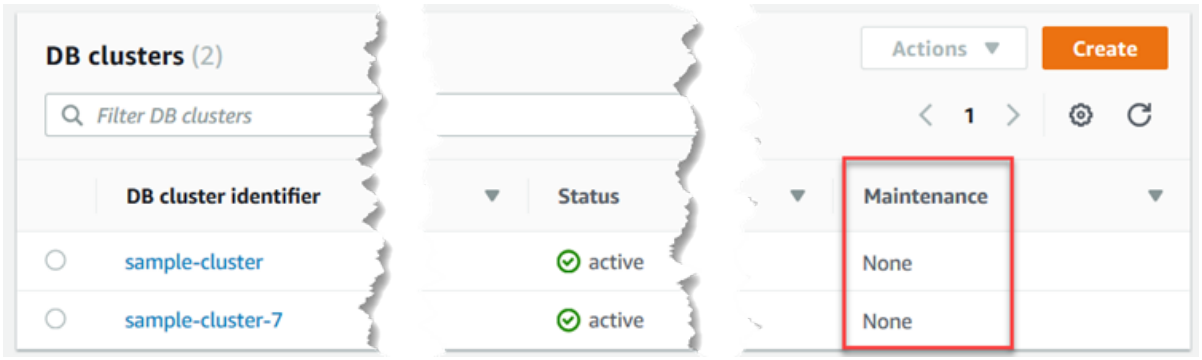
- 즉시 적용합니다.
- 다음 유지 관리 기간에 예약합니다.
- 유지 관리 기간을 이전으로 변경하여 연기합니다(엔진 패치 및 OS 업데이트만 해당). `AutoAppliedAfterDate` 해당 날짜가 지나면 다음 유지 관리 기간 동안 작업이 자동으로 적용됩니다. 일단 `ForcedApplyDate` 통과하면 더 이상 연기할 수 없습니다.

> **Note**
>
> 조치를 취하지 않으면 예정된 유지 관리 기간 동안 필요한 엔진 패치와 같은 필수 유지 관리 작업이 자동으로 적용됩니다. 선택적 패치와 마이너 버전은 자동으로 적용되지 않습니다.

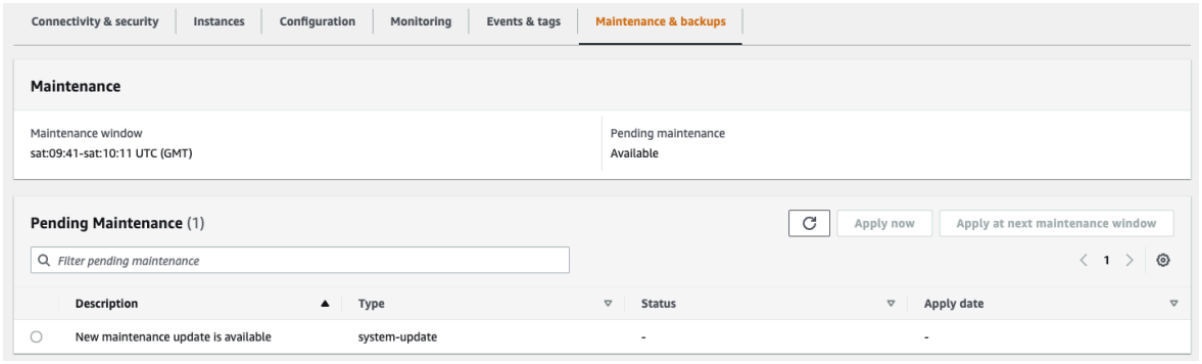
유지 관리 기간은 작업이 완료되는 데 걸리는 시간이 아니라 보류 중인 작업이 시작되는 시간을 제어합니다.

Using the AWS Management Console

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 업데이트가 보류 중일 때 클러스터의 유지 관리 열에 사용 가능, 필요 또는 다음 창이 표시됩니다.



4. 클러스터를 열고 유지 관리 및 백업을 선택하여 보류 중인 유지 관리 항목을 확인하고 조치를 취합니다.



Using the AWS CLI

를 실행`describe-pending-maintenance-actions`하여 보류 중인 항목을 확인합니다. 다음 예제에서는 보류 중인 작업이 없는 계정을 보여줍니다.

```
aws docdb describe-pending-maintenance-actions
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "PendingMaintenanceActions": []
}
```

보류 중인 작업이 있는 계정은 다음과 같은 출력을 반환합니다.

```
{
    "PendingMaintenanceActions": [
        {
```

```
            "ResourceIdentifier": "arn:aws:rds:us-
east-1:123456789012:cluster:sample-cluster",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "system-update",
                    "Description": "db-version-upgrade",
                    "CurrentApplyDate": "2026-05-15T03:01:00Z",
                    "AutoAppliedAfterDate": "2026-05-15T03:01:00Z"
                }
            ]
        }
    ]
}
```

형식으로를 사용하여 목록의 범위를 특정 클러스터로 지정할 수 `--filters`있습니다`Name=`*`filter-name`*`,Values=`*`resource-id`*`,...`. 허용되는 필터`Name`는 클러스터 식별자 또는 ARN 목록을 `db-cluster-id`가져오는 입니다. ARNs

Example

Linux, macOS, Unix의 경우:

```
aws docdb describe-pending-maintenance-actions \
   --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

Windows의 경우:

```
aws docdb describe-pending-maintenance-actions ^
   --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

적용 날짜

보류 중인 각 유지 관리 작업에는 최대 3개의 적용 날짜가 포함됩니다. 의 AWS CLI 출력에 나타나`describe-pending-maintenance-actions`며 작업이 실행될 시기를 나타냅니다. 필드는 선택적 유지 관리를 `null` 위한 것입니다.

- **`CurrentApplyDate`**- 현재 또는 다음 유지 관리 기간에 작업이 실행되도록 예약된 경우. 필수 및 강제 작업에 대해 채워집니다.
- **`AutoAppliedAfterDate`**- 클러스터 또는 인스턴스 유지 관리 기간 동안 자동 적용이 시작되는 날짜입니다. 필수 작업에 대해 채워집니다.

- **`ForcedApplyDate`**- 어려운 기한입니다. 이 날짜 이후에는 유지 관리 기간에 관계없이 작업이 자동으로 실행됩니다. 강제 작업에 대해 채워집니다.

보류 중인 작업을 연기하려면 유지 관리 기간을 이전 나중으로 이동합니다`AutoAppliedAfterDate`. 가 `AutoAppliedAfterDate` 통과하면 작업은 다음 유지 관리 기간 동안 자동으로 적용됩니다. 일단 `ForcedApplyDate` 통과하면 더 이상 연기할 수 없습니다. 정확한 지연 기간은 패치에 따라 다릅니다. 날짜는 AHD 알림 및 AWS CLI 출력에 게시됩니다.

Amazon DocumentDB 엔진 업데이트

보류 중인 엔진 패치를 식별한 경우 다음 절차 중 하나를 사용하여 적용하거나 예약합니다. AWS Management Console 또는에서 이러한 절차를 실행할 수 있습니다 AWS CLI.

Using the AWS Management Console

클러스터에 대한 업데이트를 관리하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 업데이트할 클러스터를 선택합니다.
4. 작업 메뉴에서 다음 중 하나를 선택합니다.

   - 지금 업그레이드 - 보류 중인 유지 관리를 즉시 실행합니다.
   - 다음 기간에 업그레이드 - 클러스터의 다음 유지 관리 기간에 실행합니다.

   클러스터의 유지 관리 및 백업 탭에 있는 보류 중인 유지 관리 섹션에서 지금 적용 또는 다음 유지 관리 기간에 적용을 사용할 수도 있습니다(참조보류 중인 Amazon DocumentDB 유지 관리 작업 보기).

   > **Note**
   >
   > 보류 중인 옵션이 없으면 이러한 모든 옵션이 비활성화됩니다.

Using the AWS CLI

를 사용하여 보류 중인 업데이트를 적용합니다`apply-pending-maintenance-action`.

파라미터

- **`--resource-identifier`**- 보류 중인 작업 대상이 되는 리소스의 Amazon DocumentDB Amazon 리소스 이름(ARN)입니다.
- **`--apply-action`**- 적용할 보류 중인 유지 관리 작업입니다. `system-update`를 사용하여 엔진 패치를 적용합니다.
- **`--opt-in-type`**- 옵트인 요청 유형 또는 실행 취소 여부입니다. 유효한 값:
  - `immediate`- 지금 적용합니다. 제출한 후에는 실행 취소할 수 없습니다.
  - `next-maintenance`- 리소스의 다음 유지 관리 기간에 적용됩니다.
  - `undo-opt-in`- 기존 `next-maintenance`옵트인을 취소합니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb apply-pending-maintenance-action \
    --resource-identifier arn:aws:rds:us-east-1:123456789012:db:sample-cluster-instance-1 \
    --apply-action system-update \
    --opt-in-type immediate
```

Windows의 경우:

```
aws docdb apply-pending-maintenance-action ^
    --resource-identifier arn:aws:rds:us-east-1:123456789012:db:sample-cluster-instance-1 ^
    --apply-action system-update ^
    --opt-in-type immediate
```

패치 적용 중 읽기 가용성

Amazon DocumentDB 엔진 5.0 및 8.0은 클러스터에 여러 인스턴스가 있는 경우 패치 적용 중에 읽기 가용성을 유지합니다. Amazon DocumentDB는 리더 인스턴스를 세 그룹으로 롤링 방식으로 패치하므로 나머지 리더는 트래픽을 계속 제공합니다. 패치하는 동안 라이터를 잠시 사용할 수 없습니다. 제로 읽기 가동 중지 시간을 달성하려면 읽기 기본 설정을 지정하여 읽기가 라이터로 돌아가`secondaryPreferred`거나, 또는 `primaryPreferred` 작업`primary`하거나, `secondary` 단독으로 읽기 가동 중지 시간이 발생할 수 있도록 합니다.

읽기 기본 설정 모드	라이터 업그레이드 중	리더 업그레이드 중	제로 읽기 가동 중지에 필요한 최소 리더 수
primary	읽기/쓰기 가동 중지 시간	영향 없음	해당 사항 없음
primaryPreferred	쓰기 가동 중지 시간	영향 없음	1
secondary	쓰기 가동 중지 시간	가동 중지 시간 읽기 (리더가 하나뿐인 경우)	2
secondaryPreferred	쓰기 가동 중지 시간	영향 없음	1
nearest	쓰기 가동 중지 시간	영향 없음	1

리더는 패치를 적용하는 동안 전체 클러스터 읽기 처리량이 일시적으로 감소합니다. 처리량을 안정적으로 유지하려면 업그레이드 전에 추가 리더를 프로비저닝하고 완료 후 제거합니다.

엔진 3.6 및 4.0에서는 이러한 읽기 가용성 기능이 적용되지 않습니다. 엔진 패치로 인해 가동 중지 시간이 길어져 읽기 및 쓰기 모두에 영향을 미칩니다. 가 수행하는 메이저 버전으로 업그레이드하려면 섹션을 참조하세요Amazon DocumentDB 인플레이스 주요 버전 업그레이드.

패치 가동 중지 시간 길이

엔진 패치 가동 중지 시간은 다양합니다. 가장 큰 요인은 패치 시 인스턴스에 대한 CPU 사용률 및 메모리 압력이므로 인스턴스 크기를 올바르게 조정하는 것이 중요합니다. 가동 중지 시간을 최소화하려면 최신 Amazon DocumentDB 메이저 엔진 버전을 실행하고 여러 가용 영역에 인스턴스를 분산합니다.

패치 업데이트 및 교체

Amazon DocumentDB는 릴리스 후 패치를 모니터링합니다. 드문 경우지만 문제가 식별되면 Amazon DocumentDB는 업데이트된 버전을 준비하는 동안 롤아웃을 일시 중지합니다. 이 경우 패치를 아직 받지 않은 클러스터는 더 이상 해당 패치를 사용 가능한 유지 관리 작업으로 볼 수 없으며의 예약된 변경 알림 Health Dashboard 이 취소됩니다. 영향을 받는 버전을 이미 실행 중인 클러스터는 계속 정상적으로 작동하므로 사용자의 조치가 필요하지 않습니다.

곧 업데이트된 패치가 제공됩니다. 리전에서 사용할 수 있게 되면에 설명된 대로 Health Dashboard 및 이메일을 통해 새 알림을 받게 됩니다Amazon DocumentDB 엔진 패치에 대한 알림.

마이너 버전 업그레이드

Amazon DocumentDB는 메이저 버전 5.0 이상(예: `5.0.1`)을 기반으로 마이너 버전을 게시합니다. 마이너 버전은 5.0 이전의 메이저 버전에는 게시되지 않습니다. 마이너 버전은 필수 및 선택적 엔진 패치와 다르게 작동합니다.

- 보류 중인 유지 관리 작업으로 표시되지 않으며 자동으로 적용되지 않습니다.
- AHD 또는 이메일 알림은 생성하지 않습니다. 새 마이너 버전은 Amazon DocumentDB 릴리스 정보에 발표됩니다.
- 업그레이드하려면 클러스터의 엔진 버전을 수정합니다(즉시 또는 다음 유지 관리 기간 동안). 마이너 버전 업그레이드에는 짧은 가동 중지 시간이 필요하며 단방향이므로 이전 마이너 버전으로 다운그레이드할 수 없습니다. 글로벌 클러스터의 경우 기본 클러스터보다 먼저 보조 클러스터를 업그레이드합니다.

자세히 알아보기: Amazon DocumentDB 마이너 버전 업그레이드.

Amazon DocumentDB 운영 체제 업데이트

인스턴스에 OS 업데이트가 필요한 경우가 있습니다. Amazon DocumentDB는 OS를 업데이트하여 성능을 개선하고 보안을 강화합니다. OS 업데이트는 클러스터 엔진 버전과 인스턴스 클래스를 변경하지 않고 그대로 둡니다. 엔진 패치와 마찬가지로 OS 업데이트는이 주제의 맨 위에 설명된 선택적/필수/강제 수명 주기를 사용합니다. 엔진 패치와 달리 OS 업데이트는 연기하면 시간이 지남에 따라 이러한 범주를 전환할 수 있습니다. OS 업데이트가 제공되는 즉시 적용하고 클러스터 및 인스턴스 유지 관리 기간을 비즈니스 요구 사항에 맞는 시간으로 설정합니다.

클러스터 수준 `os-upgrade` 유지 관리 작업을 사용하여 클러스터의 모든 인스턴스에 OS 업데이트를 적용합니다. Amazon DocumentDB는 인스턴스를 한 번에 몇 개씩 롤링 방식으로 업데이트하고 기본 인스턴스를 마지막으로 업데이트하여 장애 조치를 최소화합니다. 업데이트는 구성한 개별 인스턴스 유지 관리 기간이 아닌 클러스터 유지 관리 기간 중에 실행됩니다.

인스턴스가 OS 업데이트를 수신하면 버퍼 캐시가 비어 있기 시작합니다. 작업 세트가 스토리지 볼륨에서 다시 채워질 때까지 해당 인스턴스의 쿼리는 지연 시간이 길어지고가 낮아질 수 있습니다`BufferCacheHitRatio`.

Amazon DocumentDB가 기본 인스턴스를 업데이트하면 장애 조치가 복제본을 새 기본 인스턴스로 승격합니다. 애플리케이션이 이를 투명하게 처리하도록 클러스터 엔드포인트를 사용합니다. 인스턴스가

업데이트되는 동안 읽기 가용성을 유지하려면 읽기 기본 설정을 `secondaryPreferred` 또는 로 설정`primaryPreferred`하면 읽기가 사용 가능한 인스턴스로 되돌아갈 수 있습니다. 잠재적 장애 조치 대상(우선 순위가 가장 높은 계층의 복제본)을 기본 인스턴스 클래스와 동일한 인스턴스 클래스에 유지합니다. 이렇게 하면 승격 후 쓰기 성능이 저하되지 않습니다. 자세한 내용은 Amazon DocumentDB 장애 조치 섹션을 참조하세요.

클러스터 수준 `system-update` 작업`os-upgrade`과 인스턴스 수준 작업은 모두 사용 가능한 작업`describe-pending-maintenance-actions`으로에 동시에 표시될 수 있습니다. 그러나 둘 다 동시에 예약할 수는 없습니다. 인스턴스 수준 `system-update` 작업이 인스턴스에 대해 적극적으로 예약된 경우 클러스터 수준 `os-upgrade` 작업을 예약하기 전에 취소하거나 완료해야 하며 그 반대의 경우도 마찬가지입니다.

Important

OS 업데이트를 위해 Amazon DocumentDB 인스턴스가 오프라인 상태가 됩니다. 다중 인스턴스 클러스터는 영향을 최소화합니다. 단일 인스턴스 클러스터를 실행하는 경우 업데이트에 대한 보조 클러스터를 일시적으로 추가하고 나중에 제거할 수 있습니다. 보조는 존재하는 동안 일반적인 요금이 발생합니다.

Note

이전 버전과의 호환성을 위해 인스턴스 수준 `system-update` 작업을 계속 사용할 수 있습니다. 이를 사용해야 하는 경우 복제본을 먼저 업데이트하고 기본 복제본을 마지막으로 업데이트합니다. 패치 중에 장애 조치가 가동 중지 시간을 연장할 수 있으므로 동시에 패치하지 마세요.

새 선택적 OS 업데이트가 도착할 때 이벤트를 가져오려면 보안 패치 이벤트 범주`RDS-EVENT-0230`에서를 구독합니다. 자세한 내용은 Amazon DocumentDB 이벤트 구독 섹션을 참조하세요.

Note

규정 준수를 위해 선택적 필수 업데이트를 최신 상태로 유지해야 할 수 있습니다. 유지 관리 기간 동안 `os-upgrade` 작업을 정기적으로 적용합니다.

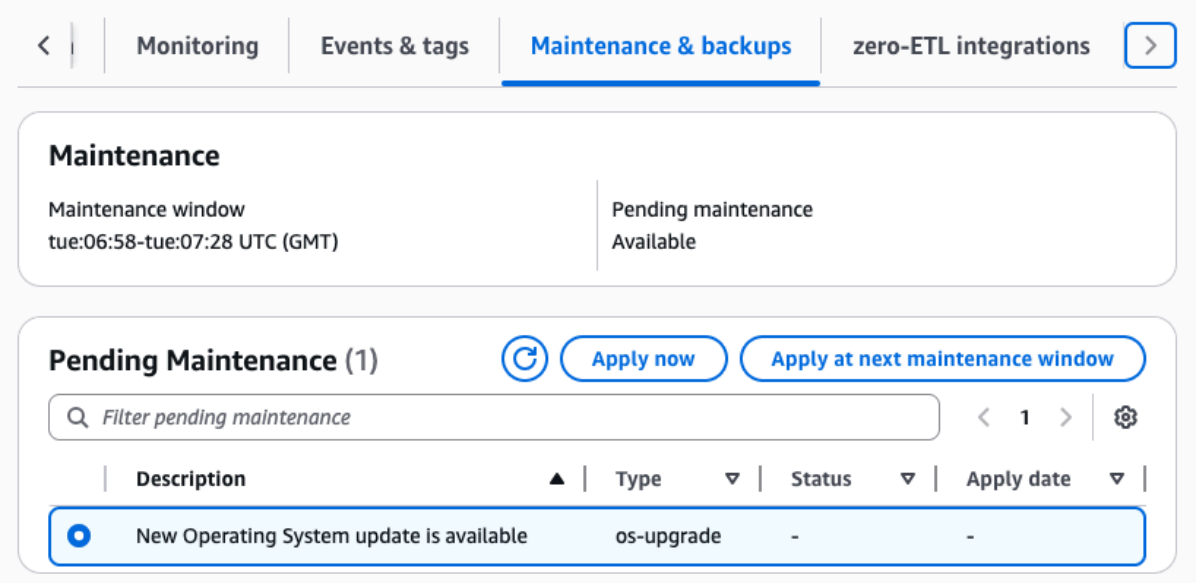
OS 업데이트는 특정 인스턴스 클래스에 연결되어 있으므로 서로 다른 인스턴스가 서로 다른 시간에 사용할 수 있게 됩니다. 클러스터가 최신 엔진 패치에 없는 경우 OS 업데이트가 표시되지 않을 수 있습니다. 먼저 최신 엔진 패치를 적용합니다(참조Amazon DocumentDB 엔진 업데이트).

AWS Management Console 또는를 사용하여 업데이트를 사용할 수 있는지 AWS CLI 확인합니다.

Using the AWS Management Console

콘솔에서 OS 업데이트를 확인하려면:

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택한 다음 클러스터 이름을 선택합니다.
3. 유지 관리 및 백업 탭을 선택합니다.
4. 유지 관리 보류 중에서 OS 업데이트를 사용할 수 있는 경우 `os-upgrade` 작업이 나타납니다.



5. `os-upgrade` 작업을 선택하고 지금 적용 또는 다음 유지 관리 기간에 적용을 선택합니다. 값이 다음 기간인 경우 작업이 시작되지 않은 한 업그레이드 연기로 연기할 수 있습니다.

Using the AWS CLI

보류 중인 OS 업데이트를 확인합니다.

```
aws docdb describe-pending-maintenance-actions
```

```
{
```

```
    "PendingMaintenanceActions": [
        {
            "ResourceIdentifier": "arn:aws:rds:aa-
example-1:111122223333:cluster:sample-cluster",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "os-upgrade",
                    "Description": "New Operating System update is available"
                }
            ]
        },
        {
            "ResourceIdentifier": "arn:aws:rds:aa-example-1:111122223333:db:sample-
cluster-instance-1",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "system-update",
                    "Description": "New Operating System update is available"
                }
            ]
        },
        {
            "ResourceIdentifier": "arn:aws:rds:aa-example-1:111122223333:db:sample-
cluster-instance-2",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "system-update",
                    "Description": "New Operating System update is available"
                }
            ]
        }
    ]
}
```

OS 업데이트는 클러스터 수준에서 로os-upgrade, 인스턴스 수준에서 로 표시됩니다system-update. 클러스터 수준 os-upgrade 작업을 사용합니다.

Example

다음 예제에서는 OS 업데이트를 즉시 적용합니다.

Linux, macOS, Unix의 경우:

```
aws docdb apply-pending-maintenance-action \
```

```
    --resource-identifier arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster \
    --apply-action os-upgrade \
    --opt-in-type immediate
```

Windows의 경우:

```
aws docdb apply-pending-maintenance-action ^
    --resource-identifier arn:aws:rds:aa-example-1:111122223333:cluster:sample-
cluster ^
    --apply-action os-upgrade ^
    --opt-in-type immediate
```

사용자 시작 업데이트

메모리가 많거나 적은 인스턴스 클래스로 바꾸거나 클러스터의 파라미터 그룹을 변경하는 등 직접 시작하는 일부 변경 사항. Amazon DocumentDB는 이를 시작하는 업데이트와 다르게 처리합니다. 자세한 내용은 다음을 참조하세요.

- Amazon DocumentDB 클러스터 수정
- Amazon DocumentDB 인스턴스 수정

아직 보류 중인 사용자 시작 변경 사항을 나열하려면:

Example

인스턴스에 대해 보류 중인 사용자 시작 변경 사항을 나열하려면

Linux, macOS, Unix의 경우:

```
aws docdb describe-db-instances \
    --query 'DBInstances[*].
[DBClusterIdentifier,DBInstanceIdentifier,PendingModifiedValues]'
```

Windows의 경우:

```
aws docdb describe-db-instances ^
    --query 'DBInstances[*].
[DBClusterIdentifier,DBInstanceIdentifier,PendingModifiedValues]'
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

이 예제에서 `sample-cluster-instance`는에 대해 보류 중인 변경 사항이 `db.r5.xlarge`있으며 `sample-cluster-instance-2`는 없습니다.

```
[
    [
        "sample-cluster",
        "sample-cluster-instance",
        {
            "DBInstanceClass": "db.r5.xlarge"
        }
    ],
    [
        "sample-cluster",
        "sample-cluster-instance-2",
        {}
    ]
]
```

글로벌 클러스터 패치 적용

글로벌 클러스터에서는 기본 및 보조의 각 멤버 클러스터가 자체 유지 관리 기간 동안 업그레이드됩니다. 모든 리전에서 필요한 엔진 패치를 사용할 수 있는 경우 AHD 및 이메일 알림을 받게 됩니다. 선택적 패치와 새 마이너 버전은 알림을 생성하지 않습니다. 이에 대한 Amazon DocumentDB 릴리스 정보를 확인하세요.

자체 적용하는 경우 항상 보조 항목을 먼저 패치하고 기본 항목을 마지막으로 패치합니다. 이 순서는 롤아웃 전반에 걸쳐 장애 조치 및 전환을 계속 사용할 수 있도록 합니다.

> **Important**
>
> 실수로 기본를 먼저 패치하는 경우 모든 보조를 가능한 한 빨리 동일한 버전으로 만듭니다. 장애 조치 및 전환은 모든 클러스터가 동일한 버전에 있을 때까지 비활성화된 상태로 유지됩니다.

아무 조치도 취하지 않으면 패치는 각 클러스터의 다음 유지 관리 기간인 세컨더리 먼저, 세컨더리가 완료되면 해당 기간의 기본 기간에 자동으로 적용됩니다.

기본 DB 클러스터와 보조 DB 클러스터를 동일한 버전으로 유지합니다. 관리형 리전 간 장애 조치는 모든 클러스터가 동일한 엔진 버전과 패치 수준을 공유하는 경우에만 글로벌 데이터베이스에서 작동합니다. 기본 버전보다 최신 엔진 버전을 사용하는 새 보조 버전을 추가하는 경우에도 마찬가지입니다. 글로벌 데이터베이스에 조인하기 전에 기본 버전에 새 보조 버전을 생성합니다.

패치 알림 후 장애 조치 및 전환이 계속 작동하도록 최대한 빨리 기본 및 보조를 최신 버전으로 업그레이드합니다. 장애 조치 또는 전환 요청이 거부되면 클러스터 간에 엔진 패치 버전을 비교하고, 일치하지 않으면 지연 클러스터에 사용 가능한 패치를 적용합니다.

서비스 연결 역할 이해

Amazon DocumentDB(MongoDB 호환)는 AWS Identity and Access Management (IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 Amazon DocumentDB에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon DocumentDB에서 사전 정의하며 서비스가 사용자를 대신하여 다른 AWS 서비스를 호출하는 데 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 Amazon DocumentDB를 더 쉽게 사용할 수 있습니다. Amazon DocumentDB에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, Amazon DocumentDB만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔터티에 연결할 수 없습니다.

먼저 역할의 관련 리소스를 삭제해야만 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 실수로 삭제할 수 없기 때문에 Amazon DocumentDB 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 IAM으로 작업하는AWS 서비스를 참조해 서비스 연결 역할 열이 예(Yes)인 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예(Yes) 링크를 선택합니다.

Amazon DocumentDB 서비스 연결 역할 권한

Amazon DocumentDB(MongoDB 호환)는 AWSServiceRoleForRDS라는 서비스 연결 역할을 사용하여 Amazon DocumentDB가 클러스터를 대신하여 AWS 서비스를 호출하도록 허용합니다.

AWSServiceRoleForRDS 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `docdb.amazonaws.com`

역할 권한 정책은 Amazon DocumentDB가 지정된 리소스에서 다음 작업을 완료하도록 허용합니다.

- ec2에 대한 작업:
  - AssignPrivateIpAddresses
  - AuthorizeSecurityGroupIngress
  - CreateNetworkInterface
  - CreateSecurityGroup
  - DeleteNetworkInterface
  - DeleteSecurityGroup
  - DescribeAvailabilityZones
  - DescribeInternetGateways
  - DescribeSecurityGroups
  - DescribeSubnets
  - DescribeVpcAttribute
  - DescribeVpcs
  - ModifyNetworkInterfaceAttribute
  - RevokeSecurityGroupIngress
  - UnassignPrivateIpAddresses
- sns에 대한 작업:
  - ListTopic
  - Publish
- cloudwatch에 대한 작업:
  - PutMetricData
  - GetMetricData
  - CreateLogStream
  - PullLogEvents
  - DescribeLogStreams
  - CreateLogGroup

> **Note**
>
> IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 연결 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 다음과 같은 오류 메시지가 표시될 수 있습니다.

리소스를 만들 수 없습니다. 서비스 연결 역할을 생성할 권한이 있는지 확인하십시오. 그렇지 않은 경우 기다렸다가 나중에 다시 시도하십시오.
이러한 오류가 표시되면 다음 권한이 활성화되어 있는지 확인합니다.

```
{
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName":"rds.amazonaws.com"
        }
    }
}
```

자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한 섹션을 참조하세요.

Amazon DocumentDB 서비스 연결 역할 생성

서비스 연결 역할은 수동으로 생성할 필요가 없습니다. 클러스터를 생성할 때 Amazon DocumentDB에서는 서비스 연결 역할을 생성합니다.

이 서비스 연결 역할을 삭제한 다음 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 클러스터를 생성할 때 Amazon DocumentDB에서는 서비스 연결 역할을 다시 생성합니다.

Amazon DocumentDB 서비스 연결 역할 수정

Amazon DocumentDB에서는 AWSServiceRoleForRDS 서비스 연결 역할을 수정할 수 없습니다. 서비스 연결 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 그러나 IAM을 사용하여 역할의 설명을 수정할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 편집을 참조하세요.

Amazon DocumentDB 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스를 더 이상 사용할 필요가 없는 경우 해당 역할을 삭제합니다. 이렇게 하면 적극적으로 모니터링되거나 유지 관리되지 않는 미사용 엔터티가 없습니다. 그러나 서비스 연결 역할을 삭제하려면 먼저 모든 클러스터를 삭제해야 합니다.

Amazon DocumentDB 서비스 연결 역할 정리

IAM을 사용하여 서비스 연결 역할을 삭제하기 전에 먼저 역할에 활성 세션이 없는지 확인하고 역할에서 사용되는 리소스를 모두 제거해야 합니다.

콘솔을 사용하여 서비스 연결 역할에 활성 세션이 있는지 확인하려면 다음을 수행합니다.

1. 에 로그인 AWS Management Console 하고에서 IAM 콘솔을 엽니다https://console.aws.amazon.com/iam/.
2. IAM 콘솔의 탐색 창에서 역할을 선택한 다음 AWSServiceRoleForRDS 역할의 이름(확인란 아님)을 선택합니다.
3. 선택한 역할의 요약 페이지에서 Access Advisor 탭을 선택합니다.
4. Access Advisor 탭에서 서비스 연결 역할의 최근 활동을 검토합니다.

> **Note**
>
> Amazon DocumentDB에서 AWSServiceRoleForRDS 역할을 사용하는지 잘 모를 경우에는 역할을 삭제할 수 있습니다. 서비스에서 역할을 사용하는 경우에는 삭제가 안 되어 역할이 사용 중인 리전을 볼 수 있습니다. 역할이 사용 중인 경우에는 세션이 종료될 때까지 기다렸다가 역할을 삭제해야 합니다. 서비스 연결 역할에 대한 세션은 취소할 수 없습니다.

AWSServiceRoleForRDS 역할을 제거하려면 먼저 모든 인스턴스 및 클러스터를 삭제해야 합니다. 인스턴스 및 클러스터 삭제에 대한 자세한 내용은 다음 주제를 참조하십시오.

- Amazon DocumentDB 인스턴스 삭제
- Amazon DocumentDB 클러스터 삭제

Amazon DocumentDB 서비스 연결 역할에 지원되는 리전

Amazon DocumentDB는 서비스가 제공되는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용은 https://docs.aws.amazon.com/documentdb/latest/devguide/regions-and-azs.html#regions-and-azs-availability 단원을 참조하십시오.

성능 최적화

이 장에서는 Amazon DocumentDB 성능을 최적화하고 일반적인 성능 문제를 해결하는 방법에 대한 지침을 제공합니다.

주제

- 일반적인 성능 문제
- 안티 패턴
- 성능 개선 팁
- 쿼리 계획 분석
- 인덱스 전용 스캔(적용된 쿼리)
- 캐시 필터 계획
- Amazon DocumentDB 모범 사례
- Amazon DocumentDB 인덱스 관리
- 컬렉션 수준 문서 압축 관리
- Amazon DocumentDB 8.0에서 사전 기반 압축 관리

일반적인 성능 문제

이 섹션에서는 Amazon DocumentDB에서 발생할 수 있는 일반적인 성능 문제와 이를 해결하는 방법을 다룹니다.

주제

- 높은 CPU 사용률
- Amazon DocumentDB 연결 문제
- 쿼리 실행 속도가 느림

높은 CPU 사용률

높은 CPU 사용률은 Amazon DocumentDB로 작업하는 동안 발생할 수 있는 문제 중 하나입니다. 이 섹션에서는 문제를 해결하기 위한 인사이트를 제공합니다.

식별 - 문제 발견

Amazon DocumentDB 인스턴스의 CPU 사용률이 높으면 모든 데이터베이스 작업에서 지연 시간이 증가하여 애플리케이션 속도가 느려집니다. Amazon DocumentDB CPU 사용량이 최적 수준 이상으로 증가하면 쿼리 실행 시간이 증가하고 연결 설정이 더 오래 걸리며 애플리케이션 응답성이 감소합니다. 기본 인스턴스의 CPU 피크는 쓰기 작업이 느리지만 복제본의 피크는 읽기 작업의 지연 시간을 높입니다.

Amazon DocumentDB에서 CPU 사용률이 높은 일반적인 원인은 다음과 같습니다.

- 컬렉션 스캔: 적절한 인덱스가 없는 쿼리는 전체 컬렉션 스캔을 강제로 실행하여 모든 문서를 검사하기 위해 과도한 CPU를 사용합니다.
- 연결 스파이크: 애플리케이션 연결의 갑작스러운 급증으로 인증 및 세션 관리에 필요한 CPU 리소스가 급증합니다.
- 복잡한 집계: 다단계 작업(정렬, 그룹화, 조인)은 특히 인덱스가 최적화되지 않은 대규모 데이터 세트에서 상당한 CPU를 소비합니다.
- 비효율적인 쿼리: 대규모 결과 집합 및 정규식 작업을 스캔하여 CPU 사용량이 높습니다.
- 쓰기 중심 워크로드: 문서 처리, 인덱스 업데이트 및 트랜잭션 로깅을 통해 CPU 부담을 자주 업데이트하고 대량으로 삽입합니다.

조사 - 지표 수집

Amazon DocumentDB는 Amazon CloudWatch를 통해 모니터링을 제공합니다. Amazon DocumentDB 지표는 클러스터 수준 및 인스턴스 수준 지표로 광범위하게 분류할 수 있습니다. CPU, 메모리 및 연결과 관련된 리소스의 경우 리소스 소비가 인스턴스에 고유하므로 인스턴스 수준 지표를 살펴봅니다. CPU 스파이크를 표시하는 인스턴스(기본 또는 복제본)를 식별하고 스파이크 시간을 기록해 둡니다.

의 CPU 사용률에 대한 CloudWatch 지표입니다`CPUUtilization`.

진단 - 근본 원인 찾기

다음은 CPU 스파이크의 인스턴스와 타임라인을 식별한 후 CPU 사용량에 큰 영향을 미치는 원인을 진단하는 것입니다. CPU 스파이크를 일으킬 수 있는 여러 가지 이유가 있으므로에서 로 시작하는 CPU 스파이크의 원인을 식별해 보겠습니다.

연결: `DatabaseConnections` 및를 포함한 CloudWatch 지표를 확인합니다. 연결이 급증하고 CPU 급증 시간과 일치하는 `DatabaseConnectionsMax`경우 짧은 시간 동안의 연결 급증이 CPU 급증의 원인일 수 있습니다.

쿼리: 연결이 아닌 경우 쿼리가 원인일 수 있습니다. 성능 인사이트를 활성화한 경우 인스턴스의 성능 인사이트 콘솔로 이동하여 인스턴스의 쿼리를 확인합니다. 쿼리를 살펴볼 또 다른 것은 프로파일러에서 가져온 것입니다.

해결 - 문제 해결

연결 버스트: 연결 버스트가 CPU 급증의 원인인 경우 애플리케이션에서 연결을 재사용하거나 연결 풀링을 구현하는 것이 좋습니다. 높은 연결 최적화에 대한 자세한 내용은 섹션을 참조하세요the section called “연결 문제”.

쿼리 배포: 기본 인스턴스의 읽기 쿼리로 인해 CPU 스파이크가 발생하는 경우 쿼리 결과가 최종적으로 일치할 수 있는 경우 읽기 작업을 복제본 인스턴스로 리디렉션하는 것이 좋습니다.

쿼리 최적화: 쿼리 또는 쿼리 세트로 인해 CPU가 급증하는 경우 쿼리가 인덱스를 사용하지 않는 경우 인덱스를 추가하는 등 쿼리를 최적화하는 것이 좋습니다. 쿼리 최적화에 대한 자세한 내용은 섹션을 참조하세요the section called “쿼리 실행 속도가 느림”.

인스턴스 크기 조정: 최적화 작업 후에도 CPU 사용률이 지속적으로 높은 경우 워크로드 요구 사항을 효과적으로 처리할 수 있도록 CPU 용량이 더 많은 더 큰 인스턴스 클래스로 업그레이드합니다.

서버리스: 단기간 CPU 급증을 일으키는 예측할 수 없는 워크로드의 경우 사용을 고려하세요Amazon DocumentDB Serverless 사용. 서버리스 인스턴스는 필요에 따라 자동으로 리소스를 확장하므로 이러한 워크로드에 고정 인스턴스 크기를 프로비저닝할 필요가 없습니다.

Amazon DocumentDB 연결 문제

식별 - 문제 발견

일반적인 원인

연결 문제는 일반적으로 세 가지 주요 영역에서 비롯됩니다.

연결 풀 소진은 애플리케이션이 클라이언트 측 연결 풀 제한 또는 서버 측 인스턴스 제한을 통해 Amazon DocumentDB에 허용되는 최대 연결에 도달할 때 발생합니다. 이 조건은 새 연결 요청이 대기열에 추가되거나 거부될 때 애플리케이션 성능 저하, 제한 시간 초과 및 잠재적 실패로 이어집니다.

인증 오버로드는 Amazon DocumentDB에 과도한 동시 인증 요청이 발생할 때, 특히 단기간에 1,000개 이상의 새 연결을 처리할 때 발생합니다. 인증 유지 관리 중에 Amazon DocumentDB는 세션 맵에 독점적 잠금을 유지하여 유지 관리가 완료될 때까지 후속 인증 시도가 대기열에 추가되도록 합니다.

Amazon DocumentDB의 구성 문제는 네트워킹, 보안 및 클라이언트 설정의 잘못된 구성으로 인해 발생하는 경우가 많습니다. 여기에는 잘못된 보안 그룹 설정, 부적절한 VPC 구성 또는 SSL/TLS 인증서

문제와 같은 항목이 포함됩니다. 안전하고 안정적인 데이터베이스 액세스를 유지하려면 적절한 구성을 이해하는 것이 중요합니다.

진단 - 근본 원인 찾기

연결 풀

MongoClient 인스턴스를 생성할 때 연결 풀이 초기화됩니다. 각 풀은 두 가지 주요 파라미터를 기반으로 연결을 유지합니다.

minPoolSize - 유지 관리되는 최소 연결 수

maxPoolSize - 허용되는 최대 연결 수

요청에 연결이 필요한 경우:

1. 풀은 사용 가능한 유휴 연결을 확인합니다.
2. 존재하지 않고 풀 크기가 < maxPoolSize인 경우 새 연결이 생성됩니다.
3. maxPoolSize인 경우 요청은 대기열에 들어갑니다.
4. 대기열이 가득 찼거나 제한 시간에 도달하면 MongoWaitQueueFullException이 발생합니다.

대기열의 동작은 다음 파라미터를 통해 처리됩니다.

waitQueueTimeoutMS - 최대 연결 대기 시간

waitQueueSize - 최대 대기 요청 수

다음은 매번 새 풀이 생성되는 Amazon DocumentDB에 연결하기 위한 문제가 있는 접근 방식의 예입니다.

```
for(Request request : requests) {
    MongoClient client = MongoClients.create(settings);
    // Process request
    client.close();
}
```

모니터링할 중요한 CloudWatch 지표는 다음과 같습니다.

- `DatabaseConnections` - 1분 간격으로 인스턴스에서 열린 연결 수(활성 및 유휴)입니다.

- `DatabaseConnectionsMax` - 1분 동안 인스턴스에서 열린 데이터베이스 연결(활성 및 유휴)의 최대 수입니다.
- `DatabaseConnectionsLimit` - 지정된 시간에 인스턴스에 허용되는 최대 동시 데이터베이스 연결 수(활성 및 유휴)입니다.
- `LowMemNumOperationsThrottled` - 1분 동안 사용 가능한 메모리 부족으로 인해 제한된 요청 수입니다.

인스턴스 클래스당 할당량은 섹션을 참조하세요인스턴스 할당량.

애플리케이션 수준에서 연결 풀 문제의 일반적인 경고 신호는 다음과 같습니다.

- 연결 획득 시간 증가
- 대기열 크기 증가
- 제한 시간 예외 수 증가

인증 오버로드

Amazon DocumentDB에 연결하는 방법은 다음과 비슷합니다.

연결 요청 → SSL 핸드셰이크 → 인증 → 세션 생성 → 연결 준비 완료

>1,000개의 새 연결을 처리할 때 SSL 핸드셰이크를 완료한 후 추가 연결 요청이 인증을 위한 대기열에 들어갑니다. 이러한 오버로드 이벤트 중에는 애플리케이션의 평균 연결 시간이 증가합니다.

모니터링할 중요한 CloudWatch 지표는 다음과 같습니다.

- `DatabaseConnections` - 1분 간격으로 인스턴스에서 열린 연결 수(활성 및 유휴)입니다.
- `DatabaseConnectionsMax` - 1분 동안 인스턴스에서 열린 데이터베이스 연결(활성 및 유휴)의 최대 수입니다.
- `DatabaseConnectionsLimit` - 지정된 시간에 인스턴스에 허용되는 최대 동시 데이터베이스 연결 수(활성 및 유휴)입니다.

구성 문제

가장 일반적인 구성 문제는 프라이빗 네트워크 환경에 액세스하지 않고 환경에서 프라이빗 Amazon DocumentDB 클러스터 엔드포인트에 연결하려고 할 때 발생합니다. Amazon DocumentDB는 Virtual Private Cloud(VPC) 전용이며 현재 퍼블릭 엔드포인트를 지원하지 않습니다. 노트북 또는 VPC 외부의 로컬 개발 환경에서 Amazon DocumentDB 클러스터에 직접 연결할 수 없습니다.

이는 다음과 같은 오류로 표시됩니다.

```
Error: couldn't connect to server...
Failed to connect to...
exception: connect failed
connection attempt failed
```

잘못된 보안 그룹 구성으로 인해 연결 실패가 발생할 수도 있습니다. Amazon DocumentDB 클러스터는 기본적으로 TCP 포트 27017에서 연결을 수신합니다. 클러스터가 배포된 포트와 다른 포트에 연결하려고 하거나 애플리케이션이 클러스터의 수신 보안 그룹 구성에서 다루지 않는 경우 애플리케이션이 실패합니다.

인증서 관리가 잘못되면 연결 문제가 발생할 수도 있습니다. 기본적으로 전송 중인 암호화는 새로 생성된 Amazon DocumentDB 클러스터에 대해 활성화됩니다. 전송 중 암호화가 활성화된 경우 global-bundle.pem 인증서를 사용하여 클러스터에 연결하려면 TLS를 사용하는 보안 연결이 필요합니다. 잘못된 인증서를 사용하려고 하면 다음과 같은 오류가 발생합니다.

```
unable to get local issuer certificate
```

TLS 파라미터를 지정하지 않고 TLS가 활성화된 클러스터에 연결하려고 하면 다음과 같은 오류가 발생합니다.

```
Server selection timed out after 30000 ms
```

해결 - 문제 해결

연결 풀: 워크로드 요구 사항에 맞게 풀 크기를 구현하거나 조정하여 연결 풀링을 검토합니다. 최적의 풀 구성은 워크로드 및 요구 사항에 따라 달라집니다. 코어 연결을 준비하고 사용할 수 있도록 minPoolSize를 유지하고 풀이 소진된 경우 빠르게 실패할 수 있을 만큼 maxWaitTime을 짧게 유지해야 합니다.

다음은 매번 새 풀을 생성하지 않고 단일 풀을 재사용하는 방법의 예입니다.

```
MongoClient client = MongoClients.create(settings);
    for(Request request : requests) {
    // Process request
```

```
}
```

인증 오버로드: 점진적 연결 확장을 구현하고 새 연결을 한 번에 1,000개로 제한하여 인증을 관리합니다. 연결 풀링을 사용하여 인증된 연결을 효과적으로 재사용합니다. Amazon DocumentDB 클러스터에 연결이 과부하되지 않도록 하려면 연결 램프 업 전략을 구현합니다.

```
public class ConnectionManager {
    private static final int BATCH_SIZE = 100;
    private static final int DELAY_MS = 1000;

    public void establishConnections(int totalRequired) {
        int established = 0;
        while (established < totalRequired) {
            int batch = Math.min(BATCH_SIZE, totalRequired - established);
            createConnections(batch);
            Thread.sleep(DELAY_MS);
            established += batch;
        }
    }
}
```

허용되는 총 연결 수를 제한하도록 연결 풀 설정을 구성할 수도 있습니다.

```
MongoClientSettings settings = MongoClientSettings.builder()
    .applyToConnectionPoolSettings(builder -> {
        builder.maxSize(500)                    // Limit total connections
               .minSize(10)                     // Maintain base connections
               .maxConnectionLifeTime(3600000)  // Rotate connections hourly
    })
    .applyToServerSettings(builder -> {
        builder.heartbeatFrequency(10000)       // Regular server checks
    })
    .build();
```

구성 문제: 애플리케이션이 Amazon DocumentDB 리소스가 있는 프라이빗 VPC 및 서브넷에 액세스할 수 있는지 확인합니다. VPC 피어링을 사용하는 경우 자세한 내용은 개발자 안내서 VPC 피어링 연결

문제 해결을 참조하세요. 지식 센터 문서인 인터넷에서 VPC 내 Amazon EC2 인스턴스로의 연결 문제를 해결하려면 어떻게 해야 합니까?를 검토할 수도 있습니다.

보안 그룹 구성의 경우 애플리케이션에서 연결을 허용하려면 Amazon DocumentDB 보안 그룹에 수신 규칙을 포함해야 합니다.

```
{
  "SecurityGroupIngress": [
    {
      "IpProtocol": "tcp",
      "FromPort": 27017,
      "ToPort": 27017,
      "SourceSecurityGroupId": "<application-security-group>",
      "Description": "DocumentDB access from application tier"
    }
  ],
  "SecurityGroupEgress": [
    {
      "IpProtocol": "-1",
      "FromPort": -1,
      "ToPort": -1,
      "CidrIp": "0.0.0.0/0"
    }
  ]
}
```

클러스터가 TLS 암호화로 구성된 경우 global-bundle.pem이라는 Amazon DocumentDB용 TLS 인증서를 다운로드하여 클러스터에 연결할 때 사용합니다.

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

장기 솔루션

인스턴스 확장은 더 큰 인스턴스 클래스로 업그레이드하거나 읽기 전용 복제본을 추가하여 연결 로드를 분산하는 방식으로 필요할 수 있습니다. 적절한 로드 밸런싱 구현은 클러스터 전체에서 최적의 리소스 사용률을 보장합니다.

애플리케이션 변경은 강력한 연결 처리, 포괄적인 모니터링 및 연결 폴링 모범 사례 준수를 구현하는 데 중점을 두어야 합니다. 여기에는 적절한 오류 처리 및 연결 수명 주기 관리가 포함됩니다.

아키텍처 개선에는 가변 워크로드를 위한 Amazon DocumentDB Serverless 채택, 정교한 재시도 로직 구현, 내결함성을 위한 설계가 포함될 수 있습니다. 연결 관리를 더 잘 처리하려면 애플리케이션 아키텍처를 재구성하는 것이 좋습니다.

모범 사례

연결 풀

적절한 연결 풀 관리 및 모니터링을 통해 애플리케이션은 안정적인 데이터베이스 연결을 유지하면서 시스템 신뢰성 및 성능에 영향을 미칠 수 있는 소진 시나리오를 방지할 수 있습니다. 워크로드의 특성에 따라 적절한 제한 시간을 구성하고 풀 크기를 조정합니다.

연결 풀 설정 예제

```
MongoClientSettings settings = MongoClientSettings.builder()
    .applyToConnectionPoolSettings(builder ->
        builder.maxSize(10))
    .applyToConnectionPoolSettings(builder ->
        builder.maxWaitQueueSize(2))
    .applyToConnectionPoolSettings(builder ->
        builder.maxConnectionIdleTime(10, TimeUnit.MINUTES))
    .build();
```

자세한 내용은 Amazon DocumentDB로 복원력이 뛰어난 애플리케이션 구축 - 1부: 클라이언트 구성을 참조하세요.

인증 오버로드

항상 워크로드에 따라 파라미터에 적절한 값을 사용하여 연결 풀링을 구현합니다. 점진적인 연결 설정 기법을 사용하고 가능한 경우 지속적인 연결을 유지합니다. 유휴 리소스가 낭비되지 않도록 적절한 연결 정리를 구현합니다.

구성 문제

애플리케이션에서 Amazon DocumentDB 리소스로 적절한 라우팅을 구성했는지 확인합니다. 전송 중 암호화에 TLS를 활용하고 최소 권한 액세스를 구현합니다. Amazon DocumentDB 자격 증명을 확인하고 연결 문자열 값을 검증합니다.

쿼리 실행 속도가 느림

식별 - 문제 발견

느리게 실행되는 쿼리는 비즈니스 요구 사항에 필요한 실행 시간을 초과하는 쿼리입니다. 느린 쿼리를 구성하는 항목의 정의는 워크로드마다 다릅니다. 프로파일러 로그 또는 성능 개선 도우미를 통해 느린 쿼리를 식별할 수 있습니다.

1. 아직 활성화되지 않은 경우 프로파일러를 활성화합니다. 프로파일러 로그는 로그 그룹:에서 Amazon CloudWatch에 기록됩니다`/aws/docdb/<cluster-identifier>/profiler`. CloudWatch Logs Insights를 사용하여 쿼리합니다.

   ecommerce.products 컬렉션에 대한 가장 느린 쿼리 10개를 가져오는 CloudWatch 로그 분석 쿼리의 예:

   ```
   filter ns="ecommerce.products" | sort millis desc | limit 10
   ```

2. 성능 개선 도우미를 사용하여 거의 실시간으로 비용이 많이 드는 쿼리를 식별할 수 있습니다. 아직 활성화되지 않은 경우 성능 개선 도우미를 활성화합니다.
   a. AWS 콘솔을 열고 Amazon DocumentDB로 이동한 다음 성능 개선 도우미로 이동하여 클러스터 인스턴스를 선택합니다.
   b. DB 로드(AAS) 타임라인과 상위 쿼리(DB 로드 기준)를 검토합니다. 쿼리 다이제스트를 확장하여 리터럴 하위 문을 봅니다.
   c. 분석이 필요한 쿼리를 캡처합니다.

   > **Note**
   >
   > 성능 개선 도우미의 모든 쿼리가 비효율적이거나 느린 쿼리는 아닙니다.

조사 - 정보 수집

1. 프로파일러는 밀리스, 반환 및 planSummary(인덱스 사용량)를 포함하여 쿼리 실행 계획과 이와 관련된 주요 지표를 제공합니다.

   ```
   {
   ```

```
    "op": "query",
    "ts": 1721374275673,
    "ns": "test.perf",
    "command": {
        "find": "perf",
        "filter": {
            "threadRunCount": 0
        },
        "$db": "test",
        "lsid": {
            "id": {
                "$binary": "oO2wEtpgQIK+y9KGByYnsw==",
                "$type": "4"
            }
        },
        "$readPreference": {
            "mode": "secondaryPreferred"
        }
    },
    "cursorExhausted": true,
    "nreturned": 0,
    "responseLength": 0,
    "protocol": "op_query",
    "millis": 137,
    "planSummary": "IXSCAN",
    "execStats": {
        "stage": "FETCH",
        "nReturned": "0",
        "executionTimeMillisEstimate": "100.346",
        "inputStage": {
            "stage": "IXSCAN",
            "nReturned": "0",
            "executionTimeMillisEstimate": "100.342",
            "indexName": "threadRunCount_1"
        }
    },
    "client": "172.31.6.165:43154",
    "appName": "ProdAppTester14",
    "user": "adminuser"
}
```

COLLSCAN 쿼리를 찾으려면:

```
filter planSummary="COLLSCAN" | sort millis desc | limit 20
```

2. 성능 개선 도우미를 사용하여 대기, 로드 및 리소스 영향(예: 쿼리당 IOPS 또는 CPU)과 같은 쿼리 실행 추세를 실시간으로 분석할 수 있습니다.

   성능 개선 도우미는 쿼리 실행 계획을 제공하지 않으므로 쿼리를 캡처하고 Amazon DocumentDB 클러스터의 쿼리에 대한 설명("executionStats")을 실행합니다.

```
db.ecommerce.products.find().explain("executionStats")
```

3. 선택적으로 프로파일러 지표를 성능 개선 도우미 데이터와 연관시킵니다(예: 프로파일러의 상위 쿼리와 성능 개선 도우미의 상위 쿼리 일치).

진단 - 근본 원인 찾기

이 단계에서는 쿼리 계획을 진단하여 잠재적 최적화를 식별합니다. 다음 흐름 사용 - 증상 → 가능한 원인:

- 증상: planSummary: "COLLSCAN"

  원인: 인덱스가 누락되었거나 잘못되었습니다.
- 증상: $group / $sort에서 집계 속도가 느림

  원인: 집계 파이프라인이 메모리에서 너무 많은 데이터를 처리하고 있을 수 있습니다.
- 증상: 인덱스가 사용되지만 성능 개선 도우미는 IO 대기별로 그룹화된 DB 로드를 표시하는 동안 지연 시간이 깁니다(planSummary: "IXSCAN").

  원인: I/O 바인딩된 쿼리, 인덱스 또는 작업 세트가 인스턴스에서 사용 가능한 버퍼 캐시를 초과합니다.
- 증상: PI는 CPU 대기 상태, 쿼리가 적기 때문에 높은 AAS를 표시합니다.

  원인: CPU 바운드 쿼리(복잡한 집계, $regex).
- 증상: 느린 쿼리가 많지만 비용이 많이 드는 planSummary를 보여주는 쿼리가 없음

  원인: 외부 병목 현상(네트워크, 제한, 유지 관리 작업, 쿼리 볼륨).

해결 - 문제 해결

수정 사항을 구현할 때는 항상 explain("executionStats")로 확인하고 성능 개선 도우미 DB 로드를 모니터링합니다.

1. planSummary: "COLLSCAN"
   - 대상 인덱스를 생성합니다.

     예: { 범주, 가격 }을 기준으로 필터링하고 가격 내림차순으로 정렬하는 빈번한 쿼리의 경우:

     ```
     db.products.createIndex({ category: 1, price: -1 })
     ```

2. $group/$sort로 집계 속도가 느림
   - $match 및 $project를 조기에 푸시하여 $group/$sort로 흐르는 문서를 줄입니다.
   - 파이프라인 초기에 필드 수를 제한합니다.

     ```
     db.products.explain("executionStats").aggregate([
       { $match: { category: "Electronics", price: { $gt: 0 } } },   // early filter
       { $project: { price: 1, category: 1 } },                     // reduce document
      size
       { $group: { _id: "$category", avgPrice: { $avg: "$price" } } }
     ])
     ```

3. 성능 개선 도우미가 IO 대기별로 그룹화된 DB 로드를 표시하는 동안 지연 시간이 높음
   - BufferCacheHitRatio가 낮은지 또는 인스턴스의 ReadIOPS가 높은지 확인합니다. 인스턴스 메모리를 늘리거나(인스턴스 클래스 업그레이드, 예: r6g.large → r6g.xlarge) NVMe 인스턴스 클래스를 사용합니다.
   - 인덱스 공간을 줄입니다.
   - 읽기 전용 복제본을 추가하여 읽기 트래픽을 오프로드합니다(쿼리가 최종 일관성으로 허용 가능한 경우 readPreference 설정을 사용하여 복제본의 트래픽을 리디렉션).
4. PI는 CPU 대기 상태, 쿼리가 적기 때문에 높은 AAS를 표시합니다.
   - 값비싼 $regex를 인덱싱된 접두사 검색 또는 $text 인덱스로 바꿉니다.
   - 배치 쓰기(삽입)를 통해 쓰기 증폭을 줄입니다.

> **Note**
>
> 변경 사항을 프로덕션으로 승격하기 전에 항상 하위 환경(비프로덕션)에서 변경 사항을 테스트합니다.

안티 패턴

이 섹션에서는 Amazon DocumentDB 성능에 부정적인 영향을 미칠 수 있는 일반적인 안티 패턴을 다룹니다.

주제

- 속성이 3개 이상인 복합 인덱스
- 장기 실행 쿼리
- 낮은 사용 및 중복 인덱스
- 대형 어레이가 있는 다중 키 인덱스

속성이 3개 이상인 복합 인덱스

개요

복합 또는 복합 인덱스는 단일 인덱스 구조 내에서 여러 필드에 대한 참조를 유지합니다. 이러한 인덱스는 여러 필드를 동시에 필터링하거나 필터링을 정렬 작업과 결합하는 쿼리의 성능을 최적화합니다. 또한 가장 왼쪽에 인덱싱된 필드의 단일 조건 쿼리에도 효과적입니다. 데이터베이스는 이러한 인덱스 항목을 활용하여 전체 컬렉션 스캔을 수행하지 않고도 일치하는 문서를 효율적으로 찾습니다.

Amazon DocumentDB는 복합 인덱스를 사용하여 인덱스 접두사라고 하는 개념인 인덱싱된 필드의 선행 하위 집합을 포함하는 쿼리를 지원할 수 있습니다. 예를 들어에 복합 인덱스가 있는 경우 `{state: 1, city: 1, zipcode: 1}` Amazon DocumentDB는 'state' 필드만 사용하거나, 'state' 및 'city' 필드를 함께 사용하거나, 'state', 'city' 및 'zipcode' 세 필드를 모두 사용하는 쿼리를 효율적으로 처리할 수 있습니다. 그러나 쿼리는 그 사이에 필드를 건너뛰지 않고 왼쪽에서 오른쪽으로 필드를 사용해야 합니다. 즉, 'city' 또는 'zipcode' 필드만 사용하거나 'state' 및 'zipcode'('city' 건너뛰기)와 같은 조합을 사용하는 쿼리는 인덱스를 완전히 활용할 수 없습니다.

대부분의 실제 시나리오에서 속성이 3개 이하인 복합 인덱스는 일반적으로 최적의 성능과 리소스 효율성을 달성합니다. 복합 인덱스에 속성이 3개 이상 있을 수 있지만 리소스 소비가 증가하여 이점을 능가하는 경우가 많습니다.

클러스터에 미치는 영향

이론적 성능을 극대화하기 위해 쿼리의 모든 조건을 포함하는 인덱스를 생성하려고 하지만 대부분의 데이터 필터링은 복합 인덱스의 처음 1~3개의 속성을 통해 이루어집니다. 이 임계값을 초과하는 추가 필드는 의미 있는 쿼리 최적화보다는 주로 인덱스 크기에 영향을 미칩니다.

스토리지 및 I/O 오버헤드: 속성이 많은 복합 인덱스는 더 간단한 대안보다 훨씬 더 많은 스토리지 공간을 소비합니다. 크기는 인덱싱된 속성 수와 인덱싱된 값 자체의 크기에 정비례합니다.

메모리 공간: 속성이 많은 복합 인덱스의 스토리지 공간이 크면 해당 메모리 공간이 생성되므로 작업 세트가 더 커지고 버퍼 풀에서 자주 액세스하는 다른 데이터가 대체됩니다.

쓰기 작업: 여러 인덱싱된 필드에 영향을 미치는 각 문서를 수정하려면 전체 복합 인덱스 항목을 업데이트하고 쓰기 작업을 완료하는 데 필요한 작업을 곱해야 합니다.

식별 방법

먼저 컬렉션의 모든 인덱스를 검토하여 속성이 3개 이상인 복합 인덱스를 식별합니다.

```
// List all indexes for the collection
db.collection.getIndexes()

// Look for indexes with 3+ fields like:

// { "userId": 1, "status": 1, "category": 1, "priority": 1, "region": 1 }
// { "orderId": 1, "customerId": 1, "productId": 1, "timestamp": 1, "warehouse": 1 }
```

문제 해결

속성이 3개 이상인 복합 인덱스를 사용하는 경우 이를 활용하는 쿼리를 식별하고 최적화 기회를 찾습니다. 이러한 인덱스를 3개 이하의 속성을 포함하는 보다 효율적인 인덱스로 대체하는 것이 좋습니다. 새 인덱스를 구현한 후 속성이 3개 이상인 이전 인덱스를 제거합니다.

> **Note**
>
> 인덱스를 삭제하기 전에 항상 이해관계자와 조정하고 성능 영향을 검증합니다.

복합 인덱스를 생성할 때 Equality, Sort, Range(ESR) 규칙을 따릅니다.

ESR(Equal Sort Range) 규칙:이 순서 지정은 먼저 등식 조건으로 데이터 세트를 필터링한 다음 축소된 세트에 정렬 작업을 적용하고 마지막으로 가능한 가장 작은 하위 집합에 대해 범위 스캔을 수행하여 인덱스 효율성을 극대화합니다.

- 필드를 평등(정확한 일치)으로 정렬,
- 정렬(정렬),
- 범위(>, <, $in).

```
// Query pattern
db.orders.find({
  userId: "user123",                    // Equality
  price: { $gte: 50, $lte: 200 }      // Range
}).sort({ createdAt: -1 })             // Sort

// Optimal index following ESR rule
db.orders.createIndex({ userId: 1, createdAt: -1, price: 1 })
//                      Equality   Sort         Range
```

장기 실행 쿼리

개요

장기 실행 쿼리는 클러스터 전반의 성능 문제로 이어질 수 있습니다. 이러한 쿼리는 Amazon DocumentDB의 다중 버전 동시성 제어(MVCC) 가비지 수집 프로세스를 방해하여 이전 버전의 문서가 누적되고 클러스터 전체에서 성능이 저하될 수 있습니다. Amazon DocumentDB는 런어웨이 쿼리가 리소스를 무기한 소비하는 것을 제한하는 안전 메커니즘으로 2시간의 서버 측 제한 시간을 구현합니다.

장기 실행 쿼리란 무엇입니까?

- 장기간(일반적으로 > 30분) 실행되는 쿼리
- 몇 시간 동안 활성 상태로 유지되는 열린 커서(커서가 활성 상태인 경우 2시간 제한 시간이 적용되지 않음)

클러스터에 미치는 영향

장기 실행 쿼리는 가비지 수집 프로세스를 방해할 수 있습니다.

- 이전 버전 누적: 가비지 수집기가 이전 문서 버전을 회수할 수 없음
- 컬렉션 및 인덱스 팽창: 컬렉션 및 인덱스 항목은 시간이 지남에 따라 누적되고 팽창이 증가하여 스토리지 비용이 증가할 수 있습니다.
- CPU 및 메모리 압력: 증가된 수의 이전 문서 버전, 인덱스 항목 및 트랜잭션 IDs를 비효율적으로 처리하여 CPU 및 메모리 압력이 증가합니다.

장기 실행 쿼리 → GC 차단 → 스토리지 증가 → CPU 및 메모리 압력 → 더 긴 쿼리

모니터링 및 감지

1: 장기 실행 쿼리를 찾으려면 currentOp 명령을 사용합니다.

```
// To find a query running for more than 30 mins
db.adminCommand({
    aggregate: 1,
    pipeline: [
        {$currentOp: {}},
        {$match:
            {$or:
                [{secs_running: {$gt: 1800}},
                 {WaitState: {$exists: true}}]}}],
    cursor: {}
});
```

2. 30분 이상 활성 상태인 커서를 찾으려면

```
// To find cursor which is running more than 30 mins
db.adminCommand({
    "currentOp": true,
    "active": true,
    "$all": true
}).inprog.filter(function(op) {
    return op.desc == "Cursor" &&
           op.secs_running > 1800 &&
           op.active == true;
}).sort((a, b) => b.microsecs_running - a.microsecs_running)
```

3. CloudWatch를 통한 가비지 수집기 진행 상황 모니터링

`LongestRunningGCProcess`- 가장 긴 활성 가비지 수집 프로세스의 초 단위 기간입니다. 1분마다 업데이트하고 1분 내에 완료되는 프로세스를 제외한 활성 작업만 추적합니다.

`AvailableMVCCIds` -0에 도달하기 전에 사용 가능한 나머지 쓰기 작업 수를 보여주는 카운터입니다. 이 카운터가 0에 도달하면 ID가 회수되고 재활용될 때까지 클러스터가 읽기 전용 모드로 전환됩니다. 카운터는 각 쓰기 작업에 따라 감소하고 폐영역 회수가 이전 MVCC ID를 재활용함에 따라 증가합니다.

> **Note**
>
> MVCC IDs와 가비지 수집 기간 연장은 장기 실행 쿼리에만 국한되지 않습니다. 리소스가 제한된 인스턴스에서 쓰기 집약적인 워크로드를 수행하면 MVCC ID 가용성이 감소하고 가비지 수집 주기가 길어질 수도 있습니다.

문제 해결 전략

- 애플리케이션에서 쿼리 제한 시간 구현
- 커서를 더 오랜 시간 동안 그대로 두지 마세요.
- 더 나은 성능을 위해 쿼리를 최적화합니다.
- 쓰기 작업 일괄 처리 선호

낮은 사용 및 중복 인덱스

개요

인덱스는 워크로드의 전체 쿼리 성능을 개선하지만 비용이 발생합니다. 각 인덱스는 스토리지 및 I/O 리소스를 사용하며, 쓰기 작업은 컬렉션의 모든 인덱스를 업데이트해야 합니다. 데이터베이스에 사용되지 않거나 사용률이 낮거나 중복된 인덱스가 포함되어 있는 경우 클러스터에 오버헤드가 발생하여 성능이 저하됩니다. 여러 개의 불필요한 인덱스는 성능 문제, 스토리지 비용 증가 및 운영 오버헤드 증가를 유발하는 안티 패턴을 생성합니다.

하위 최적 인덱싱 시나리오

- 미사용 인덱스: 일회성 쿼리 또는 현재 쿼리 패턴에서 더 이상 액세스하지 않는 이전 제품 반복에 대해 생성된 인덱스입니다. 컬렉션에서 미사용 인덱스를 식별하기 위해 indexStats를 사용할 수 있습니

다. 자세한 내용은 the section called “인덱스 사용량을 분석하고 사용하지 않는 인덱스를 식별” 단원을 참조하십시오.

- 중복 인덱스: 중복되는 키 패턴 또는 동일한 쿼리 패턴을 포함하는 여러 인덱스입니다. 예: 복합 인덱스가 속성 A에서만 쿼리를 처리할 수 있으므로 복합 인덱스가 속성 A와 B를 모두 포함하는 경우 속성 A의 단일 키 인덱스가 중복됩니다.
- 오버인덱싱: 실제 쿼리 패턴 또는 성능 요구 사항을 분석하지 않고 “경우에 따라” 인덱스를 생성합니다.
- 낮은 카디널리티 인덱스: 쿼리 최적화 이점을 최소화하는 고유한 값이 거의 없는 필드(예: 부울 필드, 상태 플래그)의 인덱스입니다.

클러스터 성능에 미치는 영향

- 스토리지 및 IO: 각 인덱스는 스토리지 공간과 IO 리소스를 사용하므로 전체 비용이 발생합니다. 인덱스가 사용되지 않거나 중복되는 경우 클러스터의 불필요한 비용을 나타냅니다.
- 성능 저하: 삽입, 업데이트 및 삭제 작업은 인덱스를 유지하여 리소스를 소비하는 오버헤드를 발생시켜야 합니다. 사용하지 않거나 중복된 인덱스는 이러한 작업에 불필요한 오버헤드를 추가합니다.
- 메모리 및 CPU 압력: 인덱스는 버퍼 풀 메모리에 대해 경쟁하며 작업 세트가 인스턴스 메모리에 맞지 않으면 자주 액세스하는 데이터를 제거하여 최적이 아닌 BufferCacheHitRatio를 생성할 수 있습니다. 또한 시스템은 인덱스 구조를 유지 관리하는 CPU 리소스를 사용합니다.

인덱스 최적화 기회를 식별하는 데 사용할 수 있는 도구

- 인덱스 검토 도구:
- 인덱스 카디널리티 감지 도구

도구 키트 수집:

```
git clone https://github.com/awslabs/amazon-documentdb-tools.git
```

1: 인덱스 검토 도구

인덱스 검토 도구는 모든 컬렉션과 인덱스를 분석하여 인덱스 목록과 사용 패턴을 제공하고 Amazon DocumentDB 클러스터에서 중복 인덱스를 식별합니다. 포괄적인 분석을 위해 모든 클러스터 인스턴스에서이 도구를 실행합니다. 자세한 내용은 인덱스 검토 도구를 참조하세요.

사용량:

```
cd performance/index-review/

python3 index-review.py --server-alias <server-alias> --uri <mongodb-uri>
```

출력 파일

- `{server-alias}-collections.csv`: 컬렉션 수준 통계
- `{server-alias}-indexes.csv`: 세부 인덱스 사용량 지표
- `{server-alias}-{timestamp}-index-review.json`: 추가 분석을 위한 원시 데이터

> **Note**
>
> 독립 별칭을 사용하여 클러스터의 각 인스턴스에 대해를 실행하여 각 인스턴스에서 통계를 생성한 다음 도구를 실행하여 누적 분석을 수행합니다.

예: `python3 index-review.py --files "{server-alias}-{timestamp}-index-review.json, {server-alias}-{timestamp}-index-review.json" --server-alias full-review`

2. 지표 카디널리티 감지 도구

이 도구는 쿼리 성능에 비효율적인 카디널리티가 낮은 인덱스를 식별합니다.

사용량

```
cd performance/index-cardinality-detection/

python3 detect-cardinality.py --uri <mongodb-uri>
```

파라미터

- `--threshold`: 카디널리티 임계값 백분율(기본값: 1%)
- `--sample-count`: 샘플링할 문서 수(기본값: 100,000)
- `--max-collections`: 데이터베이스당 스캔할 최대 컬렉션 수(기본값: 100)

자세한 내용은 https://aws.amazon.com/blogs/database/detect-and-fix-low-cardinality-indexes-in-amazon-documentdb/ 참조하십시오.

문제 해결 전략

1: 미사용 및 중복 인덱스 제거

```
// Example: Drop unused index
db.collection.dropIndex("unused_index_name")
```

> **Note**
>
> 모든 이해관계자와 상의하고 성능을 테스트하지 않고 인덱스를 삭제해서는 안 됩니다.

2. 낮은 카디널리티 인덱스 최적화

- 필터와 함께 부분 인덱스 사용
- 단일 낮은 카디널리티 인덱스를 복합 인덱스로 변환

대형 어레이가 있는 다중 키 인덱스

개요

Amazon DocumentDB의 다중 키 인덱스를 사용하면 배열 필드를 효율적으로 쿼리할 수 있습니다. 이 인덱싱 전략은 배열 데이터 내에서 검색할 때 쿼리 지연 시간을 크게 줄여 데이터 구조가 복잡한 애플리케이션에 매력적인 옵션입니다.

그러나 컬렉션 내에서 대규모 배열 또는 여러 배열 필드로 작업할 때는 리소스 영향과 성능 특성을 이해하여 정보에 입각한 아키텍처 결정을 내리는 것이 중요합니다. 배열 필드에 인덱스를 생성하면 Amazon DocumentDB는 배열의 각 요소에 대해 개별 인덱스 항목을 생성합니다.

클러스터에 미치는 영향

- 스토리지 및 IO 오버헤드: 다중 키 인덱스는 경우에 따라 기본 테이블의 최대 여러 번 스토리지 공간을 사용할 수 있으며, 크기는 인덱싱된 배열 속성, 배열의 요소 수 및 요소의 크기가 있는 컬렉션의 문서에 정비례합니다.

- 메모리 사용량: 스토리지 공간이 비교적 크므로 이러한 인덱스는 메모리에 해당 공간이 있으므로 작업 세트가 더 커집니다.
- 쓰기 작업: 다중 키 인덱스는 쓰기 작업 중에 오버헤드를 생성합니다. 각 배열 요소는 별도의 인덱스 항목을 생성하여 쓰기 작업을 완료하는 데 필요한 작업을 곱합니다.

해결 전략

- 필요한 경우 다중 키 인덱스를 생성하고 배열의 필드 수를 제한합니다.
- 컬렉션의 다중 키 인덱스 수 제한
- 데이터 모델 수정 고려

성능 개선 팁

이 섹션에서는 Amazon DocumentDB가 애플리케이션 효율성과 쿼리 실행을 개선하기 위한 5가지 성능 최적화 전략을 제공합니다.

1. 집계 파이프라인에서 $match를 첫 번째 단계로 사용

성능을 극대화하려면 항상 $match를 집계 파이프라인에서 필터링하는 첫 번째 단계로 배치하세요. Amazon DocumentDB는 $match가 파이프라인을 리드할 때 인덱스를 효과적으로 활용하므로 데이터베이스가 데이터를 조기에 필터링하고 처리 오버헤드를 줄일 수 있습니다.

```
// Optimized approach
db.orders.aggregate([
  { $match: { status: "active", category: "electronics" } }, // Index utilization
  { $group: { _id: "$category", total: { $sum: "$price" } } },
  { $sort: { total: -1 } }
])
```

영향: 조기 필터링은 후속 파이프라인 단계에서 처리되는 문서 수를 줄여 쿼리 실행 속도를 높이고 리소스 소비를 줄입니다.

2. 집계 파이프라인에서 $project를 사용하여 파이프라인 데이터 크기 최소화

집계 파이프라인 단계를 통해 필수 필드만 가져와 데이터 크기를 최소화하고 성능을 개선합니다. $project를 전략적으로 사용하여 필요한 데이터만 포함합니다.

```
// Efficient pipeline design
db.orders.aggregate([
  { $match: { orderDate: { $gte: new Date("2024-01-01") } } },
  { $project: { customerId: 1, totalAmount: 1, status: 1 } }, // Only needed fields
  { $group: { _id: "$customerId", totalSpent: { $sum: "$totalAmount" } } }
])
```

영향: 문서가 작을수록 메모리 사용량이 줄어들고 파이프라인 처리 효율성이 향상되어 전반적인 쿼리 성능이 향상됩니다.

3. 스토리지 비용, I/O 비용 및 쿼리 성능 향상을 위한 문서 압축 활성화

클러스터 파라미터 그룹에서 문서 압축을 활성화하여 스토리지 비용, I/O 비용을 절감하고 쿼리 성능을 높입니다. Amazon DocumentDB는 압축된 문서를 디스크와 RAM에 저장하여 메모리 공간 및 I/O 비용을 줄입니다.

영향:

- 사용 가능한 메모리에 맞는 더 많은 문서
- 디스크 읽기 감소로 데이터 액세스 속도 향상
- 스토리지 비용 절감, I/O 비용 절감, 쿼리 성능 향상

> **Note**
>
> Amazon DocumentDB는 버전 5.0에서 기본적으로 압축을 활성화하지 않습니다. 5.0 클러스터에 대해 컬렉션 또는 클러스터 수준에서 압축을 활성화할 수 있습니다. Amazon DocumentDB의 압축 검토 유틸리티를 사용하여 컬렉션의 압축 비율을 분석할 수 있습니다.
> Amazon DocumentDB 8.0의 경우 압축이 기본적으로 활성화됩니다.

4. 최적의 쿼리 성능을 위한 인덱스 활용

최적의 성능을 위해 쿼리가 항상 인덱스를 활용하는지 확인합니다. Amazon DocumentDB는 다양한 사용 사례에 맞게 여러 인덱스 유형을 제공합니다.

인덱싱 원칙:

- 모든 쿼리는 적절한 인덱스를 활용해야 합니다.
- Amazon DocumentDB는 여러 인덱스 유형을 제공합니다.
- 복합 인덱스는 단일 인덱스로 다양한 쿼리 셰이프를 지원하여 유연성을 극대화합니다.
- 정렬 및 필터링 작업을 함께 지원하는 인덱스 설계

인덱스 접두사 이해: 복합 인덱스는 인덱스 접두사를 통해 작동합니다. Amazon DocumentDB는 인덱스 필드의 모든 left-to-right으로 하위 집합을 사용할 수 있습니다. 예를 들어 인덱스는 다음과 같은 사용 가능한 접두사를 `{ category: 1, price: -1, inStock: 1 }` 생성합니다.

- `{ category: 1 }` - 범주별 쿼리 필터링만 지원
- `{ category: 1, price: -1 }` - 범주별 쿼리 필터링 및 가격별 정렬/필터링 지원
- `{ category: 1, price: -1, inStock: 1 }` - 전체 복합 쿼리 지원

가격, inStock 또는 inStock에 대한 쿼리만 첫 번째 필드(범주)로 시작하지 않으므로이 인덱스를 사용하지 않습니다.

인덱스를 사용하지 않는 쿼리를 식별하는 방법: explain() 메서드를 사용하여 인덱스를 사용하는 대신 쿼리 실행을 분석하고 컬렉션 스캔을 수행하는 쿼리를 식별합니다.

영향: 인덱스 사용률이 없는 쿼리는 수집 스캔을 초래하여 인스턴스에 대한 메모리 및 CPU 압력 증가와 쿼리 지연 시간 증가를 초래합니다.

5. 쿼리 패턴을 기반으로 데이터 모델 최적화

애플리케이션 쿼리 및 업데이트 방식에 맞게 데이터 모델을 조정합니다. 데이터 모델링은 고성능 Amazon DocumentDB 애플리케이션의 기반입니다.

최적화 전략:

성능을 위한 임베딩

- 단위로 자주 액세스할 때 관련 데이터를 함께 저장
- 항상 함께 검색되는 문서 임베드
- one-to-few 관계에 적합

```
// Embedded approach for frequently accessed data
```

```
{
  _id: ObjectId("..."),
  customerName: "John Doe",
  address: {
    street: "123 Main St",
    city: "Seattle",
    zipCode: "98101"
  },
  recentOrders: [
    { orderId: "ORD001", amount: 99.99, date: "2024-01-15" }
  ]
}
```

유연성에 대한 참조

- 대규모 또는 자주 액세스하지 않는 데이터에 대한 참조 사용
- 대규모 데이터 세트와의 one-to-many 관계에 권장됨
- 문서 팽창을 방지하고 업데이트 성능을 개선합니다.

컬렉션 분할 전략

대용량 문서의 일부 필드만 자주 업데이트되거나 자주 액세스하지 않는 대용량 데이터 팽창 문서가 있는 경우 컬렉션을 분할하는 것이 좋습니다.

- 자주 업데이트되는 필드를 별도의 더 작은 컬렉션에 보관
- 정적 데이터 또는 자주 액세스하지 않는 데이터를 다른 컬렉션에 저장
- 필요할 때 참조와 연결

```
// Before: Large document with mixed access patterns
{
  _id: ObjectId("..."),
  productId: "PROD123",
  name: "Wireless Headphones",        // Frequently accessed
  price: 99.99,                       // Frequently accessed
  inventory: 45,                      // Updated frequently
  lastSold: "2024-01-15",            // Updated frequently
  detailedSpecs: { /* large object */ }, // Infrequently accessed
  manualPDF: "base64...",            // Large, rarely accessed
```

```
  reviewHistory: [/* large array */] // Infrequently accessed
}

// After: Split into collections based on access patterns
// products collection (frequently accessed data)
{
  _id: ObjectId("..."),
  productId: "PROD123",
  name: "Wireless Headphones",
  price: 99.99,
  inventory: 45,
  lastSold: "2024-01-15"
}

// product_details collection (infrequently accessed data)
{
  _id: ObjectId("..."),
  productId: "PROD123",           // Reference to products collection
  detailedSpecs: { /* large object */ },
  manualPDF: "base64...",
  reviewHistory: [/* large array */]
}
```

성능 향상: 문서가 작을수록 업데이트 속도가 빨라지고 메모리 사용량이 줄어들며 캐시 효율성이 향상됩니다.

영향: 비효율적인 데이터 모델링으로 인해 쿼리가 최적화되지 않고 문서 크기가 증가하며 메모리 사용량이 증가하여 애플리케이션 성능이 저하되고 운영 비용이 증가합니다.

쿼리 계획 분석

설명 계획을 통한 쿼리 계획 분석은 Amazon DocumentDB 쿼리 성능에 대한 필수 인사이트를 제공합니다. executionStats를 사용한 쿼리 계획은 다음을 포함한 주요 지표를 보여줍니다.

- 단계당 반환된 문서(nReturned)
- 단계별 실행 시간(executionTimeMillisEstimate)
- 계획 생성 기간(planningTimeMillis)

쿼리 계획 출력을 검사하여 개발자는 실행 패턴을 분석하고, 인덱스 사용률을 평가하고, 쿼리 파이프라인 단계에서 잠재적 최적화 기회를 식별할 수 있습니다.

쿼리 계획을 분석하려면 다음 형식으로 explain() 명령을 사용할 수 있습니다.

```
db.runCommand({explain: {query document}, verbosity: "executionStats"})
db.collection.find().explain("executionStats");
```

다음은 예제 작업입니다.

```
db.collection.find({ companyname: { '$eq': 'ANYCOMPANY' }, isDeleted: { '$eq':
 false } }).sort({"createdAt":1}).limit(2).explain("executionStats");
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
  queryPlanner: {
    plannerVersion: 2,
    namespace: 'limit_test.test',
    winningPlan: {
      stage: 'LIMIT_SKIP',
      inputStage: {
        stage: 'SORT',
        sortPattern: { createdAt: 1 },
        inputStage: {
          stage: 'IXSCAN',
          indexName: 'companyname_1_createdAt_1_isDeleted_1',
          direction: 'forward',
          indexCond: {
            '$and': [
              { companyname: { '$eq': 'ANYCOMPANY' } },
              { isDeleted: { '$eq': false } }
            ]
          }
        }
      }
    }
  },
  indexFilterSet: false,
  indexFilterApplied: false,
```

```
  executionStats: {
    executionSuccess: true,
    executionTimeMillis: '4.186',
    planningTimeMillis: '3.909',
    executionStages: {
      stage: 'LIMIT_SKIP',
      nReturned: '2',
      executionTimeMillisEstimate: '0.199',
      inputStage: {
        stage: 'SORT',
        nReturned: '2',
        executionTimeMillisEstimate: '0.197',
        sortPattern: { createdAt: 1 },
        inputStage: {
          stage: 'IXSCAN',
          nReturned: '34',
          executionTimeMillisEstimate: '0.151',
          indexName: 'companyname_1_createdAt_1_isDeleted_1',
          direction: 'forward',
          indexCond: {
            '$and': [
              { companyname: { '$eq': 'ANYCOMPANY' } },
              { isDeleted: { '$eq': false } }
            ]
          }
        }
      }
    }
  },
  serverInfo: { host: 'demo-cluster', port: 27017, version: '5.0.0' },
  ok: 1,
  operationTime: Timestamp({ t: 1759915116, i: 1 })
}
```

다음은 Amazon DocumentDB 쿼리 실행 계획을 자세히 분석하여 각 구성 요소와 성능 특성을 분석한 것입니다.

전체 타이밍

executionTimeMillis는 계획 시간을 포함하여 쿼리에 걸린 총 시간을 나타냅니다.

planningTimeMillis는 쿼리에 소요된 총 계획 시간을 나타냅니다.

실행 단계

Amazon DocumentDB가 쿼리를 실행하는 데 사용하는 step-by-step 프로세스를 설명하여 다양한 작업을 통해 데이터가 흐르는 방식을 보여줍니다.

```
"executionStages": {
    "stage": "[STAGE_NAME]",
    "nReturned": "[NUMBER_OF_DOCS]",
    "executionTimeMillisEstimate": "[TIME]",
    "inputStage": {
        // Nested stages
    }
}
```

쿼리 계획의 공통 단계

다음은 쿼리 계획의 일반적인 실행 단계입니다. 각 단계는 모든 단계에서 쿼리 성능을 평가하는 데 도움이 되도록 executionTimeMillisEstimate(실행 시간) 및 nReturned(문서 수) 지표를 반환합니다.

COLLSCAN(컬렉션 스캔)

- 문서별로 전체 컬렉션 문서를 스캔합니다.
- nReturned: 컬렉션에서 일치하는 모든 문서를 반환합니다.
- 인덱스를 사용할 수 없는 경우 표시되는 비용이 많이 드는 작업일 수 있습니다.

IXSCAN(인덱스 스캔)

- 인덱스를 사용하여 일치하는 문서를 찾습니다.
- nReturned: 인덱스를 기반으로 문서만 일치
- 효율적인 작업, COLLSCAN보다 선호

SORT

- 지정된 필드를 기준으로 문서를 정렬합니다.
- 쿼리의 정렬 속성이 인덱스의 일부가 아닌 경우이 단계는 명시적으로 표시됩니다.
- nReturned: 입력 문서와 동일한 번호

- 대규모 결과 집합에 대해 메모리 집약적입니다. 최적화하려면 정렬 필드 이름을 인덱스에 추가합니다.

LIMIT_SKIP

- 반환되고 건너뛴 문서 수를 제어합니다.
- nReturned: LIMIT 값으로 제한됨
- 페이지 매김 또는 LIMIT 작업에 사용됩니다.

하위 스캔

- 중첩 쿼리 작업 수행
- nReturned: 하위 쿼리 결과에 따라 다름
- 여러 단계가 있는 복잡한 쿼리에 사용됩니다.

executionTimeMillisEstimate는 최적화에 적합한 후보입니다.

> **Note**
>
> executionStats 파라미터는 현재 업데이트 및 삭제 명령을 지원하지 않습니다.

계획 실행 통계 설명의 docsExamined 이해

를 사용하여 쿼리를 실행하면 `explain("executionStats")` Amazon DocumentDB는 쿼리 결과를 생성하기 위해 스캔한 문서 수를 이해하는 데 도움이 되는 검사 지표를 제공합니다. 이러한 지표는 비효율적인 쿼리 계획을 식별하고 성능을 최적화하는 데 유용합니다.

> **Note**
>
> Amazon DocumentDB 5.0.1 이상 및 8.0.0 이상에서만 사용할 수 있습니다.

필드

필드	설명	수준
totalDocsExamined	모든 실행 단계에서 검사한 총 문서 수입니다.	최상위 executionStats
docsExamined	특정 실행 단계에서 검사한 문서 수입니다.	단계 수준

`docsExamined` 필드는 다음 단계에 나타납니다.

단계	설명
콜스캔	컬렉션 스캔. 컬렉션의 모든 문서가 검사됩니다.
ISSCAN	인덱스 스캔. 인덱스 조건과 일치하는 문서만 검사됩니다.
FETCH	최적화 프로그램이 IXSCAN과 별도의 단계에서 문서를 검색하면 FETCH 단계는 docsExamined를 보고합니다. 인덱스 스캔 계획에서 하위 IXSCAN 단계는 docsExamined를 보고하지 않습니다. $lookup 계획에서 FETCH 단계와 하위 단계 모두 docsExamined를 보고할 수 있습니다.

> **Note**
>
> `docsExamined` 쿼리가 문서에 액세스하지 않고 인덱스에서 완전히 충족되므로 IXONLYSCAN 스테이지에는가 표시되지 않습니다.

계획 executionStats 출력 설명에 docsExamined 표시

다음 예제에서는 500,000개의 문서가 있는 컬렉션을 사용하고 다양한 쿼리 실행 단계에서가 어떻게 `docsExamined` 변경되는지 보여줍니다.

샘플 데이터 생성:

```
for (let i = 0; i < 500000; i++) {
    db.coll.insertOne({ number: i, arr: [i, [i+1]], value: "test", bool: i % 2 ===
 0 });
```

```
}
```

컬렉션 스캔(COLLSCAN)

인덱스가 없으면 Amazon DocumentDB는 전체 컬렉션 스캔을 수행합니다.

```
db.coll.find({ "number": { "$lt": 500 } }).explain("executionStats").executionStats
```

출력:

```
{
    "executionSuccess" : true,
    "nReturned" : "500",
    "executionTimeMillis" : "282.055",
    "planningTimeMillis" : "0.085",
    "totalDocsExamined" : "500000",
    "executionStages" : {
        "stage" : "COLLSCAN",
        "nReturned" : "500",
        "executionTimeMillisEstimate" : "281.915",
        "docsExamined" : "500000",
        "filter" : {
            "number" : {
                "$lt" : 500
            }
        }
    }
}
```

500,000개의 모든 문서를 검사하여 500개의 결과를 반환했습니다. 숫자 필드에 인덱스를 생성하면이 기능이 향상됩니다.

인덱스 스캔(IXSCAN)

```
db.coll.createIndex({ number: 1 });
db.coll.find({ "number": { "$lt": 5000 } }).explain("executionStats").executionStats
```

출력:

```
{
    "executionSuccess" : true,
    "nReturned" : "5000",
    "executionTimeMillis" : "3.047",
    "planningTimeMillis" : "0.296",
    "totalDocsExamined" : "5000",
    "executionStages" : {
        "stage" : "IXSCAN",
        "nReturned" : "5000",
        "executionTimeMillisEstimate" : "2.576",
        "indexName" : "number_1",
        "direction" : "forward",
        "docsExamined" : "5000",
        "indexCond" : {
            "$and" : [
                {
                    "number" : {
                        "$lt" : 5000
                    }
                }
            ]
        }
    }
}
```

인덱스를 사용하면 반환된 수와 일치하는 500,000개 문서 중 5,000개만 검사하고 가져왔습니다. 인덱스 조건은 쿼리와 일치하지 않는 495,000개의 문서를 필터링했습니다.

잔차 필터를 사용한 인덱스 스캔

```
db.coll.find({ "number": { "$lt": 5000 }, "arr": { "$gt":
 4000 } }).explain("executionStats").executionStats
```

출력:

```
{
    "executionSuccess" : true,
    "nReturned" : "999",
    "executionTimeMillis" : "15.367",
    "planningTimeMillis" : "0.115",
    "totalDocsExamined" : "5000",
    "executionStages" : {
        "stage" : "IXSCAN",
        "nReturned" : "999",
        "executionTimeMillisEstimate" : "15.170",
        "indexName" : "number_1",
        "direction" : "forward",
        "docsExamined" : "5000",
        "indexCond" : {
            "$and" : [
                {
                    "number" : {
                        "$lt" : 5000
                    }
                }
            ]
        },
        "filter" : {
            "arr" : {
                "$gt" : 4000
            }
        }
    }
}
```

인덱스 조건이 500,000개 중 5,000개의 문서와 일치하면의 잔차 필터가 결과를 999로 `arr` 줄였습니다. `docsExamined` 값 5,000은 인덱스 조건 이후이지만 잔차 필터가 적용되기 전에 검사된 모든 문서를 반영합니다.

IXSCAN을 사용하여 가져오기

```
db.coll.find({ "$or": [{ "number": { "$lt": 100000 } }, { "number": { "$gt":
 400000 } }] }).explain("executionStats").executionStats
```

출력:

```
{
    "executionSuccess" : true,
    "nReturned" : "199999",
    "executionTimeMillis" : "899.801",
    "planningTimeMillis" : "0.183",
    "totalDocsExamined" : "199999",
    "executionStages" : {
        "stage" : "FETCH",
        "nReturned" : "199999",
        "executionTimeMillisEstimate" : "894.141",
        "docsExamined" : "199999",
        "inputStage" : {
            "stage" : "IXOR",
            "nReturned" : "0",
            "executionTimeMillisEstimate" : "874.897",
            "inputStages" : [
                {
                    "stage" : "IXSCAN",
                    "nReturned" : "100000",
                    "executionTimeMillisEstimate" : "462.208",
                    "indexName" : "number_1",
                    "indexCond" : {
                        "$and" : [
                            {
                                "number" : {
                                    "$lt" : 100000
                                }
                            }
                        ]
                    }
                },
                {
                    "stage" : "IXSCAN",
                    "nReturned" : "99999",
                    "executionTimeMillisEstimate" : "412.684",
                    "indexName" : "number_1",
                    "indexCond" : {
                        "$and" : [
                            {
                                "number" : {
                                    "$gt" : 400000
```

```
                                }
                            }
                        ]
                    }
                }
            ]
        }
    }
}
```

Amazon DocumentDB 옵티마이저가 가져오기 단계를 사용하여 문서를 검색하는 경우 FETCH 단계는를 보고합니다docsExamined. 하위 IXSCAN 단계는 문서에 직접 액세스하지 않고 인덱스 키만 스캔docsExamined하므로 보고하지 않습니다.

집계 조회가 포함된 FETCH

```
db.coll.explain("executionStats").aggregate([
    { $match: { "number": { "$lt": 5 } } },
    { $lookup: { from: "coll", pipeline: [{ $match: { "number": { "$lt": 3 } } }], as:
 "sub" } }
]).executionStats
```

출력:

```
{
    "executionSuccess" : true,
    "nReturned" : "5",
    "executionTimeMillis" : "0.525",
    "planningTimeMillis" : "0.327",
    "totalDocsExamined" : "9",
    "executionStages" : {
        "stage" : "NESTED_LOOP_LOOKUP",
        "nReturned" : "5",
        "executionTimeMillisEstimate" : "0.163",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                "nReturned" : "5",
                "executionTimeMillisEstimate" : "0.039",
```

```
                "indexName" : "number_1",
                "direction" : "forward",
                "docsExamined" : "5",
                "indexCond" : {
                    "$and" : [
                        {
                            "number" : {
                                "$lt" : 5
                            }
                        }
                    ]
                }
            },
            {
                "stage" : "FETCH",
                "nReturned" : "1",
                "executionTimeMillisEstimate" : "0.009",
                "docsExamined" : "1",
                "inputStage" : {
                    "stage" : "AGGREGATE",
                    "nReturned" : "1",
                    "executionTimeMillisEstimate" : "0.044",
                    "inputStage" : {
                        "stage" : "IXSCAN",
                        "nReturned" : "3",
                        "executionTimeMillisEstimate" : "0.013",
                        "indexName" : "number_1",
                        "direction" : "forward",
                        "docsExamined" : "3",
                        "indexCond" : {
                            "$and" : [
                                {
                                    "number" : {
                                        "$lt" : 3
                                    }
                                }
                            ]
                        }
                    }
                }
            }
        ]
    }
}
```

외부 IXSCAN은 숫자 < 5와 일치하는 문서 5개를 검사했습니다. 내부 IXSCAN은 일치하는 숫자 < 3인 문서 3개를 검사했고 FETCH 단계에서는 집계된 결과 1개를 검사했습니다. `totalDocsExamined`의 9는 모든 단계(5 + 3 + 1)의 합계입니다.

인덱스 전용 스캔(적용된 쿼리)

인덱스 전용 스캔(적용된 쿼리라고도 함)은 Amazon DocumentDB가 기본 문서를 읽지 않고 인덱스에 저장된 필드에서 쿼리에 완전히 응답하는 최적화입니다. 쿼리에 필요한 값이 인덱스에 이미 있으므로 Amazon DocumentDB는 일치하는 각 문서 가져오기를 건너뛰어 무작위 I/O를 방지하고 전체 문서를 읽는 비용을 디코딩할 수 있습니다. 쿼리 계획에서 인덱스 전용 스캔은 `IXSCAN` (인덱스 스캔) 또는 `COLLSCAN` (컬렉션 스캔) 대신 `IXONLYSCAN` 단계로 나타납니다. 이렇게 하면 I/O가 줄어들고 지원되는 쿼리의 쿼리 지연 시간이 크게 단축될 수 있습니다.

쿼리는 단일 인덱스에 모두 존재하는 소수의 필드만 터치할 때, 특히 문서 자체가 큰 경우 인덱스 전용 스캔의 좋은 후보입니다. 일반적인 예로는 대시보드, 분석 롤업 및 더 넓은 복합 인덱스에서 몇 가지 필드를 읽는 조회가 있습니다.

쿼리 결과는 인덱스 전용 스캔을 사용하든 사용하지 않든 동일합니다. 쿼리 계획과 수행하는 I/O의 양만 변경됩니다.

> **Note**
>
> 인덱스 전용 스캔 지원은 Amazon DocumentDB 엔진 버전에 따라 다릅니다.
>
> - 필드가 `_id` 아닌 2개 이상의 가려진 `find` 프로젝션에는 엔진 버전 8.0.2 이상이 필요합니다. 이전 버전에서는 커버드 `find` 프로젝션에 필드가 아닌 `_id` 필드가 하나 이상 포함될 수 있습니다. 둘 이상의 필드가 있는 프로젝션은 정상 인덱스 스캔 또는 컬렉션 스캔으로 돌아갑니다.
> - 적용 대상와 같은 집계 파이프라인에 대한 인덱스 전용 스캔에는 엔진 버전 8.0.2 이상이 `$group`필요합니다.

Amazon DocumentDB가 인덱스 전용 스캔을 사용하는 경우

Amazon DocumentDB는 단일 인덱스에 작업이 읽는 모든 필드가 포함된 경우 다음 작업에 대한 인덱스 전용 스캔을 고려합니다.

커버드 **find** 프로젝션

find 쿼리는 프로젝션이 하나 이상의 최상위 필드를 단순하게 포함하고 모든 프로젝션된 필드가 단일 인덱스에 있는 경우를 검증합니다. 예를 들어의 복합 인덱스가 주어지면 { customerId: 1, status: 1, total: 1 }다음 쿼리가 포함됩니다. 세 개의 프로젝션된 필드는 모두 인덱스에 있으며 _id 명시적으로 제외됩니다. (필드_id가 아닌 필드가 두 개 이상인 프로젝션을 관리하려면 엔진 버전 8.0.2 이상이 필요합니다.)

```
db.orders.createIndex({ customerId: 1, status: 1, total: 1 })

db.orders.find(
  { customerId: 42 },
  { _id: 0, customerId: 1, status: 1, total: 1 }
)
```

우승 계획은 IXONLYSCAN 스테이지를 사용하며 스토리지에서 일치하는 문서를 가져오지 않습니다.

```
{
  "stage": "IXONLYSCAN",
  "indexName": "customerId_1_status_1_total_1",
  "direction": "forward",
  "indexCond": { "$and": [ { "customerId": { "$eq": 42 } } ] }
}
```

> **Note**
>
> 적용 대상 find 쿼리에서 결과의 필드는 프로젝션에 나열된 순서 또는 문서에 저장된 순서가 아닌 인덱스 순서(필드가 인덱스 정의에 표시되는 순서)로 반환됩니다. 이는 해당 쿼리에 대한 MongoDB 동작과 일치합니다. 컬렉션 스캔으로 응답되는 쿼리는 대신 저장 문서 순서로 필드를 반환합니다.

적용 **$group** 집계

$group 스테이지는 일반적으로 몇 개의 필드(그룹 키 및 해당 누적기에 대한 입력)만 읽지만 선행 없이 실행되는 경우가 많습니다$match. 커버링 인덱스가 없으면 Amazon DocumentDB는 컬렉션의 모든 전체 문서를 읽고 디코딩하여 해당 몇 개의 필드를 그룹화에 공급하기만 합니다.

포함된 $group 단계를 포함한 집계 파이프라인에 대한 인덱스 전용 스캔은 엔진 버전 8.0.2 이상에서 사용할 수 있습니다.

단일 인덱스에 파이프라인 접두사가 읽는 모든 필드가 포함된 경우 Amazon DocumentDB는 그룹 아래에 인덱스 전용 스캔을 배치하고 인덱스에서 직접 해당 필드를 읽을 수 있습니다. 에 복합 인덱스를 사용하여 상태별로 주문을 그룹화하고 합계를 합산하는 파이프라인을 생각해 보세요. `{ status: 1, total: 1 }`

```
db.orders.createIndex({ status: 1, total: 1 })

db.orders.aggregate([
  { $group: { _id: "$status", revenue: { $sum: "$total" } } }
])
```

파이프라인은 인덱스에 total있는 status 및 만 읽으므로 그룹은 컬렉션 스캔 IXONLYSCAN 대신에서 공급됩니다.

```
// Without a covering index
$group  <-  COLLSCAN     (reads and decodes every full document)

// With a covering index on { status, total }
$group  <-  IXONLYSCAN over { status, total }   (reads only the referenced fields)
```

앞에 $match, $skip, 또는 $sort$limit가 지원$group됩니다. 그러나 파이프라인이 포함되려면 이러한 단계가 참조하는 모든 필드(예: $match 필터 필드 또는 $sort 키)도 동일한 인덱스에 있어야 합니다. 인덱스에서 참조된 필드가 누락된 경우 Amazon DocumentDB는 컬렉션 또는 인덱스 스캔으로 돌아갑니다.

> **Note**
>
> find 프로젝션과 달리 에는 문서가 암시적으로 포함되지 $group 않습니다_id. _id의 $group는 문서 식별자가 아닌 그룹 키이므로 인덱스에 파이프라인을 덮_id기 위해 문서를 포함할 필요가 없습니다.

적용 대상 쿼리에 대한 요구 사항

Amazon DocumentDB가 인덱스 전용 스캔을 사용하려면 다음을 모두 포함해야 합니다.

- 단일 인덱스에는 쿼리가 읽는 모든 필드가 포함됩니다. 여기에는의 프로젝션된 필드find와 그룹 키, 누산기 입력, 해당 $group 파이프라인의 필터 또는 정렬 필드가 포함됩니다. 인덱스에 필드가 하나도 없는 경우 쿼리를 덮을 수 없습니다.
- 프로젝션에는 최상위 필드(용)가 포함됩니다find. 제외 프로젝션, 계산된 필드 및 $slice 및와 같은 프로젝션 연산$elemMatch자는 사용할 수 없습니다.
- **_id** 필드가 고려됩니다(의 경우find). _id는 기본적으로 포함되므로와 같은 일반적인 복합 인덱스{ a: 1, b: 1 }는 find({}, { a: 1, b: 1 })를 다루지 않습니다. 암시적 _id는 인덱스에 없습니다. 프로젝션_id에서 제외하거나({ _id: 0, a: 1, b: 1 }) 인덱스_id에를 추가합니다.
- 필터는 인덱스에 의해 완전히 충족됩니다. Amazon DocumentDB는 쿼리 조건자의 일부를 인덱스 바인딩으로 변환할 수 없는 경우 해당 부분을 잔차 필터로 유지하고, 문서를 읽어 평가하며, 스캔을 인덱스 전용 스캔으로 변환하지 않습니다. $mod, $ne, $nin, 등의 연산자는 잔차 필터를 $exists 남깁니다.
- 프로젝션된 필드는 멀티키(배열) 열이 아닙니다. 제한 사항 섹션을 참조하세요.

제한 사항

다음 경우에는 인덱스 전용 스캔을 사용할 수 없으며 정상 인덱스 스캔 또는 컬렉션 스캔으로 돌아갑니다. 이는 예상되는 동작이며 결과의 정확성에 영향을 미치지 않습니다.

- 멀티키(배열) 필드. 필드에 어떤 문서에든 배열 값이 있는 경우 해당 인덱스 열은 멀티키입니다. 멀티키 인덱스는 배열 요소당 하나의 항목을 저장하므로 인덱스에서 원래 배열을 재구성할 수 없습니다. 이러한 필드를 읽는 쿼리는 다루지 않습니다.
- 점선 또는 중첩 경로. 인덱스에 해당 경로가 포함되어 있더라도와 같은 중첩 경로의 프로젝션 또는 필드 참조"address.city"는 사용할 수 없습니다.

- 제외, 계산된 또는 연산자 프로젝션. `{ b: 0 }`, `{ total: { $add: ["$price", "$tax"] } }`또는와 같은 프로젝션에는 전체 문서가 `{ a: { $slice: 5 } }` 필요하며 적용되지 않습니다.
- 집계의 전체 문서 참조입니다. 전체 문서를 참조`$group`하는는 인덱스가 전체 문서를 재구성할 수 없으므로(예: `{ $push: "$$ROOT" }`) 다룰 수 없습니다.
- 이전 단계 재구성**`$group`**. 가 파이프라인의 적용 자격을 `$group` 상실하기 전에 나타나는 문서(예: `$project`, `$addFields``$unwind`, 또는 `$lookup`)를 재구성하는 단계입니다.
- 문자열 조건자의 데이터 정렬 불일치. 쿼리의 데이터 정렬이 인덱스의 데이터 정렬과 다르면 인덱스에서 문자열 조건자를 제공할 수 없으며 문서 가져오기를 강제하는 잔차 필터를 남깁니다. 숫자 및 유형 기반 조건자는 영향을 받지 않습니다.

인덱스 전용 스캔 확인

`explain("executionStats")` 출력을 사용하여 쿼리가 인덱스 전용 스캔에서 제공되고 문서에 접촉하지 않는지 확인합니다. 다음 두 가지를 찾습니다.

- 이기는 계획은 `IXONLYSCAN` 스테이지를 사용합니다.
- 해당 단계의 `docsExamined` 값은 `0` (및 `0`)`totalDocsExamined`이며, 이는 문서를 읽지 않고 쿼리가 인덱스에서 완전히 응답되었음을 의미합니다.

```
db.orders.find(
  { customerId: 42 },
  { _id: 0, customerId: 1, status: 1, total: 1 }
).explain("executionStats").executionStats
```

```
{
    "executionSuccess": true,
    "nReturned": "2",
    "executionTimeMillis": "0.714",
    "totalDocsExamined": "0",
    "executionStages": {
        "stage": "IXONLYSCAN",
        "nReturned": "2",
        "indexName": "customerId_1_status_1_total_1",
```

```
        "direction": "forward",
        "docsExamined": "0"
    }
}
```

> **Note**
>
> IXONLYSCAN 단계의 0이 아닌 docsExamined 값은 인덱스에 데이터가 포함되어 있음을 의미하지만 Amazon DocumentDB는 여전히 스토리지를 확인하여 각 문서 버전이 표시되는지 확인했습니다. 이는 컬렉션에 더 이상 사용되지 않는 문서 버전 가비지 컬렉션이 아직 회수되지 않은 최근 쓰기가 있는 경우 발생할 수 있습니다. 가비지 수집이 해당 문서를 처리하면 가 로 docsExamined 삭제됩니다0. 이때 쿼리는 적용 스캔의 모든 이점을 실현합니다. 가비지 수집에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB의 폐영역 회수. docsExamined 지표에 대한 자세한 내용은 쿼리 계획 분석를 참고하세요.

모범 사례

- 가장 빈번한 쿼리 프로젝트 또는 그룹이 있는 필드가 포함된 복합 인덱스를 설계하여 해당 쿼리를 처리할 수 있습니다.
- _id가 커버링 인덱스의 일부가 아닌 한 find 프로젝션({ _id: 0, ... })_id에서 제외합니다.
- 다중 키 열은 인덱스 전용 경로를 부적격화하므로 쿼리를 다루려는 경우 배열(멀티키) 필드를 인덱싱하지 마십시오.
- 포함된 스캔이 문서 액세스를 건너뛰고를 docsExamined 유지하도록 가비지 수집이 쓰기 워크로드를 따라잡도록 허용합니다0. 자세한 내용은 Amazon DocumentDB의 폐영역 회수 섹션을 참조하세요.

캐시 필터 계획

인덱스 필터라고도 하는 계획 캐시 필터는 쿼리 플래너가 특정 쿼리 셰이프에 대해 고려할 수 있는 인덱스 세트를 제한합니다. 쿼리가 필터가 있는 셰이프와 일치하면 플래너는 컬렉션의 모든 인덱스를 평가하는 대신 해당 필터에 이름이 지정된 인덱스에서만 계획을 선택합니다.

쿼리 셰이프는 조건자 값이 정규화된 쿼리 조건자, 정렬 사양 및 컬렉션 네임스페이스의 조합입니다. 값만 다른 두 쿼리는 셰이프를 공유하므로 단일 필터가 모든 쿼리를 포괄합니다. 의 출력에서 planCacheListFilters정규화된 값은 로 표시됩니다"@".

계획 캐시 필터를 사용하면 계획을 고정할 수 있으므로 데이터 배포 또는 인덱스 변경으로 인해 느린 인덱스를 선택하지 않아도 됩니다. 필터의 속성은 다음과 같습니다.

- 애플리케이션 변경은 필요하지 않습니다. 필터는 데이터베이스 명령으로 설정되고 서버에 적용되므로 코드 배포 없이 회귀를 완화할 수 있습니다. 이는 모든 호출 사이트에 추가해야 `hint`하는와 주요 차이점입니다.
- 범위는 하나의 쿼리 셰이프입니다. 셰이프는 전체 쿼리가 아닌 컬렉션, 조건자 및 정렬에 의해 정의되므로 필터는 해당 셰이프와 일치하는 쿼리만 제한합니다. 동일한 컬렉션에 대한 다른 쿼리는 정상적으로 계속 계획됩니다.
- 변경 사항은 내구성이 뛰어납니다. 필터는 인스턴스 재시작 및 엔진 패치 적용 전반에 걸쳐 유지됩니다. 컬렉션을 삭제하면 필터가 제거됩니다.
- 변경 사항은 되돌릴 수 있습니다. 필터를 지우면 셰이프가 일반적인 비용 기반 계획으로 반환되므로 기본 원인을 조사하는 동안 워크로드를 안정화할 수 있는 위험이 낮은 방법입니다.

주제

- 지원되는 명령
- 지원되는 쿼리 셰이프
- 필터 설정, 나열 및 지우기
- 필터가 적용되었는지 확인
- 예제
- 힌트 및 인덱스 변경 사항이 있는 동작

지원되는 명령

계획 캐시 필터에는 플래너 버전 2.0 이상이 필요합니다. 필터를 적용할 수 있는 명령은 다음 표와 같이 플래너 버전과 Amazon DocumentDB 마이너 버전에 따라 달라집니다.

계획 캐시 필터에 대한 명령 지원

명령	최소 플래너 버전	최소 마이너 버전
`find`	2.0	5.0.0, 8.0.0
`count`	2.0	5.0.0, 8.0.0
`update`	2.0	5.0.2, 8.0.2

명령	최소 플래너 버전	최소 마이너 버전
delete	2.0	5.0.2, 8.0.2
findAndModify	2.0	5.0.2, 8.0.2
distinct	3.0	8.0.0
aggregate	3.0	8.0.2

> **Note**
>
> 플래너 버전 3.0은 Amazon DocumentDB 8.0에서만 사용할 수 있습니다. distinct 및 aggregate 명령에는 플래너 버전 3.0이 필요하므로 이러한 두 명령에 대한 계획 캐시 필터는 Amazon DocumentDB 8.0에서만 사용할 수 있습니다. 플래너 버전 2.0을 지원하는 Amazon DocumentDB 5.0에서는 필터가 find, count, update, delete및 findAndModify 명령에 적용됩니다.

aggregate 명령의 경우 셰이프는 파이프라인 전면에서 가져옵니다. 선행는 필터를 $match 제공하고,는 정렬을 제공한 $sort 직후를 제공합니다. $skip와는 셰이프에 영향을 $limit 주지 않으며,는 해당 지점 이후의 단계를 수행하지 않습니다.

Amazon DocumentDB는 계획하기 전에 파이프라인을 최적화하고 계획 캐시 필터를 최적화된 파이프라인과 일치시킵니다. 따라서 플래너가 보는 필터와 정렬은 작성한 단계와 다를 수 있습니다. 예를 들어 파이프라인 앞에 있는 두 개의 인접 $match 단계가 단일 $and 조건자로 결합됩니다.

```
db.orders.aggregate([
  { $match: { status: "SHIPPED" } },
  { $match: { region: "us-east-1" } },
  { $group: { _id: "$customerId", total: { $sum: "$amount" } } }
])
```

플래너는 $match에 하나를 표시{$and: [{status: ...}, {region: ...}]}하므로 필터를 결합된 조건자에 설정해야 합니다.

```
db.runCommand({planCacheSetFilter: "orders",
query: { $and: [ { status: "SHIPPED" }, { region: "us-east-1" } ] },
```

```
indexes: [ "status_1_region_1" ]})
```

$lookup 또는 $graphLookup 단계의 외래 컬렉션에 설정된 필터는 해당 외래 컬렉션을 스캔하는 데 사용되는 인덱스를 구동합니다.

지원되는 쿼리 셰이프

Amazon DocumentDB 5.0.2 및 8.0.2부터 계획 캐시 필터가 있는 쿼리 셰이프에 다음 연산자가 나타날 수 있습니다.

- $text. 검색 문자열이 정규화되므로 검색하는 용어만 다른 쿼리는 하나의 셰이프를 공유합니다.
- $near 및 $nearSphere. 두 연산자 모두 동일한 셰이프로 정규화됩니다. 좌표, $minDistance및 $maxDistance는 셰이프의 일부가 아닙니다.
- $geoWithin 및 $geoIntersects. GeoJSON 지오메트리 유형은 셰이프의 일부이므로 Polygon 쿼리와 MultiPolygon 쿼리는 다른 셰이프입니다. 좌표는 셰이프의 일부가 아닙니다.
- $regex, 필드에서 직접 사용하는 경우. 패턴 및 옵션이 정규화되므로 동일한 필드의 모든 정규 표현식은 하나의 셰이프를 공유합니다.

$jsonSchema, $expr또는를 사용하는 쿼리 셰이프에 대한 오류 코드 303과 함께 필터 설정이 실패합니다$sampleRate. 다음 셰이프에 대해서도 실패합니다. 이러한 쿼리를 사용하는 쿼리는 필터 없이 계획됩니다.

- $in, $nin또는 내에 $regex 중첩된 $all. 정규식을 리터럴 값과 혼합하는 배열은 인덱스에 대해 평가할 수 없으므로 해당 셰이프의 필터는 적용할 수 없습니다.
- 의 정렬 사양입니다{$natural: 1}. $natural 정렬은 플래너를 인덱스로 제한하는 것과 호환되지 않는 컬렉션 스캔을 요청합니다.

필터 설정, 나열 및 지우기

계획 캐시 필터를 설정하려면 planCacheSetFilter 명령을 사용합니다. query 및 sort 필드는 함께 쿼리 셰이프를 지정하고 플래너가 해당 셰이프에 대해 고려할 수 있는 인덱스 이름을 indexes 나열합니다.

```
db.runCommand({
planCacheSetFilter: <collection>,
query: <query>,
```

```
sort: <sort>, // optional
indexes: [ <index1>, <index2>, ...],
comment: <any> // optional
})
```

예를 들어 다음 명령은 플래너를 정렬 없이 셰이프의 a_1 인덱스{a: {$eq: "@"}, b: {$eq: "@"}}로 제한합니다.

```
db.runCommand({planCacheSetFilter: "foo",
query: { a: 1, b: 1 },
sort: {},
indexes: [ "a_1" ]})
```

이미 있는 셰이프에 대한 필터를 설정하면 기존 필터가 대체됩니다. 다음 두 명령을 실행한 후 셰이프의 필터는 {a: {$eq: "@"}, b: {$eq: "@"}}입니다["b_1"].

```
db.runCommand({planCacheSetFilter: "foo",
query: { a: 1, b: 1 },
sort: {},
indexes: [ "a_1" ]})

db.runCommand({planCacheSetFilter: "foo",
query: { a: 6, b: 10 },
sort: {},
indexes: [ "b_1" ]})
```

컬렉션의 모든 필터를 나열하려면 planCacheListFilters 명령을 사용합니다.

```
db.runCommand({planCacheListFilters: <collection>})
```

출력에는 정규화된 값과 허용된 인덱스 목록이 있는 저장된 각 셰이프가 표시됩니다.

```
{
"filters" : [
{
"query" : { "a" : { "$eq" : "@" } },
"sort" : { },
"indexes" : [ "a_1" ]
},
{
```

```
"query" : { "a" : { "$gt" : "@" } },
"sort" : { "a" : 1 },
"indexes" : [ "a_1_b_1" ]
}
],
"ok" : 1
}
```

연산자는 보존되며 값만 로 대체됩니다"@". 와 같은 암시적 동등성{a: 1}은 명시적 형식인 로 나열{"a": {"$eq": "@"}}되며 $in 배열의 요소는 전체적으로 로 대체됩니다{"a": {"$in": "@"}}. 정렬이 없는 필터 세트는 빈 sort 문서와 함께 나열됩니다.

필터를 제거하려면 planCacheClearFilters 명령을 사용합니다.

```
db.runCommand({
planCacheClearFilters: <collection>,
query: <query pattern>, // optional
sort: <sort specification>, // optional
comment: <any> // optional
})
```

컬렉션의 모든 필터를 지우려면 query 및를 모두 생략합니다sort.

```
db.runCommand({planCacheClearFilters: "foo"})
```

query 및를 모두 제공하면 정확한 정렬이 있는 필터만 sort지워집니다. 의 값은 query 정규화되므로 모든 대표 값이 작동합니다. 정렬 없이 설정된 필터만 지우려면 빈 sort 문서를 전달합니다.

```
db.runCommand({planCacheClearFilters: "foo", query: {a: 1}, sort: {a: 1}})

db.runCommand({planCacheClearFilters: "foo", query: {a: 1}, sort: {}})
```

필터가 적용되었는지 확인

explain 출력에는 계획 캐시 필터 상태를 보고하는 두 개의 필드가 포함됩니다.

- indexFilterSet는 쿼리 셰이프에 대한 컬렉션에 필터가 있는 true 경우입니다.
- indexFilterApplied는 해당 필터 목록의 인덱스를 사용하여 쿼리를 계획한 true 경우에만 입니다.

Amazon DocumentDB 5.0.2 및 8.0.2부터 이러한 필드는 계획 캐시 필터를 지원하는 모든 명령과 해당 명령의 프로파일러 출력에 보고됩니다. 이전 마이너 버전에서는 find 명령에 대해서만 보고되었습니다. 설명 출력 읽기에 대한 자세한 내용은 섹션을 참조하세요쿼리 계획 분석.

필터는 적용하지 않고 쿼리 셰이프와 일치할 수 있습니다. 필터 목록의 인덱스 중 어느 것도 쿼리를 처리할 수 없는 경우 플래너는 필터 없이 계획을 다시 세우고 비용에 대한 인덱스를 선택합니다. 이 경우 indexFilterSet는 true 이고는 indexFilterApplied입니다false. 이는 명명된 인덱스가 쿼리의 필드를 다루지 않거나, 명명된 인덱스가 존재하지 않거나, $regex셰이프가 고정되지 않았거나 대/소문자를 구분하지 않는 패턴과 같은 인덱스 경계를 생성할 수 없는 경우에 발생합니다.

예제

다음 예제에서는 컬렉션orders과이 $lookup 예제의 customers 컬렉션을 이러한 인덱스와 함께 사용합니다.

```
db.orders.createIndex({ status: 1 })                          // status_1
db.orders.createIndex({ status: 1, orderDate: 1 })            // status_1_orderDate_1
db.orders.createIndex({ customerId: 1 })                      // customerId_1
db.customers.createIndex({ customerId: 1 })                   // customerId_1
```

예: **find** 쿼리를 복합 인덱스에 고정

를 필터링status하고를 정렬하는 쿼리를 생각해 보십시오. orderDate

```
db.orders.find({ status: "SHIPPED" }).sort({ orderDate: 1 })
```

복합 인덱스는 조건자와 정렬을 모두 status_1_orderDate_1 충족하므로 별도의 정렬이 필요하지 않습니다. 플래너가 대신를 선택하는 경우 status_1쿼리는 결과를 정렬해야 합니다. 그러면 SORT 단계가 추가되고 일치하는 주문 수가 증가함에 따라 비용이 더 많이 듭니다. 플래너를이 셰이프의 복합 인덱스로 제한하려면:

```
db.runCommand({planCacheSetFilter: "orders",
query: { status: "SHIPPED" },
sort: { orderDate: 1 },
indexes: [ "status_1_orderDate_1" ]})
```

값은 셰이프에서 정규화되므로이 하나의 필터는 { status: "PENDING" }, { status: "CANCELLED" }및 동일한 정렬status로의 다른 모든 값도 포함합니다. 별도의 셰이프인 다른 정렬 또는 정렬 없는 동일한 조건자를 다루지 않습니다. 에서 필터가 적용되었는지 확인합니다explain.

```
db.orders.find({ status: "SHIPPED" }).sort({ orderDate: 1 }).explain()
```

출력은 indexFilterSet: true 및 indexFilterApplied: true를 보고하고 IXSCAN 단계 이름은 입니다status_1_orderDate_1.

예: 선택적 인덱스**update**에 고정

업데이트는 문서를 쓰기 전에 수정할 문서를 찾아야 하며, 해당 조회는 읽기와 동일한 방식으로 인덱스를 사용합니다. 조건자를 둘 이상의 인덱스에서 제공할 수 있는 경우 플래너가 선택한 인덱스에 따라 업데이트에서 검사하는 문서 수가 결정됩니다.

```
db.orders.updateMany(
  { customerId: 4815, status: "PENDING" },
  { $set: { status: "CANCELLED" } }
)
```

customerId_1 및 모두이 조건자를 제공할 status_1 수 있습니다. 업데이트를에 고정하려면customerId_1:

```
db.runCommand({planCacheSetFilter: "orders",
query: { customerId: 4815, status: "PENDING" },
indexes: [ "customerId_1" ]})
```

쓰기 명령의 필터에는 Amazon DocumentDB 5.0.2 또는 8.0.2 및 플래너 버전 2.0 이상이 필요합니다. 업데이트explain의를 사용하여 확인합니다.

```
db.runCommand({explain: {update: "orders",
updates: [{ q: { customerId: 4815, status: "PENDING" },
             u: { $set: { status: "CANCELLED" } },
             multi: true }]}})
```

이기는 계획은 IXSCAN의에 대한 UPDATE 단계입니다customerId_1. 셰이프에 쓰기 유형이 포함되지 않으므로이 쿼리 셰이프를 사용하는 delete 및 findAndModify 명령에는 동일한 필터가 적용됩니다.

예: **aggregate** 파이프라인 및 **$lookup** 외래 컬렉션 고정

집계의 경우 셰이프는 선행 $match 단계에서 가져온 다음 $sort 바로 뒤에를 만듭니다. $group 또는와 같은 그 이상의 스테이지$project는 셰이프의 일부가 아닙니다.

```
db.orders.aggregate([
  { $match: { status: "SHIPPED" } },
  { $group: { _id: "$customerId", total: { $sum: "$amount" } } }
])
```

$match 조건자에 설정된 필터는 orders를 읽는 데 사용되는 인덱스를 구동합니다.

```
db.runCommand({planCacheSetFilter: "orders",
query: { status: "SHIPPED" },
indexes: [ "status_1" ]})
```

필터는 $lookup 또는 $graphLookup 단계가 조인하는 컬렉션에도 적용됩니다. 필터는 파이프라인이 실행되는 컬렉션이 아닌 외래 컬렉션에 설정됩니다. 다음 파이프라인에서 orders는 외부 컬렉션이고는 외부 컬렉션customers입니다.

```
db.orders.aggregate([
  { $lookup: { from: "customers",
               localField: "customerId",
               foreignField: "customerId",
               as: "customer" } }
])
```

조인 프로브는 외부 문서당 customers 한 번씩 실행되므로 선택한 인덱스가 반복적으로 적용되고 파이프라인 전체에 비용이 곱해집니다. 해당 스캔을 특정 인덱스에 고정하려면 조인 키의 등식 셰이프에 customers 대해 필터를에 설정합니다.

```
db.runCommand({planCacheSetFilter: "customers",
query: { customerId: 1 },
indexes: [ "customerId_1" ]})
```

값이 셰이프에서 정규화되므로 1 위의 값은 자리 표시자이며 모든 값은 동일한 필터를 생성합니다. 외래 수집 필터가 적용되면 최상위 indexFilterSet 및 indexFilterApplied 필드는 true를 보고하고, 선택한 필터가 낙찰 계획의 외래 측에 나타나는 인덱스를 보고합니다. 의 필터에는 Amazon DocumentDB 8.0.2 및 플래너 버전 3.0이 aggregate 필요합니다.

힌트 및 인덱스 변경 사항이 있는 동작

- 계획 캐시 필터는 보다 우선합니다hint. Amazon DocumentDB는 먼저 필터를 적용한 다음 필터의 허용된 인덱스 목록 내에 힌트를 적용합니다. 힌트가 필터가 허용하는 인덱스의 이름을 지정하면 해

당 인덱스가 사용됩니다. 힌트가 필터가 허용하지 않는 인덱스의 이름을 지정하는 경우 힌트는 무시되고 플래너는 허용되는 인덱스 중에서 비용을 기준으로 선택합니다. 두 힌트 형식 모두 인덱스 이름을 지정하든 키 패턴을 지정하든 동일한 방식으로 작동합니다.

예를 들어 컬렉션에 인덱스 `a_1_b_1`, `a_1` `b_1`및 `foo`가 있고 인덱스 목록 `{query: {a: {$eq: "@"}, b: {$eq: "@"}}, sort: {a: 1}}`를 사용하여 셰이프에 필터를 설정한 `["a_1", "a_1_b_1"]`경우 `a_1`실행은 힌트에 이름이 지정되고 필터에서 허용되므로를 `db.foo.find({ a: 10, b: 20 }).sort({a: 1}).hint({ a: 1 })` 사용합니다.

- 인덱스를 삭제해도 필터는 수정되지 않습니다. 필터는 인덱스 이름을 유지합니다. 해당 인덱스가 누락된 동안 계획은 인덱스를 건너뜁니다. 필터 목록의 다른 인덱스가 쿼리를 처리할 수 있는 경우에도 필터는 계속 적용됩니다. 명명된 인덱스를 사용할 수 없는 경우에만가 `indexFilterApplied` 보고됩니다`false`. 동일한 이름의 인덱스를 다시 생성하면 다시 사용할 수 있습니다.
- 컬렉션을 삭제하면 해당 컬렉션의 모든 필터가 제거됩니다.

Amazon DocumentDB 모범 사례

Amazon DocumentDB(MongoDB 호환)를 사용하여 작업하는 모범 사례를 알아보세요. 이 섹션은 새로운 모범 사례가 확인되는 대로 지속적으로 업데이트됩니다.

주제

- 기본 운영 지침
- 인스턴스 크기 조정
- 인덱스 작업
- 보안 모범 사례
- 비용 최적화
- 지표를 통해 성능 문제 식별
- TTL 및 시계열 워크로드
- 마이그레이션
- 클러스터 파라미터 그룹 작업
- 집계 파이프라인 쿼리
- batchInsert 및 batchUpdate

기본 운영 지침

다음은 Amazon DocumentDB로 작업할 때 모든 사용자가 따라야 하는 기본 운영 지침입니다. Amazon DocumentDB 서비스 수준 계약에 다음 지침을 따르도록 명시되어 있습니다.

- 2개의 AWS 가용 영역에 2개 이상의 Amazon DocumentDB 인스턴스로 구성된 클러스터를 배포합니다. 프로덕션 워크로드의 경우 3개 이상의 가용 영역에 3개 이상의 Amazon DocumentDB 인스턴스로 구성된 클러스터를 배포합니다.
- 명시된 서비스 제한 내에서 서비스를 사용하세요. 자세한 내용은 Amazon DocumentDB 할당량 섹션을 참조하세요.
- 메모리, CPU, 연결 및 스토리지 사용량을 모니터링합니다. 시스템 성능 및 가용성을 유지 관리하기 위해, 사용 패턴이 변경되거나 배포 용량에 도달할 때 알리도록 Amazon CloudWatch를 설정합니다.
- 용량 한도에 도달할 경우 인스턴스를 확장합니다. 애플리케이션의 예상치 못한 수요 증가를 수용할 수 있는 충분한 컴퓨팅 리소스(예: RAM, CPU)를 사용하여 인스턴스를 프로비저닝해야 합니다.
- 목표 복구 시점에 맞춰 백업 보존 기간을 설정합니다.
- 클러스터의 장애 조치를 테스트하여 사용 사례 절차에 소요되는 시간을 파악하세요. 자세한 내용은 Amazon DocumentDB 장애 조치 섹션을 참조하세요.
- 복제본 집합 모드(Amazon DocumentDB에 복제본 세트로 연결 참조)에서 클러스터 엔드포인트(Amazon DocumentDB 엔드포인트 참조)를 사용하여 Amazon DocumentDB 클러스터에 연결함으로써 장애 조치가 애플리케이션에 미치는 영향을 최소화합니다.
- 애플리케이션의 읽기 일관성 요구 사항을 충족하면서 읽기 확장성을 최대화하는 드라이버 읽기 기본 설정을 선택합니다. `secondaryPreferred` 읽기 기본 설정은 복제본 읽기를 활성화하고, 더 많은 작업을 수행할 수 있도록 기본 인스턴스를 비웁니다. 자세한 내용은 읽기 기본 설정 옵션 섹션을 참조하세요.
- 네트워크 및 데이터베이스 오류가 발생하는 경우 복원력을 발휘하도록 애플리케이션을 설계합니다. 일시적인 오류와 지속적인 오류를 구별하기 위해 드라이버의 오류 메커니즘을 사용합니다. 적절한 경우 지수 백오프 메커니즘을 사용하여 일시적인 오류를 다시 시도합니다. 재시도 논리를 구현할 때 애플리케이션에서 데이터 일관성을 고려해야 합니다.
- 모든 프로덕션 클러스터 또는 중요한 데이터가 있는 클러스터에 대해 클러스터 삭제 보호를 활성화합니다. Amazon DocumentDB 클러스터를 삭제하기 전에 최종 스냅샷을 생성합니다. 를 사용하여 리소스를 배포하는 경우 종료 방지를 CloudFormation활성화합니다. 자세한 내용은 종료 방지 및 삭제 방지 섹션을 참조하세요.
- Amazon DocumentDB 클러스터를 생성할 때 `--engine-version`은 기본적으로 최신 주요 엔진 버전에 대해 기본적으로 설정되는 파라미터 옵션입니다. 현재 기본 엔진 버전은 5.0.0입니다(참고: Amazon DocumentDB 8.0을 사용할 수 있지만 명시적으로 지정해야 함). 새 주요 엔진 버전이 출시

되면 최신 주요 엔진 버전을 반영하도록 `--engine-version`의 주요 엔진 버전이 업데이트됩니다. 따라서 프로덕션 워크로드, 특히 스크립팅, 자동화 또는 CloudFormation 템플릿에 의존하는 워크로드의 경우를 의도한 메이저 버전`--engine-version`에 명시적으로 지정합니다.

인스턴스 크기 조정

Amazon DocumentDB에서 인스턴스 크기를 선택할 때 가장 중요한 요소 중 하나는 캐시에 사용할 RAM 용량입니다. Amazon DocumentDB는 RAM의 3분의 1을 자체 서비스를 위해 예약합니다. 즉, 캐시에 사용할 수 있는 인스턴스 RAM은 3분의 2뿐입니다. 작업 집합(즉, 데이터 및 인덱스)을 메모리에 저장할 수 있는 충분한 RAM이 있는 인스턴스 유형을 선택하는 것이 Amazon DocumentDB 성능면에서 가장 좋습니다. 인스턴스 크기를 적절히 조정하면 전반적인 성능을 최적화하고 잠재적으로 I/O 비용을 최소화하는 데 도움이 됩니다. 타사 Amazon DocumentDB 크기 계산기를 사용하여 특정 워크로드의 인스턴스 크기를 추정할 수 있습니다.

애플리케이션의 작업 집합이 메모리에 적합한지 확인하려면 부하가 걸린 클러스터의 각 인스턴스별로 Amazon CloudWatch를 사용하여 `BufferCacheHitRatio`를 모니터링합니다.

`BufferCacheHitRatio` CloudWatch 지표는 스토리지 볼륨 대비 인스턴스의 메모리 캐시에서 제공되는 데이터 및 인덱스의 비율을 측정합니다. 일반적으로 작업 집합 메모리에서 데이터를 읽는 것이 저장소 볼륨에서 읽는 것보다 빠르고 비용 효율적이므로 `BufferCacheHitRatio`의 값은 가능한 한 높아야 합니다. `BufferCacheHitRatio`를 가능한 한 100%에 가깝게 유지하는 것이 바람직하지만 달성 가능한 최상의 가치는 애플리케이션의 액세스 패턴과 성능 요구 사항에 따라 달라집니다. `BufferCacheHitRatio`를 가능한 한 최대로 유지하려면 인덱스와 작업 데이터 세트가 메모리에 맞을 수 있을 만큼 충분한 RAM으로 클러스터의 인스턴스를 프로비저닝하는 것이 좋습니다.

인덱스가 메모리에 맞지 않으면 `BufferCacheHitRatio`가 더 낮게 표시될 수 있습니다. 디스크에서 계속 읽으면 추가 I/O 비용이 발생하며 메모리에서 읽는 것보다 좋은 성능을 얻을 수 없습니다. `BufferCacheHitRatio` 비율이 예상보다 낮으면 클러스터가 메모리의 작업 집합 데이터에 맞게 더 많은 RAM을 제공할 수 있도록 인스턴스 크기를 늘립니다. 인스턴스 클래스를 확장하여 `BufferCacheHitRatio`가 급격히 증가하면 애플리케이션의 작업 집합이 메모리에 맞은 것입니다. 규모 조정 작업 후 `BufferCacheHitRatio`가 더 이상 대폭 증가하지 않을 때까지 계속 확장합니다. 인스턴스의 지표 모니터링에 대한 자세한 내용은 Amazon DocumentDB 지표 섹션을 참조하세요.

워크로드 및 지연 시간 요구 사항에 따라 애플리케이션이 정상 상태 사용 중에는 더 높은 `BufferCacheHitRatio` 값을 가질 수 있지만 전체 컬렉션을 스캔해야 하는 분석 쿼리가 인스턴스에서 실행되므로 주기적으로 `BufferCacheHitRatio` 딥을 수행할 수 있습니다. `BufferCacheHitRatio`의 이러한 주기적 딥은 스토리지 볼륨에서 작업 집합 데이터를 버퍼 캐시로 다시 채워야 하는 후속 쿼리에 대해 더 높은 지연 시간으로 나타날 수 있습니다. 먼저 대표적인 프로덕

션 워크로드를 사용하여 사전 프로덕션 환경에서 워크로드를 테스트하여 성능 특성을 이해하고 프로덕션에 워크로드를 배포**BufferCacheHitRatio**합니다.

BufferCacheHitRatio는 인스턴스별 지표이므로 기본 인스턴스와 복제본 인스턴스 간에 읽기가 분산되는 방식에 따라 동일한 클러스터 내의 여러 인스턴스가 서로 다른 BufferCacheHitRatio 값을 가질 수 있습니다. 작업 워크로드가 분석 쿼리를 실행한 후 작업 집합 캐시를 다시 채울 때 발생하는 대기 시간 증가를 주기적으로 처리할 수 없는 경우 일반 작업 워크로드의 버퍼 캐시를 분석 질의의 버퍼 캐시와 분리해야 합니다. 운영 쿼리를 기본 인스턴스로, 분석 쿼리는 복제본 인스턴스로만 전송하여 BufferCacheHitRatio를 완전히 격리할 수 있습니다. 또한 일정 비율의 일반 쿼리가 해당 복제본에서 실행되고 영향을 받을 수 있음을 이해하고 분석 쿼리를 특정 복제본 인스턴스로 전송하여 부분 격리를 수행할 수도 있습니다.

적합한 BufferCacheHitRatio 값은 사용 사례 및 애플리케이션 요구 사항에 따라 다릅니다. 이 지표에는 최고 또는 최소 값이 하나도 없습니다. 비용 및 성능 측면에서 일시적으로 낮은 BufferCacheHitRatio의 절충이 허용되는지 여부만 결정할 수 있습니다.

인덱스 작업

인덱스 빌드

Amazon DocumentDB로 데이터를 가져올 때, 대규모 데이터 세트를 가져오기 전에 인덱스를 생성해야 합니다. Amazon DocumentDB 인덱스 도구를 사용하여 실행 중인 MongoDB 인스턴스 또는 mongodump 디렉터리에서 인덱스를 추출하고, Amazon DocumentDB 클러스터에서 해당 인덱스를 생성할 수 있습니다. 마이그레이션에 대한 자세한 지침은 Amazon DocumentDB 마이그레이션 및 업그레이드 섹션을 참조하세요.

인덱스 선택성

인덱스 생성을 중복 값 수가 컬렉션의 총 문서 수의 1% 미만인 필드로 제한합니다. 예를 들어 컬렉션에 100,000개의 문서가 포함되어 있는 경우, 동일한 값이 1000회 이하로 나타나는 필드에만 인덱스를 생성하세요.

고유 값이 많은 인덱스를 선택하면(즉, 카디널리티가 높음) 필터 작업에서 적은 수의 문서를 반환하므로 인덱스 스캔 중에 우수한 성능을 얻을 수 있습니다. 높은 카디널리티 인덱스의 예는 같음 조건자가 최대 하나의 문서를 반환하도록 보장하는 고유 인덱스입니다. 낮은 카디널리티의 예로는 부울 필드에 대한 인덱스와 요일에 대한 인덱스가 있습니다. 성능 저하로 인해 낮은 카디널리티 인덱스는 데이터베이스의 쿼리 최적화 프로그램에 의해 선택될 가능성이 낮습니다. 동시에 낮은 카디널리티 인덱스는 디스크 스페이스 및 I/O와 같은 리소스를 계속 사용합니다. 일반적으로 일반적인 값 빈도가 전체 컬렉션 크기의 1% 이하인 필드의 인덱스를 대상으로 해야 합니다.

또한 일반적으로 필터로 사용되는 필드에만 인덱스를 생성하고, 사용하지 않는 인덱스를 정기적으로 찾는 것이 좋습니다. 자세한 내용은 인덱스 사용량을 분석하고 사용하지 않는 인덱스를 식별하려면 어떻게 해야 하나요? 섹션을 참조하세요.

인덱스가 데이터 작성에 미치는 영향

인덱스는 컬렉션의 모든 문서를 스캔할 필요가 없으므로 쿼리 성능을 향상할 수 있지만 이러한 개선 시에 다른 불리함이 따를 수 있습니다. 컬렉션의 각 인덱스에 대해 문서가 삽입, 업데이트 또는 삭제될 때마다 데이터베이스는 컬렉션을 업데이트하고 컬렉션의 각 인덱스에 필드를 기록해야 합니다. 예를 들어 컬렉션에 9개의 인덱스가 있는 경우 데이터베이스는 클라이언트에 작업을 승인하기 전에 10개의 쓰기를 수행해야 합니다. 따라서 인덱스가 추가될 때마다 쓰기 지연 시간, I/O가 추가되고 전체 활용도가 높은 스토리지가 증가합니다.

모든 작업 집합 메모리를 유지하려면 클러스터 인스턴스의 크기를 적절하게 조정해야 합니다. 이에 따라 스토리지 볼륨에서 인덱스 페이지를 지속적으로 읽을 필요가 없으므로 성능에 부정적인 영향을 미치고 I/O 비용이 증가합니다. 자세한 내용은 인스턴스 크기 조정 섹션을 참조하세요.

최상의 성능을 얻으려면 일반 쿼리의 성능을 향상하는 데 필요한 인덱스만 추가하여 컬렉션의 인덱스 수를 최소화합니다. 워크로드는 다양하지만 컬렉션당 인덱스 수를 5 이하로 유지하는 것이 좋습니다.

누락된 인덱스 식별

가장 좋은 방법은 정기적으로 누락된 인덱스를 식별하는 것입니다. 자세한 내용은 누락된 인덱스는 어떻게 식별하나요? 섹션을 참조하세요.

사용되지 않는 인덱스 식별

사용하지 않는 인덱스를 정기적으로 식별하고 제거하는 것이 가장 좋습니다. 자세한 내용은 인덱스 사용량을 분석하고 사용하지 않는 인덱스를 식별하려면 어떻게 해야 하나요? 섹션을 참조하세요.

보안 모범 사례

Amazon DocumentDB의 보안 모범 사례는 섹션을 참조하세요Amazon DocumentDB에 대한 보안 모범 사례.

비용 최적화

다음 모범 사례는 Amazon DocumentDB를 사용할 때 비용을 관리하고 최소화하는 데 도움이 될 수 있습니다. 요금 정보는 Amazon DocumentDB(MongoDB 호환) 요금 및 Amazon DocumentDB(MongoDB 호환) FAQ를 참조하세요.

- 해당 월 예상 청구액의 50% 및 75% 임계값으로 결제 알림을 생성합니다. 결제 알림 생성에 대한 자세한 내용은 결제 경보 생성을 참조하세요.
- Amazon DocumentDB의 아키텍처는 스토리지와 컴퓨팅을 분리하므로 단일 인스턴스 클러스터도 매우 내구성이 높습니다. 클러스터 스토리지 볼륨은 3개의 가용 영역에 걸쳐 6방향으로 데이터를 복제하므로, 클러스터에 있는 인스턴스 수와 상관없이 매우 높은 내구성을 제공합니다. 일반적인 프로덕션 클러스터에는 고가용성을 제공하기 위해 3개 이상의 인스턴스가 있습니다. 그러나 고가용성이 필요하지 않은 경우 단일 인스턴스 개발 클러스터를 사용하여 비용을 최적화할 수 있습니다.
- 개발 및 테스트 시나리오의 경우 더 이상 필요하지 않을 때 클러스터를 중지하고 개발이 다시 시작되면 클러스터를 시작합니다. 자세한 내용은 Amazon DocumentDB 클러스터 중지 및 시작 섹션을 참조하세요.
- 데이터가 쓰기, 읽기 및 삭제될 때 TTL과 스트림 변경은 모두 I/O를 발생합니다. 이러한 기능을 활성화했지만 애플리케이션에서 사용하지 않는 경우 이 기능을 비활성화하면 비용이 절감될 수 있습니다.

지표를 통해 성능 문제 식별

주제

- 성능 지표 보기
- CloudWatch 경보 설정
- 성능 지표 평가
- CloudWatch 지표를 사용한 Amazon DocumentDB 인스턴스 사용량 평가
- 쿼리 튜닝

리소스 부족이나 기타 일반적인 병목 현상으로 인해 발생하는 성능 문제를 식별하기 위해 Amazon DocumentDB 클러스터에서 사용 가능한 지표를 모니터링할 수 있습니다.

성능 지표 보기

다양한 기간 동안의 평균, 최대, 최소 측정값을 보려면 성능 지표를 정기적으로 모니터링합니다. 이렇게 하면 성능이 저하된 시점을 식별하는 데 도움이 됩니다. 또한 특정 지표 임계값에 대해 Amazon CloudWatch 경보를 설정하여 해당 임계값에 이를 경우 알리도록 할 수 있습니다.

성능 문제를 해결하기 위해서는 시스템의 기준 성능을 파악해야 합니다. 새 클러스터를 설정하고 일반 워크로드로 실행한 후 서로 다른 간격(예: 1시간, 24시간, 1주, 2주)으로 모든 성능 지표의 평균값, 최댓값 및 최솟값을 파악하세요. 이렇게 하면 무엇이 정상인지를 알 수 있습니다. 이렇게 하면 작업의 최고

피크와 최저 피크 시간을 비교할 수 있습니다. 그런 다음 이 정보를 사용하여 성능이 표준 수준 이하로 떨어진 때를 식별할 수 있습니다.

AWS Management Console 또는를 사용하여 성능 지표를 볼 수 있습니다 AWS CLI. 자세한 내용은 CloudWatch 데이터 보기 섹션을 참조하세요.

CloudWatch 경보 설정

CloudWatch 경보 생성에 대한 자세한 내용을 알아보려면 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch 경보 사용을 참조하세요.

성능 지표 평가

인스턴스에는 여러 지표 범주가 있습니다. 허용되는 값을 결정하는 방법은 지표에 따라 다릅니다.

CPU

- CPU 사용률 – 사용된 컴퓨터 처리 용량의 백분율입니다.

Memory

- 여유 메모리 - 인스턴스에서 사용 가능한 RAM을 나타냅니다.
- 스왑 사용량 - 인스턴스에서 스왑 스페이스를 얼마나 많이 사용했는지를 메가바이트 단위로 나타냅니다.

입력/출력 작업

- IOPS 읽기, IOPS 쓰기 – 초당 디스크 읽기 또는 쓰기 작업의 평균 횟수입니다.
- 읽기 지연 시간, 쓰기 지연 시간 – 읽기 또는 쓰기 작업의 평균 시간(밀리초)입니다.
- 읽기 처리량, 쓰기 처리량 – 초당 디스크에서 읽거나 디스크에 쓴 평균 크기(메가바이트)입니다.
- 디스크 대기열 길이 - 디스크에 쓰기 위해 또는 디스크에서 읽기 위해 대기 중인 I/O 작업의 수입니다.

네트워크 트래픽

- 네트워크 수신 처리량, 네트워크 전송 처리량 - 인스턴스에 대한 수신 또는 전송 네트워크 트래픽 속도(초당 메가바이트)입니다.

데이터베이스 연결

- DB 연결 – 인스턴스에 연결된 클라이언트 세션의 수입니다.

일반적으로 성능 지표에 허용되는 값은 기준이 무엇인지 그리고 애플리케이션 무엇을 수행하는지에 따라 다릅니다. 기준과의 일관된 차이 또는 추세를 조사하세요.

특정 지표 유형에 대한 권장 사항 및 조언은 다음과 같습니다.

- CPU 사용량이 많음 - CPU 사용량이 많을 경우 해당 애플리케이션의 목표(처리량 또는 동시성 등)와 일치하고 예상되는 결과라면 문제가 되지 않을 수 있습니다. CPU 소비가 지속적으로 80%를 초과하는 경우 인스턴스 확장을 고려하세요.
- 높은 RAM 소비 - `FreeableMemory` 메트릭이 총 인스턴스 메모리의 10% 이하로 자주 떨어지는 경우 인스턴스 확장을 고려하세요. DocumentDB 인스턴스의 메모리 사용량이 높을 때 어떤 일이 발생하는지에 대한 자세한 내용은 Amazon DocumentDB 리소스 거버넌스를 참조하세요.
- 스왑 사용량 - 이 메트릭은 0 또는 그 근처에 있어야 합니다. 스왑 사용량이 많은 경우 인스턴스 확장을 고려하세요.
- 네트워크 트래픽 - 네트워크 트래픽의 경우 시스템 관리자에게 문의하여 해당 도메인 네트워크 및 인터넷 연결의 예상 처리량을 확인합니다. 처리량이 기대값보다 항상 낮으면 네트워크 트래픽을 검사합니다.
- 데이터베이스 연결 - 인스턴스 성능 저하 및 응답 시간 지연과 함께 사용자 연결 수가 많을 경우 데이터베이스 연결 제한을 고려해 봅니다. 인스턴스에 대한 최적의 사용자 연결 수는 해당 인스턴스 클래스와 수행하는 작업의 복잡성에 따라 다릅니다. 성능 지표와 관련하여 문제가 있을 경우 성능을 향상하기 위해 제일 먼저 할 수 있는 것은, 가장 많이 사용되고 가장 비용이 높은 쿼리를 튜닝하여 시스템 리소스에 대한 부하를 줄일 수 있는지 확인하는 것입니다.

쿼리를 튜닝해도 문제가 지속된다면 문제와 관련된 리소스(CPU, RAM, 디스크 스페이스, 네트워크 대역폭, I/O 용량)가 많은 Amazon DocumentDB 인스턴스 클래스로 업그레이드하는 것이 좋습니다.

CloudWatch 지표를 사용한 Amazon DocumentDB 인스턴스 사용량 평가

CloudWatch 지표를 사용하여 인스턴스 처리량(throughput)을 관찰하고 인스턴스 클래스가 애플리케이션에 제공하는 리소스가 충분한지 확인할 수 있습니다. 인스턴스 클래스 할당량에 대한 자세한 내용은 섹션을 참조인스턴스 할당량하고 인스턴스 클래스의 사양을 찾아 네트워크 성능을 찾으세요.

인스턴스 사용량이 인스턴스 클래스 한도에 가까워지면 성능이 느려질 수 있습니다. CloudWatch 지표를 통해 이러한 상황을 확인할 수 있으므로 더 큰 인스턴스 클래스로 수동 스케일 업할 계획을 세울 수 있습니다.

다음 CloudWatch 지표 값을 조합하여 인스턴스 클래스 한도에 가까워졌는지 확인하세요.

- NetworkThroughput – Amazon DocumentDB 클러스터의 각 인스턴스에 대해 클라이언트가 수신 및 전송한 네트워크 처리량(throughput)입니다. 이 처리량(throughput) 값에서 클러스터의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.
- StorageNetworkThroughput - Amazon DocumentDB 클러스터의 각 인스턴스가 수신하여 Amazon DocumentDB 클러스터 스토리지 볼륨으로 전송하는 네트워크 처리량입니다.

NetworkThroughput을 StorageNetworkThroughput에 추가하여 Amazon DocumentDB 클러스터의 각 인스턴스가 Amazon DocumentDB에서 수신 및 전송한 네트워크 처리량(throughput)을 확인하세요. 인스턴스에 대한 인스턴스 클래스 한도는 이 두 지표를 합한 값보다 커야 합니다.

다음 지표를 사용하여 전송 및 수신 시 클라이언트 애플리케이션의 네트워크 트래픽에 대한 추가 세부 정보를 검토할 수 있습니다.

- NetworkReceiveThroughput – Amazon DocumentDB 클러스터의 각 인스턴스가 클라이언트로부터 수신한 네트워크 처리량(throughput)입니다. 이 처리량에서 클러스터의 인스턴스와 클러스터 스토리지 볼륨 간 네트워크 트래픽은 제외됩니다.
- NetworkTransmitThroughput – Amazon DocumentDB 클러스터의 각 인스턴스가 클라이언트로 전송한 네트워크 처리량(throughput)입니다. 이 처리량에서 클러스터의 인스턴스와 클러스터 스토리지 볼륨 간 네트워크 트래픽은 제외됩니다.
- StorageNetworkThroughput - 클러스터의 각 인스턴스에 의해 Amazon DocumentDB 클러스터 스토리지 볼륨에서 수신한 네트워크 처리량(throughput)입니다.
- StorageNetworkTransmitThroughput - 클러스터의 각 인스턴스에 의해 Amazon DocumentDB 클러스터 스토리지 볼륨에서 수신한 네트워크 처리량(throughput)입니다.

이러한 지표를 모두 더해 네트워크 사용량을 인스턴스 클래스 한도와 비교하는 방법을 평가합니다. 인스턴스 클래스 한도는 이러한 결합 지표를 합한 값보다 커야 합니다.

인스턴스의 네트워크 한도와 CPU 사용률은 상호적인 관계가 있습니다. 네트워크 처리량(throughput)이 증가하면 CPU 사용률도 증가합니다. CPU 및 네트워크 사용량을 모니터링하면 리소스가 소진되는 방식 및 이유에 대한 정보를 얻을 수 있습니다.

네트워크 사용을 최소화하려면 다음을 고려하면 됩니다.

- 규모가 더 큰 인스턴스 클래스를 사용합니다.
- 쓰기 요청을 일괄적으로 나누어 전체 트랜잭션을 줄입니다.
- 읽기 전용 워크로드를 읽기 전용 인스턴스로 지정합니다.
- 사용하지 않는 인덱스를 삭제합니다.

쿼리 튜닝

클러스터 성능을 향상하는 제일 좋은 방법 중 하나는 일반적으로 가장 많이 사용하는 쿼리와 리소스를 가장 많이 사용하는 쿼리를 튜닝하여 실행 비용을 낮추는 것입니다.

프로파일러(Amazon DocumentDB 작업 프로파일링 참조)를 사용하여 클러스터에서 수행된 작업의 실행 시간 및 세부 정보를 기록할 수 있습니다. 프로파일러는 개별 쿼리 성능 및 전체 클러스터 성능을 개선하기 위해 클러스터에서 가장 느린 작업을 모니터링하는 경우에 유용합니다.

`explain` 명령을 사용하여 특정 쿼리에 대한 쿼리 계획을 분석하는 방법을 배울 수도 있습니다. 쿼리 성능을 높이기 위해 이 정보를 사용하여 쿼리나 기본 컬렉션을 수정할 수 있습니다(예: 인덱스 추가).

TTL 및 시계열 워크로드

TTL 인덱스 만료로 인한 문서 삭제가 최선의 방법입니다. 문서는 특정 기간 내에 삭제될 수 없습니다. 인스턴스 크기, 인스턴스 리소스 사용률, 문서 크기, 전체 처리량, 인덱스 수, 메모리에 인덱스 및 작업 집합이 맞는지 여부와 같은 요소는 모두 만료된 문서가 TTL 프로세스에 의해 삭제되는 시점에 영향을 줄 있습니다.

TTL 모니터가 문서를 삭제하면 삭제할 때마다 IO 비용이 발생하므로 청구 금액이 증가합니다. 처리량과 TTL 삭제율이 증가하면 I/O 사용량 증가로 인해 청구 금액이 더 증가하기 마련입니다. 하지만 TTL 인덱스를 생성하여 문서를 삭제하지 않고 시간을 기준으로 문서를 컬렉션으로 분류하고 더 이상 필요하지 않을 때 해당 컬렉션을 삭제하면 IO 비용이 발생하지 않습니다. 이 방법은 TTL 인덱스를 사용하는 것보다 훨씬 더 비용 효율적일 수 있습니다.

시계열 워크로드의 경우, 롤링 컬렉션이 데이터를 삭제하는 더 좋은 방법일 수 있고 I/O 사용량이 적기 때문에, TTL 인덱스 대신 롤링 컬렉션을 만드는 것이 좋을 수 있습니다. 컬렉션이 크거나(특히 1TB를 초과하는 컬렉션) TTL 삭제 I/O 비용이 우려되는 경우, 문서를 시간에 따라 컬렉션으로 분할하고 문서가 더 이상 필요하지 않을 때 컬렉션을 삭제합니다. 데이터 수집 속도에 따라 하루에 한 번 또는 일주일에 한 번씩 모음을 만들 수 있습니다. 요구 사항은 애플리케이션에 따라 다르지만 몇 개의 큰 컬렉션보다는 더 작은 컬렉션을 사용하는 것이 좋습니다. 이러한 컬렉션을 삭제하는 데는 IO 비용이 발생하지 않으므로 TTL 인덱스를 사용하는 것보다 더 빠르고 비용 효율적일 수 있습니다.

마이그레이션

Amazon DocumentDB로 데이터를 마이그레이션하는 모범 사례로, 먼저 Amazon DocumentDB에서 인덱스를 생성한 다음 데이터를 마이그레이션합니다. 인덱스를 먼저 생성하면 전체 시간을 줄이고 마이그레이션 속도를 높일 수 있습니다. 이렇게 하려면 Amazon DocumentDB 인덱스 도구를 사용하면 됩니다. 마이그레이션에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB 마이그레이션 및 업그레이드.

또한 프로덕션 데이터베이스를 마이그레이션하기 전에 기능, 성능, 운영 및 비용을 고려하여 Amazon DocumentDB에서 애플리케이션을 완전히 테스트하는 것이 좋습니다.

클러스터 파라미터 그룹 작업

프로덕션 클러스터에 변경 사항을 적용하기 전에 테스트 클러스터에서 클러스터 파라미터 그룹 변경 사항을 사용해 보세요. 클러스터 백업에 대한 자세한 내용은 Amazon DocumentDB에서 백업 및 복원 섹션을 참조하세요.

집계 파이프라인 쿼리

여러 단계로 집계 파이프라인 쿼리를 생성하고 쿼리에 있는 데이터의 하위 집합만 평가하는 경우 `$match` 단계를 첫 번째 단계 또는 파이프라인의 시작으로 사용합니다. `$match` 첫 번째를 사용하면 집계 파이프라인 쿼리의 후속 단계에서 처리해야 하는 문서 수가 줄어들어 쿼리 성능이 향상됩니다.

`batchInsert` 및 `batchUpdate`

높은 비율의 동시 `batchInsert` 및/또는 `batchUpdate` 작업을 수행할 때 기본 인스턴스에서 `FreeableMemory`(CloudWatch Metric)의 양이 0이 되는 경우 일괄 삽입 또는 업데이트 워크로드의 동시성을 줄이거나, 워크로드의 동시성을 줄일 수 없는 경우 `FreeableMemory`의 양을 늘리려면 인스턴스 크기를 늘립니다.

Amazon DocumentDB 인덱스 관리

주제

- Amazon DocumentDB 인덱스 생성
- Amazon DocumentDB 인덱스 유지 관리

Amazon DocumentDB 인덱스 생성

Amazon DocumentDB에서 인덱스를 구축하려면 다음과 같이 다양한 결정을 내려야 합니다:

- 얼마나 빨리 완료해야 할까요?
- 빌드가 진행되는 동안 컬렉션에 액세스할 수 없나요?
- 인스턴스의 컴퓨팅 파워 중 빌드에 할당할 수 있는 양은 얼마입니까?
- 어떤 유형의 인덱스를 만들어야 하나요?

이 섹션에서는 이러한 질문에 답하는 데 도움이 되며 인스턴스 기반 클러스터 컬렉션에서 Amazon DocumentDB 인덱스를 생성하는 명령 및 모니터링 예제를 제공합니다.

지침

다음 지침에는 새 인덱스를 생성할 때의 기본 제한 및 구성 장단점이 포함되어 있습니다:

- Amazon DocumentDB 버전 지원 - 단일 작업자 인덱싱은 모든 Amazon DocumentDB 버전에서 지원되지만 여러 작업자 인덱싱은 Amazon DocumentDB 버전 4.0, 5.0 및 8.0에서만 지원됩니다.
- 성능 트레이드오프 - 인덱스 생성 프로세스의 작업자 수를 늘리면 Amazon DocumentDB 데이터베이스의 기본 인스턴스에서 CPU 사용률과 읽기 IO가 증가합니다. 새 인덱스를 생성하는 데 필요한 리소스는 실행 중인 워크로드에 사용할 수 없습니다.
- Elastic 클러스터 - Amazon DocumentDB 엘라스틱 클러스터에서는 병렬 인덱싱이 지원되지 않습니다.
- 서버리스 클러스터 - 서버리스 클러스터의 병렬 인덱싱에는 최소 32DCUs 용량이 필요합니다. 이 임계값 미만에서는 `workers` 파라미터가 허용되지만 인덱스 빌드는 단일 작업자로 실행됩니다. 구성된 최소 용량에 따라 조정되는 최대 병렬 작업자 수: 약 명의 `min_DCU / 8` 작업자를 사용할 수 있습니다. 예를 들어 32DCUs에서는 최대 4개의 작업자를 사용할 수 있고, 64DCUs에서는 최대 8개의 작업자를 사용할 수 있습니다.

  > **Note**
  >
  > 새 병렬 작업자 제한이 적용되려면 최소 용량 설정을 변경한 후 클러스터를 다시 시작해야 합니다.

- 최대 작업자 수(인스턴스 기반) - 구성할 수 있는 최대 작업자 수는 데이터베이스 클러스터의 기본 인스턴스 크기에 따라 달라집니다. 데이터베이스 클러스터의 기본 인스턴스에 있는 총 vCPU 수의 절

반입니다. 예를 들어 vCPU가 64개 있는 db.r6g.16xlarge 인스턴스에서 최대 32개의 작업자를 실행할 수 있습니다.

> **Note**
>
> 병렬 작업자는 인스턴스 2xlarge 이상에서만 지원됩니다.

- 최소 작업자 - 구성할 수 있는 최소 작업자 수는 1개입니다. 인스턴스 기반 클러스터의 인덱스 생성에 대한 기본 설정은 작업자 2명입니다. 하지만 “작업자 스레드” 옵션을 사용하면 작업자 수를 1개로 줄일 수 있습니다. 이렇게 하면 단일 작업자로 프로세스가 실행됩니다.
- 인덱스 압축 - Amazon DocumentDB는 인덱스 압축을 지원하지 않습니다. 저장된 데이터 및 인덱스의 데이터 크기는 다른 옵션을 사용할 때보다 클 수 있습니다.
- 여러 컬렉션 인덱싱 - 데이터베이스 클러스터의 기본 인스턴스에 있는 vCPU의 절반을 구성된 작업자가 여러 컬렉션에서 인덱스를 생성하는 데 사용할 수 있습니다.
- 인덱스 유형 - Amazon DocumentDB에서 지원되는 인덱스 유형에 대한 전체 설명은 AWS 데이터베이스 블로그의 Amazon DocumentDB에서 인덱싱하는 방법을 참조하세요.

시작하기

컬렉션에서 인덱스 생성을 시작하려면 `createIndexes` 명령을 사용합니다. 기본적으로 명령어는 인덱스 생성 프로세스의 속도를 두 배 증가시키는 두 개의 병렬 작업자를 실행합니다.

예를 들어, 다음 명령 프로세스는 문서의 “user_name” 필드에 대한 색인을 만들고 색인 처리 속도를 작업자 4개로 늘리는 방법을 보여줍니다:

1. 클러스터에서 두 개의 병렬 워커를 사용하여 인덱스를 생성합니다.

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx"}]})
```

2. 인덱스 생성 프로세스의 속도를 최적화하려면 `db.runCommand createIndexes` 메소드에서 ‘작업자 스레드’ 옵션(`"workers":<number>`)을 사용하여 워커 수를 지정할 수 있습니다.

   프로세스 속도를 병렬 작업자 4명으로 늘리십시오:

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx", "workers":4}]})
```

> **Note**
>
> 작업자 수가 많을수록 인덱스 생성 진행 속도가 빨라집니다. 하지만 작업자 수가 늘어날수록 기본 인스턴스의 vCPU 및 읽기 IO에 가해지는 부하도 증가합니다. 다른 워크로드를 저하시키지 않으면서 증가된 부담을 감당할 수 있을 만큼 클러스터를 충분히 프로비저닝해야 합니다.

인덱싱 진행 상태

인덱스 생성 프로세스는 컬렉션을 초기화하고, 스캔하고, 키를 정렬하고, 마지막으로 인덱스 빌더를 통해 키를 삽입하는 방식으로 진행됩니다. 프로세스를 포그라운드에서 실행할 경우 최대 6단계, 백그라운드에서 실행할 경우 최대 9단계로 구성됩니다. 단계별로 완료율, 스캔된 저장 블록의 총 수, 정렬된 키 및 삽입된 키와 같은 상태 메트릭을 볼 수 있습니다.

mongo 쉘의 `db.currentOp()` 명령을 사용하여 인덱싱 프로세스의 진행 상황을 모니터링할 수 있습니다. 마지막 단계를 100% 완료하면 모든 색인이 성공적으로 생성되었음을 알 수 있습니다.

```
db.currentOp({"command.createIndexes": { $exists : true } })
```

> **Note**
>
> 인덱싱 진행 상태 보기는 Amazon DocumentDB 5.0에서만 지원됩니다.

인덱스 빌드 유형

인덱스 빌드의 네 가지 유형은 다음과 같습니다.

- 포그라운드 - 포그라운드 인덱스 빌드는 인덱스가 생성될 때까지 다른 모든 데이터베이스 작업을 차단합니다. Amazon DocumentDB 포그라운드 빌드는 5단계로 구성되어 있습니다.
- 포그라운드 (고유) - 단일 문서 (고유) 포그라운드 인덱스 빌드는 일반 포그라운드 빌드와 같은 다른 데이터베이스 작업을 차단합니다. 기본 포그라운드 빌드와 달리 고유 빌드는 추가 단계 (정렬 키 2)를 사용하여 중복 키를 찾습니다. 포그라운드 (고유) 빌드는 6단계로 구성되어 있습니다.
- 백그라운드 - 백그라운드 인덱스 빌드를 사용하면 인덱스를 만드는 동안 다른 데이터베이스 작업을 포그라운드에서 실행할 수 있습니다. Amazon DocumentDB 백그라운드 빌드는 8단계로 구성되어 있습니다.

- 백그라운드 (고유) - 단일 문서 (고유) 배경 인덱스 빌드를 사용하면 인덱스가 생성되는 동안 다른 데이터베이스 작업을 포그라운드에서 실행할 수 있습니다. 기본 백그라운드 빌드와 달리 고유 빌드는 추가 단계 (정렬 키 2) 를 사용하여 중복 키를 찾습니다. 백그라운드 (고유) 빌드는 9단계로 구성되어 있습니다.

인덱스 빌드 단계

단계	포그라운드	포그라운드 (고유)	배경	배경 (고유)
초기화 중	1	1	1	1
빌딩 인덱스: 초기화 중	2	2	2	2
빌딩 인덱스: 컬렉션 스캔	3	3	3	3
빌딩 인덱스: 정렬 키 1	4	4	4	4
빌딩 인덱스: 정렬 키 2		5		5
빌딩 인덱스: 키 삽입	5	6	5	6
검증: 인덱스 스캔			6	7
검증: 튜플 정렬			7	8
검증: 컬렉션 스캔			8	9

- 초기화 중 - createIndex가 인덱스 빌더를 준비하고 있습니다. 이 단계는 매우 간단해야 합니다.
- 인덱스 작성: 초기화 중 - 인덱스 빌더가 인덱스 생성을 준비 중입니다. 이 단계는 매우 간단해야 합니다.

- 인덱스 작성: 컬렉션 스캔 - 인덱스 빌더가 인덱스 키를 수집하기 위해 컬렉션 스캔을 수행합니다. 측정 단위는 “블록”입니다.

> **Note**
>
> 인덱스 빌드에 대해 둘 이상의 작업자가 구성된 경우 이 단계에 표시됩니다. 인덱스 빌드 프로세스 중에 여러 작업자를 사용하는 단계는 “스캐닝 컬렉션” 단계뿐입니다. 다른 모든 단계에는 작업자 한 명만 표시됩니다.

- 인덱스 작성: 정렬 키 1 - 인덱스 빌더가 수집된 인덱스 키를 정렬합니다. 측정 단위는 “키”입니다.
- 인덱스 작성: 정렬 키 2 - 인덱스 작성기가 죽은 튜플에 해당하는 수집된 인덱스 키를 정렬하고 있습니다. 이 단계는 고유한 인덱스 구축에만 존재합니다. 측정 단위는 “키”입니다.
- 인덱스 작성: 키 삽입 - 인덱스 빌더가 새 인덱스에 인덱스 키를 삽입합니다. 측정 단위는 “키”입니다.
- 유효성 검증: 인덱스 스캔 - createIndex는 인덱스를 스캔하여 검증이 필요한 키를 찾는 것입니다. 측정 단위는 “블록”입니다.
- 유효성 검증: 튜플 정렬 - createIndex는 인덱스 스캔 단계의 출력을 정렬합니다.
- 유효성 검증: 컬렉션 스캔 - createIndex는 컬렉션을 스캔하여 이전 두 단계에서 찾은 인덱스 키의 유효성을 검사합니다. 측정 단위는 “블록”입니다.

인덱스 빌드 출력 예제

다음 출력 예제(포그라운드 인덱스 빌드)에는 인덱스 생성 상태가 표시됩니다. “msg” 필드는 빌드의 단계와 완료율을 표시하여 빌드 진행 상황을 요약합니다. “작업자” 필드는 인덱스 빌드의 해당 단계에서 사용된 작업자 수를 나타냅니다. “progress” 필드에는 완료율을 계산하는 데 사용된 실제 수치가 표시됩니다.

> **Note**
>
> Amazon DocumentDB 버전 4.0에서는 “currentIndexBuildName”, “메시지” 및 “진행 상황” 필드가 지원되지 않습니다.

```
{
    "inprog" : [{
    …
        "command": {
```

```
            "createIndexes": "test",
            "indexes": [{
                "v": 2,
                "key": {
                    "user_name": 1
                },
                "name": "user_name_1"
            }],
            "lsid": {
                "id": UUID("094d0fba-8f41-4373-82c3-7c4c7b5ff13b")
            },
            "$db": "test"
        },
        "currentIndexBuildName": user_name_1,
        "msg": "Index Build: building index number_1, stage 6/6 building index:
 656860/1003520 (keys) 65%",
        "workers": 1,
        "progress": {
            "done": 656861,
            "total": 1003520
        },
    …
    ],

    "ok" : 1
}
```

Amazon DocumentDB 인덱스 유지 관리

주제

- 인덱스 팽창
- reIndex를 사용한 인덱스 유지 관리

인덱스 팽창

Amazon DocumentDB는 다중 버전 동시성 제어(MVCC)를 사용하여 동시 트랜잭션을 관리합니다. 문서가 삭제되거나 업데이트되면 이전 버전은 컬렉션 및 인덱스에 '데드' 버전으로 유지됩니다. 폐영역 회수 프로세스는 향후 작업을 위해 이러한 데드 버전에서 공간을 자동으로 회수합니다.

인덱스 팽창은 데드 버전 또는 더 이상 사용되지 않는 인덱스 항목이 누적되거나 페이지 내에 조각화되면서 컬렉션의 인덱스가 커질 때 발생합니다. 보고된 백분율은 향후 인덱스 항목에서 사용할 수 있는

인덱스 공간의 양을 나타냅니다. 이 팽창은 버퍼 캐시와 스토리지 모두에서 공간을 소비합니다. 팽창을 제거하려면 인덱스를 다시 빌드해야 합니다.

Example예제

다음 명령을 실행하여 인덱스의 미사용 스토리지를 확인합니다.

```
db.coll.aggregate({$indexStats:{}});
```

다음과 비슷한 결과가 반환됩니다.

```
{
    "name" : "_id_",
    "key" : {
        "_id" : 1
    },
    "host" : "devbox-test.localhost.a2z.com:27317",
    "size" : NumberLong(827392),
    "accesses" : {
        "ops" : NumberLong(40000),
        "docsRead" : NumberLong(46049),
        "since" : ISODate("2025-04-03T21:44:51.251Z")
    },
    "cacheStats" : {
        "blksRead" : NumberLong(264),
        "blksHit" : NumberLong(140190),
        "hitRatio" : 99.8121
    },
    "unusedStorageSize" : {
        "unusedSizeBytes" : 409600,
        "unusedSizePercent" : 49.51
    }
}
```

전체 컬렉션을 스캔해야 하는 reIndex 명령을 사용하여 가동 중지 없이 인덱스를 다시 빌드할 수 있습니다. reIndex를 사용한 인덱스 유지 관리 섹션을 참조하세요.

reIndex를 사용한 인덱스 유지 관리

reIndex는 인덱스를 재구축하는 데 사용되는 명령입니다. 일반적으로 인덱스가 손상되거나 비효율적일 때 사용됩니다. 시간이 지남에 따라 많은 업데이트, 삽입 또는 삭제로 인해 인덱스에 미사용 스페이

스가 누적되어 성능이 저하될 수 있습니다. 리인덱싱은 사용하지 않는 스페이스를 제거하고 인덱스의 효율성을 복원하는 데 도움이 됩니다.

reIndex 지침

- reIndex는 현재 Amazon DocumentDB 5.0에서만 지원됩니다.
- Amazon DocumentDB는 백그라운드에서 단일 인덱스 reindex를 지원하므로 여러 작업자가 사용할 수 있습니다. 이전 인덱스는 reIndex 프로세스가 실행 중일 때 쿼리에서 사용할 수 있습니다.
- Amazon DocumentDB는 currentOp를 통한 인덱싱 진행률 보고서를 지원합니다. 인덱스 생성 중에 인덱스 빌드 단계 보기와 유사한 인덱스 구축 단계를 볼 수 있습니다. 유일한 차이점은 고유하든 아니든 reIndex에는 항상 8단계가 있다는 것입니다. '빌딩 인덱스: 정렬 키 2' 단계는 없습니다.
- reIndex는 createIndexes, dropIndexes, collMod 및 renameCollection과 같은 동일한 컬렉션에서 인덱스 관련 명령을 제외한 모든 명령과 동시에 실행할 수 있습니다.
- reIndex는 현재 텍스트, 지리 공간, 벡터 및 부분 인덱스에는 지원되지 않습니다.

reIndex 구축

다음 명령을 사용하여 인덱스를 재구축합니다.

```
db.runCommand({ reIndex: "collection-name", index: "index-name"})
```

필요에 따라 재구축 프로세스에 할당된 작업자 수를 제어할 수도 있습니다.

```
db.runCommand({ reIndex: "collection-name", index: "index-name", workers: number })
```

컬렉션 수준 문서 압축 관리

Amazon DocumentDB 5.0 컬렉션 수준 문서 압축을 사용하면 컬렉션의 문서를 압축하여 스토리지 및 IO 비용을 절감할 수 있습니다. 압축된 문서의 스토리지 크기 및 압축 상태와 같은 압축 지표를 통해 스토리지 이득을 측정하여 필요에 따라 컬렉션 수준에서 문서 압축을 활성화하고 압축 지표를 확인할 수 있습니다. Amazon DocumentDB 5.0은 LZ4 압축 알고리즘을 사용하여 문서를 압축합니다.

Amazon DocumentDB는 버전 5.0부터 문서 압축을 지원합니다. 다음 내용은 컬렉션 수준 문서 압축 함수입니다.

- 기본 동작 - 5.0 클러스터의 새 컬렉션에 대한 기본 압축 설정은 클러스터 파라미터에 의해 결정됩니다default_collection_compression. 이 파라미터는 기본적으로 "비활성화"되어 있습니다.

- 기존 컬렉션 압축 - `collMod` 명령을 사용하여 기존 컬렉션의 압축 설정을 변경할 수 있습니다.
- 압축 임계값 변경 - 기본 압축 임계값은 2KB 입니다. 이 값은 `createCollection` 명령을 사용하여 새 컬렉션에 대해 지정하고 `collMod` 명령을 사용하여 기존 컬렉션에서 변경할 수 있습니다.

> **Note**
>
> Amazon DocumentDB 문서 압축은 Amazon DocumentDB 버전 3.6 및 4.0에서 지원되지 않습니다.

주제

- 문서 압축 관리
- 문서 압축 모니터링

문서 압축 관리

컬렉션에서 문서 압축 활성화

`db.createCollection()` 메서드를 사용하여 Amazon DocumentDB 5.0에서 컬렉션을 생성하는 동안 문서 압축을 활성화합니다.

```
db.createCollection( sample_collection,{
    storageEngine : {
        documentDB: {
            compression:{enable: <true | false>}
        }
    }
})
```

클러스터에서 문서 압축 활성화

`default_collection_compression`의 클러스터 파라미터를 "활성화됨"으로 설정하여 클러스터 수준의 모든 새 컬렉션에 대해 문서 압축을 기본적으로 활성화할 수 있습니다. 이 파라미터가 "활성화됨"으로 설정되면 클러스터에서 새로 생성된 컬렉션에는 기본적으로 2KB의 압축 임계값으로 압축이 활성화됩니다.

기존 컬렉션 압축

collMod 작업을 사용하고 다음 storageEngine 구성을 지정하여 기존 컬렉션에 대한 압축 설정을 수정할 수도 있습니다. 이 명령을 사용한 변경 사항은 새로 삽입된/업데이트된 문서에만 적용되며 이전에 삽입된 문서에 대한 압축은 변경되지 않습니다.

```
db.runCommand({
    collMod: "orders",
    storageEngine: {
        documentDB: {compression: {enable: <true | false>} }
    }
})
```

압축 임계값 설정

기본적으로 압축된 컬렉션의 압축 임계값은 2032바이트입니다. 압축이 활성화된 새 컬렉션을 생성할 때 createCollection 명령에서 이 임계값을 설정할 수 있습니다.

```
db.createCollection( sample_collection, {
    storageEngine : {
        documentDB: {
            compression: {
                enable: true,
                threshold: <128 - 8000>
            }
        }
    }
})
```

collMod 작업을 사용하고 다음 storageEngine 구성을 지정하여 기존 압축 컬렉션에 대한 압축 임계값을 수정할 수도 있습니다.

```
db.runCommand({
    collMod: "orders",
    storageEngine: {
        documentDB: {
            compression: {
                enable: true,
                threshold: <128 - 8000>
            }
        }
```

```
    }
})
```

압축 임계값은 128~8000바이트의 값으로만 설정할 수 있습니다. 또한 압축 임계값을 지정할 때 `enable` 옵션을 "true"로 설정해야 합니다.

문서 압축 모니터링

컬렉션이 압축되었는지 확인하고 다음과 같이 압축 비율을 계산할 수 있습니다.

mongo 쉘에서 `db.printCollectionStats()` 또는 `db.collection.stats()` 명령을 실행하여 압축 통계를 봅니다. 출력은 문서 압축의 스토리지 이득을 분석하기 위해 비교할 수 있는 기존 크기와 압축 크기를 보여줍니다. 이 예제에서는 "sample_collection"이라는 컬렉션에 대한 통계가 아래에 나와 있습니다. 1024*1024의 스케일링 인수는 아래에서 `size` 및 `storageSize` 값을 MB 단위로 출력하는 데 사용됩니다.

```
db.sample_collection.stats(1024*1024)
```

다음은 위의 명령에 대한 출력 예제입니다.

```
{
    "ns" : "test.sample_collection",
    "count" : 1000000,
    "size" : 3906.3,
    "avgObjSize" : 4096,
    "storageSize" : 1953.1,
    compression:{"enabled" : true,"threshold" : 2032},
    ...
}
```

- size - 문서 컬렉션의 원래 크기입니다.
- avgObjSize - 압축 전 평균 문서 크기를 소수점 첫째 자리까지 반올림한 값입니다. 측정 단위는 바이트입니다.
- storageSize - 압축 후 컬렉션의 스토리지 크기입니다. 측정 단위는 바이트입니다.
- 활성화됨 - 압축이 활성화 또는 비활성화되었는지 여부를 나타냅니다.

실제 압축 비율을 계산하려면 수집 크기를 스토리지 크기(크기/storageSize)로 나눕니다. 위의 예제에서 계산은 3906.3/1953.1이며, 이는 2:1 압축 비율로 변환됩니다.

Amazon DocumentDB 8.0에서 사전 기반 압축 관리

Amazon DocumentDB 8.0은 LZ4의 개선된 대안으로 zstd라는 새로운 문서 압축 알고리즘을 도입합니다. 압축 알고리즘으로 Zstd를 선택하여 Amazon DocumentDB 8.0 컬렉션에서 사전 압축을 활성화하면 컬렉션의 문서가 샘플링되어 사용자 지정 압축 사전을 생성합니다. 새 문서와 업데이트된 문서는이 사전과 zstd 알고리즘을 사용하여 압축됩니다. 이 접근 방식은 일반적으로 표준 압축 방법보다 높은 압축 비율을 달성합니다. 특히 문서 스키마가 일관되거나 필드 이름이 반복되는 컬렉션의 경우 더욱 그렇습니다.

Lz4는 빠른 압축 및 압축 해제를 위해 설계된 알고리즘입니다. 눈에 띄는 압축을 달성하면서 CPU가 가볍다고 알려져 있습니다. Zstd는 기본 설정으로 더 많은 CPU를 사용하지만 lz4보다 더 나은 압축 비율을 달성하는 범용 알고리즘입니다. 사전을 사용하면 대부분의 JSON 문서에 대한 압축이 더욱 향상됩니다. Zstd 알고리즘의 몇 가지 이점은 다음과 같습니다.

- 스토리지 비용 절감: 압축률이 높을수록 스토리지 사용량이 줄어들고 비용이 절감됩니다.
- 낮은 I/O: 압축된 문서에는 I/O가 적게 필요하므로 성능이 향상될 수 있습니다.
- 컬렉션에 최적화: 사전은 컬렉션의 데이터 패턴에 맞게 특별히 훈련됩니다.

> **Note**
>
> Amazon DocumentDB 버전 3.6, 4.0 및 5.0에서는 사전 기반 압축이 지원되지 않습니다.

성능 고려 사항

Zstd 압축에는 다음과 같은 단점이 있습니다.

- 스토리지와 CPU 비교: Zstd 압축은 더 나은 압축 비율을 달성하지만 LZ4 압축에 비해 약간 더 많은 CPU 리소스를 사용할 수 있습니다.
- 초기 압축: 효과적인 사전을 훈련하기에 충분한 문서가 삽입될 때까지 새 컬렉션이 최적의 압축을 달성하지 못할 수 있습니다. 현재 모음이 100개 이상의 문서인 경우 사전이 훈련됩니다.
- 워크로드 유형: 전체 데이터가 버퍼 캐시에 맞는 읽기 집약적인 워크로드는 압축 해제 오버헤드로 인해 지연 시간과 CPU 사용량이 증가할 수 있습니다.

Zstd 압축은 작은 문서, 문서 배열 및 반복되는 필드 이름이 있는 컬렉션에 특히 효과적입니다.

사전 기반 압축 활성화

새 컬렉션의 경우 아래 명령을 사용하여 Zstd 압축을 활성화할 수 있습니다.

```
db.createCollection("myCollection",
    {
        storageEngine: {
            documentDB: {
                compression: {
                    enable: true,
                    algorithm: "zstd"
                }
            }
        }
    }
 )
```

기존 컬렉션에서 압축을 활성화하거나 수정할 수도 있습니다.

```
db.runCommand({
    collMod: "myCollection",
    storageEngine: {
        documentDB: {
            compression: {
                enable: true,
                algorithm: "zstd"
            }
        }
    }
 })
```

클러스터의 모든 컬렉션에서 Zstd 알고리즘을 활성화하려면 클러스터 파라미터 그룹을 수정하여 파라미터 “default_collection_compression”의 값으로 “zstd”를 선택할 수 있습니다.

시작하기

Amazon DocumentDB 8.0에는 기본적으로 Zstd 압축이 켜져 있습니다. 클러스터 파라미터 그룹에서 'default_compression' 값을 비활성화로 설정하여 항상 끌 수 있습니다. Amazon DocumentDB 8.0부터 'enabled'는 default_compression에 더 이상 유효한 선택이 아니며 Zstd 및 LZ4에서 선택해야 합니다.

모니터링

다음 명령 중 하나를 사용하여 컬렉션에 대한 압축 정보를 볼 수 있습니다.

- db.runCommand({ collStats: "myCollection" }) 또는
- db.collection.stats()

이러한 명령은 압축 비율을 계산하는 데 사용할 수 있는 키 통계를 반환합니다.

- compression.algorithm: 사용된 알고리즘("lz4" 또는 "zstd")
- storageSize: 압축 후 컬렉션에서 사용하는 실제 스토리지입니다. 이 숫자에는 조각화(즉, 데이터베이스 페이지의 미사용 공간)가 포함됩니다.
- avgObjSize: 컬렉션 문서의 평균 논리적 크기를 압축 해제합니다. 컬렉션에 20,000개 이상의 문서가 있는 경우이 숫자는 20,000개 문서의 샘플을 기반으로 한 근사치입니다.
- size: 압축이 없는 컬렉션의 논리적 크기입니다. 이 숫자는 avgObjSize에 컬렉션의 총 문서 수를 곱하여 구하므로 avgObjSize가 근사치인 경우이 숫자도 근사치가 됩니다.
- count: 컬렉션의 문서 수

사전 기반 압축을 평가하는 동안 다음 CloudWatch 지표가 유용할 수 있습니다.

- CPUUtilization
- FreeableMemory
- VolumeBytesUsed
- VolumeReadIOPs
- VolumeWriteIOPs

collStats 지표:

- storageSize

- size

또한 쿼리 유형 또는 API당 지연 시간 및 처리량과 같은 애플리케이션별 지표를 추적하는 것이 유용할 수 있습니다.

Amazon DocumentDB의 보안

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 이 설명서는 Amazon DocumentDB를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 공동 책임 모델은 이를 클라우드의 보안과 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 - AWS 는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다.는 안전하게 사용할 수 있는 서비스 AWS 도 제공합니다. 서드 파티 감사원은 정기적으로 AWS 규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 검증합니다. Amazon DocumentDB(MongoDB 호환)에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램에 대한 범위의AWS 서비스를 참조하십시오.
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 데이터의 민감도, 조직의 요건 및 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

Amazon DocumentDB는 연방정 위험 및 인증 관리 프로그램(FedRAMP)에 따라 권한이 부여됩니다. 여기에는 AWS GovCloud(미국) 리전에 대한 FedRAMP High 권한 부여와 AWS 미국 동부/서부 리전에 대한 FedRAMP Moderate 권한이 있습니다. AWS 및 규정 준수 노력에 대한 자세한 내용은 AWS 규정 준수 프로그램 제공 범위 내 서비스를 참조하세요.

> **Note**
>
> 이 장은 인스턴스 기반 클러스터와 탄력적 클러스터에 모두 적용됩니다. 자세한 내용은 아래 주제를 참조하십시오.

또한 Amazon DocumentDB 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법을 알아봅니다. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 Amazon DocumentDB를 구성하는 방법을 보여 줍니다.

주제

- Amazon DocumentDB 및를 사용한 암호 관리 AWS Secrets Manager
- Amazon DocumentDB의 데이터 보호
- Amazon DocumentDB의 ID 및 액세스 관리

- 교차 서비스 혼동된 대리인 방지
- IAM 자격 증명을 사용한 인증
- SCRAM-SHA-256을 사용한 인증
- Amazon DocumentDB 사용자 관리
- 역할 기반 액세스 제어를 사용한 데이터베이스 액세스
- Amazon DocumentDB TLS 인증서 업데이트
- Amazon DocumentDB에서의 규정 준수 검증
- Amazon DocumentDB의 복원성
- Amazon DocumentDB의 인프라 보안
- Amazon DocumentDB API 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)
- Amazon DocumentDB에 대한 보안 모범 사례
- Amazon DocumentDB 이벤트 감사
- Amazon VPC 및 Amazon DocumentDB

Amazon DocumentDB 및를 사용한 암호 관리 AWS Secrets Manager

Amazon DocumentDB는 Secrets Manager와 통합되어 클러스터의 기본 사용자 암호를 관리합니다.

주제

- Secrets Manager와 Amazon DocumentDB 통합에 대한 제한 사항
- 를 사용한 기본 사용자 암호 관리 개요 AWS Secrets Manager
- 에서 기본 사용자 암호의 Amazon DocumentDB 관리 적용 AWS Secrets Manager
- Secrets Manager를 사용하여 클러스터의 기본 사용자 암호 관리

Secrets Manager와 Amazon DocumentDB 통합에 대한 제한 사항

다음 기능에서는 Secrets Manager를 사용한 기본 사용자 암호 관리가 지원되지 않습니다.

- Amazon DocumentDB 글로벌 데이터베이스의 일부로 포함된 클러스터
- Amazon DocumentDB 리전 간 읽기 전용 복제본

를 사용한 기본 사용자 암호 관리 개요 AWS Secrets Manager

를 사용하면 데이터베이스 암호를 포함하여 코드의 하드 코딩된 보안 인증 정보를 Secrets Manager에 대한 API 호출로 대체하여 프로그래밍 방식으로 보안 암호를 검색할 AWS Secrets Manager수 있습니다. Secrets Manager에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서를 참조하세요.

Secrets Manager에 데이터베이스 보안 암호를 저장하면 AWS 계정에 요금이 발생합니다. 요금에 대한 자세한 정보는 AWS Secrets Manager 요금을 참조하세요.

다음 작업 중 하나를 수행할 때 Amazon DocumentDB가 Amazon DocumentDB 클러스터의 Secrets Manager에서 기본 사용자 암호를 관리하도록 지정할 수 있습니다.

- 클러스터 생성
- 클러스터 수정

Amazon DocumentDB가 Secrets Manager에서 기본 사용자 암호를 관리하도록 지정하면 Amazon DocumentDB가 암호를 생성하여 Secrets Manager에 저장합니다. 암호와 직접 상호 작용하여 기본 사용자의 보안 인증 정보를 검색할 수 있습니다. 고객 관리형 키를 지정하여 보안 암호를 암호화하거나 Secrets Manager가 제공하는 KMS 키를 사용할 수도 있습니다.

Amazon DocumentDB는 보안 암호 설정을 관리하고 기본적으로 7일마다 보안 암호를 교체합니다. 교체 일정 같은 일부 설정을 수정할 수 있습니다. Secrets Manager에서 비밀을 관리하는 클러스터를 삭제하면 비밀와 관련 메타데이터도 삭제됩니다.

비밀의 보안 인증 정보를 사용하여 클러스터에 연결하려면 Secrets Manager에서 비밀을 검색하면 됩니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 에서 보안 암호 가져오기 AWS Secrets Manager 및 AWS Secrets Manager 보안 암호의 자격 증명과 함께 JDBC를 사용하여 SQL 데이터베이스에 연결을 참조하세요.

에서 기본 사용자 암호의 Amazon DocumentDB 관리 적용 AWS Secrets Manager

IAM 조건 키를 사용하여 AWS Secrets Manager에서 기본 사용자 암호의 Amazon DocumentDB 관리를 적용할 수 있습니다. 다음 정책은 Secrets Manager에서 Amazon DocumentDB가 기본 사용자 암호를 관리하지 않는 한 사용자가 인스턴스 또는 클러스터를 생성하거나 복원하는 것을 허용하지 않습니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "rds:CreateDBCluster"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "rds:ManageMasterUserPassword": false
                }
            }
        }
    ]
}
```

Secrets Manager를 사용하여 클러스터의 기본 사용자 암호 관리

다음 작업을 수행할 때 Secrets Manager에서 기본 사용자 암호의 Amazon DocumentDB 관리를 구성할 수 있습니다.

- Amazon DocumentDB 클러스터 생성
- Amazon DocumentDB 클러스터 수정

Amazon DocumentDB 콘솔 또는를 사용하여 이러한 작업을 수행할 수 AWS CLI 있습니다.

Amazon DocumentDB의 데이터 보호

AWS 공동 책임 모델 Amazon DocumentDB의 데이터 보호에 적용됩니다. 이 모델에 설명된 대로 AWS 는 모든를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 클라우드. 사용자는 이 인프라에 호스팅되는 콘텐츠에 대한 통제 권한을 유지할 책임이 있습니다. 사용하는 AWS 서비스 의 보안 구성과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 일반 데이터 보호 규정 (GDPR) 센터를 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM)를 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용합니다.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail. CloudTrail 추적을 사용하여 AWS 활동을 캡처하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 추적 작업을 참조하세요.
- 내의 모든 기본 보안 제어와 함께 AWS 암호화 솔루션을 사용합니다 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용합니다.
- 명령줄 인터페이스 또는 API를 AWS 통해에 액세스할 때 FIPS 140-3 검증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 연방 정보 처리 표준(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 Amazon DocumentDB 또는 기타 AWS 서비스 에서 콘솔 AWS CLI, API 또는 AWS SDKs를 사용하여 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명을 URL에 포함해서는 안 됩니다.

주제

- 클라이언트 측 필드 레벨 암호화
- Amazon DocumentDB 저장 데이터 암호화
- 전송 중 데이터 암호화
- 키 관리

클라이언트 측 필드 레벨 암호화

Amazon DocumentDB 클라이언트 측 필드 레벨 암호화(FLE)를 사용하면 Amazon DocumentDB 클러스터로 전송하기 전에 클라이언트 애플리케이션의 민감한 데이터를 암호화할 수 있습니다. 민감한 데

이터는 클러스터에서 저장 및 처리될 때 암호화된 상태로 유지되며 검색 시 클라이언트 애플리케이션에서 복호화됩니다.

주제

- 시작하기
- 클라이언트 측 FLE의 쿼리

시작하기

Amazon DocumentDB에서 클라이언트측 FLE의 초기 구성은 암호화 키 생성, 애플리케이션에 역할 연결, 애플리케이션 구성, 암호화 옵션을 통한 CRUD 작업 정의 등을 포함하는 4단계 프로세스입니다.

주제

- 1단계: 암호화 키 생성하기
- 2단계: 역할을 애플리케이션에 연결하기
- 3단계: 애플리케이션 구성
- 4단계: CRUD 작업 정의하기
- 예: 클라이언트 측 필드 수준 암호화 구성 파일

1단계: 암호화 키 생성하기

를 사용하여 민감한 데이터 필드를 암호화 및 해독하는 대칭 고객 관리형 키를 AWS Key Management Service생성하고 필요한 IAM 사용 권한을 부여합니다.는 데이터 키를 암호화하는 데 사용되는 고객 관리형 키를 AWS KMS 저장합니다. 고객 관리형 키를에 저장하면 보안 태세가 AWS KMS 강화됩니다. 데이터 키는 보조 키로, Amazon DocumentDB 컬렉션에 저장되며 Amazon DocumentDB에 문서를 저장하기 전에 민감한 필드를 암호화하는 데 필요합니다. 고객 관리형 키는 데이터 키를 암호화하여 데이터를 암호화하고 해독합니다. 글로벌 클러스터를 사용하는 경우 여러 서비스 역할이 서로 다른 리전에서 사용할 수 있는 다중 리전 키를 생성할 수 있습니다.

키 생성 방법을 AWS Key Management Service포함하여에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS Key Management Service 개념을 참조하세요.

2단계: 역할을 애플리케이션에 연결하기

적절한 AWS KMS 권한을 가진 IAM 정책을 생성합니다. 이 정책은 연결된 IAM 자격 증명이 리소스 필드에 지정된 KMS 키를 암호화하고 해독하도록 허용합니다. 애플리케이션은 인증할이 IAM 역할을 수임합니다 AWS KMS.

정책은 다음과 비슷할 것입니다.

```
{ "Effect": "Allow",
"Action": ["kms:Decrypt", "kms:Encrypt"],
"Resource": "Customer Key ARN"
}
```

3단계: 애플리케이션 구성

에서 고객 관리형 키를 정의하고 액세스에 필요한 권한이 있는 IAM 역할을 AWS KMS 생성했습니다. 필수 패키지를 가져옵니다.

```
import boto3
import json
import base64
from pymongo import MongoClient
from pymongo.encryption import (Algorithm,
                                ClientEncryption)
```

```
# create a session object:
my_session = boto3.session.Session()

# get access_key and secret_key programmatically using get_frozen_credentials() method:
current_credentials = my_session.get_credentials().get_frozen_credentials()
```

1. 를 AWS KMS 공급자 유형aws으로 지정하고 이전 단계에서 검색한 계정 자격 증명을 입력합니다.

   ```
   provider = "aws"
   kms_providers = {
       provider: {
           "accessKeyId": current_credentials.access_key,
           "secretAccessKey": current_credentials.secret_key
       }
   }
   ```

2. 고객 관리형 키를 지정하여 데이터 키를 암호화합니다.

   ```
   customer_key = {
   “region”: “AWS region of the customer_key”,
       “key”: “customer_key ARN”
   }
   ```

```
key_vault_namespace = "encryption.dataKeys"

key_alt_name = 'TEST_DATA_KEY'
```

3. MongoClient 개체를 구성합니다.

```
client = MongoClient(connection_string)

coll = client.test.coll
coll.drop()

client_encryption = ClientEncryption(
    kms_providers, # pass in the kms_providers variable from the previous step
    key_vault_namespace = key_vault_namespace,
    client,
    coll.codec_options
)
```

4. 데이터 키를 생성합니다.

```
data_key_id = client_encryption.create_data_key(provider,
    customer_key,
    key_alt_name = [key_alt_name])
```

5. 기존 데이터 키를 검색합니다.

```
data_key = DataKey("aws",
    master_key = customer_key)
key_id = data_key["_id"]
data_key_id = client[key_vault_namespace].find_one({"_id": key_id})
```

4단계: CRUD 작업 정의하기

암호화 옵션을 사용하여 CRUD 작업을 정의합니다.

1. 단일 문서를 작성/읽기/삭제할 컬렉션을 정의합니다.

```
coll = client.gameinfo.users
```

2. 명시적 암호화 - 필드를 암호화하고 다음을 삽입합니다.

> **Note**
>
> “key_id” 또는 “key_alt_name” 중 정확히 하나를 입력해야 합니다.

```
encrypted_first_name = client_encryption.encrypt(
    "Jane",
    Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Deterministic,
    key_id=data_key_id
)
encrypted_last_name = client_encryption.encrypt(
    "Doe",
    Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Deterministic,
    key_id=data_key_id
)
encrypted_dob = client_encryption.encrypt(
    "1990-01-01",
    Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Random,
    key_id=data_key_id
)

coll.insert_one(
    {"gamerTag": "jane_doe90",
    "firstName": encrypted_first_name,
    "lastName": encrypted_last_name,
    "dateOfBirth":encrypted_dob,
    "Favorite_games":["Halo","Age of Empires 2","Medal of Honor"]
})
```

예: 클라이언트 측 필드 수준 암호화 구성 파일

다음 예제에서는 각 ## #### 자신의 정보로 바꿉니다.

```
# import python packages:
import boto3
import json
import base64
from pymongo import MongoClient
from pymongo.encryption import (Algorithm,
                                ClientEncryption)
```

```
def main():

    # create a session object:
    my_session = boto3.session.Session()

    # get aws_region from session object:
    aws_region = my_session.region_name

    # get access_key and secret_key programmatically using get_frozen_credentials()
 method:
    current_credentials = my_session.get_credentials().get_frozen_credentials()
    provider = "aws"

    # define the kms_providers which is later used to create the Data Key:
    kms_providers = {
        provider: {
            "accessKeyId": current_credentials.access_key,
            "secretAccessKey": current_credentials.secret_key
        }
    }

    # enter the kms key ARN. Replace the example ARN value.
    kms_arn = "arn:aws:kms:us-east-1:123456789:key/abcd-efgh-ijkl-mnop"
    customer_key = {
        "region": aws_region,
        "key":kms_arn
    }

    # secrets manager is used to store and retrieve user credentials for connecting to
 an Amazon DocumentDB cluster.
    # retrieve the secret using the secret name. Replace the example secret key.
    secret_name = "/dev/secretKey"
    docdb_credentials = json.loads(my_session.client(service_name = 'secretsmanager',
 region_name = "us-east-1").get_secret_value(SecretId = secret_name)['SecretString'])

    connection_params = '/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false'
    conn_str = 'mongodb://' + docdb_credentials["username"] + ':' +
 docdb_credentials["password"] + '@' + docdb_credentials["host"] + ':' +
 str(docdb_credentials["port"]) + connection_params
    client = MongoClient(conn_str)

    coll = client.test.coll
```

```
    coll.drop()

    # store the encryption data keys in a key vault collection (having naming
 convention as db.collection):
    key_vault_namespace = "encryption.dataKeys"
    key_vault_db_name, key_vault_coll_name = key_vault_namespace.split(".", 1)

    # set up the key vault (key_vault_namespace) for this example:
    key_vault = client[key_vault_db_name][key_vault_coll_name]
    key_vault.drop()
    key_vault.create_index("keyAltNames", unique=True)

    client_encryption = ClientEncryption(
        kms_providers,
        key_vault_namespace,
        client,
        coll.codec_options)

    # create a new data key for the encrypted field:
    data_key_id = client_encryption.create_data_key(provider, master_key=customer_key,
 key_alt_names=["some_key_alt_name"], key_material = None)

    # explicitly encrypt a field:
    encrypted_first_name = client_encryption.encrypt(
    "Jane",
    Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Deterministic,
    key_id=data_key_id
    )
    coll.insert_one(
    {"gamerTag": "jane_doe90",
    "firstName": encrypted_first_name
    })
    doc = coll.find_one()
    print('Encrypted document: %s' % (doc,))

    # explicitly decrypt the field:
    doc["firstName"] = client_encryption.decrypt(doc["firstName"])
    print('Decrypted document: %s' % (doc,))

    # cleanup resources:
    client_encryption.close()
    client.close()

if __name__ == "__main__":
```

```
    main()
```

클라이언트 측 FLE의 쿼리

Amazon DocumentDB는 클라이언트 측 FLE 암호화 값이 있는 등식 쿼리만 지원합니다.

예를 들어 암호화된 gamerscore가 500인 문서를 쿼리하려면 클라이언트는 명시적 암호화 방법을 사용하여 쿼리 값을 암호화합니다.

```
encrypted_gamerscore_filter = client_encryption.encrypt(
    500,
    Algorithm.AEAD_AES_256_CBC_HMAC_SHA_512_Deterministic,
    key_id=data_key_id
)

coll.find( {
    "gamerscore" : { "$eq" : encrypted_gamerscore_filter }
} )
```

Amazon DocumentDB 저장 데이터 암호화

> **Note**
>
> AWS KMS 는 고객 마스터 키(CMK)라는 용어를 AWS KMS key 및 KMS 키로 대체합니다. 단, 개념은 바뀌지 않았습니다. 주요 변경 사항을 방지하기 위해 AWS KMS 는이 용어의 몇 가지 변형을 유지합니다.

클러스터를 생성할 때 스토리지 암호화 옵션을 지정하여 Amazon DocumentDB 클러스터의 저장 데이터를 암호화합니다. 스토리지 암호화는 클러스터 전반에서 활성화되고 기본 인스턴스와 복제본을 포함한 모든 인스턴스에 적용됩니다. 또한 클러스터의 스토리지 볼륨, 데이터, 인덱스, 로그, 자동 백업 및 스냅샷에도 적용됩니다.

Amazon DocumentDB는 256비트 고급 암호화 표준(AES-256)을 사용하여 AWS Key Management Service ()에 저장된 암호화 키를 사용하여 데이터를 암호화합니다AWS KMS. 정지 상태의 암호화를 사용하도록 설정한 상태에서 Amazon DocumentDB 클러스터를 사용할 때는 응용프로그램 로직이나 클라이언트 연결을 수정할 필요가 없습니다. Amazon DocumentDB는 성능에 미치는 영향을 최소화하면서 데이터의 암호화 및 복호화를 투명하게 처리합니다.

Amazon DocumentDB는와 통합 AWS KMS 되며 봉투 암호화라는 메서드를 사용하여 데이터를 보호합니다. Amazon DocumentDB 클러스터가 로 암호화되면 AWS KMS Amazon DocumentDB는 KMS 키를 사용하여 스토리지 볼륨을 암호화하는 사이퍼텍스트 데이터 키를 생성 AWS KMS 하도록 요청합니다. 사이퍼텍스트 키는 사용자가 정의한 KMS 키를 사용하여 암호화되며, 암호화된 데이터 및 저장 메타데이터와 함께 저장됩니다. Amazon DocumentDB는 암호화된 데이터에 액세스해야 하는 경우 KMS 키를 사용하여 암호화 텍스트 데이터 키를 해독 AWS KMS 하도록 요청하고 일반 텍스트 데이터 키를 메모리에 캐시하여 스토리지 볼륨의 데이터를 효율적으로 암호화하고 해독합니다.

Amazon DocumentDB의 스토리지 암호화 기능은 지원되는 모든 인스턴스 크기와 Amazon DocumentDB를 사용할 수 AWS 리전 있는 모든에서 사용할 수 있습니다.

Amazon DocumentDB 클러스터에 대해 중지 상태의 암호화 사용

또는 AWS Command Line Interface ()를 사용하여 클러스터를 프로비저닝할 때 Amazon DocumentDB 클러스터에서 저장 시 암호화를 활성화 AWS Management Console 하거나 비활성화할 수 있습니다 AWS CLI. 콘솔을 사용하여 만든 클러스터는 기본적으로 유휴 데이터 암호화가 활성화되어 있습니다. 를 사용하여 생성하는 클러스터 AWS CLI 에는 기본적으로 유휴 시 암호화가 비활성화되어 있습니다. 따라서 `--storage-encrypted` 파라미터를 사용하여 유휴 데이터 암호화를 명시적으로 활성화해야 합니다. 두 경우 모두 클러스터를 만든 후에는 유휴 데이터 암호화 옵션을 변경할 수 없습니다.

Amazon DocumentDB는 AWS KMS 를 사용하여 암호화 키를 검색 및 관리하고 이러한 키를 사용하는 방법을 제어하는 정책을 정의합니다. AWS KMS 키 식별자를 지정하지 않으면 Amazon DocumentDB는 기본 AWS 관리형 서비스 KMS 키를 사용합니다. Amazon DocumentDB는의 각 AWS 리전 에 대해 별도의 KMS 키를 생성합니다 AWS 계정. 자세한 내용은 AWS Key Management Service 개념을 참조하세요.

자체 KMS 키 생성을 시작하려면 AWS Key Management Service 개발자 안내서의 시작하기를 참조하세요.

> **Important**
>
> Amazon DocumentDB는 대칭 암호화 KMS 키만 지원하므로 클러스터를 암호화하려면 대칭 암호화 KMS 키를 사용해야 합니다. 비대칭 KMS 키를 사용하여 Amazon DocumentDB 클러스터의 데이터를 암호화하지 마세요. 자세한 내용은, AWS Key Management Service 개발자 가이드에서 비대칭 키 AWS KMS를 참조하세요 .

Amazon DocumentDB가 클러스터의 KMS 키에 더 이상 액세스할 수 없는 경우, 예를 들어 키를 소유 AWS 계정 한이 일시 중지되거나, 키가 비활성화되거나, 키 삭제가 예약되거나, Amazon DocumentDB

가 의존하는 키 정책 또는 권한이 제거되는 경우, 클러스터는 먼저 `inaccessible-encryption-credentials-recoverable` 상태로 전환됩니다. 클러스터가이 상태인 동안 Amazon DocumentDB는 클러스터의 인스턴스를 중지하고 클러스터에서 읽거나 쓸 수 없지만 KMS 키에 대한 액세스가 7일 이내에 복원되면 클러스터를 복구할 수 있습니다. 7일 이내에 액세스가 복원되지 않으면 클러스터가 터미널 `inaccessible-encryption-credentials` 상태로 전환됩니다. 터미널 상태에서는 클러스터를 더 이상 사용할 수 없으며 데이터베이스의 현재 상태를 복구할 수 없습니다. 원래 KMS 키를 사용하여 백업에서 복원하거나 point-in-time으로 복원만 수행할 수 있습니다. Amazon DocumentDB의 경우 백업은 항상 최소 1일 동안 활성화됩니다.

> **Note**
>
> 글로벌 클러스터의 일부인 클러스터는 다르게 작동합니다. Amazon DocumentDB가 더 이상 KMS 키에 액세스할 수 없음을 감지하면 글로벌 클러스터의 모든 클러스터가 터미널 `inaccessible-encryption-credentials` 상태로 직접 전환되어 복구 가능 상태를 건너뜁니다. 글로벌 클러스터의 일부인 클러스터는 글로벌 클러스터의 유일한 클러스터인 경우에만 중지하고 시작할 수 있기 때문입니다. 복구하려면 스냅샷에서 복원하거나 point-in-time으로 복원을 수행해야 합니다. 원래 클러스터를 삭제하려면 먼저 글로벌 클러스터에서 각 클러스터를 제거한 다음 삭제해야 합니다.

> **Important**
>
> 암호화된 클러스터의 KMS 키는 이미 만든 후에는 변경할 수 없습니다. 암호화된 클러스터를 생성하기 전에 암호화 키 요구 사항을 결정해야 합니다.

Using the AWS Management Console

클러스터를 생성할 때 유휴 데이터 암호화 옵션을 지정합니다. AWS Management Console을 사용하여 클러스터를 생성할 때 기본적으로 유휴 데이터 암호화가 활성화됩니다. 클러스터를 생성한 후에는 변경할 수 없습니다.

클러스터를 생성할 때 유휴 데이터 암호화 옵션을 지정하려면

1. 시작하기 섹션에 설명된 대로 Amazon DocumentDB 클러스터를 생성합니다 하지만 6단계에서 클러스터 생성을 선택하지 마세요.
2. 인증 섹션 아래에서 고급 설정 표시를 선택합니다.
3. 유휴 상태에서 암호화 섹션으로 아래로 스크롤합니다.

4. 유휴 상태 암호화에 사용할 옵션을 선택합니다. 어떤 옵션을 선택하든 클러스터를 생성한 후에는 변경할 수 없습니다.
   - 이 클러스터에서 저장 데이터를 암호화하려면 암호화 활성화를 선택합니다.
   - 이 클러스터에서 저장 데이터를 암호화하지 않으려면 암호화 비활성화를 선택합니다.
5. 원하는 프라이머리 키를 선택합니다. Amazon DocumentDB는 AWS Key Management Service (AWS KMS)를 사용하여 암호화 키를 검색 및 관리하고 이러한 키를 사용하는 방법을 제어하는 정책을 정의합니다. AWS KMS 키 식별자를 지정하지 않으면 Amazon DocumentDB는 기본 AWS 관리형 서비스 KMS 키를 사용합니다. 자세한 내용은 AWS Key Management Service 개념을 참조하세요.

   > **Note**
   >
   > 암호화된 클러스터를 생성한 후에는 해당 클러스터의 KMS를 변경할 수 없습니다. 암호화된 클러스터를 생성하기 전에 암호화 키 요구 사항을 결정해야 합니다.

6. 필요에 따라 다른 섹션을 완료하고 클러스터를 생성합니다.

Using the AWS CLI

를 사용하여 Amazon DocumentDB 클러스터를 암호화하려면 `create-db-cluster` 명령을 AWS CLI실행하고 `--storage-encrypted` 옵션을 지정합니다. 를 사용하여 생성된 Amazon DocumentDB 클러스터는 기본적으로 스토리지 암호화를 활성화하지 AWS CLI 않습니다.

다음 예제에서는 스토리지 암호화를 사용하도록 설정된 Amazon DocumentDB 클러스터를 만듭니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb create-db-cluster \
  --db-cluster-identifier mydocdbcluster \
  --port 27017 \
  --engine docdb \
  --master-username SampleUser1 \
  --master-user-password primaryPassword \
  --storage-encrypted
```

Windows의 경우:

```
aws docdb create-db-cluster ^
  --db-cluster-identifier SampleUser1 ^
  --port 27017 ^
  --engine docdb ^
  --master-username SampleUser1 ^
  --master-user-password primaryPassword ^
  --storage-encrypted
```

암호화된 Amazon DocumentDB 클러스터를 생성할 때 다음 예제와 같이 AWS KMS 키 식별자를 지정할 수 있습니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb create-db-cluster \
  --db-cluster-identifier SampleUser1 \
  --port 27017 \
  --engine docdb \
  --master-username primaryUsername \
  --master-user-password yourPrimaryPassword \
  --storage-encrypted \
  --kms-key-id key-arn-or-alias
```

Windows의 경우:

```
aws docdb create-db-cluster ^
  --db-cluster-identifier SampleUser1 ^
  --port 27017 ^
  --engine docdb ^
  --master-username SampleUser1 ^
  --master-user-password primaryPassword ^
  --storage-encrypted ^
  --kms-key-id key-arn-or-alias
```

> **Note**
>
> 암호화된 클러스터를 생성한 후에는 해당 클러스터의 KMS를 변경할 수 없습니다. 암호화된 클러스터를 생성하기 전에 암호화 키 요구 사항을 결정해야 합니다.

액세스할 수 없는 암호화 상태에서 Amazon DocumentDB 클러스터 해결

Amazon DocumentDB가 클러스터가 암호화된 KMS 키에 액세스할 수 없는 경우 Amazon DocumentDB 클러스터는 액세스할 수 없는 암호화 상태로 이동합니다. 이러한 상태는 두 가지이며 복구 경로는 클러스터가 있는 상태에 따라 달라집니다.

`inaccessible-encryption-credentials-recoverable` 상태

클러스터가 `inaccessible-encryption-credentials-recoverable` 상태인 동안 KMS 키에 대한 Amazon DocumentDB의 액세스를 복원`available`한 다음 클러스터를 시작하여 클러스터를에 반환할 수 있습니다. 이 상태를 해결하려면 다음을 수행합니다.

1. KMS 키를 소유 AWS 계정 한이 활성 상태인지 확인합니다. 계정이 일시 중지된 경우 계정을 다시 활성화합니다.
2. KMS 키가 활성화되어 있는지 확인합니다. 자세한 내용은 AWS Key Management Service 개발자 안내서의 키 활성화 및 비활성화를 참조하세요.
3. KMS 키 삭제가 예약되어 있는지 확인합니다. 그렇다면 예약된 키 삭제를 취소합니다. 자세한 내용은 AWS Key Management Service 개발자 안내서의 키 삭제 예약 및 취소를 참조하세요.
4. Amazon DocumentDB가 의존하는 KMS 키 정책 및 권한 부여가 여전히 Amazon DocumentDB가 키를 사용하도록 허용하는지 확인합니다.
5. KMS 키에 대한 액세스가 복원된 후를 사용하거나 `start-db-cluster` AWS CLI 명령을 실행 AWS Management Console 하여 클러스터를 시작합니다.

   Example

   Linux, macOS, Unix의 경우:

   ```
   aws docdb start-db-cluster \
     --db-cluster-identifier example-cluster
   ```

   Windows의 경우:

   ```
   aws docdb start-db-cluster ^
     --db-cluster-identifier example-cluster
   ```

> **Important**
>
> 7일 이내에 KMS 키에 대한 액세스가 복원되지 않으면 클러스터는 시작할 수 없는 터미널 `inaccessible-encryption-credentials` 상태로 전환됩니다.

`inaccessible-encryption-credentials` 상태

`inaccessible-encryption-credentials` 상태는 터미널입니다. 클러스터를 시작할 수 없으며 데이터베이스의 실행 상태를 복구할 수 없습니다. 데이터를 복구하려면 스냅샷에서 복원하거나 새 클러스터로 point-in-time으로 복원을 수행합니다. 복원을 수행하려면 원래 KMS 키에 대한 액세스 권한이 있어야 합니다. KMS 키가 삭제된 경우 데이터를 복구할 수 없습니다.

자세한 내용은 클러스터 스냅샷에서 복원 및 특정 시점으로 복원 섹션을 참조하세요.

> **Note**
>
> 글로벌 클러스터의 일부인 클러스터는 글로벌 클러스터의 유일한 클러스터인 경우에만 글로벌 클러스터의 일부인 클러스터를 중지하고 시작할 수 있으므로 KMS 키에 대한 액세스 권한이 손실되면 `inaccessible-encryption-credentials` 상태로 직접 전환됩니다. 이 상태의 클러스터를 삭제하려면 먼저 글로벌 클러스터에서 각 클러스터를 제거한 다음 클러스터를 개별적으로 삭제합니다.

삭제 방지 기능이 활성화되어 있기 때문에 `inaccessible-encryption-credentials` 상태인 클러스터를 삭제할 수 없는 경우 삭제를 다시 시도 AWS CLI 하기 전에를 사용하여 삭제 방지 기능을 끕니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb modify-db-cluster \
  --db-cluster-identifier example-cluster \
  --no-deletion-protection
```

Windows의 경우:

```
aws docdb modify-db-cluster ^
```

```
  --db-cluster-identifier example-cluster ^
  --no-deletion-protection
```

그런 다음 `delete-db-cluster` 명령을 사용하여 클러스터를 삭제할 수 있습니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb delete-db-cluster \
  --db-cluster-identifier example-cluster \
  --skip-final-snapshot
```

Windows의 경우:

```
aws docdb delete-db-cluster ^
  --db-cluster-identifier example-cluster ^
  --skip-final-snapshot
```

위 명령을 실행한 후에도 클러스터가 삭제되지 않으면에 문의하십시오AWS Support.

Amazon DocumentDB 암호화된 클러스터의 제한 사항

Amazon DocumentDB 암호화된 클러스터에는 다음과 같은 제한이 있습니다.

- Amazon DocumentDB 클러스터에 대해 중지된 암호화는 클러스터가 생성된 후가 아니라 생성된 시점에만 사용 가능 또는 불가능으로 설정할 수 있습니다. 그러나 암호화되지 않은 클러스터의 스냅샷을 생성한 다음 암호화 중지 옵션 지정을 통해 암호화되지 않은 스냅샷을 새 클러스터로 복원하여 암호화되지 않은 클러스터의 암호화된 복사본을 생성할 수 있습니다.

  자세한 내용은 다음 항목을 참조하세요.

  - 수동 클러스터 스냅샷 생성
  - 클러스터 스냅샷에서 복원
  - Amazon DocumentDB 클러스터 스냅샷 복사
- 스토리지 암호화를 사용하도록 설정된 Amazon DocumentDB 클러스터는 암호화를 사용하지 않도록 수정할 수 없습니다.
- Amazon DocumentDB 클러스터의 모든 인스턴스, 자동 백업, 스냅샷 및 인덱스는 동일한 KMS로 암호화됩니다.

전송 중 데이터 암호화

전송 계층 보안(TLS)을 사용하여 애플리케이션과 Amazon DocumentDB 클러스터 간의 연결을 암호화할 수 있습니다. 기본적으로 전송 중인 암호화는 새로 생성된 Amazon DocumentDB 클러스터에 대해 활성화됩니다. 클러스터를 생성할 때 또는 나중에 선택적으로 비활성화할 수 있습니다. 전송 중 데이터 암호화가 활성화된 경우 클러스터에 연결하려면 TLS를 사용하는 보안 연결이 필요합니다. TLS를 사용하여 Amazon DocumentDB에 연결하는 방법에 대한 자세한 내용은 Amazon DocumentDB에 프로그래밍 방식으로 연결를 참조하세요.

Amazon DocumentDB 클러스터 TLS 설정 관리

Amazon DocumentDB 클러스터에 대한 전송 중 데이터 암호화는 클러스터 파라미터 그룹의 TLS 파라미터를 통해 관리됩니다. AWS Management Console 또는 AWS Command Line Interface ()를 사용하여 Amazon DocumentDB 클러스터 TLS 설정을 관리할 수 있습니다AWS CLI. 현재 TLS 설정을 확인하고 수정하는 방법은 다음 섹션을 참조하세요.

Using the AWS Management Console

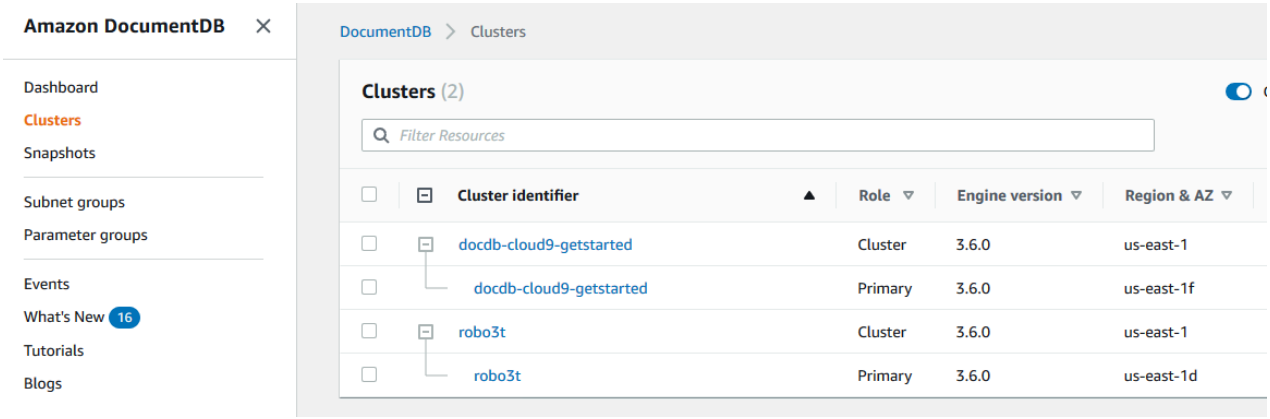
다음 단계에 따라 매개 변수 그룹 식별, TLS 값 확인, 필요한 수정 등 콘솔을 사용하여 TLS 암호화에 대한 관리 작업을 수행합니다.

> **Note**
>
> 클러스터를 생성할 때 다르게 지정하지 않으면 클러스터는 기본 클러스터 파라미터 그룹을 이용해 생성됩니다. `default` 클러스터 파라미터 그룹의 파라미터는 수정할 수 없습니다(예: `tls` 활성화/비활성화). 클러스터가 `default` 클러스터 파라미터 그룹을 사용하는 경우 해당 클러스터를 수정하여 기본이 아닌 클러스터 파라미터 그룹을 사용해야 합니다. 먼저 사용자 지정 클러스터 파라미터 그룹을 생성해야 할 수도 있습니다. 자세한 내용은 Amazon DocumentDB 클러스터 파라미터 그룹 생성 섹션을 참조하세요.

1. 클러스터에서 사용 중인 클러스터 매개 변수 그룹을 확인합니다.
   a. https://console.aws.amazon.com/docdb에서 Amazon DocumentDB 콘솔을 엽니다.
   b. 탐색 창에서 클러스터를 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

c. 참고로 클러스터 탐색 상자의 클러스터 식별자 열에는 클러스터와 인스턴스가 모두 표시됩니다. 인스턴스는 클러스터 아래에 나열됩니다. 다음 스크린샷을 참조하세요.



d. 통합할 클러스터를 선택합니다.

e. 설정 탭을 선택하여 클러스터 디테일 아래를 스크롤 다운하여 클러스터 파라미터 그룹을 위치시키세요. 클러스터 파라미터 그룹의 이름을 적어 둡니다.

클러스터의 파라미터 그룹 이름이 `default`(예: `default.docdb3.6`)인 경우 사용자 지정 클러스터 파라미터 그룹을 생성해야 하며 계속하기 전에 이를 클러스터의 파라미터 그룹으로 지정해야 합니다. 자세한 내용은 다음을 참조하세요:

1. Amazon DocumentDB 클러스터 파라미터 그룹 생성 — 사용할 수 있는 사용자 지정 클러스터 매개 변수 그룹이 없는 경우 생성합니다.
2. Amazon DocumentDB 클러스터 수정 — 사용자 지정 클러스터 매개 변수 그룹을 사용하도록 클러스터를 수정합니다.

2. **`tls`** 클러스터 파라미터의 현재 값을 확인합니다.

a. https://console.aws.amazon.com/docdb에서 Amazon DocumentDB 콘솔로 이동합니다.

b. 탐색 창에서 파라미터 그룹을 선택합니다.

c. 클러스터 파라미터 그룹 목록에서 원하는 클러스터 파라미터 그룹의 이름을 선택합니다.

d. 클러스터 파라미터 섹션을 찾습니다. 클러스터 파라미터 목록에서 `tls` 클러스터 파라미터 행을 찾습니다. 이때 다음 네 개의 열이 중요합니다:

- 클러스터 파라미터 이름 — 클러스터 매개 변수의 이름입니다. TLS를 관리하는 경우 `tls` 클러스터 파라미터를 살펴보게 됩니다.
- 값 — 각 클러스터 매개 변수의 현재 값입니다.
- 허용된 값 — 클러스터 매개 변수에 적용할 수 있는 값 목록입니다.
- 응용 유형 — 정적 또는 동적 타입. 정적 클러스터 파라미터에 대한 변경 사항은 인스턴스를 재부팅할 때에만 적용할 수 있습니다. 동적 클러스터 파라미터에 대한 변경 사항은 즉시 또는 인스턴스를 재부팅할 때 적용할 수 있습니다.

3. **`tls`** 클러스터 파라미터의 값을 수정합니다.

`tls`의 값이 잘못된 경우 이 클러스터 파라미터 그룹에 대한 값을 수정합니다. `tls` 클러스터 파라미터의 값을 변경하려면, 다음 단계에 따라 이전 섹션으로부터 계속합니다.

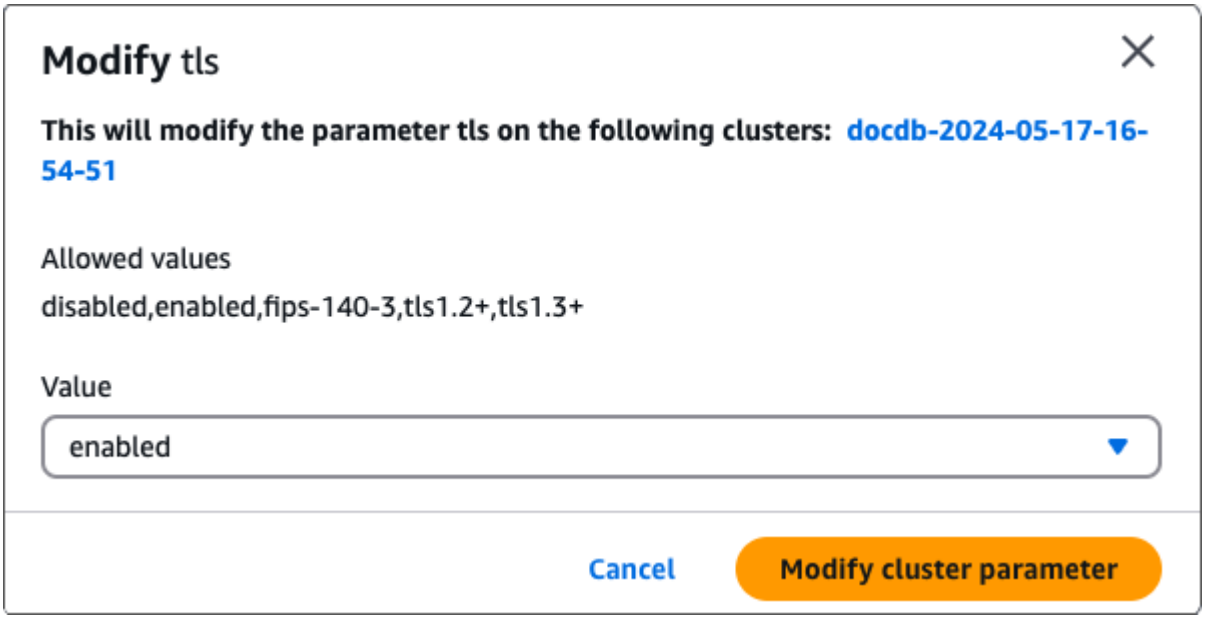
a. 클러스터 파라미터 이름의 왼쪽에 있는 버튼(`tls`)을 선택합니다.

b. 편집을 선택합니다.

c. `tls`의 값을 변경하려면 **`tls`** 수정 대화 상자의 드롭다운 목록에서 클러스터 파라미터로 사용할 값을 선택합니다.

유효한 값은 다음과 같습니다:

- 비활성화 — TLS 비활성화
- enabled - TLS를 활성화합니다. Amazon DocumentDB 5.0 이하의 경우 TLS 1.0~1.3을 활성화하지만 TLS 1.2 이상을 사용합니다. 마이너 버전 5.0.1부터 Amazon DocumentDB 5.0, Amazon DocumentDB 8.0 이상에서는 TLS 1.2 이상만 지원됩니다.
- fips-140-3 — FIPS를 사용하여 TLS를 활성화합니다. 클러스터는 연방 정보 처리 표준(FIPS) 간행물 140-3의 요구 사항에 따라 보안 연결만 허용합니다. 이는 ca-central-1, us-west-2, us-east-1, us-east-2, us-gov-east-1, us-gov-west-1 리전의 Amazon DocumentDB 5.0(엔진 버전 3.0.3727) 클러스터부터만 지원됩니다.
- tls1.2+ - TLS 버전 1.2 이상을 활성화합니다. Amazon DocumentDB 4.0(엔진 버전 2.0.10980) 및 Amazon DocumentDB(엔진 버전 3.0.11051)부터만 지원됩니다.
- tls1.3+ - TLS 버전 1.3 이상을 활성화합니다. Amazon DocumentDB 4.0(엔진 버전 2.0.10980) 및 Amazon DocumentDB(엔진 버전 3.0.11051)부터만 지원됩니다.



d. 클러스터 파라미터 수정을 선택합니다. 재부팅할 때 변경 사항이 각 클러스터 인스턴스에 적용됩니다.

4. Amazon DocumentDB 인스턴스를 재부팅합니다.

   클러스터의 각 인스턴스를 재부팅하여 변경 사항이 클러스터의 모든 인스턴스에 적용되도록 합니다.

   a. https://console.aws.amazon.com/docdb에서 Amazon DocumentDB 콘솔로 이동합니다.
   b. 탐색 창에서 인스턴스를 선택합니다.
   c. 재부팅할 인스턴스를 지정하려면 인스턴스 목록에서 인스턴스를 찾은 다음 이름 왼쪽에 있는 버튼을 선택합니다.
   d. 작업을 선택한 후 재부팅을 선택합니다. 재부팅을 선택하여 재부팅할지를 확인합니다.

Using the AWS CLI

파라미터 그룹 식별, TLS 값 확인, 필요한 수정 등 AWS CLI을 사용하여 TLS 암호화에 대한 관리 작업을 수행하려면 다음 단계를 수행합니다.

> **Note**
>
> 클러스터를 생성할 때 다르게 지정하지 않으면 클러스터는 기본 클러스터 파라미터 그룹을 이용해 생성됩니다. `default` 클러스터 파라미터 그룹의 파라미터는 수정할 수 없습니다

(예: `tls` 활성화/비활성화). 클러스터가 `default` 클러스터 파라미터 그룹을 사용하는 경우 해당 클러스터를 수정하여 기본이 아닌 클러스터 파라미터 그룹을 사용해야 합니다. 먼저 사용자 지정 클러스터 파라미터 그룹을 생성해야 할 수도 있습니다. 자세한 내용은 Amazon DocumentDB 클러스터 파라미터 그룹 생성 섹션을 참조하세요.

1. 클러스터에서 사용 중인 클러스터 파라미터 그룹 확인.

   다음 옵션으로 `describe-db-clusters` 명령을 실행합니다.

   - `--db-cluster-identifier`
   - `--query`

   다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

   ```
   aws docdb describe-db-clusters \
     --db-cluster-identifier mydocdbcluster \
     --query 'DBClusters[*].[DBClusterIdentifier,DBClusterParameterGroup]'
   ```

   이 작업의 출력은 다음과 같습니다(JSON 형식).

   ```
   [
       [
           "mydocdbcluster",
           "myparametergroup"
       ]
   ]
   ```

   클러스터의 파라미터 그룹 이름이 `default`(예: `default.docdb3.6`)인 경우 사용자 지정 클러스터 파라미터 그룹이 있어야 하며 계속하기 전에 이를 클러스터의 파라미터 그룹으로 지정해야 합니다. 자세한 정보는 다음 주제를 참조하세요:

   1. Amazon DocumentDB 클러스터 파라미터 그룹 생성 — 사용할 수 있는 사용자 지정 클러스터 파라미터 그룹이 없다면 만들어야 합니다.
   2. Amazon DocumentDB 클러스터 수정 — 사용자 지정 클러스터 매개 변수 그룹을 사용하도록 클러스터를 수정합니다.

2. **`tls`** 클러스터 파라미터의 현재 값을 확인합니다.

이 클러스터 파라미터 그룹에 대한 자세한 내용은 다음 옵션과 함께 describe-db-cluster-parameters 명령을 사용합니다:

- `--db-cluster-parameter-group-name`
- `--query`

  출력을 `ParameterName`, `ParameterValue`, `AllowedValues` 및 `ApplyType` 관심 필드로만 제한합니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

```
aws docdb describe-db-cluster-parameters \
  --db-cluster-parameter-group-name myparametergroup \
  --query 'Parameters[*].[ParameterName,ParameterValue,AllowedValues,ApplyType]'
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
[
    [
        "audit_logs",
        "disabled",
        "enabled,disabled",
        "dynamic"
    ],
    [
        "tls",
        "disabled",
        "disabled,enabled,fips-140-3,tls1.2+,tls1.3+",
        "static"
    ],
    [
        "ttl_monitor",
        "enabled",
        "disabled,enabled",
        "dynamic"
    ]
]
```

3. **`tls`** 클러스터 파라미터의 값을 수정합니다.

tls의 값이 잘못된 경우 이 클러스터 파라미터 그룹에 대한 값을 수정합니다. tls 클러스터 파라미터의 값을 변경하려면 다음 옵션과 함께 modify-db-cluster-parameter-group 명령을 사용합니다.

- --db-cluster-parameter-group-name — 선택 사항. 수정할 클러스터 파라미터 그룹의 이름입니다. default.* 클러스터 파라미터 그룹은 지정할 수 없습니다.
- --parameters - 필수입니다. 클러스터 파라미터 그룹에서 수정할 파라미터의 목록입니다.
  - ParameterName - 필수입니다. 수정할 클러스터 파라미터의 이름입니다.
  - ParameterValue - 필수입니다. 이 클러스터 파라미터의 새 값입니다. 클러스터 파라미터의 AllowedValues 중 하나이어야 합니다.
    - enabled - 클러스터는 TLS를 사용한 보안 연결을 허용합니다. Amazon DocumentDB 5.0 이하의 경우 TLS 1.0~1.3을 활성화하지만 TLS 1.2 이상을 사용합니다. 마이너 버전 5.0.1부터 Amazon DocumentDB 5.0, Amazon DocumentDB 8.0 이상에서는 TLS 1.2 이상만 지원됩니다.
    - disabled — 클러스터는 TLS를 사용한 보안 연결을 허용하지 않습니다.
    - fips-140-3 — 클러스터는 연방 정보 처리 표준(FIPS) 간행물 140-3의 요구 사항에 따라 보안 연결만 허용합니다. 이는 ca-central-1, us-west-2, us-east-1, us-east-2, us-gov-east-1, us-gov-west-1 리전의 Amazon DocumentDB 5.0(엔진 버전 3.0.3727) 클러스터부터만 지원됩니다.
    - tls1.2+ - 클러스터는 TLS 버전 1.2 이상을 사용한 보안 연결을 허용합니다. Amazon DocumentDB 4.0(엔진 버전 2.0.10980) 및 Amazon DocumentDB 5.0(엔진 버전 3.0.11051)부터만 지원됩니다.
    - tls1.3+ - 클러스터는 TLS 버전 1.3 이상을 사용한 보안 연결을 허용합니다. Amazon DocumentDB 4.0(엔진 버전 2.0.10980) 및 Amazon DocumentDB 5.0(엔진 버전 3.0.11051)부터만 지원됩니다.
  - ApplyMethod — 이 수정이 적용되지 않는 경우. tle 같은 정적 클러스터 파라미터의 경우 이 값이 pending-reboot이어야 합니다.
    - pending-reboot — 인스턴스가 재부팅된 후에만 변경 사항이 인스턴스에 적용됩니다. 이 변경 사항이 클러스터의 모든 인스턴스에 적용되려면 각 클러스터 인스턴스를 개별적으로 재부팅해야 합니다.

다음 예제에서는 각각의 ### ## ## ###를 자신의 클러스터 정보로 바꿉니다.

다음 코드는 `tls`를 비활성화하고, DB 인스턴스가 재부팅될 때 각 인스턴스에 변경사항을 적용합니다.

```
aws docdb modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=tls,ParameterValue=disabled,ApplyMethod=pending-
reboot"
```

다음 코드는를 활성화하여 인스턴스가 재부팅될 때 각 인스턴스에 변경 사항을 `tls`적용합니다.

```
aws docdb modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=tls,ParameterValue=enabled,ApplyMethod=pending-
reboot"
```

다음 코드는 `fips-140-3`이 있는 TLS를 활성화하여, 각 인스턴스가 재부팅될 때 변경 사항을 적용합니다.

```
aws docdb modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup2 \
  --parameters "ParameterName=tls,ParameterValue=fips-140-3,ApplyMethod=pending-
reboot"
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "DBClusterParameterGroupName": "myparametergroup"
}
```

4. Amazon DocumentDB 인스턴스를 재부팅합니다.

   클러스터의 각 인스턴스를 재부팅하여 변경 사항이 클러스터의 모든 인스턴스에 적용되도록 합니다. Amazon DocumentDB 인스턴스를 재부팅하려면 다음 옵션을 실행하여 `reboot-db-instance` 명령을 수행합니다:

   - `--db-instance-identifier`

다음 코드는 mydocdbinstance 인스턴스를 재부팅합니다.

다음 예제에서는 각각의 *### ## ## ###*를 자신의 클러스터 정보로 바꿉니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb reboot-db-instance \
  --db-instance-identifier mydocdbinstance
```

Windows의 경우:

```
aws docdb reboot-db-instance ^
  --db-instance-identifier mydocdbinstance
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "DBInstance": {
        "AutoMinorVersionUpgrade": true,
        "PubliclyAccessible": false,
        "PreferredMaintenanceWindow": "fri:09:32-fri:10:02",
        "PendingModifiedValues": {},
        "DBInstanceStatus": "rebooting",
        "DBSubnetGroup": {
            "Subnets": [
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1a"
                    },
                    "SubnetIdentifier": "subnet-4e26d263"
                },
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1c"
                    },
                    "SubnetIdentifier": "subnet-afc329f4"
```

```
                },
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1e"
                    },
                    "SubnetIdentifier": "subnet-b3806e8f"
                },
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1d"
                    },
                    "SubnetIdentifier": "subnet-53ab3636"
                },
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1b"
                    },
                    "SubnetIdentifier": "subnet-991cb8d0"
                },
                {
                    "SubnetStatus": "Active",
                    "SubnetAvailabilityZone": {
                        "Name": "us-east-1f"
                    },
                    "SubnetIdentifier": "subnet-29ab1025"
                }
            ],
            "SubnetGroupStatus": "Complete",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-91280df6",
            "DBSubnetGroupName": "default"
        },
        "PromotionTier": 2,
        "DBInstanceClass": "db.r5.4xlarge",
        "InstanceCreateTime": "2018-11-05T23:10:49.905Z",
        "PreferredBackupWindow": "00:00-00:30",
        "KmsKeyId": "arn:aws:kms:us-east-1:012345678901:key/0961325d-a50b-44d4-
b6a0-a177d5ff730b",
        "StorageEncrypted": true,
        "VpcSecurityGroups": [
            {
```

```
                "Status": "active",
                "VpcSecurityGroupId": "sg-77186e0d"
            }
        ],
        "EngineVersion": "3.6.0",
        "DbiResourceId": "db-SAMPLERESOURCEID",
        "DBInstanceIdentifier": "mydocdbinstance",
        "Engine": "docdb",
        "AvailabilityZone": "us-east-1a",
        "DBInstanceArn": "arn:aws:rds:us-east-1:012345678901:db:sample-cluster-
instance-00",
        "BackupRetentionPeriod": 1,
        "Endpoint": {
            "Address": "mydocdbinstance.corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
            "Port": 27017,
            "HostedZoneId": "Z2R2ITUGPM61AM"
        },
        "DBClusterIdentifier": "mydocdbcluster"
    }
}
```

인스턴스가 재부팅되는 데 몇 분 정도 걸릴 수 있습니다. 사용 가능 상태인 경우에만 인스턴스를 사용할 수 있습니다. 콘솔 또는 AWS CLI를 사용하여 인스턴스의 상태를 모니터링할 수 있습니다. 자세한 내용은 Amazon DocumentDB 인스턴스 상태 모니터링 섹션을 참조하세요.

키 관리

Amazon DocumentDB는 AWS Key Management Service (AWS KMS)를 사용하여 암호화 키를 검색하고 관리합니다. AWS KMS 는 안전하고 가용성이 높은 하드웨어와 소프트웨어를 결합하여 클라우드에 맞게 확장된 키 관리 시스템을 제공합니다. 를 사용하면 암호화 키를 AWS KMS생성하고 이러한 키를 사용할 수 있는 방법을 제어하는 정책을 정의할 수 있습니다.는를 AWS KMS 지원 AWS CloudTrail하므로 키 사용을 감사하여 키가 적절하게 사용되고 있는지 확인할 수 있습니다.

KMS 키를 Amazon DocumentDB 및 Amazon Simple Storage Service(Amazon S3), Amazon Relational Database Service(RDS), Amazon Elastic Block Store(Amazon EBS) 및 Amazon Redshift와 같은 지원되는 AWS 서비스와 함께 사용할 수 있습니다. 가 지원하는 서비스 목록은 AWS Key Management Service 개발자 안내서의 AWS 서비스 사용 방법을 AWS KMS AWS KMS참조하세요. 에 대한 자세한 내용은 란 무엇입니까 AWS Key Management Service?를 AWS KMS참조하십시오.

Amazon DocumentDB의 ID 및 액세스 관리

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 AWS 서비스 되는 입니다. IAM 관리자는 어떤 사용자가 Amazon DocumentDB 리소스를 사용할 수 있도록 인증(로그인)되고 권한 부여(권한 있음)될 수 있는지 제어합니다. IAM은 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

주제

- 대상
- ID를 통한 인증
- 정책을 사용하여 액세스 관리
- Amazon DocumentDB에서 IAM을 사용하는 방법
- Amazon DocumentDB의 ID 기반 정책 예
- Amazon DocumentDB ID 및 액세스 문제 해결
- Amazon DocumentDB 리소스에 대한 액세스 권한 관리
- Amazon DocumentDB에 대한 ID 기반 정책(IAM 정책) 사용
- AWS Amazon DocumentDB에 대한 관리형 정책
- Amazon DocumentDB API 권한: 작업, 리소스 및 조건 참조

대상

AWS Identity and Access Management (IAM)를 사용하는 방법은 역할에 따라 다릅니다.

- 서비스 사용자 - 기능에 액세스할 수 없는 경우 관리자에게 권한 요청(참조Amazon DocumentDB ID 및 액세스 문제 해결)
- 서비스 관리자 - 사용자 액세스 결정 및 권한 요청 제출(Amazon DocumentDB에서 IAM을 사용하는 방법 참조)
- IAM 관리자 - 액세스를 관리하기 위한 정책 작성(Amazon DocumentDB의 ID 기반 정책 예 참조)

ID를 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여에 로그인하는 방법입니다. AWS 계정 루트 사용자, IAM 사용자 또는 IAM 역할을 수임하여 인증되어야 합니다.

AWS IAM Identity Center (IAM Identity Center), Single Sign-On 인증 또는 Google/Facebook 자격 증명과 같은 자격 증명 소스의 자격 증명을 사용하여 페더레이션 자격 증명으로 로그인할 수 있습니다. 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 AWS 계정에 로그인하는 방법 섹션을 참조하세요.

프로그래밍 방식 액세스를 위해는 요청에 암호화 방식으로 서명할 수 있는 SDK 및 CLI를 AWS 제공합니다. 자세한 내용은 IAM 사용 설명서의 API 요청용AWS Signature Version 4 섹션을 참조하세요.

AWS 계정 루트 사용자

를 생성할 때 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 AWS 계정 theroot 사용자라는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 일상적인 태스크에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명이 필요한 작업은 IAM 사용 설명서의 루트 사용자 자격 증명이 필요한 작업 섹션을 참조하세요.

페더레이션 ID

가장 좋은 방법은 인간 사용자에게 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 서비스 사용하여에 액세스하도록 요구하는 것입니다.

페더레이션 자격 증명은 엔터프라이즈 디렉터리, 웹 자격 증명 공급자 또는 자격 증명 소스의 자격 증명을 AWS 서비스 사용하여 Directory Service 에 액세스하는 사용자입니다. 페더레이션 ID는 임시 자격 증명을 제공하는 역할을 수임합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center를 추천합니다. 자세한 정보는 AWS IAM Identity Center 사용 설명서의 What is IAM Identity Center?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가진 ID입니다. 장기 자격 증명이 있는 IAM 사용자 대신 임시 자격 증명을 사용하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 사용하여에 액세스하도록 인간 사용자에게 요구하기를 참조하세요.

IAM 그룹은 IAM 사용자 모음을 지정하고 대규모 사용자 집합에 대한 관리 권한을 더 쉽게 만듭니다. 자세한 내용은 IAM 사용 설명서의 IAM 사용자 사용 사례 섹션을 참조하세요.

IAM 역할

IAM 역할은 임시 자격 증명을 제공하는 특정 권한이 있는 자격 증명입니다. 사용자에서 IAM 역할(콘솔)로 전환하거나 또는 API 작업을 호출하여 역할을 수임할 수 있습니다. AWS CLI AWS 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

IAM 역할은 페더레이션 사용자 액세스, 임시 IAM 사용자 권한, 교차 계정 액세스, 교차 서비스 액세스 및 Amazon EC2에서 실행되는 애플리케이션에 유용합니다. 자세한 내용은 IAM 사용 설명서의 교차 계정 리소스 액세스를 참조하세요.

정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결될 때 권한을 정의합니다.는 보안 주체가 요청할 때 이러한 정책을 AWS 평가합니다. 대부분의 정책은에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서에 대한 자세한 내용은 IAM 사용 설명서의 JSON 정책 개요 섹션을 참조하세요.

정책을 사용하여 관리자는 어떤 보안 주체가 어떤 리소스에 대해 어떤 조건에서 작업을 수행할 수 있는지 정의하여 누가 무엇을 액세스할 수 있는지 지정합니다.

기본적으로 사용자 및 역할에는 어떠한 권한도 없습니다. IAM 관리자는 IAM 정책을 생성하고 사용자가 수임할 수 있는 역할에 추가합니다. IAM 정책은 작업을 수행하기 위해 사용하는 방법과 관계없이 작업에 대한 권한을 정의합니다.

ID 기반 정책

ID 기반 정책은 ID(사용자, 사용자 그룹 또는 역할)에 연결하는 JSON 권한 정책 문서입니다. 이러한 정책은 자격 증명이 수행할 수 있는 작업, 대상 리소스 및 이에 관한 조건을 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책(단일 ID에 직접 포함) 또는 관리형 정책(여러 ID에 연결된 독립 실행형 정책)일 수 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택 섹션을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 예를 들어 IAM 역할 신뢰 정책 및 Amazon S3 버킷 정책이 있습니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한

정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

기타 정책 유형

AWS 는 보다 일반적인 정책 유형에서 부여한 최대 권한을 설정할 수 있는 추가 정책 유형을 지원합니다.

- 권한 경계 - ID 기반 정책에서 IAM 엔터티에 부여할 수 있는 최대 권한을 설정합니다. 자세한 정보는 IAM 사용 설명서의 IAM 엔터티의 권한 범위를 참조하세요.
- 서비스 제어 정책(SCP) - AWS Organizations내 조직 또는 조직 단위에 대한 최대 권한을 지정합니다. 자세한 내용은AWS Organizations 사용 설명서의 서비스 제어 정책을 참조하세요.
- 리소스 제어 정책(RCP) – 계정의 리소스에 사용할 수 있는 최대 권한을 설정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCP)을 참조하세요.
- 세션 정책 – 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 자세한 내용은 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

Amazon DocumentDB에서 IAM을 사용하는 방법

IAM을 사용하여 Amazon DocumentDB에 대한 액세스를 관리하기 전에 Amazon DocumentDB에서 사용할 수 있는 IAM 기능에 대해 알아봅니다.

Amazon DocumentDB에서 사용할 수 있는 IAM 기능

IAM 특성	인스턴스 기반 클러스터	엘라스틱 클러스터
자격 증명 기반 정책	예	예
리소스 기반 정책	아니요	아니요

IAM 특성	인스턴스 기반 클러스터	엘라스틱 클러스터
정책 작업	예	예
정책 리소스	예	예
정책 조건 키(서비스별)	예	예
ACL	아니요	아니요
ABAC(정책 내 태그)	부분적	예
임시 보안 인증	예	예
엔터티 권한	예	예
서비스 역할	예	예
서비스 연결 역할	아니요	예

Amazon DocumentDB 및 기타 AWS 서비스에서 대부분의 IAM 기능을 사용하는 방법을 전체적으로 알아보려면 IAM 사용 설명서의 AWS IAM으로 작업하는 서비스를 참조하세요.

Amazon DocumentDB의 ID 기반 정책

ID 기반 정책 지원: 예

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. JSON 정책에서 사용할 수 있는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

Amazon DocumentDB의 ID 기반 정책 예

Amazon DocumentDB ID 기반 정책 예제를 보려면 Amazon DocumentDB의 ID 기반 정책 예 단원을 참조하십시오.

Amazon DocumentDB 내의 리소스 기반 정책

리소스 기반 정책 지원: 아니요

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 보안 주체가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는가 포함될 수 있습니다 AWS 서비스.

교차 계정 액세스를 활성화하려는 경우, 전체 계정이나 다른 계정의 IAM 개체를 리소스 기반 정책의 보안 주체로 지정할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM에서 교차 계정 리소스 액세스를 참조하세요.

Amazon DocumentDB의 정책 작업

정책 작업 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action`요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

> **Note**
>
> 일부 관리 기능의 경우 Amazon DocumentDB는 Amazon Relational Database Service(RDS)와 공유되는 운영 기술을 사용합니다.
> RDS 작업의 목록을 보려면 서비스 인증 참조의 Amazon Relational Database Service에서 정의한 작업을 참조하십시오.
> Amazon DocumentDB 탄력적 클러스터에 대한 정책 작업을 보려면 서비스 승인 참조의 Amazon DocumentDB 탄력적 클러스터에서 정의한 작업을 참조하십시오.

Amazon DocumentDB의 정책 작업은 작업 앞에 다음 접두사를 사용합니다.

```
aws
```

단일 문에서 여러 작업을 지정하려면 쉼표로 구분합니다.

```
"Action": [
      "aws:action1",
      "aws:action2"
         ]
```

Amazon DocumentDB ID 기반 정책 예제를 보려면 Amazon DocumentDB의 ID 기반 정책 예 단원을 참조하십시오.

Amazon DocumentDB용 정책 리소스

정책 리소스 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

> **Note**
>
> 일부 관리 기능의 경우 Amazon DocumentDB는 Amazon Relational Database Service(RDS)와 공유되는 운영 기술을 사용합니다.
> RDS 리소스 유형 및 해당 ARN 목록을 보려면 서비스 승인 참조에서 Amazon Relational Database Service에서 정의한 리소스를 참조하십시오. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 Amazon Relational Database Service에서 정의한 작업을 참조하십시오.
> Amazon DocumentDB 탄력적 클러스터의 리소스 유형을 보려면 서비스 승인 참조의 Amazon DocumentDB 탄력적 클러스터에서 정의한 리소스 유형을 참조하세요.

Amazon DocumentDB ID 기반 정책 예제를 보려면 Amazon DocumentDB의 ID 기반 정책 예 단원을 참조하십시오.

Amazon DocumentDB의 정책 조건 키

서비스별 정책 조건 키 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Condition` 요소는 정의된 기준에 따라 문이 실행되는 시기를 지정합니다. 같음(equals) 또는 미만(less than)과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

> **Note**
>
> 일부 관리 기능의 경우 Amazon DocumentDB는 Amazon Relational Database Service(RDS)와 공유되는 운영 기술을 사용합니다.
> RDS 조건 키 목록을 보려면 서비스 승인 참조의 Amazon Relational Database Service에 대한 조건 키를 참조하십시오. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 Amazon Relational Database Service에서 정의한 작업을 참조하십시오.
> Amazon DocumentDB 탄력적 클러스터에 대한 조건 키를 보려면 서비스 승인 참조의 Amazon DocumentDB 탄력적 클러스터에 대한 조건 키를 참조하세요.

Amazon DocumentDB ID 기반 정책 예제를 보려면 Amazon DocumentDB의 ID 기반 정책 예 단원을 참조하십시오.

Amazon DocumentDB의 ACL

ACL 지원: 아니요

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon DocumentDB의 ABAC

> **Note**
>
> ABAC는 인스턴스 기반 클러스터에서는 일부만 지원되지만 엘라스틱 클러스터에서는 지원됩니다.

속성 기반 액세스 제어(ABAC)는 태그라고 불리는 속성을 기반으로 권한을 정의하는 권한 부여 전략입니다. IAM 엔터티 및 AWS 리소스에 태그를 연결한 다음 보안 주체의 태그가 리소스의 태그와 일치할 때 작업을 허용하는 ABAC 정책을 설계할 수 있습니다.

태그에 근거하여 액세스를 제어하려면 `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`또는 `aws:TagKeys`조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

서비스가 모든 리소스 유형에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 유형에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

ABAC에 대한 자세한 내용은 IAM 사용 설명서의 ABAC 권한 부여를 통한 권한 정의를 참조하세요. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 속성 기반 액세스 제어(ABAC) 사용을 참조하세요.

Amazon DocumentDB에서 임시 보안 인증 정보 사용

임시 자격 증명 지원: 예

임시 자격 증명은 AWS 리소스에 대한 단기 액세스를 제공하며 페더레이션 또는 전환 역할을 사용할 때 자동으로 생성됩니다. 장기 액세스 키를 사용하는 대신 임시 자격 증명을 동적으로 생성하는 것이 AWS 좋습니다. 자세한 내용은 IAM 사용 설명서의 IAM의 임시 보안 자격 증명 및 IAM으로 작업하는 AWS 서비스 섹션을 참조하세요.

Amazon DocumentDB에 대한 교차 서비스 보안 주체 권한

전달 액세스 세션(FAS) 지원: 예

전달 액세스 세션(FAS)은를 호출하는 보안 주체의 권한을 다운스트림 서비스에 대한 요청 AWS 서비스 과 AWS 서비스함께 사용합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.

Amazon DocumentDB의 서비스 역할

서비스 역할 지원: 예

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 AWS 서비스 AWS에 권한을 위임할 역할 생성을 참조하세요.

> **Warning**
>
> 서비스 역할에 대한 권한을 변경하면 Amazon DocumentDB 기능이 중단될 수 있습니다. Amazon DocumentDB에서 관련 지침을 제공하는 경우에만 서비스 역할을 편집합니다.

Amazon DocumentDB의 서비스 연결 역할

> **Note**
>
> 서비스 연결 역할은 인스턴스 기반 클러스터에서는 지원되지 않지만 엘라스틱 클러스터에서는 지원됩니다.

서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 태스크를 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 표시 AWS 계정 되며 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 IAM으로 작업하는AWS 서비스를 참조하세요. 서비스 연결 역할 열에서 Yes가 포함된 서비스를 테이블에서 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예(Yes) 링크를 선택합니다.

Amazon DocumentDB의 ID 기반 정책 예

기본적으로 사용자 및 역할은 Amazon DocumentDB 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성(콘솔)을 참조하세요.

각 리소스 유형에 대한 ARN 형식을 비롯하여 Amazon DocumentDB에서 정의되는 작업 및 리소스 유형에 대한 자세한 내용은 서비스 승인 참조의 Amazon Relational Database Service에 사용되는 작업, 리소스 및 조건 키를 참조하십시오.

> **Note**
>
> 일부 관리 기능의 경우 Amazon DocumentDB는 Amazon Relational Database Service(RDS)와 공유되는 운영 기술을 사용합니다.

Amazon DocumentDB 탄력적 클러스터에 대한 정책 작업에 대해 알아보려면, 서비스 승인 참조의 Amazon DocumentDB 탄력적 클러스터에 대한 작업, 리소스, 조건 키를 참조하십시오.

주제

- 정책 모범 사례
- Amazon DocumentDB 콘솔 사용
- 사용자가 자신의 고유한 권한을 볼 수 있도록 허용

정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 Amazon DocumentDB 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따르세요.

- AWS 관리형 정책을 시작하고 최소 권한으로 전환 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반적인 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 에서 사용할 수 있습니다 AWS 계정. 사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 추가로 줄이는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책 또는 AWS 직무에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 IAM의 정책 및 권한을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. AWS 서비스와 같은 특정를 통해 사용되는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 CloudFormation. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer에서 정책 검증을 참조하세요.

- 다중 인증(MFA) 필요 -에서 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 AWS 계정켭니다. API 작업을 직접적으로 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA를 통한 보안 API 액세스를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

Amazon DocumentDB 콘솔 사용

Amazon DocumentDB(MongoDB 호환) 콘솔에 액세스하려면 최소한의 권한 세트가 있어야 합니다. 이러한 권한은에서 Amazon DocumentDB 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다 AWS 계정. 최소 필수 권한보다 더 제한적인 ID 기반 정책을 생성하는 경우, 콘솔이 해당 정책에 연결된 엔티티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

AWS CLI 또는 AWS API만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다. 대신, 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자와 역할이 Amazon DocumentDB 콘솔을 계속 사용할 수 있도록 하려면 Amazon DocumentDB *ConsoleAccess* 또는 *ReadOnly* AWS 관리형 정책도 엔터티에 연결합니다. 자세한 내용은 IAM 사용 설명서의 사용자에게 권한 추가를 참조하십시오.

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로이 작업을 완료할 수 있는 권한이 포함됩니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
```

```
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon DocumentDB ID 및 액세스 문제 해결

다음 정보를 사용하여 Amazon DocumentDB 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단하고 수정할 수 있습니다.

주제

- Amazon DocumentDB에서 작업을 수행할 권한이 없음
- iam:PassRole을 수행하도록 인증되지 않음
- 내 외부의 사람이 내 Amazon DocumentDB 리소스 AWS 계정 에 액세스하도록 허용하고 싶습니다.

Amazon DocumentDB에서 작업을 수행할 권한이 없음

작업을 수행할 권한이 없다는 오류가 표시되면 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음의 예제 오류는 `mateojackson` IAM 사용자가 콘솔을 사용하여 가상 *`my-example-widget`* 리소스에 대한 세부 정보를 보려고 하지만 가상 `aws:`*`GetWidget`* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 aws:GetWidget on resource: my-example-widget
```

이 경우, aws:*GetWidget* 작업을 사용하여 *my-example-widget* 리소스에 액세스할 수 있도록 mateojackson 사용자 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

iam:PassRole을 수행하도록 인증되지 않음

iam:PassRole 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 Amazon DocumentDB에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스 에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 역할을 서비스에 전달할 권한이 있어야 합니다.

다음 예제 오류는 marymajor라는 IAM 사용자가 콘솔을 사용하여 Amazon DocumentDB에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 권한이 없습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우, Mary가 iam:PassRole작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 외부의 사람이 내 Amazon DocumentDB 리소스 AWS 계정 에 액세스하도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하십시오.

- Amazon DocumentDB에서 이러한 기능을 지원하는지 여부를 알아보려면 Amazon DocumentDB에서 IAM을 사용하는 방법 단원을 참조하십시오.
- 소유 AWS 계정 한의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 소유한 다른의 IAM 사용자에게 액세스 권한 제공을 참조 AWS 계정 하세요.
- 타사에 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사가 AWS 계정 소유한에 대한 액세스 권한 제공을 AWS 계정참조하세요.

- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.

Amazon DocumentDB 리소스에 대한 액세스 권한 관리

모든 AWS 리소스는에서 소유하며 AWS 계정, 리소스를 생성하거나 액세스할 수 있는 권한은 권한 정책에 의해 관리됩니다. 계정 관리자는 IAM 자격 증명(즉, 사용자, 그룹 및 역할)에 권한 정책을 연결할 수 있으며 일부 서비스(예: AWS Lambda)는 리소스에 권한 정책 연결을 지원합니다.

> **Note**
>
> 계정 관리자(또는 관리자 사용자)는 관리자 권한이 있는 사용자입니다. 자세한 내용은 IAM 사용 설명서의 IAM 모범 사례 단원을 참조하십시오.

주제

- Amazon DocumentDB 리소스 및 작업
- 리소스 소유권 이해
- 리소스 액세스 관리
- 정책 요소 지정: 작업, 효과, 리소스, 보안 주체
- 정책에서 조건 지정

Amazon DocumentDB 리소스 및 작업

Amazon DocumentDB에서 기본 리소스는 클러스터입니다. Amazon DocumentDB는 기본 리소스와 함께 사용할 수 있는 다른 리소스(예: 인스턴스, 파라미터 그룹, 이벤트 구독 등)를 지원합니다. 이 리소스를 하위 리소스라고 합니다.

다음 표에 나와 있는 것처럼 이러한 리소스와 하위 리소스에는 고유한 Amazon 리소스 이름(ARN)이 연결되어 있습니다.

리소스 유형	ARN 형식
Cluster	`arn:aws:rds:` *region*`:`*account-id* `:cluster:` *db-cluster-name*
클러스터 파라미터 그룹	`arn:aws:rds:` *region*`:`*account-id* `:cluster-pg:` *cluster-parameter-group-name*
클러스터 스냅샷	`arn:aws:rds:` *region*`:`*account-id* `:cluster-snapshot:` *cluster-snapshot-name*
Instance	`arn:aws:rds:` *region*`:`*account-id* `:db:`*db-instance-name*
보안 그룹	`arn:aws:rds:` *region*`:`*account-id* `:secgrp:`*security-group-name*
서브넷 그룹	`arn:aws:rds:` *region*`:`*account-id* `:subgrp:`*subnet-group-name*

Amazon DocumentDB은 Amazon DocumentDB 리소스를 처리하기 위한 작업을 제공합니다. 사용 가능한 작업 목록은 작업 단원을 참조하십시오.

리소스 소유권 이해

리소스 소유자는 리소스를 AWS 계정 생성한 입니다. 즉, 리소스 소유자는 리소스를 생성하는 요청을 인증하는 AWS 계정 보안 주체 엔터티(루트 계정, IAM 사용자 또는 IAM 역할)의 입니다. 다음 예제에서는 이러한 작동 방법을 설명합니다.

- 의 루트 계정 자격 증명을 사용하여 인스턴스와 같은 Amazon DocumentDB 리소스를 AWS 계정 생성하는 경우 AWS 계정 는 Amazon DocumentDB 리소스의 소유자입니다.
- 에서 IAM 사용자를 생성하고 해당 사용자에게 Amazon DocumentDB 리소스를 생성할 수 있는 권한을 AWS 계정 부여하는 경우 사용자는 Amazon DocumentDB 리소스를 생성할 수 있습니다. 하지만 AWS 계정사용자가 속한이 Amazon DocumentDB 리소스를 소유합니다.
- 에서 Amazon DocumentDB 리소스를 생성할 수 있는 권한이 AWS 계정 있는 IAM 역할을 생성하는 경우 해당 역할을 수임할 수 있는 사람은 누구나 Amazon DocumentDB 리소스를 생성할 수 있습니다. 역할 AWS 계정이 속한이 Amazon DocumentDB 리소스를 소유합니다.

리소스 액세스 관리

권한 정책은 누가 무엇에 액세스 할 수 있는지를 나타냅니다. 다음 섹션에서는 권한 정책을 만드는 데 사용 가능한 옵션에 대해 설명합니다.

> **Note**
>
> 이 섹션에서는 Amazon DocumentDB의 맥락에서 IAM을 사용하는 방법에 대해 설명합니다. IAM 서비스에 대한 자세한 내용은 다루지 않습니다. IAM 설명서의 전체 내용은 IAM 사용자 안내서의 IAM이란?을 참조하세요. IAM 정책 구문과 설명에 대한 자세한 내용은 IAM 사용자 안내서의 AWS IAM 정책 참조를 참조하세요.

IAM ID에 연결된 정책을 ID 기반 정책(IAM 정책)이라고 합니다. 리소스에 연결된 정책을 리소스 기반 정책이라고 합니다. Amazon DocumentDB는 ID 기반 정책(IAM 정책)만 지원합니다.

주제

- 자격 증명 기반 정책(IAM 정책)
- 리소스 기반 정책

자격 증명 기반 정책(IAM 정책)

정책을 IAM 보안 인증에 연결할 수 있습니다. 예를 들면, 다음을 수행할 수 있습니다.

- 계정 내 사용자 또는 그룹에 권한 정책 연결 - 계정 관리자는 특정 사용자에 연결된 권한 정책을 사용하여 해당 사용자에게 Amazon DocumentDB 리소스(예: 인스턴스) 생성 권한을 부여할 수 있습니다.
- 역할에 권한 정책 연결(교차 계정 권한 부여) – ID 기반 권한 정책을 IAM 역할에 연결하여 교차 계정 권한을 부여할 수 있습니다. 예를 들어 관리자는 다음과 같이 다른 AWS 계정 또는 AWS 서비스에 교차 계정 권한을 부여하는 역할을 생성할 수 있습니다.
  1. 계정 A 관리자는 IAM 역할을 생성하고 계정 A의 리소스에 대한 권한을 부여하는 역할에 권한 정책을 연결합니다.
  2. 계정 A 관리자는 계정 B를 역할을 수임할 보안 주체로 식별하는 역할에 신뢰 정책을 연결합니다.
  3. 그런 다음 계정 B 관리자는 계정 B의 사용자에게 역할을 수임할 권한을 위임할 수 있습니다. 이렇게 하면 계정 B의 사용자가 계정 A에서 리소스를 생성하거나 액세스할 수 있습니다. 역할을 수임할 권한을 AWS 서비스에 부여하려는 경우 신뢰 정책의 보안 주체가 AWS 서비스 보안 주체가 될 수도 있습니다.

IAM을 사용하여 권한을 위임하는 방법에 대한 자세한 내용은 IAM 사용자 안내서의 액세스 관리를 참조하세요.

다음은 ID가 123456789012인 사용자에게 AWS 계정계정에 대한 인스턴스를 생성하도록 허용하는 정책 예제입니다. 새로운 인스턴스는 `default`로 시작하는 옵션 그룹 및 파라미터 그룹과 `default` 서브넷 그룹을 사용해야 합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreateDBInstanceOnly",
            "Effect": "Allow",
            "Action": [
                "rds:CreateDBInstance"
            ],
            "Resource": [
                "arn:aws:rds:*:123456789012:db:test*",
                "arn:aws:rds:*:123456789012:pg:cluster-pg:default*",
                "arn:aws:rds:*:123456789012:subgrp:default"
            ]
        }
    ]
}
```

Amazon DocumentDB에서 ID 기반 정책 사용에 대한 자세한 내용은 Amazon DocumentDB에 대한 ID 기반 정책(IAM 정책) 사용 단원을 참조하십시오. 사용자, 그룹, 역할 및 권한에 대한 자세한 내용은 IAM 사용 설명서의 ID(사용자, 그룹 및 역할)을 참조하십시오.

리소스 기반 정책

Amazon Simple Storage Service(S3)와 같은 다른 서비스도 리소스 기반 권한 정책을 지원합니다. 예를 들어 Amazon S3 버킷에 정책을 연결하여 해당 버킷에 대한 액세스 권한을 관리할 수 있습니다. Amazon DocumentDB는 리소스 기반 액세스 정책을 지원하지 않습니다.

정책 요소 지정: 작업, 효과, 리소스, 보안 주체

각 Amazon DocumentDB 리소스(Amazon DocumentDB 리소스 및 작업 참조)에서 이 서비스는 API 작업을 정의합니다. 자세한 내용은 작업을 참조하십시오. 이러한 API 작업에 대한 권한을 부여하기 위해 Amazon DocumentDB에서는 정책에서 지정할 수 있는 작업을 정의합니다. API 작업을 실시하려면 둘 이상의 작업에 대한 권한이 필요할 수 있습니다.

다음은 기본 정책 요소입니다.

- 리소스 – 정책에서 Amazon 리소스 이름(ARN)을 사용하여 정책을 적용할 리소스를 식별합니다.
- 작업 – 작업 키워드를 사용하여 허용 또는 거부할 리소스 작업을 식별합니다. 예를 들어 `rds:DescribeDBInstances` 권한은 사용자에게 `DescribeDBInstances` 작업 수행을 허용합니다.
- 결과 – 사용자가 특정 작업을 요청하는 경우의 결과를 지정합니다. 이는 허용 또는 거부 중에 하나가 될 수 있습니다. 명시적으로 리소스에 대한 액세스 권한을 부여(허용)하지 않는 경우, 액세스는 묵시적으로 거부됩니다. 다른 정책에서 액세스 권한을 부여하는 경우라도 사용자가 해당 리소스에 액세스할 수 없도록 하기 위해 리소스에 대한 권한을 명시적으로 거부할 수도 있습니다.
- 보안 주체 – ID 기반 정책(IAM 정책)에서 정책이 연결되는 사용자는 암시적인 보안 주체입니다. 리소스 기반 정책의 경우, 사용자, 계정, 서비스 또는 권한의 수신자인 기타 개체를 지정합니다(리소스 기반 정책에만 해당). Amazon DocumentDB는 리소스 기반 액세스 정책을 지원하지 않습니다.

IAM 정책 구문과 설명에 대한 자세한 내용은 IAM 사용 설명서의 AWS IAM 정책 참조를 참조하십시오.

모든 Amazon DocumentDB API 작업과 해당 작업이 적용되는 리소스를 보여주는 표는 Amazon DocumentDB API 권한: 작업, 리소스 및 조건 참조 단원을 참조하십시오.

정책에서 조건 지정

권한을 부여할 때 IAM 정책 언어를 사용하여 정책이 적용되는 조건을 지정할 수 있습니다. 예를 들어, 특정 날짜 이후에만 정책을 적용할 수 있습니다. 정책 언어에서의 조건 지정에 관한 자세한 내용은 IAM 사용 설명서의 조건을 참조하십시오.

조건을 표시하려면 미리 정의된 조건 키를 사용합니다. Amazon DocumentDB에는 IAM 정책에 사용할 수 있는 서비스별 컨텍스트 키가 없습니다. 모든 서비스에서 사용할 수 있는 전역 조건 컨텍스트 키 목록은 IAM 사용 설명서의 사용 가능한 전역 조건 키를 참조하세요.

Amazon DocumentDB에 대한 ID 기반 정책(IAM 정책) 사용

> **Important**
>
> 일부 관리 기능의 경우 Amazon DocumentDB는 Amazon RDS와 공유되는 운영 기술을 사용합니다. Amazon DocumentDB 콘솔 AWS CLI, 및 API 호출은 Amazon RDS API에 대한 호출로 로깅됩니다.
>
> 먼저 Amazon DocumentDB 리소스에 대한 액세스 관리를 위한 기본 개념과 옵션을 설명하는 소개 주제를 검토합니다. 자세한 내용은 Amazon DocumentDB 리소스에 대한 액세스 권한 관리 섹션을 참조하세요.

이 항목에서는 계정 관리자가 IAM ID(사용자, 그룹, 역할)에 권한 정책을 연결할 수 있는 ID 기반 정책의 예를 제공합니다.

다음은 IAM 정책 예제입니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreateDBInstanceOnly",
            "Effect": "Allow",
            "Action": [
                "rds:CreateDBInstance"
            ],
            "Resource": [
                "arn:aws:rds:*:123456789012:db:test*",
                "arn:aws:rds:*:123456789012:pg:cluster-pg:default*",
                "arn:aws:rds:*:123456789012:subgrp:default"
            ]
        }
    ]
}
```

정책은 다음 IAM 사용자 권한을 지정하는 단일 명령문을 포함합니다.

- 이 정책은 IAM 사용자가 CreateDBInstance 작업을 사용하여 인스턴스를 생성할 수 있도록 허용합니다(이는 create-db-instance AWS CLI 작업 및 에도 적용됨 AWS Management Console).
- `Resource` 요소는 사용자가 리소스 위치에서 또는 리소스를 사용하여 작업을 수행할 수 있도록 지정합니다. Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 이 ARN에는 리소스가 속한 서비스의 이름(`rds`), AWS 리전 (`*`는이 예제의 모든 리전을 나타냄), 사용자 계정 번호(`123456789012`는이 예제의 사용자 ID) 및 리소스 유형이 포함됩니다.

  위의 예제에서 `Resource` 요소는 사용자 리소스에 대해 다음과 같은 정책 제약 조건을 지정합니다.

  - 새 인스턴스의 인스턴스 식별자는 `test`로 시작해야 합니다(예: `testCustomerData1`, `test-region2-data`).
  - 새로운 인스턴스의 클러스터 파라미터 그룹은 `default`로 시작해야 합니다.
  - 새로운 인스턴스의 서브넷 그룹은 `default` 서브넷 그룹이 되어야 합니다.

ID 기반 정책에서는 권한을 가질 보안 주체를 지정하지 않으므로 이 정책은 `Principal` 요소를 지정하지 않습니다. 정책을 사용자에게 연결할 경우, 사용자는 암시적인 보안 주체입니다. IAM 역할에 권한 정책을 연결하면 역할의 신뢰 정책에서 식별된 보안 주체가 권한을 얻습니다.

모든 Amazon DocumentDB API 작업과 해당 작업이 적용되는 리소스를 보여주는 표는 Amazon DocumentDB API 권한: 작업, 리소스 및 조건 참조 단원을 참조하십시오.

Amazon DocumentDB 콘솔 사용에 필요한 권한

Amazon DocumentDB 콘솔에서 작업하려면 최소한의 권한이 사용자에게 필요합니다. 이러한 권한을 통해 사용자는에 대한 Amazon DocumentDB 리소스를 설명하고 Amazon EC2 보안 및 네트워크 정보를 포함한 기타 관련 정보를 AWS 계정 제공할 수 있습니다.

최소 필수 권한보다 더 제한적인 IAM 정책을 만들면 콘솔은 해당 IAM 정책에 연결된 사용자에 대해 의도대로 작동하지 않습니다. 이 사용자가 Amazon DocumentDB 콘솔을 사용할 수 있도록 하려면 `AmazonDocDBConsoleFullAccess` 관리형 정책을 사용자에게 연결합니다(AWS Amazon DocumentDB에 대한 관리형 정책 참조).

AWS CLI 또는 Amazon DocumentDB API만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다.

고객 관리형 정책 예제

이 섹션에서는 다양한 Amazon DocumentDB 작업에 대한 권한을 부여하는 사용자 정책의 예를 제공합니다. 이러한 정책은 Amazon DocumentDB API 작업, AWS SDKs 또는를 사용할 때 작동합니다 AWS CLI. 콘솔을 사용하는 경우 Amazon DocumentDB 콘솔 사용에 필요한 권한의 설명과 같이 콘솔에 특정한 추가 권한을 부여해야 합니다.

일부 관리 기능의 경우 Amazon DocumentDB는 Amazon Relational Database Service(RDS) 및 Amazon Neptune과 공유되는 운영 기술을 사용합니다.

> **Note**
>
> 모든 예에서는 미국 동부(버지니아 북부) 리전(us-east-1)을 사용하며 가상의 계정 ID를 포함합니다.

예제

- 예제 1: 사용자가 모든 Amazon DocumentDB 리소스에서 Describe 작업을 수행할 수 있도록 허용
- 예제 2: 사용자의 인스턴스 삭제 방지
- 예 3: 스토리지 암호화가 활성화되지 않은 경우 사용자가 클러스터를 생성하지 못하도록 차단

예제 1: 사용자가 모든 Amazon DocumentDB 리소스에서 Describe 작업을 수행할 수 있도록 허용

다음 권한 정책은 사용자에게 Describe로 시작하는 모든 작업을 실행할 수 있는 권한을 부여합니다. 이 작업은 인스턴스와 같은 Amazon DocumentDB 리소스에 대한 정보를 보여 줍니다. Resource 요소에 와일드카드 문자(*)가 있으면 계정이 소유한 모든 Amazon DocumentDB 리소스에 작업이 허용됩니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowRDSDescribe",
         "Effect":"Allow",
         "Action":"rds:Describe*",
         "Resource":"*"
```

```
      }
   ]
}
```

예제 2: 사용자의 인스턴스 삭제 방지

다음 권한 정책은 사용자의 특정 인스턴스 삭제를 방지하는 권한을 부여합니다. 예를 들어, 관리자가 아닌 모든 사용자에 대해 프로덕션 인스턴스를 삭제할 수 있는 권한을 거부해야 할 수 있습니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyDelete1",
         "Effect":"Deny",
         "Action":"rds:DeleteDBInstance",
         "Resource":"arn:aws:rds:us-east-1:123456789012:db:my-db-instance"
      }
   ]
}
```

예 3: 스토리지 암호화가 활성화되지 않은 경우 사용자가 클러스터를 생성하지 못하도록 차단

다음 권한 정책은 스토리지 암호화가 활성화되지 않은 경우 Amazon DocumentDB 클러스터를 생성할 수 있는 사용자의 권한을 거부합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "PreventUnencryptedDocumentDB",
         "Effect": "Deny",
         "Action": "RDS:CreateDBCluster",
         "Condition": {
         "Bool": {
```

```
            "rds:StorageEncrypted": "false"
        },
            "StringEquals": {
            "rds:DatabaseEngine": "docdb"
            }
        },
        "Resource": "*"
        }
    ]
}
```

AWS Amazon DocumentDB에 대한 관리형 정책

사용자, 그룹 및 역할에 권한을 추가하려면 직접 정책을 작성하는 것보다 AWS 관리형 정책을 사용하는 것이 더 쉽습니다. 팀에 필요한 권한만 제공하는 IAM 고객 관리형 정책을 생성하기 위해서는 시간과 전문 지식이 필요합니다. 빠르게 시작하려면 AWS 관리형 정책을 사용할 수 있습니다. 이 정책은 일반적인 사용 사례를 다루며 사용자의 AWS 계정에서 사용할 수 있습니다. AWS 관리형 정책에 대한 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

AWS 서비스는 AWS 관리형 정책을 유지 관리하고 업데이트합니다. AWS 관리형 정책에서는 권한을 변경할 수 없습니다. 서비스는 때때로 AWS 관리형 정책에 추가 권한을 추가하여 새 기능을 지원합니다. 이 유형의 업데이트는 정책이 연결된 모든 ID(사용자, 그룹 및 역할)에 적용됩니다. 서비스는 새 기능이 시작되거나 새 작업을 사용할 수 있게 될 때 AWS 관리형 정책을 업데이트할 가능성이 높습니다. 서비스는 AWS 관리형 정책에서 권한을 제거하지 않으므로 정책 업데이트로 인해 기존 권한이 손상되지 않습니다.

또한는 여러 서비스에 걸쳐 있는 직무에 대한 관리형 정책을 AWS 지원합니다. 예를 들어 관리`ViewOnlyAccess` AWS 형 정책은 많은 AWS 서비스 및 리소스에 대한 읽기 전용 액세스를 제공합니다. 서비스가 새 기능을 시작하면는 새 작업 및 리소스에 대한 읽기 전용 권한을 AWS 추가합니다. 직무 정책의 목록과 설명은 IAM 사용 설명서의 직무에 관한AWS 관리형 정책을 참조하세요.

계정의 사용자에게 연결할 수 있는 다음 AWS 관리형 정책은 Amazon DocumentDB에 고유합니다.

- AmazonDocDBFullAccess - 루트의 모든 Amazon DocumentDB 리소스에 대한 전체 액세스 권한을 부여합니다 AWS 계정.
- AmazonDocDBReadOnlyAccess - 루트의 모든 Amazon DocumentDB 리소스에 대한 읽기 전용 액세스 권한을 부여합니다 AWS 계정.
- AmazonDocDBConsoleFullAccess - AWS Management Console를 사용하여 Amazon DocumentDB 및 Amazon DocumentDB 엘라스틱 클러스터 리소스를 관리할 수 있는 전체 액세스를 부여합니다.

- AmazonDocDBElasticReadOnlyAccess - 루트의 모든 Amazon DocumentDB 탄력적 클러스터 리소스에 대한 읽기 전용 액세스 권한을 부여합니다 AWS 계정.
- AmazonDocDBElasticFullAccess - 루트의 모든 Amazon DocumentDB 탄력적 클러스터 리소스에 대한 전체 액세스 권한을 부여합니다 AWS 계정.

AmazonDocDBFullAccess

이 정책은 모든 Amazon DocumentDB 작업에 대한 전체 액세스 권한을 허용하는 관리 권한을 보안 주체에게 부여합니다. 해당 정책의 권한은 다음과 같이 그룹화됩니다.

- Amazon DocumentDB 권한은 모든 Amazon DocumentDB 작업을 허용합니다.
- 이 정책의 일부 Amazon EC2 권한은 API 요청에서 전달된 리소스를 검증하는 데 필요합니다. 이는 Amazon DocumentDB가 클러스터에서 리소스를 성공적으로 사용할 수 있도록 하기 위한 것입니다. 이 정책의 나머지 Amazon EC2 권한은 Amazon DocumentDB가 클러스터에 연결하는 데 필요한 AWS 리소스를 생성하도록 허용합니다.
- Amazon DocumentDB 권한은 API 호출 중에 요청에서 전달된 리소스를 검증하는 데 사용됩니다. Amazon DocumentDB가 전달된 키를 Amazon DocumentDB 클러스터에서 사용할 수 있도록 하기 위해 필요합니다.
- CloudWatch Logs는 Amazon DocumentDB가 로그 전송 대상에 도달할 수 있는지, 브로커 로그 사용에 유효한지 확인하는 데 필요합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "rds:AddRoleToDBCluster",
                "rds:AddSourceIdentifierToSubscription",
                "rds:AddTagsToResource",
                "rds:ApplyPendingMaintenanceAction",
                "rds:CopyDBClusterParameterGroup",
                "rds:CopyDBClusterSnapshot",
                "rds:CopyDBParameterGroup",
                "rds:CreateDBCluster",
                "rds:CreateDBClusterParameterGroup",
                "rds:CreateDBClusterSnapshot",
```

```
                "rds:CreateDBInstance",
                "rds:CreateDBParameterGroup",
                "rds:CreateDBSubnetGroup",
                "rds:CreateEventSubscription",
                "rds:DeleteDBCluster",
                "rds:DeleteDBClusterParameterGroup",
                "rds:DeleteDBClusterSnapshot",
                "rds:DeleteDBInstance",
                "rds:DeleteDBParameterGroup",
                "rds:DeleteDBSubnetGroup",
                "rds:DeleteEventSubscription",
                "rds:DescribeAccountAttributes",
                "rds:DescribeCertificates",
                "rds:DescribeDBClusterParameterGroups",
                "rds:DescribeDBClusterParameters",
                "rds:DescribeDBClusterSnapshotAttributes",
                "rds:DescribeDBClusterSnapshots",
                "rds:DescribeDBClusters",
                "rds:DescribeDBEngineVersions",
                "rds:DescribeDBInstances",
                "rds:DescribeDBLogFiles",
                "rds:DescribeDBParameterGroups",
                "rds:DescribeDBParameters",
                "rds:DescribeDBSecurityGroups",
                "rds:DescribeDBSubnetGroups",
                "rds:DescribeEngineDefaultClusterParameters",
                "rds:DescribeEngineDefaultParameters",
                "rds:DescribeEventCategories",
                "rds:DescribeEventSubscriptions",
                "rds:DescribeEvents",
                "rds:DescribeOptionGroups",
                "rds:DescribeOrderableDBInstanceOptions",
                "rds:DescribePendingMaintenanceActions",
                "rds:DescribeValidDBInstanceModifications",
                "rds:DownloadDBLogFilePortion",
                "rds:FailoverDBCluster",
                "rds:ListTagsForResource",
                "rds:ModifyDBCluster",
                "rds:ModifyDBClusterParameterGroup",
                "rds:ModifyDBClusterSnapshotAttribute",
                "rds:ModifyDBInstance",
                "rds:ModifyDBParameterGroup",
                "rds:ModifyDBSubnetGroup",
                "rds:ModifyEventSubscription",
```

```
                "rds:PromoteReadReplicaDBCluster",
                "rds:RebootDBInstance",
                "rds:RemoveRoleFromDBCluster",
                "rds:RemoveSourceIdentifierFromSubscription",
                "rds:RemoveTagsFromResource",
                "rds:ResetDBClusterParameterGroup",
                "rds:ResetDBParameterGroup",
                "rds:RestoreDBClusterFromSnapshot",
                "rds:RestoreDBClusterToPointInTime"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
        {
            "Action": [
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcs",
                "kms:ListAliases",
                "kms:ListKeyPolicies",
                "kms:ListKeys",
                "kms:ListRetirableGrants",
                "logs:DescribeLogStreams",
                "logs:GetLogEvents",
                "sns:ListSubscriptions",
                "sns:ListTopics",
                "sns:Publish"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
        {
            "Action": "iam:CreateServiceLinkedRole",
            "Effect": "Allow",
```

```
            "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
            "Condition": {
                "StringLike": {
                    "iam:AWS ServiceName": "rds.amazonaws.com"
                }
            }
        }
    ]
}
```

AmazonDocDBReadOnlyAccess

이 정책은 Amazon DocumentDB의 정보를 볼 수 있는 읽기 전용 권한을 사용자에게 부여합니다. 이 정책이 첨부된 보안 주체는 기존 리소스를 업데이트하거나 삭제할 수 없으며 새 Amazon DocumentDB 리소스를 생성할 수도 없습니다. 예를 들어 이러한 권한이 있는 주체는 자신의 계정과 연결된 클러스터 및 구성 목록을 볼 수 있지만 클러스터의 구성이나 설정을 변경할 수는 없습니다. 해당 정책의 권한은 다음과 같이 그룹화됩니다.

- Amazon DocumentDB 권한을 사용하면 Amazon DocumentDB 리소스를 나열하고, 설명하고, 그에 대한 정보를 얻을 수 있습니다.
- Amazon EC2 권한은 클러스터와 연결된 Amazon VPC, 서브넷, 보안 그룹, ENI를 설명하는 데 사용됩니다.
- Amazon DocumentDB 권한은 클러스터와 연결된 키를 설명하는 데 사용됩니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "rds:DescribeAccountAttributes",
                "rds:DescribeCertificates",
                "rds:DescribeDBClusterParameterGroups",
                "rds:DescribeDBClusterParameters",
                "rds:DescribeDBClusterSnapshotAttributes",
                "rds:DescribeDBClusterSnapshots",
                "rds:DescribeDBClusters",
```

```
                "rds:DescribeDBEngineVersions",
                "rds:DescribeDBInstances",
                "rds:DescribeDBLogFiles",
                "rds:DescribeDBParameterGroups",
                "rds:DescribeDBParameters",
                "rds:DescribeDBSubnetGroups",
                "rds:DescribeEventCategories",
                "rds:DescribeEventSubscriptions",
                "rds:DescribeEvents",
                "rds:DescribeOrderableDBInstanceOptions",
                "rds:DescribePendingMaintenanceActions",
                "rds:DownloadDBLogFilePortion",
                "rds:ListTagsForResource"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeInternetGateways",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcs"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "kms:ListKeys",
                "kms:ListRetirableGrants",
                "kms:ListAliases",
                "kms:ListKeyPolicies"
```

```
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "logs:DescribeLogStreams",
                "logs:GetLogEvents"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:logs:*:*:log-group:/aws/rds/*:log-stream:*",
                "arn:aws:logs:*:*:log-group:/aws/docdb/*:log-stream:*"
            ]
        }
    ]
}
```

AmazonDocDBConsoleFullAccess

다음을 위해를 사용하여 Amazon DocumentDB 리소스를 관리할 AWS Management Console 수 있는 전체 액세스 권한을 부여합니다.

- 모든 Amazon DocumentDB 및 Amazon DocumentDB 클러스터 작업을 허용할 수 있는 Amazon DocumentDB 권한
- 이 정책의 일부 Amazon EC2 권한은 API 요청에서 전달된 리소스를 검증하는 데 필요합니다. 이는 Amazon DocumentDB가 클러스터에서 리소스를 성공적으로 프로비저닝하고 유지할 수 있도록 하기 위한 것입니다. 이 정책의 나머지 Amazon EC2 권한은 Amazon DocumentDB가 VPCEndpoint와 같은 클러스터에 연결하는 데 필요한 AWS 리소스를 생성하도록 허용합니다.
- AWS KMS 권한은에 대한 API 호출 중에 요청에서 전달된 리소스를 검증하는 AWS KMS 데 사용됩니다. 이는 Amazon DocumentDB가 전달된 키를 사용해 Amazon DocumentDB 엘라스틱 클러스터에서 저장 데이터를 암호화 및 암호 해독할 수 있도록 하기 위해 필요합니다.
- CloudWatch Logs는 Amazon DocumentDB가 로그 전송 대상에 도달할 수 있는지, 로그 사용 감사 및 프로파일링에 유효한지 확인하는 데 필요합니다.
- 주어진 암호를 검증하고 이를 사용하여 Amazon DocumentDB 엘라스틱 클러스터의 관리자를 설정하려면 Secrets Manager 권한이 필요합니다.
- Amazon DocumentDB 클러스터 관리 작업에는 Amazon RDS 권한이 필요합니다. 일부 관리 기능의 경우 Amazon DocumentDB는 Amazon RDS와 공유되는 운영 기술을 사용합니다.

- SNS 권한은 보안 주체에게 Amazon Simple Notification Service(SNS) 구독 및 주제를 허용하고 Amazon SNS 메시지를 게시하도록 허용합니다.
- 지표 및 로그 게시에 필요한 서비스 연결 역할을 생성하려면 IAM 권한이 필요합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DocdbSids",
            "Effect": "Allow",
            "Action": [
                "docdb-elastic:CreateCluster",
                "docdb-elastic:UpdateCluster",
                "docdb-elastic:GetCluster",
                "docdb-elastic:DeleteCluster",
                "docdb-elastic:ListClusters",
                "docdb-elastic:CreateClusterSnapshot",
                "docdb-elastic:GetClusterSnapshot",
                "docdb-elastic:DeleteClusterSnapshot",
                "docdb-elastic:ListClusterSnapshots",
                "docdb-elastic:RestoreClusterFromSnapshot",
                "docdb-elastic:TagResource",
                "docdb-elastic:UntagResource",
                "docdb-elastic:ListTagsForResource",
                "docdb-elastic:CopyClusterSnapshot",
                "docdb-elastic:StartCluster",
                "docdb-elastic:StopCluster",
                "docdb-elastic:GetPendingMaintenanceAction",
                "docdb-elastic:ListPendingMaintenanceActions",
                "docdb-elastic:ApplyPendingMaintenanceAction",
                "rds:AddRoleToDBCluster",
                "rds:AddSourceIdentifierToSubscription",
                "rds:AddTagsToResource",
                "rds:ApplyPendingMaintenanceAction",
                "rds:CopyDBClusterParameterGroup",
                "rds:CopyDBClusterSnapshot",
                "rds:CopyDBParameterGroup",
                "rds:CreateDBCluster",
                "rds:CreateDBClusterParameterGroup",
```

```
                "rds:CreateDBClusterSnapshot",
                "rds:CreateDBInstance",
                "rds:CreateDBParameterGroup",
                "rds:CreateDBSubnetGroup",
                "rds:CreateEventSubscription",
                "rds:CreateGlobalCluster",
                "rds:DeleteDBCluster",
                "rds:DeleteDBClusterParameterGroup",
                "rds:DeleteDBClusterSnapshot",
                "rds:DeleteDBInstance",
                "rds:DeleteDBParameterGroup",
                "rds:DeleteDBSubnetGroup",
                "rds:DeleteEventSubscription",
                "rds:DeleteGlobalCluster",
                "rds:DescribeAccountAttributes",
                "rds:DescribeCertificates",
                "rds:DescribeDBClusterParameterGroups",
                "rds:DescribeDBClusterParameters",
                "rds:DescribeDBClusterSnapshotAttributes",
                "rds:DescribeDBClusterSnapshots",
                "rds:DescribeDBClusters",
                "rds:DescribeDBEngineVersions",
                "rds:DescribeDBInstances",
                "rds:DescribeDBLogFiles",
                "rds:DescribeDBParameterGroups",
                "rds:DescribeDBParameters",
                "rds:DescribeDBSecurityGroups",
                "rds:DescribeDBSubnetGroups",
                "rds:DescribeEngineDefaultClusterParameters",
                "rds:DescribeEngineDefaultParameters",
                "rds:DescribeEventCategories",
                "rds:DescribeEventSubscriptions",
                "rds:DescribeEvents",
                "rds:DescribeGlobalClusters",
                "rds:DescribeOptionGroups",
                "rds:DescribeOrderableDBInstanceOptions",
                "rds:DescribePendingMaintenanceActions",
                "rds:DescribeValidDBInstanceModifications",
                "rds:DownloadDBLogFilePortion",
                "rds:FailoverDBCluster",
                "rds:ListTagsForResource",
                "rds:ModifyDBCluster",
                "rds:ModifyDBClusterParameterGroup",
                "rds:ModifyDBClusterSnapshotAttribute",
```

```
                "rds:ModifyDBInstance",
                "rds:ModifyDBParameterGroup",
                "rds:ModifyDBSubnetGroup",
                "rds:ModifyEventSubscription",
                "rds:ModifyGlobalCluster",
                "rds:PromoteReadReplicaDBCluster",
                "rds:RebootDBInstance",
                "rds:RemoveFromGlobalCluster",
                "rds:RemoveRoleFromDBCluster",
                "rds:RemoveSourceIdentifierFromSubscription",
                "rds:RemoveTagsFromResource",
                "rds:ResetDBClusterParameterGroup",
                "rds:ResetDBParameterGroup",
                "rds:RestoreDBClusterFromSnapshot",
                "rds:RestoreDBClusterToPointInTime"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "DependencySids",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics",
                "ec2:AllocateAddress",
                "ec2:AssignIpv6Addresses",
                "ec2:AssignPrivateIpAddresses",
                "ec2:AssociateAddress",
                "ec2:AssociateRouteTable",
                "ec2:AssociateSubnetCidrBlock",
                "ec2:AssociateVpcCidrBlock",
                "ec2:AttachInternetGateway",
                "ec2:AttachNetworkInterface",
                "ec2:CreateCustomerGateway",
                "ec2:CreateDefaultSubnet",
                "ec2:CreateDefaultVpc",
                "ec2:CreateInternetGateway",
                "ec2:CreateNatGateway",
                "ec2:CreateNetworkInterface",
                "ec2:CreateRoute",
```

```
                "ec2:CreateRouteTable",
                "ec2:CreateSecurityGroup",
                "ec2:CreateSubnet",
                "ec2:CreateVpc",
                "ec2:CreateVpcEndpoint",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAddresses",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeCustomerGateways",
                "ec2:DescribeInstances",
                "ec2:DescribeNatGateways",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribePrefixLists",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroupReferences",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeVpcs",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:ModifySubnetAttribute",
                "ec2:ModifyVpcAttribute",
                "ec2:ModifyVpcEndpoint",
                "kms:DescribeKey",
                "kms:ListAliases",
                "kms:ListKeyPolicies",
                "kms:ListKeys",
                "kms:ListRetirableGrants",
                "logs:DescribeLogStreams",
                "logs:GetLogEvents",
                "sns:ListSubscriptions",
                "sns:ListTopics",
                "sns:Publish"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "DocdbSLRSid",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
```

```
            "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "rds.amazonaws.com"
                }
            }
        },
        {
            "Sid": "DocdbElasticSLRSid",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/docdb-
elastic.amazonaws.com/AWSServiceRoleForDocDB-Elastic",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "docdb-elastic.amazonaws.com"
                }
            }
        }
    ]
}
```

AmazonDocDBElasticReadOnlyAccess

이 정책은 Amazon DocumentDB의 엘라스틱 클러스터 정보를 볼 수 있는 읽기 전용 권한을 사용자에게 부여합니다. 이 정책이 첨부된 보안 주체는 기존 리소스를 업데이트하거나 삭제할 수 없으며 새 Amazon DocumentDB 리소스를 생성할 수도 없습니다. 예를 들어 이러한 권한이 있는 주체는 자신의 계정과 연결된 클러스터 및 구성 목록을 볼 수 있지만 클러스터의 구성이나 설정을 변경할 수는 없습니다. 해당 정책의 권한은 다음과 같이 그룹화됩니다.

- Amazon DocumentDB 엘라스틱 클러스터 권한을 사용하면 Amazon DocumentDB 엘라스틱 클러스터 리소스를 나열하고, 설명하고, 그에 대한 정보를 얻을 수 있습니다.
- CloudWatch 권한은 서비스 지표를 확인하는 데 사용됩니다.

JSON

```
{
   "Version":"2012-10-17",
```

```
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "docdb-elastic:ListClusters",
            "docdb-elastic:GetCluster",
            "docdb-elastic:ListClusterSnapshots",
            "docdb-elastic:GetClusterSnapshot",
            "docdb-elastic:ListTagsForResource"
         ],
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": [
            "cloudwatch:GetMetricData",
            "cloudwatch:ListMetrics",
            "cloudwatch:GetMetricStatistics"
         ],
         "Resource": "*"
      }
   ]
}
```

AmazonDocDBElasticFullAccess

이 정책은 모든 Amazon DocumentDB 엘라스틱 클러스터의 Amazon DocumentDB 작업에 대한 전체 액세스 권한을 허용하는 관리 권한을 보안 주체에게 부여합니다.

이 정책은 조건 내에서 AWS 태그(https://docs.aws.amazon.com/tag-editor/latest/userguide/tagging.html)를 사용하여 리소스에 대한 액세스 범위를 지정합니다. 암호를 사용하는 경우 태그 키 `DocDBElasticFullAccess`과 태그 값으로 태그를 지정해야 합니다. 고객 관리형 키를 사용하는 경우 태그 키 `DocDBElasticFullAccess`과 태그 값으로 태그를 지정해야 합니다.

해당 정책의 권한은 다음과 같이 그룹화됩니다.

- Amazon DocumentDB 엘라스틱 클러스터 권한은 모든 Amazon DocumentDB 작업을 허용합니다.
- 이 정책의 일부 Amazon EC2 권한은 API 요청에서 전달된 리소스를 검증하는 데 필요합니다. 이는 Amazon DocumentDB가 클러스터에서 리소스를 성공적으로 프로비저닝하고 유지할 수 있도록 하기 위한 것입니다. 이 정책의 나머지 Amazon EC2 권한은 Amazon DocumentDB가 VPC 엔드포인트와 같은 클러스터에 연결하는 데 필요한 AWS 리소스를 생성하도록 허용합니다.

- AWS KMS Amazon DocumentDB가 전달된 키를 사용하여 Amazon DocumentDB 탄력적 클러스터 내에서 저장 데이터를 암호화하고 해독하려면 권한이 필요합니다.

> **Note**
>
> 고객 관리형 키에는 키 `DocDBElasticFullAccess`과 태그 값이 있는 태그가 있어야 합니다.

- 주어진 암호를 검증하고 이를 사용하여 Amazon DocumentDB 엘라스틱 클러스터의 관리자를 설정하려면 SecretsManager 권한이 필요합니다.

> **Note**
>
> 사용되는 암호에는 키 `DocDBElasticFullAccess`과 태그 값이 있는 태그가 있어야 합니다.

- 지표 및 로그 게시에 필요한 서비스 연결 역할을 생성하려면 IAM 권한이 필요합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DocdbElasticSid",
            "Effect": "Allow",
            "Action": [
                "docdb-elastic:CreateCluster",
                "docdb-elastic:UpdateCluster",
                "docdb-elastic:GetCluster",
                "docdb-elastic:DeleteCluster",
                "docdb-elastic:ListClusters",
                "docdb-elastic:CreateClusterSnapshot",
                "docdb-elastic:GetClusterSnapshot",
                "docdb-elastic:DeleteClusterSnapshot",
                "docdb-elastic:ListClusterSnapshots",
                "docdb-elastic:RestoreClusterFromSnapshot",
                "docdb-elastic:TagResource",
                "docdb-elastic:UntagResource",
                "docdb-elastic:ListTagsForResource",
```

```
                "docdb-elastic:CopyClusterSnapshot",
                "docdb-elastic:StartCluster",
                "docdb-elastic:StopCluster",
                "docdb-elastic:GetPendingMaintenanceAction",
                "docdb-elastic:ListPendingMaintenanceActions",
                "docdb-elastic:ApplyPendingMaintenanceAction"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "EC2Sid",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVpcEndpoint",
                "ec2:DescribeVpcEndpoints",
                "ec2:DeleteVpcEndpoints",
                "ec2:ModifyVpcEndpoint",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:DescribeAvailabilityZones",
                "secretsmanager:ListSecrets"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:CalledViaFirst": "docdb-elastic.amazonaws.com"
                }
            }
        },
        {
            "Sid": "KMSSid",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "*",
```

```
            "Condition": {
                "StringLike": {
                    "kms:ViaService": [
                        "docdb-elastic.*.amazonaws.com"
                    ],
                    "aws:ResourceTag/DocDBElasticFullAccess": "*"
                }
            }
        },
        {
            "Sid": "KMSGrantSid",
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/DocDBElasticFullAccess": "*",
                    "kms:ViaService": [
                        "docdb-elastic.*.amazonaws.com"
                    ]
                },
                "Bool": {
                    "kms:GrantIsForAWSResource": true
                }
            }
        },
        {
            "Sid": "SecretManagerSid",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:ListSecretVersionIds",
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:GetResourcePolicy"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "secretsmanager:ResourceTag/DocDBElasticFullAccess": "*"
                },
                "StringEquals": {
                    "aws:CalledViaFirst": "docdb-elastic.amazonaws.com"
```

```
                }
            }
        },
        {
            "Sid": "CloudwatchSid",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:ListMetrics",
                "cloudwatch:GetMetricStatistics"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "SLRSid",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/docdb-
elastic.amazonaws.com/AWSServiceRoleForDocDB-Elastic",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "docdb-elastic.amazonaws.com"
                }
            }
        }
    ]
}
```

AmazonDocDB-ElasticServiceRolePolicy

AWS Identity and Access Management 엔`AmazonDocDBElasticServiceRolePolicy`터티에 연결할 수 없습니다. 이 정책은 Amazon DocumentDB에 사용자를 대신하여 작업을 수행할 수 있도록 하는 서비스 연결 역할에 연결됩니다. 자세한 내용은 탄력적 클러스터에서 서비스 연결 역할 섹션을 참조하세요.

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": [
                        "AWS/DocDB-Elastic"
                    ]
                }
            }
        }
    ]
}
```

AWS 관리형 정책에 대한 Amazon DocumentDB 업데이트

변경	설명	Date
AmazonDocDBElasticFullAccess, AmazonDocDBConsoleFullAccess - 변경	보류 중인 유지 관리 작업을 추가하도록 정책이 업데이트되었습니다.	2/11/2025
AmazonDocDBElasticFullAccess, AmazonDocDBConsoleFullAccess - 변경	시작/중지 클러스터를 추가하고 클러스터 스냅샷 작업을 복사하도록 정책이 업데이트되었습니다.	2/21/2024
AmazonDocDBElasticReadOnlyAccess, AmazonDocDBElasticFullAccess - 변경	`cloudwatch:GetMetricData` 작업을 추가하기 위해 정책이 업데이트되었습니다.	6/21/2023
AmazonDocDBElasticReadOnlyAccess - 새 정책	Amazon DocumentDB 엘라스틱 클러스터를 위한 새로운 관리형 정책	6/8/2023

변경	설명	Date
AmazonDocDBElasticFullAccess - 새 정책	Amazon DocumentDB 엘라스틱 클러스터를 위한 새로운 관리형 정책	6/5/2023
AmazonDocDB-ElasticServiceRolePolicy - 새 정책	Amazon DocumentDB는 Amazon DocumentDB 탄력적 클러스터에 대한 새로운 AWS ServiceRoleForDocDB 탄력적 서비스 연결 역할을 생성합니다. Amazon DocumentDB	11/30/2022
AmazonDocDBConsoleFullAccess - 변경 사항	Amazon DocumentDB 글로벌 및 탄력적 클러스터 권한을 추가하도록 정책이 업데이트되었습니다.	11/30/2022
AmazonDocDBConsoleFullAccess, AmazonDocDBFullAccess, AmazonDocDBReadOnlyAccess - 새 정책	서비스를 시작합니다.	1/19/2017

Amazon DocumentDB API 권한: 작업, 리소스 및 조건 참조

IAM ID에 연결할 수 있는 Amazon DocumentDB에 대한 ID 기반 정책(IAM 정책) 사용 및 쓰기 권한 정책(ID 기반 정책)을 설정할 때 다음 단원을 참조하십시오.

다음 목록에는 각 Amazon DocumentDB API 작업이 각각 나열되어 있습니다. 목록에는 작업을 수행할 수 있는 권한을 부여할 수 있는 해당 작업, 권한을 부여할 수 있는 AWS 리소스, 세분화된 액세스 제어를 위해 포함할 수 있는 조건 키가 포함되어 있습니다. 정책의 `Action` 필드에서 작업을 지정하고, `Resource` 필드에서 리소스 값을 지정하고, `Condition` 필드에서 조건을 지정합니다. 조건에 대한 자세한 내용은 정책에서 조건 지정 섹션을 참조하세요.

Amazon DocumentDB 정책에서 AWS전체 조건 키를 사용하여 조건을 표시할 수 있습니다. AWS전체 키의 전체 목록은 IAM 사용 설명서의 사용 가능한 키를 참조하세요.

IAM 정책은 IAM 정책 시뮬레이터로 테스트할 수 있습니다. Amazon DocumentDB AWS 작업을 포함하여 각 작업에 필요한 리소스 및 파라미터 목록을 자동으로 제공합니다. 또한 IAM 정책 시뮬레이터 정책은 사용자가 지정하는 작업마다 필요한 권한을 결정합니다. IAM 정책 테스트에 대한 자세한 내용은 IAM 사용 설명서의 IAM 정책 시뮬레이터로 IAM 정책 테스트 단원을 참조하십시오.

> **Note**
>
> 작업을 지정하려면 `rds:` 접두사 다음에 API 작업 명칭을 사용합니다(예: `rds:CreateDBInstance`).

다음 표에는 Amazon RDS API 작업을 비롯한 관련 작업과 리소스, 그리고 조건 키가 나와 있습니다.

주제

- 리소스 수준 권한을 지원하는 Amazon DocumentDB 작업
- 리소스 수준 권한을 지원하지 않는 Amazon DocumentDB 작업

리소스 수준 권한을 지원하는 Amazon DocumentDB 작업

리소스 수준 권한은 사용자가 작업을 수행할 수 있는 리소스를 지정하는 기능을 제공합니다. Amazon DocumentDB는 리소스 수준 권한을 부분적으로 지원합니다. 즉, 필요 조건을 지정하거나 사용 가능한 특정 리소스를 지정하여 사용자가 특정 Amazon DocumentDB 작업 사용할 수 있는지를 제어할 수 있습니다. 예를 들어 특정 인스턴스로 제한된 변경 권한을 사용자에게 부여할 수 있습니다.

다음 표에는 Amazon DocumentDB API 작업을 비롯한 관련 작업과 리소스, 그리고 조건 키가 나와 있습니다.

> **Note**
>
> 일부 관리 기능의 경우 Amazon DocumentDB는 Amazon RDS와 공유되는 운영 기술을 사용합니다. Amazon DocumentDB 작업 및 권한에 대한 자세한 내용은 서비스 승인 참조에서 Amazon RDS에 사용되는 작업, 리소스 및 조건 키를 참조하세요.

Amazon DocumentDB API 작업 및 작업	리소스	조건 키
AddTagsToResource	Instance	`rds:db-tag`

Amazon DocumentDB API 작업 및 작업	리소스	조건 키
rds:AddTagsToResource	arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	
	서브넷 그룹 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
ApplyPendingMaintenanceAction rds:ApplyPendingMaintenanceAction	Instance arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
CopyDBClusterSnapshot rds:CopyDBClusterSnapshot	클러스터 스냅샷 arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
CreateDBCluster rds:CreateDBCluster	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	rds:cluster-tag
	클러스터 파라미터 그룹 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag

Amazon DocumentDB API 작업 및 작업	리소스	조건 키
	서브넷 그룹 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
CreateDBClusterParameterGroup rds:CreateDBClusterParameterGroup	클러스터 파라미터 그룹 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
CreateDBClusterSnapshot rds:CreateDBClusterSnapshot	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	rds:cluster-tag
	클러스터 스냅샷 arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
CreateDBInstance rds:CreateDBInstance	Instance arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:DatabaseClass rds:db-tag
	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	rds:cluster-tag

Amazon DocumentDB API 작업 및 작업	리소스	조건 키
CreateDBSubnetGroup rds:CreateDBSubnetGroup	서브넷 그룹 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
DeleteDBInstance rds:DeleteDBInstance	Instance arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
DeleteDBSubnetGroup rds:DeleteDBSubnetGroup	서브넷 그룹 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
DescribeDBClusterParameterGroups rds:DescribeDBClusterParameterGroups	클러스터 파라미터 그룹 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
DescribeDBClusterParameters rds:DescribeDBClusterParameters	클러스터 파라미터 그룹 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag

Amazon DocumentDB API 작업 및 작업	리소스	조건 키
DescribeDBClusters rds:DescribeDBClusters	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
DescribeDBClusterSnapshotAttributes rds:DescribeDBClusterSnapshotAttributes	클러스터 스냅샷 arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
DescribeDBSubnetGroups rds:DescribeDBSubnetGroups	서브넷 그룹 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
DescribePendingMaintenanceActions rds:DescribePendingMaintenanceActions	Instance arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:DatabaseClass rds:db-tag
FailoverDBCluster rds:FailoverDBCluster	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag

Amazon DocumentDB API 작업 및 작업	리소스	조건 키
ListTagsForResource rds:ListTagsForResource	Instance arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
	서브넷 그룹 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
ModifyDBCluster rds:ModifyDBCluster	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	rds:cluster-tag
	클러스터 파라미터 그룹 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
ModifyDBClusterParameterGroup rds:ModifyDBClusterParameterGroup	클러스터 파라미터 그룹 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag

Amazon DocumentDB API 작업 및 작업	리소스	조건 키
ModifyDBClusterSnapshotAttribute rds:ModifyDBClusterSnapshotAttribute	클러스터 스냅샷 arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
ModifyDBInstance rds:ModifyDBInstance	Instance arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:DatabaseClass rds:db-tag
RebootDBInstance rds:RebootDBInstance	Instance arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
RemoveTagsFromResource rds:RemoveTagsFromResource	Instance arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
	서브넷 그룹 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
ResetDBClusterParameterGroup rds:ResetDBClusterParameterGroup	클러스터 파라미터 그룹 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag

Amazon DocumentDB API 작업 및 작업	리소스	조건 키
RestoreDBClusterFromSnapshot rds:RestoreDBClusterFromSnapshot	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
	클러스터 스냅샷 arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
RestoreDBClusterToPointInTime rds:RestoreDBClusterToPointInTime	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
	서브넷 그룹 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag

리소스 수준 권한을 지원하지 않는 Amazon DocumentDB 작업

작업 사용 권한의 부여 여부를 결정하는 IAM 정책에서는 모든 Amazon DocumentDB 작업을 사용할 수 있습니다. 하지만 모든 Amazon DocumentDB 작업이 리소스 수준 권한을 지원하는 것은 아닙니다. 여기에서 리소스 수준 권한이란 작업이 가능한 리소스를 지정할 수 있는 권한을 말합니다. 현재 다음 Amazon DocumentDB API 작업은 리소스 수준 권한을 지원하지 않습니다. 따라서 IAM 정책에서 이러한 작업을 사용하려면 `Resource` 요소에 * 와일드카드를 사용하여 해당 작업 리소스를 모두 사용할 수 있는 권한을 사용자에게 부여해야 합니다.

- `rds:DescribeDBClusterSnapshots`
- `rds:DescribeDBInstances`

교차 서비스 혼동된 대리인 방지

혼동된 대리자 문제는 작업을 수행할 권한이 없는 엔터티가 권한이 더 많은 엔터티에게 작업을 수행하도록 강요할 수 있는 보안 문제입니다. 에서 AWS교차 서비스 가장은 혼동된 대리자 문제를 초래할 수 있습니다. 교차 서비스 가장은 한 서비스(직접 호출하는 서비스)가 다른 서비스(직접 호출되는 서비스)를 직접 호출할 때 발생할 수 있습니다. 직접 호출하는 서비스는 다른 고객의 리소스에 대해 액세스 권한이 없는 방식으로 작동하게 권한을 사용하도록 조작될 수 있습니다. 이를 방지하기 위해 AWS 에서는 계정의 리소스에 대한 액세스 권한이 부여된 서비스 위탁자를 사용하여 모든 서비스에 대한 데이터를 보호하는 데 도움이 되는 도구를 제공합니다.

리소스 정책에서 aws:SourceArn 및 aws:SourceAccount 전역 조건 컨텍스트 키를 사용하여 Amazon DocumentDB가 리소스에 다른 서비스를 제공하는 권한을 제한합니다. 하나의 리소스만 교차 서비스 액세스와 연결되도록 허용하려는 경우 aws:SourceArn을 사용하세요. 해당 계정의 모든 리소스가 교차 서비스 사용과 연결되도록 허용하려는 경우 aws:SourceAccount을(를) 사용합니다.

혼동된 대리자 문제로부터 보호하는 가장 효과적인 방법은 리소스의 전체 ARN이 포함된 aws:SourceArn 전역 조건 컨텍스트 키를 사용하는 것입니다. 리소스의 전체 ARN을 모르거나 여러 리소스를 지정하는 경우, ARN의 알 수 없는 부분에 대해 와일드카드 문자(*)를 포함한 aws:SourceArn 글로벌 조건 컨텍스트 키를 사용합니다. 예: arn:aws:*servicename*:*:*123456789012*:*

만약 aws:SourceArn 값에 Amazon S3 버킷 ARN과 같은 계정 ID가 포함되어 있지 않은 경우, 권한을 제한하려면 두 글로벌 조건 컨텍스트 키를 모두 사용해야 합니다.

의 값은 클러스터여야 aws:SourceArn 합니다.

다음 예제에서는 Amazon DocumentDB에서 aws:SourceArn 및 aws:SourceAccount 전역 조건 컨텍스트 키를 사용하여 혼동된 대리자 문제를 방지하는 방법을 보여줍니다.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "servicename.amazonaws.com"
    },
    "Action": "servicename:ActionName",
    "Resource": [
      "arn:aws:servicename:::ResourceName/*"
```

```
    ],
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:servicename:*:123456789012:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

IAM 자격 증명을 사용한 인증

Amazon DocumentDB 사용자 및 애플리케이션은 IAM 사용자 및 역할을 사용하여 Amazon DocumentDB 클러스터에 인증할 수 있습니다. Amazon DocumentDB IAM 인증은 비밀번호가 없는 인증 방법입니다. 또한 클라이언트 애플리케이션은 IAM 역할/사용자를 사용할 때 Amazon DocumentDB 클러스터에 비밀번호 암호를 전송하지 않습니다. 대신 클라이언트 연결은 임시 보안 토큰을 AWS STS 사용하여 인증됩니다. 이제 관리자가 아닌 사용자 및 애플리케이션은 다른 Amazon DocumentDB 클러스터 및 기타 AWS 서비스에 연결할 때 동일한 IAM 자격 증명 ARN을 사용할 수 있습니다.

암호 기반 인증과 IAM 인증을 모두 사용하여 Amazon DocumentDB 클러스터에 대한 사용자 및 애플리케이션을 인증하도록 선택할 수도 있습니다. IAM 인증은 Amazon DocumentDB 인스턴스 기반 클러스터 버전 5.0에서만 사용할 수 있습니다. Amazon DocumentDB 기본 사용자는 IAM 자격 증명 ARN을 사용한 IAM 인증이 지원되지 않습니다.

> **Note**
>
> 기본 사용자는 기존 비밀번호 기반 인증을 통해서만 인증할 수 있습니다.

주제

- IAM 사용자 및 역할을 사용한 인증 시작하기
- AWS IAM을 사용하여 Amazon DocumentDB에 인증하도록 AWS 컴퓨팅 유형 구성
- IAM 인증 요청 모니터링
- IAM 인증 사용
- IAM을 지원하는 드라이버
- IAM 자격 증명 인증 FAQ

IAM 사용자 및 역할을 사용한 인증 시작하기

IAM 자격 증명이 있는 Amazon DocumentDB 사용자 및 역할은 `$external` 데이터베이스에서 생성 및 관리됩니다.

사용자 생성

기본 사용자로 연결한 다음 IAM 사용자 및 역할을 생성합니다.

```
use $external;
db.createUser(
    {
        user: "arn:aws:iam::123456789123:user/iamuser",
        mechanisms: ["MONGODB-AWS"],
        roles: [ { role: "readWrite", db: "readWriteDB" } ]
    }
);
```

또는 IAM 역할을 사용하여 Amazon DocumentDB 사용자를 추가합니다.

```
use $external;
db.createUser(
    {
        user: "arn:aws:iam::123456789123:role/iamrole",
        mechanisms: ["MONGODB-AWS"],
        roles: [ { role: "readWrite", db: "readWriteDB" } ]
    }
);
```

IAM 사용자 또는 역할 수정

기존 IAM 사용자를 수정합니다.

```
use $external;
db.updateUser(
  "arn:aws:iam::123456789123:user/iamuser",
  {
    roles: [ { role: "read", db: "readDB" } ]
  }
);
```

기존 IAM 역할을 수정합니다.

```
use $external;
db.updateUser(
  "arn:aws:iam::123456789123:role/iamrole",
  {
    roles: [ { role: "read", db: "readDB" } ]
  }
);
```

IAM 사용자로부터 역할을 부여하거나 취소하려면

```
use $external;
db.grantRolesToUser(
  "arn:aws:iam::123456789123:user/iamuser",
  [ { db: "admin", role: "readWriteAnyDatabase" } ]
);
```

```
use $external;
db.revokeRolesFromUser(
  "arn:aws:iam::123456789123:user/iamuser",
  [ { db: "admin", role: "readWriteAnyDatabase" } ]
);
```

IAM 역할에서 역할을 부여하거나 취소하려면

```
use $external;
db.grantRolesToUser(
  "arn:aws:iam::123456789123:user/iamrole",
  [ { db: "admin", role: "readWriteAnyDatabase" } ]
);
```

```
use $external;
db.revokeRolesFromUser(
  "arn:aws:iam::123456789123:user/iamrole",
  [ { db: "admin", role: "readWriteAnyDatabase" } ]
);
```

IAM 사용자 또는 역할 삭제

기존 IAM 사용자를 삭제하려면

```
use $external;
db.dropUser("arn:aws:iam::123456789123:user/iamuser");
```

기존 IAM 역할을 삭제하려면

```
use $external;
db.dropUser("arn:aws:iam::123456789123:role/iamrole");
```

AWS IAM을 사용하여 인증하도록 연결 URI 구성

AWS IAM을 사용하여 인증하려면 다음 URI 파라미터를 authSource로 $external 사용하고 authMechanism로 사용합니다MONGODB-AWS. IAM 사용자를 사용하는 경우 사용자 이름과 암호 필드가 각각 액세스 키와 보안 키로 바뀝니다. IAM 역할을 수임하는 경우 현재 환경에 연결됩니다(예: AWS Lambda 함수, Amazon EC2 인스턴스). MONGODB-AWS 메커니즘을 사용하여 인증할 때 자격 증명을 구체적으로 전달할 필요가 없습니다. MONGODB-AWS 인증 메커니즘을 지원하는 MongoDB 드라이버를 사용하는 경우 드라이버는 컴퓨팅 인스턴스(예: Amazon EC2, Lambda 함수 등)에서 IAM 역할 자격 증명을 검색할 수도 있습니다. 다음 예제에서는 mongo 쉘을 사용하여 (IAM 사용자의 것) 액세스 키와 비밀 키를 수동으로 전달하여 MONGODB-AWS를 사용하여 인증하는 예제로 Amazon DocumentDB에 대한 인증을 보여줍니다.

다음 예제에서는 Python 코드를 사용하여 자격 증명을 명시적으로 전달하지 않고(환경에 연결된 IAM 역할 사용) MONGODB-AWS를 사용하여 인증하여 Amazon DocumentDB 에 대한 인증을 보여줍니다.

```
##Create a MongoDB client, open a connection to Amazon DocumentDB using an IAM role
    client = pymongo.MongoClient('mongodb://<DocDBEndpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false&authSource=
%24external&authMechanism=MONGODB-AWS')
```

다음 예제에서는 mongo 쉘을 사용하여 액세스 키 및 보안 키(IAM 사용자의 것)를 수동으로 전달하여 MONGODB-AWS 메커니즘을 사용하여 인증하는 것으로 Amazon DocumentDB에 대한 인증을 보여줍니다.

```
$ mongo 'mongodb://<access_key>:<secret_key>@<cluster_endpoint>:<db_port>/test?
authSource=%24external&authMechanism=MONGODB-AWS'
```

다음 예제에서는 mongo 쉘을 사용하여 자격 증명을 명시적으로 전달하지 않고(환경에 연결된 IAM 역할 사용) MONGODB-AWS를 사용하여 인증하여 Amazon DocumentDB 에 대한 인증을 보여줍니다.

```
$ mongo 'mongodb://<cluster_endpoint>:<db_port>/test?authSource=
%24external&authMechanism=MONGODB-AWS'
```

AWS IAM을 사용하여 Amazon DocumentDB에 인증하도록 AWS 컴퓨팅 유형 구성

Amazon EC2/AWS Lambda/AWS Fargate사용

Amazon EC2는 다음 환경 변수를 사용합니다. EC2 인스턴스에 연결된 IAM 역할 또는 Lambda 함수 또는 Amazon ECS 태스크와 연결된 실행 IAM 역할이 있는 경우 이러한 변수가 자동으로 채워지고 드라이버는 환경에서 이러한 값을 가져올 수 있습니다.

```
AWS_ACCESS_KEY_ID
AWS_SECRET_ACCESS_KEY
AWS_SESSION_TOKEN
```

환경 변수에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 Lambda 환경 변수 사용을 참조하세요.

Amazon EKS 사용

Amazon Elastic Kubernetes Service(Amazon EKS) 포드에 역할을 할당하면 다음 두 가지 환경 변수가 자동으로 설정됩니다.

```
AWS_WEB_IDENTITY_TOKEN_FILE - path of web identity token file
AWS_ROLE_ARN - Name of IAM role to connect with
```

이러한 변수의 도움을 받아에 대한 AWS SDK 호출을 사용하여 코드에서 역할을 수동으로 수임합니다`AssumeRoleWithWebIdentity`.

- `ProviderID` 파라미터를 생략합니다.
- `AWS_WEB_IDENTITY_TOKEN_FILE` 환경 변수에 설명된 파일에서 `WebIdentityToken` 파라미터 값을 찾습니다.

Amazon EKS에 관한 자세한 내용은 Amazon EKS 사용 설명서의 Amazon EKS란을 참조하세요.

IAM 인증 요청 모니터링

Amazon DocumentDB 감사 사용

Amazon CloudWatch 의 감사 로그 폴더로 이동하여 다른 검색 패턴을 사용하여 IAM 인증에 대한 로그를 가져옵니다. 예를 들어 `{ $.param.mechanism = "MONGODB-AWS" }`를 '모든 로그 스트림 검색'의 검색 패턴으로 사용합니다.

감사에서 지원되는 이벤트에 대한 자세한 내용은 Amazon DocumentDB 이벤트 감사 섹션을 참조하세요.

Amazon CloudWatch 지표 사용

`StsGetCallerIdentityCalls`:이 지표는 리전화된 AWS Security Token Service (AWS STS) 엔드포인트에 대해 Amazon DocumentDB 인스턴스가 수행하는 `GetCallerIdentity` 호출 수를 보여줍니다. 데이터베이스 인스턴스가 STS를 `GetCallerIdentity` 호출해야 하는 이유에 대한 `MONGODB-AWS` 인증 사양을 참조하세요.

IAM 인증 사용

자체 데이터베이스에서 사용자 이름과 암호를 관리하지 않으려면 IAM 인증을 사용할 수 있습니다. IAM 인증은 Amazon DocumentDB 인스턴스 기반 클러스터 버전 5.0에서만 사용할 수 있습니다.

IAM 인증은 AWS Security Token Service ()에 종속됩니다AWS STS. IAM 인증을 사용할 때 AWS STS 제한 예외가 발생하면 연결 속도를 낮춥니다.

IAM 할당량은 IAM 사용 설명서의 IAM 및 AWS STS 할당량을 참조하세요.

IAM을 지원하는 드라이버

Amazon DocumentDB 5.0 및 `MONGODB-AWS` 인증 메커니즘을 지원하는 드라이버는 Amazon DocumentDB 에서 IAM 인증 구현을 사용해야 합니다.

> **Important**
>
> Amazon DocumentDB에 대한 IAM ID 인증에서 지원되지 않는 Node.js 드라이버(6.13.1 이전 버전)에는 알려진 제한 사항이 있습니다. Node.js 드라이버를 사용하는 Node.js 드라이버 및 도구(예: mongosh)는 Node.js 드라이버 버전 6.13.1 이상을 사용하도록 업그레이드해야 합니다.

IAM 자격 증명 인증 FAQ

참조할 수 있는 샘플이 있나요?

샘플 사용 사례 및 구성은 다음 페이지를 참조하세요.

- 인간 사용자가 IAM 사용자 및 IAM 역할을 사용하여 Amazon DocumentDB에 인증하는 방법
- IAM 역할을 사용하여 Amazon DocumentDB에 대한 비밀번호 없는 인증

Python 드라이버를 사용하는 동안 “pymongo.errors.ConfigurationError: MONGODB-AWS authentication requires pymongo-auth-aws” 오류가 발생합니다. 이 문제를 해결하려면 어떻게 해야 하나요?

IAM 인증으로 Python 드라이버를 설치하는 동안 다음 문을 사용해야 합니다.

```
pip install 'pymongo[aws]'
```

그러면 IAM 인증이 작동하는 데 필요한 추가 AWS 종속성이 설치됩니다.

IAM 역할 임시 자격 증명이 만료되면 연결이 끊어집니까?

아니요. 임시 IAM 자격 증명은 연결 및 인증을 설정하는 데만 사용됩니다. 그런 다음 Amazon DocumentDB 클러스터에서 모든 추가 권한이 발생합니다. IAM 자격 증명이 교체/만료되더라도 연결이 끊기거나 경과되지 않습니다.

SCRAM-SHA-256을 사용한 인증

Amazon DocumentDB(MongoDB 호환)는 암호 기반 인증을 위한 Salted Challenge Response Authentication Mechanism(SCRAM)을 지원합니다. SCRAM을 사용하면 서버는 클라이언트가 암호를 서버에 전송하지 않고 사용자의 암호를 알고 있는지 확인합니다. Amazon DocumentDB는 `SCRAM-SHA-1` 및의 두 가지 SCRAM 메커니즘을 지원합니다`SCRAM-SHA-256`.는 더 강력한 SHA-256 해시 함수를 `SCRAM-SHA-256` 사용하고 FIPS 모드에서 작동합니다.

사용자를 생성하거나 업데이트할 때 사용자에게 SCRAM 메커니즘을 하나 또는 둘 다 할당합니다. 로 생성된 사용자는 변경 없이 `SCRAM-SHA-1` 계속 작동합니다. SCRAM-SHA-256 인증은 Amazon DocumentDB 인스턴스 기반 클러스터 마이너 버전 5.0.1 이상 및 8.0.1 이상에서 사용할 수 있습니다.

> SCRAM 및 IAM 인증은 상호 배타적입니다.
>
> 사용자는 SCRAM 메커니즘 또는 `MONGODB-AWS` 메커니즘을 가질 수 있지만 둘 다 가질 수는 없습니다. IAM 자격 증명 인증은 섹션을 참조하세요IAM 자격 증명을 사용한 인증.

주제

- SCRAM 메커니즘을 사용하여 사용자 생성
- 기존 사용자 업데이트
- SCRAM-SHA-256을 사용한 인증

SCRAM 메커니즘을 사용하여 사용자 생성

사용자를 생성할 때 mechanisms 필드로 활성화할 SCRAM 메커니즘을 지정할 수 있습니다. 다음 예제에서는 두 메커니즘SCRAM-SHA-256이 모두 포함된 사용자SCRAM-SHA-1만 생성합니다.

```
db.runCommand({
    createUser: 'sha1user',
    pwd: passwordPrompt(),
    roles: ['readWrite'],
    mechanisms: ['SCRAM-SHA-1']
});
db.runCommand({
    createUser: 'sha256user',
    pwd: passwordPrompt(),
    roles: ['readWrite'],
    mechanisms: ['SCRAM-SHA-256']
});
db.runCommand({
    createUser: 'bothuser',
    pwd: passwordPrompt(),
    roles: ['readWrite'],
    mechanisms: ['SCRAM-SHA-1','SCRAM-SHA-256']
});
```

기본 메커니즘

mechanisms 필드를 지정하지 않으면 Amazon DocumentDB는 사용자에 SCRAM-SHA-256 대해 SCRAM-SHA-1 및를 모두 활성화합니다. 다음 두 명령은 동일합니다.

```
db.runCommand({
    createUser: 'user1',
    pwd: passwordPrompt(),
    roles: ['readWrite']
});
```

```
// is equivalent to:
db.runCommand({
    createUser: 'user1',
    pwd: passwordPrompt(),
    roles: ['readWrite'],
    mechanisms: ['SCRAM-SHA-1','SCRAM-SHA-256']
});
```

기본적으로 두 메커니즘을 모두 활성화하면 만 지원하는 이전 MongoDB 드라이버와의 호환성이 유지됩니다SCRAM-SHA-1. 최신 드라이버와 FIPS 모드는를 선호합니다SCRAM-SHA-256. 사용자가 로 인증하지 않도록 하려면 모든 사용자의 SCRAM-SHA-256 mechanisms 필드에만를 SCRAM-SHA-1지정합니다.

기존 사용자 업데이트

에서 생성된 사용자는 변경 없이 SCRAM-SHA-1 계속 지원됩니다. 기존 사용자를에서 SCRAM-SHA-1로 전환하려면 새 암호와 SCRAM-SHA-256 메커니즘으로 사용자를 SCRAM-SHA-256업데이트합니다.

```
db.runCommand({
    updateUser: 'existing',
    pwd: passwordPrompt(),
    mechanisms: ['SCRAM-SHA-256']
});
```

SCRAM-SHA-256을 사용한 인증

드라이버는 각 연결에서 첫 번째 명령을 보내기 전에 연결 문자열 파라미터를 사용하여 자동으로 인증합니다. 연결 URISCRAM-SHA-256에서를 선택하려면 authMechanism 파라미터를 로 설정합니다SCRAM-SHA-256.

```
mongodb://username:password@hostname:port/authenticationDatabase?authMechanism=SCRAM-
SHA-256
```

mongosh에서 명시적 인증 요청을 보낼 수도 있습니다.

```
db.auth({user: 'sha256user', pwd: passwordPrompt(), mechanism: 'SCRAM-SHA-256'})
Enter password:
1
```

연결 옵션에서 인증 메커니즘을 지정하지 않으면 드라이버는 대상 데이터베이스 사용자가 지원하는 메커니즘을 자동으로 선택합니다.

Amazon DocumentDB 사용자 관리

Amazon DocumentDB에서 사용자는 암호와 함께 클러스터에 대해 인증합니다. 각 클러스터에는 클러스터 생성 중에 설정되는 기본 로그인 자격 증명이 있습니다.

> **Note**
>
> 2020년 3월 26일 이전에 생성된 모든 새 사용자에게는 `dbAdminAnyDatabase`, `readWriteAnyDatabase` 및 `clusterAdmin` 역할이 부여되었습니다. 모든 사용자를 재평가하고 필요에 따라 역할을 수정하여 클러스터의 모든 사용자에 대해 최소 권한을 적용하는 것이 좋습니다.
> 자세한 내용은 역할 기반 액세스 제어를 사용한 데이터베이스 액세스 섹션을 참조하세요.

기본 및 `serviceadmin` 사용자

새로 생성된 Amazon DocumentDB 클러스터에는 기본 사용자와 `serviceadmin` 사용자의 두 사용자가 있습니다.

기본 사용자는 관리 작업을 수행하고 역할을 가진 추가 사용자를 생성할 수 있는 권한이 있는 단일 사용자입니다. Amazon DocumentDB 클러스터에 처음 연결하는 경우 기본 로그인 자격 증명을 사용하여 인증해야 합니다. 기본 사용자는 클러스터가 생성될 때 Amazon DocumentDB 클러스터에 대한 이러한 관리 권한을 받고 `root`의 역할이 부여됩니다.

클러스터가 생성될 때 `serviceadmin` 사용자는 암시적으로 생성됩니다. 모든 Amazon DocumentDB 클러스터에는 클러스터 AWS 를 관리할 수 있는 기능을 제공하는 `serviceadmin` 사용자가 있습니다. `serviceadmin`으로 로그인하거나, 그 암호를 삭제, 변경, 이름을 바꾸거나, 또는 그 권한을 변경할 수 없습니다. 이렇게 하려고 하면 오류가 발생합니다.

> **Note**
>
> Amazon DocumentDB 클러스터의 기본 및 `serviceadmin` 사용자는 삭제할 수 없으며 기본 사용자에 대한 `root`의 역할은 취소할 수 없습니다.
> 기본 사용자 암호를 잊어버린 경우 AWS Management Console 또는를 사용하여 재설정할 수 있습니다 AWS CLI.

추가 사용자 생성

기본 사용자(또는 createUser 역할이 있는 사용자)로 연결한 후 아래와 같이 새 사용자를 생성할 수 있습니다.

```
db.createUser(
    {
        user: "sample-user-1",
        pwd: "password123",
        roles:
            [{"db":"admin", "role":"dbAdminAnyDatabase" }]
    }
)
```

사용자 세부 정보를 보려면 다음과 같이 show users 명령을 사용할 수 있습니다. dropUser 명령을 사용하여 사용자를 추가로 제거할 수 있습니다. 자세한 내용은 공통 명령 섹션을 참조하세요.

```
show users
{
    "_id" : "serviceadmin",
    "user" : "serviceadmin",
    "db" : "admin",
    "roles" : [
     {
            "role" : "root",
            "db" : "admin"
        }
    ]
},

{
    "_id" : "myPrimaryUser",
    "user" : "myPrimaryUser",
    "db" : "admin",
    "roles" : [
     {
            "role" : "root",
            "db" : "admin"
        }
    ]
},
```

```
{
    "_id" : "sample-user-1",
    "user" : "sample-user-1",
    "db" : "admin",
    "roles" : [
     {
            "role" : "dbAdminAnyDatabase",
            "db" : "admin"
     }
    ]
}
```

이 예에서 새 사용자 sample-user-1는 admin 데이터베이스에 기인합니다. 신규 사용자의 경우 항상 그렇습니다. Amazon DocumentDB에는 authenticationDatabase라는 개념이 없으므로 모든 인증은 admin 데이터베이스 컨텍스트에서 수행됩니다.

사용자를 만들 때 역할 지정 시 db 필드를 생략하면 Amazon DocumentDB에서 연결이 실행되는 데이터베이스에 역할의 속성을 암시적으로 지정합니다. 예를 들어, sample-database 데이터베이스에 대해 연결이 실행되고 다음 명령을 실행하면 sample-user-2 사용자가 admin 데이터베이스에 생성되고 sample-database 데이터베이스에 대한 readWrite 권한이 부여됩니다.

```
db.createUser(
    {
        user: "sample-user-2",
        pwd: "password123",
        roles:
            ["readWrite"]
    }
)
```

모든 데이터베이스에서 범위가 지정된 역할(예: readInAnyDatabase)을 가진 사용자를 생성하려면 사용자를 생성할 때 admin 데이터베이스 컨텍스트에 있거나 사용자를 생성할 때 역할에 대한 데이터베이스를 명백하게 명시해야 합니다.

데이터베이스의 컨텍스트를 전환하려면 다음 명령을 사용할 수 있습니다.

```
use admin
```

역할 기반 액세스 제어 및 클러스터의 사용자 사이에 최소 권한 적용에 대한 자세한 내용은 역할 기반 액세스 제어를 사용한 데이터베이스 액세스 섹션을 참조하십시오.

Amazon DocumentDB의 암호 자동 교체

를 사용하면 코드의 하드코딩된 자격 증명(암호 포함)을 Secrets Manager에 대한 API 호출로 대체하여 프로그래밍 방식으로 보안 암호를 검색할 AWS Secrets Manager수 있습니다. 이렇게 하면 보안 암호가 해당 위치에 있지 않기 때문에 여러분의 코드를 검사하는 누군가에 의해 보안 암호가 손상되지 않도록 방지할 수 있습니다. 또한 사용자가 지정한 일정에 따라 Secrets Manager가 자동으로 보안 암호를 교체하도록 구성할 수 있습니다. 따라서 단기 보안 암호로 장기 보안 암호를 교체할 수 있어 손상 위험이 크게 줄어듭니다.

Secrets Manager를 사용하면 Secrets Manager가 제공하는 AWS Lambda 함수를 사용하여 Amazon DocumentDB 암호(즉, 암호)를 자동으로 교체할 수 있습니다.

Amazon DocumentDB와의 AWS Secrets Manager 및 기본 통합에 대한 자세한 내용은 다음을 참조하세요.

- 블로그: AWS Secrets Manager에서 Amazon DocumentDB 및 Amazon Redshift의 자격 증명을 교체하는 방법
- AWS Secrets Manager란 무엇입니까?
- 비밀 AWS Secrets Manager 로테이트
- Secrets Manager의 Amazon DocumentDB 자격 증명

역할 기반 액세스 제어를 사용한 데이터베이스 액세스

Amazon DocumentDB (MongoDB 호환)의 역할 기반 액세스 제어 (RBAC)를 사용하여 데이터베이스에서 사용자가 수행할 수 있는 작업에 대한 액세스를 제한할 수 있습니다. RBAC는 사용자에게 하나 이상의 역할을 부여하여 작동합니다. 이러한 역할은 사용자가 데이터베이스 리소스에서 수행할 수 있는 작업을 결정합니다. Amazon DocumentDB는 현재, `read`, `readWrite`, `readAnyDatabase`, `clusterAdmin` 등 데이터베이스 수준에서 범위가 지정된 기본 제공 역할과, 요구 사항에 따라 특정 작업으로 범위를 지정할 수 있는 사용자 정의 역할과 컬렉션과 같은 세분화된 리소스를 모두 지원합니다.

RBAC의 일반적인 사용 사례에는 클러스터의 데이터베이스에 대한 읽기 전용 액세스 권한을 가진 사용자를 생성하여 최소 권한을 적용하는 것과 단일 사용자가 클러스터의 지정된 데이터베이스에 액세스할 수 있도록 하는 다중 테넌트 애플리케이션 디자인이 포함됩니다.

> **Note**
>
> 2020년 3월 26일 이전에 생성된 모든 새 사용자에게는 `dbAdminAnyDatabase`, `readWriteAnyDatabase` 및 `clusterAdmin` 역할이 부여되었습니다. 기존 사용자를 모두 재평가하고 필요에 따라 역할을 수정하여 클러스터에 대한 최소 권한을 적용하는 것이 좋습니다.

주제

- RBAC 개념
- RBAC 내장된 역할 시작하기
- RBAC 사용자 정의 역할 시작하기
- Amazon DocumentDB에 사용자로 연결
- 공통 명령
- 기능적 차이
- 한도
- 역할 기반 액세스 제어를 사용한 데이터베이스 액세스

RBAC 개념

다음은 역할 기반 액세스 제어와 관련된 중요한 용어와 개념입니다. Amazon DocumentDB 사용자에 관한 자세한 내용은 Amazon DocumentDB 사용자 관리 단원을 참조하세요.

- 사용자 - 데이터베이스에 대해 인증하고 작업을 수행할 수 있는 개별 엔터티입니다.
- 암호 - 사용자를 인증하는 데 사용되는 암호입니다.
- 역할 - 사용자에게 하나 이상의 데이터베이스에서 작업을 수행할 수 있는 권한을 부여합니다.
- 관리 데이터베이스 - 사용자가 저장되고 권한이 부여되는 데이터베이스입니다.
- 데이터베이스(**db**) - 문서를 저장하기 위한 컬렉션을 포함하는 클러스터 내의 네임스페이스입니다.

다음 명령은 `sample-user`라는 사용자를 생성합니다.

```
db.createUser({user: "sample-user", pwd: "abc123", roles: [{role: "read", db: "sample-database"}]})
```

이 예시는 다음과 같이 설정되어 있습니다.

- `user: "sample-user"` - 사용자 이름을 나타냅니다.
- `pwd: "abc123"` - 사용자 암호를 나타냅니다.
- `role: "read", "db: "sample-database"` - `sample-user` 사용자에게 `sample-database`에서 읽기 권한이 있음을 나타냅니다.


```
db.createUser({user: "sample-user", pwd: "abc123", roles: [{role: "read", db: "sample-database"}]})
```

다음 예제에서는 `db.getUser(sample-user)`를 사용하여 `sample-user` 사용자를 가져온 후 출력을 보여 줍니다. 이 예제에서 `sample-user` 사용자는 `admin` 데이터베이스에 상주하지만 `sample-database`데이터베이스에 대한 읽기 역할이 있습니다.

사용자를 만들 때 역할 지정 시 db 필드를 생략하면 Amazon DocumentDB에서 연결이 실행되는 데이터베이스에 역할의 속성을 암시적으로 지정합니다. 예를 들어, `sample-database` 데이터베이스에 대해 연결이 실행되고 다음 명령을 실행하면 `sample-user` 사용자가 `admin` 데이터베이스에 생성되고 `sample-database` 데이터베이스에 대한 `readWrite` 권한이 부여됩니다.

```
db.createUser({user: "sample-user", pwd: "abc123", roles: ["readWrite"]})
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
   "user":"sample-user",
   "roles":[
      {
```

```
            "db":"sample-database",
            "role":"readWrite"
        }
    ]
}
```

모든 데이터베이스에서 범위가 지정된 역할(예: readAnyDatabase)을 가진 사용자를 생성하려면 사용자를 생성할 때 admin 데이터베이스 컨텍스트에 있거나 사용자를 생성할 때 역할에 대한 데이터베이스를 명백하게 명시해야 합니다. admin 데이터베이스에 대해 명령을 실행하려면 use admin 명령을 사용할 수 있습니다. 자세한 내용은 공통 명령 섹션을 참조하세요.

RBAC 내장된 역할 시작하기

역할 기반 액세스 제어를 시작하는 데 도움이 되도록 이 단원에서는 서로 다른 작업 기능을 가진 세 명의 사용자에 대한 역할을 생성하여 최소 권한을 적용하는 예제 시나리오를 설명합니다.

- user1은 클러스터의 모든 데이터베이스를 보고 액세스할 수 있어야 하는 새로운 관리자입니다.
- user2는 동일한 클러스터에서 하나의 데이터베이스 sample-database-1에만 액세스해야 하는 신입 직원입니다.
- user3은 동일한 클러스터에서 이전에 액세스할 수 없었던 다른 데이터베이스 sample-database-2를 보고 액세스해야 하는 기존 직원입니다.

나중에 user1과 user2 모두 회사를 떠나서 그들의 액세스를 취소해야 합니다.

사용자를 생성하고 역할을 부여하려면 클러스터에 인증하는 사용자에게 createUser 및 grantRole에 대한 작업을 수행할 수 있는 연결된 역할이 있어야 합니다. 예를 들어 admin 및 userAdminAnyDatabase 역할은 모두 이러한 능력을 부여할 수 있습니다. 각 역할의 작업에 대한 자세한 내용은 역할 기반 액세스 제어를 사용한 데이터베이스 액세스 단원을 참조하십시오.

> **Note**
>
> Amazon DocumentDB에서 모든 사용자 및 역할 작업(예: create, get, drop, grant, revoke, admin 등)은 admin 데이터베이스에 대해 명령을 실행하는지 여부에 관계없이 데이터베이스에서 암시적으로 수행됩니다.

먼저 클러스터에 현재 사용자 및 역할이 무엇인지 이해하려면 다음 예제와 같이 show users 명령을 실행할 수 있습니다. 두 명의 사용자, 즉 serviceadmin과 클러스터의 기본 사용자가 표시됩니다. 이

두 사용자는 항상 존재하며 삭제할 수 없습니다. 자세한 내용은 Amazon DocumentDB 사용자 관리 섹션을 참조하세요.

```
show users
```

user1의 경우 다음 명령을 사용하여 전체 클러스터의 모든 데이터베이스에 대해 읽기 및 쓰기 액세스 권한이 있는 역할을 만듭니다.

```
db.createUser({user: "user1", pwd: "abc123", roles: [{role: "readWriteAnyDatabase", db:
 "admin"}]})
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
   "user":"user1",
   "roles":[
      {
         "role":"readWriteAnyDatabase",
         "db":"admin"
      }
   ]
}
```

user2의 경우 다음 명령을 사용하여 데이터베이스 sample-database-1에 대한 읽기 전용 액세스 권한이 있는 역할을 만듭니다.

```
db.createUser({user: "user2", pwd: "abc123", roles: [{role: "read", db: "sample-
database-1"}]})
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
   "user":"user2",
   "roles":[
      {
         "role":"read",
         "db":"sample-database-1"
      }
   ]
}
```

user3이 기존 사용자인 시나리오를 시뮬레이션하려면 먼저 사용자 user3를 만든 다음 user3에 새 역할을 할당합니다.

```
db.createUser({user: "user3", pwd: "abc123", roles: [{role: "readWrite", db: "sample-
database-1"}]})
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
   "user":"user3",
   "roles":[
      {
         "role":"readWrite",
         "db":"sample-database-1"
      }
   ]
}
```

이제 사용자 user3이 생성되었으므로 user3에 read, sample-database-2 역할을 할당합니다.

```
db.grantRolesToUser("user3", [{role: "read", db: "sample-database-2"}])
```

마지막으로, user1과 user2 모두 회사를 떠나고 클러스터에 대한 그들의 액세스가 취소되어야 합니다. 다음과 같이 사용자를 삭제하여 이 작업을 수행할 수 있습니다.

```
db.dropUser("user1")
db.dropUser("user2")
```

모든 사용자에게 적절한 역할이 있는지 확인하려면 다음 명령을 사용하여 모든 사용자를 나열할 수 있습니다.

```
show users
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
   "_id":"serviceadmin",
   "user":"serviceadmin",
```

```
   "db":"admin",
   "roles":[
      {
         "db":"admin",
         "role":"root"
      }
   ]
}
{
   "_id":"master-user",
   "user":"master-user",
   "db":"admin",
   "roles":[
      {
         "db":"admin",
         "role":"root"
      }
   ]
}
{
   "_id":"user3",
   "user":"user3",
   "db":"admin",
   "roles":[
      {
         "db":"sample-database-2",
         "role":"read"
      },
      {
         "db":"sample-database-1",
         "role":"readWrite"
      }
   ]
}
```

RBAC 사용자 정의 역할 시작하기

사용자 정의 역할을 시작하는 데 도움이 되도록 이 단원에서는 서로 다른 작업 기능을 가진 세 명의 사용자에 대한 역할을 생성하여 최소 권한을 적용하는 예제 시나리오를 설명합니다.

이 예에서 이하의 내용이 모두 적용됩니다.

- `user1`은 클러스터의 모든 데이터베이스를 보고 액세스할 수 있어야 하는 새로운 관리자입니다.

- user2은 동일한 클러스터에서 하나의 데이터베이스 sample-database-1에만 '찾기' 작업이 필요한 신입 직원입니다.
- user3은 동일한 클러스터에서 이전에 액세스할 수 없었던 다른 데이터베이스 sample-database-2 내의 특정 컬렉션, col2를 보고 액세스해야 하는 기존 직원입니다.
- user1의 경우 다음 명령을 사용하여 전체 클러스터의 모든 데이터베이스에 대해 읽기 및 쓰기 액세스 권한이 있는 역할을 만듭니다.

```
db.createUser(
 {
     user: "user1", pwd: "abc123",
     roles: [{role: "readWriteAnyDatabase", db: "admin"}]
 }
 )
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
   "user":"user1",
   "roles":[
      {
         "role":"readWriteAnyDatabase",
         "db":"admin"
      }
   ]
}
```

user2의 경우, 다음 명령을 사용하여 데이터베이스 sample-database-1의 모든 컬렉션에 대해 '찾기' 권한이 있는 역할을 생성합니다. 이 역할을 사용하면 연결된 모든 사용자가 찾기 쿼리만 실행할 수 있다는 점에 유의하세요.

```
db.createRole(
{
    role: "findRole",
    privileges: [
    {
        resource: {db: "sample-database-1", collection: ""}, actions: ["find"]
    }],
    roles: []
}
```

```
)
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
   "role":"findRole",
   "privileges":[
      {
         "resource":{
            "db":"sample-database-1",
            "collection":""
         },
         "actions":[
            "find"
         ]
      }
   ],
   "roles":[

   ]
}
```

그런 다음 사용자(user2)를 만들고 최근에 만든 역할을 findRole 사용자에게 연결합니다.

```
db.createUser(
{
    user: "user2",
    pwd: "abc123",
    roles: []
})

 db.grantRolesToUser("user2",["findRole"])
```

user3이 기존 사용자인 시나리오를 시뮬레이션하려면 먼저 사용자 user3를 만든 후, 다음 단계에서 user3에 새 역할을 할당할 collectionRole이라는 새 역할을 생성합니다.

이제 새 역할을 user3에 할당할 수 있습니다. 이 새 역할을 통해 user3이 sample-database-2 내의 하나의 특정 컬렉션 col2에 액세스를 삽입, 업데이트, 삭제하고 액세스 권한을 찾을 수 있습니다.

```
db.createUser(
{
    user: "user3",
```

```
    pwd: "abc123",
    roles: []
})

db.createRole(
{
    role: "collectionRole",
    privileges: [
    {
        resource: {db: "sample-database-2", collection: "col2"}, actions: ["find",
 "update", "insert", "remove"]
    }],
    roles: []
}
)
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
   "role":"collectionRole",
   "privileges":[
      {
         "resource":{
            "db":"sample-database-2",
            "collection":"col2"
         },
         "actions":[
            "find",
            "update",
            "insert",
            "remove"
         ]
      }
   ],
   "roles":[

   ]
}
```

이제 사용자 user3이 생성되었으므로 user3에 역할 collectionFind을 부여합니다.

```
db.grantRolesToUser("user3",["collectionRole"])
```

마지막으로, user1과 user2 모두 회사를 떠나고 클러스터에 대한 그들의 액세스가 취소되어야 합니다. 다음과 같이 사용자를 삭제하여 이 작업을 수행할 수 있습니다.

```
db.dropUser("user1")
db.dropUser("user2")
```

모든 사용자에게 적절한 역할이 있는지 확인하려면 다음 명령을 사용하여 모든 사용자를 나열할 수 있습니다.

```
show users
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
   "_id":"serviceadmin",
   "user":"serviceadmin",
   "db":"admin",
   "roles":[
      {
         "db":"admin",
         "role":"root"
      }
   ]
}
{
   "_id":"master-user",
   "user":"master-user",
   "db":"admin",
   "roles":[
      {
         "db":"admin",
         "role":"root"
      }
   ]
}
{
   "_id":"user3",
   "user":"user3",
   "db":"admin",
   "roles":[
      {
         "db":"admin",
```

```
         "role":"collectionRole"
      }
   ]
}
```

Amazon DocumentDB에 사용자로 연결

Amazon DocumentDB 클러스터에 연결할 때 특정 데이터베이스의 컨텍스트에서 연결합니다. 기본적으로 연결 문자열에 데이터베이스를 지정하지 않으면 `test` 데이터베이스의 컨텍스트에서 클러스터에 자동으로 연결됩니다. `test` 데이터베이스의 컬렉션에 대해 `insert` 및 `find` 같은 모든 컬렉션 레벨 명령이 발행됩니다.

컨텍스트에 있는 또는, 달리 말해, 명령이 발행되는 데이터베이스를 보려면 다음과 같이 mongo 쉘에서 db 명령을 사용하십시오.

쿼리:

```
db
```

출력:

```
test
```

기본 연결이 `test` 데이터베이스 컨텍스트에 있을 수 있지만 해당 연결과 연결된 사용자가 `test` 데이터베이스에서 작업을 수행할 수 있는 권한이 있는 것은 아닙니다. 앞의 예제 시나리오에서 `sample-database-1` 데이터베이스에 대한 `readWrite` 역할이 있는 사용자 `user3`으로 인증하는 경우 연결의 기본 컨텍스트는 `test` 데이터베이스입니다. 그러나 `test` 데이터베이스의 컬렉션에 문서를 삽입하려고 하면 권한 부여 실패 오류 메시지가 나타납니다. 아래와 같이 해당 사용자가 해당 데이터베이스에서 해당 명령을 수행할 권한이 없기 때문입니다.

쿼리:

```
db
```

출력:

```
test
```

쿼리:

```
db.col.insert({x:1})
```

출력:

```
WriteCommandError({ "ok" : 0, "code" : 13, "errmsg" : "Authorization failure" })
```

sample-database-1 데이터베이스에 대한 연결 컨텍스트를 변경하면 사용자가 그렇게 할 수 있는 권한을 가진 컬렉션에 쓸 수 있습니다.

쿼리:

```
use sample-database-1
```

출력:

```
switched to db sample-database-1
```

쿼리:

```
db.col.insert({x:1})
```

출력:

```
WriteResult({ "nInserted" : 1})
```

특정 사용자를 사용하여 클러스터에 인증하는 경우 연결 문자열에서 데이터베이스를 지정할 수도 있습니다. 이렇게 하면 사용자가 admin 데이터베이스에 인증된 후 use 명령을 수행할 필요가 없습니다.

다음 연결 문자열은 admin 데이터베이스에 대해 사용자를 인증하지만 연결 컨텍스트는 sample-database-1 데이터베이스에 대한 것입니다.

```
mongo "mongodb://user3:abc123@sample-cluster.node.us-east-1.docdb.amazonaws.com:27017/
sample-database-2"
```

공통 명령

이 단원에서는 Amazon DocumentDB에서 역할 기반 액세스 제어를 사용하는 공통 명령의 예를 제공합니다. 사용자 및 역할을 만들고 수정하려면 admin 데이터베이스의 컨텍스트에 있어야 합니다. use admin 명령을 사용하여 admin 데이터베이스로 전환할 수 있습니다.

> **Note**
>
> 사용자 및 역할에 대한 수정은 admin 데이터베이스에서 암시적으로 발생합니다. 모든 데이터베이스에서 범위가 지정된 역할(예: readAnyDatabase)을 가진 사용자를 생성하려면 사용자를 생성할 때 admin 데이터베이스(즉 use admin) 컨텍스트에 있거나 사용자를 생성할 때 역할에 대한 데이터베이스를 명백하게 명시해야 합니다(이 단원의 예제 2 참조).

예제 1: 데이터베이스foo에 대한 read 역할을 가진 사용자를 만듭니다.

```
db.createUser({user: "readInFooBar", pwd: "abc123", roles: [{role: "read", db:
 "foo"}]})
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
   "user":"readInFooBar",
   "roles":[
      {
         "role":"read",
         "db":"foo"
      }
   ]
}
```

예제 2: 모든 데이터베이스에서 읽기 권한이 있는 사용자를 만듭니다.

```
db.createUser({user: "readAllDBs", pwd: "abc123", roles: [{role: "readAnyDatabase", db:
 "admin"}]})
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
   "user":"readAllDBs",
   "roles":[
      {
         "role":"readAnyDatabase",
         "db":"admin"
      }
   ]
```

```
}
```

예제 3: 새 데이터베이스의 기존 사용자에게 read 역할을 부여합니다.

```
db.grantRolesToUser("readInFooBar", [{role: "read", db: "bar"}])
```

예제 4: 사용자의 역할을 업데이트합니다.

```
db.updateUser("readInFooBar", {roles: [{role: "read", db: "foo"}, {role: "read", db:
 "baz"}]})
```

예제 5: 사용자의 데이터베이스에 대한 액세스 권한을 취소합니다.

```
db.revokeRolesFromUser("readInFooBar", [{role: "read", db: "baz"}])
```

예제 6: 기본 제공 역할을 설명합니다.

```
db.getRole("read", {showPrivileges:true})
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
   "role":"read",
   "db":"sample-database-1",
   "isBuiltin":true,
   "roles":[

   ],
   "inheritedRoles":[

   ],
   "privileges":[
      {
         "resource":{
            "db":"sample-database-1",
            "collection":""
         },
         "actions":[
            "changeStream",
            "collStats",
            "dbStats",
```

```
            "find",
            "killCursors",
            "listCollections",
            "listIndexes"
         ]
      }
   ],
   "inheritedPrivileges":[
      {
         "resource":{
            "db":"sample-database-1",
            "collection":""
         },
         "actions":[
            "changeStream",
            "collStats",
            "dbStats",
            "find",
            "killCursors",
            "listCollections",
            "listIndexes"
         ]
      }
   }
```

예제 7: 클러스터에서 사용자를 삭제합니다.

```
db.dropUser("readInFooBar")
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
true
```

예제 8: 특정 컬렉션에 대한 읽기 및 쓰기 권한이 있는 역할 생성합니다.

```
db.createRole(
{
    role: "collectionRole",
    privileges: [
    {
        resource: {db: "sample-database-2", collection: "col2"}, actions: ["find",
 "update", "insert", "remove"]
```

```
    }],
    roles: []
}
)
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
   "role":"collectionRole",
   "privileges":[
      {
         "resource":{
            "db":"sample-database-2",
            "collection":"col2"
         },
         "actions":[
            "find",
            "update",
            "insert",
            "remove"
         ]
      }
   ],
   "roles":[

   ]
}
```

예제 9: 사용자를 생성하고 사용자 정의 역할을 할당합니다.

```
db.createUser(
{
    user: "user3",
    pwd: "abc123",
    roles: []
})

db.grantRolesToUser("user3",["collectionRole"])
```

예제 10: 사용자 정의 역할에 추가 권한을 부여합니다.

```
db.grantPrivilegesToRole(
```

```
  "collectionRole",
  [
    {
      resource: { db: "sample-database-1", collection: "col1" },
      actions: ["find", "update", "insert", "remove"]
    }
  ]
)
```

예제 11: 사용자 정의 역할에서 권한을 제거합니다.

```
db.revokePrivilegesFromRole(
  "collectionRole",
  [
    {
      resource: { db: "sample-database-1", collection: "col2" },
      actions: ["find", "update", "insert", "remove"]
    }
  ]
)
```

예제 12: 기존 사용자 정의 역할을 업데이트합니다.

```
db.updateRole(
  "collectionRole",
  {
    privileges: [
    {
        resource: {db: "sample-database-3", collection: "sample-collection-3"},
 actions: ["find", "update", "insert", "remove"]
    }],
    roles: []
  }
)
```

기능적 차이

Amazon DocumentDB에서, 사용자 및 역할 정의는 `admin` 데이터베이스에 저장되고 사용자는 `admin` 데이터베이스에 대해 인증됩니다. 이 기능은 MongoDB Community Edition과 다르지만 MongoDB Atlas와 일치합니다.

Amazon DocumentDB는 또한 클러스터의 컬렉션 내에서 발생하는 변경 이벤트 시퀀스를 시간 순서대로 제공하는 변경 스트림을 지원합니다. `listChangeStreams` 작업은 클러스터 수준에서(즉, 모든 데이터베이스에 걸쳐) 적용되고 `modifyChangeStreams` 작업은 데이터베이스 수준 및 클러스터 수준에서 적용됩니다.

한도

다음 표에는 Amazon DocumentDB의 역할 기반 액세스 제어에 대한 제한이 포함되어 있습니다.

설명	한도
클러스터당 사용자 수	1000
사용자와 연결된 역할 수	1000
사용자 정의 역할 수	100
권한과 관련된 리소스 수	100

역할 기반 액세스 제어를 사용한 데이터베이스 액세스

역할 기반 액세스 제어를 사용하면 사용자를 생성하고 사용자에게 하나 이상의 역할을 부여하여 사용자가 데이터베이스 또는 클러스터에서 수행할 수 있는 작업을 결정할 수 있습니다.

다음은 Amazon DocumentDB에서 현재 지원되는 기본 제공 역할 목록입니다.

> **Note**
>
> Amazon DocumentDB 4.0 및 5.0에서는, `ListCollection` 및 `ListDatabase` 명령을 사용하면 선택적으로 `authorizedCollections` 및 `authorizedDatabases` 매개변수를 사용하여 사용자가 각각 `listCollections` 및 `listDatabase` 역할을 요구하지 않고도 액세스 권한이 있는 컬렉션과 데이터베이스를 나열할 수 있습니다. 또한, 이제 사용자는 `KillCursor` 역할이 없이도 자신의 커서를 죽일 수도 있습니다.

Database user

역할 이름	설명	작업
read	사용자에게 지정된 데이터베이스에 대한 읽기 액세스 권한을 부여합니다.	changeStreams collStats dbStats find killCursors listIndexes listCollections
readWrite	사용자에게 지정된 데이터베이스에 대한 읽기 및 쓰기 액세스 권한을 부여합니다.	read 권한의 모든 작업. createCollection dropCollection createIndex dropIndex insert killCursors listIndexes listCollections remove

역할 이름	설명	작업
		update

Cluster user

역할 이름	설명	작업
readAnyDatabase	사용자에게 클러스터의 모든 데이터베이스에 대한 읽기 권한을 부여합니다.	read 권한의 모든 작업. listChangeStreams listDatabases
readWriteAnyDatabase	사용자에게 클러스터의 모든 데이터베이스에 대한 읽기 및 쓰기 액세스 권한을 부여합니다.	readWrite 권한의 모든 작업. listChangeStreams listDatabases
userAdminAnyDatabase	사용자에게 지정된 데이터베이스에 대해 사용자가 갖는 역할 또는 권한을 할당하고 수정할 수 있는 기능을 부여합니다.	changeCustomData changePassword createUser dropRole dropUser grantRole listDatabases revokeRole

역할 이름	설명	작업
		viewRole viewUser
dbAdminAnyDatabase	사용자에게 지정된 데이터베이스에서 데이터베이스 관리 역할을 수행할 수 있는 기능을 부여합니다.	dbAdmin 권한의 모든 작업. dropCollection listDatabases listChangeStreams modifyChangeStreams

Superuser

역할 이름	설명	작업
root	사용자에게 readWriteAnyDatabase , dbAdminAnyDatabase , userAdminAnyDatabase , clusterAdmin , restore및 backup 역할이 결합된 모든 역할의 리소스 및 작업에 대한 액세스 권한을 부여합니다.	readWriteAnyDatabase , dbAdminAnyDatabase , userAdminAnyDatabase , clusterAdmin , restore 및 backup의 모든 작업

Database administrator

역할 이름	설명	작업
dbAdmin	사용자에게 지정된 데이터베이스에서 관리 작업을 수행할 수 있는 기능을 부여합니다.	bypassDocumentValidation collMod collStats createCollection createIndex dropCollection dropDatabase dropIndex dbStats find killCursors listIndexes listCollections modifyChangeStreams
dbOwner	사용자에게 dbAdmin 및 readWrite 역할을 결합하여 지정된 데이터베이스에서 모든 관리 작업을 수행할 수 있는 기능을 부여합니다.	dbAdmin 및 readWrite 의 모든 작업.

Cluster administrator

역할 이름	설명	작업
clusterAdmin	clusterManager , clusterMonitor 및 hostManager 역할을 결합하여 사용자에게 가장 큰 클러스터 관리 액세스 권한을 부여합니다.	clusterManager , clusterMonitor 및 hostManager 의 모든 작업. listChangeStreams dropDatabase modifyChangeStreams
clusterManager	사용자에게 지정된 클러스터에서 관리 및 모니터링 작업을 수행할 수 있는 기능을 부여합니다.	listChangeStreams listSessions modifyChangeStreams replSetGetConfig
clusterMonitor	사용자에게 모니터링 도구에 대한 읽기 전용 액세스 기능을 부여합니다.	collStats dbStats find getParameter hostInfo indexStats killCursors

역할 이름	설명	작업
		listChangeStreams listCollections listDatabases listIndexes listSessions replSetGetConfig serverStatus top
hostManager	사용자에게 서버를 모니터링하고 관리할 수 있는 기능을 부여합니다.	auditConfigure killCursors killAnyCursor killAnySession killop

Backup administrator

역할 이름	설명	작업
backup	사용자에게 데이터를 백업하는 데 필요한 액세스 권한을 부여합니다.	getParameter insert find

역할 이름	설명		작업

			`listChangeStreams`
			`listCollections`
			`listDatabases`
			`listIndexes`
			`update`

역할 이름	설명	작업
restore	사용자에게 데이터를 복원하는 데 필요한 액세스 권한을 부여합니다.	bypassDocumentValidation changeCustomData changePassword collMod createCollection createIndex createUser dropCollection dropRole dropUser getParameter grantRole find insert listCollections modifyChangeStreams revokeRole

역할 이름	설명	작업

		`remove`
		
		`viewRole`
		
		`viewUser`
		
		`update`

Amazon DocumentDB TLS 인증서 업데이트

주제

- 애플리케이션 및 Amazon DocumentDB 클러스터 업데이트
- 자동 서버 인증서 교체
- 문제 해결
- FAQ

Amazon DocumentDB 클러스터의 인증 기관(CA) 인증서는 2024년 8월에 업데이트되었습니다. 전송 계층 보안(TLS)이 활성화(기본 설정)된 Amazon DocumentDB 클러스터를 사용하고 클라이언트 애플리케이션 및 서버 인증서를 교체하지 않은 경우, 애플리케이션과 Amazon DocumentDB 클러스터 간의 연결 문제를 완화하려면 다음 단계를 수행해야 합니다.

- 1단계: 새 CA 인증서 다운로드 및 애플리케이션 업데이트
- 2단계: 서버 인증서 업데이트

CA 및 서버 인증서는 Amazon DocumentDB에 대한 표준 유지 관리 및 보안 모범 사례의 일부로 업데이트되었습니다. 클라이언트 애플리케이션은 새 CA 인증서를 신뢰할 수 있는 스토어에 추가해야 하며, 이 만료 날짜 이전에 새 CA 인증서를 사용하도록 기존 Amazon DocumentDB 인스턴스를 업데이트해야 합니다.

애플리케이션 및 Amazon DocumentDB 클러스터 업데이트

이 섹션의 단계에 따라 애플리케이션의 CA 인증서 번들(1단계)과 클러스터의 서버 인증서(2단계)를 업데이트합니다. 프로덕션 환경에 변경 사항을 적용하기 전에 개발 또는 스테이징 환경에서 이러한 단계를 테스트하는 것이 좋습니다.

> **Note**
>
> Amazon DocumentDB 클러스터가 있는 각 AWS 리전 에서 1단계와 2단계를 완료해야 합니다.

1단계: 새 CA 인증서 다운로드 및 애플리케이션 업데이트

새 CA 인증서를 다운로드하고 응용 프로그램을 업데이트하여 새 CA 인증서를 사용하여 Amazon DocumentDB에 TLS 연결을 만듭니다. https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem에서 새 CA 인증서 번들을 다운로드합니다. 그러면 `global-bundle.pem`라는 파일이 다운로드됩니다.

> **Note**
>
> 이전 CA 인증서(`rds-ca-2019-root.pem`)와 새 CA 인증서(`rds-ca-rsa2048-g1`, `rds-ca-rsa4096-g1`, `rds-ca-ecc384-g1`)가 모두 포함된 키 스토어에 액세스하는 경우 키 스토어에서 `global-bundle` 항목을 선택하는지 확인하세요.

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

그런 다음 새 인증서 번들을 사용하도록 애플리케이션을 업데이트합니다. 새 CA 번들에는 이전 CA 인증서(rds-ca-2019)와 새 CA 인증서(rds-ca-rsa2048-g1, rds-ca-rsa4096-g1, rds-ca-ecc384-g1)가 모두 포함됩니다. 새 CA 번들에 두 CA 인증서가 모두 포함되어 있으므로 애플리케이션과 클러스터를 두 단계로 업데이트할 수 있습니다.

Java 애플리케이션의 경우 새 CA 인증서로 새 신뢰 스토어를 생성해야 합니다. 지침은 TLS가 활성화된 상태에서 연결 주제의 Java 탭을 참조하세요.

애플리케이션에서 최신 CA 인증서 번들을 사용하고 있는지 확인하려면 최신 CA 번들을 사용하고 있는지 어떻게 확인할 수 있습니까? 섹션을 참조하세요. 애플리케이션에서 최신 CA 인증서 번들을 이미 사용하고 있는 경우 2단계로 건너뛸 수 있습니다.

애플리케이션에 CA 번들을 사용하는 경우의 예는 전송 중 데이터 암호화 및 TLS가 활성화된 상태에서 연결 섹션을 참조하세요.

> **Note**
>
> 현재 MongoDB Go Driver 1.2.1은 `sslcertificateauthorityfile`에서 하나의 CA 서버 인증서만 허용합니다. TLS가 활성화된 경우 Go를 사용하여 Amazon DocumentDB에 연결하려면 섹션을 참조TLS가 활성화된 상태에서 연결하세요.

2단계: 서버 인증서 업데이트

응용프로그램이 새 CA 번들을 사용하도록 업데이트되면, 다음 단계는 Amazon DocumentDB 클러스터의 각 인스턴스를 수정하여 서버 인증서를 업데이트하는 것입니다. 새 서버 인증서를 사용하도록 인스턴스를 수정하려면 다음 지침을 참조하세요.

Amazon DocumentDB는 DB 인스턴스의 DB 서버 인증서에 서명할 수 있는 다음 CA를 제공합니다.

- rds-ca-ecc384-g1—ECC 384 프라이빗 키 알고리즘과 SHA384 서명 알고리즘을 갖춘 인증 기관을 사용합니다. 이 CA는 자동 서버 인증서 교체를 지원합니다. 이는 현재 Amazon DocumentDB 4.0 및 5.0에서만 지원됩니다.
- rds-ca-rsa2048-g1 - 대부분의에서 RSA 2048 프라이빗 키 알고리즘 및 SHA256 서명 알고리즘과 함께 인증 기관을 사용합니다 AWS 리전. 이 CA는 자동 서버 인증서 교체를 지원합니다.
- rds-ca-rsa4096-g1—RSA 4096 프라이빗 키 알고리즘과 SHA384 서명 알고리즘을 갖춘 인증 기관을 사용합니다. 이 CA는 자동 서버 인증서 교체를 지원합니다.

> **Note**
>
> 를 사용하는 경우 describe-certificates를 사용하여 이전 섹션에 나열된 인증 기관의 유효성을 확인할 AWS CLI수 있습니다.

이러한 CA 인증서는 지역 및 글로벌 인증서 번들에 포함되어 있습니다. rds-ca-rsa2048-g1, rds-ca-rsa4096-g1, rds-ca-ecc384-g1 CA를 데이터베이스에 사용하면 Amazon DocumentDB가 데이터베이스에서 DB 서버 인증서를 관리합니다. Amazon DocumentDB는 DB 서버 인증서를 만료되기 전에 자동으로 교체합니다.

> **Note**
>
> 클러스터가 다음 엔진 패치 버전에서 실행 중인 경우 Amazon DocumentDB는 인증서 교체를 위해 재부팅할 필요가 없습니다.

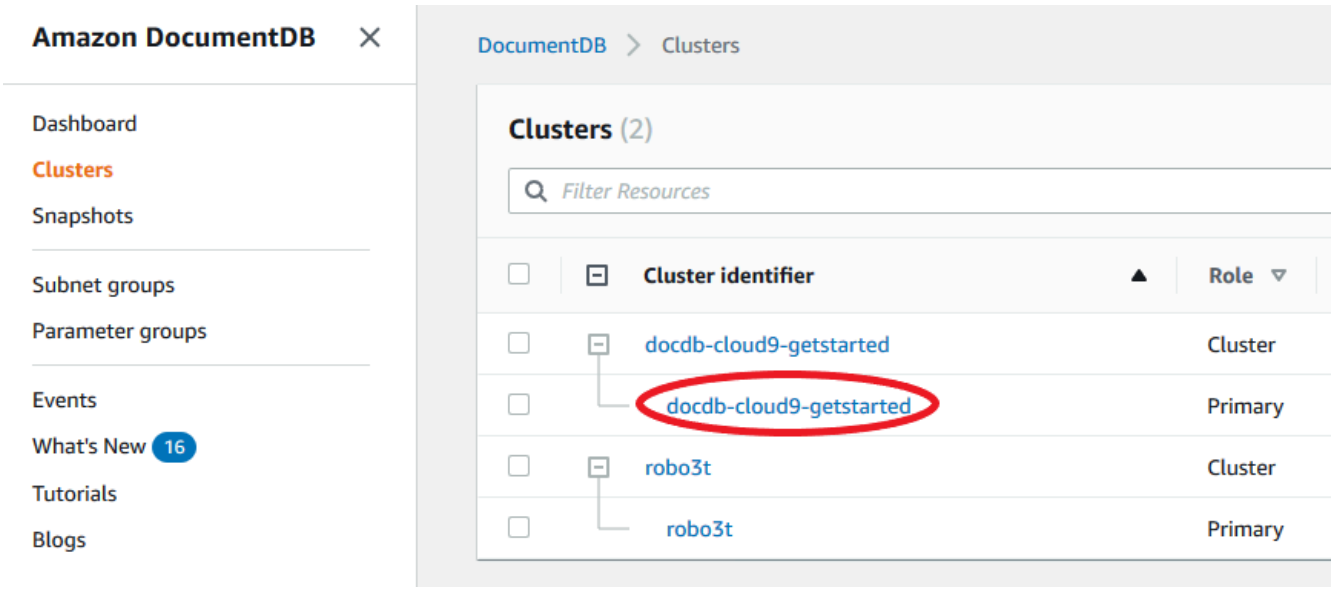
- Amazon DocumentDB 3.6: 1.0.208662 이상
- Amazon DocumentDB 4.0: 2.0.10179 이상
- Amazon DocumentDB 5.0: 3.0.4780 이상

현재 Amazon DocumentDB 엔진 패치 버전을 확인하려면 `db.runCommand({getEngineVersion: 1})`의 명령을 실행합니다. 서버 인증서를 업데이트하기 전에 1단계 항목을 완료했는지 확인합니다.

Using the AWS Management Console

AWS Management Console을 사용하여 기존 Amazon DocumentDB 인스턴스의 이전 서버 인증서를 식별하고 교체하기 위해 다음 단계를 완료하세요.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 화면 오른쪽 상단의 리전 목록에서 클러스터가 있는 AWS 리전 를 선택합니다.
3. 콘솔 왼쪽의 탐색 창에서 클러스터를 선택합니다.
4. 이전 서버 인증서 (`rds-ca-2019`) 에 아직 남아 있는 인스턴스를 식별해야 할 수도 있습니다. 클러스터 테이블 맨 오른쪽에 있는 인증 기관 열에서 이 작업을 수행할 수 있습니다.
5. 클러스터 테이블의 맨 왼쪽에 클러스터 식별자 열이 보일 것입니다. 인스턴스는 다음 스크린샷과 마찬가지로 클러스터 아래에 나열됩니다.



6. 관심 있는 인스턴스의 왼쪽에 있는 박스를 선택합니다.
7. 작업을 선택한 다음 수정을 선택합니다.
8. 인증 기관에서 이 인스턴스에 대한 새 서버 인증서(rds-ca-rsa2048-g1)를 선택합니다.
9. 다음 페이지에서 변경 사항 요약을 볼 수 있습니다. 연결이 중단되지 않도록 인스턴스를 수정하기 전에 애플리케이션이 최신 인증서 CA 번들을 사용하고 있는지 확인하는 추가 알림이 있습니다.
10. 다음 유지 관리 기간 중에 수정 사항을 적용하거나 즉시 적용하도록 선택할 수 있습니다. 서버 인증서를 즉시 수정하려는 경우 즉시 적용 옵션을 사용합니다.
11. Modify instance(인스턴스 수정)를 선택하여 업데이트를 완료합니다.

Using the AWS CLI

AWS CLI을 사용하여 기존 Amazon DocumentDB 인스턴스의 이전 서버 인증서를 식별하고 교체하기 위해 다음 단계를 완료하세요.

1. 인스턴스를 즉시 수정하려면 클러스터의 각 인스턴스에 대해 다음 명령을 실행합니다.

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier>
 --ca-certificate-identifier rds-ca-rsa2048-g1 --apply-immediately
```

2. 클러스터의 다음 유지 관리 기간 동안 새 CA 인증서를 사용하도록 클러스터의 인스턴스를 수정하려면 클러스터의 각 인스턴스에 대해 다음 명령을 실행합니다.

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier>
 --ca-certificate-identifier rds-ca-rsa2048-g1 --no-apply-immediately
```

자동 서버 인증서 교체

Amazon DocumentDB는 자동 서버 인증서 교체를 지원합니다. 서버 인증서는 각 클러스터 인스턴스에 발급된 리프 인증서입니다. 루트 CA 인증서와 달리 서버 인증서의 유효 기간은 짧으며(12개월) Amazon DocumentDB는 사용자의 조치 없이 자동으로 교체를 처리합니다. Amazon DocumentDB는 자동 교체에 동일한 루트 CA를 사용하므로 사용자가 새 CA 번들을 다운로드할 필요가 없습니다.

> **Important**
>
> Amazon DocumentDB 클러스터에 연결할 때 각 서버 인증서를 직접 신뢰하는 대신 루트 CA 번들을 신뢰합니다. 이를 통해 서버 인증서가 교체된 후 연결 오류가 방지됩니다. TLS가 활성화된 상태에서 연결 섹션을 참조하세요.

Amazon DocumentDB는 사용자가 선호하는 유지 관리 기간 내에 서버 인증서의 유효 기간이 반쯤 남은 시점에서 인증서 교체를 시도합니다. 새 서버 인증서는 12개월 동안 유효합니다.

`describe-db-engine-versions` 명령을 사용하고 `SupportsCertificateRotationWithoutRestart` 플래그를 검사하여 엔진 버전이 재시작 없이 인증서를 교체하는 기능을 지원하는지 파악합니다.

> **Note**
>
> 클러스터가 다음 엔진 패치 버전에서 실행 중인 경우 Amazon DocumentDB는 재시작 없이 서버 인증서를 교체하는 기능을 지원합니다.
>
> - Amazon DocumentDB 3.6: 1.0.208662 이상
> - Amazon DocumentDB 4.0: 2.0.10179 이상
> - Amazon DocumentDB 5.0: 3.0.4780 이상
>
> `db.runCommand({getEngineVersion: 1})` 명령을 실행하여 현재 Amazon DocumentDB 엔진 패치 버전을 확인할 수 있습니다.
> 이전 엔진 패치 버전을 사용하는 경우 Amazon DocumentDB는 서버 인증서를 교체하고 원하는 유지 관리 기간에 데이터베이스 재시작 이벤트를 예약합니다.

문제 해결

인증서 교체의 일부로 클러스터에 연결하는 데 문제가 있는 경우 다음을 시도하세요.

- 클라이언트가 최신 인증서 번들을 사용하고 있는지 확인합니다. 최신 CA 번들을 사용하고 있는지 어떻게 확인할 수 있습니까? 섹션을 참조하세요.
- 인스턴스가 최신 인증서를 사용하고 있는지 확인합니다. 내 Amazon DocumentDB 인스턴스가 이전/새 서버 인증서를 사용하고 있는지 어떻게 알 수 있습니까? 섹션을 참조하세요.

- 애플리케이션에서 최신 인증서 CA를 사용하고 있는지 확인합니다. Java 및 Go와 같은 일부 드라이버는 인증서 번들에서 신뢰 저장소로 여러 인증서를 가져오려면 추가 코드가 필요합니다. TLS를 사용하여 Amazon DocumentDB에 연결하는 방법에 대한 자세한 내용은 Amazon DocumentDB에 프로그래밍 방식으로 연결을 참조하세요.
- 고객 지원 센터에 문의.. 질문이나 문제가 있으면 지원에 연락하세요.

FAQ

다음은 TLS 인증서에 대한 몇 가지 일반적인 질문에 대한 답변입니다.

질문이나 문제가 있는 경우 어떻게 해야 합니까?

질문이나 문제가 있으면 지원에 연락하세요.

Amazon DocumentDB 클러스터에 연결하는 데 TLS를 사용했는지 어떻게 알 수 있습니까?

클러스터의 클러스터 파라미터 그룹에 대한 `tls` 파라미터를 확인하여 클러스터에서 TLS를 사용하고 있는지 확인할 수 있습니다. `tls` 파라미터가 `enabled`로 설정된 경우 TLS 인증서를 사용하여 클러스터에 연결되어 있는 것입니다. 자세한 내용은 Amazon DocumentDB 클러스터 파라미터 그룹 관리 섹션을 참조하세요.

CA와 서버 인증서를 업데이트하는 이유는 무엇입니까?

Amazon DocumentDB CA 및 서버 인증서는 Amazon DocumentDB에 대한 표준 유지 관리 및 보안 모범 사례의 일부로 업데이트되고 있습니다.

만료일까지 조치를 취하지 않으면 어떻게 됩니까?

만료된 CA 인증서를 사용하여 TLS를 통해 Amazon DocumentDB 클러스터에 연결하는 경우 TLS를 통해 연결하는 애플리케이션은 더 이상 Amazon DocumentDB 클러스터와 통신할 수 없습니다.

Amazon DocumentDB는 만료가 이뤄질 때까지 데이터베이스 인증서를 자동으로 교체하지 않습니다. 만료일 이전 또는 이후에 새 CA 인증서를 사용하려면 애플리케이션과 클러스터를 업데이트해야 합니다.

내 Amazon DocumentDB 인스턴스가 이전/새 서버 인증서를 사용하고 있는지 어떻게 알 수 있습니까?

이전 서버 인증서를 여전히 사용하는 Amazon DocumentDB 인스턴스를 식별하려면 Amazon DocumentDB AWS Management Console 또는를 사용할 수 있습니다 AWS CLI.

사용 AWS Management Console

이전 인증서를 사용하는 클러스터의 인스턴스를 식별하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 화면 오른쪽 상단의 리전 목록에서 인스턴스가 있는 AWS 리전 를 선택합니다.
3. 콘솔 왼쪽의 탐색 창에서 클러스터를 선택합니다.
4. 인증 기관 열(테이블의 맨 오른쪽 근처)에는 아직 이전 서버 인증서(`rds-ca-2019`)에 있는 인스턴스, 그리고 새 서버 인증서(`rds-ca-rsa2048-g1`)에 있는 인스턴스가 모두 표시됩니다.

사용 AWS CLI

이전 서버 인증서를 사용하는 클러스터의 인스턴스를 식별하려면 `describe-db-clusters` 명령을 다음과 함께 사용합니다.

```
aws docdb describe-db-instances \
    --filters Name=engine,Values=docdb \
    --query 'DBInstances[*].
{CertificateVersion:CACertificateIdentifier,InstanceID:DBInstanceIdentifier}'
```

Amazon DocumentDB 클러스터에서 개별 인스턴스를 수정하여 서버 인증서를 업데이트하려면 어떻게 해야 합니까?

지정된 클러스터의 모든 인스턴스에 대한 서버 인증서를 동시에 업데이트합니다. 클러스터의 인스턴스를 수정하기 위해 콘솔 또는 AWS CLI를 사용할 수 있습니다.

> **Note**
>
> 서버 인증서를 업데이트하기 전에 1단계를 완료했는지 확인합니다.

사용 AWS Management Console

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 화면 오른쪽 상단의 리전 목록에서 클러스터가 있는 AWS 리전 를 선택합니다.
3. 콘솔 왼쪽의 탐색 창에서 클러스터를 선택합니다.
4. 인증 기관 열(테이블의 맨 오른쪽 근처)에는 아직 이전 서버 인증서(`rds-ca-2019`)에 있는 인스턴스가 표시됩니다.
5. 클러스터 테이블의 클러스터 식별자에서 수정할 인스턴스를 선택합니다.
6. 작업을 선택한 다음 수정을 선택합니다.
7. 인증 기관에서 이 인스턴스에 대한 새 서버 인증서(`rds-ca-rsa2048-g1`)를 선택합니다.
8. 다음 페이지에서 변경 사항 요약을 볼 수 있습니다. 연결이 중단되지 않도록 인스턴스를 수정하기 전에 애플리케이션이 최신 인증서 CA 번들을 사용하고 있는지 확인하는 추가 알림이 있습니다.
9. 다음 유지 관리 기간 중에 수정 사항을 적용하거나 즉시 적용하도록 선택할 수 있습니다.
10. Modify instance(인스턴스 수정)를 선택하여 업데이트를 완료합니다.

사용 AWS CLI

AWS CLI을 사용하여 기존 Amazon DocumentDB 인스턴스의 이전 서버 인증서를 식별하고 교체하기 위해 다음 단계를 완료하세요.

1. 인스턴스를 즉시 수정하려면 클러스터의 각 인스턴스에 대해 다음 명령을 실행합니다.

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier> --
ca-certificate-identifier rds-ca-rsa2048-g1 --apply-immediately
```

2. 클러스터의 다음 유지 관리 기간 동안 새 CA 인증서를 사용하도록 클러스터의 인스턴스를 수정하려면 클러스터의 각 인스턴스에 대해 다음 명령을 실행합니다.

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier> --
ca-certificate-identifier rds-ca-rsa2048-g1 --no-apply-immediately
```

기존 클러스터에 새 인스턴스를 추가하면 어떻게 됩니까?

생성된 모든 새 인스턴스는 이전 서버 인증서를 사용하며, 이전 CA 인증서를 사용하는 TLS 연결이 필요합니다. 2024년 1월 25일 이후에 생성된 모든 새 Amazon DocumentDB 인스턴스는 기본적으로 새 인증서 rds-ca-rsa2048-g1을 사용합니다.

클러스터에 인스턴스 대체 또는 장애 조치가 있는 경우 어떻게 됩니까?

클러스터에 인스턴스 대체가 있는 경우, 생성된 새 인스턴스는 인스턴스가 이전에 사용했던 것과 동일한 서버 인증서를 계속 사용합니다. 모든 인스턴스에 대한 서버 인증서를 동시에 업데이트하는 것이 좋습니다. 클러스터에 장애 조치가 발생하면 새로운 기본 서버 인증서가 사용됩니다.

TLS를 사용하여 클러스터에 연결하지 않는 경우에도 각 인스턴스를 업데이트해야 합니까?

TLS를 활성화하는 것이 좋습니다. TLS를 활성화하지 않은 경우에도 TLS를 사용하여 향후 클러스터에 연결할 계획인 경우 Amazon DocumentDB 인스턴스에서 인증서를 교체하는 것이 좋습니다. Amazon DocumentDB 클러스터 연결에 TLS를 사용할 계획이 없다면, 별도로 필요한 작업이 없습니다.

TLS를 사용하여 클러스터에 연결하지 않고 나중에 연결할 계획이라면 어떻게 해야 합니까?

2024년 1월 이전에 클러스터를 생성한 경우 이전 섹션의 1단계 및 2단계를 수행하여 애플리케이션에서 업데이트된 CA 번들을 사용하고 있는지 및 각 Amazon DocumentDB 인스턴스가 최신 서버 인증서를 사용 중인지 확인합니다. 2024년 1월 25일 이후 클러스터를 생성한 경우, 클러스터에 이미 최신 서버 인증서(rds-ca-rsa2048-g1)가 있습니다. 애플리케이션이 최신 CA 번들을 사용하고 있는지 확인하려면 TLS를 사용하여 클러스터에 연결하지 않는 경우에도 각 인스턴스를 업데이트해야 합니까? 단원을 참조하십시오.

2024년 8월 이후로 기한을 연장할 수 있습니까?

애플리케이션이 TLS를 통해 연결되는 경우, 기한을 연장할 수 없습니다.

최신 CA 번들을 사용하고 있는지 어떻게 확인할 수 있습니까?

최신 번들인지 확인하려면 다음 명령을 사용하세요. 이 명령을 실행하려면 Java가 설치되어 있어야 하고 Java 도구가 쉘의 PATH 변수에 있어야 합니다. 자세한 내용은 Java 사용 섹션을 참조하세요.

macOS 및 Amazon Linux

```
keytool -printcert -v -file global-bundle.pem
```

Windows

```
keytool -printcert -v -file global-bundle.p7b
```

CA 번들 이름에 "RDS"가 표시되는 이유는 무엇입니까?

인증서 관리와 같은 일부 관리 기능의 경우 Amazon DocumentDB는 Amazon Relational Database Service(RDS)와 공유되는 운영 기술을 사용합니다.

새 인증서는 언제 만료됩니까?

새 서버 인증서는 (일반적으로) 다음과 같이 만료됩니다.

- rds-ca-rsa2048-g1—2061년 만료
- rds-ca-rsa4096-g1—2121년 만료
- rds-ca-ecc384-g1—2121년 만료

인증서가 만료되기 전에 조치를 취하지 않으면 어떤 종류의 오류가 발생하나요?

오류 메시지는 드라이버에 따라 달라집니다. 일반적으로 '인증서가 만료되었습니다'라는 문자열이 포함된 인증서 검증 오류가 표시됩니다.

새 서버 인증서를 적용한 경우 이전 서버 인증서로 되돌릴 수 있습니까?

인스턴스를 이전 서버 인증서로 되돌려야 하는 경우 클러스터의 모든 인스턴스에 대해 되돌립니다. AWS Management Console 또는를 사용하여 클러스터의 각 인스턴스에 대한 서버 인증서를 되돌릴 수 있습니다 AWS CLI.

사용 AWS Management Console

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 화면 오른쪽 상단의 리전 목록에서 클러스터가 있는 AWS 리전 를 선택합니다.
3. 콘솔 왼쪽의 탐색 창에서 클러스터를 선택합니다.
4. 클러스터 테이블의 클러스터 식별자에서 수정할 인스턴스를 선택합니다. 작업을 선택한 후 수정을 선택합니다.
5. 인증 기관에서 이전 서버 인증서(`rds-ca-2019`)를 선택할 수 있습니다.

6. 수정 사항의 요약을 보려면 계속을 선택합니다.
7. 이 결과 페이지에서 수정 사항이 다음 유지 관리 기간에 적용되도록 예약하거나 수정 사항을 즉시 적용하도록 선택할 수 있습니다. 선택한 다음 Modify instance(인스턴스 수정)를 선택합니다.

> **Note**
>
> 수정 사항을 즉시 적용하기로 선택하면 보류 중 수정 사항 대기열에 있는 변경 사항까지 모두 적용됩니다. 보류 중 수정 사항 중 하나라도 가동 중지를 필요로 하는 경우 이 옵션을 선택하면 예기치 못한 가동 중지가 발생할 수 있습니다.

사용 AWS CLI

```
aws docdb modify-db-instance --db-instance-identifier <db_instance_name> ca-
certificate-identifier rds-ca-2019 <--apply-immediately | --no-apply-immediately>
```

`--no-apply-immediately`를 선택하면, 클러스터의 다음 유지 관리 기간 중에 변경 사항이 적용됩니다.

스냅샷이나 특정 시점으로 복구를 통해 복원하는 경우 새 서버 인증서가 사용됩니까?

2024년 8월 이후 스냅샷을 복원하거나 시점 복원을 수행하면 생성된 새 클러스터에서 새 CA 인증서가 사용됩니다.

Mac OS X Catalina에서 내 Amazon DocumentDB 클러스터에 직접 연결하는 데 문제가 있으면 어떻게 됩니까?

Mac OS는 신뢰할 수 있는 인증서에 대한 요구 사항을 업데이트했습니다. 신뢰할 수 있는 인증서로 인정받으려면, 이제 유효 기간이 397일 이하여야 합니다(https://support.apple.com/en-us/HT211025 참조).

> **Note**
>
> 이 제한은 최신 버전의 Mac OS에서 적용됩니다.

Amazon DocumentDB 인스턴스 인증서는 4년 이상 유효하며 이는 Mac OS의 최대 유효 기간보다 더 깁니다. Mac OS를 실행하는 컴퓨터에서 Amazon DocumentDB 클러스터에 직접 연결하려면 TLS 연결을 만들 때 유효하지 않은 인증서를 허용해야 합니다. 이 경우 유효하지 않은 인증서는 유효 기간이

397일보다 길다는 것을 의미합니다. Amazon DocumentDB 클러스터에 연결할 때 유효하지 않은 인증서를 허용하기 전에 어떤 위험이 따를 수 있는지 이해해야 합니다.

를 사용하여 Mac OS에서 Amazon DocumentDB 클러스터에 연결하려면 `tlsAllowInvalidCertificates` 파라미터를 AWS CLI사용합니다.

```
mongo --tls --host <hostname> --username <username> --password <password> --port 27017
 --tlsAllowInvalidCertificates
```

Amazon DocumentDB에서의 규정 준수 검증

서드 파티 감사자는 다음을 포함하는 여러 AWS 규정 준수 프로그램의 일환으로 Amazon DocumentDB의 보안 및 규정 준수를 평가합니다.

- SOC(시스템 및 조직 제어) 1, 2, 3 자세한 내용은 SOC를 참조하세요.
- 연방정부의 위험 및 인증 관리 프로그램(FedRAMP). 자세한 내용은 규정 준수 프로그램 제공 범위 내AWS 서비스를 참조하세요.
- PCI DSS(지불 카드 산업 데이터 보안 표준) 자세한 내용은 PCI DSS를 참조하세요.
- ISO 9001, 27001, 27017 및 27018. 자세한 내용은 ISO 인증을 참조하세요.
- HIPAA(미국 건강 보험 양도 및 책임에 관한 법) BAA(비즈니스 제휴 계약) 자세한 내용은 HIPAA 규정 준수 섹션을 참조하세요.

AWS 는 규정 준수 프로그램 제공 범위 내 AWS 서비스에서 특정 AWS 규정 준수 프로그램 범위 내에서 자주 업데이트되는 서비스 목록을 제공합니다.

를 사용하여 서드 파티 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 AWS 아티팩트에서 보고서 다운로드를 참조하세요.

AWS 규정 준수 프로그램에 대한 자세한 내용은 규정 AWS 준수 프로그램을 참조하세요.

Amazon DocumentDB 사용 시 귀하의 규정 준수 책임은 데이터의 민감도, 조직의 규정 준수 목표, 관련 법률과 규정에 따라 결정됩니다. Amazon DocumentDB 사용 시 HIPAA 또는 PCI와 같은 표준을 준수해야 하는 경우는 도움이 되는 리소스를 AWS 제공합니다.

- AWS 규정 준수 리소스 - 업계 및 위치에 적용될 수 있는 워크북 및 가이드 모음입니다.
- 보안 및 규정 준수 빠른 시작 가이드 - 아키텍처 고려 사항을 설명하고 보안 및 규정 준수 중심 기준 환경을 배포하기 위한 단계를 제공하는 배포 가이드입니다 AWS.

- AWS 구성 - 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가하는 서비스입니다.
- AWS Security Hub - 보안 업계 표준 및 모범 사례 준수 여부를 확인하는 데 도움이 AWS 되는 내 보안 상태에 대한 포괄적인 보기입니다.
- HIPAA 보안 및 규정 준수를 위한 설계 백서 - 기업이를 사용하여 HIPAA 준수 애플리케이션을 AWS 생성하는 방법을 설명하는 백서입니다.

Amazon DocumentDB의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다.는 물리적으로 분리되고 격리된 여러 가용 영역을 AWS 리전 제공하며,이 가용 영역은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹과 연결됩니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 복수 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

Amazon DocumentDB 클러스터는 최소 2개 이상의 가용 영역에 서브넷이 2개 이상인 Amazon VPC에서만 생성할 수 있습니다. 두 개 이상의 가용 영역에 클러스터 인스턴스를 배포하면, 가용 영역에 장애가 발생할 경우에도 Amazon DocumentDB를 통해 클러스터에서 인스턴스를 사용할 수 있습니다. Amazon DocumentDB 클러스터의 클러스터 볼륨은 항상 세 개의 가용 영역에 배포되므로 데이터 손실의 가능성을 최소화하고 스토리지의 내구성을 높일 수 있습니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

AWS 글로벌 인프라 외에도 Amazon DocumentDB는 데이터 복원성과 백업 요구 사항을 지원하는 데 도움이 되는 몇 가지 기능을 제공합니다.

내결함성 및 자가 복구 스토리지

스토리지 볼륨의 각 10GB 부분은 세 개의 가용 영역에 걸쳐 여섯 가지 방식으로 복제됩니다. Amazon DocumentDB는 데이터베이스 쓰기 가용성에 영향을 주지 않고 최대 2개의 데이터 사본 손실을 투명하게 처리하고 읽기 가용성에 영향을 주지 않고 최대 3개의 데이터 사본 손실을 투명하게 처리하는 내결함성 스토리지를 사용합니다. 또한 Amazon DocumentDB 스토리지는 자가 복구 기능을 제공하므로 데이터 블록과 디스크를 지속적으로 스캔하여 오류가 있는지 확인하고 자동으로 교체합니다.

수동 백업 및 복원

Amazon DocumentDB는 장기 보존 및 복구를 위해 클러스터의 전체 백업을 생성할 수 있는 기능을 제공합니다. 자세한 내용은 Amazon DocumentDB에서 백업 및 복원 섹션을 참조하세요.

시점 복구

특정 시점으로 복구를 사용하면 우발적인 쓰기 또는 삭제 작업으로부터 Amazon DocumentDB 클러스터를 보호할 수 있습니다. 특정 시점으로 복구를 설정해 두면 온디맨드 백업의 생성, 유지 관리, 예약을 걱정할 필요가 없습니다. 자세한 내용은 특정 시점으로 복원 섹션을 참조하세요.

Amazon DocumentDB의 인프라 보안

관리형 서비스인 Amazon DocumentDB는 AWS 글로벌 네트워크 보안으로 보호됩니다. AWS 보안 서비스 및가 인프라를 AWS 보호하는 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안 모범 사례를 사용하여 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 인프라 보호를 참조하세요 AWS .

AWS 에서 게시한 API 호출을 사용하여 네트워크를 통해 Amazon DocumentDB에 액세스합니다. 클라이언트는 다음을 지원해야 합니다.

- Transport Layer Security(TLS). TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

어떤 네트워크 위치에서든 이러한 API 작업을 호출할 수 있습니다. Amazon DocumentDB 정책을 사용하여 특정 Amazon Virtual Private Cloud(Amazon VPC) 엔드포인트 또는 특정 VPC에서 액세스를 제어할 수도 있습니다. 이렇게 하면 네트워크 내의 특정 VPC에서만 지정된 Amazon DocumentDB 리소스에 대한 AWS 네트워크 액세스가 효과적으로 격리됩니다.

> **Note**
>
> Amazon DocumentDB는 리소스 기반 액세스 정책을 지원하지 않습니다.

Amazon DocumentDB API 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)

> **Note**
>
> Amazon DocumentDB 탄력적 클러스터는 AWS PrivateLink VPC 엔드포인트를 지원하지 않습니다.

인터페이스 VPC 엔드포인트를 생성하여 VPC와 Amazon DocumentDB API 엔드포인트 간에 프라이빗 연결을 설정할 수 있습니다. 인터페이스 엔드포인트는 로 구동됩니다 AWS PrivateLink.

Amazon DocumentDB 인스턴스 기반 클러스터에는 인터페이스 VPC 엔드포인트 연결이 필요하지 않지만, 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 Direct Connect 연결 없이 Amazon DocumentDB API 작업에 비공개로 액세스할 수 AWS PrivateLink 있습니다. VPC의 Amazon DocumentDB 인스턴스는 데이터베이스 인스턴스 및 데이터베이스 클러스터를 시작, 수정 또는 종료하기 위해 Amazon DocumentDB API 엔드포인트와 통신하는 데 퍼블릭 IP 주소가 필요하지 않습니다. 또한 Amazon DocumentDB 인스턴스가 사용 가능한 Amazon DocumentDB API 작업을 사용하기 위해 퍼블릭 IP 주소가 필요하지 않습니다. VPC와 Amazon DocumentDB 간의 트래픽은 Amazon 네트워크를 벗어나지 않습니다.

각 인터페이스 엔드포인트는 서브넷에서 하나 이상의 탄력적 네트워크 인터페이스로 표현됩니다. 자세한 내용은 Amazon EC2 사용 설명서의 탄력적 네트워크 인터페이스를 참조하세요.

VPC 엔드포인트에 대한 자세한 내용은 Amazon Virtual Private Cloud(AWS PrivateLink) 사용 설명서의 인터페이스 VPC 엔드포인트를 AWS 서비스 사용하여 액세스를 참조하세요. Amazon DocumentDB 작업에 대한 자세한 내용은 Amazon DocumentDB API 참조를 참조하세요.

주제

- VPC 엔드포인트에 대한 고려 사항
- 리전 가용성
- Amazon DocumentDB API에 대한 인터페이스 VPC 엔드포인트 생성
- Amazon DocumentDB API에 대한 VPC 엔드포인트 정책 생성

VPC 엔드포인트에 대한 고려 사항

Amazon DocumentDB API 엔드포인트에 대한 인터페이스 VPC 엔드포인트를 설정하기 전에 Amazon Virtual Private Cloud(AWS PrivateLink) 사용 설명서의 인터페이스 엔드포인트 사전 조건을 검토해야 합니다.

Amazon DocumentDB 리소스 관리와 관련된 모든 Amazon DocumentDB API 작업은 AWS PrivateLink를 사용하여 VPC에서 수행할 수 있습니다.

VPC 엔드포인트 정책은 Amazon DocumentDB API 엔드포인트에 대해 지원됩니다. 기본적으로, 엔드포인트를 통해 Amazon DocumentDB API 작업에 대한 전체 액세스가 허용됩니다. 자세한 내용은 Amazon 버추얼 프라이빗 클라우드(AWS PrivateLink) 사용 설명서의 엔드포인트 정책을 사용한 VPC 엔드포인트에 대한 액세스 제어를 참조하세요.

리전 가용성

Amazon DocumentDB API는 현재 AWS 리전다음에서 VPC 엔드포인트를 지원합니다.

- 미국 동부(오하이오)
- 미국 동부(버지니아 북부)
- 미국 서부(오레곤)
- 아프리카(케이프타운)
- 아시아 태평양(홍콩)
- 아시아 태평양(뭄바이)
- 아시아 태평양(하이데라바드)
- 아시아 태평양(오사카)
- 아시아 태평양(서울)
- 아시아 태평양(싱가포르)
- 아시아 태평양(시드니)
- 아시아 태평양(도쿄)
- 캐나다(중부)
- 중국(베이징)
- 중국(닝샤)
- 유럽(프랑크푸르트)

- 유럽(아일랜드)
- 유럽(런던)
- 유럽(파리)
- 유럽(스페인)
- 유럽(밀라노)
- 중동(UAE)
- 남아메리카(상파울루)
- AWS GovCloud(미국 동부)
- AWS GovCloud(미국 서부)

Amazon DocumentDB API에 대한 인터페이스 VPC 엔드포인트 생성

Amazon VPC 콘솔 또는 ()를 사용하여 Amazon DocumentDB API에 대한 VPC 엔드포인트를 생성할 수 있습니다AWS CLI. AWS Command Line Interface 자세한 내용은 Amazon Virtual Private Cloud(AWS PrivateLink) 사용 설명서의 인터페이스 VPC 엔드포인트를 AWS 서비스 사용하여 액세스를 참조하세요.

서비스 이름 `com.amazonaws.region.rds`을 사용하여 Amazon DocumentDB API에 대한 VPC 엔드포인트를 생성합니다.

중국을 제외하고 엔드포인트 AWS 리전 에 대해 프라이빗 DNS를 활성화한 경우 rds.us-east-1.amazonaws.com AWS 리전같은에 대한 기본 DNS 이름을 사용하여 VPC 엔드포인트로 Amazon DocumentDB에 API 요청을 할 수 있습니다. 중국(베이징) 및 중국(닝샤)의 경우 각각 rds-api.cn-north-1.amazonaws.com.cn 및 rds-api.cn-northwest-1.amazonaws.com.cn 사용하여 VPC 엔드포인트로 API 요청을 수행할 AWS 리전수 있습니다.

자세한 내용은 Amazon Virtual Private Cloud(AWS PrivateLink) 사용 설명서의 인터페이스 VPC 엔드포인트를 AWS 서비스 사용하여 액세스를 참조하세요.

Amazon DocumentDB API에 대한 VPC 엔드포인트 정책 생성

Amazon DocumentDB API에 대한 액세스를 제어하는 VPC 엔드포인트에 엔드포인트 정책을 연결할 수 있습니다. 이 정책은 다음 정보를 지정합니다.

- 작업을 수행할 수 있는 위탁자.

- 수행할 수 있는 작업.
- 작업을 수행할 수 있는 리소스

자세한 내용은 Amazon 버추얼 프라이빗 클라우드(AWS PrivateLink) 사용 설명서의 엔드포인트 정책을 사용한 VPC 엔드포인트에 대한 액세스 제어를 참조하세요.

예제: Amazon DocumentDB API 작업에 대한 VPC 엔드포인트 정책

다음은 Amazon DocumentDB API에 대한 엔드포인트 정책의 예입니다. 이 정책은 엔드포인트에 연결될 때 모든 리소스의 모든 보안 주체에 대한 액세스 권한을 나열된 Amazon DocumentDB API 작업에 부여합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Principal": "*",
      "Effect": "Allow",
      "Action": [
        "rds:CreateDBInstance",
        "rds:ModifyDBInstance",
        "rds:CreateDBSnapshot"
      ],
      "Resource": "*"
    }
  ]
}
```

예: 지정된의 모든 액세스를 거부하는 VPC 엔드포인트 정책 AWS 계정

다음 VPC 엔드포인트 정책은 엔드포인트를 사용하여 리소스에 대한 모든 액세스를 거부합니다 AWS 계정 123456789012. 이 정책은 다른의 모든 작업을 허용합니다 AWS 계정.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
```

```
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
    }
  ]
}
```

Amazon DocumentDB에 대한 보안 모범 사례

Amazon DocumentDB는 Amazon VPC 격리,를 사용한 저장 데이터 암호화 AWS KMS, TLS를 사용한 전송 중 데이터 암호화, IAM을 통한 세분화된 액세스 제어를 비롯한 보안 기능을 제공합니다. 다음 모범 사례를 사용하여 Amazon DocumentDB 클러스터, 인스턴스 및 데이터를 보호합니다.

AWS Identity and Access Management (IAM) 역할을 사용하여 Amazon DocumentDB API 작업에 대한 액세스를 제어합니다. 여기에는 클러스터, 보안 그룹 및 파라미터 그룹과 같은 Amazon DocumentDB 리소스를 생성, 수정 또는 삭제하는 작업이 포함됩니다. 애플리케이션 또는 기타 AWS 서비스가 Amazon DocumentDB 리소스에 액세스해야 하는 경우 애플리케이션에 장기 자격 증명을 저장하는 대신 IAM 역할을 사용하여 임시 자격 증명을 제공합니다. IAM 역할을 생성할 때 최소 권한 원칙을 적용합니다. 자세한 내용은 IAM 사용 설명서의 IAM 역할 및 역할에 대한 일반적인 시나리오를 참조하세요.

- 역할 기반 액세스 제어와 함께 최소 권한을 적용합니다.
- AWS 계정 루트 사용자를 사용하여 Amazon DocumentDB 리소스를 관리하지 마세요. 대신 AWS Identity and Access Management Identity Center를 사용하여 임시 자격 증명으로 인간 사용자를 생성합니다. 자세한 내용은 IAM 사용 설명서의 IAM Identity Center 사용자를 참조하세요.
- 각 자격 증명에 직무를 수행하는 데 필요한 최소 권한 세트를 부여합니다.
- IAM 정책을 사용하여 권한을 관리하고 최소 권한 원칙을 따릅니다. 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.
- 장기 자격 증명이 있는 IAM 사용자를 사용하는 경우 해당 자격 증명을 정기적으로 교체합니다.

> **Warning**
>
> 이 시나리오에서는 프로그래밍 방식 액세스 권한과 장기 보안 인증이 있는 IAM 사용자가 필요하며 이는 보안 위험을 내포합니다. 이 위험을 줄이려면 이러한 사용자에게 작업을 수행하는 데 필요한 권한만 제공하고 더 이상 필요하지 않을 경우 이러한 사용자를 제거하는 것이

좋습니다. 필요한 경우 액세스 키를 업데이트할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 액세스 키 업데이트를 참조하세요.

- Secrets Manager를 구성하여 Amazon DocumentDB에 대한 비밀번호를 자동으로 교체합니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 Amazon DocumentDB의 보안 암호 교체 및 보안 암호 교체를 참조하세요. Amazon DocumentDB
- TLS(전송 계층 보안) 및 비활성화되어 있는 암호화를 사용하여 데이터를 암호화합니다.

Amazon DocumentDB 이벤트 감사

Amazon DocumentDB(MongoDB 호환)에서는 클러스터에서 수행된 이벤트를 감사할 수 있습니다. 기록되는 이벤트의 예제로는 성공 또는 실패한 인증 시도, 데이터베이스에서 모음 삭제 또는 인덱스 생성이 있습니다. 기본적으로 Amazon DocumentDB에서 감사는 비활성화되어 있으므로 사용자가 이 기능을 선택해야 합니다.

감사가 활성화되면 Amazon DocumentDB가 데이터 정의 언어(DDL), 데이터 조작 언어(DML), 인증, 권한 부여 및 사용자 관리 이벤트를 Amazon CloudWatch Logs에 기록합니다. 감사가 활성화되면 Amazon DocumentDB가 클러스터의 감사 기록(JSON 문서)을 Amazon CloudWatch Logs로 내보냅니다. Amazon CloudWatch Logs를 사용하여 Amazon DocumentDB 감사 이벤트를 분석, 모니터링 및 보관할 수 있습니다.

Amazon DocumentDB가 감사 활성화에 대해 추가 요금을 부과하지는 않지만 CloudWatch Logs 사용에 따른 표준 요금이 부과됩니다. CloudWatch Logs 요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하세요.

Amazon DocumentDB 감사 기능은 로 모니터링되는 서비스 리소스 사용량과 크게 다릅니다 AWS CloudTrail. CloudTrail은 AWS Command Line Interface (AWS CLI) 또는 클러스터, 인스턴스, 파라미터 그룹 및 스냅샷과 같은 리소스 AWS Management Console 에서 수행되는 작업을 기록합니다. CloudTrail을 사용한 리소스에 대한 감사는 기본적으로 활성화되어 있으며 비활성화할 수 없습니다. Amazon DocumentDB 감사 기능은 옵트인 기능입니다. 데이터베이스, 컬렉션, 인덱스 및 사용자와 같은 객체의 클러스터 내에서 발생하는 작업을 기록합니다.

주제

- 지원되는 이벤트
- 감사 활성화
- 감사 비활성화
- 감사 이벤트 액세스

- DML 감사 이벤트 필터링

지원되는 이벤트

Amazon DocumentDB 감사는 다음 이벤트 카테고리를 지원합니다.

- 데이터 정의 언어 (DDL) - 데이터베이스 관리 작업, 연결, 사용자 관리 및 권한 부여를 포함합니다.
- 데이터 조작 언어 읽기 이벤트(DML 읽기) - `find()` 및 다양한 집계 연산자, 산술 연산자, 부울 연산자 및 기타 읽기 쿼리 연산자를 포함합니다.
- 데이터 조작 언어 쓰기 이벤트(DML 쓰기) - `insert()`, `update()`, `delete()`, 및 `bulkWrite()` 연산자를 포함합니다

이벤트 유형은 다음과 같습니다.

이벤트 유형	범주	설명
`authCheck`	권한 부여	결과 코드 0: 성공
		결과 코드 13: 권한 없이 작업을 수행하려는 시도.
`authenticate`	Connection	새로운 연결에서 성공 또는 실패한 인증 시도.
`auditConfigure`	DDL	감사 필터 구성.
`createDatabase`	DDL	새 데이터베이스 생성.
`createCollection`	DDL	데이터베이스 내에서 새 모음 생성.
`createIndex`	DDL	모음 내에서 새 인덱스 생성.
`dropCollection`	DDL	데이터베이스에서 모음 삭제.

이벤트 유형	범주	설명
dropDatabase	DDL	데이터베이스 삭제.
dropIndex	DDL	모음에서 인덱스 삭제.
modifyChangeStreams	DDL	변경 스트림 생성.
renameCollection	DDL	데이터베이스 내에서 컬렉션 이름을 변경.
createRole	관리 역할	역할 생성
dropAllRolesFromDatabase	관리 역할	데이터베이스에서 모든 역할 삭제.
dropRole	관리 역할	역할 삭제.
grantPrivilegesToRole	관리 역할	역할에 권한 부여.
grantRolesToRole	관리 역할	사용자 정의 역할에 역할 부여.
revokePrivilegesFromRole	관리 역할	역할의 권한 취소.
revokeRolesFromRole	관리 역할	사용자 정의 역할에서 역할 취소.
updateRole	관리 역할	역할 업데이트.
createUser	사용자 관리	새 사용자 생성.
dropAllUsersFromDatabase	사용자 관리	데이터베이스에서 모든 사용자 삭제.
dropUser	사용자 관리	기존 사용자 삭제.
grantRolesToUser	사용자 관리	사용자에게 역할 부여.
revokeRolesFromUser	사용자 관리	사용자의 역할 취소.

이벤트 유형	범주	설명
updateUser	UserManagement	기존 사용자 업데이트.
insert	DML 쓰기	컬렉션에 1개 또는 여러 문서를 삽입합니다.
delete	DML 쓰기	컬렉션에서 1개 또는 여러 문서를 삭제합니다.
update	DML 쓰기	컬렉션에 있는 기존 문서 또는 여러 문서를 수정합니다.
bulkWrite	DML 쓰기	실행 순서를 제어하여 여러 쓰기 작업을 수행합니다.
setAuditConfig	DML 쓰기	DML 감사를 위한 새 필터를 설정합니다.
count	DML 읽기	컬렉션 또는 뷰의 find() 쿼리와 일치하는 문서 수를 반환합니다.
countDocuments	DML 읽기	컬렉션 또는 뷰의 쿼리와 일치하는 문서 수를 반환합니다.
find	DML 읽기	컬렉션 또는 뷰에서 문서를 선택하고 선택한 문서에 커서를 반환합니다.
getAuditConfig	DML 읽기	DML 감사를 위해 현재 필터를 검색합니다.

이벤트 유형	범주	설명
`findAndModify`	DML 읽기 및 DML 쓰기	1개의 문서를 수정 및 반환합니다.
`findOneAndDelete`	DML 읽기 및 DML 쓰기	필터 및 정렬 기준에 따라 단일 문서를 삭제하고 삭제된 문서를 반환합니다.
`findOneAndReplace`	DML 읽기 및 DML 쓰기	지정된 필터를 기반으로 단일 문서를 대체합니다.
`findOneAndUpdate`	DML 읽기 및 DML 쓰기	필터 및 정렬 기준에 따라 단일 문서를 업데이트합니다.
`aggregate`	DML 읽기 및 DML 쓰기	집계 파이프라인의 API를 지원합니다.
`distinct`	DML 읽기	단일 컬렉션 또는 뷰에서 지정된 필드의 고유 값을 찾아 결과를 배열로 반환합니다.

> **Note**
>
> DML 이벤트 문서 매개 변수 필드의 값에는 1KB 크기 제한이 있습니다. Amazon DocumentDB는 값이 1KB를 초과하는 경우 값을 잘라냅니다.

> **Note**
>
> TTL 삭제 이벤트는 현재 감사되지 않습니다.

감사 활성화

클러스터에서 감사를 활성화하는 절차는 두 단계로 이루어져 있습니다. 두 단계를 모두 완료하지 않으면 감사 로그가 CloudWatch Logs로 전송되지 않습니다.

1단계. audit_logs 클러스터 파라미터 활성화

감사를 활성화하려면 파라미터 그룹에서 `audit_logs` 파라미터를 수정해야 합니다. `audit_logs`는 로깅할 이벤트의 쉼표로 구분된 목록입니다. 이벤트는 모두 소문자로 지정해야 하며 목록 항목 사이에 공백이 없어야 합니다.

이 파라미터 그룹에 대해 다음 값을 지정할 수 있습니다.

값	설명	
`ddl`	이를 설정하면 createDatabase, dropDatabase, createCollection, dropCollection, createIndex, dropIndex, authCheck, authenticate, createUser, dropUser, grantRolesToUser, revokeRolesFromUser, updateUser, dropAllUsersFromDatabase 등과 같은 DDL 이벤트에 대한 감사가 활성화됩니다	
`dml_read`	이를 설정하면 찾기, 정렬 횟수, 고유, 그룹, 프로젝트, 언와인드, geoNear, geoIntersects, geoWithin 및 기타 MongoDB 읽기 쿼	

값	설명	
	리 연산자와 같은 DML 읽기 이벤트를 감사할 수 있습니다.	
dml_write	이를 설정하면 insert(), update(), delete() 및 bulkWrite()와 같은 DML 쓰기 이벤트에 대한 감사가 활성화됩니다.	
all	이를 설정하면 쿼리 읽기, 쿼리 쓰기, 데이터베이스 작업 및 관리자 작업과 같은 데이터베이스 이벤트를 감사할 수 있습니다.	
none	이렇게 설정하면 감사가 비활성화됩니다.	

값	설명	
enabled(레거시)	이는 'ddl'과 동일한 레거시 파라미터 설정입니다. 이를 설정하면 ccreateDatabase, dropDatabase, createCollection, dropCollection, createIndex, dropIndex, authCheck, authenticate, createUser, dropUser, grantRolesToUser, revokeRolesFromUser, updateUser, dropAllUsersFromDatabase 등과 같은 DDL 이벤트에 대한 감사가 활성화됩니다 이것은 레거시 설정이므로 이 설정을 사용하지 않는 것이 좋습니다.	
disabled (레거시)	이는 'none'과 동일한 레거시 파라미터 설정입니다. 이것은 레거시 설정이므로 이 설정을 사용하지 않는 것이 좋습니다.	

> **Note**
>
> audit_logs 클러스터 매개변수의 기본값은 none(레거시 'disabled')입니다.

위에서 언급한 값을 조합하여 사용할 수도 있습니다.

값	설명
`ddl, dml_read`	이를 설정하면 DDL 이벤트 및 DML 읽기 이벤트를 감사할 수 있습니다.
`ddl, dml_write`	이를 설정하면 DDL 이벤트 및 DML 읽기를 감사할 수 있음
`dml_read, dml_write`	이를 설정하면 모든 DDL 이벤트를 감사할 수 있음

> **Note**
>
> 기본 파라미터 그룹은 수정할 수 없습니다.

자세한 내용은 다음을 참조하세요.

- Amazon DocumentDB 클러스터 파라미터 그룹 생성

  사용자 지정 파라미터 그룹을 생성한 후 `audit_logs` 파라미터 값을 `all`로 변경하여 파라미터 그룹을 수정합니다.
- Amazon DocumentDB 클러스터 파라미터 그룹 수정

2단계. Amazon CloudWatch Logs 내보내기 활성화

`audit_logs` 클러스터 파라미터의 값을 `enabled`, `ddl`, `dml_read`, `dml_write`로 설정한 경우 이와 동시에 Amazon DocumentDB가 Amazon CloudWatch로 로그를 내보내도록 활성화해야 합니다. 이들 단계를 하나라도 생략하면 감사 로그가 CloudWatch로 전송되지 않습니다.

클러스터를 생성하거나, 특정 시점 복원을 수행하거나, 스냅샷을 복원할 때 다음 단계에 따라 CloudWatch Logs를 활성화할 수 있습니다.

Using the AWS Management Console

콘솔을 사용하여 CloudWatch로 로그를 내보낼 수 있도록 Amazon DocumentDB를 활성화하려면 다음 주제를 참조하세요.

- 클러스터를 생성하는 경우, 를 사용하여 클러스터 및 기본 인스턴스 생성 AWS Management Console에서 클러스터 생성: 추가 구성(5단계, 로그 내보내기)을 참조하세요.
- 기존 클러스터를 수정하는 경우 — Amazon DocumentDB 클러스터 수정
- 클러스터 스냅샷 복원을 수행하는 경우 — 클러스터 스냅샷에서 복원
- 특정 시점으로 복구를 수행하는 경우 — 특정 시점으로 복원

Using the AWS CLI

새 클러스터를 생성할 때 감사 로그를 활성화하려면

다음 코드는 클러스터 `sample-cluster`을 생성하고 CloudWatch 감사 로그를 활성화합니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb create-db-cluster \
    --db-cluster-identifier sample-cluster \
    --port 27017 \
    --engine docdb \
    --master-username master-username \
    --master-user-password password \
    --db-subnet-group-name default \
    --enable-cloudwatch-logs-exports audit
```

Windows의 경우:

```
aws docdb create-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --port 27017 ^
    --engine docdb ^
    --master-username master-username ^
    --master-user-password password ^
```

```
    --db-subnet-group-name default ^
    --enable-cloudwatch-logs-exports audit
```

기존 클러스터를 수정할 때 감사 로그를 활성화하려면

다음 코드는 클러스터 sample-cluster을 수정하고 CloudWatch 감사 로그를 활성화합니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb modify-db-cluster \
   --db-cluster-identifier sample-cluster \
   --cloudwatch-logs-export-configuration '{"EnableLogTypes":["audit"]}'
```

Windows의 경우:

```
aws docdb modify-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --cloudwatch-logs-export-configuration '{"EnableLogTypes":["audit"]}'
```

이러한 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "DBCluster": {
        "HostedZoneId": "ZNKXH85TT8WVW",
        "StorageEncrypted": false,
        "DBClusterParameterGroup": "default.docdb4.0",
        "MasterUsername": "<user-name>",
        "BackupRetentionPeriod": 1,
        "Port": 27017,
        "VpcSecurityGroups": [
            {
                "Status": "active",
                "VpcSecurityGroupId": "sg-77186e0d"
            }
        ],
        "DBClusterArn": "arn:aws:rds:us-east-1:900083794985:cluster:sample-cluster",
        "Status": "creating",
        "Engine": "docdb",
        "EngineVersion": "4.0.0",
        "MultiAZ": false,
```

```
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1c",
            "us-east-1f"
        ],
        "DBSubnetGroup": "default",
        "DBClusterMembers": [],
        "ReaderEndpoint": "sample-cluster.cluster-ro-corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
        "EnabledCloudwatchLogsExports": [
            "audit"
        ],
        "PreferredMaintenanceWindow": "wed:03:08-wed:03:38",
        "AssociatedRoles": [],
        "ClusterCreateTime": "2019-02-13T16:35:04.756Z",
        "DbClusterResourceId": "cluster-YOS52CUXGDTNKDQ7DH72I4LED4",
        "Endpoint": "sample-cluster.cluster-corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
        "PreferredBackupWindow": "07:16-07:46",
        "DBClusterIdentifier": "sample-cluster"
    }
}
```

감사 비활성화

CloudWatch Logs 내보내기를 비활성화하고 `audit_logs` 파라미터를 비활성화하여 감사를 비활성화할 수 있습니다.

CloudWatch Logs 내보내기 비활성화

AWS Management Console 또는를 사용하여 감사 로그 내보내기를 비활성화할 수 있습니다 AWS CLI.

Using the AWS Management Console

다음 절차에서는를 사용하여 로그 AWS Management Console 를 CloudWatch로 내보내는 Amazon DocumentDB를 비활성화합니다.

감사 로그를 비활성화하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.

2. 탐색 창에서 클러스터를 선택합니다. 그런 다음 로그 내보내기를 비활성화할 클러스터의 이름 왼쪽에 있는 버튼을 선택합니다.
3. 작업을 선택한 후 수정을 선택합니다.
4. 로그 내보내기 섹션으로 아래로 스크롤하여 비활성을 선택합니다.
5. 계속을 선택합니다.
6. 변경 사항을 검토하고 이 변경 사항을 클러스터에 적용할 시기를 선택합니다.
   - 예약된 다음 유지 관리 기간에 적용
   - 즉시 적용
7. 클러스터 수정을 선택합니다.

Using the AWS CLI

다음 코드는 클러스터 `sample-cluster`을 수정하고 CloudWatch 감사 로그를 비활성화합니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb modify-db-cluster \
   --db-cluster-identifier sample-cluster \
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["audit"]}'
```

Windows의 경우:

```
aws docdb modify-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["audit"]}'
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "DBCluster": {
        "DBClusterParameterGroup": "default.docdb4.0",
        "HostedZoneId": "ZNKXH85TT8WVW",
        "MasterUsername": "<user-name>",
        "Status": "available",
        "Engine": "docdb",
```

```
        "Port": 27017,
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1c",
            "us-east-1f"
        ],
        "EarliestRestorableTime": "2019-02-13T16:35:50.387Z",
        "DBSubnetGroup": "default",
        "LatestRestorableTime": "2019-02-13T16:35:50.387Z",
        "DBClusterArn": "arn:aws:rds:us-east-1:900083794985:cluster:sample-
cluster2",
        "Endpoint": "sample-cluster2.cluster-corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
        "ReaderEndpoint": "sample-cluster2.cluster-ro-corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
        "BackupRetentionPeriod": 1,
        "EngineVersion": "4.0.0",
        "MultiAZ": false,
        "ClusterCreateTime": "2019-02-13T16:35:04.756Z",
        "DBClusterIdentifier": "sample-cluster2",
        "AssociatedRoles": [],
        "PreferredBackupWindow": "07:16-07:46",
        "DbClusterResourceId": "cluster-YOS52CUXGDTNKDQ7DH72I4LED4",
        "StorageEncrypted": false,
        "PreferredMaintenanceWindow": "wed:03:08-wed:03:38",
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "Status": "active",
                "VpcSecurityGroupId": "sg-77186e0d"
            }
        ]
    }
}
```

audit_logs 파라미터 비활성화

클러스터에 대한 `audit_logs` 파라미터를 비활성화하려면 `audit_logs` 파라미터 값이 `disabled`인 파라미터 그룹을 사용하도록 클러스터를 수정하면 됩니다. 또는 클러스터의 파라미터 그룹에서 `audit_logs` 파라미터 값을 `disabled`로 수정하면 됩니다.

자세한 내용은 다음 항목을 참조하세요.

- Amazon DocumentDB 클러스터 수정
- Amazon DocumentDB 클러스터 파라미터 그룹 수정

감사 이벤트 액세스

다음 단계에 따라 Amazon CloudWatch에서 감사 이벤트에 액세스합니다.

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 사용자가 Amazon DocumentDB 클러스터와 동일한 리전에 있어야 합니다.
3. 탐색 창에서 로그를 선택합니다.
4. 클러스터에 대한 감사 로그를 찾으려면 목록에서 **/aws/docdb/*yourClusterName*/audit**을 찾아 선택합니다.

   각 인스턴스에 대한 감사 이벤트는 해당 인스턴스 이름 아래에서 찾을 수 있습니다.

DML 감사 이벤트 필터링

DML 감사 필터링 시작하기

DML 감사 이벤트는 Amazon CloudWatch에 기록되기 전에 필터링할 수 있습니다. 이 기능을 사용하려면 감사 로그 및 DML 로깅을 활성화해야 합니다. Amazon DocumentDB는 `atype`, `command`, `user`, `namespace` 및 `auditAuthorizationSuccess`에서 필터링을 지원합니다.

> **Note**
>
> DDL 이벤트는 필터링되지 않습니다.

`db.adminCommand( { command } )` 작업에서 `setAuditConfig`, `filter`및 `auditAuthorizationSuccess` 파라미터를 사용해 감사 필터를 지정하여 언제든지 감사 필터링을 활성화할 수 있습니다.

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter:
         {
            //filter conditions
```

```
        },
      auditAuthorizationSuccess: true | false
   }
)
```

다음 명령을 실행하여 감사 필터 설정을 검색할 수도 있습니다.

```
db.admin.runCommand( { getAuditConfig: 1})
```

보안 요구 사항

권한이 있는 auditConfigure 작업이 있는 데이터베이스 사용자/역할만 DML 감사 필터를 설정하거나 나열할 때 admindb에 대한 명령을 실행할 수 있습니다. [clusterAdmin, hostManager, root]에서 기본 제공 역할 중 하나를 사용하거나 auditConfigure 권한이 있는 사용자 지정 역할을 생성할 수 있습니다. 다음은 auditConfigure 권한과 함께 기존 역할을 사용하는 예제와 사용자 지정 역할을 사용하는 예제입니다.

역할이 내장된 사용자:

```
use admin
db.createUser(
  {
    user: "myClusterAdmin",
    pwd: "password123",
    roles: [ { role: "clusterAdmin", db: "admin" } ]
  }
)
```

사용자 지정 역할이 있는 사용자:

```
use admin
db.createRole(
   {
     role: "myRole",
     privileges: [
       { resource: { cluster: true }, actions: [ "auditConfigure" ] }
     ],
     roles: []
   }
)
db.createUser(
  {
```

```
    user: "myUser",
    pwd: "myPassword",
    roles: [ { role: "myRole", db: "admin" } ]
  }
)
```

사용 사례 필터링

예: 명령을 기준으로 이벤트 필터링

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
        "$and": [
         {
            "param.command":
               {
                  $in: [ "find","count", "insert", "delete", "update",
 "findandmodify" ]
               }
         }
         ]
      },
      auditAuthorizationSuccess: true
   }
)
```

예: 사용자 이름으로 이벤트 필터링

이 예제에서는 사용자 "myUser"만 로깅됩니다.

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
      "$and": [
         {
            "param.user":
               {
                  $in: [ "myUser" ]
               }
         }
```

```
        ]},
      auditAuthorizationSuccess: true})
```

예제: **atype** 항목을 사용해 필터링

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {atype: "authCheck"},
      auditAuthorizationSuccess: true
   })
```

> **Note**
>
> 모든 DML 로그의 authCheck 항목은 atype입니다. DDL의 경우만 atype 항목이 다릅니다. filter에서 authCheck 이외의 값을 입력하면 CloudWatch 에서 DML 로그가 생성되지 않습니다.

예: 연산자가 조인한 여러 필터를 사용하여 필터링

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
      "$and": [
         {
            "param.command":
               {
                  $in: [ "find","count", "insert", "delete", "update",
 "findandmodify" ]
               }
         }
         ],
       "$nor": [
         {
            "param.command":
               {
                  $in: ["count", "insert", "delete", "update", "findandmodify" ]
               }
         }]
       },
```

```
      auditAuthorizationSuccess: true})
```

> **Note**
>
> 최상위 수준에서는 $and, $or, 및 $nor 항목만 지원됩니다. 다른 연산자는 지원되지 않으며 오류가 발생합니다.

예: **auditAuthorizationSuccess**의 이벤트를 기준으로 필터링

이 필터에서는 권한 부여를 성공적으로 통과한 모든 명령이 로그되지 않습니다.

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {},
      auditAuthorizationSuccess: false
   }
)
```

예: **$in** 및 **$nin** 조건을 사용한 필터링

$in 및 $nin에서 둘 다 사용하는 경우 조건 간에 암시적인 'and'가 존재하므로 명령이 로그되지 않습니다. 이 예제에서는 정규식이 find 명령을 차단하므로 아무 것도 로그되지 않습니다.

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
      "$and": [
         {
            atype: "authCheck",
            "param.command":
               {
                  $in: [ "find", "insert", "delete", "update", "findandmodify" ],
                  $nin: ["count", "insert", "delete", "update", "findandmodify" ],
                  $not: /^find.*/
               }
         },
         ],
       "$or": [
         {
```

```
            "param.command":
               {
                  $nin: ["count", "insert", "delete", "update", "findandmodify" ]
               }
         }]
       },
      auditAuthorizationSuccess: true})
```

예제: **namespace** 항목을 사용해 필터링

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
      "$and": [
         {
            "param.ns":
               {
                  $in: [ "test.foo" ]
               }
         }
         ]},
      auditAuthorizationSuccess: true})
```

예제: 기본 필터로 재설정

기본값으로 재설정하면 모든 DML 감사 이벤트가 로그됩니다. 필터링을 기본값으로 재설정하려면 다음 명령을 실행합니다.

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {},
      auditAuthorizationSuccess: true
   }
)
```

Amazon VPC 및 Amazon DocumentDB

Amazon Virtual Private Cloud(VPC)를 사용하면 Amazon DocumentDB 인스턴스와 같은 AWS 리소스를 Virtual Private Cloud(VPC)로 시작할 수 있습니다.

VPC를 사용하면 가상 네트워킹 환경을 완벽하게 제어할 수 있습니다. 자기만의 IP 주소 범위를 선택하고, 서브넷을 생성하고, 라우팅 및 액세스 제어 목록을 구성할 수 있습니다. VPC에서 클러스터를 실행하는 데는 추가 비용이 들지 않습니다.

계정에는 기본 VPC가 있습니다. 달리 지정하지 않는 한 새 클러스터는 모두 기본 VPC에서 생성됩니다.

주제

- VPC에서의 DocumentDB 클러스터
- VPC에서 Amazon DocumentDB 클러스터에 액세스
- DocumentDB 클러스터와 함께 사용할 IPv4 전용 VPC 생성
- DocumentDB 클러스터와 함께 사용할 듀얼 스택 VPC 생성

아래에서 Amazon DocumentDB 클러스터와 관련된 VPC 기능에 대한 토론을 찾을 수 있습니다. Amazon VPC에 대한 자세한 내용은 Amazon VPC 사용 설명서를 참조하세요.

VPC에서의 DocumentDB 클러스터

Amazon DocumentDB 클러스터는 가상 사설 클라우드(VPC) 안에 있습니다. VPC는의 다른 가상 네트워크와 논리적으로 격리된 가상 네트워크입니다 AWS 클라우드. Amazon VPC를 사용하면 Amazon DocumentDB 클러스터 또는 Amazon EC2 인스턴스와 같은 AWS 리소스를 VPC로 시작할 수 있습니다. VPC는 계정과 함께 제공되는 기본 VPC일 수도 있고, 사용자가 만들 수도 있습니다. 모든 VPCs AWS 계정과 연결됩니다.

기본 VPC에는 VPC 내의 리소스를 격리하는 데 사용할 수 있는 3개의 서브넷이 있습니다. 또한 VPC 외부에서 VPC 내부의 리소스에 액세스할 수 있도록 하는 데 사용할 수 있는 인터넷 게이트웨이도 있습니다.

VPC 내부 및 VPC 외부의 Amazon DocumentDB 클러스터와 관련된 시나리오는 VPC에서 Amazon DocumentDB 클러스터에 액세스 섹션을 참조하세요.

주제

- VPC에서 클러스터 사용
- 서브넷 그룹을 사용한 작업
- Amazon DocumentDB IP 주소 지정
- VPC에서 클러스터 생성

VPC에서 클러스터 사용

다음은 VPC에서 클러스터를 사용할 때 유용한 몇 가지 팁입니다.

- VPC는 최소 2개 이상의 서브넷을 보유해야 합니다. 이러한 서브넷은 클러스터를 배포 AWS 리전 하려는의 서로 다른 두 가용 영역에 있어야 합니다. 서브넷은 사용자가 지정할 수 있고 보안 및 운영상의 필요를 바탕으로 클러스터를 그룹화할 수 있게 해주는 VPC IP 주소 범위의 한 부분입니다.
- VPC 내의 클러스터를 공개적으로 액세스할 수 있게 하려면 VPC 속성 `DNS hostnames` 및 `DNS resolution`을 활성화해야 합니다.
- VPC에는 직접 만든 서브넷 그룹이 있어야 합니다. 생성한 서브넷을 지정하여 DB 서브넷 그룹을 생성합니다. AmazonDocumentDB 는 클러스터의 기본 인스턴스와 연결할 서브넷과 해당 서브넷 그룹 내의 IP 주소를 선택합니다. 기본 인스턴스는 서브넷이 포함된 가용 영역을 사용합니다.
- 해당 VPC에는 클러스터에 대한 액세스를 허용하는 VPC 보안 그룹이 있어야 합니다.

  자세한 내용은 VPC에서 Amazon DocumentDB 클러스터에 액세스 섹션을 참조하세요.
- 각 서브넷의 CIDR 블록은 장애 조치와 컴퓨팅 조정을 포함한 유지 관리 활동 중에 Amazon DocumentDB가 사용할 예비 IP 주소를 수용할 만큼 충분히 커야 합니다. 예를 들어, 일반적으로 10.0.0.0/24 및 10.0.1.0/24와 같은 범위는 충분히 큽니다.
- VPC는 `default` 또는 `dedicated` 중 하나의 `instance tenancy` 속성을 가질 수 있습니다. 모든 기본 VPC에는 `instance tenancy` 속성이 기본값으로 설정되어 있으며, 기본 VPC는 어떠한 인스턴스 클래스라도 지원할 수 있습니다.

  `instance tenancy` 속성이 `dedicated`로 설정된 전용 VPC에 클러스터를 사용하도록 선택한 경우 클러스터의 인스턴스 클래스는 승인된 Amazon EC2 전용 인스턴스 유형 중 하나여야 합니다. 예를 들어 `r5.large` EC2 전용 인스턴스는 `db.r5.large` 인스턴스 클래스에 해당합니다. VPC의 인스턴스 테넌시에 대한 자세한 내용은 Amazon Elastic Compute Cloud 사용 설명서의 Amazon EC2 인스턴스를 참조하세요.

  전용 인스턴스에 존재할 수 있는 인스턴스 유형에 대한 자세한 내용은 Amazon EC2 요금 페이지에서 Amazon EC2 전용 인스턴스를 참조하세요.

> **Note**
>
> `instance tenancy` 속성을 클러스터에 대해 `dedicated`로 설정하면 클러스터가 전용 호스트에서 실행된다는 보장이 없습니다.

서브넷 그룹을 사용한 작업

서브넷은 사용자가 보안 및 운영상의 필요를 바탕으로 리소스를 그룹화하기 위해 지정하는 VPC IP 주소 범위의 특정 부분입니다. 서브넷 그룹은 사용자가 VPC에서 만든 다음 클러스터에 대해 지정하는 서브넷(일반적으로 프라이빗)의 모음입니다. 서브넷 그룹을 사용하면 AWS CLI 또는 Amazon DocumentDB API를 사용하여 클러스터를 생성할 때 특정 VPC를 지정할 수 있습니다. 콘솔을 사용하는 경우 사용할 VPC와 서브넷 그룹을 선택할 수 있습니다.

각 서브넷 그룹은 지정된 AWS 리전에서 두 개 이상의 가용 영역에 서브넷이 있어야 합니다. VPC에서 클러스터를 생성할 때 이에 대한 서브넷 그룹을 선택합니다. Amazon DocumentDB는 클러스터의 기본 인스턴스와 연결할 서브넷과 해당 서브넷 내의 IP 주소를 서브넷 그룹 내에서 선택합니다. 데이터베이스는 서브넷이 포함된 가용 영역을 사용합니다. DocumentDB는 항상 사용 가능한 IP 주소 공간이 있는 서브넷에서 IP 주소를 할당합니다.

서브넷 그룹의 서브넷은 네트워크 액세스 제어 목록(ACL) 및 라우팅 테이블에 대해 설정한 구성에 따라 퍼블릭 또는 프라이빗입니다. 글러스터에 공개적으로 액세스할 수 있도록 하려면 서브넷 그룹의 모든 서브넷이 퍼블릭이어야 합니다. 공개적으로 액세스할 수 있는 클러스터와 연결된 서브넷이 퍼블릭에서 프라이빗으로 변경되면 클러스터 가용성에 영향을 줄 수 있습니다.

듀얼 스택 모드를 지원하는 서브넷 그룹을 생성하려면 서브넷 그룹에 추가하는 각 서브넷에 인터넷 프로토콜 버전 6(IPv6) CIDR 블록이 연결되어 있는지 확인합니다. 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 Amazon DocumentDB IP 주소 지정 및 VPC에 IPv6 지원을 참조하세요.

Amazon DocumentDB가 VPC에 클러스터를 생성할 때는 DB 서브넷 그룹의 IP 주소를 사용하여 클러스터에 네트워크 인터페이스를 할당합니다. 기본 IP 주소가 장애 조치 중에 변경되므로 항상 도메인 이름 시스템(DNS) 이름을 사용하여 클러스터에 연결합니다.

> **Note**
>
> VPC에서 실행하는 각각의 클러스터에 대해 Amazon DocumentDB가 복구 작업에 사용할 수 있도록 DB 서브넷 그룹의 각 서브넷에 한 개 이상의 주소를 예약해야 합니다.

공유 서브넷

공유 VPC에서 클러스터를 생성할 수 있습니다.

공유 VPC를 사용할 때 염두에 두어야 할 몇 가지 고려 사항은 다음과 같습니다.

- 클러스터를 공유 VPC 서브넷에서 비공유 VPC 서브넷으로 또는 그 반대로 이동할 수 있습니다.

- 공유 VPC의 참여자는 VPC에 보안 그룹을 생성해야 클러스터를 생성할 수 있습니다.
- 공유 VPC의 소유자와 참여자는 DocumentDB 쿼리를 사용하여 데이터베이스에 액세스할 수 있습니다. 그러나 리소스 생성자만 리소스에 대한 API 호출을 수행할 수 있습니다.

Amazon DocumentDB IP 주소 지정

IP 주소가 있으면 VPC 내의 리소스가 서로 통신하고, 인터넷을 통해 다른 리소스와 통신할 수 있습니다. Amazon DocumentDB는 IPv4와 IPv6 주소 지정 프로토콜을 모두 지원합니다. 기본적으로 Amazon DocumentDB와 Amazon VPC는 IPv4 주소 지정 프로토콜을 사용합니다. 이 동작은 끌 수 없습니다. VPC를 생성할 때 VPC에 IPv4 CIDR 블록(프라이빗 IPv4 주소)를 지정해야 합니다. IPv6 CIDR 블록을 VPC와 서브넷에 할당하고 그 블록에 속한 IPv6 주소를 서브넷의 클러스터에 할당할 수도 있습니다.

> **Note**
>
> 듀얼 스택 모드(IPv6 주소 지정)는 Amazon DocumentDB 버전 4.0 및 5.0에서만 지원됩니다.

IPv6 프로토콜을 지원하면 지원되는 IP 주소 수가 늘어납니다. IPv6 프로토콜을 사용하면 향후에 인터넷이 성장할 때를 대비해 사용 가능한 주소를 충분히 확보할 수 있습니다. 신규 및 기존 DocumentDB 리소스는 VPC 내에서 IPv4 및 IPv6 주소를 사용할 수 있습니다. 애플리케이션의 다른 부분에서 사용되는 두 프로토콜 간의 네트워크 트래픽을 구성, 보안 및 변환하면 운영 오버헤드가 발생할 수 있습니다. Amazon DocumentDB 리소스의 IPv6 프로토콜을 표준화하여 네트워크 구성을 간소화할 수 있습니다.

주제

- IPv4 주소
- IPv6 주소
- 듀얼 스택 모드

IPv4 주소

VPC를 생성할 때 VPC의 IPv4 주소 범위를 `10.0.0.0/16`와 같은 CIDR 블록 형태로 지정해야 합니다. 서브넷 그룹은 이 CIDR 블록에서 클러스터가 사용할 수 있는 IP 주소의 범위를 정의합니다. 이 IP 주소는 프라이빗 또는 퍼블릭일 수 있습니다.

프라이빗 IPv4 주소는 인터넷을 통해 연결할 수 없는 IP 주소입니다. 프라이빗 IPv4 주소는 클러스터 및 같은 VPC 내의 Amazon EC2 인스턴스와 같은 기타 리소스 간의 통신을 위해 사용될 수 있습니다. 각 클러스터에는 VPC 내 통신을 위한 프라이빗 IP 주소가 있습니다.

퍼블릭 IP 주소는 인터넷을 통해 연결할 수 있는 IPv4 주소입니다. DocumentDB 클러스터에는 퍼블릭 IP 주소 지정이 허용되지 않습니다. 모든 퍼블릭 IP 주소는 인터넷 게이트웨이와 퍼블릭 서브넷의 EC2에 의해 확인되어야 합니다.

일반적인 Amazon DocumentDB 시나리오에서 사용할 수 있는 프라이빗 IPv4 주소만을 사용해 VPC를 생성하는 방법은 DocumentDB 클러스터와 함께 사용할 IPv4 전용 VPC 생성 섹션을 참조하세요.

IPv6 주소

IPv6 CIDR 블록을 VPC와 서브넷에 연결하고 그 블록에 속한 IPv6 주소를 VPC의 리소스에 할당할 수도 있습니다. 각 IPv6 주소는 전역적으로 고유합니다.

VPC에 대한 IPv6 CIDR 블록은 Amazon의 IPv6 주소 풀에서 자동으로 할당되므로 범위를 직접 선택할 수는 없습니다.

IPv6 주소에 연결할 때 다음 조건이 충족되는지 확인하세요.

- IPv6를 통한 클라이언트에서 데이터베이스로의 트래픽이 허용되도록 클라이언트가 구성됩니다.
- 클러스터에서 사용하는 DocumentDB 보안 그룹은 IPv6를 통한 클라이언트에서 데이터베이스로의 트래픽이 허용되도록 올바르게 구성됩니다.
- 클라이언트 운영 체제 스택은 IPv6 주소의 트래픽을 허용하며 운영 체제 드라이버 및 라이브러리는 올바른 기본 클러스터 엔드포인트(IPv4 또는 IPv6)를 선택하도록 구성됩니다.

IPv6에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 IP 주소 지정을 참조하세요.

듀얼 스택 모드

클러스터가 IPv4 및 IPv6 주소 지정 프로토콜 모두를 통해 통신할 수 있을 때 듀얼 스택 모드로 실행됩니다. 따라서 리소스는 IPv4, IPv6 또는 모두를 통해 클러스터와 통신할 수 있습니다. DocumentDB는 프라이빗 듀얼 스택 모드 클러스터의 IPv6 엔드포인트에 대한 인터넷 게이트웨이 액세스를 비활성화합니다. DocumentDB는 IPv6 엔드포인트가 프라이빗이며 VPC 내에서만 액세스할 수 있도록 하기 위해 이 작업을 수행합니다.

주제

- 듀얼 스택 모드 및 서브넷 그룹
- 듀얼 스택 모드 클러스터 사용
- 듀얼 스택 모드를 사용하도록 IPv4 전용 클러스터 수정
- 듀얼 스택 모드 리전 및 버전 가용 여부
- 듀얼 스택 네트워크 클러스터에 대한 제한 사항

듀얼 스택 모드 및 서브넷 그룹

듀얼 스택 모드를 사용하려면 클러스터와 연결하는 서브넷 그룹의 각 서브넷에 IPv6 CIDR 블록이 연결되어 있어야 합니다. 새 서브넷 그룹을 생성하거나 기존 서브넷 그룹을 수정하여 이 요구 사항을 충족하도록 수정할 수 있습니다. 클러스터가 듀얼 스택 모드가 된 후에는 클라이언트가 정상적으로 연결할 수 있습니다. 클라이언트 보안 방화벽과 DocumentDB 클러스터 보안 그룹이 IPv6을 통한 트래픽을 허용하도록 정확하게 구성되어 있는지 확인합니다. 연결을 위해 클라이언트는 클러스터의 엔드포인트를 사용합니다. 클라이언트 애플리케이션은 데이터베이스에 연결할 때 기본 프로토콜을 지정할 수 있습니다. 듀얼 스택 모드에서 클러스터는 클라이언트의 기본 네트워크 프로토콜(IPv4 또는 IPv6)을 감지하고 연결에 해당 프로토콜을 사용합니다.

서브넷 삭제 또는 CIDR 연결 해제로 인해 서브넷 그룹이 듀얼 스택 모드 지원을 중지하는 경우 서브넷 그룹과 연결된 클러스터에 대해 네트워크 상태가 호환되지 않을 위험이 있습니다. 또한 새 듀얼 스택 모드 클러스터를 생성할 때 서브넷 그룹을 사용할 수 없습니다.

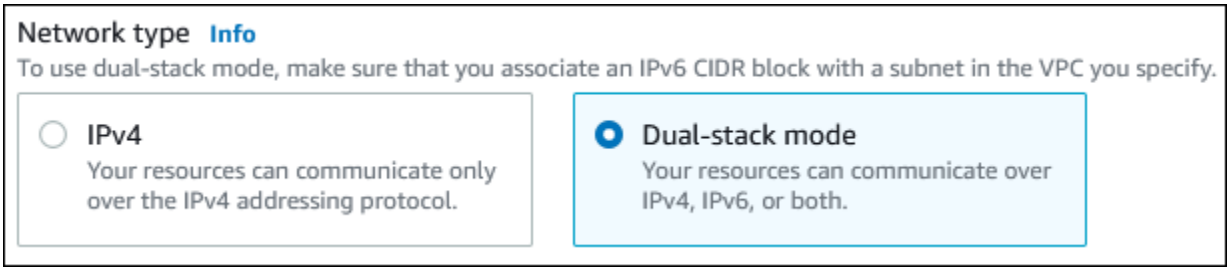
를 사용하여 서브넷 그룹이 듀얼 스택 모드를 지원하는지 확인하려면 서브넷 그룹의 세부 정보 페이지에서 네트워크 유형을 AWS Management Console확인합니다. 를 사용하여 서브넷 그룹이 듀얼 스택 모드를 지원하는지 확인하려면 `describe-db-subnet-groups` 명령을 AWS CLI실행하고 출력`SupportedNetworkTypes`에서를 확인합니다.

읽기 전용 복제본은 독립적인 클러스터로 취급되며 기본 클러스터와 다른 네트워크 유형을 가질 수 있습니다. 읽기 전용 복제본의 기본 클러스터의 네트워크 유형을 변경하는 경우 읽기 전용 복제본은 영향을 받지 않습니다. 클러스터를 복원할 때 지원되는 모든 네트워크 유형으로 복원할 수 있습니다.

듀얼 스택 모드 클러스터 사용

클러스터를 생성하거나 수정할 때 리소스가 클러스터와 IPv4, IPv6 또는 둘 다를 통해 통신할 수 있도록 허용하기 위해 듀얼 스택 모드를 지정할 수 있습니다.

AWS Management Console 를 사용하여 클러스터를 생성하거나 수정할 때 네트워크 유형 섹션에서 듀얼 스택 모드를 지정할 수 있습니다. 다음 이미지는 콘솔의 네트워크 유형 섹션을 보여줍니다.



AWS CLI 를 사용하여 클러스터를 생성하거나 수정할 때 듀얼 스택 모드를 사용하려면 `--network-type` 옵션을 `DUAL`로 설정합니다. DocumentDB API를 사용하여 클러스터를 생성하거나 수정할 때 듀

얼 스택 모드를 사용하려면 NetworkType 파라미터를 DUAL로 설정합니다. 지정된 DocumentDB 엔진 버전 또는 서브넷 그룹에서 듀얼 스택 모드가 지원되지 않는 경우 NetworkTypeNotSupported 오류가 반환됩니다.

클러스터 생성에 대한 자세한 내용은 Amazon DocumentDB 클러스터 생성 섹션을 참조하세요. 클러스터 수정에 대한 자세한 내용은 Amazon DocumentDB 클러스터 수정 섹션을 참조하세요.

콘솔을 사용하여 클러스터가 이중 스택 모드인지 확인하려면 해당 클러스터의 연결 및 보안 탭에서 네트워크 유형을 확인하세요.

듀얼 스택 모드를 사용하도록 IPv4 전용 클러스터 수정

듀얼 스택 모드를 사용하도록 IPv4 전용 클러스터를 수정할 수 있습니다. 그러려면 클러스터의 네트워크 유형을 변경합니다.

유지 관리 기간 동안 Amazon DocumentDB 클러스터의 네트워크 유형을 변경하는 것이 좋습니다. modify-db-cluster 명령을 사용하여 네트워크 유형을 수동으로 설정할 수 있습니다.

듀얼 스택 모드를 사용하도록 클러스터를 수정하기 전에 서브넷 그룹이 듀얼 스택 모드를 지원하는지 확인하세요. 클러스터와 연결된 서브넷 그룹이 듀얼 스택 모드를 지원하지 않는 경우 클러스터를 수정할 때 이를 지원하는 다른 서브넷 그룹을 지정하세요. 클러스터의 서브넷 그룹을 수정하면 가동 중지가 발생할 수 있습니다.

듀얼 스택 모드를 사용하도록 클러스터를 변경하기 전에 클러스터의 서브넷 그룹을 수정할 경우, 서브넷 그룹이 변경 전후에 클러스터에 대해 유효한지 확인하세요.

NetworkType 파라미터를 로 설정한 상태에서 ModifyDBCluster 호출을 실행DUAL하여 네트워크를 듀얼 스택 모드로 변경합니다. 동일한 API 직접 호출에서 다른 파라미터를 NetworkType과 함께 추가하면 가동 중지가 발생할 수 있습니다. 여러 파라미터를 수정하려면 다른 파라미터를 사용하여 다른 ModifyDBCluster 요청을 보내기 전에 네트워크 유형 수정을 성공적으로 완료했는지 확인합니다.

변경 후 클러스터에 연결할 수 없는 경우 선택한 네트워크(IPv4 또는 IPv6)의 데이터베이스로의 트래픽을 허용하도록 클라이언트 및 데이터베이스 보안 방화벽과 라우팅 테이블이 정확하게 구성되어 있는지 확인하세요. IPv6 주소를 사용하여 연결하려면 운영 체제 파라미터, 라이브러리 또는 드라이버를 수정해야 할 수도 있습니다.

듀얼 스택 모드를 사용하도록 IPv4 전용 클러스터 수정

1. 듀얼 스택 모드를 지원하도록 서브넷 그룹을 수정하거나 듀얼 스택 모드를 지원하는 서브넷 그룹을 생성합니다.

a. IPv6 CIDR 블록을 VPC와 연결

   자세한 내용은 Amazon VPC 사용 설명서의 VPC에 CIDR 블록 추가 또는 제거를 참조하세요.

b. IPv6 CIDR 블록을 서브넷 그룹의 모든 서브넷에 연결합니다.

   자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 서브넷에 IPv6 CIDR 블록 추가 또는 제거를 참조하세요.

c. 서브넷 그룹이 듀얼 스택 모드를 지원하는지 확인합니다.

   를 사용하는 경우 서브넷 그룹을 AWS Management Console선택하고 지원되는 네트워크 유형 값이 이중인지 확인합니다.

   를 사용하는 경우 `describe-db-subnet-groups` 명령을 AWS CLI실행하고 클러스터 `SupportedNetworkType` 값이 인지 확인합니다`Dual`.

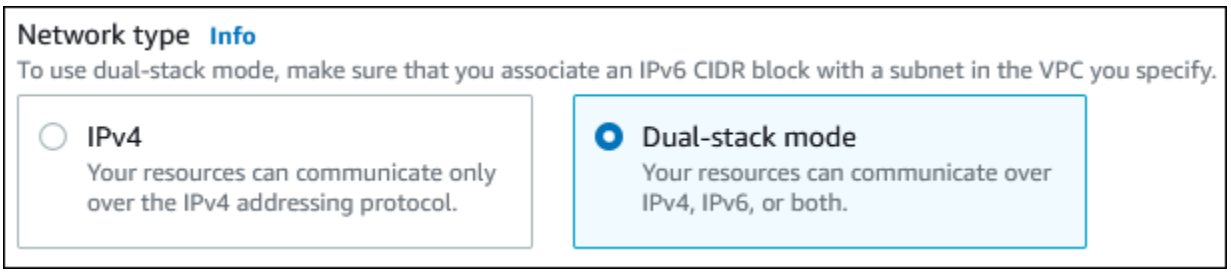
2. 데이터베이스에 대한 IPv6 연결을 허용하도록 클러스터와 연결된 보안 그룹을 수정하거나 IPv6 연결을 허용하는 새 보안 그룹을 생성합니다.

   자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 보안 그룹 규칙을 참조하세요.

3. 듀얼 스택 모드를 지원하도록 클러스터를 수정합니다. 이렇게 하려면 네트워크 유형을 듀얼 스택 모드로 설정하세요.

   콘솔을 사용 중인 경우 다음 설정이 올바른지 확인합니다.

   - 네트워크 유형 - 듀얼 스택 모드



   - 서브넷 그룹 - 이전 단계에서 구성한 서브넷 그룹
   - 보안 그룹 - 이전 단계에서 구성한 보안

   를 사용하는 경우 다음 설정이 올바른지 AWS CLI확인합니다.

   - `--network-type` — `dual`
   - `--db-subnet-group-name` - 이전 단계에서 구성한 서브넷 그룹

- `--vpc-security-group-ids` - 이전 단계에서 구성한 VPC 보안 그룹

예제:

```
aws docdb modify-db-cluster --db-cluster-identifier <cluster-name> --network-type
 "DUAL"
```

4. 클러스터가 듀얼 스택 모드를 지원하는지 확인합니다.

   콘솔을 사용 중인 경우 클러스터의 연결 및 보안 탭을 선택하고 네트워크 유형 값이 듀얼 스택 모드인지 확인합니다.

   를 사용하는 경우 `describe-db-cluster` 명령을 AWS CLI실행하고 클러스터 `NetworkType` 값이 인지 확인합니다`dual`.

   클러스터 엔드포인트에서 `dig` 명령을 실행하여 연결된 IPv6 주소를 식별합니다.

   ```
   dig <db-cluster-endpoint> AAAA
   ```

   IPv6 주소가 아닌 클러스터 엔드포인트를 사용하여 클러스터에 연결합니다.

듀얼 스택 모드 리전 및 버전 가용 여부

기능 가용 여부 및 지원은 AWS 리전에 따라 다릅니다.

리전 지원

Amazon DocumentDB의 듀얼 스택 모드는 현재 모든 상용 버전에서 사용할 수 있습니다 AWS 리전.

버전 지원

듀얼 스택 모드는 Amazon DocumentDB 버전 4.0, 5.0 및 8.0에서 지원됩니다. 이러한 버전 중 하나에서 듀얼 스택 모드에 액세스할 수 없는 경우 클러스터에서 최신 엔진 패치 버전을 실행하고 있는지 확인합니다.

듀얼 스택 네트워크 클러스터에 대한 제한 사항

듀얼 스택 네트워크 클러스터에는 다음과 같은 제한이 적용됩니다.

- 클러스터는 IPv6 프로토콜을 단독으로 사용할 수 없습니다. IPv4를 단독으로 사용하거나 IPv4 및 IPv6 프로토콜(듀얼 스택 모드)을 사용할 수 있습니다.

- Amazon DocumentDB에서는 네이티브 IPv6 서브넷을 지원하지 않습니다.
- 듀얼 스택 모드를 사용하는 클러스터는 프라이빗이어야 합니다. 공개적으로 액세스할 수 없습니다.

VPC에서 클러스터 생성

다음 절차에 따라 VPC에서 클러스터를 생성할 수 있습니다. 기본 VPC를 사용하려면 2단계부터 시작하고 이미 생성된 VPC 및 서브넷 그룹을 사용할 수 있습니다. 필요하면 추가 VPC를 생성할 수도 있습니다.

> **Note**
>
> VPC의 클러스터에 공개적으로 액세스할 수 있도록 하려면 VPC 속성 `DNS hostnames` 및 `DNS resolution`을 활성화하여 VPC에 대한 DNS 정보를 업데이트해야 합니다. VPC 인스턴스의 DNS 업데이트에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 VPC의 DNS 속성 보기 및 업데이트를 참조하세요.

VPC에 클러스터를 생성하려면 아래 단계를 따릅니다.

- 1단계: VPC 생성
- 2단계: 서브넷 그룹 생성
- 3단계: VPC 보안 그룹 만들기
- 4단계: VPC에서 클러스터 생성

1단계: VPC 생성

최소 2개의 가용 영역에 서브넷이 있는 VPC를 생성합니다. 서브넷 그룹을 생성할 때 이들 서브넷이 사용됩니다. 기본 VPC가 있는 경우에는 해당 AWS 리전의 각 가용 영역에 서브넷이 자동으로 생성됩니다.

자세한 내용은 DocumentDB 클러스터와 함께 사용할 IPv4 전용 VPC 생성 섹션을 참조하거나 Amazon Virtual Private Cloud 사용 설명서의 VPC 생성을 참조하세요.

2단계: 서브넷 그룹 생성

서브넷 그룹은 사용자가 VPC에 생성한 다음 클러스터에 대해 지정하는 서브넷(일반적으로 프라이빗)의 모음입니다. 서브넷 그룹을 사용하면 AWS CLI 또는 DocumentDB API를 사용하여 클러스터를

생성할 때 특정 VPC를 지정할 수 있습니다. 를 사용하는 경우 사용할 VPC와 서브넷만 선택할 AWS Management Console수 있습니다. 각 서브넷 그룹은 해당 AWS 리전에서 두 개 이상의 가용 영역에 한 개 이상의 서브넷이 있어야 합니다. 모범 사례로서, 각 서브넷 그룹에는 AWS 리전의 가용 영역마다 하나 이상의 서브넷이 있어야 합니다.

클러스터에 공개적으로 액세스할 수 있도록 하려면 서브넷 그룹의 서브넷에 인터넷 게이트웨이가 있어야 합니다. 서브넷용 인터넷 게이트웨이에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 인터넷 게이트웨이를 사용하여 인터넷 액세스 활성화를 참조하세요.

> **Note**
>
> 로컬 영역의 서브넷 그룹에는 서브넷이 하나만 있을 수 있습니다.

VPC에서 클러스터를 생성할 때 이에 대한 서브넷 그룹을 선택합니다. Amazon DocumentDB는 클러스터와 연결할 서브넷과 해당 서브넷 내의 IP 주소를 선택합니다. 서브넷 그룹이 없는 경우 Amazon DocumentDB는 클러스터를 생성할 때 기본 서브넷 그룹을 생성합니다. DocumentDB에서 탄력적 네트워크 인터페이스를 생성하고 해당 IP 주소로 클러스터에 연결합니다. 클러스터는 서브넷이 포함된 가용 영역을 사용합니다.

이 단계에서는 서브넷 그룹을 생성하고 VPC용으로 만든 서브넷을 추가합니다.

서브넷 그룹 생성

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 서브넷 그룹을 선택합니다.
3. 생성을 선택합니다.
4. 이름에 서브넷 그룹의 이름을 입력합니다.
5. 설명에 서브넷 그룹에 대한 설명을 입력합니다.
6. 서브넷 추가 섹션의 VPC에서 기본 VPC 또는 생성한 VPC를 선택합니다. 그런 다음 가용 영역에서 해당 서브넷을 포함하는 가용 영역을 선택한 후 서브넷에서 해당 서브넷을 선택합니다.
7. 생성을 선택합니다.

   새 서브넷 그룹은 DocumentDB 콘솔의 서브넷 그룹 목록에 표시됩니다. 서브넷 그룹을 선택하면 창 하단에 있는 세부 정보 창에서 그룹과 연결된 모든 서브넷을 포함한 세부 정보를 확인할 수 있습니다.

3단계: VPC 보안 그룹 만들기

클러스터를 생성하기 전에 클러스터와 연결할 VPC 보안 그룹을 생성합니다. VPC 보안 그룹을 생성하지 않는 경우 클러스터를 생성할 때 기본 보안 그룹을 사용할 수 있습니다. 클러스터에 대한 보안 그룹을 생성하는 방법에 대한 지침은 섹션을 참조DocumentDB 클러스터와 함께 사용할 IPv4 전용 VPC 생성하거나 Amazon Virtual Private Cloud 사용 설명서의 보안 그룹을 사용하여 AWS 리소스에 대한 트래픽 제어를 참조하세요.

4단계: VPC에서 클러스터 생성

이 단계에서는 이전 단계에서 만든 VPC 이름, 서브넷 그룹, VPC 보안 그룹을 사용해 클러스터를 만듭니다.

> **Note**
>
> VPC의 클러스터에 공개적으로 액세스할 수 있도록 하려면 VPC 속성 `DNS hostnames` 및 `DNS resolution`을 활성화해야 합니다. 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 VPC의 DNS 속성 보기 및 업데이트를 참조하세요.

클러스터 생성 방법에 대한 자세한 내용은 Amazon DocumentDB 클러스터 생성 섹션을 참조하세요.

연결 섹션에 메시지가 표시되면 VPC 이름, 서브넷 그룹 및 VPC 보안 그룹을 입력합니다.

> **Note**
>
> 현재 DocumentDBDocumentDB 클러스터에 대해서는 VPC 업데이트가 지원되지 않습니다.

VPC에서 Amazon DocumentDB 클러스터에 액세스

Amazon DocumentDB에서는 VPC에 있는 클러스터에 액세스할 수 있도록 다음 시나리오를 지원합니다.

주제

- 동일한 VPC에 있는 Amazon EC2 인스턴스가 VPC 내에 있는 클러스터에 액세스
- VPC에 있는 클러스터에 다른 VPC에 있는 Amazon EC2 인스턴스가 액세스

동일한 VPC에 있는 Amazon EC2 인스턴스가 VPC 내에 있는 클러스터에 액세스

VPC에서 클러스터를 사용하는 일반적인 사례는 동일한 VPC의 Amazon EC2 인스턴스에서 실행 중인 애플리케이션 서버와 데이터를 공유하는 것입니다.

동일한 VPC에서 EC2 인스턴스와 클러스터 간 액세스를 관리하는 가장 간단한 방법은 다음과 같습니다.

- 클러스터가 위치할 VPC 보안 그룹을 만듭니다. 이 보안 그룹을 사용해 클러스터에 대한 액세스를 제한할 수 있습니다. 예를 들어 이 보안 그룹에 대한 사용자 지정 규칙을 생성할 수 있습니다. 이렇게 하면 클러스터를 생성할 때 할당한 포트와 개발 또는 기타 목적으로 클러스터에 액세스하는 데 사용하는 IP 주소를 사용하여 TCP 액세스를 허용할 수 있습니다.
- EC2 인스턴스(웹 서버 및 클라이언트)가 포함될 VPC 보안 그룹을 생성합니다. 이 보안 그룹은 필요할 경우 VPC의 라우팅 테이블을 사용해 인터넷에서 EC2 인스턴스 액세스를 허용할 수 있습니다. 예를 들어, 이 보안 그룹에서 TCP가 포트 22를 통해 EC2 인스턴스에 액세스하도록 허용하는 규칙을 설정할 수 있습니다.
- 클러스터에 대한 보안 그룹에서 EC2 인스턴스에 대해 생성한 보안 그룹으로부터의 연결을 허용하는 사용자 지정 규칙을 만듭니다. 이러한 규칙을 통해 보안 그룹의 모든 구성원은 클러스터에 액세스할 수 있게 됩니다.

별도의 가용 영역에서 구성된 추가 퍼블릭 및 프라이빗 서브넷이 있습니다. DocumentDB 서브넷 그룹은 최소 2개의 가용 영역에 서브넷이 필요합니다. 추가 서브넷을 사용하면 향후 다중 AZ 클러스터 배포로 쉽게 전환할 수 있습니다.

이 시나리오에 대해 퍼블릭 서브넷과 프라이빗 서브넷 모두에서 VPC를 생성하는 방법에 대한 지침은 DocumentDB 클러스터와 함께 사용할 IPv4 전용 VPC 생성 섹션을 참조하세요.

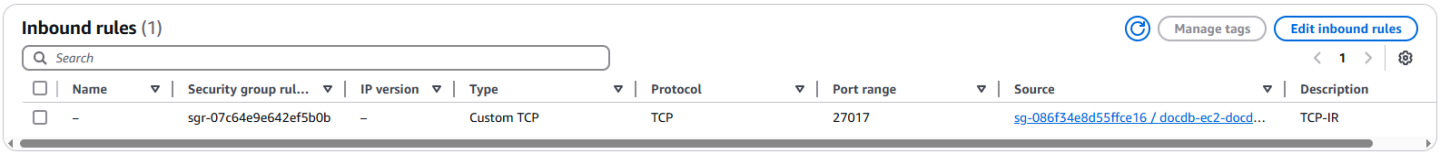
> **Tip**
>
> 클러스터를 생성할 때 Amazon EC2 인스턴스와 DocumentDB 클러스터 간의 네트워크 연결을 자동으로 설정할 수 있습니다. 자세한 내용은 Amazon EC2 자동 연결 섹션을 참조하세요.

VPC 보안 그룹에서 다른 보안 그룹의 연결을 허용하는 규칙을 생성하려면 다음을 수행합니다.

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.
2. 탐색 창에서 보안 그룹을 찾아 선택합니다.

3. 다른 보안 그룹의 구성원에 대한 액세스를 허용할 보안 그룹을 선택하거나 생성합니다. 클러스터에 사용할 보안 그룹입니다. 인바운드 규칙 탭을 선택한 후 인바운드 규칙 편집을 선택합니다.
4. 인바운드 규칙 편집 페이지에서 규칙 추가]를 선택합니다.
5. 유형에서 사용자 지정 TCP와 같이 클러스터를 생성할 때 사용한 포트에 해당하는 항목을 선택합니다.
6. 소스 필드에 일치하는 보안 그룹을 나열하는 보안 그룹의 ID를 입력합니다. 이 보안 그룹에서 보호 중인 리소스에 대한 액세스를 허용할 구성원이 포함된 보안 그룹을 선택합니다. 위 시나리오에서 이 보안 그룹을 EC2 인스턴스에 사용합니다.
7. 필요한 경우 유형으로 모든 TCP를 지정하고 소스 필드에 보안 그룹을 입력한 규칙을 생성하여 TCP 프로토콜에 대해 단계를 반복합니다. UDP 프로토콜을 사용하려는 경우 유형이 모든 UDP이고 소스에 보안 그룹이 있는 규칙을 생성합니다.
8. 규칙 저장을 선택합니다.

다음 화면은 소스에 대한 보안 그룹이 포함된 인바운드 규칙을 보여 줍니다.



EC2 인스턴스에서 인스턴스에 연결하는 방법에 대한 자세한 내용은 Amazon EC2 자동 연결 섹션을 참조하세요.

VPC에 있는 클러스터에 다른 VPC에 있는 Amazon EC2 인스턴스가 액세스

클러스터가 액세스 시 사용할 EC2 인스턴스와 다른 VPC에 있는 경우 클러스터에 액세스하기 위해 VPC 피어링을 사용할 수 있습니다.

VPC 피어링 연결은 프라이빗 IP 주소를 사용하여 두 VPC 간에 트래픽을 라우팅할 수 있도록 하기 위한 두 VPC 사이의 네트워킹 연결입니다. 동일한 네트워크에 속하는 경우와 같이 VPC의 리소스가 서로 통신할 수 있습니다. 자체 VPC, 다른 AWS 계정VPCs 또는 다른 계정의 VPC 간에 VPC 피어링 연결을 생성할 수 있습니다 AWS 리전. VPC 피어링에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 VPC 피어링을 참조하세요.

DocumentDB 클러스터와 함께 사용할 IPv4 전용 VPC 생성

일반적인 시나리오에는 Amazon VPC 서비스를 기반으로 하는 가상 프라이빗 클라우드(VPC)의 클러스터가 포함됩니다. 예를 들어, 이 VPC는 동일한 VPC에서 실행 중인 서비스 또는 애플리케이션과 데이터를 공유할 수 있습니다. 이 주제에서는 이 시나리오를 위한 VPC를 생성합니다.

주제

- 1단계: 프라이빗 서브넷과 퍼블릭 서브넷이 있는 VPC 생성
- 2단계: 퍼블릭 애플리케이션을 위한 VPC 보안 그룹 생성
- 3단계: 프라이빗 클러스터에 대한 VPC 보안 그룹 생성
- 4단계: 서브넷 그룹 생성
- VPC 삭제

퍼블릭 인터넷이 아닌 애플리케이션에서만 클러스터를 사용할 수 있어야 합니다. 따라서 퍼블릭 서브넷과 프라이빗 서브넷을 모두 포함하여 VPC를 생성합니다. 퍼블릭 서브넷에서 애플리케이션을 호스팅하므로 애플리케이션에서 퍼블릭 인터넷에 액세스할 수 있습니다. 클러스터는 프라이빗 서브넷에서 호스팅됩니다. 애플리케이션은 동일한 VPC 내에서 호스팅되므로 클러스터에 연결할 수 있습니다. 그러나 퍼블릭 인터넷에서는 클러스터를 사용할 수 없으므로 보안이 강화됩니다.

이 주제의 해당 절차에서는 별도의 가용 영역에서 추가 퍼블릭 및 프라이빗 서브넷을 구성합니다. 이러한 서브넷은 해당 절차에서 사용하지 않습니다. DocumentDB 서브넷 그룹은 최소 2개의 가용 영역에 서브넷이 필요합니다. 추가 서브넷을 사용하면 2개 이상의 DocumentDB 인스턴스를 쉽게 구성할 수 있습니다.

이 주제에서는 Amazon DocumentDB 클러스터에 대한 VPC 구성에 대해 설명합니다. Amazon VPC에 대한 자세한 내용은 Amazon VPC 사용 설명서를 참조하세요.

> **Tip**
>
> 클러스터를 생성할 때 Amazon EC2 인스턴스와 DocumentDB 클러스터 간의 네트워크 연결을 자동으로 설정할 수 있습니다. 네트워크 구성은 이 시나리오에서 설명한 것과 비슷합니다. 자세한 내용은 Amazon EC2 자동 연결 섹션을 참조하세요.

1단계: 프라이빗 서브넷과 퍼블릭 서브넷이 있는 VPC 생성

다음은 퍼블릭 서브넷과 프라이빗 서브넷을 모두 포함하는 VPC를 생성하는 절차입니다.

VPC 및 서브넷을 생성하려면

1. https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.
2. 의 오른쪽 상단 모서리에서 VPC를 생성할 리전을 AWS Management Console선택합니다. 이 예에서는 미국 서부(오레곤) 리전을 사용합니다.

3. 왼쪽 상단 모서리에서 VPC 대시보드를 선택합니다. VPC 생성을 시작하려면 VPC 생성을 선택합니다.
4. VPC 설정의 생성할 리소스에서 VPC 등을 선택합니다.
5. VPC 설정을 위해 다음 값을 설정합니다.

   - 이름 태그 자동 생성 - **`example`**
   - IPv4 CIDR 블록 - **`10.0.0.0/16`**
   - IPv6 CIDR 블록 - IPv6 CIDR 블록 없음
   - 테넌시 - 기본값
   - 가용 영역(AZ) 수 - 2
   - AZ 사용자 지정 - 기본값 유지
   - 퍼블릭 서브넷 수 - 2
   - 프라이빗 서브넷 수 – 2
   - 서브넷 CIDR 블록 사용자 지정 - 기본값 유지
   - NAT 게이트웨이($) – 없음
   - VPC 엔드포인트 – 없음
   - DNS 옵션 - 기본값 유지
6. VPC 생성을 선택합니다.

2단계: 퍼블릭 애플리케이션을 위한 VPC 보안 그룹 생성

이제 퍼블릭 액세스를 위한 보안 그룹을 생성합니다. VPC의 퍼블릭 EC2 인스턴스에 연결하려면 VPC 보안 그룹에 인바운드 규칙을 추가해야 합니다. 이를 통해 인터넷에서 트래픽을 연결할 수 있습니다.

VPC 보안 그룹을 생성하려면

1. https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.
2. VPC 대시보드, 보안 그룹, 보안 그룹 생성을 차례대로 선택합니다.
3. 보안 그룹 생성 페이지에서 다음 값을 설정합니다.

   - 보안 그룹 이름 – **`example-securitygroup`**
   - 설명 – **`Application security group`**
   - VPC - vpc-example과 같이 이전에 생성한 VPC를 선택합니다.
4. 인바운드 규칙을 보안 그룹에 추가합니다.

a. Secure Shell(SSH)을 사용하여 VPC의 EC2 인스턴스에 연결하는 데 사용할 IP 주소를 지정합니다. 퍼블릭 IP 주소를 확인하려면 다른 브라우저 창 또는 탭에서 https://checkip.amazonaws.com의 서비스를 사용합니다. IP 주소의 예는 `203.0.113.25/32`입니다.

   대부분의 경우 고정 IP 주소가 없는 방화벽 뒤나 인터넷 서비스 제공업체(ISP)를 통해 연결하는 경우가 많습니다. 그렇다면 클라이언트 컴퓨터에서 사용하는 IP 주소 범위를 찾습니다.

   > **Warning**
   >
   > SSH 액세스에 `0.0.0.0/0`을 사용하는 경우 모든 IP 주소가 SSH를 사용하여 퍼블릭 인스턴스에 액세스할 수 있도록 활성화합니다. 이 방법은 테스트 환경에서 잠시 사용하는 것은 괜찮지만 프로덕션 환경에서는 안전하지 않습니다. 프로덕션에서는 SSH를 사용하여 인스턴스에 액세스할 수 있는 특정 IP 주소 또는 주소 범위만 인증합니다.

b. 인바운드 규칙 섹션에서 규칙 추가를 선택합니다.

c. 새로운 인바운드 규칙으로 다음 값을 설정하여 SSH에서 Amazon EC2 인스턴스에 액세스하도록 허용합니다. 이 작업을 수행하면 EC2 인스턴스에 연결하여 애플리케이션 및 다른 유틸리티를 설치할 수 있습니다. 또한 EC2 인스턴스에 연결하여 애플리케이션의 콘텐츠를 업로드할 수도 있습니다.

   - 유형 - **SSH**
   - 소스 - a단계에서 생성한 IP 주소 또는 범위(예: **203.0.113.25/32**)입니다.

d. 규칙 추가를 선택합니다.

e. 새 인바운드 규칙에 다음 값을 설정하여 애플리케이션에 대한 HTTP 액세스를 허용합니다.

   - 유형 - **HTTP**
   - 소스 - **0.0.0.0/0**

5. 보안 그룹을 생성하려면 보안 그룹 생성을 선택합니다.

   다른 절차에서 나중에 필요하므로 보안 그룹 ID를 적어 둡니다.

3단계: 프라이빗 클러스터에 대한 VPC 보안 그룹 생성

클러스터를 프라이빗으로 유지하려면 프라이빗 액세스를 위한 보조 보안 그룹을 생성합니다. VPC의 프라이빗 클러스터에 연결하려면 애플리케이션으로부터의 트래픽만 허용하는 인바운드 규칙을 VPC 보안 그룹에 추가합니다.

VPC 보안 그룹을 생성하려면

1. https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.
2. VPC 대시보드, 보안 그룹, 보안 그룹 생성을 차례대로 선택합니다.
3. 보안 그룹 생성 페이지에서 다음 값을 설정합니다.
   - 보안 그룹 이름 – **example-securitygroup**
   - 설명 – **Instance security group**
   - VPC - vpc-example과 같이 이전에 생성한 VPC를 선택합니다.
4. 인바운드 규칙을 보안 그룹에 추가합니다.
   a. 인바운드 규칙 섹션에서 규칙 추가를 선택합니다.
   b. 새로운 인바운드 규칙으로 다음 값을 설정하여 Amazon EC2 인스턴스의 포트 27017에서 DocumentDB 트래픽을 허용합니다. 이 작업을 수행하면 애플리케이션에서 클러스터에 연결할 수 있습니다. 그러면 애플리케이션의 데이터를 데이터베이스에 저장하고 검색할 수 있습니다.
      - 유형 - **Custom TCP**
      - 소스 - 이 주제의 앞부분에서 만든 애플리케이션 보안 그룹의 식별자(예: sg-9edd5cfb)입니다.
   c. 규칙 추가를 선택합니다.
   d. 새 인바운드 규칙에 다음 값을 설정하여 애플리케이션에 대한 HTTP 액세스를 허용합니다.
      - 유형 - **HTTP**
      - 소스 - **0.0.0.0/0**
5. 보안 그룹을 생성하려면 보안 그룹 생성을 선택합니다.

4단계: 서브넷 그룹 생성

서브넷 그룹은 사용자가 VPC에서 만든 후 클러스터에 대해 지정하는 서브넷의 모음입니다. 서브넷 그룹에서 클러스터를 생성할 때 특정 VPC를 지정할 수 있습니다.

서브넷 그룹 생성

1. VPC에서 데이터베이스의 프라이빗 서브넷을 식별합니다.

   a. https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.

   b. VPC 대시보드를 선택한 다음 서브넷을 선택합니다.

   c. example-subnet-private1-us-west-2a 및 example-subnet-private2-us-west-2b와 같이 1단계에서 생성한 서브넷의 서브넷 ID를 기록해 둡니다. 서브넷 그룹을 생성할 때 서브넷 ID가 필요합니다.

2. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.

   Amazon VPC 콘솔이 아닌 Amazon DocumentDB 콘솔에 연결해야 합니다.

3. 탐색 창에서 서브넷 그룹을 선택합니다.

4. 생성을 선택합니다.

5. 서브넷 그룹 생성 페이지에서 서브넷 그룹 세부 정보 섹션에 해당 값을 설정합니다.

   - 이름 – **`example-db-subnet-group`**
   - 설명 – **`Instance security group`**

6. 서브넷 추가 섹션에서 해당 값을 설정합니다.

   - VPC - vpc-example과 같이 이전에 생성한 VPC를 선택합니다.
   - 가용 영역 - 1단계에서 생성한 두 가용 영역을 모두 선택합니다. 예: us-west-2a 및 us-west-2b
   - 서브넷 - 1단계에서 생성한 프라이빗 서브넷을 선택합니다.

7. 생성을 선택합니다.

새 서브넷 그룹은 DocumentDB 콘솔의 서브넷 그룹 목록에 표시됩니다. 서브넷 그룹을 선택하여 세부 정보 창에서 세부 정보를 볼 수 있습니다. 이러한 세부 정보에는 그룹과 연결된 모든 서브넷이 포함됩니다.

> **Note**
>
> 이 VPC를 생성하여 DocumentDB 클러스터와 연결한 경우 Amazon DocumentDB 클러스터 생성의 지침에 따라 클러스터를 생성합니다.

VPC 삭제

더 이상 필요하지 않은 VPC 및 그 안에 사용되는 다른 리소스는 삭제할 수 있습니다.

> **Note**
>
> 이 주제에서 생성한 VPC에 리소스를 추가한 경우 VPC를 삭제하기 전에 해당 리소스를 삭제해야 할 수 있습니다. 예를 들어 해당 리소스에는 Amazon EC2 인스턴스 또는 DocumentDB 클러스터가 포함될 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC의 삭제를 참조하세요.

VPC 및 관련 리소스를 삭제하려면

1. 서브넷 그룹을 삭제합니다.
   a. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
   b. 탐색 창에서 서브넷 그룹을 선택합니다.
   c. example-db-subnet-group과 같이 삭제할 서브넷 그룹을 선택합니다.
   d. 삭제를 선택한 다음 확인 창에서 삭제를 선택합니다.
2. VPC ID를 기록합니다.
   a. https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.
   b. VPC 대시보드를 선택한 다음 해당 VPC를 선택합니다.
   c. 목록에서 생성한 VPC를 식별합니다(예:vpc-example).
   d. 생성한 VPC의 VPC ID를 기록합니다. 이후 단계에서 해당 VPC ID가 필요합니다.
3. 보안 그룹을 삭제합니다.
   a. https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.
   b. VPC 대시보드를 선택한 다음 보안 그룹을 선택합니다.
   c. example-securitygroup과 같은 Amazon DocumentDB 클러스터의 보안 그룹을 선택합니다.
   d. 작업에서 보안 그룹 삭제를 선택한 후 확인 페이지에서 삭제를 선택합니다.
   e. 보안 그룹 페이지로 돌아가 example-securitygroup과 같은 Amazon EC2 인스턴스에 대한 보안 그룹을 선택합니다.
   f. 작업에서 보안 그룹 삭제를 선택한 후 확인 페이지에서 삭제를 선택합니다.

4. VPC를 삭제합니다.

   a. https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.

   b. VPC 대시보드를 선택한 다음 해당 VPC를 선택합니다.

   c. vpc-example과 같이 삭제할 VPC를 선택합니다.

   d. 작업에서 VPC 삭제를 선택합니다.

      확인 페이지에는 VPC와 연결된 서브넷을 포함하여 삭제될 VPC와 연결된 다른 리소스가 표시됩니다.

   e. 확인 대화 상자에 **delete**를 입력한 후 삭제를 선택합니다.

DocumentDB 클러스터와 함께 사용할 듀얼 스택 VPC 생성

일반적인 시나리오에는 Amazon VPC 서비스를 기반으로 하는 가상 프라이빗 클라우드(VPC)의 클러스터가 포함됩니다. 이 VPC는 동일한 VPC에서 실행 중인 퍼블릭 Amazon EC2 인스턴스와 데이터를 공유합니다. 이 주제에서는 이 시나리오를 위한 VPC를 생성합니다.

주제

- 1단계: 프라이빗 서브넷과 퍼블릭 서브넷이 있는 VPC 생성
- 2단계: 퍼블릭 Amazon EC2 인스턴스에 대한 VPC 보안 그룹 생성
- 3단계: 프라이빗 클러스터에 대한 VPC 보안 그룹 생성
- 4단계: 서브넷 그룹 생성
- 5단계: 듀얼 스택 모드로 Amazon EC2 인스턴스 생성
- 6단계: 듀얼 스택 모드에서 클러스터 생성
- 7단계: Amazon EC2 인스턴스 및 DB 클러스터 연결
- VPC 삭제

이 절차에서는 듀얼 스택 모드에서 실행되는 데이터베이스와 함께 작동하는 이 시나리오의 VPC를 생성합니다. IPv6 주소 지정 프로토콜을 통한 연결을 지원하는 듀얼 스택 모드입니다. IP 주소에 대한 자세한 내용은 Amazon DocumentDB IP 주소 지정 단원을 참조하십시오.

듀얼 스택 네트워크 클러스터는 대부분의 리전에서 지원됩니다. 자세한 내용은 듀얼 스택 모드 리전 및 버전 가용 여부 섹션을 참조하세요. 이중 스택 모드의 제한 사항을 보려면 듀얼 스택 네트워크 클러스터에 대한 제한 사항을 참조하세요.

이 주제와 IPv4 전용 주제는 동일한 VPC에 퍼블릭 및 프라이빗 서브넷을 생성합니다. 하나의 VPC에서 Amazon DocumentDB 클러스터를 생성하고 다른 VPC에서 Amazon EC2 인스턴스를 생성하는 방법에 대한 자세한 내용은 VPC에서 Amazon DocumentDB 클러스터에 액세스 섹션을 참조하세요.

퍼블릭 인터넷이 아닌 Amazon EC2 인스턴스에서만 DocumentDB 클러스터를 사용할 수 있어야 합니다. 따라서 퍼블릭 서브넷과 프라이빗 서브넷을 모두 포함하여 VPC를 생성합니다. 퍼블릭 서브넷에서 EC2 인스턴스를 호스팅하므로 EC2 인스턴스에서 퍼블릭 인터넷에 액세스할 수 있습니다. 클러스터는 프라이빗 서브넷에서 호스팅됩니다. EC2 인스턴스는 동일한 VPC 내에서 호스팅되므로 클러스터에 연결할 수 있습니다. 그러나 퍼블릭 인터넷에서는 클러스터를 사용할 수 없으므로 보안이 강화됩니다.

이 주제의 해당 절차에서는 별도의 가용 영역에서 추가 퍼블릭 및 프라이빗 서브넷을 구성합니다. 이러한 서브넷은 해당 절차에서 사용하지 않습니다. DocumentDB 서브넷 그룹은 최소 2개의 가용 영역에 서브넷이 필요합니다. 추가 서브넷을 사용하면 2개 이상의 DocumentDB 인스턴스를 쉽게 구성할 수 있습니다.

듀얼 스택 모드를 사용하는 클러스터를 생성하려면 네트워크 유형 설정에서 듀얼 스택 모드를 지정합니다. 동일한 설정으로 클러스터를 수정할 수도 있습니다. 클러스터 생성에 대한 자세한 내용은 Amazon DocumentDB 클러스터 생성 섹션을 참조하세요. DB 클러스터 수정에 대한 자세한 내용은 Amazon DocumentDB 클러스터 수정 섹션을 참조하세요.

이 주제에서는 Amazon DocumentDB 클러스터에 대한 VPC 구성에 대해 설명합니다. Amazon VPC에 대한 자세한 내용은 Amazon VPC 사용 설명서를 참조하세요.

1단계: 프라이빗 서브넷과 퍼블릭 서브넷이 있는 VPC 생성

다음은 퍼블릭 서브넷과 프라이빗 서브넷을 모두 포함하는 VPC를 생성하는 절차입니다.

VPC 및 서브넷을 생성하려면

1. https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.
2. 의 오른쪽 상단 모서리에서 VPC를 생성할 리전을 AWS Management Console선택합니다. 이 예에서는 미국 서부(오레곤) 리전을 사용합니다.
3. 왼쪽 상단 모서리에서 VPC 대시보드를 선택합니다. VPC 생성을 시작하려면 VPC 생성을 선택합니다.
4. VPC 설정의 생성할 리소스에서 VPC 등을 선택합니다.
5. VPC 설정을 위해 다음 값을 설정합니다.

    - 이름 태그 자동 생성 - **example-dual-stack**

- IPv4 CIDR 블록 - **`10.0.0.0/16`**
- IPv6 CIDR 블록 - Amazon 제공 IPv6 CIDR 블록
- 테넌시 - 기본값
- 가용 영역(AZ) 수 - 2
- AZ 사용자 지정 - 기본값 유지
- 퍼블릭 서브넷 수 - 2
- 프라이빗 서브넷 수 – 2
- 서브넷 CIDR 블록 사용자 지정 - 기본값 유지
- NAT 게이트웨이($) – 없음
- 송신 전용 인터넷 게이트웨이 - 없음
- VPC 엔드포인트 – 없음
- DNS 옵션 - 기본값 유지

6. VPC 생성을 선택합니다.

2단계: 퍼블릭 Amazon EC2 인스턴스에 대한 VPC 보안 그룹 생성

이제 퍼블릭 액세스를 위한 보안 그룹을 생성합니다. VPC의 퍼블릭 EC2 인스턴스에 연결하려면 인터넷으로부터의 트래픽 연결을 허용하는 VPC 보안 그룹에 인바운드 규칙을 추가해야 합니다.

VPC 보안 그룹을 생성하려면

1. https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.
2. VPC 대시보드, 보안 그룹, 보안 그룹 생성을 차례대로 선택합니다.
3. 보안 그룹 생성 페이지에서 다음 값을 설정합니다.
   - 보안 그룹 이름 – **`example-dual-stack-securitygroup`**
   - 설명 – **`Dual-stack security group`**
   - VPC - vpc-example-dual-stack과 같이 이전에 생성한 VPC를 선택합니다.
4. 인바운드 규칙을 보안 그룹에 추가합니다.
   a. Secure Shell(SSH)을 사용하여 VPC의 EC2 인스턴스에 연결하는 데 사용할 IP 주소를 지정합니다. 퍼블릭 IP 주소를 확인하려면 다른 브라우저 창 또는 탭에서 https://checkip.amazonaws.com의 서비스를 사용합니다.

인터넷 프로토콜 버전 4(IPv4) 주소 범위의 예는 입니다`203.0.113.25/32`. IPv6(인터넷 프로토콜 버전 6) 주소 범위의 예는 `2001:db8:1234:1a00::/64`입니다.

대부분의 경우 고정 IP 주소가 없는 방화벽 뒤나 인터넷 서비스 제공업체(ISP)를 통해 연결하는 경우가 많습니다. 그렇다면 클라이언트 컴퓨터에서 사용하는 IP 주소 범위를 찾습니다.

> **Warning**
>
> IPv4에 `0.0.0.0/0`, IPv6에 `::0`을 사용하면 SSH를 통해 모든 IP 주소가 퍼블릭 인스턴스에 액세스하도록 설정할 수 있습니다. 이 방법은 테스트 환경에서 잠시 사용하는 것은 괜찮지만 프로덕션 환경에서는 안전하지 않습니다. 프로덕션에서는 특정 IP 주소나 주소 범위만 인스턴스에 액세스하도록 허용하세요.

b. 인바운드 규칙 섹션에서 규칙 추가를 선택합니다.

c. 새로운 인바운드 규칙으로 다음 값을 설정하여 SSH에서 Amazon EC2 인스턴스에 액세스하도록 허용합니다. 이 작업을 수행하면 EC2 인스턴스에 연결하여 애플리케이션 또는 다른 유틸리티를 설치할 수 있습니다. EC2 인스턴스에 액세스할 수 있도록 IP 주소를 지정합니다.

- 유형 - **SSH**
- 소스 - a단계의 IP 주소 또는 범위입니다. IPv4 IP 주소 범위의 예는 **203.0.113.25/3**2입니다. IPv6 IP 주소 범위의 예는 **2001:DB8::/32**입니다.

d. 규칙 추가를 선택합니다.

5. 보안 그룹을 생성하려면 보안 그룹 생성을 선택합니다.

   다른 절차에서 나중에 필요하므로 보안 그룹 ID를 적어 둡니다.

3단계: 프라이빗 클러스터에 대한 VPC 보안 그룹 생성

클러스터를 프라이빗으로 유지하려면 프라이빗 액세스를 위한 보조 보안 그룹을 생성합니다. VPC의 프라이빗 클러스터에 연결하려면 VPC 보안 그룹에 인바운드 규칙을 추가합니다. 이 규칙은 Amazon EC2 인스턴스로부터의 트래픽만 허용합니다.

VPC 보안 그룹을 생성하려면

1. https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.
2. VPC 대시보드, 보안 그룹, 보안 그룹 생성을 차례대로 선택합니다.
3. 보안 그룹 생성 페이지에서 다음 값을 설정합니다.

- 보안 그룹 이름 – **example-dual-stack-cluster-securitygroup**
- 설명 – **Dual-stack cluster security group**
- VPC - vpc-example-dual-stack과 같이 이전에 생성한 VPC를 선택합니다.

4. 인바운드 규칙을 보안 그룹에 추가합니다.

   a. 인바운드 규칙 섹션에서 규칙 추가를 선택합니다.

   b. 새로운 인바운드 규칙으로 다음 값을 설정하여 Amazon EC2 인스턴스의 포트 27017에서 DocumentDB 트래픽을 허용합니다. 이 작업을 수행하면 EC2 인스턴스에서 클러스터에 연결할 수 있습니다. 이를 통해 EC2 인스턴스의 데이터를 데이터베이스에 전송할 수 있습니다.

      - 유형 - **Custom TCP**
      - 소스 - 이 주제의 앞부분에서 만든 EC2 보안 그룹의 식별자(예: sg-9edd5cfb)입니다.

5. 보안 그룹을 생성하려면 보안 그룹 생성을 선택합니다.

4단계: 서브넷 그룹 생성

서브넷 그룹은 사용자가 VPC에서 만든 후 클러스터에 대해 지정하는 서브넷의 모음입니다. 서브넷 그룹을 사용하여 클러스터를 생성할 때 특정 VPC를 지정할 수 있습니다. DUAL과 호환 가능한 서브넷 그룹을 생성하려면 모든 서브넷이 DUAL과 호환 가능해야 합니다. DUAL과 호환 가능하려면 서브넷에 IPv6 CIDR이 연결되어 있어야 합니다.

서브넷 그룹 생성

1. VPC에서 데이터베이스의 프라이빗 서브넷을 식별합니다.

   a. https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.

   b. VPC 대시보드를 선택한 다음 서브넷을 선택합니다.

   c. example-dual-stack-subnet-private1-us-west-2a 및 example-dual-stack-subnet-private2-us-west-2b와 같이 1단계에서 생성한 서브넷의 서브넷 ID를 기록해 둡니다. 서브넷 그룹을 생성할 때 서브넷 ID가 필요합니다.

2. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.

   Amazon VPC 콘솔이 아닌 Amazon DocumentDB 콘솔에 연결해야 합니다.

3. 탐색 창에서 서브넷 그룹을 선택합니다.

4. 생성을 선택합니다.

5. 서브넷 그룹 생성 페이지에서 서브넷 그룹 세부 정보 섹션에 해당 값을 설정합니다.
   - 이름 – **example-dual-stack-cluster-subnet-group**
   - 설명 – **Dual-stack cluster subnet group**
6. 서브넷 추가 섹션에서 해당 값을 설정합니다.
   - VPC - vpc-example-dual-stack과 같이 이전에 생성한 VPC를 선택합니다.
   - 가용 영역 - 1단계에서 생성한 두 가용 영역을 모두 선택합니다. 예: us-west-2a 및 us-west-2b
   - 서브넷 - 1단계에서 생성한 프라이빗 서브넷을 선택합니다.
7. 생성을 선택합니다.

새 서브넷 그룹은 DocumentDB 콘솔의 서브넷 그룹 목록에 표시됩니다. 서브넷 그룹을 선택하여 세부 정보 창에서 세부 정보를 볼 수 있습니다. 이러한 세부 정보에는 그룹과 연결된 모든 서브넷이 포함됩니다.

5단계: 듀얼 스택 모드로 Amazon EC2 인스턴스 생성

Amazon EC2 인스턴스를 생성하려면 Amazon Elastic Compute Cloud 사용 설명서의 콘솔에서 인스턴스 시작 마법사를 사용하여 EC2 인스턴스 시작의 지침을 따르세요.

인스턴스 세부 정보 구성 페이지에서 이러한 값을 설정하고 다른 값은 기본값으로 유지합니다.

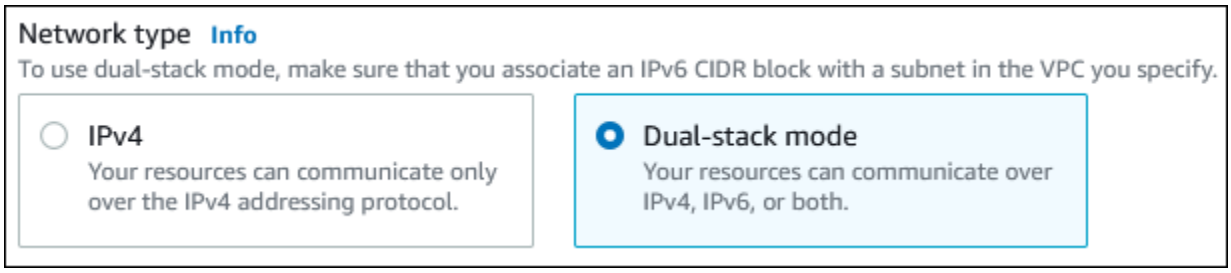
- 네트워크 - 1단계: 프라이빗 서브넷과 퍼블릭 서브넷이 있는 VPC 생성에서 생성한 vpc-example-dual-stack-vpc(vpc-identifier)와 같이 퍼블릭 및 프라이빗 서브넷이 있는 기존 VPC를 선택합니다.
- 서브넷 - 2단계: 퍼블릭 Amazon EC2 인스턴스에 대한 VPC 보안 그룹 생성에서 생성한 subnet-identifier | example-dual-stack-subnet-public1-us-east-2a | us-east-2a와 같은 기존 퍼블릭 서브넷을 선택합니다.
- 퍼블릭 IP 자동 할당 - 활성화를 선택합니다.
- IPv6 IP 자동 할당 - 활성화를 선택합니다.
- 방화벽(보안 그룹) - 기존 보안 그룹 선택을 선택합니다.
- 공통 보안 그룹 - 2단계: 퍼블릭 Amazon EC2 인스턴스에 대한 VPC 보안 그룹 생성에서 생성한 example-dual-stack-securitygroup과 같은 기존 보안 그룹을 선택합니다. 선택한 보안 그룹에 SSH(Secure Shell) 및 HTTP 액세스에 대한 인바운드 규칙이 포함되어 있는지 확인합니다.

6단계: 듀얼 스택 모드에서 클러스터 생성

이 단계에서는 듀얼 스택 모드로 실행되는 DB 클러스터를 생성합니다. !!!주의: 이 섹션은 IPv6 업데이트가 콘솔에 적용되면 편집해야 합니다!!!

듀얼 스택 모드에서 클러스터 생성

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 콘솔의 오른쪽 상단에서 DocumentDB 클러스터를 생성할 AWS 리전 를 선택합니다. 이 예에서는 미국 동부(오하이오) 리전을 사용합니다.
3. 탐색 창에서 클러스터를 선택합니다.
4. 클러스터 목록 페이지에서 생성을 선택합니다.
5. Amazon DocumentDB 클러스터 생성 페이지에서 인스턴스 기반 클러스터 옵션이 선택되어 있는지 확인합니다.
6. 연결 섹션의 네트워크 유형에서 듀얼 스택 모드를 선택합니다.



7. 페이지 하단에서 고급 설정 표시를 켭니다.
8. 네트워크 설정 섹션에서 해당 값을 설정합니다.
   - 가상 프라이빗 클라우드(VPC) - 1단계: 프라이빗 서브넷과 퍼블릭 서브넷이 있는 VPC 생성에서 생성한 vpc-example-dual-stack(vpc-identifier)과 같이 퍼블릭 및 프라이빗 서브넷이 있는 기존 VPC를 선택합니다.

     VPC는 서로 다른 가용 영역에 서브넷이 있어야 합니다.
   - 서브넷 그룹 - 4단계: 서브넷 그룹 생성에서 생성한 example-dual-stack-cluster-subnet-group과 같은 VPC용 서브넷 그룹을 선택합니다.
   - 퍼블릭 액세스 - 아니요를 선택합니다.
   - VPC 보안 그룹 (방화벽) - 기존 항목 선택을 선택합니다.

- 기존 VPC 보안 그룹 – 3단계: 프라이빗 클러스터에 대한 VPC 보안 그룹 생성에서 생성한 example-dual-stack-cluster-securitygroup과 같이 프라이빗 액세스에 적합하게 구성된 기존 VPC 보안 그룹을 선택합니다.

  각각에 연결된 X를 선택해 기본 보안 그룹 같은 다른 보안 그룹을 제거합니다.

- 가용 영역 - 1단계에서 생성한 가용 영역을 선택합니다. 예: us-west-2a.

  AZ 간 트래픽을 방지하려면 클러스터와 EC2 인스턴스가 동일한 가용 영역에 있어야 합니다.

9. 나머지 섹션에서 클러스터 설정을 지정합니다. 각 설정에 대한 자세한 내용은 Amazon DocumentDB 클러스터 생성 섹션을 참조하세요.

7단계: Amazon EC2 인스턴스 및 DB 클러스터 연결

Amazon EC2 인스턴스와 DocumentDB 클러스터가 듀얼 스택 모드로 생성된 후에는 IPv6 프로토콜을 사용하여 각 항목에 연결할 수 있습니다. IPv6 프로토콜을 사용하여 EC2 인스턴스에 연결하려면 Amazon Elastic Compute Cloud 사용 설명서에서 EC2 인스턴스에 연결의 지침을 따르세요.

EC2 인스턴스에서 DocumentDB 클러스터에 연결하려면 Amazon EC2를 수동으로 연결의 5단계: MongoDB 쉘 설치의 지침을 따르세요(동일한 절차에서 후속 6단계와 7단계를 계속 진행).

VPC 삭제

더 이상 필요하지 않은 VPC 및 그 안에 사용되는 다른 리소스는 삭제할 수 있습니다.

> **Note**
>
> 이 주제에서 생성한 VPC에 리소스를 추가한 경우 VPC를 삭제하기 전에 해당 리소스를 삭제해야 할 수 있습니다. 예를 들어 해당 리소스에는 Amazon EC2 인스턴스 또는 DocumentDB 클러스터가 포함될 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC의 삭제를 참조하세요.

VPC 및 관련 리소스를 삭제하려면

1. 서브넷 그룹을 삭제합니다.

   a. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.

b. 탐색 창에서 서브넷 그룹을 선택합니다.

c. example-dual-stack-cluster-subnet-group과 같은 삭제할 서브넷 그룹을 선택합니다.

d. 삭제를 선택한 다음 확인 창에서 삭제를 선택합니다.

2. VPC ID를 기록합니다.

a. https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.

b. VPC 대시보드를 선택한 다음 해당 VPC를 선택합니다.

c. 목록에서 vpc-example-dual-stack과 같이 생성한 VPC를 식별합니다.

d. 생성한 VPC의 VPC ID를 기록합니다. 이후 단계에서 해당 VPC ID가 필요합니다.

3. 보안 그룹을 삭제합니다.

a. https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.

b. VPC 대시보드를 선택한 다음 보안 그룹을 선택합니다.

c. example-dual-stack-securitygroup과 같은 Amazon DocumentDB 클러스터의 보안 그룹을 선택합니다.

d. 작업에서 보안 그룹 삭제를 선택한 후 확인 페이지에서 삭제를 선택합니다.

e. 보안 그룹 페이지로 돌아가 example-securitygroup과 같은 Amazon EC2 인스턴스에 대한 보안 그룹을 선택합니다.

f. 작업에서 보안 그룹 삭제를 선택한 후 확인 페이지에서 삭제를 선택합니다.

4. NAT 게이트웨이를 삭제합니다.

a. https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.

b. VPC 대시보드를 선택한 다음 보안 그룹을 선택합니다.

c. 생성한 VPC의 NAT 게이트웨이를 선택합니다. VPC ID를 사용하여 올바른 NAT 게이트웨이를 식별합니다.

d. 작업에서 NAT 게이트웨이 삭제(Delete NAT gateway)를 선택합니다.

e. 확인 대화 상자에 **`delete`**를 입력한 후 삭제를 선택합니다.

5. VPC를 삭제합니다.

a. https://console.aws.amazon.com/vpc Amazon VPC 콘솔을 엽니다.

b. VPC 대시보드를 선택한 다음 해당 VPC를 선택합니다.

c. vpc-example-dual-stack과 같이 삭제할 VPC를 선택합니다.

d. 작업에서 VPC 삭제를 선택합니다.

확인 페이지에는 VPC와 연결된 서브넷을 포함하여 삭제될 VPC와 연결된 다른 리소스가 표시됩니다.

e. 확인 대화 상자에 **delete**를 입력한 후 삭제를 선택합니다.

6. 탄력적 IP 주소를 해제합니다.

a. https://console.aws.amazon.com/ec2에서 EC2 콘솔을 엽니다.

b. EC2 대시보드를 선택한 다음 탄력적 IP 주소를 선택합니다.

c. 릴리스하려는 탄력적 IP 주소를 선택합니다.

d. 작업(Actions)에서 탄력적 IP 주소 릴리스(Release Elastic IP addresses)를 선택합니다.

e. 확인 대화 상자에서 해제를 선택합니다.

Amazon DocumentDB에서 백업 및 복원

Amazon DocumentDB(MongoDB 호환)는 1~35일 동안 Amazon Simple Storage Service(S3)에 데이터를 지속적으로 백업하므로 백업 보존 기간 내에 어떤 시점으로든 신속하게 복구가 가능합니다. 또한 Amazon DocumentDB는 이러한 연속 백업 프로세스의 일환으로 데이터의 스냅샷을 자동으로 생성합니다.

> **Note**
>
> 이는 서비스 관리형 Amazon S3 버킷이며 백업 파일에 액세스할 수 없습니다. 자체 백업을 제어하려면 섹션을 참조하세요데이터 덤프, 복원, 가져오기 및 내보내기.

백업 보존 기간 이후에도 클러스터 데이터의 수동 스냅샷을 생성하여 백업 데이터를 보관할 수 있습니다. 백업 프로세스는 클러스터의 성능에 영향을 주지 않습니다.

이 섹션에서는 Amazon DocumentDB의 백업 기능에 대한 사용 사례를 설명하고 Amazon DocumentDB 클러스터에 대한 백업을 관리하는 방법을 보여 줍니다.

주제

- 백업 및 복원: 개념
- 백업 스토리지 사용량 파악
- 데이터 덤프, 복원, 가져오기 및 내보내기
- 클러스터 스냅샷 고려 사항
- 자동 스냅샷과 수동 스냅샷 비교
- 수동 클러스터 스냅샷 생성
- Amazon DocumentDB 클러스터 스냅샷 복사
- Amazon DocumentDB 클러스터 스냅샷 공유
- 클러스터 스냅샷에서 복원
- 특정 시점으로 복원
- 클러스터 스냅샷 삭제

백업 및 복원: 개념

명사	설명	API(동사)

백업 보관 기간	특정 시점으로 복원을 수행할 수 있는 1~35일 사이의 기간입니다.	`create-db-cluster`
		`modify-db-cluster`
		`restore-db-cluster-to-point-in-time`

Amazon DocumentDB 스토리지 볼륨	세 개의 가용 영역에 걸쳐 여섯 가지 방식으로 데이터를 복제하는 가용성과 내구성이 뛰어난 스토리지 볼륨입니다. Amazon DocumentDB 클러스터는 클러스터의 인스턴스 수에 관계없이 내구성이 우수합니다.	`create-db-cluster` `delete-db-cluster`
백업 기간	자동 스냅샷이 생성	`create-db-cluster`

명사	설명	API(동사)

	되는 기간(일)입니다.	`describe-db-cluster`
		
		`modify-db-cluster`

자동 스냅샷	Amazon DocumentDB에서 지속적 백업 프로세스에 의해 자동으로 생성되고 클러스터 전체를 백업하는 일별 스냅샷입니다.	`restore-db-cluster-from-snapshot`
		
		`describe-db-cluster-snapshot-attributes`
		
		`describe-db-cluster-snapshots`

수동 스냅샷	백업 보존 기간 이후에 클러스터의 전체 백업을 보관하기 위해 수동으로 생성하는 스냅샷입니다.	`create-db-cluster-snapshot`
		
		`copy-db-cluster-snapshot`
		
		`delete-db-cluster-snapshot`
		
		`describe-db-cluster-snapshot-attributes`
		
		`describe-db-cluster-snapshots`
		
		`modify-db-cluster-snapshot-attribute`

백업 스토리지 사용량 파악

Amazon DocumentDB 백업 스토리지는 백업 보존 기간 내 연속 백업과 보존 기간이 경과된 수동 스냅샷으로 구성됩니다. 백업 스토리지 사용량을 제어하려면 백업 보존 간격을 줄이거나 더 이상 필요하지 않은 경우 이전 수동 스냅샷을 제거하거나, 또는 두 가지 방법을 모두 사용할 수 있습니다. Amazon DocumentDB 백업에 대한 일반적인 정보는 Amazon DocumentDB에서 백업 및 복원 섹션을 참조하세

요. Amazon DocumentDB 백업 스토리지에 대한 요금 정보는 Amazon DocumentDB 요금 웹 페이지를 참조하세요.

비용을 제어하기 위해 연속 백업과 수동 스냅샷(보존 기간 이후에도 계속 유지됨)에 사용되는 스토리지의 양을 모니터링할 수 있습니다. 그런 다음 백업 보존 간격을 줄이거나 더 이상 필요하지 않은 경우 수동 스냅샷을 제거할 수 있습니다.

다음과 같이 Amazon CloudWatch 지표인 `TotalBackupStorageBilled`, `SnapshotStorageUsed` 및 `BackupRetentionPeriodStorageUsed`를 사용하여 Amazon DocumentDB 백업에 사용되는 스토리지의 양을 검토 및 모니터링할 수 있습니다.

- `BackupRetentionPeriodStorageUsed`는 현재 연속 백업을 저장하는 데 사용되는 백업 스토리지 양을 나타냅니다. 이 지표 값은 보존 기간 중에 발생하는 변경의 양과 클러스터 볼륨의 크기에 따라 달라집니다. 하지만, 결제 목적상, 이 지표는 보존 기간 동안 누적 클러스터 볼륨 크기를 초과하지 않습니다. 예를 들어 클러스터 크기가 100GiB이고 보존 기간이 2일이면 `BackRetentionPeriodStorageUsed`에 대한 최대 값은 200GiB(100GiB+100GiB)입니다.

- `SnapshotStorageUsed`는 백업 보존 기간 이후에 수동 스냅샷을 저장하는 데 사용되는 백업 스토리지의 양을 나타냅니다. 보존 기간 내에서 생성된 수동 스냅샷은 백업 스토리지 계산에 포함되지 않습니다. 마찬가지로, 자동 스냅샷도 백업 스토리지 계산에 포함되지 않습니다. 각 스냅샷의 크기는 스냅샷을 생성할 당시의 클러스터 볼륨 크기입니다. `SnapshotStorageUsed` 값은 유지하는 스냅샷 수와 각 스냅샷의 크기에 따라 달라집니다. 예를 들어 보존 기간 이후에 계속 유지되는 스냅샷이 하나 있고 해당 스냅샷을 생성할 당시의 클러스터 볼륨 크기가 100GiB라고 가정해 보겠습니다. `SnapshotStorageUsed`의 양은 100GiB입니다.

- `TotalBackupStorageBilled`는 `BackupRetentionPeriodStorageUsed` 및 `SnapshotStorageUsed`의 합계 - 당일의 클러스터 볼륨 크기에 해당하는 사용 가능한 백업 스토리지의 양을 나타냅니다. 예를 들어 클러스터 크기가 100GiB이고 보존 기간이 하루이며 해당 보존 기간 이후에도 계속 유지되는 스냅샷이 한 개 있는 경우, `TotalBackupStorageBilled`는 100GiB(100GiB+100GiB-100GiB)입니다.

- 이러한 지표는 각 Amazon DocumentDB 클러스터에 대해 독립적으로 계산됩니다.

CloudWatch 콘솔을 통해 CloudWatch 지표를 사용하여 Amazon DocumentDB 클러스터를 모니터링하고 보고서를 작성할 수 있습니다. CloudWatch 지표를 사용하는 방법에 대한 자세한 내용은 Amazon DocumentDB에서 모니터링 및 로깅 섹션을 참조하세요.

데이터 덤프, 복원, 가져오기 및 내보내기

`mongodump`, `mongorestore`, `mongoexport` 및 `mongoimport` 유틸리티를 사용하여 Amazon DocumentDB 클러스터 내부 및 외부로 데이터를 이동할 수 있습니다. 이 섹션에서는 이러한 각 도구의 목적과 성능을 높일 수 있는 구성에 대해 설명합니다.

> **MongoDB 데이터베이스 도구 버전 요구 사항**
>
> Amazon DocumentDB용 버전 100.11.0까지 MongoDB 데이터베이스 도구를 사용합니다. Amazon DocumentDB 다운로드하려면 MongoDB 웹 사이트의 MongoDB 데이터베이스 도구 릴리스를 참조하세요. MongoDB

주제

- mongodump
- mongorestore
- mongoexport
- mongoimport
- 자습서

mongodump

`mongodump` 유틸리티는 MongoDB 데이터베이스의 이진(BSON) 백업을 생성합니다. `mongodump` 도구는 데이터를 이진 형식으로 저장하여 실현되는 크기 효율성 때문에 Amazon DocumentDB 클러스터로 복원하려는 경우 소스 MongoDB 배포에서 데이터를 덤프하는 권장 방법입니다.

명령을 수행하는 데 사용하는 인스턴스 또는 머신에서 사용할 수 있는 리소스에 따라 `--numParallelCollections` 옵션을 사용하여 덤프되는 병렬 연결 수를 기본 1개에서 더 늘려 `mongodump` 속도를 높일 수 있습니다. 일반적으로 Amazon DocumentDB 클러스터의 기본 인스턴스에서 vCPU당 하나의 작업자로 시작하는 것이 좋습니다.

사용 예시

다음은 Amazon DocumentDB cluster 클러스터 `sample-cluster`에서 mongodump 유틸리티를 사용하는 예입니다.

```
mongodump --ssl \
    --host="sample-cluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --collection=sample-collection \
    --db=sample-database \
    --out=sample-output-file \
    --numParallelCollections 4  \
    --username=sample-user \
    --password=abc0123 \
    --sslCAFile global-bundle.pem
```

mongorestore

mongorestore 유틸리티를 사용하면 mongodump 유틸리티로 생성한 데이터베이스의 이진(BSON) 백업을 복원할 수 있습니다. 복원 시 `--numInsertionWorkersPerCollection` 옵션을 사용해 각 컬렉션의 작업자 수(기본값 1)를 늘려 복원 성능을 개선할 수 있습니다. 일반적으로 Amazon DocumentDB 클러스터의 기본 인스턴스에서 vCPU당 하나의 작업자로 시작하는 것이 좋습니다.

사용 예시

다음은 Amazon DocumentDB cluster 클러스터 `sample-cluster`에서 mongorestore 유틸리티를 사용하는 예입니다.

```
mongorestore --ssl \
    --host="sample-cluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --username=sample-user \
    --password=abc0123 \
    --sslCAFile global-bundle.pem <fileToBeRestored>
```

mongoexport

mongoexport 도구는 Amazon DocumentDB의 데이터를 JSON, CSV 또는 TSV 파일 형식으로 내보냅니다. mongoexport 도구는 사람 또는 기계가 읽어야 하는 데이터를 내보내는 권장 방법입니다.

> **Note**
>
> mongoexport는 병렬 내보내기를 직접 지원하지 않습니다. 그러나 다양한 컬렉션에 대해 여러 mongoexport 작업을 동시에 실행하여 성능을 높일 수 있습니다.

사용 예시

다음은 Amazon DocumentDB cluster 클러스터 sample-cluster에서 mongoexport 도구를 사용하는 예입니다.

```
mongoexport --ssl \
    --host="sample-cluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --collection=sample-collection \
    --db=sample-database \
    --out=sample-output-file \
    --username=sample-user \
    --password=abc0123 \
    --sslCAFile global-bundle.pem
```

mongoimport

mongoimport 도구는 JSON, CSV 또는 TSV 파일의 내용을 Amazon DocumentDB 클러스터로 가져옵니다. --numInsertionWorkers 파라미터를 사용하여 가져오기 작업을 병렬화하고 속도를 높일 수 있습니다(기본값은 1).

사용 예시

다음은 Amazon DocumentDB cluster 클러스터 sample-cluster에서 mongoimport 도구를 사용하는 예입니다.

```
mongoimport --ssl \
    --host="sample-cluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --collection=sample-collection \
    --db=sample-database \
    --file=<yourFile> \
    --numInsertionWorkers 4 \
    --username=sample-user \
    --password=abc0123 \
    --sslCAFile global-bundle.pem
```

자습서

다음 자습서에서는 mongodump, mongorestore, mongoexport 및 mongoimport 유틸리티를 사용하여 Amazon DocumentDB 클러스터 내/외부로 데이터를 이동하는 방법에 대해 설명합니다.

1. 사전 조건 — 시작하기 전에 Amazon DocumentDB 클러스터가 프로비저닝되고 클러스터와 동일한 VPC의 Amazon EC2 인스턴스에 액세스할 수 있는지 확인하세요. 자세한 내용은 Amazon EC2를 사용하여 연결 단원을 참조하십시오.

   mongo 유틸리티 도구를 사용하려면 다음과 같이 mongodb-org-tools 패키지가 EC2 인스턴스에 설치되어 있어야 합니다.

   ```
   sudo yum install mongodb-org-tools-4.0.18
   ```

   Amazon DocumentDB는 기본적으로 전송 계층 보안(TLS) 암호화를 사용하므로 mongo 쉘을 사용하여 연결하려면 다음과 같이 Amazon RDS 인증 기관(CA) 파일도 다운로드해야 합니다.

   ```
   wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
   ```

2. 샘플 데이터 다운로드 — 이 자습서에서는 식당에 대한 정보가 포함된 샘플 데이터를 다운로드합니다.

   ```
   wget https://raw.githubusercontent.com/ozlerhakan/mongodb-json-files/master/
   datasets/restaurant.json
   ```

3. 샘플 데이터를 Amazon DocumentDB로 가져오기 — 데이터가 논리적 JSON 형식이므로 mongoimport유틸리티를 사용하여 데이터를 Amazon DocumentDB 클러스터로 가져옵니다.

   ```
   mongoimport --ssl \
       --host="tutorialCluster.amazonaws.com:27017" \
       --collection=restaurants \
       --db=business \
       --file=restaurant.json \
       --numInsertionWorkers 4 \
       --username=<yourUsername> \
       --password=<yourPassword> \
       --sslCAFile global-bundle.pem
   ```

4. **mongodump**로 데이터 덤프 — 이제 Amazon DocumentDB 클러스터에 데이터가 있으므로 mongodump 유틸리티를 사용하여 해당 데이터의 이진 덤프를 가져올 수 있습니다.

```
mongodump --ssl \
    --host="tutorialCluster.us-east-1.docdb.amazonaws.com:27017"\
    --collection=restaurants \
    --db=business \
    --out=restaurantDump.bson \
    --numParallelCollections 4 \
    --username=<yourUsername> \
    --password=<yourPassword> \
    --sslCAFile global-bundle.pem
```

5. **restaurants** 컬렉션 삭제 — business 데이터베이스에서 restaurants 컬렉션을 복원하기 전에 먼저 다음과 같이 해당 데이터베이스에 이미 있는 컬렉션을 삭제해야 합니다.

```
use business
```

```
db.restaurants.drop()
```

6. **mongorestore**로 데이터 복원 — 3단계에서 데이터의 이진 덤프가 생성되었으므로 이제 mongorestore 유틸리티를 사용하여 데이터를 Amazon DocumentDB 클러스터로 복원할 수 있습니다.

```
mongorestore --ssl \
    --host="tutorialCluster.us-east-1.docdb.amazonaws.com:27017" \
    --numParallelCollections 4 \
    --username=<yourUsername> \
    --password=<yourPassword> \
    --sslCAFile global-bundle.pem restaurantDump.bson
```

7. **mongoexport**를 사용하여 데이터 내보내기 — 1단계에서 가져온 파일과 같은 JSON 파일 형식으로 클러스터에서 데이터를 내보내어 자습서를 완료합니다.

```
mongoexport --ssl \
    --host="tutorialCluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --collection=restaurants \
    --db=business \
    --out=restaurant2.json \
    --username=<yourUsername> \
    --password=<yourPassword> \
    --sslCAFile global-bundle.pem
```

8. 검증 — 다음 명령을 사용하여 5단계의 출력이 1단계와 동일한 결과를 내는지 검증할 수 있습니다.

```
wc -l restaurant.json
```

이 명령의 출력:

```
2548 restaurant.json
```

```
wc -l restaurant2.json
```

이 명령의 출력:

```
2548 restaurant2.json
```

클러스터 스냅샷 고려 사항

Amazon DocumentDB는 클러스터 백업 기간 동안 매일 클러스터의 자동 스냅샷을 생성합니다. Amazon DocumentDB는 사용자가 지정한 백업 보존 기간에 따라 클러스터의 자동 스냅샷을 저장합니다. 필요할 경우 백업 보존 기간 중 어느 특정 시점으로든 클러스터를 복구할 수 있습니다. 동일한 리전에서 동일 클러스터에 대한 복사 작업을 수행하고 있는 동안에는 자동 스냅샷이 생성되지 않습니다.

주제

- 백업 스토리지
- 백업 기간
- 백업 보관 기간
- 클러스터 스냅샷 암호화 복사

자동 클러스터 스냅샷 이외에 클러스터 스냅샷을 수동으로 생성할 수도 있습니다. 자동 스냅샷과 수동 스냅샷을 모두 복사할 수 있습니다. 자세한 내용은 수동 클러스터 스냅샷 생성 및 Amazon DocumentDB 클러스터 스냅샷 복사 섹션을 참조하세요.

> **Note**
>
> 자동 스냅샷을 생성하려면 클러스터가 사용 가능한 상태에 있어야 합니다.

Amazon DocumentDB 자동화된 클러스터 스냅샷은 공유할 수 없습니다. 이 문제를 해결하려면 자동화된 스냅샷을 복사하여 수동 스냅샷을 만든 다음 그 복사본을 공유하면 됩니다. 스냅샷 복사에 대한 자세한 내용은 Amazon DocumentDB 클러스터 스냅샷 복사 섹션을 참조하세요. 스냅샷에서 클러스터 복원에 대한 자세한 내용은 클러스터 스냅샷에서 복원 섹션을 참조하세요.

백업 스토리지

각의 Amazon DocumentDB 백업 스토리지 AWS 리전 는 백업 보존 기간에 필요한 백업 스토리지로 구성되며, 여기에는 해당 리전의 자동 및 수동 클러스터 스냅샷이 포함됩니다. 기본 백업 보존 기간은 1일입니다. 백업 스토리지에 대한 요금 정보는 Amazon DocumentDB 요금 웹 페이지를 참조하세요.

클러스터를 삭제하면 클러스터의 모든 자동 스냅샷이 삭제되고 복구할 수 없습니다. 그러나 클러스터를 삭제할 때 수동 스냅샷은 삭제되지 않습니다. 클러스터를 삭제하기 전에 Amazon DocumentDB에서 최종 스냅샷(수동 스냅샷)이 생성되도록 선택한 경우, 그 최종 스냅샷을 사용하여 클러스터를 복구할 수 있습니다.

스냅샷 및 스토리지에 대한 자세한 내용은 백업 스토리지 사용량 파악 섹션을 참조하세요.

백업 기간

자동 스냅샷은 기본 백업 기간 동안 매일 생성됩니다. 스냅샷 시간이 백업 기간에 할당된 시간보다 오래 걸릴 경우 백업 프로세스는 백업 기간이 종료한 후에도 완료 시까지 계속 실행됩니다. 백업 기간은 해당 클러스터에 대한 주간 유지 보수 기간과 겹칠 수 없습니다.

클러스터를 생성할 때 원하는 백업 기간을 지정하지 않으면 Amazon DocumentDB이 기본 30분 백업 기간을 할당합니다. 클러스터의 리전과 연결된 8시간 블록 시간 중에서 이 시간이 임의로 선택됩니다. 클러스터를 수정하여 기본 백업 기간을 변경할 수 있습니다. 자세한 내용은 Amazon DocumentDB 클러스터 수정 단원을 참조하십시오.

리전 이름	리전	UTC 시간 블록
미국 동부(오하이오)	us-east-2	03:00~11:00
미국 동부(버지니아 북부)	us-east-1	03:00~11:00
미국 서부(오리건)	us-west-2	06:00~14:00

리전 이름	리전	UTC 시간 블록
Africa (Cape Town)	af-south-1	03:00~11:00
Asia Pacific (Hong Kong)	ap-east-1	06:00~14:00
아시아 태평양(하이데라바드)	ap-south-2	06:30-14:30
아시아 태평양(말레이시아)	ap-southeast-5	13:00~21:00
아시아 태평양(뭄바이)	ap-south-1	06:00~14:00
아시아 태평양(오사카)	ap-northeast-3	12:00–20:00
아시아 태평양(서울)	ap-northeast-2	13:00~21:00
아시아 태평양(싱가포르)	ap-southeast-1	14:00–22:00
아시아 태평양(시드니)	ap-southeast-2	12:00–20:00
아시아 태평양(자카르타)	ap-southeast-3	08:00~16:00
아시아 태평양(멜버른)	ap-southeast-4	11:00~19:00
아시아 태평양(태국)	ap-southeast-7	15:00~23:00
아시아 태평양(도쿄)	ap-northeast-1	13:00~21:00
캐나다(중부)	ca-central-1	03:00~11:00
캐나다 서부(캘거리)	ca-west-1	18:00-02:00
중국(베이징)	cn-north-1	06:00~14:00
중국(닝샤)	cn-northwest-1	06:00~14:00
유럽(프랑크푸르트)	eu-central-1	21:00~05:00
유럽(취리히)	eu-central-2	02:00~10:00
유럽(아일랜드)	eu-west-1	22:00~06:00

리전 이름	리전	UTC 시간 블록
Europe (London)	eu-west-2	22:00~06:00
Europe (Milan)	eu-south-1	02:00~10:00
유럽(파리)	eu-west-3	23:59~07:29
유럽(스페인)	eu-south-2	02:00~10:00
유럽(스톡홀름)	eu-north-1	04:00~12:00
멕시코(중부)	mx-central-1	03:00~11:00
중동(UAE)	me-central-1	05:00~13:00
남아메리카(상파울루)	sa-east-1	00:00~08:00
이스라엘(텔아비브)	il-central-1	04:00-12:00
AWS GovCloud(미국 동부)	us-gov-east-1	17:00–01:00
AWS GovCloud(미국 서부)	us-gov-west-1	06:00~14:00

백업 보관 기간

백업 보존 기간은 자동 백업이 자동으로 삭제되기 이전에 보관되는 일 수입니다. Amazon DocumentDB는 1~35일의 백업 보존 기간을 지원합니다.

클러스터를 생성하면서 백업 보존 기간을 설정할 수 있습니다. 백업 보존 기간을 명시적으로 설정하지 않은 경우 기본 백업 보존 기간 1일이 클러스터에 할당됩니다. 클러스터를 생성한 후 AWS Management Console 또는를 사용하여 클러스터를 수정하여 백업 보존 기간을 수정할 수 있습니다 AWS CLI. 자세한 내용은 Amazon DocumentDB 클러스터 수정 단원을 참조하십시오.

클러스터 스냅샷 암호화 복사

클러스터 및 스냅샷 암호화는 KMS 암호화 키를 기반으로 합니다. KMS 키 ID는 Amazon 리소스 이름(ARN), KMS 키 식별자 또는 KMS 암호화 키에 대한 KMS 키 별칭입니다.

다음과 같은 지침과 제한 사항이 적용됩니다.

- 스냅샷을 생성할 때 클러스터에서 암호화가 추론됩니다. 클러스터가 암호화된 경우 클러스터의 스냅샷은 동일한 KMS 키로 암호화됩니다. 클러스터가 암호화되지 않으면 스냅샷은 암호화되지 않습니다.
- Amazon Web Services 계정에서 암호화된 클러스터 스냅샷을 복사하는 경우, `KmsKeyId` 값을 지정하여 새 KMS 암호화 키로 사본을 암호화할 수 있습니다. `KmsKeyId` 값을 지정하지 않으면 DB 클러스터 스냅샷의 사본을 원본 DB 스냅샷과 동일한 KMS 키로 암호화합니다.
- 다른 Amazon Web Services 계정에서 공유한 암호화된 DB 클러스터 스냅샷을 복사하는 경우, `KmsKeyId` 값을 지정해야 합니다.
- 암호화된 클러스터 스냅샷을 다른 Amazon Web Services 지역에 `KmsKeyId` 복사하려면 대상 지역의 클러스터 스냅샷 사본을 암호화하는 데 사용할 KMS 키 ID를 설정합니다. KMS 암호화 키는 해당 키를 만든 Amazon Web Services 리전에 고유하며, 한 Amazon Web Services 리전의 암호화 키를 다른 Amazon Web Services 리전에서 사용할 수는 없습니다.
- 암호화되지 않은 클러스터 스냅샷의 복사하고 `KmsKeyId` 파라미터에 대한 값을 지정하려고 시도하면 오류가 반환됩니다.

자동 스냅샷과 수동 스냅샷 비교

다음은 Amazon DocumentDB(MongoDB 호환) 자동 스냅샷 및 수동 스냅샷의 주요 특성입니다.

Amazon DocumentDB 자동 스냅샷의 주요 특성은 다음과 같습니다.

- 자동 스냅샷 이름 지정 — 자동 스냅샷 이름은 `rds:<cluster-name>-yyyy-mm-dd-hh-mm` 패턴을 따릅니다. 여기서 `yyyy-mm-dd-hh-mm`은 스냅샷이 생성된 날짜 및 시간을 나타냅니다.
- 일정에 따라 자동으로 생성됨 — 클러스터를 생성하거나 수정할 때 백업 보존 기간을 1~35일 사이의 정수 값으로 설정할 수 있습니다. 기본적으로 새 클러스터의 백업 보존 기간은 1일입니다. 백업 보존 기간은 자동 스냅샷이 자동으로 삭제되기 이전에 보관되는 일 수를 정의합니다. Amazon DocumentDB 클러스터에 대한 자동 백업을 비활성화할 수 없습니다.

  백업 보존 기간 설정 이외에 자동 스냅샷이 생성되는 기간(일)을 나타내는 백업 기간을 설정할 수도 있습니다.
- 자동 스냅샷 삭제 — 자동 스냅샷 클러스터를 삭제하면 자동 스냅샷이 삭제됩니다. 자동 스냅샷을 수동으로 삭제할 수 없습니다.
- 증분식 — 백업 보존 기간 동안 데이터베이스 업데이트가 증분식 변경 기록 방식으로 기록됩니다.
- 자동 스냅샷에서 복원 — AWS Management Console 또는 AWS CLI를 사용하여 자동 스냅샷에서 복원할 수 있습니다. 를 사용하여 스냅샷에서 복원 AWS CLI하는 경우 클러스터를 사용할 수 있게 된 후 인스턴스를 별도로 추가해야 합니다.

- 공유 — Amazon DocumentDB의 자동 클러스터 스냅샷은 공유할 수 없습니다. 이 문제를 해결하려면 자동화된 스냅샷을 복사하여 수동 스냅샷을 만든 다음 그 복사본을 공유하면 됩니다. 스냅샷 복사에 대한 자세한 내용은 Amazon DocumentDB 클러스터 스냅샷 복사 섹션을 참조하세요. 스냅샷에서 클러스터 복원에 대한 자세한 내용은 클러스터 스냅샷에서 복원 섹션을 참조하세요.
- 백업 보존 기간 내의 어떤 시점으로든 복원 가능 — 데이터베이스 업데이트가 증분식으로 기록되므로 백업 보존 기간 내의 어떤 시점으로든 복원할 수 있습니다.

  를 사용하여 자동 스냅샷 또는 point-in-time으로 복원에서 복원 AWS CLI하는 경우 클러스터를 사용할 수 있게 된 후 인스턴스를 별도로 추가해야 합니다.

Amazon DocumentDB 수동 스냅샷의 주요 특성은 다음과 같습니다.

- 온디맨드 생성 - Amazon DocumentDB 수동 스냅샷은 Amazon DocumentDB 관리 콘솔 또는를 사용하여 온디맨드로 생성됩니다 AWS CLI.
- 수동 스냅샷 삭제 — 수동 Amazon DocumentDB 스냅샷은 콘솔 또는 AWS CLI를 사용하여 명시적으로 삭제한 경우에만 삭제됩니다. 클러스터를 삭제해도 수동 스냅샷은 삭제되지 않습니다.
- 전체 백업 — 수동 스냅샷을 생성하면 클러스터 데이터의 전체 백업이 생성된 후 저장됩니다.
- 수동 스냅샷 이름 지정 — 수동 스냅샷 이름을 지정합니다. Amazon DocumentDB는 이름에 `datetime` 스탬프를 추가하지 않으므로 이름에 포함시키려면 해당 정보를 추가해야 합니다.
- 수동 스냅샷에서 복원 — 콘솔 또는 AWS CLI를 사용하여 수동 스냅샷에서 복원할 수 있습니다. 를 사용하여 스냅샷에서 복원 AWS CLI하는 경우 클러스터를 사용할 수 있게 된 후 인스턴스를 별도로 추가해야 합니다.
- Service Quotas - 당 최대 100개의 수동 스냅샷으로 제한됩니다 AWS 리전.
- 공유 — 수동 클러스터 스냅샷을 공유할 수 있고, 권한 있는 AWS 계정에서 이를 복사할 수 있습니다. 암호화되거나 암호화되지 않은 수동 스냅샷을 공유할 수 있습니다. 스냅샷 복사에 대한 자세한 내용은 Amazon DocumentDB 클러스터 스냅샷 복사 섹션을 참조하세요.
- 수동 스냅샷 생성 시점으로 복원 — 수동 스냅샷에서 복원하면 수동 스냅샷이 생성된 시점으로 복원됩니다.

  를 사용하여 스냅샷에서 복원 AWS CLI하는 경우 클러스터를 사용할 수 있게 된 후 인스턴스를 별도로 추가해야 합니다.

수동 클러스터 스냅샷 생성

AWS Management Console 또는를 사용하여 수동 스냅샷을 생성할 수 있습니다 AWS CLI. 스냅샷을 생성하는 데 걸리는 시간은 데이터베이스 크기에 따라 다릅니다. 스냅샷을 생성하는 경우 다음을 수행해야 합니다.

1. 백업할 클러스터를 식별합니다.
2. 스냅샷의 이름을 지정합니다. 나중에 이 스냅샷에서 복원할 수 있습니다.

Using the AWS Management Console

를 사용하여 수동 스냅샷을 생성하려면 아래 방법 중 하나를 따를 AWS Management Console수 있습니다.

1. 방법 1:

   1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
   2. 탐색 창에서 스냅샷를 선택합니다.

      > **Tip**
      >
      > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택하세요.

   3. 스냅샷 페이지에서 생성을 선택합니다.
   4. 클러스터 스냅샷 생성 페이지에서:

      a. 클러스터 식별자 — 클러스터의 드롭다운 목록에서 스냅샷을 생성할 클러스터를 선택합니다.

      b. 스냅샷 식별자 — 스냅샷의 이름을 입력합니다.

      스냅샷 명명 제약 조건:

      - 길이는 [1~255]글자, 숫자 또는 하이픈입니다.
      - 첫 번째 문자는 글자이어야 합니다.
      - 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.

- AWS 계정, 리전별로 모든 클러스터 (Amazon RDS, Amazon Neptune 및 Amazon DocumentDB)에 대해 고유해야 합니다.

c. 생성(Create)을 선택합니다.

2. 방법 2:

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

3. 클러스터 페이지에서 스냅샷을 생성하려는 클러스터의 왼쪽에 있는 버튼을 선택합니다.
4. 작업 메뉴에서 스냅샷 만들기을 선택합니다.
5. 클러스터 스냅샷 생성 페이지에서:

a. 스냅샷 식별자 — 스냅샷의 이름을 입력합니다.

스냅샷 명명 제약 조건:

- 길이는 [1~63] 글자, 숫자 또는 하이픈입니다.
- 첫 번째 문자는 글자이어야 합니다.
- 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.
- AWS 계정, 리전별로 모든 클러스터 (Amazon RDS, Amazon Neptune 및 Amazon DocumentDB)에 대해 고유해야 합니다.

b. 생성(Create)을 선택합니다.

Using the AWS CLI

를 사용하여 클러스터 스냅샷을 생성하려면 다음 파라미터와 함께 `create-db-cluster-snapshot` 작업을 AWS CLI사용합니다.

파라미터

- **--db-cluster-identifier** - 필수입니다. 스냅샷을 생성할 클러스터의 이름입니다. 이 클러스터가 있고 사용 가능해야 합니다.
- **--db-cluster-snapshot-identifier** - 필수입니다. 생성하려는 수동 스냅샷의 이름입니다.

다음 예제에서는 sample-cluster라는 클러스터에 대해 sample-cluster-snapshot이라는 스냅샷을 만듭니다.

Linux, macOS, Unix의 경우:

```
aws docdb create-db-cluster-snapshot \
    --db-cluster-identifier sample-cluster \
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

Windows의 경우:

```
aws docdb create-db-cluster-snapshot ^
    --db-cluster-identifier sample-cluster ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBClusterSnapshot": {
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1b",
            "us-east-1c"
        ],
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot",
        "DBClusterIdentifier": "sample-cluster",
        "SnapshotCreateTime": "2020-04-24T04:59:08.475Z",
        "Engine": "docdb",
        "Status": "creating",
        "Port": 0,
        "VpcId": "vpc-abc0123",
        "ClusterCreateTime": "2020-01-10T22:13:38.261Z",
        "MasterUsername": "master-user",
        "EngineVersion": "4.0.0",
```

```
        "SnapshotType": "manual",
        "PercentProgress": 0,
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DBClusterSnapshotArn": "arn:aws:rds:us-east-1:<accountID>:cluster-
snapshot:sample-cluster-snapshot"
    }
}
```

Amazon DocumentDB 클러스터 스냅샷 복사

Amazon DocumentDB에서는 동일한 AWS 리전 또는 다른에 스냅샷을 복사할 수 있습니다 AWS 리전. 공유 스냅샷을 동일한 AWS 리전 또는 다른의 계정에 복사할 수도 있습니다 AWS 리전. 스냅샷 공유에 대한 자세한 내용은 Amazon DocumentDB 클러스터 스냅샷 공유 단원을 참조하세요.

> **Note**
>
> Amazon DocumentDB은 유지하는 백업 및 스냅샷 데이터의 양과 유지 기간에 따라 요금을 청구합니다. Amazon DocumentDB 백업 및 스냅샷과 연결된 스토리지에 대한 자세한 내용은 백업 스토리지 사용량 파악 섹션을 참조하세요. Amazon DocumentDB 스토리지에 대한 요금 정보는 Amazon DocumentDB 요금 섹션을 참조하세요.

주제

- 공유 스냅샷 복사
- 에서 스냅샷 복사 AWS 리전
- 제한 사항
- 암호화 처리
- 파라미터 그룹 고려 사항
- 클러스터 스냅샷 복사

공유 스냅샷 복사

다른 AWS 계정에서 공유한 스냅샷을 복사할 수 있습니다. 다른 AWS 계정에서 공유된 암호화된 스냅샷을 복사하는 경우 스냅샷을 암호화하는 데 사용된 AWS KMS 암호화 키에 액세스할 수 있어야 합니다. 자세한 내용은 암호화 처리 단원을 참조하십시오.

에서 스냅샷 복사 AWS 리전

소스 스냅샷의와 AWS 리전 다른에 스냅샷을 복사하는 경우 AWS 리전각 복사본은 전체 스냅샷입니다. 전체 스냅샷 복사에는 Amazon DocumentDB 클러스터 복원에 필요한 모든 데이터 및 메타데이터가 포함됩니다.

제한 사항

다음은 스냅샷을 복사할 때 적용되는 몇몇 제한 사항입니다.

- 대상 스냅샷이 사용 가능해지기 전에 원본 스냅샷을 삭제할 경우 스냅샷 복사가 실패할 수 있습니다. 원본 스냅샷을 삭제하기 전에 대상 스냅샷의 상태가 `AVAILABLE`인지 확인하세요.
- 계정당 단일 대상 리전으로 최대 5개의 스냅샷 복사 요청이 진행될 수 있습니다.
- AWS 리전 관련 및 복사할 데이터의 양에 따라 리전 간 스냅샷 복사를 완료하는 데 몇 시간이 걸릴 수 있습니다. 경우에 따라 지정된 소스 AWS 리전으로부터 대량의 교차 리전 스냅샷 복사 요청이 있을 수 있습니다. 이러한 경우 Amazon DocumentDB는 진행 중인 일부 복사본이 완료될 때까지 해당 소스의 새 리전 간 복사 요청을 대기열 AWS 리전 에 넣을 수 있습니다. 대기열에 있는 복사 요청에 대한 진행 정보는 표시되지 않습니다. 복사가 시작되면 진행 정보가 표시됩니다.

암호화 처리

AWS KMS 암호화 키를 사용하여 암호화된 스냅샷을 복사할 수 있습니다. 암호화된 스냅샷을 복사할 경우 스냅샷의 사본도 암호화해야 합니다. 동일한 내에서 암호화된 스냅샷을 복사 AWS 리전하는 경우 원본 스냅샷과 동일한 암호화 키로 복사본을 AWS KMS 암호화하거나 다른 AWS KMS 암호화 키를 지정할 수 있습니다. 리전 간에 암호화된 스냅샷을 복사하는 경우 키는 리전별로 다르므로 원본 스냅샷에 사용된 것과 동일한 AWS KMS 암호화 AWS KMS 키를 복사에 사용할 수 없습니다. 대신 대상 AWS 리전 n에 유효한 AWS KMS 키를 지정해야 합니다.

소스 스냅샷은 복사 프로세스 전체에서 암호화를 유지합니다. 자세한 내용은 Amazon DocumentDB의 데이터 보호 단원을 참조하십시오.

> **Note**
>
> Amazon DocumentDB 클러스터 스냅샷의 경우 암호화되지 않은 DB 클러스터 스냅샷은 스냅샷을 복사할 때 암호화할 수 없습니다.

파라미터 그룹 고려 사항

리전 간에 스냅샷을 복사하는 경우 복사에는 원래 Amazon DocumentDB 클러스터에서 사용된 파라미터 그룹이 포함되지 않습니다. 스냅샷을 복원하여 새 클러스터를 생성하면 해당 클러스터 AWS 리전는 생성된에 대한 기본 파라미터 그룹을 가져옵니다. 새 클러스터에 원본과 같은 파라미터를 지정하려면 다음을 수행해야 합니다.

1. 대상에서 원래 클러스터와 동일한 설정으로 Amazon DocumentDB 클러스터 파라미터 그룹을 AWS 리전생성합니다. Amazon DocumentDB 새에 이미 있는 경우 해당를 사용할 AWS 리전수 있습니다.
2. 대상에서 스냅샷을 복원한 후 새 Amazon DocumentDB 클러스터를 AWS 리전수정하고 이전 단계에서 새 파라미터 그룹 또는 기존 파라미터 그룹을 추가합니다. 자세한 내용은 Amazon DocumentDB 클러스터 수정 단원을 참조하십시오.

클러스터 스냅샷 복사

다음과 AWS CLI같이 AWS Management Console 또는를 사용하여 Amazon DocumentDB 클러스터를 복사할 수 있습니다.

Using the AWS Management Console

를 사용하여 클러스터 스냅샷의 복사본을 만들려면 다음 단계를 AWS Management Console완료합니다. 이 절차는 동일하거나 리전 간에 암호화된 클러스터 스냅샷 AWS 리전 또는 암호화되지 않은 클러스터 스냅샷을 복사하는 데 사용됩니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 스냅샷을 선택한 다음 복사하려는 스냅샷 왼쪽에 있는 버튼을 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택합니다.

3. 작업 메뉴에서 복사를 선택합니다.
4. 표시되는 클러스터 스냅샷 복사본 만들기 페이지에서 설정 섹션을 완료하세요.

a. 대상 지역 — 선택 사항. 클러스터 스냅샷을 다른에 복사하려면 대상 리전 AWS 리전 에서 해당 스냅샷을 AWS 리전선택합니다.

b. 새 스냅샷 식별자 — 새 스냅샷의 이름을 입력합니다.

대상 스냅샷 명명 제약 조건:

- 기본 스냅샷의 이름일 수 없습니다.
- 길이는 [1~63]글자, 숫자 또는 하이픈입니다.
- 첫 번째 문자는 글자이어야 합니다.
- 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.
- 리전별로 Amazon RDS, Neptune 및 Amazon DocumentDB의 모든 클러스터 AWS 계정에 대해 고유해야 합니다.

c. 태그 복사 — 소스 스냅샷에 있는 태그를 스냅샷 복사본으로 복사하려면 태그 복사를 선택합니다.

5. 유휴 상태에서 암호화 섹션을 완료합니다.

a. 유휴 상태에서 암호화 — 암호화된 스냅샷에서 암호화되지 않은 복사본을 생성할 수 없으므로 스냅샷이 암호화되면 이 옵션을 사용할 수 없습니다. 스냅샷이 암호화된 경우 저장 시 암호화 중에 AWS KMS key 사용되는를 변경할 수 있습니다.

암호화된 스냅샷 복사에 대한 자세한 내용은 클러스터 스냅샷 암호화 복사 섹션을 참조하세요.

저장된 암호화에 대한 자세한 내용은 Amazon DocumentDB 저장 데이터 암호화 섹션을 참조하세요.

b. AWS KMS 키 - 드롭다운 목록에서 다음 중 하나를 선택합니다.

- (기본값) aws/rds - 계정 번호와 AWS KMS 키 ID가이 옵션 다음에 나열됩니다.
- <some-key-name> — 키를 생성하면 해당 키가 나열되고 이를 선택할 수 있습니다.
- 키 ARN 입력 — ARN 상자에 AWS KMS 키의 Amazon 리소스 이름(ARN)을 입력합니다. ARN 형식은 `arn:aws:kms:<region>:<accountID>:key/<key-id>` 입니다.

6. 선택한 스냅샷을 복사하려면 스냅샷 복사를 선택합니다. 선택한 스냅샷을 복사하지 않으려면 취소를 선택합니다.

Using the AWS CLI

클러스터 스냅샷을 복사하려면 AWS CLI다음 파라미터와 함께 `copy-db-cluster-snapshot` 작업을 사용합니다. 스냅샷을 다른에 복사하는 경우 스냅샷을 복사 AWS 리전 할에서 명령을 AWS 리전실행합니다.

- **`--source-db-cluster-snapshot-identifier`** — 필수입니다. 복사할 클러스터 스냅샷의 식별자입니다. 클러스터 스냅샷이 있어야 하며 사용 가능한 상태여야 합니다. 스냅샷을 다른에 복사 AWS 리전 하거나 공유 클러스터 스냅샷을 복사하는 경우이 식별자는 소스 클러스터 스냅샷의 ARN 형식이어야 합니다. 이 파라미터는 대소문자를 구분하지 않습니다.
- **`--target-db-cluster-snapshot-identifier`** — 필수입니다. 소스 클러스터 스냅샷에서 생성할 새 클러스터 스냅샷의 식별자입니다. 이 파라미터는 대소문자를 구분하지 않습니다.

  대상 스냅샷 명명 제약 조건:

  - 기본 스냅샷의 이름일 수 없습니다.
  - 길이는 [1~63]글자, 숫자 또는 하이픈입니다.
  - 첫 번째 문자는 글자이어야 합니다.
  - 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.
  - 리전별로 Amazon RDS, Neptune 및 Amazon DocumentDB의 모든 클러스터 AWS 계정에 대해 고유해야 합니다.
- **`--source-region`** - 스냅샷을 다른에 복사하는 경우 암호화된 클러스터 스냅샷을 복사 AWS 리전 할를 AWS 리전지정합니다.

  스냅샷을 다른 AWS 리전 리전에 복사하려고 하며 `--source-region`을 지정하지 않은 경우, 대신 `pre-signed-url` 옵션을 지정해야 합니다. `pre-signed-url` 값은 AWS 리전 클러스터 스냅샷이 복사되는 소스에서 `CopyDBClusterSnapshot` 작업을 호출하기 위한 서명 버전 4 서명 요청이 포함된 URL이어야 합니다. `pre-signed-url`에 대한 자세한 내용은 CopyDbClusterSnapshot을 참조하세요.
- **`--kms-key-id`** — DB 클러스터 스냅샷의 복사본을 암호화하는 데 사용할 키의 KMS 키 식별자입니다.

  암호화된 클러스터 스냅샷을 다른에 복사하는 경우 AWS 리전이 파라미터가 필요합니다. 대상에 대한 KMS 키를 지정해야 합니다 AWS 리전.

  동일한에서 암호화된 클러스터 스냅샷을 복사하는 경우 AWS 리전 AWS KMS 키 파라미터는 선택 사항입니다. 클러스터 스냅샷의 사본은 소스 클러스터 스냅샷과 동일한 AWS KMS 키로 암호

화됩니다. 복사본 AWS KMS 을 암호화하는 데 사용할 새 암호화 키를 지정하려면이 파라미터를 사용하면 됩니다.

- **`--copy-tags`** - 선택 사항입니다. 복사할 태그 및 값입니다.

진행 중인 복사 작업을 취소하려면 대상 클러스터 스냅샷이 복사 상태에 있는 동안 `--target-db-cluster-snapshot-identifier` 또는 `TargetDBClusterSnapshotIdentifier`로 식별된 해당 DB 클러스터 스냅샷을 삭제합니다.

Example

예 1: 암호화되지 않은 스냅샷을 동일한 리전에 복사

다음 AWS CLI 예제에서는 소스 스냅샷 AWS 리전 과 동일한 `sample-cluster-snapshot-copy`에 `sample-cluster-snapshot` 라는 복사본을 생성합니다. 복사가 완료되면 원래 스냅샷의 모든 태그가 스냅샷 사본에 복사됩니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb copy-db-cluster-snapshot \
    --source-db-cluster-snapshot-identifier sample-cluster-snapshot \
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy \
    --copy-tags
```

Windows의 경우:

```
aws docdb copy-db-cluster-snapshot ^
    --source-db-cluster-snapshot-identifier sample-cluster-snapshot ^
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy ^
    --copy-tags
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBClusterSnapshot": {
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1b",
            "us-east-1c"
        ],
```

```
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot-copy",
        "DBClusterIdentifier": "sample-cluster",
        "SnapshotCreateTime": "2020-03-27T08:40:24.805Z",
        "Engine": "docdb",
        "Status": "copying",
        "Port": 0,
        "VpcId": "vpc-abcd0123",
        "ClusterCreateTime": "2020-01-10T22:13:38.261Z",
        "MasterUsername": "master-user",
        "EngineVersion": "4.0.0",
        "SnapshotType": "manual",
        "PercentProgress": 0,
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:111122223333:key/sample-key-id",
        "DBClusterSnapshotArn": "arn:aws:rds:us-east-1:111122223333:cluster-
snapshot:sample-cluster-snapshot-copy",
        "SourceDBClusterSnapshotArn": "arn:aws:rds:us-east-1:111122223333:cluster-
snapshot:sample-cluster-snapshot"
    }
}
```

Example

예제 2: 암호화되지 않은 스냅샷 복사 AWS 리전

다음 AWS CLI 예제에서는 ARN이 `sample-cluster-snapshot`있는의 복사본을 생성합니다`arn:aws:rds:us-east-1:123456789012:cluster-snapshot:sample-cluster-snapshot`. 이 복사본의 이름은 `sample-cluster-snapshot-copy` 이며 명령이 실행되는 AWS 리전 에 있습니다.

Linux, macOS, Unix의 경우:

```
aws docdb copy-db-cluster-snapshot \
    --source-db-cluster-snapshot-identifier arn:aws:rds:us-
east-1:123456789012:cluster-snapshot:sample-cluster-snapshot \
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy
```

Windows의 경우:

```
aws docdb copy-db-cluster-snapshot ^
    --source-db-cluster-snapshot-identifier arn:aws:rds:us-
east-1:123456789012:cluster-snapshot:sample-cluster-snapshot ^
```

```
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBClusterSnapshot": {
        "AvailabilityZones": [
            "us-east-1a",
            "us-east-1b",
            "us-east-1c"
        ],
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot-copy",
        "DBClusterIdentifier": "sample-cluster",
        "SnapshotCreateTime": "2020-04-29T16:45:51.239Z",
        "Engine": "docdb",
        "AllocatedStorage": 0,
        "Status": "copying",
        "Port": 0,
        "VpcId": "vpc-abc0123",
        "ClusterCreateTime": "2020-04-28T16:43:00.294Z",
        "MasterUsername": "master-user",
        "EngineVersion": "4.0.0",
        "LicenseModel": "docdb",
        "SnapshotType": "manual",
        "PercentProgress": 0,
        "StorageEncrypted": false,
        "DBClusterSnapshotArn": "arn:aws:rds:us-east-1:123456789012:cluster-
snapshot:sample-cluster-snapshot-copy",
        "SourceDBClusterSnapshotArn": "arn:aws:rds:us-east-1:123456789012:cluster-
snapshot:sample-cluster-snapshot",
    }
}
```

Example

예제 3:에 암호화된 스냅샷 복사 AWS 리전

다음 AWS CLI 예제에서는 us-west-2 리전`sample-cluster-snapshot`에서 us-east-1 리전으로의 사본을 생성합니다. 이 명령은 us-east-1 리전에서 호출됩니다.

Linux, macOS, Unix의 경우:

```
aws docdb copy-db-cluster-snapshot \
```

```
    --source-db-cluster-snapshot-identifier arn:aws:rds:us-
west-2:123456789012:cluster-snapshot:sample-cluster-snapshot \
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy \
    --source-region us-west-2 \
    --kms-key-id sample-us-east-1-key
```

Windows의 경우:

```
aws docdb copy-db-cluster-snapshot ^
    --source-db-cluster-snapshot-identifier arn:aws:rds:us-
west-2:123456789012:cluster-snapshot:sample-cluster-snapshot ^
    --target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy ^
    --source-region us-west-2 ^
    --kms-key-id sample-us-east-1-key
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBClusterSnapshot": {
        "AvailabilityZones": [],
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot-copy",
        "DBClusterIdentifier": "sample-cluster",
        "SnapshotCreateTime": "2020-04-29T16:45:53.159Z",
        "Engine": "docdb",
        "AllocatedStorage": 0,
        "Status": "copying",
        "Port": 0,
        "ClusterCreateTime": "2020-04-28T16:43:07.129Z",
        "MasterUsername": "chimera",
        "EngineVersion": "4.0.0",
        "LicenseModel": "docdb",
        "SnapshotType": "manual",
        "PercentProgress": 0,
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:123456789012:key/0846496a-
d48e-41c4-9353-86d8301d7e35",
        "DBClusterSnapshotArn": "arn:aws:rds:us-east-1:123456789012:cluster-
snapshot:sample-cluster-snapshot-copy",
        "SourceDBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:sample-cluster-snapshot",
    }
}
```

Example

예제 4: 암호화되지 않은 공유 스냅샷 복사 AWS 리전

다음 AWS CLI 예제인 계정 -는 계정에서 sample-cluster-snapshot 공유하는 암호화되지 않은 클러스터 스냅샷의 사본을 us-east-1 리전999999999999에서 us-west-2 리전으로 123456789012생성합니다. 이 명령은 us-west-2 리전에서 호출됩니다. 스냅샷 공유에 대한 자세한 내용은 스냅샷 공유 단원을 참조하세요.

Linux, macOS, Unix의 경우:

```
aws docdb copy-db-cluster-snapshot \
--region us-west-2 \
--source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:999999999999:cluster-
snapshot:sample-cluster-snapshot \
--target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy
```

Windows의 경우:

```
aws docdb copy-db-cluster-snapshot ^
--region us-west-2 ^
--source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:999999999999:cluster-
snapshot:sample-cluster-snapshot ^
--target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
 "DBClusterSnapshots": [
 {
 "AvailabilityZones": [],
 "DBClusterSnapshotIdentifier": "sample-cluster-snapshot-copy",
 "DBClusterIdentifier": "sample-cluster",
 "SnapshotCreateTime": "2025-08-22T11:27:00.497000+00:00",
 "Engine": "docdb",
 "Status": "copying",
 "Port": 0,
 "ClusterCreateTime": "2024-07-02T16:44:50.246000+00:00",
 "MasterUsername": "master-user",
 "EngineVersion": "5.0.0",
 "SnapshotType": "manual",
 "PercentProgress": 0,
```

```
 "StorageEncrypted": false,
 "DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:sample-cluster-snapshot-copy",
 "SourceDBClusterSnapshotArn": "arn:aws:rds:us-east-1:999999999999:cluster-
snapshot:sample-cluster-snapshot"
 }
 ]
}
```

Example

예제 5:에 암호화된 공유 스냅샷 복사 AWS 리전

다음 AWS CLI 예제에서는 계정 - us-east-1 리전999999999999에서 us-west-2 리전으로 계정에서 `sample-cluster-snapshot` 공유하는 암호화된 클러스터 스냅샷의 사본을 `123456789012` 생성합니다. 대상 스냅샷은 고객 관리형 KMS 키인 `arn:aws:kms:us-west-2:123456789012:key/6c1f3264-1797-472b-ba37-03011e682d28`로 암호화됩니다. 이 명령은 us-west-2 리전에서 호출됩니다. 스냅샷 공유에 대한 자세한 내용은 스냅샷 공유 단원을 참조하세요.

Linux, macOS, Unix의 경우:

```
aws docdb copy-db-cluster-snapshot \
--region us-west-2 \
--source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:999999999999:cluster-
snapshot:sample-cluster-snapshot \
--target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy \
--kms-key-id arn:aws:kms:us-west-2:123456789012:key/6c1f3264-1797-472b-
ba37-03011e682d28
```

Windows의 경우:

```
aws docdb copy-db-cluster-snapshot ^
--region us-west-2 ^
--source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:999999999999:cluster-
snapshot:sample-cluster-snapshot ^
--target-db-cluster-snapshot-identifier sample-cluster-snapshot-copy ^
--kms-key-id arn:aws:kms:us-west-2:123456789012:key/6c1f3264-1797-472b-
ba37-03011e682d28
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
 "DBClusterSnapshots": [
 {
 "AvailabilityZones": [],
 "DBClusterSnapshotIdentifier": "sample-cluster-snapshot-copy",
 "DBClusterIdentifier": "sample-cluster",
 "SnapshotCreateTime": "2025-08-22T11:27:00.497000+00:00",
 "Engine": "docdb",
 "Status": "copying",
 "Port": 0,
 "ClusterCreateTime": "2024-07-02T16:44:50.246000+00:00",
 "MasterUsername": "master-user",
 "EngineVersion": "5.0.0",
 "SnapshotType": "manual",
 "PercentProgress": 0,
 "StorageEncrypted": true,
 "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/6c1f3264-1797-472b-
ba37-03011e682d28",
 "DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:sample-cluster-snapshot-copy",
 "SourceDBClusterSnapshotArn": "arn:aws:rds:us-east-1:999999999999:cluster-
snapshot:sample-cluster-snapshot"
 }
 ]
}
```


> **Note**
>
> 암호화된 스냅샷 복사에 대한 자세한 내용은 클러스터 스냅샷 암호화 복사 섹션을 참조하세요.
> 저장된 암호화에 대한 자세한 내용은 Amazon DocumentDB 저장 데이터 암호화 섹션을 참조하세요.

Amazon DocumentDB 클러스터 스냅샷 공유

Amazon DocumentDB를 사용하면 다른 방법으로 수동 클러스터 스냅샷을 공유할 수 있습니다.

- 암호화 여부에 관계없이 수동 클러스터 스냅샷을 공유하면 권한 있는 AWS 계정이 스냅샷을 복사할 수 있습니다.

- 수동 클러스터 스냅샷을 암호화하든 암호화되지 않든 공유하면 권한 있는 AWS 계정이 스냅샷의 복사본을 만들고 복원하는 대신 스냅샷에서 클러스터를 직접 복원할 수 있습니다.

> **Note**
>
> 자동 클러스터 스냅샷을 공유하려면 자동 스냅샷을 복사하여 수동 클러스터 스냅샷을 생성한 다음 해당 복사본을 공유합니다. 이 프로세스는 AWS 백업 생성 리소스에도 적용됩니다.

수동 스냅샷을 최대 20개의 다른 스냅샷과 공유할 수 있습니다 AWS 계정. 암호화되지 않은 수동 스냅샷을 퍼블릭으로도 공유할 수 있습니다. 그러면 모든 계정에서 해당 스냅샷을 사용할 수 있습니다. 스냅샷을 퍼블릭으로 공유할 경우, 어떤 퍼블릭 스냅샷에도 비공개 정보가 포함되지 않도록 유의하세요.

수동 스냅샷을 다른 AWS 계정와 공유하고 AWS CLI 또는 Amazon DocumentDB API를 사용하여 공유 스냅샷에서 클러스터를 복원하는 경우 공유 스냅샷의 Amazon 리소스 이름(ARN)을 스냅샷 식별자로 지정해야 합니다.

암호화된 스냅샷 공유

다음 제한은 암호화된 스냅샷 공유에 적용됩니다.

- 암호화된 스냅샷을 퍼블릭으로는 공유할 수 없습니다.
- 스냅샷을 공유한 계정의 기본 AWS KMS 암호화 키를 사용하여 암호화된 스냅샷은 공유할 수 없습니다.

다음 단계에 따라 암호화된 스냅샷을 공유합니다.

1. 스냅샷을 암호화하는 데 사용된 AWS Key Management Service (AWS KMS) 암호화 키를 스냅샷에 액세스할 수 있는 계정과 공유합니다.

   키 정책에 다른 AWS 계정을 추가하여 다른 계정과 AWS KMS 암호화 AWS KMS 키를 공유할 수 있습니다. 키 정책 업데이트에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS KMS에서 키 정책 사용을 참조하세요. 키 정책 생성의 예는 이번 주제 후반부의 암호화된 스냅샷을 복사할 수 있도록 IAM 정책 생성 섹션을 참조하세요.
2. 아래와 AWS CLI같이를 사용하여 암호화된 스냅샷을 다른 계정과 공유합니다. ???

AWS KMS 암호화 키에 대한 액세스 허용

다른 사용자가 계정에서 공유된 암호화된 스냅샷을 복사 AWS 계정 하려면 스냅샷을 공유하는 계정이 스냅샷을 암호화한 AWS KMS 키에 액세스할 수 있어야 합니다. 키에 대한 다른 계정 액세스를 허용하려면 AWS KMS 키 정책의 보안 주체로 공유하려는 계정의 ARN으로 키의 AWS KMS 키 정책을 AWS KMS 업데이트합니다. 그런 다음 `kms:CreateGrant` 작업을 허용합니다.

AWS KMS 암호화 키에 대한 액세스 권한을 계정에 부여한 후 암호화된 스냅샷을 복사하려면 해당 계정에 아직 AWS Identity and Access Management (IAM) 사용자가 없는 경우 해당 계정을 생성해야 합니다. 또한 해당 계정은 사용자가 AWS KMS 키를 사용하여 암호화된 스냅샷을 복사할 수 있도록 허용하는 IAM 정책을 해당 IAM 사용자에게 연결해야 합니다. 계정은 IAM 사용자여야 하며 AWS KMS 보안 제한으로 인해 루트 AWS 계정 자격 증명일 수 없습니다.

다음 키 정책 예제에서 사용자 123451234512은 AWS KMS 암호화 키의 소유자입니다. 사용자 123456789012는 키를 공유 중인 계정입니다. 이 업데이트된 키 정책은 계정에 AWS KMS 키에 대한 액세스 권한을 부여합니다. 이를 위해 사용자 123456789012의 루트 AWS 계정 자격 증명에 대한 ARN을 정책의 보안 주체로 포함하고 `kms:CreateGrant` 작업을 허용합니다.

JSON

```
{
    "Id": "key-policy-1",
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {"AWS": [
                "arn:aws:iam::123451234512:user/KeyUser",
                "arn:aws:iam::123456789012:root"
            ]},
            "Action": [
                "kms:CreateGrant",
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
            "Resource": "*"},
            {
```

```
            "Sid": "Allow attachment of persistent resources",
            "Effect": "Allow",
            "Principal": {"AWS": [
                "arn:aws:iam::123451234512:user/KeyUser",
                "arn:aws:iam::123456789012:root"
            ]},
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
            "Resource": "*",
            "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
        }
    ]
}
```

암호화된 스냅샷을 복사할 수 있도록 IAM 정책 생성

외부 AWS 계정 사용자가 AWS KMS 키에 액세스할 수 있는 경우 해당 계정의 소유자는 계정에 대해 생성된 IAM 사용자가 해당 AWS KMS 키로 암호화된 암호화된 스냅샷을 복사하도록 허용하는 정책을 생성할 수 있습니다.

다음 예제에서는 IAM 사용자 for AWS 계정 123456789012에 연결할 수 있는 정책을 보여줍니다. 이 정책을 통해 IAM 사용자는 us-west-2 리전c989c1dd-a3f2-4a5d-8d96-e793d082ab26의 AWS KMS 키로 암호화된 계정 123451234512에서 공유 스냅샷을 복사할 수 있습니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowUseOfTheKey",
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey",
```

```
                "kms:CreateGrant",
                "kms:RetireGrant"
            ],
            "Resource": ["arn:aws:kms:us-west-2:123451234512:key/c989c1dd-
a3f2-4a5d-8d96-e793d082ab26"]
        },
        {
            "Sid": "AllowAttachmentOfPersistentResources",
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
            "Resource": ["arn:aws:kms:us-west-2:123451234512:key/c989c1dd-
a3f2-4a5d-8d96-e793d082ab26"],
            "Condition": {
                "Bool": {
                    "kms:GrantIsForAWSResource": true
                }
            }
        }
    ]
}
```

키 정책 업데이트에 관한 세부 정보는 AWS Key Management Service 개발자 안내서의 키 정책 AWS KMS을 참조하세요.

스냅샷 공유

AWS Management Console 또는 AWS CLI를 사용하여 Amazon DocumentDB 수동 클러스터 스냅샷(또는 자동 스냅샷의 사본)을 공유할 수 있습니다.

Using the AWS Management Console

를 사용하여 스냅샷을 공유하려면 다음 단계를 AWS Management Console완료하세요.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 [Snapshots]를 선택합니다.
3. 공유할 수동 스냅샷을 선택합니다.

4. 작업 드롭다운 메뉴에서 공유를 선택합니다.
5. DB 스냅샷 공개 여부에서 다음 옵션 중 하나를 선택합니다.

   - 소스가 암호화되지 않은 경우 퍼블릭을 선택하여 모든 AWS 계정이 수동 스냅샷에서 클러스터를 복원하도록 허용합니다. 또는 수동 스냅샷에서 클러스터를 복원하도록 지정한 AWS 계정만 허용하려면 프라이빗을 선택합니다.

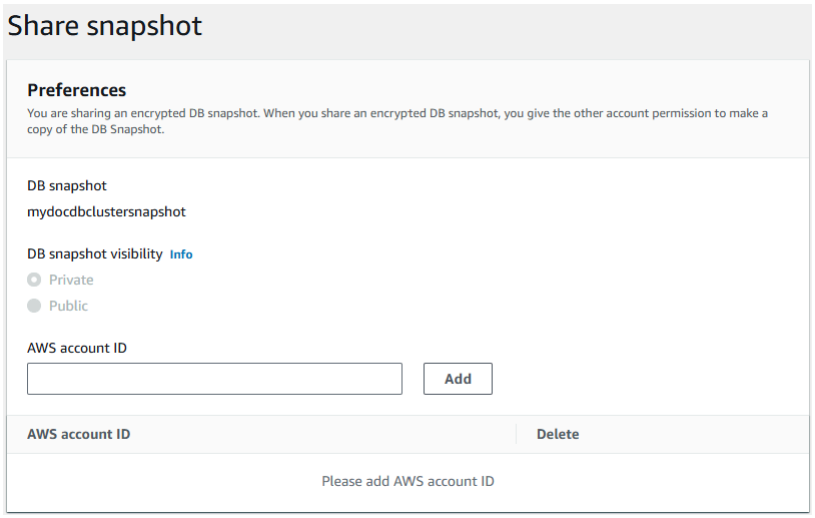
     > **Warning**
     >
     > DB 스냅샷 가시성을 퍼블릭으로 설정하면 모든 AWS 계정이 수동 스냅샷에서 클러스터를 복원하고 데이터에 액세스할 수 있습니다. 프라이빗 정보가 포함된 수동 클러스터 스냅샷을 퍼블릭으로 공유하지 마세요.

   - 원본 DB 클러스터가 암호화되어 있는 경우 암호화된 스냅샷을 퍼블릭으로 공유할 수 없으므로 DB snapshot visibility(DB 스냅샷 가시성)는 Private(프라이빗)으로 설정됩니다.

     > **Note**
     >
     > 기본값으로 암호화된 스냅샷은 공유할 AWS KMS key 수 없습니다.

6. AWS 계정 ID에가 수동 스냅샷에서 클러스터를 복원하도록 허용할 계정의 계정 AWS 식별자를 입력한 다음 추가를 선택합니다. 최대 20개의 AWS 계정까지 추가 계정 식별자를 포함하도록 반복합니다 AWS .

   허용된 AWS 계정 목록에 계정 식별자를 추가할 때 오류가 발생하면 잘못된 AWS 계정 식별자 오른쪽에 있는 삭제를 선택하여 목록에서 삭제할 수 있습니다.



7. 수동 스냅샷 복원을 허용하려는 모든 AWS 계정의 식별자를 추가한 후 저장을 선택하여 변경 사항을 저장합니다.

Using the AWS CLI

를 사용하여 스냅샷을 공유하려면 Amazon DocumentDB `modify-db-snapshot-attribute` 작업을 AWS CLI사용합니다. `--values-to-add` 파라미터를 사용하여 수동 스냅샷을 복원할 권한이 AWS 계정 있는의 IDs 목록을 추가합니다.

다음 예제에서는 123451234512 및 이라는 두 AWS 계정 식별자가 라는 스냅샷을 복원 123456789012하도록 허용합니다`manual-snapshot1`. 또한 스냅샷이 프라이빗으로 표시되도록 `all` 속성 값을 지웁니다.

Linux, macOS, Unix의 경우:

```
aws docdb modify-db-cluster-snapshot-attribute \
    --db-cluster-snapshot-identifier sample-cluster-snapshot \
    --attribute-name restore \
    --values-to-add '["123451234512","123456789012"]'
```

Windows의 경우:

```
aws docdb modify-db-cluster-snapshot-attribute ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot ^
    --attribute-name restore ^
    --values-to-add '["123451234512","123456789012"]'
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBClusterSnapshotAttributesResult": {
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot",
        "DBClusterSnapshotAttributes": [
            {
                "AttributeName": "restore",
                "AttributeValues": [
                    "123451234512",
                    "123456789012"
                ]
            }
        ]
```

```
    }
}
```

목록에서 AWS 계정 식별자를 제거하려면 `--values-to-remove` 파라미터를 사용합니다. 다음 예제에서는 AWS 계정 ID 123456789012가 스냅샷을 복원하지 못하도록 합니다.

Linux, macOS, Unix의 경우:

```
aws docdb modify-db-cluster-snapshot-attribute \
    --db-cluster-snapshot-identifier sample-cluster-snapshot \
    --attribute-name restore \
    --values-to-remove '["123456789012"]'
```

Windows의 경우:

```
aws docdb modify-db-cluster-snapshot-attribute ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot ^
    --attribute-name restore ^
    --values-to-remove '["123456789012"]'
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBClusterSnapshotAttributesResult": {
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot",
        "DBClusterSnapshotAttributes": [
            {
                "AttributeName": "restore",
                "AttributeValues": [
                    "123451234512"
                ]
            }
        ]
    }
}
```

클러스터 스냅샷에서 복원

Amazon DocumentDB(MongoDB 호환)는 스토리지 볼륨의 클러스터 스냅샷을 만듭니다. 클러스터 스냅샷에서 복원하여 새로운 클러스터를 생성할 수 있습니다. 클러스터를 복원할 때 복원 원본으로 사용

할 클러스터 스냅샷의 이름을 입력한 다음, 이 복원 작업에서 생성되는 새 클러스터의 이름을 입력합니다. 복원 시 새 클러스터가 생성되므로 스냅샷에서 기존 클러스터로의 복원은 불가능합니다.

클러스터 스냅샷에서 클러스터를 복원하는 경우:

- 이 작업에서는 클러스터만 복원하고 해당 클러스터에 대한 인스턴스는 복원하지 않습니다. `--db-cluster-identifier`에 복원된 클러스터의 식별자를 지정하여 복원된 클러스터의 인스턴스를 생성하려면 `create-db-instance` 작업을 호출해야 합니다. 클러스터가 사용 가능할 때에만 인스턴스를 생성할 수 있습니다.
- 암호화된 스냅샷은 암호화되지 않은 클러스터로 복원할 수 없습니다. 그러나 AWS KMS 키를 지정하여 암호화되지 않은 스냅샷을 암호화된 클러스터로 복원할 수 있습니다.
- 암호화된 스냅샷에서 클러스터를 복원하려면 AWS KMS 키에 액세스할 수 있어야 합니다.

> **Note**
>
> 3.6 클러스터를 4.0 클러스터로 복원할 수는 없지만 한 클러스터 버전에서 다른 클러스터 버전으로 마이그레이션할 수는 있습니다. 자세한 내용은 Amazon DocumentDB 마이그레이션 및 업그레이드 섹션을 참조하세요.

Using the AWS Management Console

다음 절차에서는 Amazon DocumentDB 관리 콘솔을 사용하여 클러스터 스냅샷에서 Amazon DocumentDB 클러스터를 복원하는 방법을 보여줍니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 스냅샷을 선택한 다음 클러스터 복원에 사용하려는 스냅샷 왼쪽에 있는 버튼을 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

3. 작업 메뉴에서 복원을 선택합니다.
4. 스냅샷 복원 페이지에서 구성 섹션을 완료합니다.

a. 클러스터 식별자 — 새 클러스터의 이름입니다. Amazon DocumentDB 제공 이름을 수락하거나 원하는 이름을 입력할 수 있습니다. Amazon DocumentDBsupplied 제공 이름은 `docdb-` 및 UTC 타임스탬프 형식입니다(예: `docdb-yyyy-mm-dd-hh-mm-ss`).

b. 인스턴스 클래스 — 새 클러스터의 인스턴스 클래스입니다. 기본 인스턴스 클래스를 수락하거나 드롭다운 목록에서 인스턴스 클래스를 선택할 수 있습니다.

c. 인스턴스 수 — 이 클러스터를 사용하여 생성할 인스턴스 수입니다. 기본값인 3개의 인스턴스(기본 읽기/쓰기 1개 및 읽기 전용 복제본 2개)를 수락하거나 드롭다운 목록에서 인스턴스 수를 선택할 수 있습니다.

5. 클러스터 스토리지 구성을 위해서는 스토리지 옵션을 선택합니다.

> **Note**
>
> Amazon DocumentDB I/O 최적화 스토리지 구성은 Amazon DocumentDB 5.0 및 8.0 엔진 버전에서만 사용할 수 있습니다.

6. 클러스터 구성이 마음에 들면 클러스터 복원을 선택하고 클러스터가 복원되는 동안 기다립니다.

7. 기본값이 아닌 Amazon VPC 또는 보안 그룹 지정과 같은 일부 구성을 변경하려면 페이지 왼쪽 하단에서 고급 설정 표시를 선택한 후 다음 단계를 계속 진행합니다.

a. 네트워크 설정 섹션을 완료합니다.

- 가상 프라이빗 클라우드(VPC) — 현재 VPC를 수락하거나 드롭다운 목록에서 VPC를 선택합니다.
- 서브넷 그룹 — `default` 서브넷 그룹을 수락하거나 드롭다운 목록에서 하나를 선택합니다.
- VPC 보안 그룹 — `default (VPC)` 보안 그룹을 수락하거나 목록에서 하나를 선택합니다.

b. 클러스터 옵션 섹션을 작성합니다.

- 데이터베이스 포트 — 기본 포트인 `27017`을 수락하거나, 위쪽 화살표 또는 아래쪽 화살표를 사용하여 애플리케이션 연결에 사용하려는 포트를 설정합니다.

c. 암호화 섹션을 작성합니다.

- 저장된 데이터 암호화 — 스냅샷이 암호화되면 이 옵션은 사용할 수 없습니다. 스냅샷이 암호화되지 않으면 다음에서 한 개를 선택할 수 있습니다.

- 모든 클러스터 데이터를 암호화하려면 Enable encryption-at-rest(유휴상태 암호화 활성)을 선택합니다. 이 옵션을 선택하면 KMS 키를 지정해야 합니다.
- 클러스터 데이터를 암호화하지 않으려면 Disable encryption-at-rest(유휴상태 암호화 비활성)을 선택합니다. 이 옵션을 선택하면 암호화 섹션이 완료됩니다.

- AWS KMS 키 - 드롭다운 목록에서 다음 중 하나를 선택합니다.
  - (기본값) aws/rds - 계정 번호와 AWS KMS 키 ID가이 옵션 다음에 나열됩니다.
  - 고객 관리형 키 -이 옵션은 AWS Identity and Access Management (IAM) 콘솔에서 IAM 암호화 키를 생성한 경우에만 사용할 수 있습니다. 이 키를 선택하여 클러스터를 암호화할 수 있습니다.
  - 키 ARN 입력 - ARN 상자에 AWS KMS 키의 Amazon 리소스 이름(ARN)을 입력합니다. ARN 형식은 `arn:aws:kms:<region>:<accountID>:key/<key-id>`입니다.

d. 로그 내보내기 섹션을 완료합니다.

- CloudWatch에 게시할 로그 유형 선택 — 다음 중 하나를 선택합니다.
  - 활성화됨 — 클러스터가 DDL 로깅을 Amazon CloudWatch Logs로 내보낼 수 있도록 합니다.
  - 비활성화됨 — 클러스터가 DDL 로그를 Amazon CloudWatch Logs로 내보내는 것을 방지합니다. 비활성이 기본값입니다.
- IAM 역할 — 목록에서 RDS 서비스 연결 역할을 선택합니다.

e. 태그 섹션을 완료합니다.

- 태그 추가 — 키 상자에 클러스터의 태그 이름을 입력합니다. 값 상자에 태그 값을 입력합니다(선택 사항). 태그는 AWS Identity and Access Management (IAM) 정책과 함께 사용하여 Amazon DocumentDB 리소스에 대한 액세스를 관리하고 리소스에 적용할 수 있는 작업을 제어합니다.

f. 삭제 방지 섹션을 완료합니다.

- 삭제 방지 활성화 — 클러스터가 실수로 삭제되지 않도록 보호합니다. 이 옵션이 활성화되면 클러스터를 삭제할 수 없습니다.

8. 클러스터 복원을 선택합니다.

Using the AWS CLI

를 사용하여 스냅샷에서 클러스터를 복원하려면 다음 파라미터와 함께 `restore-db-cluster-from-snapshot` 작업을 AWS CLI사용합니다. 자세한 내용은 RestoreDBClusterFromSnapshot을 참조하세요.

- **`--db-cluster-identifier`** - 필수입니다. 작업에서 생성되는 클러스터 이름 이 이름으로 된 클러스터는 이 작업 이전에 존재할 수 없습니다.

  클러스터 명명 제약 조건:

  - 길이는 [1-63]개의 글자, 숫자 또는 하이픈입니다.
  - 첫 번째 문자는 글자이어야 합니다.
  - 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.
  - 리전별로 Amazon RDS, Neptune 및 Amazon DocumentDB의 모든 클러스터 AWS 계정에 대해 고유해야 합니다.
- **`--snapshot-identifier`** — 필수입니다. 복원하는 데 사용되는 스냅샷의 이름입니다. 이 이름으로 된 스냅샷이 있어야 하며 사용 가능한 상태여야 합니다.
- **`--engine`** — 필수입니다. `docdb`여야 합니다.
- **`--storage-type standard | iopt1`** – 선택 사항. 기본값: `standard`.
- **`--kms-key-id`** - 선택 사항입니다. 암호화된 스냅샷을 복원하거나 암호화되지 않은 스냅샷에서 복원할 때 클러스터를 암호화할 때 사용할 AWS KMS 키 식별자의 ARN입니다. 키 ID를 AWS KMS 제공하면 스냅샷이 암호화되었는지 여부에 관계없이 복원된 클러스터가 AWS KMS 키로 암호화됩니다.

  `--kms-key-id` 형식은 `arn:aws:kms:<region>:<accountID>:key/<key-id>`입니다. `--kms-key-id` 파라미터 값을 지정하지 않으면 다음과 같이 진행됩니다.
  - 의 스냅샷`--snapshot-identifier`이 암호화된 경우 복원된 클러스터는 스냅샷을 암호화하는 데 사용된 것과 동일한 AWS KMS 키를 사용하여 암호화됩니다.
  - `--snapshot-identifier` 에 스냅샷이 암호화되지 않으면 복원된 클러스터는 암호화되지 않습니다.

Linux, macOS, Unix의 경우:

```
aws docdb restore-db-cluster-from-snapshot \
    --db-cluster-identifier sample-cluster-restore \
    --snapshot-identifier sample-cluster-snapshot \
```

```
    --engine docdb \
    --kms-key-id arn:aws:kms:us-east-1:123456789012:key/SAMPLE-KMS-KEY-ID
```

Windows의 경우:

```
aws docdb restore-db-cluster-from-snapshot ^
    --db-cluster-identifier sample-cluster-restore ^
    --snapshot-identifier sample-cluster-snapshot ^
    --engine docdb ^
    --kms-key-id arn:aws:kms:us-east-1:123456789012:key/SAMPLE-KMS-KEY-ID
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBCluster": {
        "AvailabilityZones": [
            "us-east-1c",
            "us-east-1b",
            "us-east-1a"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "sample-cluster-restore",
        "DBClusterParameterGroup": "default.docdb4.0",
        "DBSubnetGroup": "default",
        "Status": "creating",
        "Endpoint": "sample-cluster-restore.cluster-node.us-
east-1.docdb.amazonaws.com",
        "ReaderEndpoint": "sample-cluster-restore.cluster-node.us-
east-1.docdb.amazonaws.com",
        "MultiAZ": false,
        "Engine": "docdb",
        "EngineVersion": "4.0.0",
        "Port": 27017,
        "MasterUsername": "<master-user>",
        "PreferredBackupWindow": "02:00-02:30",
        "PreferredMaintenanceWindow": "tue:09:50-tue:10:20",
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcdefgh",
                "Status": "active"
            }
        ],
```

```
        "HostedZoneId": "ABCDEFGHIJKLM",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/<sample-key-id>",
        "DbClusterResourceId": "cluster-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "DBClusterArn": "arn:aws:rds:us-east-1:<accountID>:cluster:sample-cluster-
restore",
        "AssociatedRoles": [],
        "ClusterCreateTime": "2020-04-01T01:43:40.871Z",
        "DeletionProtection": true
    }
}
```

클러스터 상태가 사용 가능으로 전환되면, 클러스터에 적용할 인스턴스를 최소 한 개 생성합니다.

Linux, macOS, Unix의 경우:

```
aws docdb create-db-instance \
    --db-cluster-identifier sample-cluster-restore  \
    --db-instance-identifier sample-cluster-restore-instance \
    --availability-zone us-east-1b \
    --promotion-tier 2 \
    --db-instance-class db.r5.large \
    --engine docdb
```

Windows의 경우:

```
aws docdb create-db-instance ^
    --db-cluster-identifier sample-cluster-restore  ^
    --db-instance-identifier sample-cluster-restore-instance ^
    --availability-zone us-east-1b ^
    --promotion-tier 2 ^
    --db-instance-class db.r5.large ^
    --engine docdb
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "sample-cluster-restore-instance",
        "DBInstanceClass": "db.r5.large",
        "Engine": "docdb",
        "DBInstanceStatus": "creating",
        "PreferredBackupWindow": "02:00-02:30",
```

```
        "BackupRetentionPeriod": 1,
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcdefgh",
                "Status": "active"
            }
        ],
        "AvailabilityZone": "us-west-2b",
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-6242c31a",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-abcdefgh",
                    "SubnetAvailabilityZone": {
                        "Name": "us-west-2a"
                    },
                    "SubnetStatus": "Active"
                },
                {
                    ...
                }
            ]
        },
        "PreferredMaintenanceWindow": "fri:09:43-fri:10:13",
        "PendingModifiedValues": {},
        "EngineVersion": "4.0.0",
        "AutoMinorVersionUpgrade": true,
        "PubliclyAccessible": false,
        "DBClusterIdentifier": "sample-cluster-restore",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/<sample-key-id>",
        "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "CACertificateIdentifier": "rds-ca-2019",
        "PromotionTier": 2,
        "DBInstanceArn": "arn:aws:rds:us-east-1:<accountID>:db:sample-cluster-
restore-instance"
    }
}
```

특정 시점으로 복원

AWS Management Console 또는 AWS Command Line Interface ()를 사용하여 클러스터의 백업 보존 기간 내에 있는 시점으로 클러스터를 복원할 수 있습니다AWS CLI.

> **Note**
>
> 3.6 클러스터를 4.0 클러스터로 특정 시점으로 복원할 수는 없지만 한 클러스터 버전에서 다른 클러스터 버전으로 마이그레이션할 수는 있습니다. 자세한 내용은 Amazon DocumentDB 마이그레이션 및 업그레이드 섹션을 참조하세요.

클러스터를 특정 시점으로 복원할 경우 다음에 유의하세요.

- 새 클러스터는 기본 파라미터 그룹으로 생성된다는 점만 제외하고, 소스 클러스터와 동일한 구성으로 생성됩니다. 새 클러스터의 파라미터 그룹을 원본 클러스터의 파라미터 그룹으로 설정하려면 클러스터가 사용 가능한 상태로 된 이후에 클러스터를 수정합니다. 클러스터를 수정하는 방법에 대한 자세한 정보는 Amazon DocumentDB 클러스터 수정을 참조하세요.

Using the AWS Management Console

AWS Management Console에서 다음을 완료하여 보존 기간 중 특정 시점으로 클러스터를 복원할 수 있습니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다. 클러스터 목록에서 복원할 클러스터 왼쪽에 있는 버튼을 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘 (≡)을 선택합니다.

3. 작업 메뉴에서 특정 시점으로 복구를 선택합니다.
4. 복원 시간 섹션을 완료합니다. 여기서는 복원할 날짜 및 시간을 지정합니다.

a. 복원 날짜 — 가장 빠른 복원 시간과 최근 복원 시간 사이의 날짜를 선택하거나 입력합니다.

b. 복원 시간 — 가장 빠른 복원 시간과 최근 복원 시간 사이의 시간, 분, 초를 선택하거나 입력합니다.

5. 구성 섹션을 완료합니다.

a. 클러스터 식별자 — 기본 식별자를 수락하거나 원하는 식별자를 입력합니다.

클러스터 명명 제약 조건:

- 길이는 [1-63]개의 글자, 숫자 또는 하이픈입니다.
- 첫 번째 문자는 글자이어야 합니다.
- 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.
- 리전별로 Amazon RDS, Neptune 및 Amazon DocumentDB의 모든 클러스터 AWS 계정에 대해 고유해야 합니다.

b. 인스턴스 클래스 — 목록에서 클러스터 인스턴스에 사용할 인스턴스 클래스를 선택합니다.

c. 인스턴스 수 — 드롭다운 목록에서 클러스터 복원 시 생성할 인스턴스 수를 선택합니다.

6. 클러스터 스토리지 구성을 위해서는 스토리지 옵션을 선택합니다.

> **Note**
>
> Amazon DocumentDB I/O 최적화 스토리지 구성은 Amazon DocumentDB 5.0 및 8.0 엔진 버전에서만 사용할 수 있습니다.

7. 선택 사항. 네트워크 설정 및 클러스터 옵션을 구성하고 로그 내보내기를 활성화하려면 고급 설정 표시를 선택하고 다음 섹션을 완료합니다. 그렇지 않으면 다음 단계로 계속 진행합니다.

- 네트워크 설정
  1. 가상 프라이빗 클라우드(VPC)) — 목록에서 이 클러스터에 사용할 VPC를 선택합니다.
  2. 서브넷 그룹 — 목록에서 이 클러스터에 사용할 서브넷 그룹을 선택합니다.
  3. VPC 보안 그룹 — 목록에서 이 클러스터에 사용할 VPC 보안 그룹을 선택합니다.

- 클러스터 옵션

1. 포트 — 기본 포트(27017)를 수락하거나 위쪽 화살표 또는 아래쪽 화살표를 사용하여 이 클러스터와 통신할 포트를 설정합니다.

- 로그 내보내기
  1. 감사 로그 — 감사 로그를 Amazon CloudWatch Logs로 내보내기를 활성화하려면 이 옵션을 선택합니다. 이 옵션을 선택하는 경우, 클러스터의 사용자 지정 파라미터 그룹에서 `audit_logs`를 활성화해야 합니다. 자세한 내용은 Amazon DocumentDB 이벤트 감사 단원을 참조하십시오.
  2. 프로파일러 로그 — 작업 프로파일러 Amazon CloudWatch Logs를 로 내보내려면 이 옵션을 선택합니다. 이 옵션을 선택하는 경우, 클러스터의 사용자 지정 파라미터 그룹에서 다음 파라미터도 수정해야 합니다.
     - `profiler` — `enabled`으로 설정합니다.
     - `profiler_threshold_ms` — 작업 프로파일링에 대한 임계값을 설정하려면 값 `[0-INT_MAX]`로 설정합니다.
     - `profiler_sampling_rate` — 프로파일링할 느린 작업 비율을 설정하려면 값 `[0.0-1.0]`으로 설정합니다.

     자세한 내용은 Amazon DocumentDB 작업 프로파일링 단원을 참조하십시오.
  3. 프로파일러 로그 — 프로파일러 로그를 Amazon CloudWatch로 내보내기
  4. IAM 역할 — 목록에서 RDS 서비스 연결 역할을 선택합니다.
- Tags
  1. 태그 추가 — 키 상자에 클러스터의 태그 이름을 입력합니다. 값 상자에 태그 값을 입력합니다(선택 사항). AWS Identity and Access Management (IAM) 정책과 함께 이러한 태그를 사용하여 Amazon DocumentDB 리소스에 대한 액세스를 관리하고 해당 리소스에 적용 가능한 작업을 제어할 수 있습니다.
- 삭제 방지
  1. 삭제 방지 활성화 — 클러스터가 실수로 삭제되지 않도록 보호합니다. 이 옵션이 활성화되면 클러스터를 삭제할 수 없습니다.

8. 클러스터를 복원하려면 클러스터 생성을 선택합니다. 또는 취소를 선택하여 작업을 취소할 수 있습니다.

Using the AWS CLI

스냅샷의 백업 보존 기간을 사용하여 클러스터를 특정 시점으로 복원하려면 `restore-db-cluster-to-point-in-time` 작업을 다음 파라미터와 함께 사용합니다.

- **`--db-cluster-identifier`** - 필수입니다. 생성할 새 클러스터의 이름입니다. 이 클러스터는 작업 이전에 존재할 수 없습니다. 파라미터 값은 다음 제약 조건을 충족해야 합니다.

  클러스터 명명 제약 조건:

  - 길이는 [1-63]개의 글자, 숫자 또는 하이픈입니다.
  - 첫 번째 문자는 글자이어야 합니다.
  - 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.
  - 리전별로 Amazon RDS, Neptune 및 Amazon DocumentDB의 모든 클러스터 AWS 계정에 대해 고유해야 합니다.
- **`--restore-to-time`** — 클러스터를 복원할 UTC 날짜 및 시간입니다. 예를 들어 `2018-06-07T23:45:00Z`입니다.

  시간 제약:

  - 클러스터에 대해 복원 가능한 최신 시간보다 이전이어야 합니다.
  - `--use-latest-restorable-time` 파라미터를 제공하지 않은 경우에 지정해야 합니다.
  - `--use-latest-restorable-time` 파라미터가 `true`인 경우에는 지정할 수 없습니다.
  - `--restore-type` 파라미터 값이 `copy-on-write`인 경우에는 지정할 수 없습니다.
- **`--source-db-cluster-identifier`** — 복원할 소스 클러스터의 이름입니다. 이 클러스터가 있고 사용 가능해야 합니다.
- **`--use-latest-restorable-time`** 또는 **`--no-use-latest-restorable-time`** — 복원 가능한 최신 백업 시간으로 복원할지 여부를 나타냅니다. `--restore-to-time` 파라미터를 제공한 경우에는 지정할 수 없습니다.
- **`--storage-type standard | iopt1`** – 선택 사항. 기본값: `standard`.

AWS CLI 작업은 해당 클러스터의 인스턴스가 아닌 클러스터`restore-db-cluster-to-point-in-time`만 복원합니다. `--db-cluster-identifier`에 복원된 클러스터의 식별자를 지정하여 복원된 클러스터의 인스턴스를 생성하려면 `create-db-instance` 작업을 간접적으로 호출해야 합니다. `restore-db-cluster-to-point-in-time` 작업이 완료되고 복원된 클러스터를 사용 가능할 경우에만 인스턴스를 생성할 수 있습니다.

Example

다음 예에서는 `sample-cluster-snapshot` 스냅샷에서 복원 가능한 최신 시간으로 `sample-cluster-restored`를 생성합니다.

Linux, macOS, Unix의 경우:

```
aws docdb restore-db-cluster-to-point-in-time \
    --db-cluster-identifier sample-cluster-restored \
    --source-db-cluster-identifier sample-cluster-snapshot \
    --use-latest-restorable-time
```

Windows의 경우:

```
aws docdb restore-db-cluster-to-point-in-time ^
    --db-cluster-identifier sample-cluster-restored ^
    --source-db-cluster-identifier sample-cluster-snapshot ^
    --use-latest-restorable-time
```

Example

다음 예에서는 `sample-cluster-snapshot` 스냅샷에서 `sample-cluster` 백업 보존 기간 내에 있는 2018년 12월 11일 03:15(UTC)로 `sample-cluster-restored`를 생성합니다.

Linux, macOS, Unix의 경우:

```
aws docdb restore-db-cluster-to-point-in-time \
    --db-cluster-identifier sample-cluster-restore \
    --source-db-cluster-identifier sample-cluster \
    --restore-to-time 2020-05-12T03:15:00Z
```

Windows의 경우:

```
aws docdb restore-db-cluster-to-point-in-time ^
    --db-cluster-identifier sample-cluster-restore ^
    --source-db-cluster-identifier sample-cluster ^
    --restore-to-time 2020-05-12T03:15:00Z
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBCluster": {
```

```
        "AvailabilityZones": [
            "us-east-1c",
            "us-west-2b",
            "us-west-2a"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "sample-cluster-restored",
        "DBClusterParameterGroup": "sample-parameter-group",
        "DBSubnetGroup": "default",
        "Status": "creating",
        "Endpoint": "sample-cluster-restored.node.us-east-1.docdb.amazonaws.com",
        "ReaderEndpoint": "sample-cluster-restored.node.us-
east-1.docdb.amazonaws.com",
        "MultiAZ": false,
        "Engine": "docdb",
        "EngineVersion": "4.0.0",
        "Port": 27017,
        "MasterUsername": "master-user",
        "PreferredBackupWindow": "02:00-02:30",
        "PreferredMaintenanceWindow": "tue:09:50-tue:10:20",
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abc0123",
                "Status": "active"
            }
        ],
        "HostedZoneId": "ABCDEFGHIJKLM",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID^>:key/sample-key",
        "DbClusterResourceId": "cluster-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "DBClusterArn": "arn:aws:rds:us-east-1:<accountID>:cluster:sample-cluster-
restored",
        "AssociatedRoles": [],
        "ClusterCreateTime": "2020-04-24T20:14:36.713Z",
        "DeletionProtection": false
    }
}
```

클러스터 스냅샷 삭제

수동 스냅샷은 AWS Management Console 또는를 사용하여 수동으로 삭제할 때만 삭제되는 전체 백업입니다 AWS CLI. 자동 스냅샷은 스냅샷의 보존 기간이 만료되거나 스냅샷의 클러스터를 삭제하는 경우에만 삭제되므로 수동으로 삭제할 수 없습니다.

Using the AWS Management Console

를 사용하여 수동 클러스터 스냅샷을 삭제하려면 다음 단계를 AWS Management Console완료합니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 스냅샷를 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택하세요.

3. 스냅샷 목록에서 삭제할 스냅샷 왼쪽에 있는 버튼을 선택합니다. 스냅샷은 수동 유형이어야 합니다.

   1. 유형 열 또는 아래에서 `manual` 또는`automatic`로 나열되어 있는지 확인하여 스냅샷 유형이 수동인지 확인할 수 있습니다.
4. 작업 메뉴에서 삭제를 선택합니다. 삭제 옵션을 사용할 수 없는 경우 자동 스냅샷을 선택했을 수 있습니다.
5. 삭제 확인 화면에서 스냅샷을 삭제하려면 삭제를 선택합니다. 스냅샷을 유지하려면 취소를 선택합니다.

Using the AWS CLI

Amazon DocumentDB 수동 클러스터 스냅샷은 AWS CLI를 사용하여 수동으로 삭제할 수 있는 전체 백업입니다. 자동 스냅샷을 수동으로 삭제할 수 없습니다.

를 사용하여 수동 클러스터 스냅샷을 삭제하려면 다음 파라미터와 함께 `delete-db-cluster-snapshot` 작업을 AWS CLI사용합니다.

파라미터

- **--db-cluster-snapshot-identifier** — 필수입니다. 삭제할 수동 스냅샷의 이름입니다.

다음 예에서는 클러스터 스냅샷 sample-cluster-snapshot을 삭제합니다.

Linux, macOS, Unix의 경우:

```
aws docdb delete-db-cluster-snapshot \
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

Windows의 경우:

```
aws docdb delete-db-cluster-snapshot ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

이 작업의 출력에는 삭제한 클러스터 스냅샷의 세부 정보가 나열됩니다.

Amazon DocumentDB에서 모니터링 및 로깅

AWS 서비스를 모니터링하는 것은 시스템을 건강하게 유지하고 최적으로 작동하는 데 중요한 부분입니다. AWS 솔루션의 모든 부분에서 모니터링 데이터를 수집하여 장애 또는 성능 문제를 더 쉽게 디버깅하고 수정할 수 있습니다. 솔루션 모니터링을 시작하기 전에 다음 질문을 고려하세요 AWS .

- 모니터링의 목표
- 모니터링할 리소스는 무엇입니까?
- 이 리소스를 얼마나 자주 모니터링합니까?
- 사용할 모니터링 도구
- 누가 모니터링을 담당합니까?
- 문제가 발생한 경우 알릴 대상은 누구이며 어떤 의미가 있습니까?

현재 성능 패턴을 이해하고 성능 이상을 식별하고 문제를 해결할 수 있는 방법을 공식화하려면 다양한 시간대 및 다양한 로드 조건에 대한 기준 성능 지표를 설정해야 합니다. 솔루션을 모니터링할 AWS 때 향후 참조 및 기준 설정을 위해 과거 모니터링 데이터를 저장합니다.

일반적으로 성능 지표에 허용되는 값은 기준이 무엇인지 그리고 애플리케이션 무엇을 수행하는지에 따라 다릅니다. 기준과의 일관된 차이 또는 추세를 조사하십시오. 특정 지표 유형에 대한 참고 정보는 다음과 같습니다.

- CPU 또는 RAM 사용량이 많음 - CPU 또는 RAM 사용량이 많을 경우 해당 애플리케이션의 목표(처리량 또는 동시성)와 일치하고 예상되는 결과라면 문제가 되지 않을 수 있습니다.
- 스토리지 볼륨 사용량 - 총 스토리지 볼륨 공간의 85% 이상이 계속 사용될 경우 스토리지 사용량(`VolumeBytesUsed`)을 검사합니다. 스토리지 볼륨에서 데이터를 삭제할 수 있는지 또는 다른 시스템에 데이터를 아카이브하여 공간을 확보할 수 있는지 결정합니다. 자세한 정보는 Amazon DocumentDB 스토리지 및 Amazon DocumentDB 할당량(을)를 참조하세요.
- 네트워크 트래픽 - 네트워크 트래픽의 경우 시스템 관리자에게 문의하여 해당 도메인 네트워크 및 인터넷 연결의 예상 처리량을 확인합니다. 처리량이 기대값보다 항상 낮으면 네트워크 트래픽을 검사합니다.
- 데이터베이스 연결 - 인스턴스 성능 저하 및 응답 시간 지연과 함께 사용자 연결 수가 많을 경우 데이터베이스 연결 제한을 고려해 봅니다. 인스턴스에 대한 최적의 사용자 연결 수는 해당 인스턴스 클래스와, 수행하는 작업의 복잡성에 따라 다릅니다.

- IOPS 지표 - OPS 지표의 기대값은 디스크 사양 및 서버 구성에 따라 다르므로 해당 기준에 일반적인 값을 파악합니다. 값이 기준과 계속 차이가 나는지 검사합니다. 최적의 IOPS 성능을 위해, 일반적인 작업 세트가 메모리에 적합하고 읽기 및 쓰기 작업을 최소화하는지 확인합니다.

Amazon DocumentDB(MongoDB 호환)은 모니터링을 통해 Amazon DocumentDB 클러스터와 인스턴스의 상태와 성능을 평가할 수 있는 Amazon CloudWatch 지표를 다양하게 제공합니다. Amazon DocumentDB 콘솔, CloudWatch API AWS CLI, 성능 개선 도우미 등 다양한 도구를 사용하여 Amazon DocumentDB 지표를 볼 수 있습니다.

주제

- Amazon DocumentDB 클러스터 상태 모니터링
- Amazon DocumentDB 인스턴스 상태 모니터링
- Amazon DocumentDB 권장 사항 보기
- Amazon DocumentDB 이벤트 구독 사용
- CloudWatch를 사용하여 Amazon DocumentDB 모니터링
- 를 사용하여 Amazon DocumentDB API 호출 로깅 AWS CloudTrail
- Amazon DocumentDB 작업 프로파일링
- 성능 개선 도우미로 모니터링

Amazon DocumentDB 클러스터 상태 모니터링

클러스터 상태는 클러스터의 상태를 나타냅니다. Amazon DocumentDB 콘솔 또는 명령을 사용하여 클러스터의 상태를 볼 수 있습니다 AWS CLI `describe-db-clusters`.

주제

- 클러스터 상태 값
- 클러스터 상태 모니터링

클러스터 상태 값

다음 표에는 클러스터 상태에 유효한 값이 나열되어 있습니다.

클러스터 상태	설명
active	클러스터가 활성 상태입니다. 이 상태는 엘라스틱 클러스터에만 적용됩니다.
available	클러스터에 문제가 없으며 사용할 수 있습니다. 이 상태는 인스턴스 기반 클러스터에만 적용됩니다.
backing-up	현재 클러스터를 백업 중입니다.
creating	클러스터를 생성 중입니다. 생성 도중에는 액세스할 수 없습니다.
deleting	클러스터를 삭제 중입니다. 삭제 도중에는 액세스할 수 없습니다.
failing-over	기본 인스턴스에서 Amazon DocumentDB 복제본에 대한 장애 조치가 수행 중입니다.
inaccessible-encryption-credentials	클러스터를 암호화하거나 복호화하는 데 사용되는 AWS KMS 키에는 액세스할 수 없습니다.
maintenance	클러스터에 유지 관리 업데이트를 적용 중입니다. 이 상태는 Amazon DocumentDB가 한참 전에 미리 예약하는 클러스터 수준의 유지 관리에 사용됩니다.

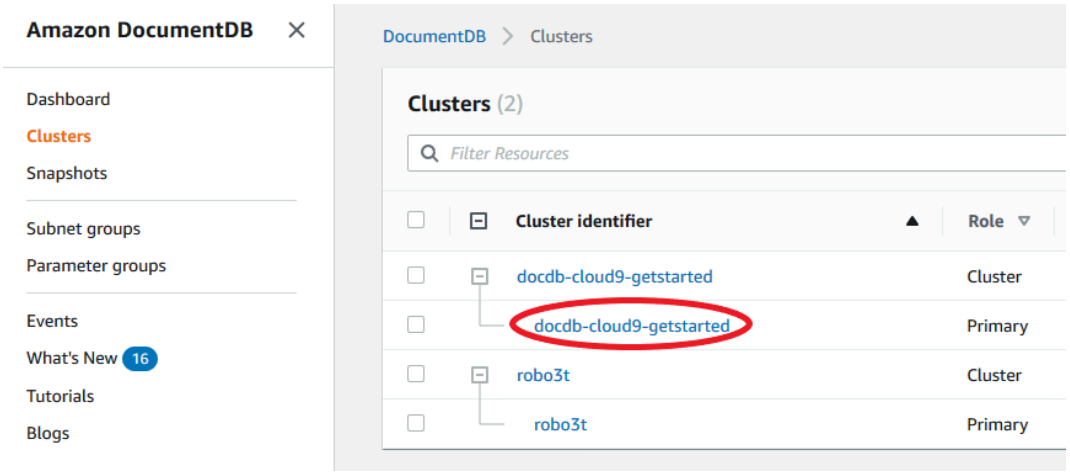
클러스터 상태	설명
`migrating`	클러스터 스냅샷이 클러스터로 복원 중입니다.
`migration-failed`	마이그레이션이 실패했습니다.
`modifying`	고객의 클러스터 수정 요청으로 인해 클러스터를 수정 중입니다.
`renaming`	고객의 이름 바꾸기 요청으로 인해 클러스터 이름을 바꾸는 중입니다.
`resetting-master-credentials`	고객의 재설정 요청으로 인해 클러스터 사용자 자격 증명을 재설정하는 중입니다.
`upgrading`	클러스터 엔진 버전이 현재 업그레이드 중입니다.

클러스터 상태 모니터링

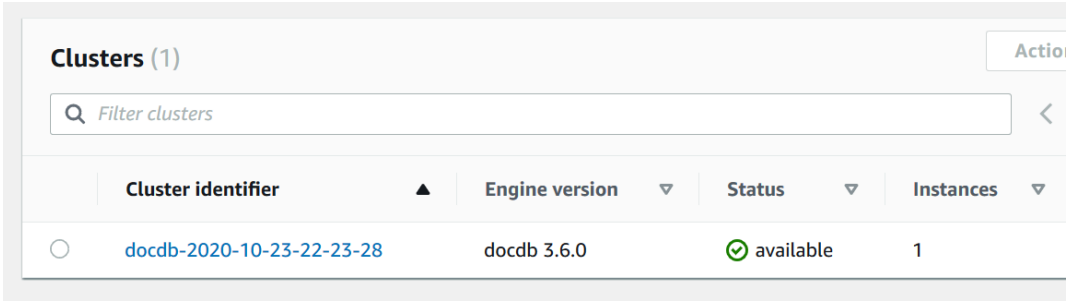
Using the AWS Management Console

AWS Management Console 를 사용하여 클러스터의 상태를 확인할 때는 다음 절차를 따르십시오.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터(Clusters)를 선택합니다.
3. 클러스터 탐색 상자에 클러스터 식별자 열이 표시됩니다. 인스턴스는 다음 스크린샷과 마찬가지로 클러스터 아래에 나열됩니다.



4. 클러스터 식별자 열에서 관심 있는 인스턴스의 이름을 찾습니다. 그런 다음, 인스턴스의 상태를 찾으려면 아래 그림과 같이 상태 열에 대한 해당 행을 읽습니다.



Using the AWS CLI

AWS CLI 를 사용하여 클러스터의 상태를 확인할 때는 `describe-db-clusters` 작업을 사용합니다. 다음 코드는 `sample-cluster` 클러스터의 상태를 찾습니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster  \
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

Windows의 경우:

```
aws docdb describe-db-clusters ^
    --db-cluster-identifier sample-cluster  ^
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
[
    [
        "sample-cluster",
        "available"
    ]
]
```

Amazon DocumentDB 인스턴스 상태 모니터링

Amazon DocumentDB는 데이터베이스에 구성된 각 인스턴스의 현재 상태에 대한 정보를 제공합니다.

Amazon DocumentDB 인스턴스의 상태를 확인할 수 있는 세 가지 유형이 있습니다.

- 인스턴스 상태:이 상태는의 클러스터 테이블 상태 열에 표시 AWS Management Console 되며 인스턴스의 현재 수명 주기 조건을 보여줍니다. 상태 열에 표시된 값은 `DescribeDBCluster` API 응답의 `Status` 필드에서 파생됩니다.
- 인스턴스 상태:이 상태는의 클러스터 테이블의 인스턴스 상태 열에 표시 AWS Management Console 되며 데이터 관리 및 검색을 담당하는 구성 요소인 데이터베이스 엔진이 실행 중인지 여부를 보여줍니다. 인스턴스 상태 열에 표시된 값은 Amazon CloudWatch `EngineUptime` 시스템 지표를 기반으로 합니다.
- 유지 관리 상태:이 상태는의 클러스터 테이블 유지 관리 열에 표시 AWS Management Console 되며 인스턴스에 적용해야 하는 유지 관리 이벤트의 상태를 나타냅니다. 유지 관리 상태는 다른 인스턴스 상태와는 독립적이며 `PendingMaintenanceAction` API에서 파생됩니다. 유지 관리 상태에 대한 자세한 내용은 Amazon DocumentDB 유지 관리 섹션을 참조하세요.

주제

- 인스턴스 상태 값
- AWS Management Console 또는를 사용하여 인스턴스 상태 모니터링 AWS CLI
- 인스턴스 상태 값
- 를 사용하여 인스턴스 상태 모니터링 AWS Management Console

인스턴스 상태 값

다음 표에는 인스턴스에 대해 가능한 상태 값과 각 상태별 요금이 표시되어 있습니다. 인스턴스와 스토리지에 요금이 청구되는지, 스토리지에만 청구되는지, 혹은 청구되지 않는지 표시됩니다. 모든 인스턴스 상태의 백업 사용에는 항상 청구됩니다.

인스턴스 상태	청구	설명
`available`	청구	인스턴스에 문제가 없으며 사용할 수 있습니다.
`backing-up`	청구	현재 인스턴스를 백업 중입니다.
`configuring-log-exports`	청구	Amazon CloudWatch Logs에 로그 파일을 게시하는 기능이 이 인스턴스에 대해 활성화 또는 비활성화 상태입니다.
`creating`	미청구	인스턴스를 생성 중입니다. 생성 중인 인스턴스에는 액세스할 수 없습니다.
`deleting`	미청구	인스턴스를 삭제 중입니다.
`failed`	미청구	인스턴스 오류가 발생하여 Amazon DocumentDB에서 이를 복구하지 못했습니다. 데이터를 복구하려면 인스턴스의 복원 가능한 가장 최근 시간에 대한 시점 복원을 수행합니다.
`inaccessible-encryption-credentials`	미청구	인스턴스를 암호화하거나 복호화하는 데 사용되는 AWS KMS 키에 액세스할 수 없습니다.
`incompatible-network`	미청구	Amazon DocumentDB에서 인스턴스에 대한 복구 작업을 시도했으나 VPC의 상태로 인하여 작업을 완료할 수 없어 복구하지 못했습니다. 예를 들어 서브넷에 사용 가능한 IP 주소가 모두 사용 중이어서 Amazon DocumentDB에서 인스턴스에 대한 IP 주소를 받을 수 없는 경우 이러한 상태가 발생할 수 있습니다.

인스턴스 상태	청구	설명
maintenance	청구	Amazon DocumentDB는 인스턴스에 유지 관리 업데이트를 적용합니다. 이 상태는 Amazon DocumentDB가 한참 전에 미리 예약하는 인스턴스 수준의 유지 관리에 사용됩니다. 현재 이 상태를 통해 고객에게 유지 관리 작업을 추가로 알릴 방법을 평가 중입니다.
modifying	청구	인스턴스 수정 요청으로 인해 인스턴스를 수정 중입니다.
rebooting	청구	인스턴스 재부팅이 필요한 요청 또는 Amazon DocumentDB 프로세스로 인해 인스턴스를 재부팅 중입니다.
renaming	청구	이름 바꾸기 요청으로 인해 인스턴스 이름을 바꾸는 중입니다.
resetting-master-credentials	청구	재설정 요청으로 인해 인스턴스 사용자 자격 증명을 재설정하는 중입니다.
restore-error	청구	특정 시점으로 복원을 시도하거나 스냅샷에서 복원을 시도할 때 인스턴스에서 오류가 발생했습니다.
starting	요금 부과 스토리지	인스턴스를 시작하는 중입니다.
stopped	요금 부과 스토리지	인스턴스가 중지되었습니다.
stopping	요금 부과 스토리지	인스턴스를 중지 중입니다.

인스턴스 상태	청구	설명
`storage-full`	청구	인스턴스에 할당된 스토리지 용량이 거의 다 찼습니다. 즉시 해결해야 하는 중요한 상태이므로 인스턴스를 수정하여 스토리지를 확장합니다. 스토리지 공간이 부족해지면 경고하도록 Amazon CloudWatch 경보를 설정해 이러한 상황이 발생하지 않도록 할 수 있습니다.

AWS Management Console 또는를 사용하여 인스턴스 상태 모니터링 AWS CLI

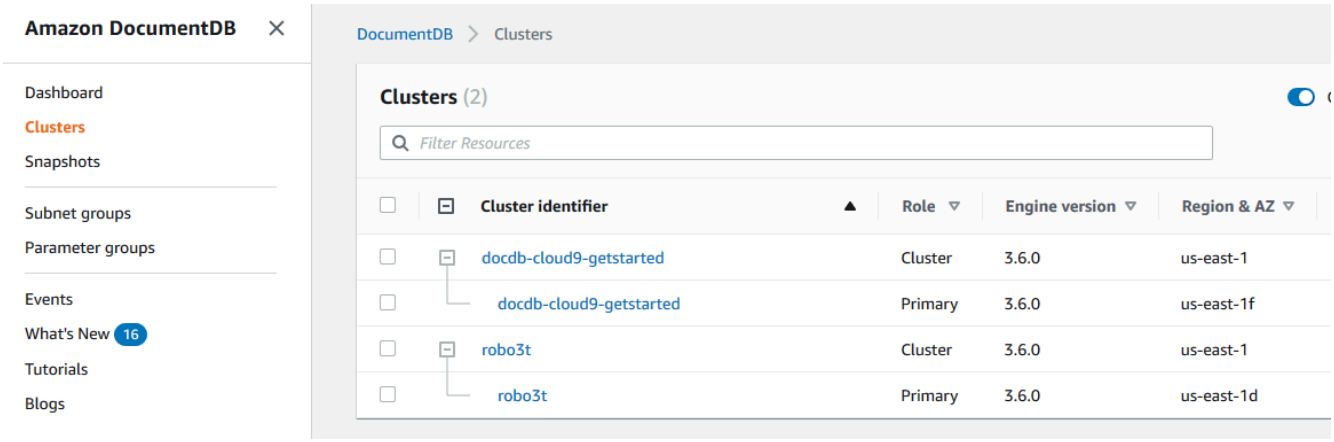
AWS Management Console 또는를 사용하여 인스턴스의 상태를 AWS CLI 모니터링합니다.

Using the AWS Management Console

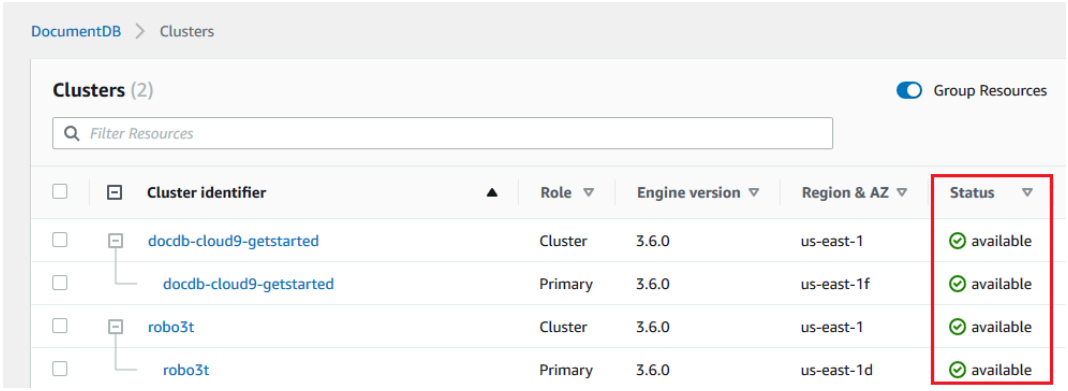
AWS Management Console 를 사용하여 클러스터의 상태를 확인할 때는 다음 절차를 따르십시오.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터(Clusters)를 선택합니다.

> **Note**
>
> 참고로 클러스터 탐색 상자의 클러스터 식별자 열에는 클러스터와 인스턴스가 모두 표시됩니다. 인스턴스는 다음 이미지와 마찬가지로 클러스터 아래에 나열됩니다.



3. 관심 있는 인스턴스의 이름을 찾으십시오. 그런 다음 인스턴스의 상태를 찾으려면 다음 그림과 같이 상태 열에 대한 해당 행을 읽습니다.



Using the AWS CLI

AWS CLI 를 사용하여 클러스터의 상태를 확인할 때는 `describe-db-instances` 작업을 사용합니다. 다음 코드는 `sample-cluster-instance-01` 인스턴스의 상태를 찾습니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb describe-db-instances \
          --db-instance-identifier sample-cluster-instance-01  \
          --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceStatus]'
```

Windows의 경우:

```
aws docdb describe-db-instances ^
          --db-instance-identifier sample-cluster-instance-01  ^
```

```
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceStatus]'
```

이 작업의 출력은 다음과 같습니다.

```
[
    [
        "sample-cluster-instance-01",
        "available"
    ]
]
```

인스턴스 상태 값

다음 표는 인스턴스를 위한 잠재적인 상태 값을 나열하고 있습니다. 의 클러스터 테이블에 있는 인스턴스 상태 열은 데이터 저장 AWS Management Console, 관리 및 검색을 담당하는 구성 요소인 데이터베이스 엔진이 정상적으로 작동하는지 여부를 보여줍니다. 또한 이 열은 CloudWatch에서 사용할 수 있는 `EngineUptime` 시스템 지표에 각 인스턴스의 상태가 표시되는지 여부도 나타냅니다.

인스턴스 상태	설명
정상	데이터베이스 엔진이 Amazon DocumentDB 인스턴스에서 실행 중입니다.
비정상	데이터베이스 엔진이 실행 중이 아니거나 1분이 채 지나지 않아 다시 시작되었습니다.

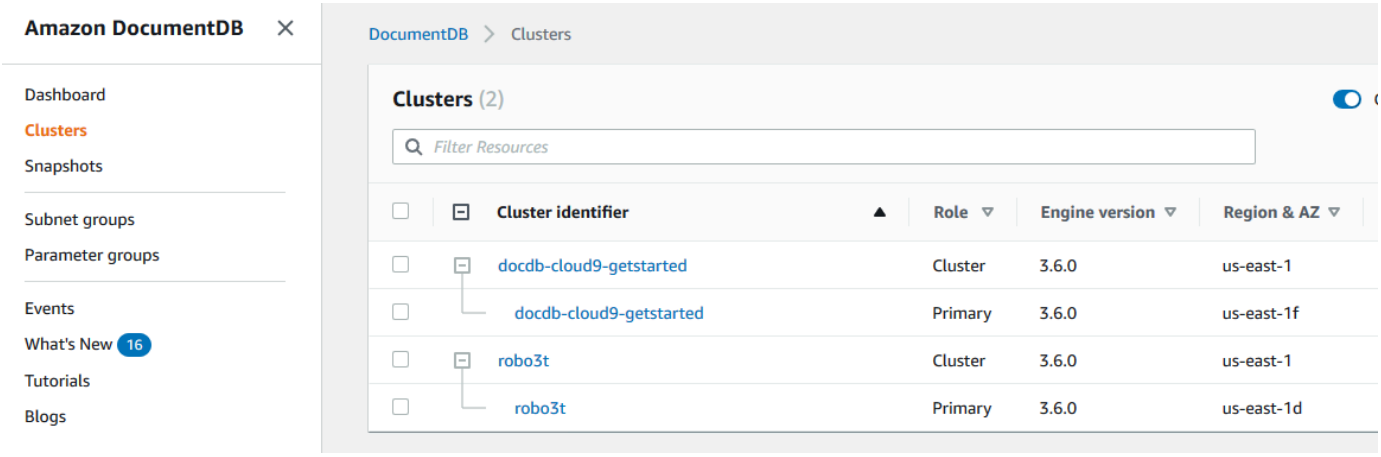
를 사용하여 인스턴스 상태 모니터링 AWS Management Console

AWS Management Console 를 사용하여 인스턴스의 상태를 모니터링합니다.

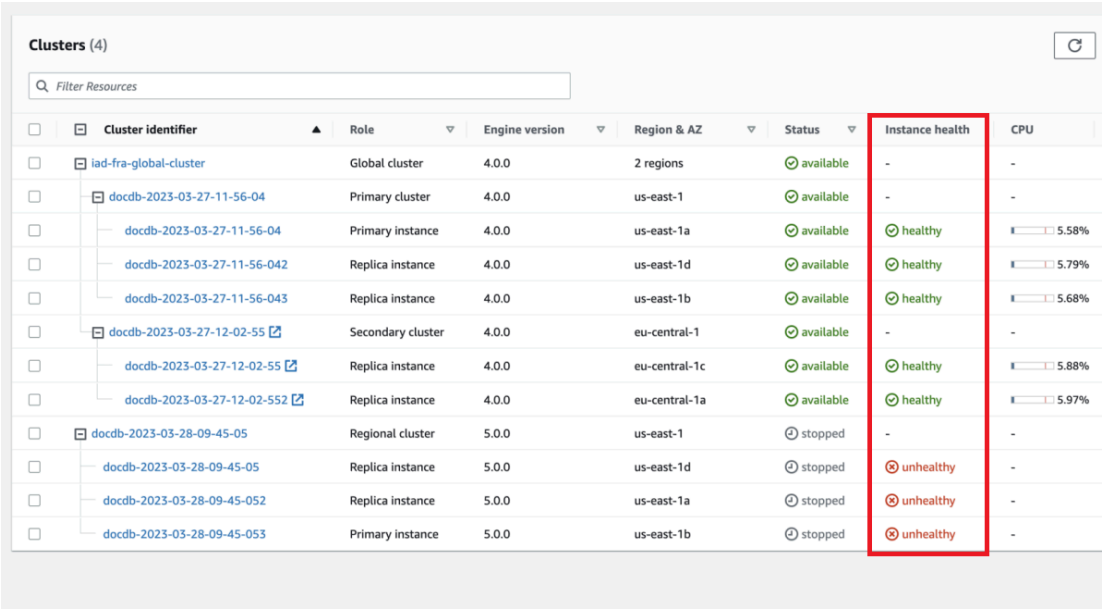
AWS Management Console를 사용할 때 다음 단계에 따라 인스턴스의 상태를 파악합니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

> **Note**
>
> 참고로 클러스터 탐색 상자의 클러스터 식별자 열에는 클러스터와 인스턴스가 모두 표시됩니다. 인스턴스는 다음 이미지와 마찬가지로 클러스터 아래에 나열됩니다.



3. 관심 있는 인스턴스의 이름을 찾으십시오. 그런 다음 인스턴스의 상태를 찾으려면 다음 그림과 같이 인스턴스 상태 열에 대한 해당 행을 읽습니다.



> **Note**
>
> 인스턴스 상태 폴링은 60초마다 발생하며 CloudWatch `EngineUptime` 시스템 지표를 기반으로 합니다. 인스턴스 상태 열의 값은 자동으로 업데이트됩니다.

Amazon DocumentDB 권장 사항 보기

Amazon DocumentDB는 인스턴스 및 클러스터와 같은 데이터베이스 리소스에 대한 자동화된 권장 사항 목록을 제공합니다. 이러한 권장 사항은 클러스터와 인스턴스 구성 분석을 통해 모범 사례 지침을 제공합니다.

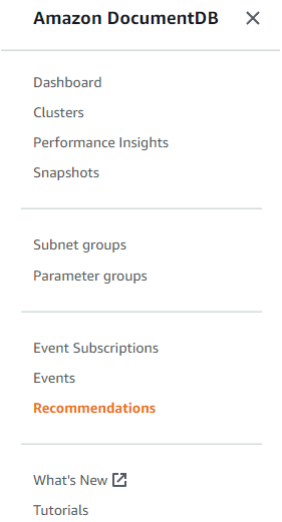
이러한 권장 사항의 예는 다음을 참조하세요.

유형	설명	권장 사항	추가 정보
1개의 인스턴스	클러스터에는 하나의 인스턴스만 포함됩니다.	성능 및 가용성: 다른 가용 영역에 동일한 인스턴스 클래스를 가진 다른 인스턴스를 추가합니다.	Amazon DocumentDB 고가용성 및 복제

Amazon DocumentDB는 리소스가 생성되거나 수정될 때 리소스에 대해 권장 사항을 생성합니다. Amazon DocumentDB는 리소스를 주기적으로 스캔하고 권장 사항을 생성합니다.

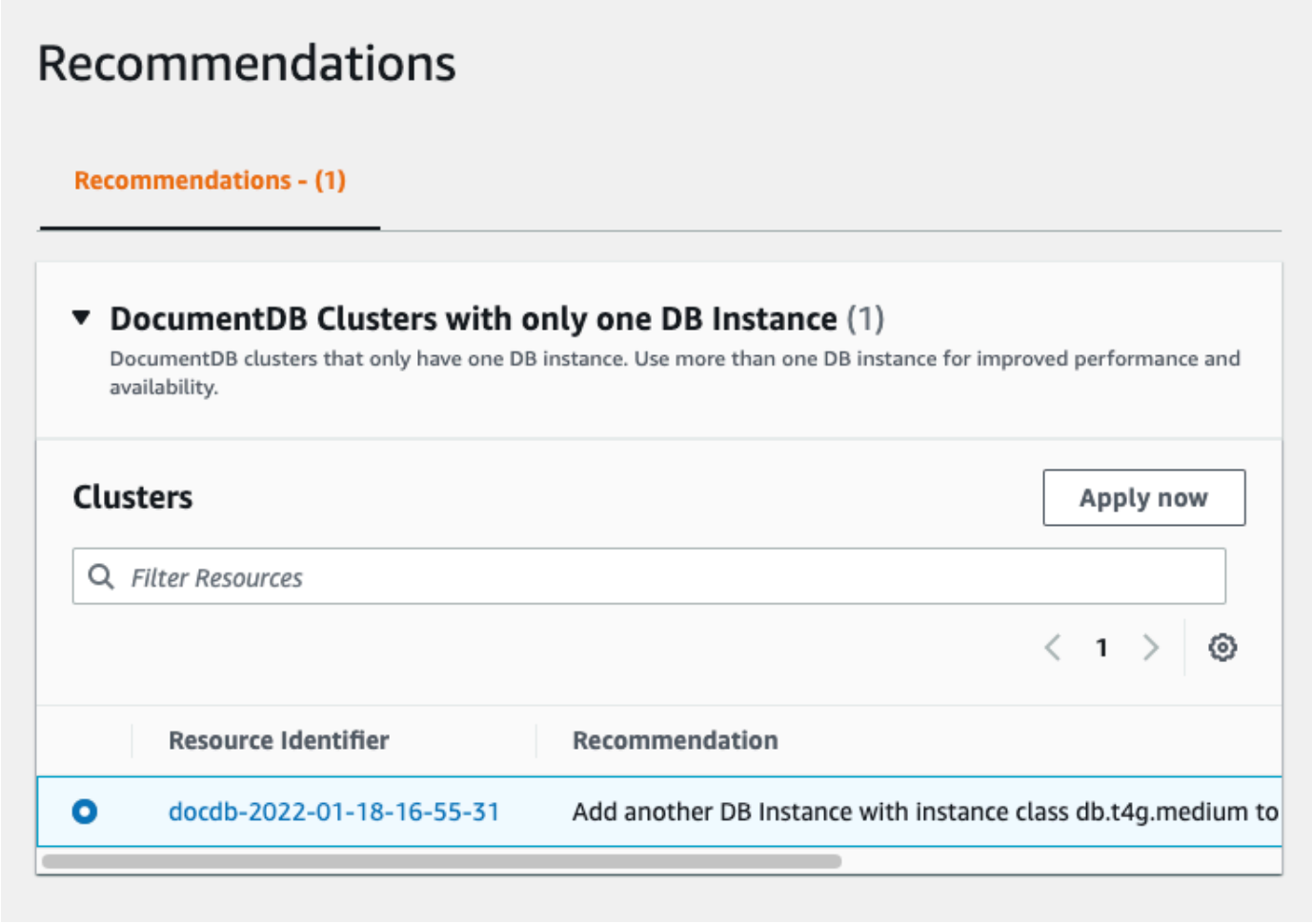
Amazon DocumentDB 권장 사항을 확인하고 이에 대한 조치를 취하려면

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 권장 사항을 선택합니다.



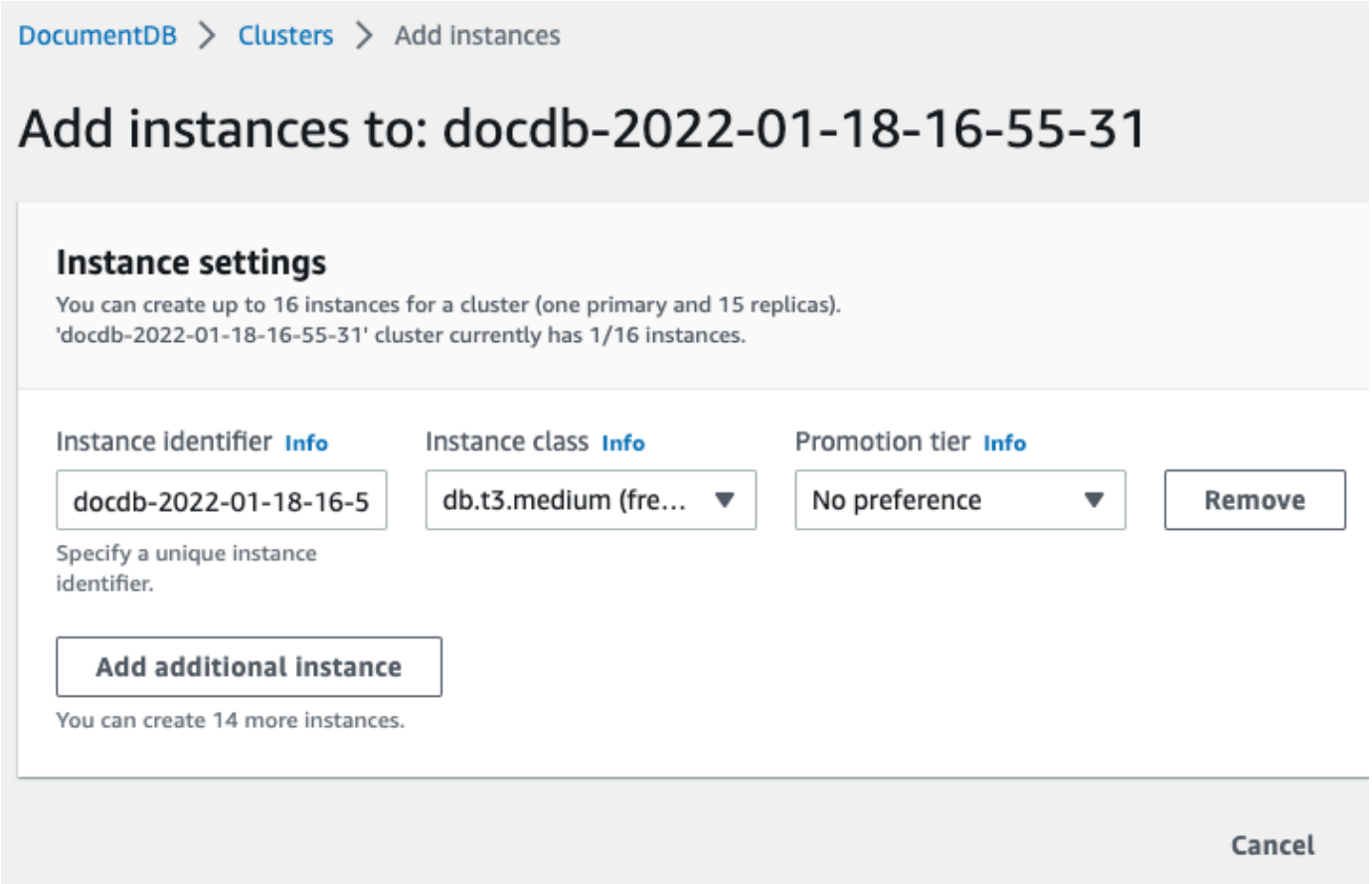
3. 권장 사항 대화 상자에서 관심 섹션을 펼치고 권장 작업을 선택합니다.

   다음 예제에서 권장 작업은 인스턴스가 하나뿐인 Amazon DocumentDB 클러스터에 적용됩니다. 성능과 가용성을 개선하려면 다른 인스턴스를 추가하는 것이 좋습니다.



4. 지금 적용을 선택합니다.

   이 예시에서는 인스턴스 추가 대화상자가 나타납니다.



5. 새 인스턴스의 설정을 수정하고 생성을 선택합니다.

Amazon DocumentDB 이벤트 구독 사용

Amazon DocumentDB는 Amazon Simple Notification Service(SNS)를 사용하여 Amazon DocumentDB 이벤트 발생 시 알림을 제공합니다. 이러한 알림은 이메일 AWS 리전, 문자 메시지 또는 HTTP 엔드포인트 호출과 같이에 대해 Amazon SNS에서 지원하는 모든 형식일 수 있습니다.

Amazon DocumentDB는 이러한 이벤트를 가입할 수 있는 카테고리로 그룹화하여 해당 카테고리의 이벤트가 발생할 때 알림을 받을 수 있도록 합니다. 구독 가능한 이벤트 범주로는 DB 인스턴스, DB 클러스터, DB 스냅샷, DB 클러스터 스냅샷, DB 파라미터 그룹, DB 보안 그룹 등이 있습니다. 예를 들어 임의의 DB 인스턴스에 대한 Backup 카테고리를 구독할 경우 백업 관련 이벤트가 발생하여 DB 인스턴스에 영향을 끼칠 때마다 알림 메시지가 수신됩니다. 또한 이벤트 알림 메시지 구독이 변경되어도 알림 메시지가 수신됩니다.

이벤트는 클러스터와 인스턴스 수준에서 모두 발생하므로 클러스터나 인스턴스에 가입하면 이벤트를 수신할 수 있습니다.

이벤트 알림 메시지는 구독 생성 시 입력한 주소로 보내집니다. 모든 이벤트 알림을 받는 구독과 프로덕션 인스턴스에 대한 중요한 이벤트만 포함하는 다른 구독 등 여러 가지 구독을 생성할 수 있습니다. 구독을 삭제하지 않고도 알림을 쉽게 끌 수 있습니다. 이렇게 하려면 Amazon DocumentDB 콘솔에서 활성화 라디오 버튼을 아니오로 설정하세요.

> **Important**
>
> Amazon DocumentDB는 이벤트 스트림에서 전송되는 이벤트 순서를 보장하지 않습니다. 이벤트 순서는 변경될 수 있습니다.

Amazon DocumentDB는 Amazon SNS 주제의 Amazon 리소스 이름(ARN)을 사용하여 각 구독을 식별합니다. 구독을 작성할 때 Amazon DocumentDB 콘솔이 ARN을 작성합니다.

Amazon DocumentDB 이벤트 구독에 대한 청구는 Amazon SNS를 통해 이루어집니다. Amazon SNS 요금은 이벤트 알림 사용 시 적용됩니다. 자세한 내용은 Amazon Simple Notification Service Pricing을 참조하세요. 아마존 Amazon SNS 요금 외에 이벤트 구독료는 Amazon DocumentDB에서 청구하지 않습니다.

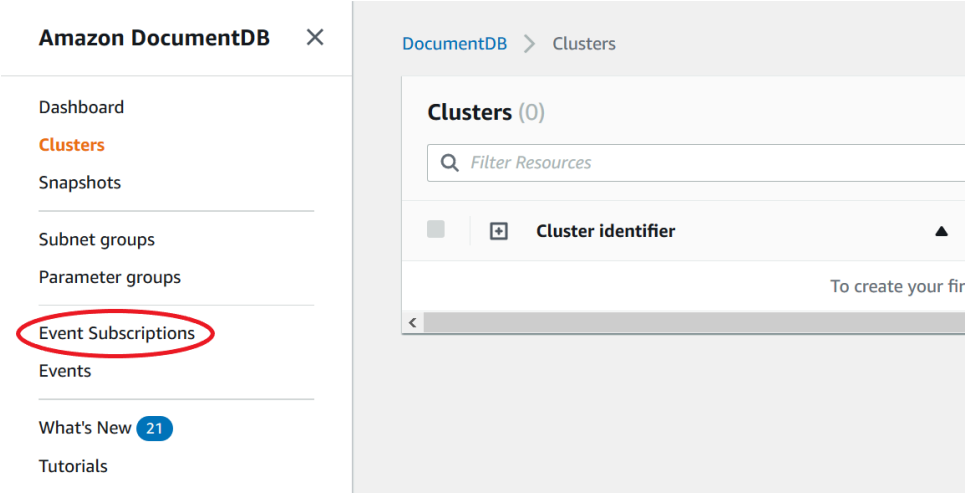
주제

- Amazon DocumentDB 이벤트 구독
- Amazon DocumentDB 이벤트 알림 구독 관리
- Amazon DocumentDB 이벤트 범주 및 메시지

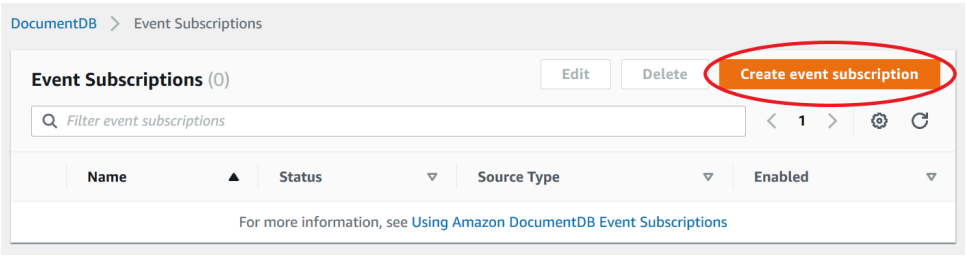
Amazon DocumentDB 이벤트 구독

Amazon DocumentDB 콘솔을 사용하여 다음과 같이 이벤트 구독을 구독할 수 있습니다:

1. 에서에 로그인 AWS Management Console 합니다https://console.aws.amazon.com/docdb.
2. 탐색 창에서 [이벤트 구독]을 선택합니다.

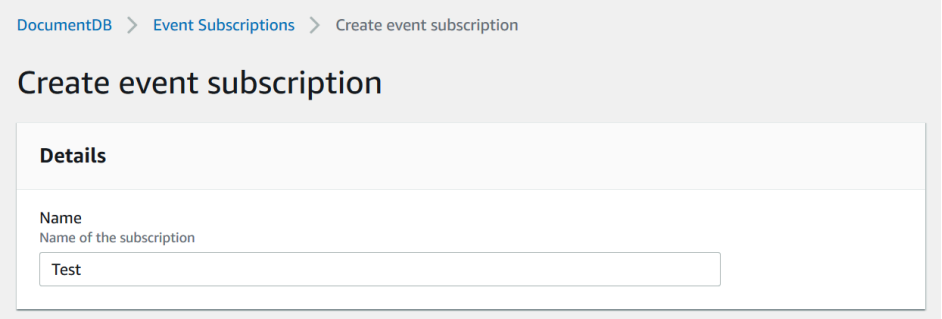


3. 이벤트 구독 창에서 이벤트 구독 생성을 선택합니다.

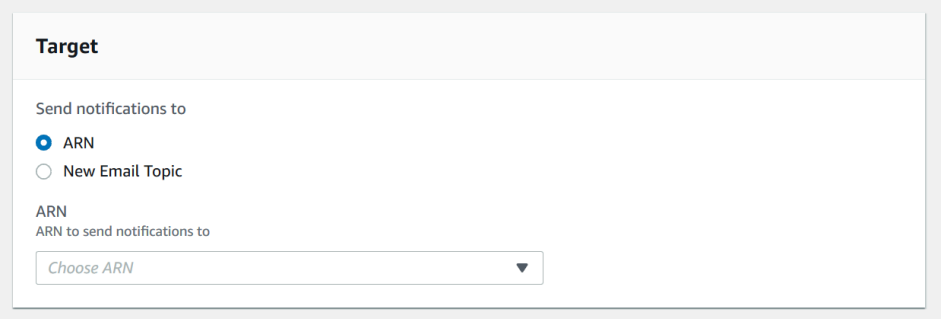


4. [Create event subscription] 대화 상자에서 다음과 같이 실행합니다.

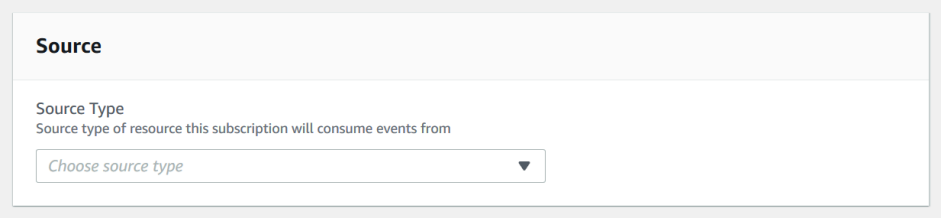
   - 이름에서 이벤트 알림 구독 이름을 입력합니다.



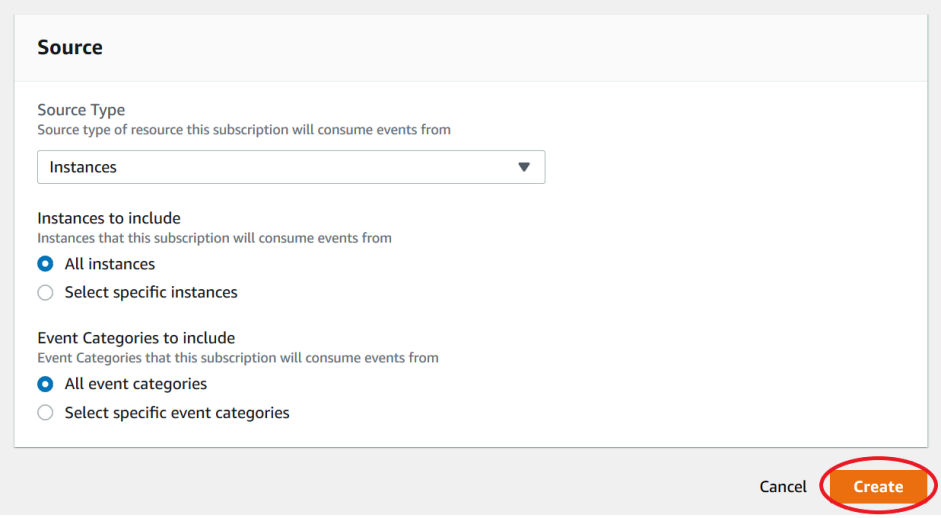
   - 대상에서, 알림을 전송할 위치를 선택합니다. 기존의 ARN을 선택하거나 새 이메일 토픽을 선택하여 주제의 이름과 수신인 목록을 입력할 수 있습니다.



- 소스에서 소스 유형을 선택합니다. 선택한 원본 유형에 따라 이벤트 범주와 이벤트 알림을 수신할 원본을 선택합니다.



- 생성(Create)을 선택합니다.



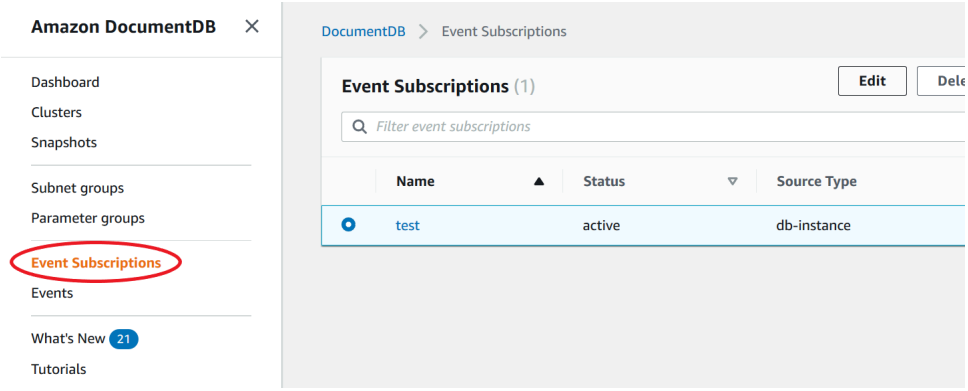
Amazon DocumentDB 이벤트 알림 구독 관리

Amazon DocumentDB 콘솔의 탐색 창에서 이벤트 구독 을 선택하면 구독 범주와 현재 구독 목록을 볼 수 있습니다. 특정 구독을 수정하거나 삭제할 수도 있습니다.

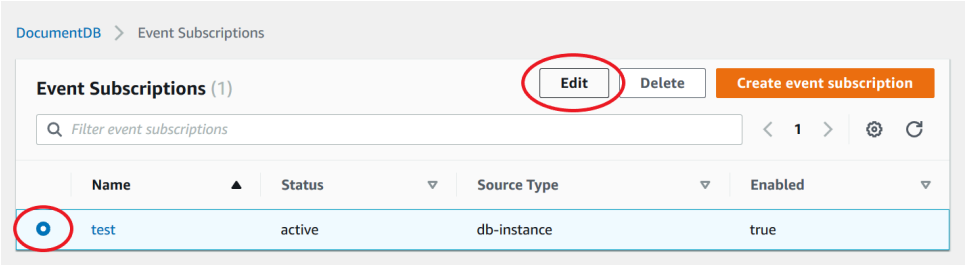
현재 Amazon DocumentDB 이벤트 알림 구독을 수정하려면 다음과 같이 하세요

1. 에서에 로그인 AWS Management Console 합니다https://console.aws.amazon.com/docdb.

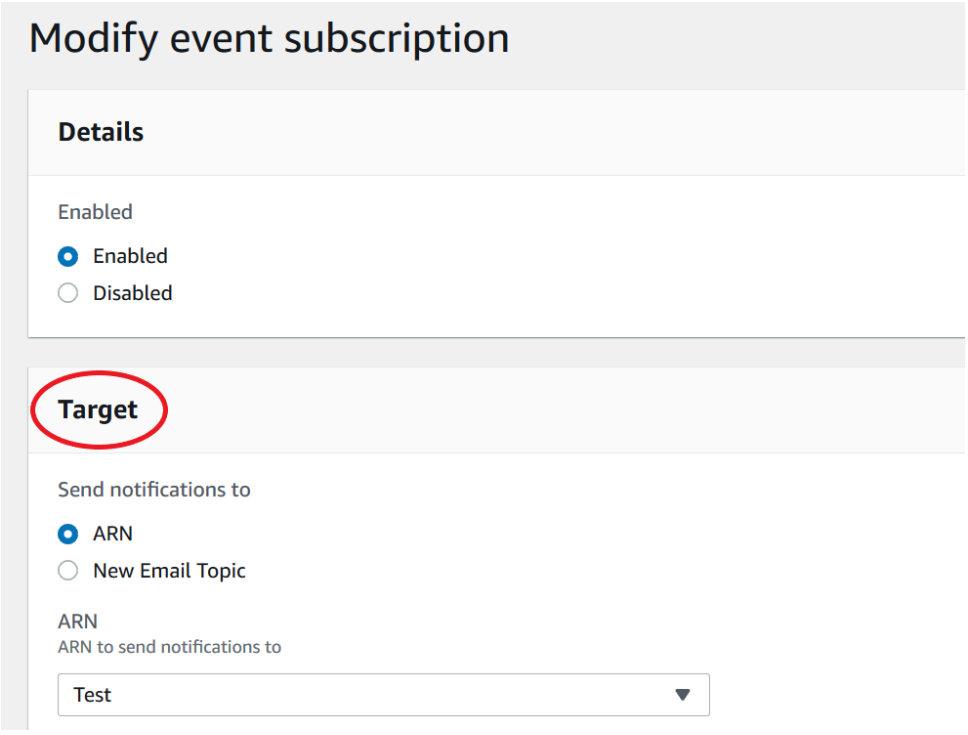
2. 탐색 창에서 [Event subscriptions]를 선택합니다. [Event subscriptions] 창에 이벤트 알림 구독이 모두 표시됩니다.



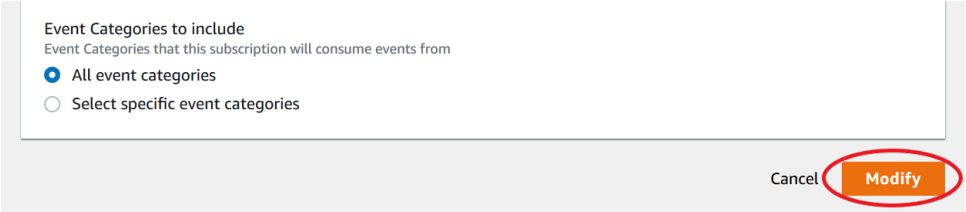
3. [Event subscriptions] 창에서 수정할 구독을 선택한 다음 [Edit]를 선택합니다.



4. 대상 또는 원본 섹션에서 구독을 변경합니다. 소스 섹션에서 소스 식별자를 선택하거나 선택 취소하여 소스 식별자를 추가하거나 제거할 수 있습니다.



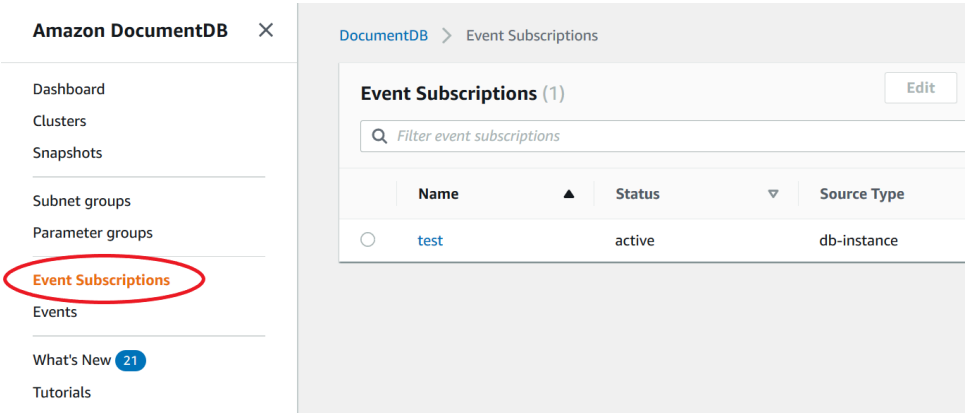
5. 수정을 선택합니다. Amazon DocumentDB 콘솔은 구독이 수정되고 있음을 나타냅니다.



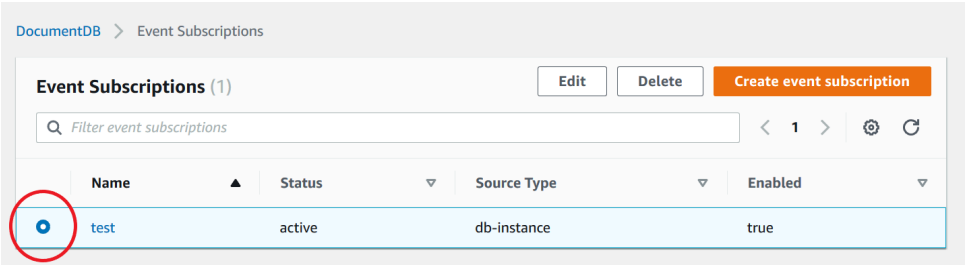
Amazon DocumentDB 이벤트 알림 구독 삭제

필요 없는 구독은 삭제할 수 있습니다. 그러면 해당 주제의 모든 구독자에게는 구독 시 지정한 이벤트 알림 메시지가 발송되지 않습니다.

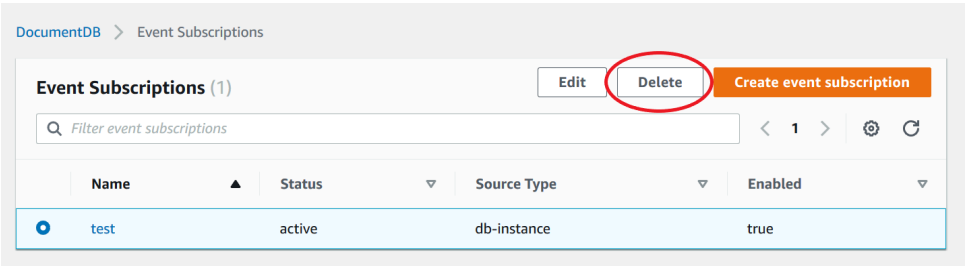
1. 에서에 로그인 AWS Management Console 합니다https://console.aws.amazon.com/docdb.
2. 탐색 창에서 [이벤트 구독]을 선택합니다.



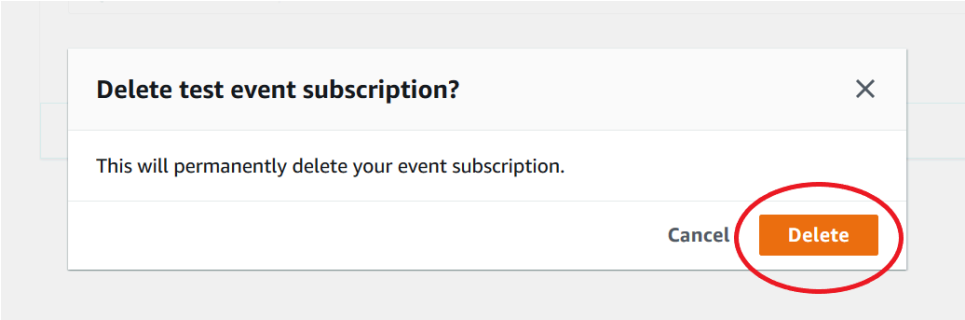
3. 이벤트 구독 패널에서 삭제할 구독을 선택합니다.



4. 삭제를 선택합니다.



5. 이 알림을 영구적으로 삭제할지 묻는 팝업 창이 나타납니다. 삭제를 선택합니다.



Amazon DocumentDB 이벤트 범주 및 메시지

Amazon DocumentDB는 콘솔을 사용하여 가입할 수 있는 카테고리에서 상당한 수의 이벤트를 생성합니다. 각 범주는 인스턴스, 클러스터, 스냅샷 또는 매개 변수 그룹이 될 수 있는 소스 유형에 적용됩니다.

> **Note**
>
> Amazon DocumentDB는 기존의 Amazon RDS 이벤트 정의와 ID를 사용합니다.

인스턴스에서 발생한 Amazon DocumentDB 이벤트

범주	설명
가용성	인스턴스 재시작
가용성	인스턴스 셧다운.
구성 변경	인스턴스 클래스에 수정을 적용하는 중입니다.
구성 변경	인스턴스 클래스에 수정을 적용했습니다.
구성 변경	기본 자격 증명을 재설정합니다.
생성	인스턴스가 생성되었습니다.
삭제	인스턴스가 삭제되었습니다.

범주	설명
실패	호환되지 않는 구성 또는 기본 스토리지 문제로 인해 인스턴스가 실패했습니다. 인스턴스에 대한 시점 복원을 시작합니다.
알림	인스턴스 멈춤.
알림	인스턴스 시작.
알림	인스턴스가 중지된 최대 허용 시간을 초과하여 시작되고 있습니다.
복구	인스턴스 복구가 시작되었습니다. 복구 시간은 복구할 데이터 용량에 따라 달라집니다.
복구	인스턴스 복구가 완료되었습니다.
보안 패치	인스턴스에 대해 운영 체제 업데이트를 사용할 수 있습니다. 업데이트에 적용에 대한 자세한 내용은 Amazon DocumentDB 유지 관리 섹션을 참조하세요.

클러스터에서 발생하는 Amazon DocumentDB 이벤트

범주	설명
생성	클러스터가 생성됨.
삭제	클러스터가 삭제되었습니다.
장애 조치	이전 프라이머리를 다시 추진합니다.
장애 조치	인스턴스로의 장애 조치가 완료되었습니다.
장애 조치	DB 인스턴스에 대한 장애 조치 시작: %초
장애 조치	DB 인스턴스에 대한 동일한 AZ 장애 조치 시작: %초

범주	설명
장애 조치	DB 인스턴스에 대한 동일한 AZ 장애 조치 시작: %초
유지 관리	클러스터가 패치되었습니다.
유지 관리	데이터베이스 클러스터를 업그레이드할 수 없는 상태: %초
알림	클러스터가 중지되었습니다.
알림	클러스터가 시작되었습니다.
알림	클러스터 중지에 실패했습니다.
알림	클러스터가 중지되는 최대 허용 시간을 초과하여 시작되고 있습니다.
알림	클러스터 이름이 %초에서 %초로 변경되었습니다.

클러스터 스냅샷에서 발생한 Amazon DocumentDB 이벤트

다음 표는 Amazon DocumentDB 클러스터 스냅샷이 원본 유형인 경우 이벤트 범주와 이벤트 목록을 보여줍니다.

범주	설명
백업	수동 클러스터 스냅샷을 생성하는 중입니다.
백업	수동 클러스터 스냅샷이 생성되었습니다.
백업	자동화된 클러스터 스냅샷을 생성하는 중입니다.
백업	자동화된 클러스터 스냅샷이 생성되었습니다.

파라미터 그룹에서 발생하는 Amazon DocumentDB 이벤트

다음 표는 파라미터 그룹이 소스 유형일 때 이벤트 카테고리와 이벤트 목록을 나타냅니다.

범주	설명
구성 변경	적용 방법 %초를 사용하여 매개 변수 %초를 %초로 업데이트했습니다

CloudWatch를 사용하여 Amazon DocumentDB 모니터링

Amazon DocumentDB(MongoDB 호환)는 Amazon CloudWatch와 통합되므로 클러스터의 운영 지표를 수집하고 분석할 수 있습니다. CloudWatch 콘솔, Amazon DocumentDB 콘솔, AWS Command Line Interface (AWS CLI) 또는 CloudWatch API를 사용하여 이러한 지표를 모니터링할 수 있습니다.

또한 CloudWatch를 사용하면 지표 값이 지정한 임계값을 위반할 경우 알림을 받을 수 있도록 알람을 설정할 수 있습니다. 위반이 발생할 경우 수정 조치를 취하기 위해 Amazon CloudWatch Events를 설정할 수도 있습니다. CloudWatch 및 알람 사용에 대한 자세한 내용은 Amazon CloudWatch 설명서를 참조하세요.

주제

- Amazon DocumentDB 지표
- CloudWatch 데이터 보기
- Amazon DocumentDB 차원
- opcounter 지표 모니터링
- 데이터베이스 연결 모니터링

Amazon DocumentDB 지표

Amazon DocumentDB 클러스터 및 인스턴스의 상태 및 성능을 모니터링하려면 Amazon DocumentDB 콘솔에서 다음 지표를 볼 수 있습니다.

> **Note**
>
> 다음 표의 지표는 인스턴스 기반 클러스터와 탄력적 클러스터 모두에 적용됩니다.

주제

- 리소스 사용률 지표
- 지연 시간 지표
- NVMe 지원 인스턴스 지표
- 운영 지표
- 명령 수준 지표
- 처리량 지표
- 시스템 지표
- T3 인스턴스 지표

리소스 사용률 지표

지표	설명	
`BackupRetentionPeriodStorageUsed`	Amazon DocumentDB의 보존 기간 내에서 시점 복원 기능을 지원하는 데 사용되는 총 백업 저장소 양(바이트)입니다. `TotalBackupStorageBilled` 지표를 통해 보고되는 총계에 포함됩니다. 각 Amazon DocumentDB 클러스터마다 개별적으로 계산됩니다.	
`ChangeStreamLogSize`	클러스터에서 변경 스트림 로그를 저장하는 데 사용하는 스토리지의 양(MB)입니다. 이 값은 클러스터의 전체 스토리지 하위 집합(`VolumeBytesUsed`)이며 클러스터 비용에 영향을 줍니다. 스토리지 요금 정보는 Amazon DocumentDB 제품 페이지를 참조하세요. 변경 스트림 로그	

지표	설명
	크기는 클러스터에서 발생하는 변경 양과 변경 스트림 장기 보존 기간의 함수입니다. 변경 스트림에 대한 자세한 내용은 Amazon DocumentDB에서 변경 스트림 사용 섹션을 참조하세요.
CPUUtilization	인스턴스의 CPU 사용률
DatabaseConnections	1분 간격으로 인스턴스에서 열린 연결 수(활성 및 유휴)입니다.
DatabaseConnectionsMax	1분 동안 인스턴스당 열려 있는 최대 데이터베이스 연결 수(활성 및 유휴)입니다.
DatabaseConnectionsLimit	지정된 시간에 인스턴스에 허용되는 최대 동시 데이터베이스 연결 수(활성 및 유휴)입니다.
DatabaseConnectionsActiveMax	1분 동안 인스턴스의 최대 활성 데이터베이스 연결 수입니다. 활성 연결은 유휴 연결이 아닌 현재 작업을 실행하는 연결입니다. 인스턴스 유형당 활성 연결 할당량은 섹션을 참조하세요인스턴스 할당량.
DatabaseCursors	1분 간격으로 인스턴스에서 열린 커서 수입니다.
DatabaseCursorsMax	1분 동안 인스턴스에서 열린 최대 커서 수입니다.

지표	설명
DatabaseCursorsLimit	지정된 시간에 인스턴스에 허용되는 최대 커서 수입니다.
DatabaseCursorsTimedOut	1분 동안 제한 시간이 초과된 커서 수입니다.
FreeableMemory	사용 가능한 RAM 크기(바이트).
FreeLocalStorage	이 지표는 각 인스턴스에서 임시 테이블 및 로그로 사용할 수 있는 스토리지 크기를 보고합니다. 이 값은 인스턴스 클래스에 따라 달라집니다. 인스턴스 클래스를 큰 것으로 선택하면 인스턴스의 여유 스토리지 공간을 늘릴 수 있습니다. (DocumentDB Serverless에는 적용되지 않습니다.)
LowMemThrottleQueueDepth	가용 메모리 부족으로 인해 1분 간격으로 제한 현상이 발생한 요청의 대기열 길이입니다.
LowMemThrottleMaxQueueDepth	가용 메모리 부족으로 인해 1분 동안 제한 현상이 발생한 요청의 최대 대기열 깊이입니다.
LowMemNumOperationsThrottled	가용 메모리 부족으로 인해 1분 동안 제한 현상이 발생한 요청 수입니다.

지표	설명
SnapshotStorageUsed	백업 보존 기간을 벗어난 특정 Amazon DocumentDB 클러스터에 대해 모든 스냅샷에서 사용한 백업 저장소의 총 양(바이트)입니다. TotalBackupStorageBilled 지표를 통해 보고되는 총계에 포함됩니다. 각 Amazon DocumentDB 클러스터마다 개별적으로 계산됩니다.
SwapUsage	인스턴스에서 사용된 스왑 공간 크기
TotalBackupStorageBilled	지정된 Amazon DocumentDB 클러스터에 대해 청구되는 총 백업 저장소 양(바이트)입니다. BackupRetentionPeriodStorageUsed 및 SnapshotStorageUsed 지표로 측정되는 백업 스토리지를 포함합니다. 각 Amazon DocumentDB 클러스터마다 개별적으로 계산됩니다.
TransactionsOpen	1분 간격으로 인스턴스에서 열린 트랜잭션 수입니다.
TransactionsOpenMax	1분 동안 인스턴스에서 열린 최대 트랜잭션 수입니다.
TransactionsOpenLimit	지정된 시간에 인스턴스에 허용되는 최대 동시 트랜잭션 수입니다.

지표	설명
VolumeBytesUsed	클러스터가 사용하는 스토리지의 양(바이트)입니다. 이 값은 클러스터의 비용에 영향을 미칩니다. 요금 정보는 Amazon DocumentDB 제품 페이지를 참조하세요.

지연 시간 지표

지표	설명
DBClusterReplicaLagMaximum	클러스터의 기본 인스턴스와 각 Amazon DocumentDB 인스턴스 간의 최대 지연 시간(밀리초)입니다.
DBClusterReplicaLagMinimum	기본 인스턴스와 클러스터의 각 복제본 인스턴스 사이에 발생하는 최소 지연 시간(밀리초)
DBInstanceReplicaLag	기본 인스턴스에서 복제본 인스턴스로 업데이트를 복제할 때의 지연 시간(밀리초)
ReadLatency	디스크 I/O 연산당 평균 처리 시간입니다.
WriteLatency	디스크 I/O 연산당 평균 처리 시간(밀리초)

NVMe 지원 인스턴스 지표

지표	설명
NVMeStorageCacheHitRatio	계층형 캐시에서 처리하는 요청 비율입니다.
FreeNVMeStorage	사용 가능한 임시 NVMe 스토리지 크기입니다.
ReadIOPSNVMeStorage	임시 NVMe 스토리지에 대한 평균 디스크 읽기 I/O 작업 수입니다.
ReadLatencyNVMeStorage	임시 NVMe 스토리지의 디스크 읽기 I/O 작업당 소요된 평균 시간입니다.
ReadThroughputNVMeStorage	임시 NVMe 스토리지의 초당 디스크에서 읽은 평균 바이트 수입니다.
WriteIOPSNVMeStorage	임시 NVMe 스토리지에 대한 평균 디스크 쓰기 I/O 작업 수입니다.
WriteLatencyNVMeStorage	임시 NVMe 스토리지의 디스크 쓰기 I/O 작업당 소요된 평균 시간입니다.
WriteThroughputNVMeStorage	임시 NVMe 스토리지에 디스크에 쓰여진 초당 평균 바이트 수입니다.

운영 지표

지표	설명
DocumentsDeleted	1분 동안 삭제된 문서 수입니다.
DocumentsInserted	1분 동안 삽입된 문서 수입니다.
DocumentsReturned	1분 동안 반환된 문서 수입니다.
DocumentsUpdated	1분 동안 업데이트된 문서 수입니다.
OpcountersCommand	1분 동안 실행된 명령 수입니다.
OpcountersDelete	1분 동안 실행된 삭제 작업 수입니다.
OpcountersGetmore	1분 동안 실행된 겟모어(getmore) 수입니다.
OpcountersInsert	1분 동안 실행된 삽입 작업 수입니다.
OpcountersQuery	1분 동안 실행된 쿼리 수입니다.
OpcountersUpdate	1분 동안 실행된 업데이트 작업 수입니다.
TransactionsStarted	1분 동안 인스턴스에서 시작된 트랜잭션 수입니다.
TransactionsCommitted	1분 동안 인스턴스에서 커밋된 트랜잭션 수입니다.

지표	설명
TransactionsAborted	1분 동안 인스턴스에서 중단된 트랜잭션 수입니다.
TTLDeletedDocuments	1분 동안 TTLMonitor에서 삭제한 문서 수입니다.

명령 수준 지표

Amazon DocumentDB는 명령당 지연 시간 및 동시성 지표를 CloudWatch에 게시합니다. 이러한 지표는 워크로드에 가장 큰 영향을 미치는 작업을 식별하고 테일 지연 시간 문제 또는 동시성 압력을 감지하는 데 도움이 됩니다.

다음 표의 각 명령은 1분 동안 세 가지 지표를 내보냅니다.

- AvgDuration - 명령의 평균 실행 시간으로, 마이크로초 단위입니다.
- P100Duration - 마이크로초 단위의 명령의 최대 실행 시간(100번째 백분위수)입니다.
- MaxConcurrent - 명령의 최대 동시 실행 수입니다.

지표 이름은 패턴을 따릅니다Command.*commandName*.*metricName*. 예: Command.find.AvgDuration 또는 Command.aggregate.MaxConcurrent.

지표	설명
Command.find	찾기를 사용하여 쿼리를 읽습니다.
Command.insert	문서 삽입 작업.
Command.findAndModify	원자성 find-and-modify 작업.
Command.update	문서 업데이트 작업.
Command.delete	문서 삭제 작업.
Command.aggregate	집계 파이프라인 실행.

지표	설명
Command.count	개수 작업.
Command.distinct	고유한 값 쿼리입니다.
Command.getMore	커서 배치 가져오기 작업.
Command.abortTransaction	트랜잭션 중단 작업.
Command.commitTransaction	트랜잭션 커밋 작업.

처리량 지표

지표	설명
NetworkReceiveThroughput	클러스터의 인스턴스 하나가 클라이언트에서 수신하는 네트워크 처리량(bps) 이 처리량에서 클러스터의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.
NetworkThroughput	Amazon DocumentDB 클러스터의 각 인스턴스에서 클라이언트에서 수신 및 클라이언트로 전송된 네트워크 처리량(초당 바이트)입니다. 이 처리량에서 클러스터의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.
NetworkTransmitThroughput	클러스터의 인스턴스 하나가 클라이언트로 전송하는 네트워크 처리량(bps). 이 처리량에서 클러스터의 인스턴스와 클러스

지표	설명
	터 볼륨 간 네트워크 트래픽은 제외됩니다.
ReadIOPS	초당 평균 디스크 읽기 I/O 연산 수 Amazon DocumentDB는 1분 간격으로 IOPS에 대한 읽기 및 쓰기를 별도로 보고합니다.
ReadThroughput	초당 디스크에서 읽은 평균 바이트 수입니다.
StorageNetworkReceiveThroughput	클러스터의 각 인스턴스에 의해 Amazon DocumentDB 클러스터 스토리지 볼륨에서 수신한 네트워크 처리량(초당 바이트)입니다.
StorageNetworkTransmitThroughput	클러스터의 각 인스턴스에 의해 Amazon DocumentDB 클러스터 스토리지 볼륨에서 전송한 네트워크 처리량(초당 바이트)입니다.
StorageNetworkThroughput	Amazon DocumentDB 클러스터의 각 인스턴스에 의해 Amazon DocumentDB 클러스터 스토리지 볼륨에서 수신 및 전송한 네트워크 처리량(초당 바이트)입니다.

지표	설명	
VolumeReadIOPs	5분마다 보고되는 클러스터 볼륨에서 요금이 청구된 읽기 I/O 작업의 평균 수. 요금이 청구된 읽기 작업은 클러스터 볼륨 수준에서 계산되며, 클러스터의 모든 인스턴스에 대해 집계된 후 5분 간격으로 보고됩니다. 이 값은 5분 동안의 읽기 작업 측정치 값을 사용하여 계산됩니다. 요금 부과된 읽기 작업 측정치의 값을 300로 나눠서 초당 요금 부과된 읽기 작업의 양을 확인할 수 있습니다. 예를 들어, VolumeReadIOPs 이 13,686을 반환하면 초당 청구된 읽기 작업은 45입니다(13,686 / 300 = 45.62). 버퍼 캐시에 없는 데이터베이스 페이지를 요청하는 쿼리에 대해서는 요금 부과된 읽기 작업이 발생하므로 스토리지에서 로드해야 합니다. 쿼리 결과를 스토리지에서 읽은 후 버퍼 캐시로 로드하면 요금 부과된 작업이 급증할 수 있습니다.	

지표	설명	
VolumeWriteIOPs	5분마다 보고되는 클러스터 볼륨에서 요금이 청구된 쓰기 I/O 작업의 평균 수. 요금이 청구된 쓰기 작업은 클러스터 볼륨 수준에서 계산되며, 클러스터의 모든 인스턴스에 대해 집계된 후 5분 간격으로 보고됩니다. 이 값은 5분 동안의 쓰기 작업 측정치 값을 사용하여 계산됩니다. 요금이 청구된 쓰기 작업 측정치의 값을 300초로 나눠서 초당 요금이 청구된 쓰기 작업의 양을 확인할 수 있습니다. 예를 들어 VolumeWriteIOPs 이 13,686을 반환하면 초당 청구된 쓰기 작업은 45입니다(13,686 / 300 = 45.62). 참고로 VolumeReadIOPs 및 VolumeWriteIOPs 지표는 DocumentDB 스토리지 계층에서 계산되며 여기에는 기본 및 복제 인스턴스에서 수행한 IO가 포함됩니다. 데이터는 20-30분마다 집계된 후 5분 간격으로 보고되므로 일정 기간 동안 지표에 대해 동일한 데이터 지점이 생성됩니다. 1분 간격으로 삽입 작업과 상호 연관시킬 지표를 찾으려면 인스턴스 수준 WriteiOps 측정치를 사용할 수 있습니다. 지표는 Amazon DocumentDB 기본 인스턴스의	

지표	설명
	모니터링 탭에서 사용할 수 있습니다.
WriteIOPS	초당 평균 디스크 쓰기 I/O 연산 수. 클러스터 수준에서 사용할 경우 WriteIOPs 은 클러스터의 모든 인스턴스에서 평가됩니다. 1분의 간격을 두고 읽기 IOPS와 쓰기 IOPS를 따로 보고합니다.
WriteThroughput	초당 디스크에 쓴 평균 바이트 수.

시스템 지표

지표	설명
AvailableMVCCIds	0에 도달하기 전에 사용 가능한 나머지 쓰기 작업 수를 보여주는 카운터입니다. 이 카운터가 0에 도달하면 ID가 회수되고 재활용될 때까지 클러스터가 읽기 전용 모드로 전환됩니다. 카운터는 각 쓰기 작업에 따라 감소하고 폐영역 회수가 이전 MVCC ID를 재활용함에 따라 증가합니다.
BufferCacheHitRatio	버퍼 캐시에서 처리하는 요청 비율입니다.
DiskQueueDepth	디스크에 쓰기 위해 또는 디스크에서 읽기 위해 대기 중인 I/O 작업 수입니다.

지표	설명
EngineUptime	인스턴스 실행 시간(초)
IndexBufferCacheHitRatio	버퍼 캐시에서 제공되는 인덱스 요청의 백분율입니다. 인덱스, 컬렉션 또는 데이터베이스를 삭제한 직후 지표가 100% 이상 급증할 수 있습니다. 이는 60초 후에 자동으로 수정됩니다. 향후 패치 업데이트에서 이러한 제한 사항이 수정될 예정입니다.
LongestActiveGCRuntime	가장 긴 활성 폐영역 회수 프로세스의 기간(초)입니다. 1분마다 업데이트하고 1분 내에 완료되는 프로세스를 제외한 활성 작업만 추적합니다.

T3 인스턴스 지표

지표	설명
CPUCreditUsage	측정 기간 동안 사용된 CPU 크레딧 수.
CPUCreditBalance	인스턴스에서 발생한 CPU 크레딧 수. CPU에서 버스트가 발생하고 CPU 크레딧이 획득 속도보다 빠르게 소비될 때 크레딧 밸런스가 고갈됩니다.
CPUSurplusCreditBalance	CPUCreditBalance 값이 0일 때 CPU 성능을 유지하기 위해 사용된 잉여 CPU 크레딧의 수입니다.

지표	설명	
CPUSurplusCreditsCharged	잉여 CPU 크레딧 수가 24시간 동안 얻을 수 있는 최대 CPU 크레딧 수를 초과하므로 추가 요금이 발생합니다. 자세한 내용은 CPU 크레딧 모니터링을 참조하세요.	

CloudWatch 데이터 보기

Amazon CloudWatch DocumentDB 콘솔, AWS Command Line Interface (AWS CLI) 또는 CloudWatch API를 사용하여 Amazon CloudWatch 데이터를 볼 수 있습니다. Amazon DocumentDB

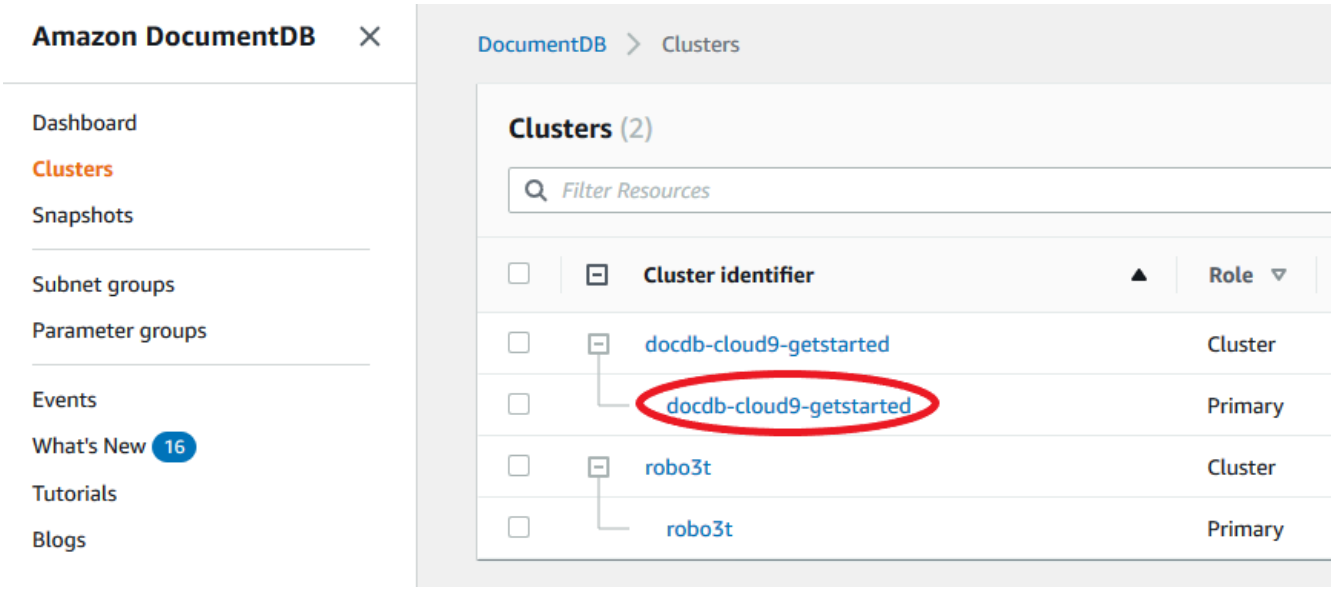
Using the AWS Management Console

Amazon DocumentDB 관리 콘솔을 사용하여 CloudWatch 지표를 보려면 다음 단계를 완료합니다.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

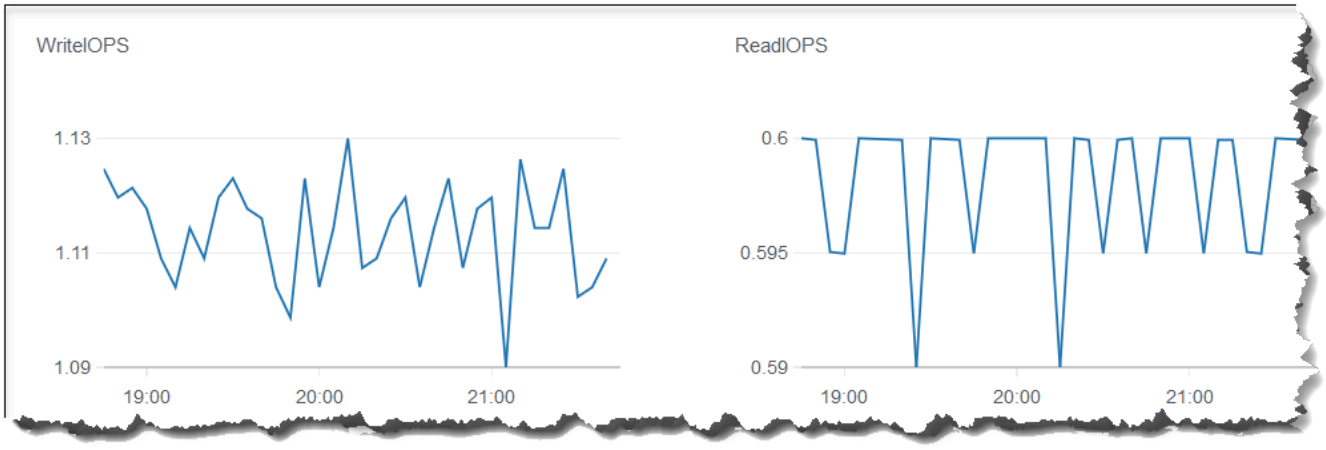
   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

3. 클러스터 탐색 상자에 클러스터 식별자 열이 표시됩니다. 인스턴스는 다음 스크린샷과 마찬가지로 클러스터 아래에 나열됩니다.



4. 인스턴스 목록에서 지표를 사용할 인스턴스의 이름을 선택합니다.
5. 결과 인스턴스 요약 페이지에서 모니터링 탭을 선택하여 Amazon DocumentDB 인스턴스의 지표를 그래픽으로 표시합니다. 각 지표에 대한 그래픽이 생성되기 전에 CloudWatch 그래프를 채우는 데 몇 분 정도 걸릴 수 있습니다.

   다음 이미지는 Amazon DocumentDB 콘솔, `WriteIOPS` 및 `ReadIOPS`에 있는 두 개의 CloudWatch 지표를 그래픽으로 표현한 것입니다.



Using the CloudWatch Management Console

CloudWatch 관리 콘솔을 사용하여 CloudWatch 지표를 보려면 다음 단계를 수행합니다.

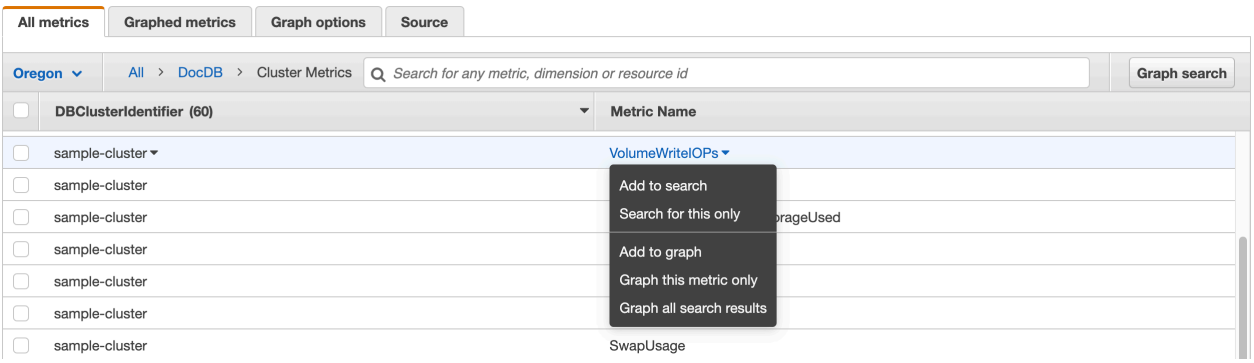
1. 에 로그인 AWS Management Console하고에서 Amazon DocumentDB 콘솔을 엽니다https://console.aws.amazon.com/cloudwatch.

2. 탐색 창에서 지표를 선택합니다. 그런 다음 서비스 이름 목록에서 DocDB를 선택합니다.
3. 지표 차원(예: 클러스터 지표)을 선택합니다.
4. 모든 지표 탭은 DocDB에서 해당 차원의 모든 지표를 표시합니다.

   a. 테이블을 정렬하려면 열 머리글을 사용합니다.

   b. 측정치를 그래프로 표시하려면 측정치 옆에 있는 확인란을 선택합니다. 모든 지표를 선택하려면 테이블의 머리글 행에 있는 확인란을 선택합니다.

   c. 지표별로 필터링하려면 지표 이름을 마우스로 가리키고 지표 이름 옆의 드롭다운 화살표를 선택합니다. 그런 다음 다음 이미지와 같이 검색에 추가를 선택합니다.



Using the AWS CLI

Amazon DocumentDB용 CloudWatch 데이터를 보려면 다음 파라미터를 사용하여 CloudWatch `get-metric-statistics` 작업을 수행합니다.

파라미터

- **`--namespace`** - 필수입니다. CloudWatch 지표를 원하는 서비스 네임스페이스. Amazon DocumentDB의 경우, 이것은 `AWS/DocDB`이어야 합니다.
- **`--metric-name`** - 필수입니다. 데이터를 원하는 지표의 이름.
- **`--start-time`** - 필수입니다. 반환할 첫 번째 데이터 포인트를 결정하는 타임스탬프입니다.

  지정된 값이 포함되므로, 지정된 타임스탬프를 가진 데이터 포인트가 결과에 포함됩니다. 타임스탬프는 ISO 8601 UTC 형식(예: 2016-10-03T23:00:00Z)이어야 합니다.
- **`--end-time`** - 필수입니다. 반환할 마지막 데이터 포인트를 결정하는 타임스탬프입니다.

  지정된 값이 포함되므로, 지정된 타임스탬프를 가진 데이터 포인트가 결과에 포함됩니다. 타임스탬프는 ISO 8601 UTC 형식(예: 2016-10-03T23:00:00Z)이어야 합니다.

- **--period** - 필수입니다. 반환되는 데이터 포인트의 세부 수준(초)입니다. 일반 분해능 지표의 경우 기간은 최소 1분(60초)이고 60의 배수여야 합니다. 1분 미만의 간격으로 수집되는 고분해능 지표의 경우 기간은 1, 5, 10, 30, 60 또는 60의 배수입니다.
- **--dimensions** - 선택 사항. 지표에 여러 차원이 포함된 경우 각 차원에 대한 값을 포함해야 합니다. CloudWatch는 각각의 고유한 차원의 조합을 별도의 지표로 처리합니다. 특정 차원 조합이 게시되지 않은 경우 해당 조합에 대한 통계를 검색할 수 없습니다. 지표 생성 시 사용된 것과 동일한 차원을 지정해야 합니다.
- **--statistics** - 선택 사항. 백분위수 이외의 지표 통계입니다. 백분위수 통계의 경우 `ExtendedStatistics` 섹션을 참조하세요. `GetMetricStatistics`를 호출할 경우 `Statistics` 또는 `ExtendedStatistics` 중 하나만 지정해야 합니다.

  허용되는 값:

  - `SampleCount`
  - `Average`
  - `Sum`
  - `Minimum`
  - `Maximum`
- **--extended-statistics** - 선택 사항. `percentile` 통계입니다. p0.0~p100 사이의 값을 지정합니다. `GetMetricStatistics`를 호출할 경우 `Statistics` 또는 `ExtendedStatistics` 중 하나만 지정해야 합니다.
- **--unit** - 선택 사항. 주어진 지표의 단위입니다. 지표가 여러 단위로 보고될 수 있습니다. 단위 결과가 반환 중인 모든 단위로 제공되지 않습니다. 지표가 보고되지 않는 단위만 지정한 경우 호출 결과는 null입니다.

  가능한 값은 다음과 같습니다.

  - `Seconds`
  - `Microseconds`
  - `Milliseconds`
  - `Bytes`
  - `Kilobytes`
  - `Megabytes`
  - `Gigabytes`
  - `Terabytes`

- Bits
- Kilobytes
- Megabits
- Gigabits
- Terabits
- Percent
- Count
- Bytes/Second
- Kilobytes/Second
- Megabytes/Second
- Gigabytes/Second
- Terabytes/Second
- Bits/Second
- Kilobits/Second
- Megabits/Second
- Gigabits/Second
- Terabits/Second
- Count/Second
- None

Example

다음 예에서는 60초마다 샘플을 생성하여 2시간 기간에 대한 최대 CPUUtilization을 찾습니다.

Linux, macOS, Unix의 경우:

```
aws cloudwatch get-metric-statistics \
       --namespace AWS/DocDB \
       --dimensions \
           Name=DBInstanceIdentifier,Value=docdb-2019-01-09-23-55-38 \
       --metric-name CPUUtilization \
       --start-time 2019-02-11T05:00:00Z \
       --end-time 2019-02-11T07:00:00Z \
       --period 60 \
```

```
        --statistics Maximum
```

Windows의 경우:

```
aws cloudwatch get-metric-statistics ^
       --namespace AWS/DocDB ^
       --dimensions ^
           Name=DBInstanceIdentifier,Value=docdb-2019-01-09-23-55-38 ^
       --metric-name CPUUtilization ^
       --start-time 2019-02-11T05:00:00Z ^
       --end-time 2019-02-11T07:00:00Z ^
       --period 60 ^
       --statistics Maximum
```

이 작업의 출력은 다음과 같습니다.

```
{
       "Label": "CPUUtilization",
       "Datapoints": [
           {
               "Unit": "Percent",
               "Maximum": 4.49152542374361,
               "Timestamp": "2019-02-11T05:51:00Z"
           },
           {
               "Unit": "Percent",
               "Maximum": 4.25000000000485,
               "Timestamp": "2019-02-11T06:44:00Z"
           },

           ********* some output omitted for brevity *********

           {
               "Unit": "Percent",
               "Maximum": 4.33333333331878,
               "Timestamp": "2019-02-11T06:07:00Z"
           }
       ]
   }
```

Amazon DocumentDB 차원

Amazon DocumentDB에 대한 지표는 계정 또는 작업에 대한 값으로 지정됩니다. CloudWatch 콘솔을 사용하여 다음 표의 차원 중 하나로 필터링된 Amazon DocumentDB 데이터를 검색할 수 있습니다.

차원	설명
`DBClusterIdentifier`	특정 Amazon DocumentDB 클러스터에 대해 요청하는 데이터를 필터링합니다.
`DBClusterIdentifier, Role`	특정 Amazon DocumentDB 클러스터에 대해 요청하는 데이터를 필터링하여 인스턴스 역할별 지표(쓰기/읽기)를 집계합니다. 예를 들어 클러스터에 속하는 모든 READER 인스턴스에 대한 지표를 집계할 수 있습니다.
`DBInstanceIdentifier`	특정 데이터베이스 인스턴스에 대해 요청하는 데이터를 필터링합니다.

opcounter 지표 모니터링

유휴 클러스터의 경우 Opcounter 지표는 값이 0이 아닙니다 (보통 ~50). 이는 Amazon DocumentDB가 주기적인 상태 점검, 내부 작업 및 지표 수집 작업을 수행하기 때문입니다.

데이터베이스 연결 모니터링

`db.runCommand( { serverStatus: 1 })`과 같은 데이터베이스 엔진 명령을 사용하여 연결 수를 볼 때 CloudWatch를 통해 `DatabaseConnections`에 표시된 것보다 최대 10개 더 많은 연결을 볼 수 있습니다. 이것은 Amazon DocumentDB가 `DatabaseConnections`에 설명되지 않는 주기적인 상태 검사 및 지표 수집 작업을 수행하기 때문에 발생합니다. `DatabaseConnections`는 고객 시작 연결만 나타냅니다.

를 사용하여 Amazon DocumentDB API 호출 로깅 AWS CloudTrail

Amazon DocumentDB(MongoDB 호환)는 Amazon DocumentDB(MongoDB 호환)에서 사용자, 역할 또는 AWS CloudTrail서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스와 통합됩

니다. CloudTrail은 Amazon DocumentDB 콘솔의 호출 및 Amazon DocumentDB SDK에 대한 코드 호출을 포함하여 Amazon DocumentDB에 대한 모든 AWS CLI API 호출을 이벤트로 캡처합니다. 추적을 생성하는 경우 Amazon DocumentDB에 대한 이벤트를 포함하여 CloudTrail 이벤트를 Amazon S3 버킷에 지속적으로 전달할 수 있습니다. 추적을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록에서 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 Amazon DocumentDB(MongoDB 호환)에 대한 요청, 요청이 이루어진 IP 주소, 요청자, 요청이 이루어진 시기 및 기타 세부 정보를 확인할 수 있습니다.

> **Important**
>
> 일부 관리 기능의 경우 Amazon DocumentDB는 Amazon Relational Database Service(RDS)와 공유되는 운영 기술을 사용합니다. Amazon DocumentDB 콘솔 AWS CLI및 API 호출은 Amazon RDS API에 대한 호출로 로깅됩니다.

자세한 내용은 AWS CloudTrail 사용 설명서를 AWS CloudTrail참조하세요.

CloudTrail의 Amazon DocumentDB 정보

CloudTrail은 계정을 생성할 AWS 계정 때에서 활성화됩니다. Amazon DocumentDB(MongoDB 호환)에서 활동이 발생하면 해당 활동은 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. 에서 최근 이벤트를 보고 검색하고 다운로드할 수 있습니다 AWS 계정. 자세한 설명은 CloudTrail 이벤트 기록으로 이벤트 보기를 참조하세요.

Amazon DocumentDB(MongoDB 호환)에 대한 이벤트를 AWS 계정포함하여에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 트레일을 생성하면 기본적으로 모든 AWS 리전에 트레일이 적용됩니다. 추적은 AWS 파티션에 있는 모든 리전의 이벤트를 로깅하고 지정된 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은AWS CloudTrail 사용 설명서에서 다음 주제를 참조하세요.

- 트레일 생성 개요
- CloudTrail 지원 서비스 및 통합
- CloudTrail에서 Amazon SNS 알림 구성
- 여러 리전에서 CloudTrail 로그 파일 수신
- 여러 계정에서 CloudTrail 로그 파일 수신

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 보안 인증 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 사용자 자격 증명으로 했는지 여부
- 역할 또는 페더레이션 사용자에 대한 임시 보안 인증을 사용하여 요청이 생성되었는지 여부.
- 요청이 다른 AWS 서비스에서 이루어졌는지 여부입니다.

자세한 설명은 CloudTrail userIdentity 요소를 참조하세요.

Amazon DocumentDB 작업 프로파일링

Amazon DocumentDB(MongoDB 호환)를 사용하여 클러스터에서 수행된 작업의 실행 시간 및 세부 정보를 기록할 수 있습니다. 프로파일러는 개별 쿼리 성능 및 전체 클러스터 성능을 개선하기 위해 클러스터에서 가장 느린 작업을 모니터링하는 경우에 유용합니다.

프로파일러 특성은 기본적으로 비활성화됩니다. 이 옵션을 사용하도록 설정하면 프로파일러 로그 고객 정의 임계값(예: 100ms)보다 오래 걸리는 작업을 Amazon CloudWatch Logs에 기록합니다. 로깅되는 세부 정보에는 프로파일링된 명령, 시간, 계획 요약 및 클라이언트 메타데이터가 포함됩니다. 작업이 CloudWatch 로그에 기록된 후 CloudWatch 로그 인사이트를 사용하여 Amazon DocumentDB 프로파일링 데이터를 분석, 모니터링 및 보관할 수 있습니다. 공통적인 쿼리는 공통 쿼리 섹션에 제공됩니다.

활성화되면 프로파일러는 클러스터에 있는 추가 리소스를 사용합니다. 높은 임계값(예: 500ms)으로 시작하고 값을 점진적으로 낮추어 느린 작업을 식별합니다. 임계값 50ms로 시작하면 처리량이 많은 애플리케이션의 경우 클러스터에서 성능 문제가 발생할 수 있습니다. 프로파일러는 클러스터 수준에서 사용 가능하며 클러스터의 모든 인스턴스와 데이터베이스에서 작동합니다. Amazon DocumentDB 로그는 작업을 최상의 방법으로 Amazon CloudWatch Logs에 기록합니다.

Amazon DocumentDB는 프로파일러를 활성화하기 위해 추가 요금을 부과하지 않지만 CloudWatch 로그 사용에 대한 표준 요금이 부과됩니다. CloudWatch 로그 요금에 대한 자세한 내용은 Amazon CloudWatch 요금을 참조하십시오.

주제

- 지원되는 연산자
- 제한 사항
- Amazon DocumentDB 프로파일러 활성화

- Amazon DocumentDB 프로파일러 비활성화
- 프로파일러 로그 내보내기 비활성화
- Amazon DocumentDB 프로파일러 로그에 액세스
- 공통 쿼리

지원되는 연산자

Amazon DocumentDB 프로파일러는 다음 작업을 지원합니다:

- `aggregate`
- `count`
- `delete`
- `distinct`
- `find`(OP_QUERY 및 명령)
- `findAndModify`
- `insert`
- `update`

제한 사항

슬로우 쿼리 프로파일러는 쿼리의 전체 결과 집합이 한 배치에 들어갈 수 있고 결과 집합이 16MB(최대 BSON 크기) 미만인 경우에만 프로파일러 로그를 내보낼 수 있습니다. 16MB 이상의 결과 세트는 자동으로 여러 배치로 분할됩니다.

대부분의 드라이버 또는 쉘은 작은 기본 배치 크기를 설정할 수 있습니다. 쿼리의 일부로 배치 크기를 지정할 수 있습니다. 느린 쿼리 로그를 캡처하려면 예상 결과 집합의 크기를 초과하는 배치 크기를 사용합니다. 결과 세트 크기를 잘 모르거나 차이가 있는 경우 배치 크기를 큰 숫자(예: 100k)로 설정합니다.

그러나 더 큰 배치 크기를 사용하면 응답이 클라이언트로 전송되기 전에 데이터베이스에서 더 많은 결과를 검색해야 합니다. 일부 쿼리의 경우 결과를 얻기 전에 지연이 더 길어질 수 있습니다. 전체 결과 집합을 소비할 계획이 없는 경우, 더 많은 I/O를 사용하여 쿼리를 처리하고 결과를 버릴 가능성이 있습니다.

Amazon DocumentDB 프로파일러 활성화

클러스터에서 프로파일러를 활성화하는 작업은 3단계 프로세스입니다. 모든 단계를 완료하거나 프로파일링 로그가 CloudWatch Logs로 전송되지 않도록 합니다. 프로파일러는 클러스터 수준에서 설정되며 클러스터의 모든 데이터베이스와 인스턴스에 대해 수행됩니다.

클러스터에서 프로파일러를 활성화하려면

1. 기본 클러스터 파라미터 그룹을 수정할 수 없으므로, 사용 가능한 사용자 지정 클러스터 파라미터 그룹이 있는지 확인하십시오. 자세한 내용은 Amazon DocumentDB 클러스터 파라미터 그룹 생성 섹션을 참조하십시오.
2. 사용 가능한 사용자 지정 클러스터 파라미터 그룹을 사용하여 `profiler`, `profiler_threshold_ms` 및 `profiler_sampling_rate` 파라미터를 수정합니다. 자세한 내용은 Amazon DocumentDB 클러스터 파라미터 그룹 수정 섹션을 참조하십시오.
3. 사용자 지정 클러스터 매개 변수 그룹을 사용하고 `profiler` CloudWatch Logs로 로그 내보내기를 사용하도록 클러스터를 생성하거나 수정합니다.

다음 섹션에서는 AWS Management Console 및 AWS Command Line Interface ()를 사용하여 이러한 단계를 구현하는 방법을 보여줍니다AWS CLI.

Using the AWS Management Console

1. 시작하기 전에 Amazon DocumentDB 클러스터와 사용자 정의 클러스터 파라미터 그룹이 없는 경우, 이를 생성합니다. 자세한 내용은 Amazon DocumentDB 클러스터 파라미터 그룹 생성 및 Amazon DocumentDB 클러스터 생성 섹션을 참조하십시오.
2. 사용 가능한 사용자 지정 클러스터 파라미터 그룹을 사용하여 다음 파라미터를 수정합니다. 자세한 내용은 Amazon DocumentDB 클러스터 파라미터 그룹 수정 섹션을 참조하십시오.

   - `profiler` - 쿼리 프로파일링을 활성화하거나 비활성화합니다. 허용된 값은 `enabled` 및 `disabled`입니다. 기본값은 `disabled`입니다. 프로파일링을 활성화하려면 값을 `enabled`로 설정합니다.
   - `profiler_threshold_ms` - `profiler`를 `enabled`으로 설정하면 `profiler_threshold_ms`보다 더 오래 걸리는 모든 명령이 CloudWatch 로그에 기록됩니다. 허용된 값은 `[50-INT_MAX]`입니다. 기본값은 `100`입니다.
   - `profiler_sampling_rate` - 프로파일링하거나 로깅해야 하는 느린 작업의 조각입니다. 허용된 값은 `[0.0-1.0]`입니다. 기본값은 `1.0`입니다.

3. 사용자 지정 클러스터 매개 변수 그룹을 사용하도록 클러스터를 수정하고 Amazon CloudWatch에 게시자 프로파일러 로그 내보내기를 설정합니다.

   a. 탐색 창에서 클러스터를 선택하여 사용자 지정 파라미터 그룹을 클러스터에 추가합니다.
   b. 파라미터 그룹을 연결할 클러스터의 이름 왼쪽에 있는 버튼을 선택합니다. 작업, 수정 순으로 선택하여 클러스터를 수정합니다.
   c. 클러스터 옵션의 위 단계에서 사용자 지정 파라미터 그룹을 선택하여 클러스터에 추가합니다.
   d. 로그 내보내기에서 Amazon CloudWatch에 게시자로 등록할 프로파일 로그를 선택합니다.
   e. 수정 사항의 요약을 보려면 계속을 선택합니다.
   f. 변경 사항을 확인한 후 즉시 적용하거나 수정 일정 아래의 다음 유지 관리 기간 중에 적용할 수 있습니다.
   g. 클러스터를 새 파라미터 그룹으로 업데이트하려면 클러스터 수정을 선택합니다.

Using the AWS CLI

다음 절차는 `sample-cluster` 클러스터에 지원되는 모든 작업에 대해 프로파일러를 활성화합니다.

1. 시작하기 전에 다음 명령을 실행한 후 이름에 `default`가 없고 `docdb3.6`이 파라미터 그룹 패밀리인 클러스터 파라미터 그룹의 출력을 검토하여 사용 가능한 사용자 지정 클러스터 파라미터 그룹이 있는지 확인합니다. 기본이 아닌 클러스터 파라미터 그룹이 없는 경우 Amazon DocumentDB 클러스터 파라미터 그룹 생성 단원을 참조하십시오.

   ```
   aws docdb describe-db-cluster-parameter-groups \
       --query 'DBClusterParameterGroups[*].
   [DBClusterParameterGroupName,DBParameterGroupFamily]'
   ```

   다음 출력에서는 `sample-parameter-group` 만 두 기준을 모두 충족합니다.

   ```
   [
          [
              "default.docdb3.6",
              "docdb3.6"
          ],
          [
              "sample-parameter-group",
   ```

```
            "docdb3.6"
        ]
]
```

2. 사용자 지정 클러스터 파라미터 그룹을 사용하여 다음 파라미터를 수정합니다:

   - `profiler` - 쿼리 프로파일링을 활성화하거나 비활성화합니다. 허용된 값은 `enabled` 및 `disabled`입니다. 기본값은 `disabled`입니다. 프로파일링을 활성화하려면 값을 `enabled`로 설정합니다.
   - `profiler_threshold_ms` - `profiler`를 `enabled`으로 설정하면 `profiler_threshold_ms`보다 더 오래 걸리는 모든 명령이 CloudWatch에 기록됩니다. 허용된 값은 `[50-INT_MAX]`입니다. 기본값은 `100`입니다.
   - `profiler_sampling_rate` - 프로파일링하거나 로깅해야 하는 느린 작업의 조각입니다. 허용된 값은 `[0.0-1.0]`입니다. 기본값은 `1.0`입니다.

```
aws docdb modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name sample-parameter-group \
    --parameters
 ParameterName=profiler,ParameterValue=enabled,ApplyMethod=immediate \

 ParameterName=profiler_threshold_ms,ParameterValue=100,ApplyMethod=immediate \

 ParameterName=profiler_sampling_rate,ParameterValue=0.5,ApplyMethod=immediate
```

3. 이전 단계의 `sample-parameter-group` 사용자 지정 클러스터 파라미터 그룹을 사용하고 파라미터 `--enable-cloudwatch-logs-exports`를 `profiler`으로 설정하도록 Amazon DocumentDB 클러스터를 수정합니다.

   다음 코드는 이전 단계에서 `sample-parameter-group`를 사용하도록 클러스터 `sample-cluster`을 수정하고 사용하도록 설정된 CloudWatch 로그 내보내기에 `profiler`을 추가합니다.

```
aws docdb modify-db-cluster \
       --db-cluster-identifier sample-cluster \
       --db-cluster-parameter-group-name sample-parameter-group \
       --cloudwatch-logs-export-configuration '{"EnableLogTypes":["profiler"]}'
```

   이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBCluster": {
        "AvailabilityZones": [
            "us-east-1c",
            "us-east-1b",
            "us-east-1a"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "sample-cluster",
        "DBClusterParameterGroup": "sample-parameter-group",
        "DBSubnetGroup": "default",
        "Status": "available",
        "EarliestRestorableTime": "2020-04-07T02:05:12.479Z",
        "Endpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "ReaderEndpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "MultiAZ": false,
        "Engine": "docdb",
        "EngineVersion": "3.6.0",
        "LatestRestorableTime": "2020-04-08T22:08:59.317Z",
        "Port": 27017,
        "MasterUsername": "test",
        "PreferredBackupWindow": "02:00-02:30",
        "PreferredMaintenanceWindow": "tue:09:50-tue:10:20",
        "DBClusterMembers": [
            {
                "DBInstanceIdentifier": "sample-instance-1",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            },
            {
                "DBInstanceIdentifier": "sample-instance-2",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            }
        ],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcd0123",
                "Status": "active"
            }
        ],
```

```
        "HostedZoneId": "ABCDEFGHIJKLM",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DbClusterResourceId": "cluster-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "DBClusterArn": "arn:aws:rds:us-east-1:<accountID>:cluster:sample-
cluster",
        "AssociatedRoles": [],
        "ClusterCreateTime": "2020-01-10T22:13:38.261Z",
        "EnabledCloudwatchLogsExports": [
            "profiler"
        ],
        "DeletionProtection": true
    }
}
```

Amazon DocumentDB 프로파일러 비활성화

프로파일러를 실행 중지하려면 `profiler` 매개 변수와 `profiler` 로그를 CloudWatch Logs로 내보내기를 모두 실행 중지해야 합니다.

프로파일러 비활성화

다음과 AWS CLI같이 AWS Management Console 또는를 사용하여 `profiler` 파라미터를 비활성화할 수 있습니다.

Using the AWS Management Console

다음 절차에서는를 사용하여 Amazon DocumentDB AWS Management Console 를 비활성화합니다`profiler`.

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다. 그런 다음 프로파일러를 비활성화하려는 클러스터 파라미터 그룹의 이름을 선택합니다.
3. 그 결과 표시되는 클러스터 파라미터 페이지에서 `profiler` 파라미터 왼쪽에 있는 버튼을 선택하고 편집을 선택합니다.
4. 프로파일러 수정 대화 상자의 목록에서 `disabled`를 선택합니다.
5. 클러스터 파라미터 수정을 선택합니다.

Using the AWS CLI

AWS CLI를 사용하여 클러스터에서 `profiler`를 비활성화하려면 다음과 같이 클러스터를 수정합니다.

```
aws docdb modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name sample-parameter-group \
    --parameters
 ParameterName=profiler,ParameterValue=disabled,ApplyMethod=immediate
```

프로파일러 로그 내보내기 비활성화

다음과 AWS CLI같이 AWS Management Console 또는를 사용하여 CloudWatch Logs로 `profiler` 로그 내보내기를 비활성화할 수 있습니다.

Using the AWS Management Console

다음 절차에서는를 사용하여 CloudWatch AWS Management Console 로 로그를 내보내는 Amazon DocumentDB를 비활성화합니다.

1. https://console.aws.amazon.com/docdb에서 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다. 로그 내보내기를 비활성화할 클러스터의 이름 왼쪽에 있는 버튼을 선택합니다.
3. 작업 메뉴에서 수정을 선택합니다.
4. 로그 내보내기 섹션까지 아래로 스크롤하여 프로필러 로그를 선택 취소합니다.
5. 계속을 선택합니다.
6. 변경 사항을 검토하고, 언제 이 변경 사항을 클러스터에 적용할지를 선택합니다:
   - 예약된 다음 유지 관리 기간에 적용
   - 즉시 적용
7. 클러스터 수정을 선택합니다.

Using the AWS CLI

다음 코드는 클러스터`sample-cluster` 를 수정하고 CloudWatch 프로파일러 로그를 비활성화합니다.

Example

Linux, macOS, Unix의 경우:

```
aws docdb modify-db-cluster \
   --db-cluster-identifier sample-cluster \
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["profiler"]}'
```

Windows의 경우:

```
aws docdb modify-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["profiler"]}'
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
{
    "DBCluster": {
        "AvailabilityZones": [
            "us-east-1c",
            "us-east-1b",
            "us-east-1a"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "sample-cluster",
        "DBClusterParameterGroup": "sample-parameter-group",
        "DBSubnetGroup": "default",
        "Status": "available",
        "EarliestRestorableTime": "2020-04-08T02:05:17.266Z",
        "Endpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "ReaderEndpoint": "sample-cluster.node.us-east-1.docdb.amazonaws.com",
        "MultiAZ": false,
        "Engine": "docdb",
        "EngineVersion": "3.6.0",
        "LatestRestorableTime": "2020-04-09T05:14:44.356Z",
        "Port": 27017,
        "MasterUsername": "test",
        "PreferredBackupWindow": "02:00-02:30",
        "PreferredMaintenanceWindow": "tue:09:50-tue:10:20",
        "DBClusterMembers": [
            {
                "DBInstanceIdentifier": "sample-instance-1",
```

```
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            },
            {
                "DBInstanceIdentifier": "sample-instance-2",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            }
        ],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-abcd0123",
                "Status": "active"
            }
        ],
        "HostedZoneId": "ABCDEFGHIJKLM",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:<accountID>:key/sample-key",
        "DbClusterResourceId": "cluster-ABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "DBClusterArn": "arn:aws:rds:us-east-1:<accountID>:cluster:sample-cluster",
        "AssociatedRoles": [],
        "ClusterCreateTime": "2020-01-10T22:13:38.261Z",
        "DeletionProtection": true
    }
}
```

Amazon DocumentDB 프로파일러 로그에 액세스

Amazon CloudWatch에서 프로필 로그에 액세스하려면 다음 단계를 수행합니다.

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 사용자가 Amazon DocumentDB 클러스터와 동일한 리전에 있어야 합니다.
3. 탐색 창에서 로그를 선택합니다.
4. 클러스터에 대한 프로파일러 로그를 찾으려면, 목록에서 `/aws/docdb/`*`yourClusterName`*`/profiler`을 선택합니다.

   각 인스턴스에 대한 프로파일 로그는 해당 인스턴스 이름 아래에서 사용할 수 있습니다.

공통 쿼리

다음은 프로파일링된 명령을 분석하는 데 사용할 수 있는 몇 가지 공통적인 쿼리입니다. CloudWatch Logs Insights에 대한 자세한 내용은 CloudWatch Logs Insights로 로그 데이터 분석 및 샘플 쿼리를 참조하십시오.

지정된 모음에서 가장 느린 10개 작업 가져오기

```
filter ns="test.foo" | sort millis desc | limit 10
```

모음에서 60ms보다 오래 걸린 모든 업데이트 작업 가져오기

```
filter millis > 60 and op = "update"
```

지난 달에 가장 느린 10개 작업 가져오기

```
sort millis desc | limit 10
```

COLLSCAN 계획 요약을 사용하여 모든 쿼리 가져오기

```
filter planSummary="COLLSCAN"
```

성능 개선 도우미로 모니터링

성능 개선 도우미는 기존 Amazon DocumentDB 모니터링 특성을 확장한 것으로서 데이터베이스 성능을 표시하여 성능 문제를 분석하는 데 효과적입니다. 성능 개선 도우미 대시보드를 사용하면 데이터베이스 로드를 시각화하고 대기, 쿼리문, 호스트 또는 애플리케이션별로 로드를 필터링할 수 있습니다.

> **Note**
>
> 성능 개선 도우미는 Amazon DocumentDB 3.6, 4.0, 5.0 및 8.0 인스턴스 기반 클러스터에서만 사용할 수 있습니다.

어떻게 유용할까요?

- 데이터베이스 성능 가시화 — 언제 어디에서 데이터베이스의 로드가 있는지 파악하기 위해 로드를 시각화합니다
- 데이터베이스 로드의 원인 파악 — 어떤 쿼리, 호스트, 애플리케이션이 인스턴스 로드에 영향을 미치는지 확인합니다
- 데이터베이스에 로드가 있을 때 판단 — 성능 개선 도우미 대시보드를 확대하여 특정 이벤트에 초점을 맞추거나 축소하여 더 큰 시간에 걸쳐 추세를 확인할 수 있습니다
- 데이터베이스 로드에 대한 알림 — CloudWatch에서 새로운 데이터베이스 로드 메트릭에 자동으로 액세스하여 다른 Amazon DocumentDB 메트릭과 함께 DB 로드 메트릭을 모니터링하고 해당 메트릭에 대한 알림을 설정할 수 있습니다

Amazon DocumentDB 성능 개선 도우미의 한계는 무엇입니까?

- AWS GovCloud(미국 동부) 및 AWS GovCloud(미국 서부) 리전의 성능 개선 도우미를 사용할 수 없음
- Amazon DocumentDB용 성능 개선 도우미는 최대 7일 간의 성능 데이터를 유지합니다
- 1,024바이트보다 긴 쿼리는 성능 개선 도우미에 집계되지 않습니다

주제

- 성능 개선 도우미 개념
- 성능 개선 도우미 활성화 및 비활성화
- 성능 개선 도우미에 대한 액세스 정책 구성
- 성능 개선 도우미 대시보드를 사용한 지표 분석
- 성능 개선 도우미 API를 사용하여 지표 검색
- 성능 개선 도우미를 위한 Amazon CloudWatch 지표
- 카운터 지표에 대한 성능 개선 도우미

성능 개선 도우미 개념

주제

- 평균 활성 세션
- 측정 기준
- 최대 vCPU

평균 활성 세션

데이터베이스 로드(DB 로드)는 데이터베이스의 활동 수준을 측정합니다. 성능 개선 도우미의 핵심 지표는 DB Load이며, 1초 간격으로 수집됩니다. DBLoad 메트릭의 단위는 Amazon DocumentDB 인스턴스에 대한 AAS(평균 활성 세션 수)입니다.

활성 세션은 Amazon DocumentDB 인스턴스에 작업을 제출하고 응답을 기다리는 연결입니다. 예를 들어 Amazon DocumentDB 인스턴스에 쿼리를 제출하면 인스턴스가 쿼리를 처리하는 동안 데이터베이스 세션이 활성화됩니다.

평균 활성 세션을 구하기 위해 성능 개선 도우미는 쿼리를 동시에 실행하는 세션 수를 샘플링합니다. AAS는 총 세션 수를 총 샘플 수로 나눈 값입니다. 다음 표는 실행 중인 쿼리의 연속된 5개 샘플을 보여줍니다.

샘플	쿼리를 실행 중인 세션 수	AAS	계산
1	2	2	2개 세션/1개 샘플
2	0	1	2개 세션/2개 샘플
3	4	2	6개 세션/3개 샘플
4	0	1.5	6개 세션/4개 샘플
5	4	2	10개 세션/5개 샘플

앞의 예제에서 1-5의 시간 간격에 대한 DB 로드는 2 AAS입니다. DB 로드의 증가는 평균적으로 데이터베이스에서 더 많은 세션이 실행되고 있음을 의미합니다.

측정 기준

DB Load 지표는 '차원'이라는 하위 구성 요소로 구분할 수 있기 때문에 다른 시계열 지표와 다릅니다. 차원을 DB Load 지표의 다양한 특성에 대한 범주로 생각할 수 있습니다. 성능 문제를 진단할 때 가장 유용한 차원은 대기 상태 및 상위 쿼리입니다.

대기 상태

대기 상태는 쿼리 문이 계속 실행되려면 특정 이벤트가 발생할 때까지 쿼리 문이 기다리도록 합니다. 예를 들어 잠긴 리소스의 잠금이 해제될 때까지 쿼리 문이 대기할 수 있습니다. DB Load와 대기 상태

와 결합하여 전체 세션 상태를 이해할 수 있습니다. Amazon DocumentDB의 다양한 대기 상태는 다음과 같습니다:

Amazon DocumentDB 대기 상태	대기 상태 설명
래치	래치 대기 상태는 세션이 버퍼 풀의 페이징을 기다리고 있을 때 발생합니다. 버퍼 풀에서 자주 호출되는 것은 시스템에서 처리 중인 대규모 쿼리, 수집 검색이 빈번할 때 또는 버퍼 풀이 너무 작아서 작업 세트를 처리할 수 없을 때 더 자주 발생할 수 있습니다.
CPU	CPU 대기 상태는 세션이 CPU에서 대기할 때 발생합니다.
CollectionLock	CollectionLock 대기 상태는 세션이 컬렉션에 대한 잠금을 획득하기 위해 대기할 때 발생합니다. 이러한 이벤트는 컬렉션에 DDL 작업이 있을 때 발생합니다.
DocumentLock	DocumentLock 대기 상태는 세션이 문서에 대한 잠금 획득을 대기 중일 때 발생합니다. 동일한 문서에 대한 동시 쓰기 횟수가 많으면 해당 문서의 DocumentLock 대기 상태가 늘어나는데 기여하게 됩니다.
SystemLock	SystemLock 대기 상태는 세션이 시스템에서 대기할 때 발생합니다. 이 문제는 시스템에서 자주 실행되는 쿼리, 오래 실행되는 트랜잭션 또는 높은 동시성이 있을 때 발생할 수 있습니다.
IO	IO 대기 상태는 IO에서 세션이 완료될 때 발생합니다.
BufferLock	BufferLock 대기 상태는 세션이 버퍼의 공유 페이지에서 잠금을 획득하기 위해 대기할 때 발생합니다. 다른 프로세스가 요청된 페이지에서 열

Amazon DocumentDB 대기 상태	대기 상태 설명
	린 커서를 유지하고 있는 경우 BufferLock 대기 상태가 길어질 수 있습니다.
LowMemThrottle	Amazon DocumentDB 인스턴스의 과도한 메모리 부족으로 인해 세션이 대기 중일 때 LowMemThrottle 대기 상태가 발생합니다. 이 상태가 오래 지속되면 인스턴스를 확장하여 추가 메모리를 제공하는 것을 고려합니다. 자세한 내용은 리소스 거버너를 참조하세요.
BackgroundActivity	BackgroundActivity 대기 상태는 세션이 내부 시스템 프로세스에서 대기할 때 발생합니다.
기타	기타 대기 상태는 내부 대기 상태입니다. 이 상태가 오래 지속되면 이 쿼리를 종료하는 것을 고려하십시오. 자세한 내용은 긴 실행 중이거나 차단된 쿼리를 어떻게 찾고 종료하나요?를 참조하십시오.

상위 쿼리

대기 상태는 병목 현상을 보여주는 반면 상위 쿼리는 DB 로드에 가장 큰 기여를 하는 쿼리를 보여줍니다. 예를 들어 현재 데이터베이스에서 여러 쿼리가 실행 중이더라도 단일 쿼리가 DB 로드의 99%를 소비할 수 있습니다. 이 경우 로드가 높으면 쿼리에 문제가 있음을 나타낼 수 있습니다.

최대 vCPU

대시보드에서 데이터베이스 로드 차트는 세션 정보를 수집, 집계 및 표시합니다. 활성 세션이 최대 CPU를 초과하는지 확인하려면 최대 vCPU 선과의 관계를 확인합니다. 최대 vCPU 값은 Amazon DocumentDB 인스턴스의 vCPU(가상 CPU) 코어 수에 따라 결정됩니다.

DB 로드가 최대 vCPU 선을 상회하는 경우가 잦아지고 CPU가 기본 대기 상태라면 CPU에서 과부하가 발생한 것입니다. 이 경우 인스턴스에 대한 연결을 조절하거나 CPU 로드가 큰 쿼리를 조정하거나 더 큰 인스턴스 클래스를 고려할 수 있습니다. 대기 상태의 인스턴스가 높고 일관적이라는 것은 해결해야 할 병목 현상이나 리소스 경합 문제가 있을 수 있음을 나타냅니다. DB 로드가 최대 vCPU 선을 넘지 않는다 하더라도 이러한 문제가 나타날 수 있습니다.

성능 개선 도우미 활성화 및 비활성화

성능 개선 도우미를 사용하려면 DB 인스턴스에서 이 기능을 활성화합니다. 필요한 경우 나중에 이를 비활성화할 수 있습니다. 성능 개선 도우미를 활성화하거나 비활성화해도 가동 중지, 재부팅 또는 장애 조치가 발생하지 않습니다.

성능 개선 도우미 에이전트는 DB 호스트에서 제한된 CPU 및 메모리를 사용합니다. DB 로드가 높을 경우 에이전트는 데이터 수집 빈도를 줄여 성능에 미치는 영향을 제한합니다.

클러스터 생성 시 성능 개선 도우미 활성화

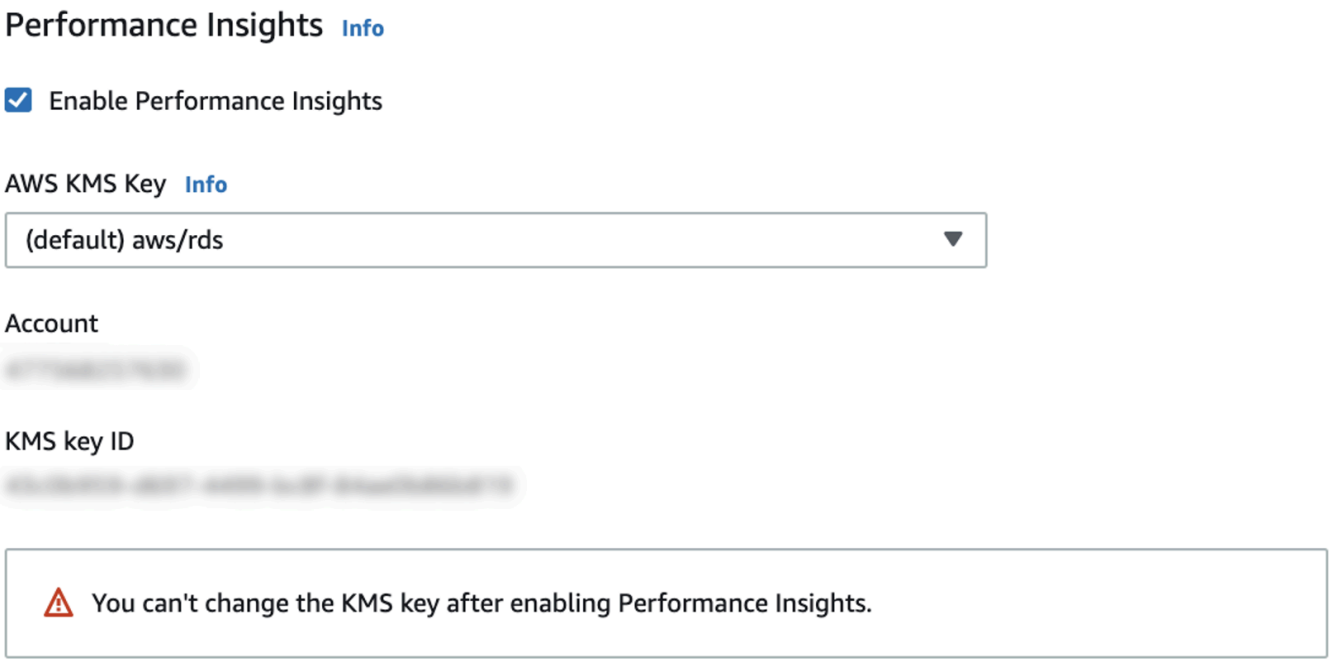
콘솔에서 새 DB 인스턴스를 생성하거나 수정할 때 성능 개선 도우미를 활성화하거나 비활성화할 수 있습니다.

사용 AWS Management Console

콘솔에서 Amazon DocumentDB 클러스터를 작성할 때 성능 개선 도우미를 활성화할 수 있습니다. 새 Amazon DocumentDB 클러스터를 작성할 때 성능 개선 도우미 섹션에서 성능 개선 도우미 활성화를 선택하여 성능 개선 도우미를 활성화합니다.

콘솔 지침

1. 클러스터를 만드는 방법에 대한 지침은 Amazon DocumentDB 클러스터 생성을 참조하세요.
2. 성능 개선 도우미 섹션에서 성능 개선 도우미 활성화를 선택합니다.



> **Note**
>
> 성능 개선 도우미 데이터 보존 기간은 7일입니다.

AWS KMS 키 - AWS KMS 키를 지정합니다. 성능 개선 도우미는 AWS KMS 키를 사용하여 잠재적으로 민감한 모든 데이터를 암호화합니다. 데이터는 암호화된 상태로 전송 및 저장됩니다. 자세한 내용은 성능 개선 도우미에 대한 AWS AWS KMS 정책 구성을 참조하세요.

인스턴스 수정 시 활성화 및 비활성화

콘솔 또는 AWS CLI을 사용하여 성능 개선 도우미를 활성화하도록 DB 인스턴스를 수정할 수 있습니다.

Using the AWS Management Console

콘솔 지침

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 클러스터를 선택하십시오.
3. DB 인스턴스를 선택하고 수정을 선택합니다.
4. 성능 개선 도우미 섹션에서 성능 개선 도우미 활성화 또는 성능 개선 도우미 비활성화를 선택합니다.

   > **Note**
   >
   > 성능 개선 도우미 활성화를 선택한 경우 AWS AWS KMS 키를 지정할 수 있습니다. 성능 개선 도우미는 AWS KMS 키를 사용하여 잠재적으로 민감한 모든 데이터를 암호화합니다. 데이터는 암호화된 상태로 전송 및 저장됩니다. 자세한 내용은 정지 상태의 Amazon DocumentDB Data 암호화를 참조하십시오.

5. Continue(계속)를 선택합니다.
6. 수정 스케줄링에 대해 즉시 적용을 선택합니다. 다음 예약된 유지 관리 기간 동안 적용을 선택하면 인스턴스는 이 설정을 무시하고 즉시 성능 개선 도우미를 활성화합니다.
7. Modify Instance(인스턴스 수정)를 선택합니다.

Using the AWS CLI

`create-db-instance` 또는 `modify-db-instance` AWS AWS CLI 명령을 사용할 때를 지정하여 성능 개선 도우미를 활성화하거나 `--enable-performance-insights`를 지정하여 비활성화할 수 있습니다`--no-enable-performance-insights`.

다음 절차에서는 AWS AWS CLI를 사용하여 DB 인스턴스의 성능 개선 도우미를 활성화 또는 비활성화하는 방법을 설명합니다.

AWS AWS CLI 지침

`modify-db-instance` AWS AWS CLI 명령을 호출하고 다음 값을 제공합니다.

- `--db-instance-identifer` — DB 인스턴스 이름.
- 활성화하려면 `--enable-performance-insights` 또는 비활성화하려면 `--no-enable-performance-insights`

Example

다음 예제에서는 `sample-db-instance`에 대한 성능 개선 도우미를 활성화합니다:

For Linux, macOS, or Unix:

```
aws docdb modify-db-instance \
    --db-instance-identifier sample-db-instance \
    --enable-performance-insights
```

For Windows:

```
aws docdb modify-db-instance ^
    --db-instance-identifier sample-db-instance ^
    --enable-performance-insights
```

성능 개선 도우미에 대한 액세스 정책 구성

성능 개선 도우미에 액세스하려면 AWS Identity and Access Management (IAM)의 적절한 권한이 있어야 합니다. 액세스 권한 부여 옵션은 다음과 같습니다:

- 권한 세트 또는 역할에 `AmazonRDSPerformanceInsightsReadOnly` 관리형 정책을 연결합니다.

- 사용자 지정 IAM 정책을 생성하고 권한 세트 또는 역할에 연결합니다.

또한 성능 개선 도우미를 활성화할 때 고객 관리형 키를 지정한 경우 계정의 사용자에게 KMS 키에 대한 `kms:Decrypt` 및 `kms:GenerateDataKey` 권한이 있는지 확인합니다.

> **Note**
>
> AWS KMS 키 및 보안 그룹 관리를 통한 encryption-at-rest를 위해 Amazon DocumentDB는 Amazon RDS와 공유되는 운영 기술을 활용합니다.

AmazonRDSPerformanceInsightsReadOnly 정책을 IAM 보안 주체에 연결

`AmazonRDSPerformanceInsightsReadOnly`는 Amazon DocumentDB 성능 개선 도우미 API의 모든 읽기 전용 작업에 대한 액세스 권한을 부여하는 AWS관리형 정책입니다. 현재 이 API의 모든 작업은 읽기 전용입니다. `AmazonRDSPerformanceInsightsReadOnly`를 권한 세트 또는 역할에 연결하면 수신자는 성능 개선 도우미를 다른 콘솔 기능과 함께 사용할 수 있습니다.

성능 개선 도우미를 위한 사용자 지정 IAM 정책 만들기

`AmazonRDSPerformanceInsightsReadOnly` 정책이 없는 사용자의 경우, 사용자 관리형 IAM 정책을 생성 또는 수정하여 성능 개선 도우미에 대한 액세스 권한을 부여할 수 있습니다. 정책을 권한 세트 또는 역할에 연결할 때 수신자는 성능 개선 도우미을 사용할 수 있습니다.

사용자 지정 정책을 생성하는 방법

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 Policies를 선택합니다.
3. 정책 생성을 선택합니다.
4. 정책 생성 페이지에서 JSON 탭을 선택합니다.
5. 다음 텍스트를 복사하여 붙여넣습니다. *us-east-1*을 AWS 리전 이름으로 바꾸고 *111122223333*을 고객 계정 번호로 바꿉니다.

   JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rds:DescribeDBInstances",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "rds:DescribeDBClusters",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:DescribeDimensionKeys",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:GetDimensionKeyDetails",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:GetResourceMetadata",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:GetResourceMetrics",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:ListAvailableResourceDimensions",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:ListAvailableResourceMetrics",
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
        }
    ]
}
```

6. 정책 검토를 선택합니다.
7. 정책의 이름과 설명(선택 사항)을 지정한 다음 정책 검토를 선택합니다.

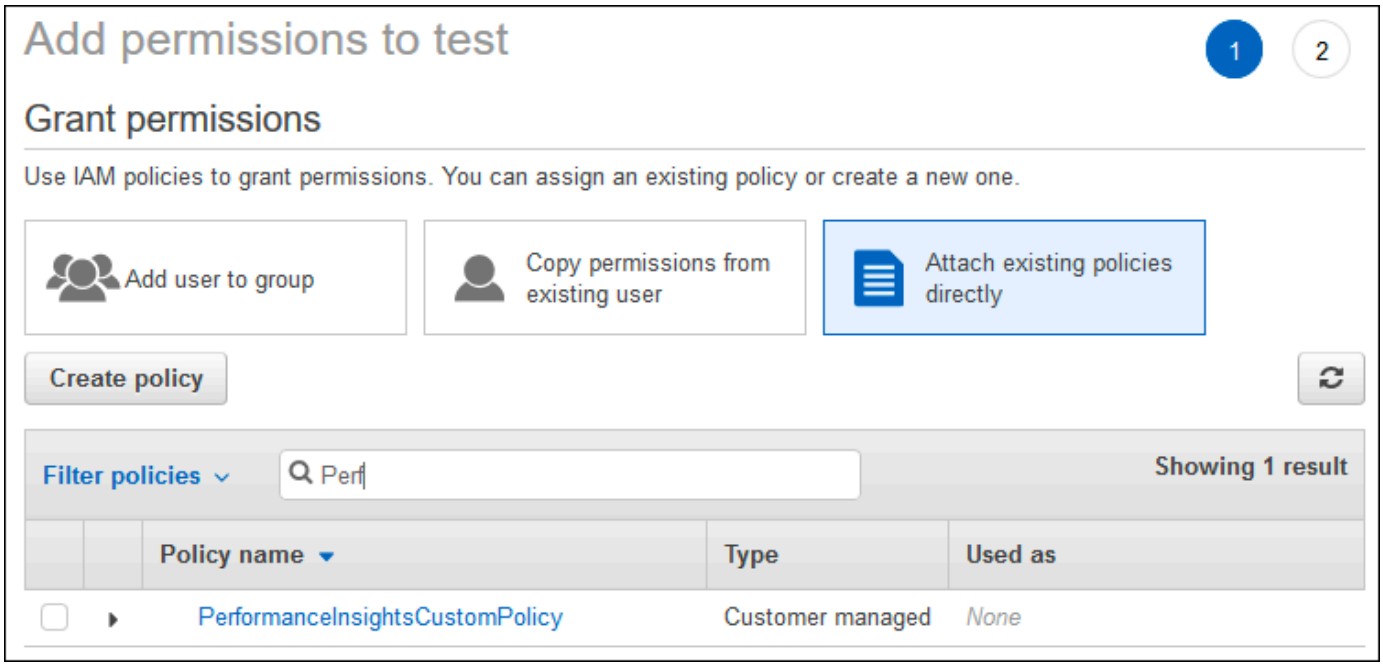
이제 정책을 권한 세트 또는 역할에 연결할 수 있습니다. 다음 절차에서는 이 목적으로 사용할 수 있는 사용자가 이미 있다고 가정합니다.

사용자에게 정책을 연결

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 사용자를 선택합니다.
3. 목록에서 기존 사용자를 선택합니다.

> **Important**
>
> 성능 개선 도우미을 사용하려면 사용자 정의 정책 외에 Amazon DocumentDB에 액세스할 수 있어야 합니다. 예를 들어 AmazonDocDBReadOnlyAccess 사전 정의 정책은 Amazon DocDB에 대한 읽기 전용 액세스를 제공합니다. 자세한 내용은 정책을 사용한 액세스 권리를 참조하세요.

4. 요약 페이지에서 권한 추가를 선택합니다.
5. 기존 정책 직접 첨부를 선택합니다. 검색에 다음과 같이 정책 이름의 첫 문자 몇 개를 입력합니다.



6. 정책을 선택하고 다음: 검토를 선택합니다.

7. 권한 추가를 선택합니다.

성능 개선 도우미에 대한 AWS KMS 정책 구성

성능 개선 도우미는 AWS KMS key 를 사용하여 민감한 데이터를 암호화합니다. API 또는 콘솔을 통해 성능 개선 도우미를 활성화하면 다음 옵션이 있습니다:

- 기본값을 선택합니다 AWS 관리형 키.

  Amazon DocumentDB는 새 DB 인스턴스에 AWS 관리형 키 를 사용합니다. Amazon DocumentDB는 AWS 계정에 AWS 관리형 키 대한를 생성합니다. AWS 계정에는 AWS 리전마다 Amazon DocumentDB에 AWS 관리형 키 대해 다른이 있습니다.
- 고객 관리형 키를 선택합니다.

  고객 관리형 키를 지정하는 경우 성능 개선 도우미 API를 호출하는 계정의 사용자는 KMS 키에 대한 `kms:Decrypt` 및 `kms:GenerateDataKey` 권한이 필요합니다. IAM 정책을 사용하는 대신 키 정책을 사용하여 KMS AWS KMS 키의 권한을 관리합니다. 자세한 내용은 AWS KMS에서 키 정책 사용 섹션을 참조하세요.

Example

다음 샘플 키 정책은 KMS 키 정책에 문을 추가하는 방법을 보여 줍니다. 이러한 문을 통해 성능 개선 도우미에 액세스할 수 있습니다. 사용 방법에 따라 몇 가지 제한을 변경할 AWS KMS수 있습니다. 정책에 문을 추가하기 전에 모든 문을 제거하세요.

성능 개선 도우미 대시보드를 사용한 지표 분석

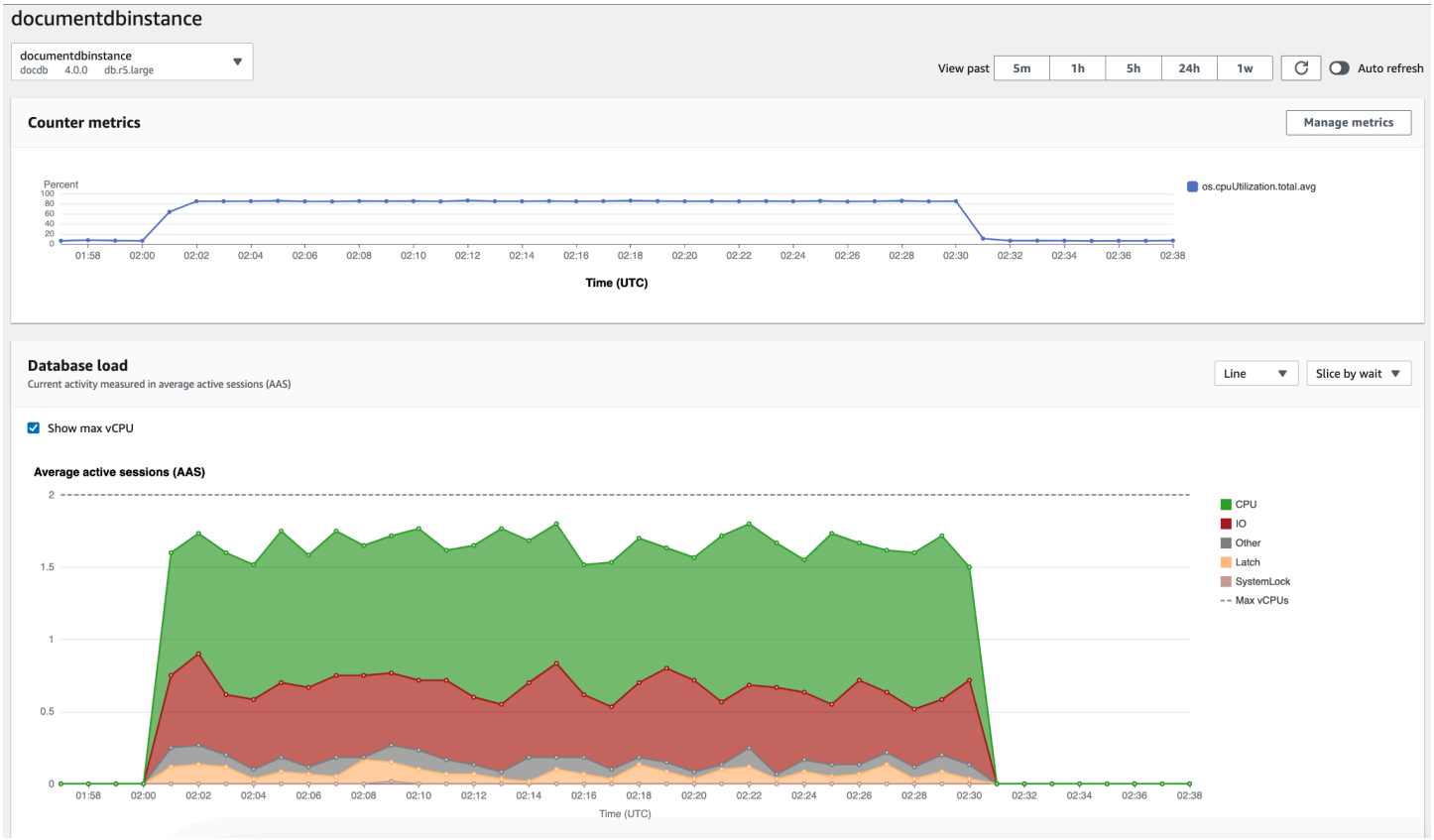
성능 개선 도우미 대시보드에는 성능 문제를 분석하여 해결할 수 있는 데이터베이스 성능 정보가 포함됩니다. 기본 대시보드 페이지에서 데이터베이스 로드 (DB load)에 대한 정보를 볼 수 있습니다. 대기 상태나 쿼리와 같은 차원별로 DB 로드를 "슬라이스"할 수 있습니다.

주제

- 성능 개선 도우미 대시보드 개요
- 성능 개선 도우미 대시보드 열기
- 대기 상태별 데이터베이스 로드 분석
- 상위 쿼리 탭 개요
- 데이터베이스 로드 차트에서 확대

성능 개선 도우미 대시보드 개요

대시보드는 성능 개선 도우미와 상호 작용하는 가장 간편한 방법입니다. 다음 예제는 Amazon DocumentDB 인스턴스의 대시보드를 보여줍니다. 성능 개선 도우미 대시보드는 기본적으로 마지막 1시간 동안 수집된 데이터를 표시합니다.



대시보드는 다음과 같은 부분으로 나뉩니다:

1. 카운터 지표 – 특정 성능 카운터 지표의 데이터를 보여줍니다.
2. 데이터베이스 로드 – 최대 vCPU 선으로 표시된 바와 같이 DB 로드와 DB 인스턴스 용량을 비교하는 방식을 표시합니다.
3. 상위 차원 – DB 로드에 기여하는 상위 차원을 보여줍니다. 이러한 차원에는 `waits`, `queries`, `hosts`, `databases`, 및 `applications`가 포함됩니다.

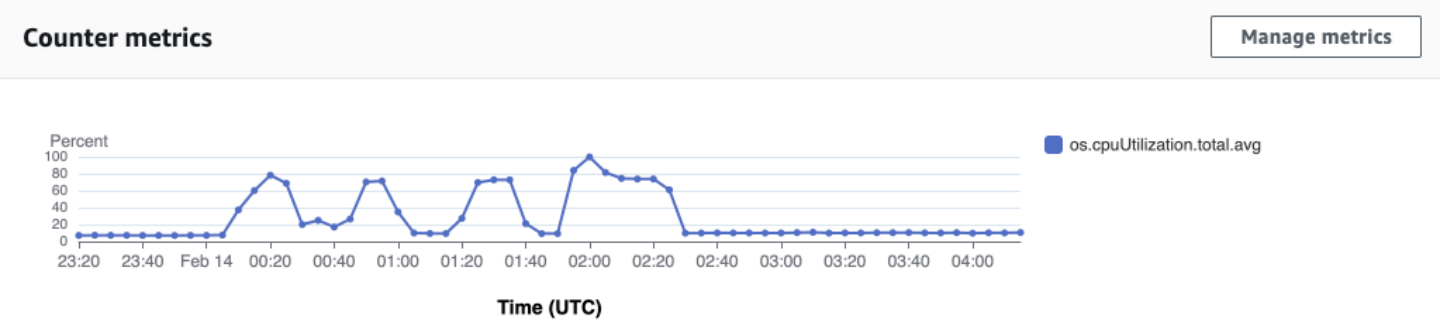
주제

- 카운터 지표 차트
- 데이터베이스 로드 차트
- 상위 측정기준 테이블

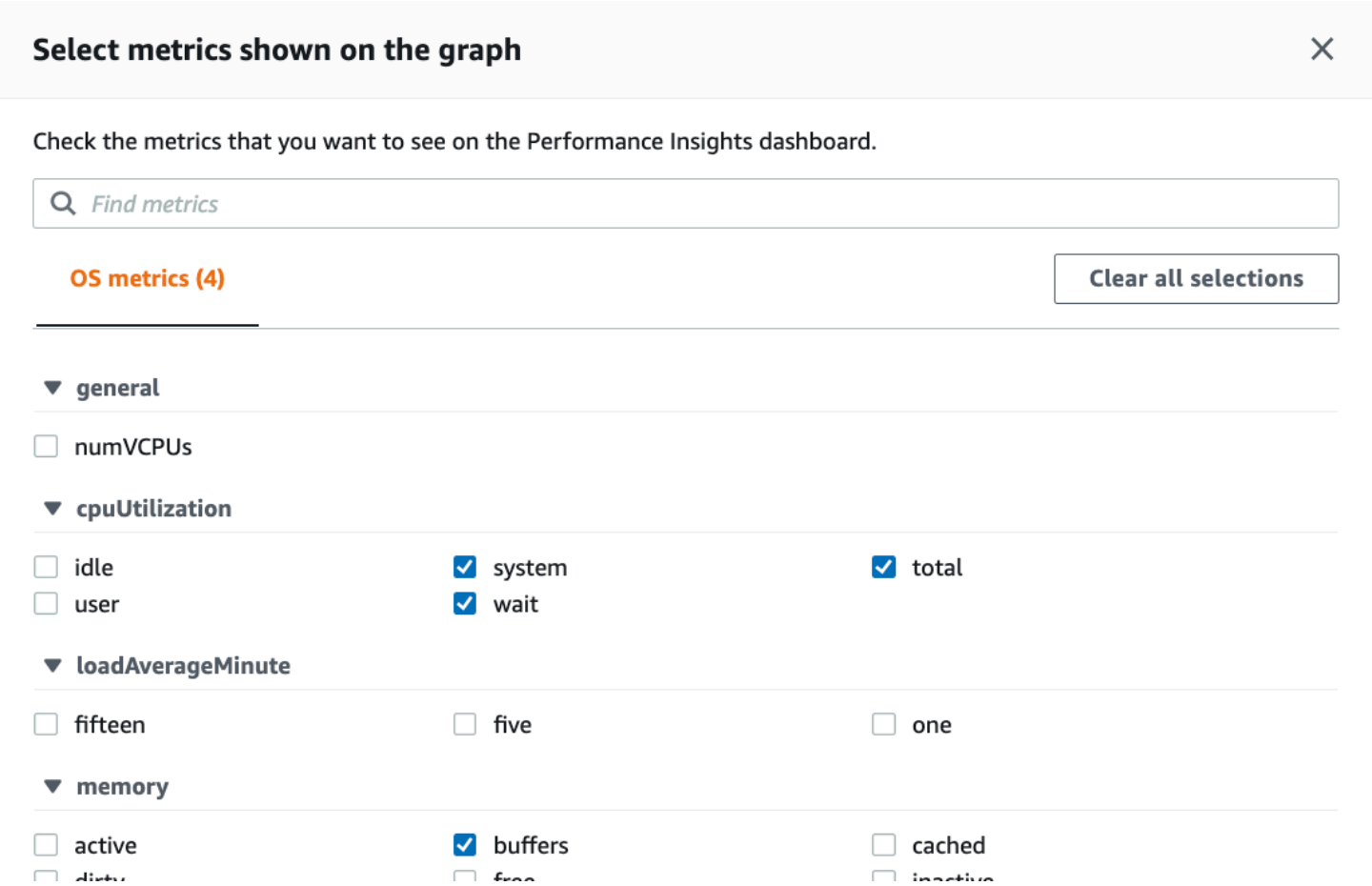
카운터 지표 차트

계수기 지표를 통해 성능 개선 도우미 대시보드에 최대 10개의 추가 그래프가 포함되도록 사용자 지정할 수 있습니다. 이 그래프는 수십 개의 운영 체제 지표를 보여줍니다. 이 정보와 데이터베이스 로드를 연관지으면 성능 문제를 식별하고 분석하는 데 도움이 됩니다.

카운터 지표 차트에는 성능 카운터의 데이터가 표시됩니다.



지표 관리를 선택하여 성능 카운터를 변경합니다. 다음 스크린샷과 같이 여러 OS 메트릭을 선택할 수 있습니다. 지표에 대한 세부 정보를 보려면 지표 이름 위에 마우스 포인터를 놓습니다.



데이터베이스 로드 차트

데이터베이스 로드 차트는 최대 vCPU 선으로 표시된 바와 같이 데이터베이스 활동과 인스턴스 용량을 비교하는 방식을 표시합니다. 기본적으로 누적 꺾은선형 차트는 단위 시간당 평균 활성 세션으로 DB 로드를 나타냅니다. DB 로드는 대기 상태에 따라 슬라이스(그룹화)됩니다.

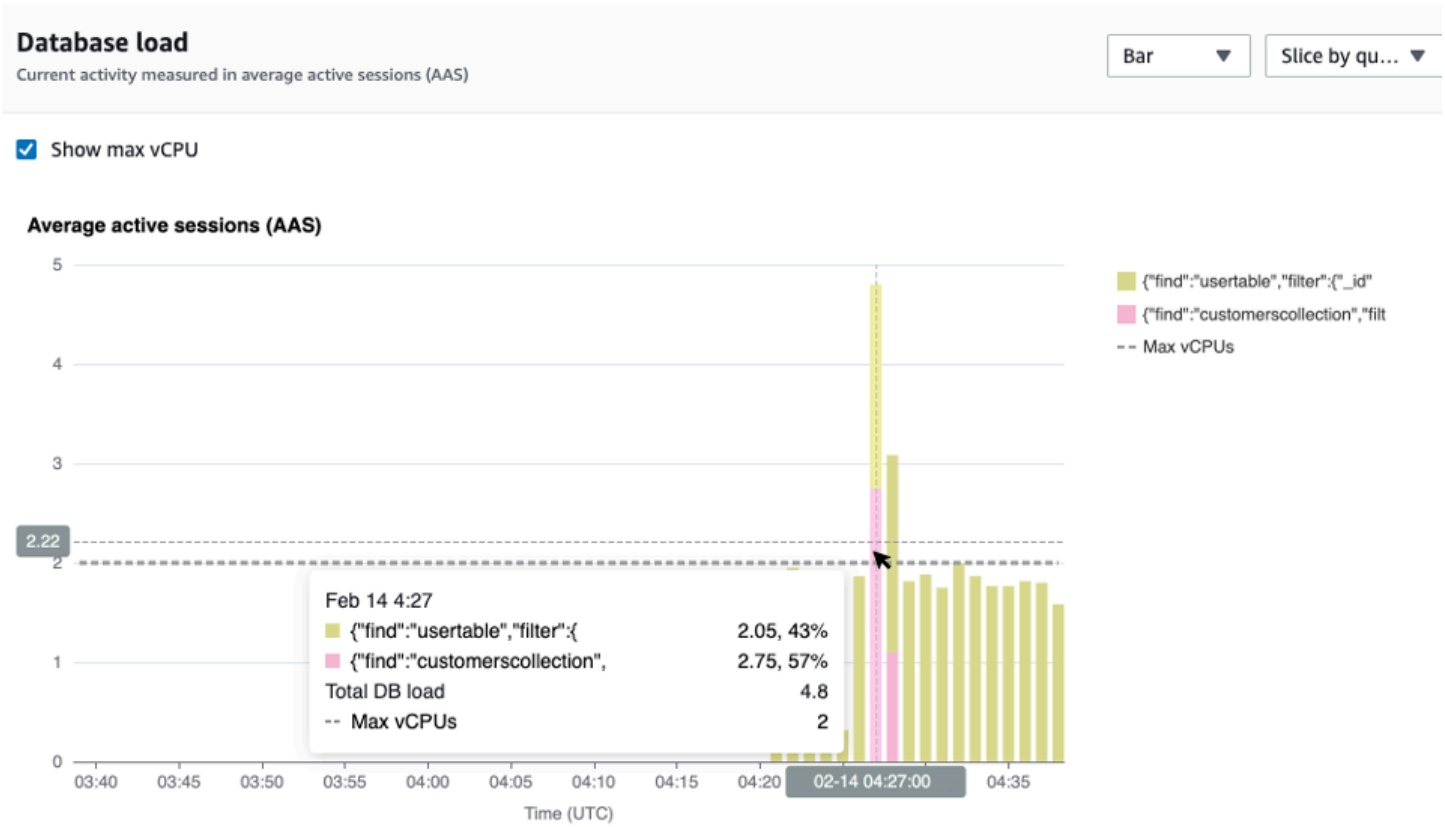
차원을 기준으로 분할된 DB 로드

지원되는 차원별로 그룹화된 활성 세션으로 로드를 표시하도록 선택할 수 있습니다. 다음 이미지는 Amazon DocumentDB 인스턴스의 치수를 보여줍니다.

차원 항목에 대한 DB 로드 세부 정보

차원 내의 DB 로드 항목에 대한 세부 정보를 보려면 항목 이름 위로 마우스를 가져갑니다. 다음 이미지는 쿼리문에 대한 세부 정보를 보여줍니다.

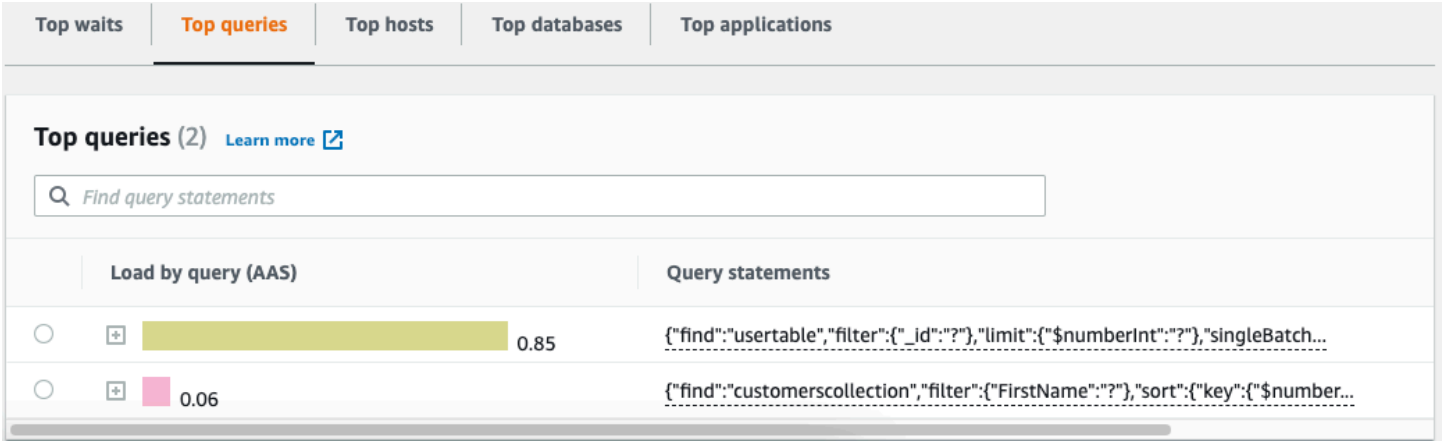
범례에서 선택한 기간의 항목에 대한 세부 정보를 보려면 해당 항목 위에 마우스 포인터를 놓습니다.



상위 측정기준 테이블

상위 측정기준 테이블은 DB 로드를 다른 차원으로 슬라이스합니다. 차원은 DB 로드의 다양한 특성에 대한 카테고리 또는 "슬라이스"입니다. 측정기준이 쿼리인 경우 상위 쿼리(Top queries)에서는 DB 로드에 가장 많이 기여하는 쿼리 문을 보여줍니다.

다음 차원 탭 중 하나를 선택합니다.



다음 표는 각 탭에 대한 간략한 설명을 제공합니다.

탭 명	설명
상위 대기	대기 이벤트가 백엔드가 대기 중인 이

테이블 이름

이벤트

현재 실행 중인 쿼리문

연결된 클라이언트의 호스트 IP 및 포트

테이블 이름

데이터베이스와 연결된 데이터베이스의 이름

탭 이름

상위 쿼리(Top queries)

애플리케이션 이름

상위 쿼리(Top queries) 탭을 사용하여 쿼리를 분석하는 방법을 알아보려면 상위 쿼리 탭 개요 섹션을 참조하세요.

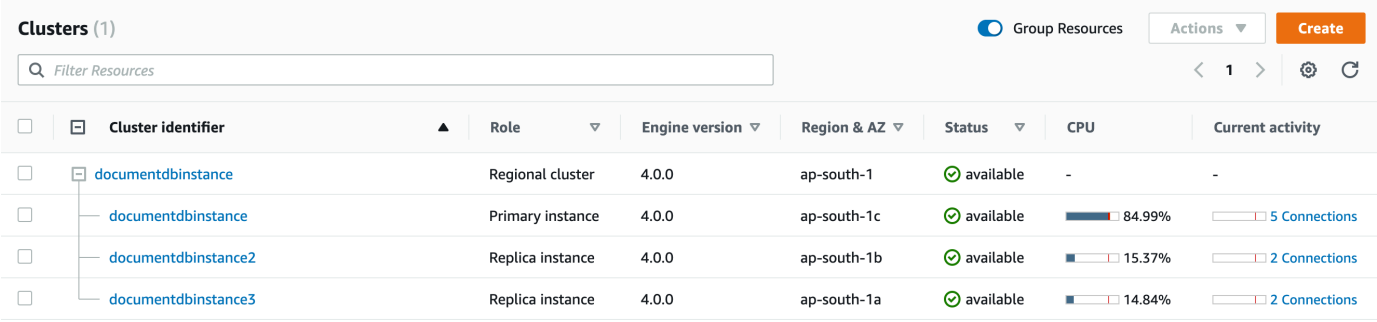
성능 개선 도우미 대시보드 열기

AWS 관리 콘솔에서 성능 개선 도우미 대시보드를 보려면 다음 단계를 사용합니다.

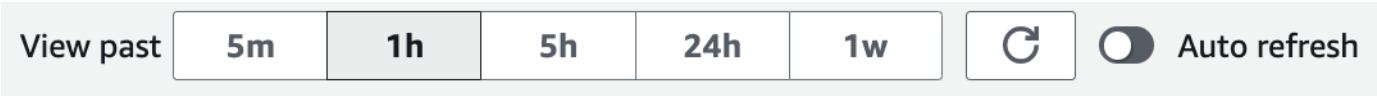
1. https://console.aws.amazon.com/docdb/ 에서 성능 개선 도우미 콘솔을 엽니다.
2. DB 인스턴스를 선택합니다. Amazon DocumentDB 인스턴스에 대한 성능 개선 도우미 대시보드가 표시됩니다.

   성능 개선 도우미가 사용 가능한 Amazon DocumentDB 인스턴스의 경우, 인스턴스 목록에서 세션 항목을 선택하여 대시보드에 도달할 수도 있습니다. 현재 활동에서 세션 항목은 지난 5분 동안 평균 활동 세션의 데이터베이스 로드를 보여 줍니다. 로드가 막대 모양으로 표시됩니다. 막대가 비어

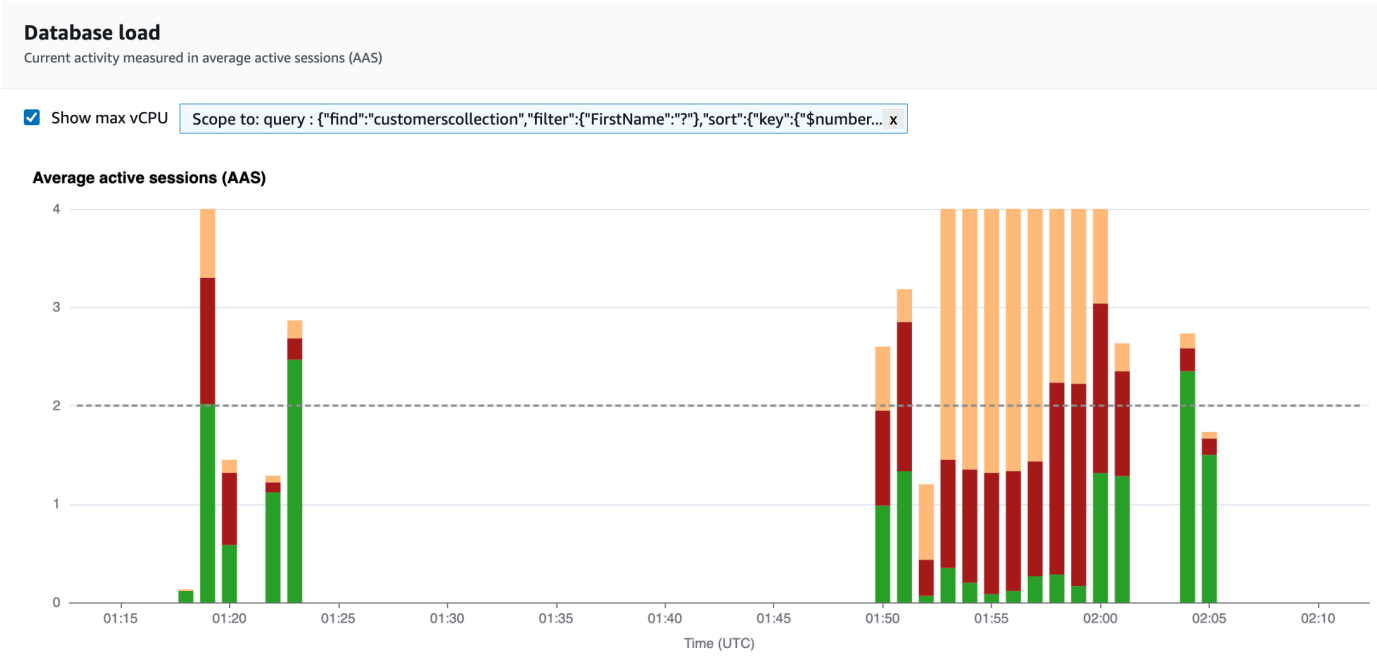
있으면 DB 인스턴스가 유휴 상태임을 뜻합니다. 로드가 증가하면 막대가 파란색으로 채워집니다. 로드에서 인스턴스 클래스의 가상 CPU(vCPU) 수를 전달하면 막대가 빨간색으로 바뀌고 병목 가능성을 나타냅니다.



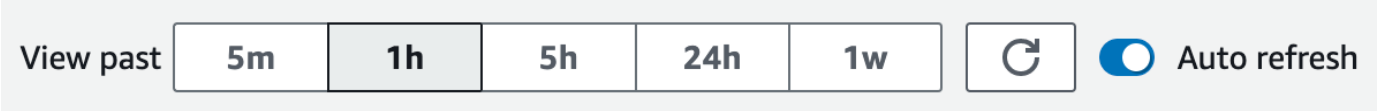
3. (선택 사항) 오른쪽 상단의 버튼을 선택하여 다른 시간 간격을 선택합니다. 예를 들어 간격을 1시간으로 변경하려면 1h를 선택합니다.



다음 스크린샷에서 DB 로드 간격은 1시간입니다.



4. 데이터를 자동으로 새로 고치려면 자동 새로 고침을 활성화합니다.



성능 개선 도우미 대시보드는 새 데이터로 자동으로 고쳐집니다. 새로 고침 속도는 표시되는 데이터의 양에 따라 다릅니다:

- 5분은 5초마다 새로 고칩니다.
- 1시간은 1분마다 새로 고칩니다.
- 5시간은 1분마다 새로 고칩니다.
- 24시간은 5분마다 새로 고칩니다.
- 1주는 1시간마다 새로 고칩니다.

대기 상태별 데이터베이스 로드 분석

데이터베이스 로드(DB 로드) 차트에 병목현상이 나타나면 어디에서 로드가 발생하는지 알 수 있습니다. 이렇게 하려면 데이터베이스 로드 차트 아래의 상위 로드 항목 테이블을 살펴보세요. 특정 항목(예: 쿼리 또는 응용프로그램)을 선택하여 해당 항목을 드릴다운하고 세부 정보를 확인합니다.

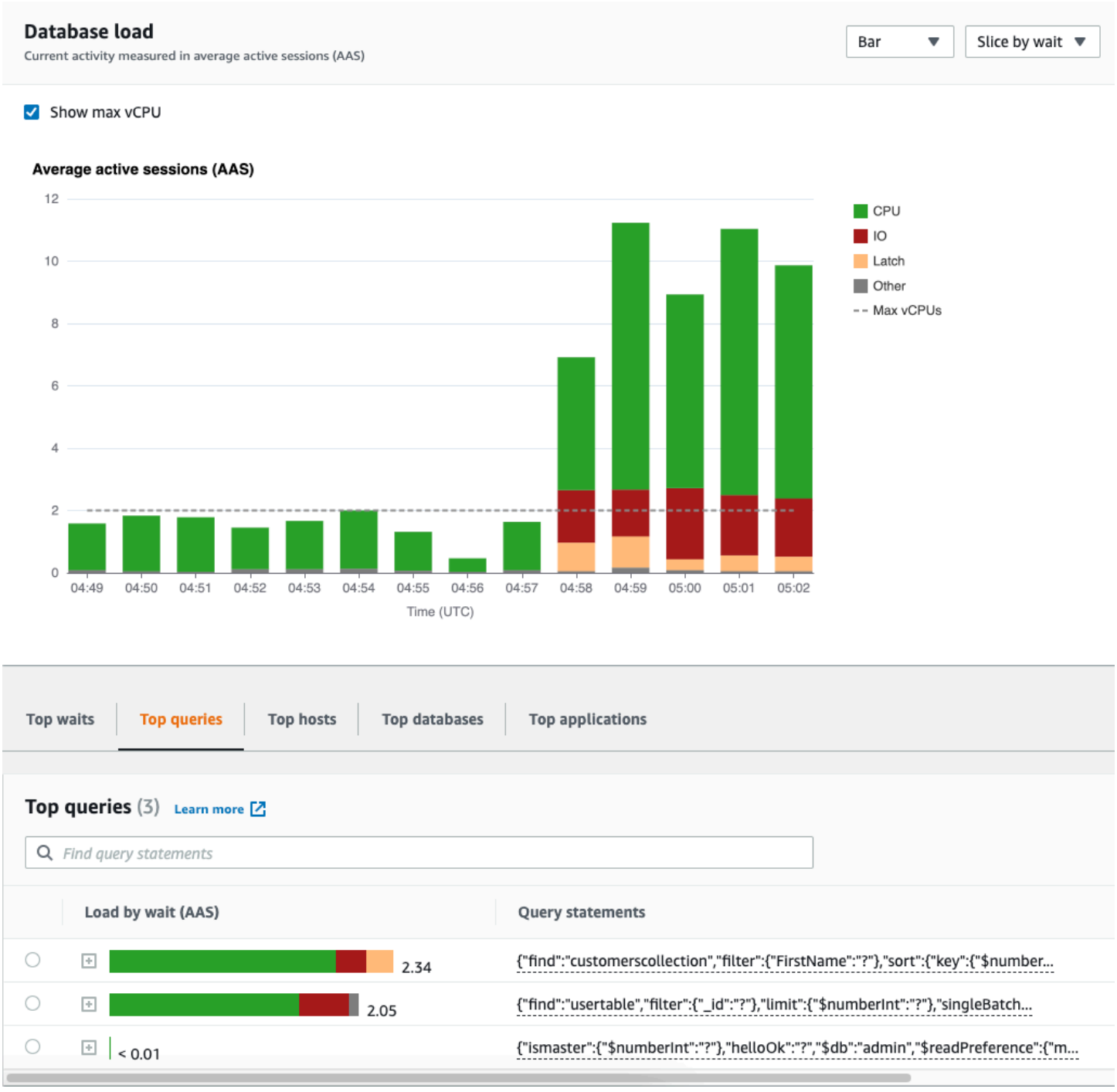
대기 및 상위 쿼리로 그룹화된 DB 로드는 일반적으로 성능 문제에 대한 가장 많은 통찰력을 제공합니다. 대기 상태를 기준으로 구분된 DB 로드는 데이터베이스의 리소스 또는 동시성 병목 현상 유무를 표시합니다. 이 경우, 상위 로드 항목 테이블의 상위 쿼리 탭에는 해당 로드를 구동하는 쿼리가 표시됩니다.

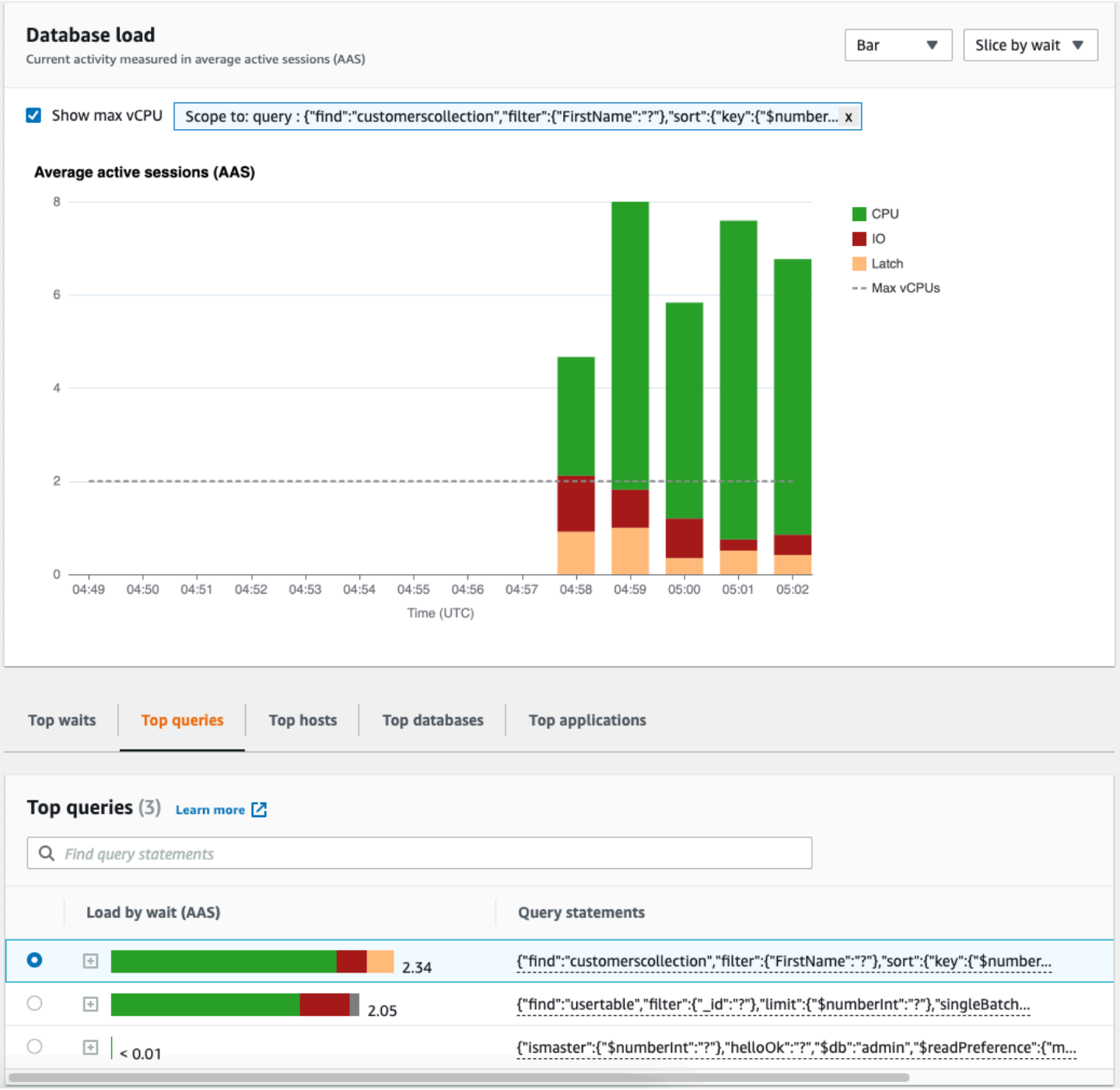
성능 문제를 진단하는 일반 워크플로우는 다음과 같습니다:

1. 데이터베이스 로드 차트를 보면서 데이터베이스 로드가 최대 CPU 선을 상회하는지 확인합니다.
2. 상회하는 경우가 있으면 데이터베이스 로드 차트를 보면서 원인이 되는 대기 상태를 식별합니다.
3. 상위 로드 항목 테이블의 상위 쿼리 탭이 해당 대기 상태에 가장 크게 기여하는 쿼리를 확인하여 로드를 유발하는 다이제스트 쿼리를 식별합니다. 이러한 내용들을 대기별 로드(AAS) 열로 식별할 수 있습니다.
4. 상위 쿼리 탭에서 이러한 다이제스트 쿼리 중 하나를 선택하여 해당 쿼리를 확장하고 해당 쿼리가 구성된 하위 쿼리를 확인합니다.

또한 상위 호스트 또는 상위 애플리케이션을 각각 선택하면 어떤 호스트나 애플리케이션이 가장 많은 로드를 주고 있는지 확인할 수 있습니다. 애플리케이션 이름은 Amazon DocumentDB 인스턴스에 대한 연결 문자열에 지정되어 있습니다. Unknown은 애플리케이션 필드가 지정되지 않았음을 나타냅니다.

예를 들어 다음 대시보드에서 CPU 대기 시간은 대부분의 DB 로드를 차지합니다. 상위 쿼리에서 상위 쿼리를 선택하면 데이터베이스 로드 차트의 범위가 선택 쿼리로 인해 발생한 가장 많은 로드를 집중적으로 표시합니다.





상위 쿼리 탭 개요

기본적으로 상위 쿼리 탭에는 DB 로드에 가장 큰 기여를 하는 쿼리가 표시됩니다. 쿼리 텍스트를 분석하여 쿼리를 조정하는 데 도움이 될 수 있습니다.

주제

- 쿼리 다이제스트

- 대기별 로드(AAS)
- 자세한 쿼리 정보 보기
- 명령문 쿼리 텍스트에 액세스
- 명령문 쿼리 텍스트 보기 및 다운로드

쿼리 다이제스트

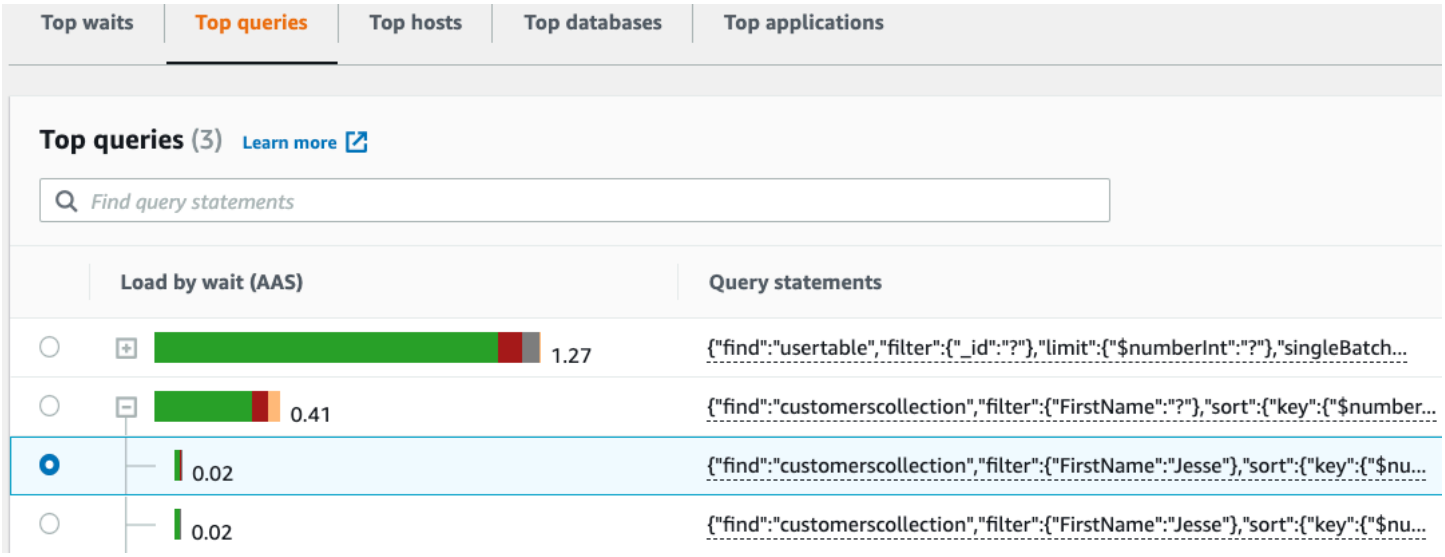
쿼리 다이제스트는 구조적으로 유사하지만 다른 리터럴 값을 가질 수 있는 여러 실제 쿼리의 합성입니다. 다이제스트는 하드 코딩된 값을 물음표로 바꿉니다. 예를 들어 쿼리 다이제스트는 다음과 같습니다:

```
{"find":"customerscollection","filter":{"FirstName":"?"},"sort":{"key":
{"$numberInt":"?"}},"limit":{"$numberInt":"?"}}
```

이 다이제스트에는 다음 하위 쿼리가 포함될 수 있습니다:

```
{"find":"customerscollection","filter":{"FirstName":"Karrie"},"sort":{"key":
{"$numberInt":"1"}},"limit":{"$numberInt":"3"}}
{"find":"customerscollection","filter":{"FirstName":"Met"},"sort":{"key":
{"$numberInt":"1"}},"limit":{"$numberInt":"3"}}
{"find":"customerscollection","filter":{"FirstName":"Rashin"},"sort":{"key":
{"$numberInt":"1"}},"limit":{"$numberInt":"3"}}
```

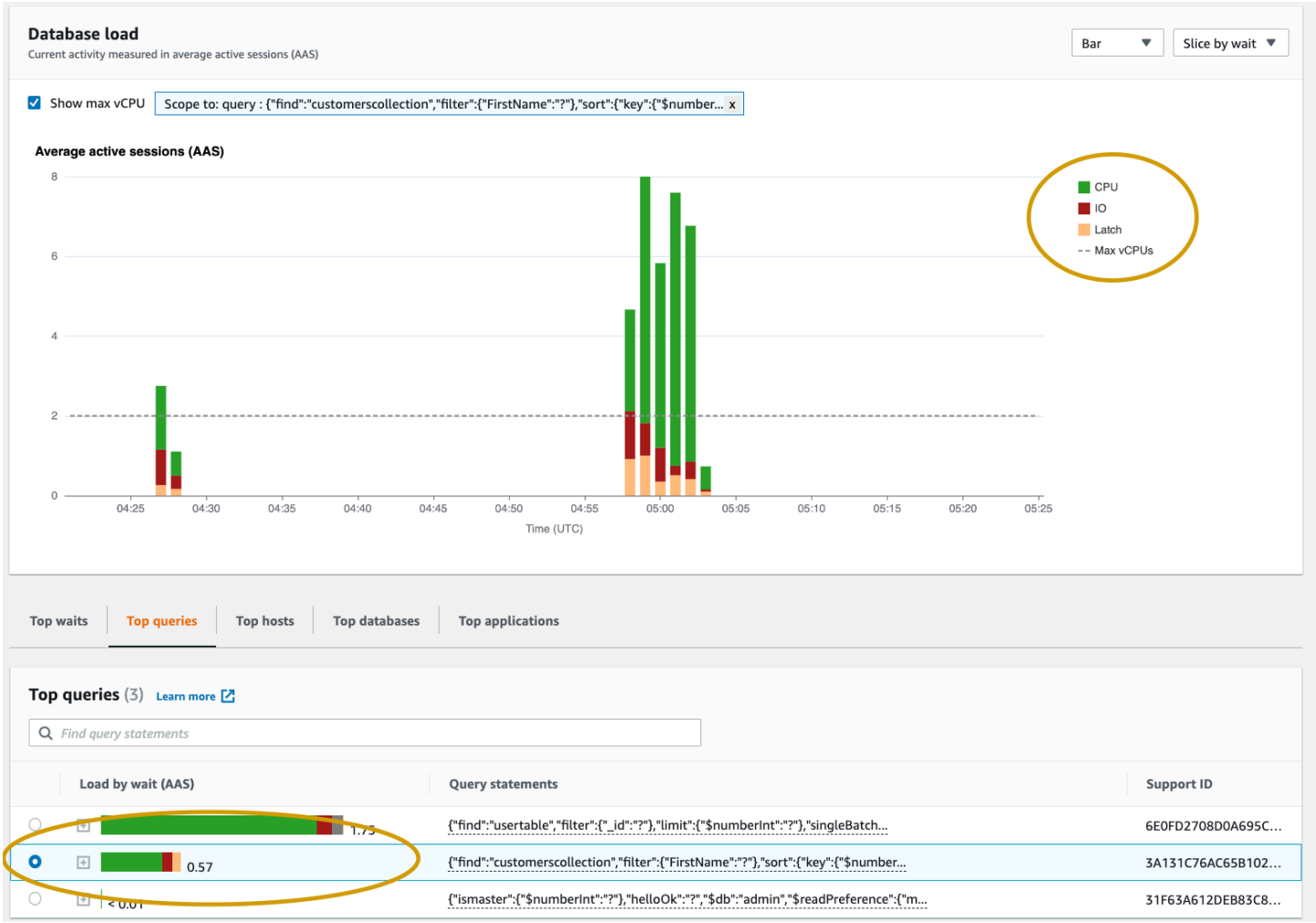
요약에서 문자 그대로의 쿼리 문을 보려면 쿼리를 선택한 다음 더하기 기호(+)를 선택합니다. 다음 스크린샷에서 선택한 쿼리는 다이제스트입니다.



> **Note**
>
> 쿼리 요약은 유사한 쿼리 문을 그룹화하지만 중요한 정보는 수정하지 않습니다.

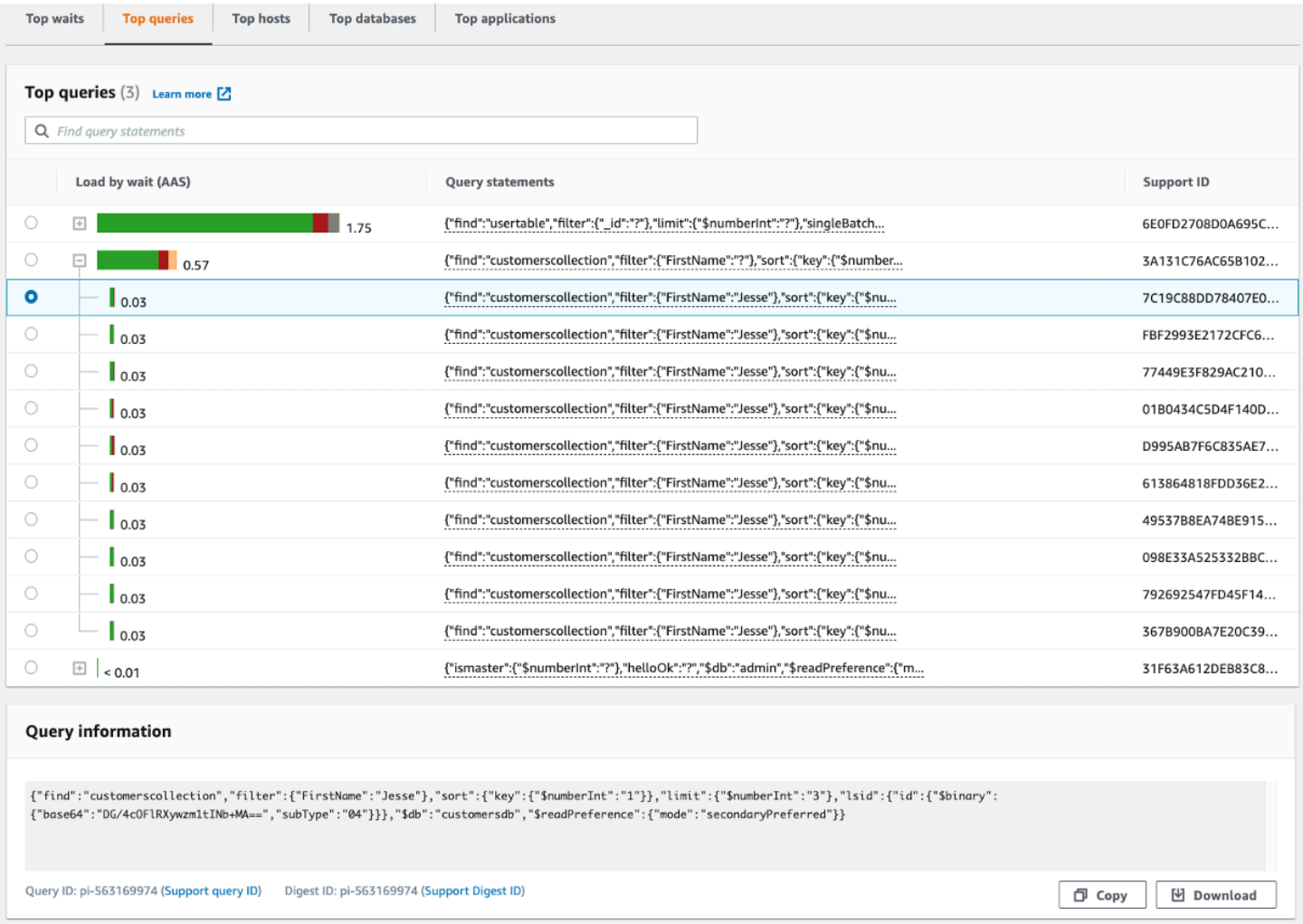
대기별 로드(AAS)

상위 쿼리에서 대기별 로드(AAS) 열은 각 상위 로드 항목과 관련된 데이터베이스 로드의 백분율을 나타냅니다. 이 열에는 현재 DB 로드 차트에서 어떤 그룹화 기준을 선택하든 그 기준에 따라 해당 항목의 로드가 반영됩니다. 예를 들어 DB 로드차트를 대기 상태별로 그룹화할 수 있습니다. 이 경우 대기별 DB 로드 막대는 쿼리가 영향을 미치는 대기 상태의 정도를 크기, 세그먼트 및 컬러 코드로 표시합니다. 또한 선택한 쿼리에 영향을 미치는 대기 상태를 표시합니다.



자세한 쿼리 정보 보기

상위 쿼리 표에서, 다이제스트 문을 열어 해당 정보를 볼 수 있습니다. 맨 아래 창에 정보가 나타납니다.



다음과 같은 식별자(ID) 유형은 쿼리 문과 연결되어 있습니다.

1. 쿼리 ID 지원 – 쿼리 ID의 해시 값입니다. 이 값은 AWS Support에서 작업할 때 쿼리 ID를 참조하는 용도로만 사용됩니다. AWS Support는 실제 쿼리 IDs 및 쿼리 텍스트에 액세스할 수 없습니다.
2. 다이제스트 ID 지원 – 다이제스트 ID의 해시 값입니다. 이 값은 AWS Support로 작업할 때 다이제스트 ID를 참조하는 용도로만 사용됩니다. AWS Support는 실제 다이제스트 IDs 및 쿼리 텍스트에 액세스할 수 없습니다.

명령문 쿼리 텍스트에 액세스

기본적으로 상위 쿼리 테이블의 각 행은 각 쿼리문에 대해 500바이트의 쿼리 텍스트를 보여줍니다. 다이제스트 문이 500바이트 이상인 경우 성능 개선 도우미 대시보드에서 해당 문을 열어 더 많은 텍스트를 볼 수 있습니다. 이 경우 표시되는 쿼리의 최대 길이는 1KB입니다. 하위 SQL 문을 보는 경우 다운로드를 선택할 수도 있습니다.

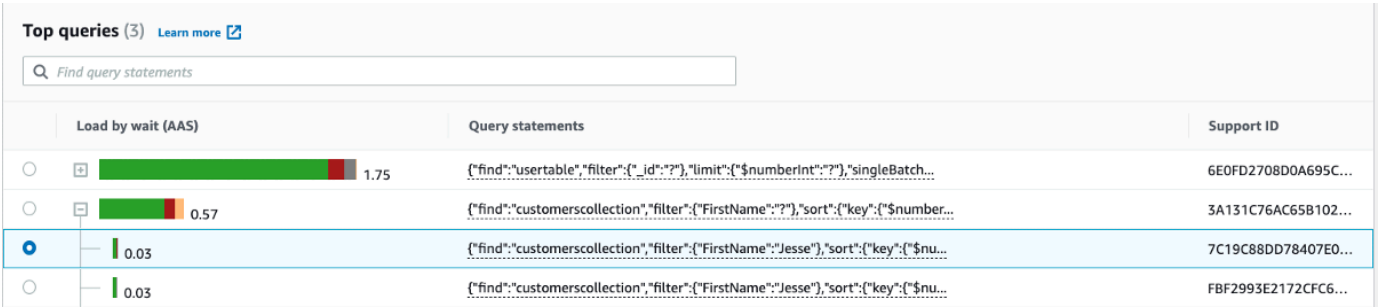
명령문 쿼리 텍스트 보기 및 다운로드

성능 개선 도우미 대시보드에서 쿼리 텍스트를 보거나 다운로드할 수 있습니다.

성능 개선 도우미 대시보드에서 더 많은 쿼리 텍스트를 보려면 다음과 같이 하십시오

1. https://console.aws.amazon.com/docdb/ 에서 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 성능 개선 도우미을 선택합니다.
3. DB 인스턴스를 선택합니다. 선택한 DB 인스턴스에 대한 성능 개선 도우미 대시보드가 표시됩니다.

   500바이트 이상의 텍스트가 있는 SQL 문은 다음 이미지와 유사합니다:



4. 쿼리 정보 섹션을 조사하여 쿼리 텍스트를 더 확인합니다.

성능 개선 도우미 대시보드는 전체 쿼리문마다 최대 1KB까지 표시할 수 있습니다.

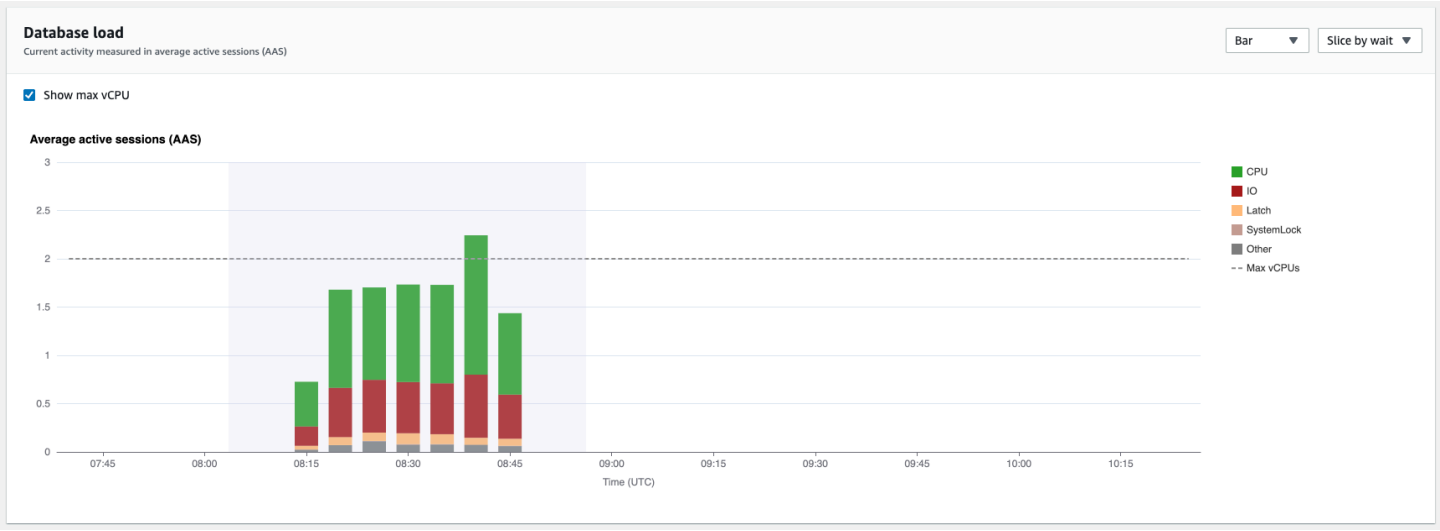
> **Note**
>
> 쿼리문을 복사하거나 다운로드하려면 팝업 차단을 비활성화합니다.

데이터베이스 로드 차트에서 확대

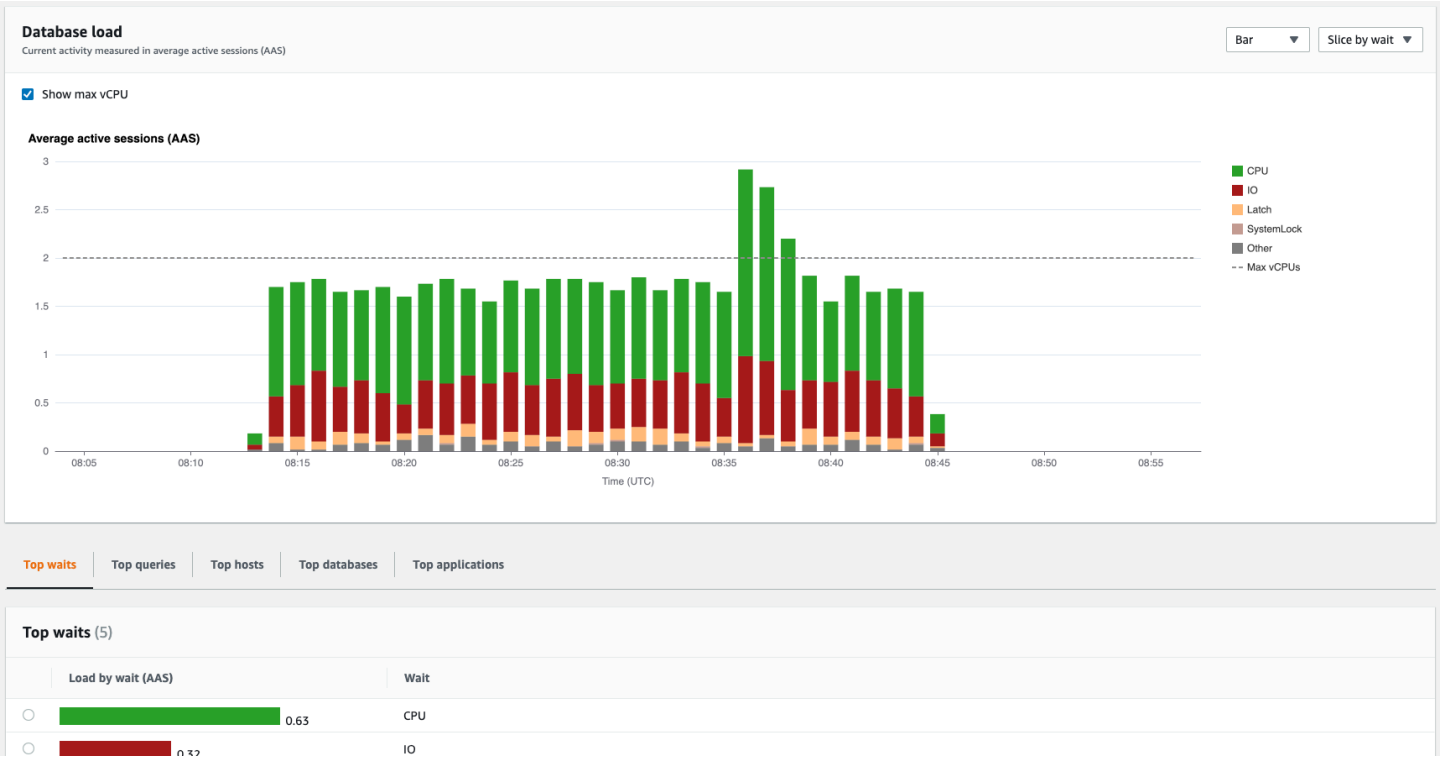
성능 개선 도우미 사용자 인터페이스의 다른 특성을 사용해 성능 데이터를 분석할 수 있습니다.

클릭하여 끌어 확대

성능 개선 도우미 인터페이스에서 로드 차트의 작은 부분을 선택하여 확대해 자세히 볼 수 있습니다.



로드 차트의 한 부분을 확대하려면 시작 시간을 선택하고 원하는 기간 끝까지 끕니다. 이렇게 하면 선택한 영역이 강조 표시됩니다. 마우스를 놓으면 로드 차트가 선택한 영역을 확대하고 상위 항목 테이블이 다시 계산됩니다.



성능 개선 도우미 API를 사용하여 지표 검색

성능 개선 도우미를 활성화하면 API에서 인스턴스 성능에 대한 가시성을 제공합니다. Amazon CloudWatch Logs는 AWS 서비스에 대한 판매 모니터링 지표를 위한 신뢰할 수 있는 소스를 제공합니다.

성능 개선 도우미는 평균 활성 세션(AAS) 수로 측정되는 데이터베이스 로드에 대한 도메인 별 보기를 제공합니다. 이 지표는 API 소비자에게 2차원 시계열 데이터 세트로 표시됩니다. 데이터의 시간 차원은 쿼리된 시간 범위 내 각 시점에 대한 DB 로드 데이터를 제공합니다. 각 시점에서는 요청된 차원에 관해 해당 시점에서 측정되는 전체 로드를 분해합니다(예: `Query`, `Wait-state`, `Application` 또는 `Host`).

Amazon DocumentDB 성능 개선 도우미는 데이터베이스 성능을 분석하고 문제를 해결할 수 있도록 Amazon DocumentDB 인스턴스를 모니터링합니다. 성능 개선 도우미 데이터를 볼 수 있는 한 가지 방법은 AWS Management Console에서 보는 것입니다. 또한 성능 개선 도우미는 사용자가 자신의 데이터를 쿼리할 수 있도록 퍼블릭 API도 제공합니다. API를 사용하여 다음을 수행할 수 있습니다:

- 데이터를 데이터베이스로 오프로드
- 기존 모니터링 대시보드에 성능 개선 도우미 데이터 추가
- 모니터링 도구 구축

성능 개선 도우미 API를 사용하려면 Amazon DocumentDB 인스턴스 중 하나에서 성능 개선 도우미를 활성화합니다. 성능 개선 도우미 활성화에 대한 자세한 내용은 성능 개선 도우미 활성화 및 비활성화(을)를 참조하십시오. 성능 개선 도우미 API에 대한 자세한 내용은 성능 개선 도우미 API 참조를 참조하십시오.

성능 개선 도우미 API에서는 다음과 같은 작업을 제공합니다.

성능 개선 도우미 작업	AWS CLI 명령	설명
DescribeDimensionKeys	aws pi describe-dimension-keys	특정 기간에 대해 지표의 상위 N개 차원 키를 검색합니다.
GetDimensionKeyDetails	aws pi get-dimension-key-details	DB 인스턴스 또는 데이터 소스에 지정된 차원 그룹의 속성을 검색합니다. 예를 들어 쿼리 ID를 지정하고 차원 세부 정보를 사용할 수 있는 경

성능 개선 도우미 작업	AWS CLI 명령	설명
		우 GetDimensionKeyDetails 는 이 ID에 연결된 차원 db.query.statement 의 전체 텍스트를 검색합니다. GetResourceMetrics 및 DescribeDimensionKeys 는 대용량 쿼리 문 텍스트 검색을 지원하지 않으므로 이 작업을 유용하게 사용할 수 있습니다.
GetResourceMetadata	aws pi get-resource-metadata	다양한 기능에 대한 특성을 검색합니다. 예를 들어 메타데이터는 특정 DB 인스턴스에서 특성이 켜지거나 꺼짐을 나타낼 수 있습니다.
GetResourceMetrics	aws pi get-resource-metrics	일정 기간의 데이터 소스 집합에 대한 성능 개선 도우미 지표를 검색합니다. 특정 차원 그룹 및 차원을 제공하고 각 그룹에 집계 및 필터링 기준을 제공할 수 있습니다.
ListAvailableResourceDimensions	aws pi list-available-resource-dimensions	지정된 인스턴스에서 지정된 각 지표 유형에 대해 쿼리할 수 있는 차원을 검색합니다.
ListAvailableResourceMetrics	aws pi list-available-resource-metrics	지정된 DB 인스턴스에 대해 쿼리할 수 있는 지정된 지표 유형의 사용 가능한 모든 지표를 검색합니다.

주제

- AWS CLI 성능 개선 도우미용

- 시계열 지표 조회
- AWS CLI 성능 개선 도우미 예제

AWS CLI 성능 개선 도우미용

AWS CLI를 사용해 성능 개선 도우미 데이터를 볼 수 있습니다. 명령줄에 다음과 같이 입력하여 성능 개선 도우미용 AWS CLI 명령에 대한 도움말을 볼 수 있습니다.

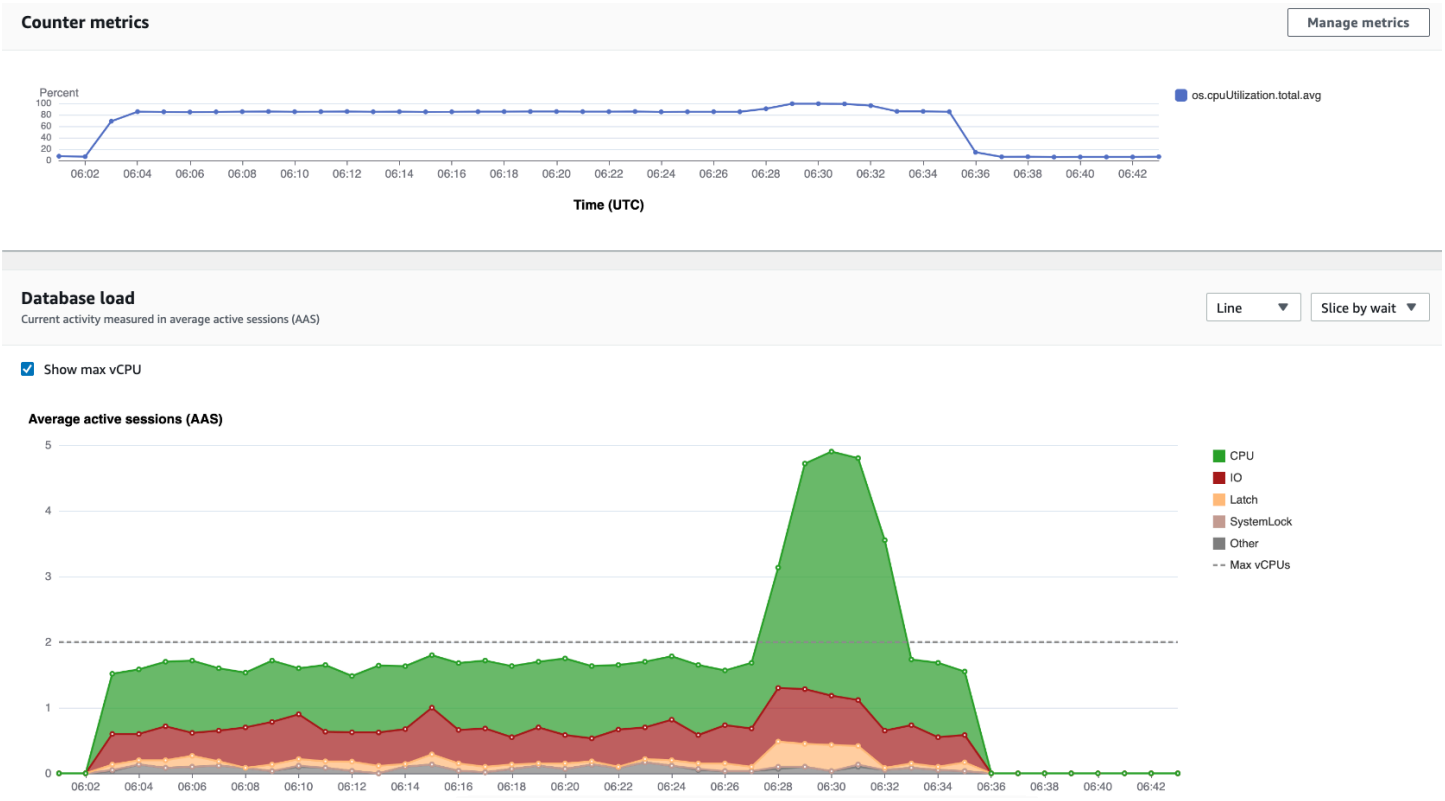
```
aws pi help
```

가 AWS CLI 설치되어 있지 않은 경우 설치에 대한 자세한 내용은 사용 설명서의 AWS 명령줄 인터페이스 설치를 참조하세요. AWS CLI

시계열 지표 조회

`GetResourceMetrics` 연산은 성능 개선 도우미 데이터에서 시계열 지표를 하나 이상 조회합니다. `GetResourceMetrics`에는 지표 및 기간이 필요하고 데이터 포인트 목록이 포함된 응답을 반환합니다.

예를 들어는 다음 이미지와 같이를 AWS Management Console `GetResourceMetrics` 사용하여 카운터 지표 차트와 데이터베이스 로드 차트를 채웁니다.



GetResourceMetrics에서 반환하는 지표는 db.load를 제외하고 모두 표준 시계열 지표입니다. 이 지표는 Database Load(데이터베이스 로드) 차트에 표시됩니다. db.load 지표는 차원이라는 하위 구성 요소로 구분할 수 있다는 점에서 다른 시계열 지표와 다릅니다. 앞의 이미지에서 db.load는 db.load를 구성하는 대기 상태에 따라 구분되고 그룹화됩니다.

> **Note**
>
> GetResourceMetrics에서는 db.sampleload도 반환할 수 있지만 db.load 지표는 대부분의 경우 적절합니다.

GetResourceMetrics에서 반환하는 카운터 지표에 대한 자세한 내용은 카운터 지표에 대한 성능 개선 도우미를 참조하십시오.

지표에 대해서는 다음 계산이 지원됩니다:

- 평균 – 일정 기간 동안 지표의 평균 값입니다. .avg를 지표 이름에 추가합니다.
- 최소 – 일정 기간 동안 지표의 최소 값입니다. .min를 지표 이름에 추가합니다.
- 최대 – 일정 기간 동안 지표의 최대 값입니다. .max를 지표 이름에 추가합니다.
- 합계 – 일정 기간 동안 지표 값의 합계입니다. .sum를 지표 이름에 추가합니다.
- 샘플 수 – 일정 기간 동안 지표가 수집된 횟수입니다. .sample_count를 지표 이름에 추가합니다.

예를 들어 지표를 300초 (5분) 동안 분당 1회씩 수집한다고 가정합시다. 각 분의 값은 1, 2, 3, 4, 5입니다. 이 경우 다음과 같은 계산 결과가 반환됩니다:

- 평균 – 3
- 최소 – 1
- 최대 – 5
- 합계 – 15
- 샘플 수 – 5

get-resource-metrics AWS CLI 명령 사용에 대한 자세한 내용은 섹션을 참조하세요get-resource-metrics.

`--metric-queries` 옵션의 경우 결과를 얻고자 하는 쿼리를 한 개 이상 지정하십시오. 각 쿼리는 필수인 `Metric`과 선택 사항인 `GroupBy` 및 `Filter` 파라미터로 구성됩니다. 다음은 `--metric-queries` 옵션 사양을 보여주는 예입니다.

```
{
   "Metric": "string",
   "GroupBy": {
     "Group": "string",
     "Dimensions": ["string", ...],
     "Limit": integer
   },
   "Filter": {"string": "string"
     ...}
```

AWS CLI 성능 개선 도우미 예제

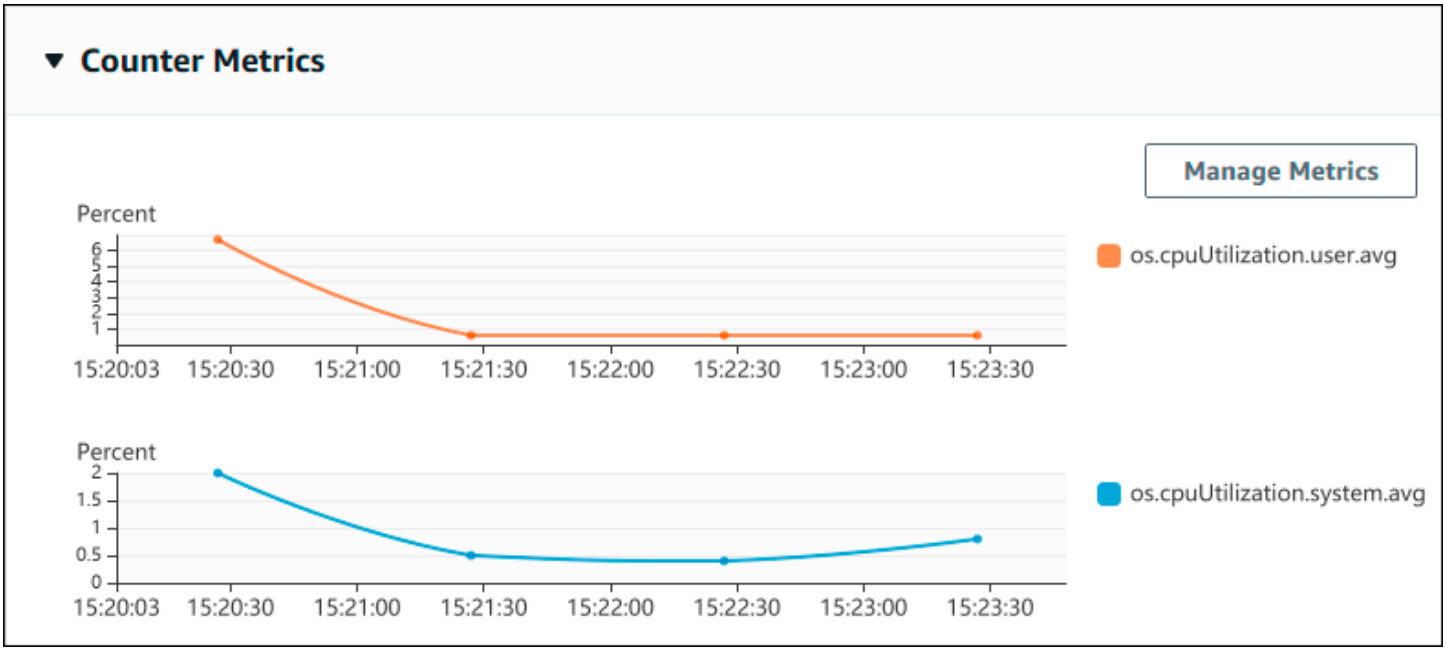
다음 예제에서는 성능 개선 도우미에 AWS CLI 를 사용하는 방법을 보여줍니다.

주제

- 카운터 지표 검색
- 상위 대기 상태에 대한 DB 로드 평균 검색
- 상위 쿼리에 대한 DB 평균 로드 검색
- 쿼리로 필터링된 DB 로드 평균 검색

카운터 지표 검색

다음 스크린샷은 AWS Management Console에 표시되는 카운터 지표 차트 2개를 나타낸 것입니다.



다음 예에서는 카운터 지표 차트 2개를 생성하기 위해 AWS Management Console 이 사용하는 것과 동일한 데이터를 수집하는 방법을 보여줍니다.

Linux, macOS 또는 Unix의 경우는 다음과 같습니다:

```
aws pi get-resource-metrics \
   --service-type DOCDB \
   --identifier db-ID \
   --start-time 2022-03-13T8:00:00Z \
   --end-time   2022-03-13T9:00:00Z \
   --period-in-seconds 60 \
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

Windows의 경우:

```
aws pi get-resource-metrics ^
   --service-type DOCDB ^
   --identifier db-ID ^
   --start-time 2022-03-13T8:00:00Z ^
   --end-time   2022-03-13T9:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

--metrics-query 옵션에 대해 파일을 지정하면 명령이 더 쉽게 읽히도록 할 수 있습니다. 다음 예에서는 옵션에 대해 query.json이라는 파일을 사용합니다. 이 파일의 콘텐츠는 다음과 같습니다.

```
[
    {
        "Metric": "os.cpuUtilization.user.avg"
    },
    {
        "Metric": "os.cpuUtilization.idle.avg"
    }
]
```

다음 명령을 실행하여 파일을 사용합니다.

Linux, macOS 또는 Unix의 경우는 다음과 같습니다:

```
aws pi get-resource-metrics \
   --service-type DOCDB \
   --identifier db-ID \
   --start-time 2022-03-13T8:00:00Z \
   --end-time   2022-03-13T9:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

Windows의 경우:

```
aws pi get-resource-metrics ^
   --service-type DOCDB ^
   --identifier db-ID ^
   --start-time 2022-03-13T8:00:00Z ^
   --end-time   2022-03-13T9:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

앞의 예에서는 옵션에 다음 값을 지정합니다:

- --service-type – Amazon DocumentDB 용 DOCDB
- --identifier – DB 인스턴스에 대한 리소스 ID입니다
- --start-time 및 --end-time – 쿼리할 기간에 대한 ISO 8601 DateTime 값으로서, 지원되는 형식은 여러 가지입니다

다음과 같이 1시간 범위로 쿼리합니다:

- `--period-in-seconds` – 1분당 쿼리에 대한 `60`
- `--metric-queries` – 쿼리 2개의 배열, 각 쿼리는 지표 1개에만 해당됨.

  지표 이름에는 지표를 유용한 범주로 분류하기 위해 점이 사용되고, 마지막 요소는 함수입니다. 예시에서 함수는 각 쿼리에 대해 `avg`입니다. Amazon CloudWatch와 마찬가지로 지원되는 함수는 `min`, `max`, `total` 및 `avg`입니다.

응답은 다음과 비슷합니다.

```
{
    "AlignedStartTime": "2022-03-13T08:00:00+00:00",
    "AlignedEndTime": "2022-03-13T09:00:00+00:00",
    "Identifier": "db-NQF3TTMFQ3GTOKIMJODMC3KQQ4",
    "MetricList": [
        {
            "Key": {
                "Metric": "os.cpuUtilization.user.avg"
            },
            "DataPoints": [
                {
                    "Timestamp": "2022-03-13T08:01:00+00:00", //Minute1
                    "Value": 3.6
                },
                {
                    "Timestamp": "2022-03-13T08:02:00+00:00", //Minute2
                    "Value": 2.6
                },
                //.... 60 datapoints for the os.cpuUtilization.user.avg metric
        {
            "Key": {
                "Metric": "os.cpuUtilization.idle.avg"
            },
            "DataPoints": [
                {
                    "Timestamp": "2022-03-13T08:01:00+00:00",
                    "Value": 92.7
                },
                {
                    "Timestamp": "2022-03-13T08:02:00+00:00",
                    "Value": 93.7
```

```
                },
                //.... 60 datapoints for the os.cpuUtilization.user.avg metric
            ]
        }
    ] //end of MetricList
} //end of response
```

응답에는 Identifier, AlignedStartTime 및 AlignedEndTime이 있습니다. --period-in-seconds 값이 60인 경우 시작 및 종료 시간은 분 단위로 맞춰져 있습니다. --period-in-seconds 값이 3600인 경우 시작 및 종료 시간은 시간 단위로 맞춰져 있습니다.

응답의 MetricList에는 다수의 항목이 있는데, 각각 Key 및 DataPoints 항목이 포함되어 있습니다. 각 DataPoint에는 Timestamp 및 Value이 있습니다. 쿼리는 1시간에 걸친 분당 데이터에 대한 것이므로 각 Datapoints 목록에는 Timestamp1/Minute1, Timestamp2/Minute2 등에서 최대 Timestamp60/Minute60까지 60개의 데이터 포인트가 있습니다.

쿼리는 두 가지 카운터 지표에 대한 것이므로 MetricList 응답에는 두 개의 요소가 있습니다.

상위 대기 상태에 대한 DB 로드 평균 검색

다음 예제는가 누적 영역 선 그래프를 생성하는 데 AWS Management Console 사용하는 것과 동일한 쿼리입니다. 이 예제에서는 로드를 상위 7개 대기 상태에 따라 나눈 마지막 시간 동안 db.load.avg을 검색합니다. 명령은 카운터 지표 검색의 명령과 동일합니다. 그러나 query.json 파일의 컨텐츠는 다음과 같습니다.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_state", "Limit": 7 }
    }
]
```

다음 명령을 실행합니다.

Linux, macOS 또는 Unix의 경우는 다음과 같습니다:

```
aws pi get-resource-metrics \
   --service-type DOCDB \
   --identifier db-ID \
   --start-time 2022-03-13T8:00:00Z \
   --end-time   2022-03-13T9:00:00Z \
   --period-in-seconds 60 \
```

```
   --metric-queries file://query.json
```

Windows의 경우:

```
aws pi get-resource-metrics ^
   --service-type DOCDB ^
   --identifier db-ID ^
   --start-time 2022-03-13T8:00:00Z ^
   --end-time   2022-03-13T9:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

예제에서는 상위 7개 대기 상태 중 db.load.avg과 GroupBy의 메트릭을 지정합니다. 이 예의 유효 값에 대한 자세한 내용은 성능 개선 도우미 API 참조의 DimensionGroup을 참조하십시오.

응답은 다음과 비슷합니다.

```
{
    "AlignedStartTime": "2022-04-04T06:00:00+00:00",
    "AlignedEndTime": "2022-04-04T06:15:00+00:00",
    "Identifier": "db-NQF3TTMFQ3GTOKIMJODMC3KQQ4",
    "MetricList": [
        {//A list of key/datapoints
            "Key": {
                //A Metric with no dimensions. This is the total db.load.avg
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": "2022-04-04T06:01:00+00:00",//Minute1
                    "Value": 0.0
                },
                {
                    "Timestamp": "2022-04-04T06:02:00+00:00",//Minute2
                    "Value": 0.0
                },
                //... 60 datapoints for the total db.load.avg key
                ]
        },
        {
            "Key": {
```

```
                //Another key. This is db.load.avg broken down by CPU
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_state.name": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": "2022-04-04T06:01:00+00:00",//Minute1
                    "Value": 0.0
                },
                {
                    "Timestamp": "2022-04-04T06:02:00+00:00",//Minute2
                    "Value": 0.0
                },
                //... 60 datapoints for the CPU key
            ]
        },//... In total we have 3 key/datapoints entries, 1) total, 2-3) Top Wait
 States
    ] //end of MetricList
} //end of response
```

이 응답에는 MetricList에 항목이 3개 있습니다. 총 db.load.avg에 대해 하나의 엔트리가 있으며, 상위 3개 대기 상태 중 하나에 따라 나누어진 db.load.avg에 대해 각각 3개의 엔트리가 있습니다. 그룹화 차원이 있었기 때문에(첫 번째 예제와 달리) 메트릭의 각 그룹화에는 하나의 키가 있어야 합니다. 기본 카운터 지표 사용 사례처럼 각 지표에 키가 한 개만 있을 수는 없습니다.

상위 쿼리에 대한 DB 평균 로드 검색

다음 예에서는 상위 10개 쿼리 문을 기준으로 db.wait_state를 그룹화합니다. SQL 문에는 두 가지 그룹이 있습니다:

- db.query – {"find":"customers","filter":{"FirstName":"Jesse"},"sort": {"key":{"$numberInt":"1"}}}와 같은 전체 쿼리문
- db.query_tokenized – {"find":"customers","filter": {"FirstName":"?"},"sort":{"key":{"$numberInt":"?"}},"limit": {"$numberInt":"?"}}와 같은 토큰화된 쿼리문

데이터베이스 성능을 분석할 때는 파라미터에 의해서만 다른 쿼리문을 하나의 논리 항목으로 고려하는 것이 유용할 수 있습니다. 따라서 쿼리 시에는 db.query_tokenized를 사용할 수 있습니다. 그러

나 특히 explain()에 관심이 있는 경우에는 때때로 파라미터가 있는 전체 쿼리 문을 검토하는 것이 더 유용합니다. 토큰화된 쿼리와 전체 쿼리 사이에는 부모-자녀 관계가 있으며, 여러 전체 쿼리(자녀)가 동일한 토큰화된 쿼리(부모) 아래에 그룹화됩니다.

이 예의 명령은 상위 대기 상태에 대한 DB 로드 평균 검색의 명령과 유사합니다. 그러나 query.json 파일의 컨텐츠는 다음과 같습니다.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.query_tokenized", "Limit": 10 }
    }
]
```

다음 예에는 db.query_tokenized가 사용됩니다.

Linux, macOS 또는 Unix의 경우는 다음과 같습니다:

```
aws pi get-resource-metrics \
   --service-type DOCDB \
   --identifier db-ID \
   --start-time 2022-03-13T8:00:00Z \
   --end-time   2022-03-13T9:00:00Z \
   --period-in-seconds 3600 \
   --metric-queries file://query.json
```

Windows의 경우:

```
aws pi get-resource-metrics ^
   --service-type DOCDB ^
   --identifier db-ID ^
   --start-time 2022-03-13T8:00:00Z ^
   --end-time   2022-03-13T9:00:00Z  ^
   --period-in-seconds 3600 ^
   --metric-queries file://query.json
```

이 예에서는 1시간 동안 쿼리를 실행하는데 1분은 초 단위로 구성됩니다.

예제에서는 상위 7개 대기 상태 중 db.load.avg과 GroupBy의 메트릭을 지정합니다. 이 예의 유효 값에 대한 자세한 내용은 성능 개선 도우미 API 참조의 DimensionGroup을 참조하십시오.

응답은 다음과 비슷합니다.

```
{
    "AlignedStartTime": "2022-04-04T06:00:00+00:00",
    "AlignedEndTime": "2022-04-04T06:15:00+00:00",
    "Identifier": "db-NQF3TTMFQ3GTOKIMJODMC3KQQ4",
    "MetricList": [
        {//A list of key/datapoints
            "Key": {
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                //... 60 datapoints for the total db.load.avg key
                ]
        },
               {
            "Key": {//Next key are the top tokenized queries
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.query_tokenized.db_id": "pi-1064184600",
                    "db.query_tokenized.id": "77DE8364594EXAMPLE",
                    "db.query_tokenized.statement": "{\"find\":\"customers\",\"filter
\":{\"FirstName\":\"?\"},\"sort\":{\"key\":{\"$numberInt\":\"?\"}},\"limit\"
:{\"$numberInt\":\"?\"},\"$db\":\"myDB\",\"$readPreference\":{\"mode\":\"primary\"}}"
                }
            },
            "DataPoints": [
            //... 60 datapoints
            ]
        },
        // In total 11 entries, 10 Keys of top tokenized queries, 1 total key
    ] //End of MetricList
} //End of response
```

이 응답은 `MetricList`에 11개의 항목이 있는데 (전체 1개, 최상위 토큰화 SQL 10개) 각 항목에는 시간당 `DataPoints`가 24개입니다.

토큰화된 커리의 경우 각 차원 목록에 3개의 항목이 있습니다:

- `db.query_tokenized.statement` – 토큰화된 쿼리문.
- `db.query_tokenized.db_id` — 성능 개선 도우미에서 자동으로 생성하는 합성 ID입니다. 이 예에서는 `pi-1064184600` 합성 ID를 반환합니다.

- db.query_tokenized.id – 성능 개선 도우미 내부의 쿼리에 대한 ID입니다.

  에서 AWS Management Console이 ID를 지원 ID라고 합니다. ID는 AWS Support에서 데이터베이스 문제를 해결하는 데 도움이 되도록 검사할 수 있는 데이터이기 때문에이 이름이 지정됩니다.는 데이터의 보안 및 개인 정보를 매우 심각하게 AWS 받아들이며 거의 모든 데이터는 로 암호화되어 저장됩니다 AWS KMS key. 따라서 내부의 누구도이 데이터를 볼 AWS 수 없습니다. 앞의 예에서 tokenized.statement와 tokenized.db_id 모두 암호화되어 저장됩니다. 데이터베이스에 문제가 있는 경우 AWS Support ID를 참조하여 Support에서 도움을 받을 수 있습니다.

쿼리 시 Group에서 GroupBy을 지정하면 편리할 수 있습니다. 그러나 반환되는 데이터에 대한 더 세분화된 제어를 위해서는 차원 목록을 지정하십시오. 예를 들어 db.query_tokenized.statement만 필요한 경우에는 query.json file에 Dimensions 속성을 추가할 수 있습니다.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": {
            "Group": "db.query_tokenized",
            "Dimensions":["db.query_tokenized.statement"],
            "Limit": 10
        }
    }
]
```

쿼리로 필터링된 DB 로드 평균 검색

이 예에서 해당되는 API 쿼리는 상위 쿼리에 대한 DB 평균 로드 검색의 명령과 유사합니다. 그러나 query.json 파일의 컨텐츠는 다음과 같습니다.

```
[
 {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_state", "Limit": 5  },
        "Filter": { "db.query_tokenized.id": "AKIAIOSFODNN7EXAMPLE" }
    }
]
```

이 응답에서는 query.json 파일에 지정된 토큰화된 쿼리 AKIAIOSFODNN7EXAMPLE의 기여도에 따라 모든 값이 필터링됩니다. 키는 필터링된 쿼리에 영향을 준 상위 5개 대기 상태이기 때문에 필터가 없는 쿼리와 다른 순서를 따를 수도 있습니다.

성능 개선 도우미를 위한 Amazon CloudWatch 지표

성능 개선 도우미는 Amazon CloudWatch에 지표를 자동으로 게시합니다. 동일한 데이터는 성능 개선 도우미에서 쿼리할 수 있지만 CloudWatch에 지표가 있으면 CloudWatch 경보를 더 쉽게 추가할 수 있습니다. 또한 기존 CloudWatch 대시보드에 지표를 더 쉽게 추가할 수 있습니다.

측정치	설명
DBLoad	Amazon DocumentDB의 활성 세션 수입니다. 일반적으로 사용자는 활성 세션의 평균 개수에 대한 데이터를 원합니다. 성능 개선 도우미에서 이 데이터는 `db.load.avg` 로 쿼리됩니다.
DBLoadCPU	대기 상태 유형이 CPU인 활성 세션 수입니다. 성능 개선 도우미에서 이 데이터는 `db.load.avg` 로 쿼리되며 대기 상태 유형인 CPU를 기준으로 필터링됩니다.
DBLoadNonCPU	대기 상태 유형이 CPU가 아닌 활성 세션 수입니다.

> **Note**
>
> 이러한 메트릭은 DB 인스턴스에 로드가 있는 경우에만 CloudWatch에 게시됩니다.

CloudWatch 콘솔, AWS CLI또는 CloudWatch API를 사용하여 이러한 지표를 검사할 수 있습니다.

예를 들어, get-metric-statistics 명령을 실행하여 DBLoad 지표에 대한 통계를 가져올 수 있습니다.

```
aws cloudwatch get-metric-statistics \
    --region ap-south-1 \
    --namespace AWS/DocDB \
    --metric-name DBLoad  \
    --period 360 \
```

```
    --statistics Average \
    --start-time 2022-03-14T8:00:00Z \
    --end-time 2022-03-14T9:00:00Z \
    --dimensions Name=DBInstanceIdentifier,Value=documentdbinstance
```

이 예에서는 다음과 비슷한 출력이 생성됩니다.

```
{
    "Datapoints": [
        {
            "Timestamp": "2022-03-14T08:42:00Z",
            "Average": 1.0,
            "Unit": "None"
        },
        {
            "Timestamp": "2022-03-14T08:24:00Z",
            "Average": 2.0,
            "Unit": "None"
        },
        {
            "Timestamp": "2022-03-14T08:54:00Z",
            "Average": 6.0,
            "Unit": "None"
        },
        {
            "Timestamp": "2022-03-14T08:36:00Z",
            "Average": 5.7,
            "Unit": "None"
        },
        {
            "Timestamp": "2022-03-14T08:06:00Z",
            "Average": 4.0,
            "Unit": "None"
        },
        {
            "Timestamp": "2022-03-14T08:00:00Z",
            "Average": 5.2,
            "Unit": "None"
        }
    ],
    "Label": "DBLoad"
}
```

CloudWatch 콘솔에서 DB_PERF_INSIGHTS 메트릭 산술 함수를 사용하여 Amazon DocumentDB 성능 개선 도우미 카운터 메트릭을 쿼리할 수 있습니다. DB_PERF_INSIGHTS 함수에는 분 단위로 DBLoad 메트릭도 포함됩니다. 이러한 지표에 대해 CloudWatch 경보를 설정할 수 있습니다. 경보를 생성하는 방법에 대한 자세한 내용은 AWS 데이터베이스에서 성능 개선 도우미 카운터 지표에 대한 경보 생성을 참조하세요.

CloudWatch에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch란 무엇입니까?를 참조하세요.

카운터 지표에 대한 성능 개선 도우미

카운터 메트릭은 성능 개선 도우미 대시보드의 운영 체제 메트릭입니다. 이 정보와 데이터베이스 로드를 연관 지으면 성능 문제를 식별하고 분석하는 데 도움이 됩니다.

성능 개선 도우미 운영 체제 카운터

Amazon DocumentDB 성능 개선 도우미와 함께 사용할 수 있는 운영 체제 카운터는 다음과 같습니다.

카운터	유형	측정치
활성화	메모리	os.memory.active
버퍼	메모리	os.memory.buffers
캐시됨	메모리	os.memory.cached
더티	메모리	os.memory.dirty
사용 가능	메모리	os.memory.free
비활성	메모리	os.memory.inactive
매핑됨	메모리	os.memory.mapped
pageTables	메모리	os.memory.pageTables
슬래브	메모리	os.memory.slab
총합	메모리	os.memory.total
writeback	메모리	os.memory.writeback

카운터	유형	측정치
유휴	cpuUtilization	os.cpuUtilization.idle
시스템	cpuUtilization	os.cpuUtilization.system
총합	cpuUtilization	os.cpuUtilization.total
사용자	cpuUtilization	os.cpuUtilization.user
대기	cpuUtilization	os.cpuUtilization.wait
1	loadAverageMinute	os.loadAverageMinute.one
15	loadAverageMinute	os.loadAverageMinute.fifteen
5	loadAverageMinute	os.loadAverageMinute.five
캐시됨	스왑	os.swap.cached
사용 가능	스왑	os.swap.free
인	스왑	os.swap.in
아웃	스왑	os.swap.out
총합	스왑	os.swap.total
rx	네트워크	os.network.rx
tx	네트워크	os.network.tx
numVCPUs	일반	os.general.numVCPUs

Amazon DocumentDB Serverless 사용

Amazon DocumentDB Serverless는 애플리케이션 수요에 따라 Amazon DocumentDB 데이터베이스의 용량을 동적으로 조정하는 프로세스를 자동화하는 온디맨드 오토 스케일링 구성입니다. 클러스터가 사용하는 리소스에 대해서만 청구됩니다. 따라서 DocumentDB Serverless는 예산 범위를 벗어나지 않도록 하고 사용하지 않는 컴퓨터 리소스에 대한 비용을 지불하지 않도록 하는 데 도움이 됩니다.

이러한 유형의 자동화는 멀티테넌트 데이터베이스, 분산 데이터베이스, 개발 및 테스트 시스템과 같이 매우 가변적이고 예측할 수 없는 워크로드가 있는 환경에 특히 유용합니다.

주제

- DocumentDB Serverless 사용 사례
- Amazon DocumentDB Serverless의 장점
- Amazon DocumentDB Serverless 작동 방식
- DocumentDB Serverless 요구 사항 및 제한 사항
- Amazon DocumentDB Serverless를 사용하는 클러스터 생성
- Amazon DocumentDB Serverless로 마이그레이션
- Amazon DocumentDB Serverless 관리
- Amazon DocumentDB Serverless 인스턴스 제한
- Amazon DocumentDB Serverless 규모 조정 구성
- Amazon DocumentDB Serverless 모니터링

DocumentDB Serverless 사용 사례

Amazon DocumentDB 프로비저닝 클러스터와 DocumentDB Serverless는 개발 및 테스트 환경부터 높은 확장성과 가용성이 요구되는 가장 까다로운 비즈니스 크리티컬 애플리케이션에 이르기까지 다양한 유형의 데이터베이스 워크로드를 지원합니다. 그러나 DocumentDB Serverless는 고객의 워크로드에 또 다른 차원, 즉 예측할 수 없는 워크로드를 가진 웹 사이트 및 애플리케이션을 지원하는 능력을 더해줍니다.

DocumentDB Serverless는 다음과 같은 사용 사례에 특히 유용합니다.

- 가변 워크로드 - 작업의 증가가 갑작스럽고 예측할 수 없는 워크로드를 실행하고 있는 경우입니다. 비가 내리기 시작하면 활동이 급증하는 트래픽 사이트를 예로 들 수 있습니다. 또 다른 하나는 세일 또는 특별 프로모션을 제공할 때 트래픽이 증가하는 전자 상거래 사이트입니다. DocumentDB

Serverless를 사용하면 애플리케이션의 피크 로드를 처리하는 데 필요한 용량을 충족하도록 데이터베이스가 용량을 자동으로 확장하고 활동이 급증하는 시점이 지나면 용량을 다시 줄입니다. DocumentDB Serverless를 사용하면 더 이상 피크 또는 평균 용량에 맞추어 프로비저닝하지 않아도 됩니다. 최악의 상황을 처리하기 위해 용량 상한을 지정할 수 있으며, 필요한 경우가 아니면 그 용량이 사용되지 않습니다.

  - DocumentDB Serverless의 세분화된 스케일링 덕분에 데이터베이스 요구 사항과 용량을 매우 비슷하게 일치시키는 데 도움이 됩니다. 프로비저닝된 클러스터의 경우 확장하려면 완전히 새로운 인스턴스를 추가해야 합니다. DocumentDB Serverless는 약간 더 많은 용량만 필요할 때 절반의 DCU를 추가할 수 있습니다. 워크로드의 증가를 처리하는 데 필요한 추가 용량에 따라 ACU의 0.5, 1, 1.5, 2와 같이 0.5 단위로 DCU를 추가할 수 있습니다. 또한 워크로드가 감소하고 용량이 더 이상 필요하지 않을 때 0.5, 1, 1.5, 2와 같이 0.5 단위로 DCU를 제거할 수 있습니다.
- 멀티태넌트 애플리케이션 - DocumentDB Serverless를 사용하면 플릿에서 각 애플리케이션에 대한 데이터베이스 용량을 개별적으로 관리할 필요가 없습니다. DocumentDB Serverless는 사용자를 대신해 개별 데이터베이스 용량을 관리합니다.
  - 테넌트별로 클러스터를 생성할 수 있습니다. 이렇게 하면 복제나 스냅샷 복구와 같은 기능을 사용하여 각 테넌트에 적합하도록 고가용성 및 재해 복구를 향상시킬 수 있습니다.
  - 각 테넌트는 하루 중 시간, 연중 시기, 프로모션 이벤트 등에 따라 바쁜 기간과 유휴 기간이 있을 수 있습니다. 각 클러스터는 용량 범위가 넓을 수 있습니다. 따라서 활동이 적은 클러스터는 인스턴스 요금이 최소화됩니다. 어느 클러스터든 높은 활동 기간을 처리하도록 빠르게 확장할 수 있습니다.
- 새 애플리케이션 - 새 애플리케이션을 배포하는 중인데, 필요한 인스턴스 크기를 잘 모릅니다. DocumentDB Serverless를 사용하면 인스턴스를 하나 이상 사용하여 클러스터를 설정하고 애플리케이션의 필요 용량에 따라 데이터베이스의 크기를 자동으로 조정할 수 있습니다.
- 혼용 애플리케이션 - 온라인 트랜잭션 처리(OLTP) 애플리케이션이 있지만 쿼리 트래픽이 주기적으로 급증한다고 가정하겠습니다. 클러스터에서 DocumentDB Serverles에 대한 승격 티어를 지정하면 리더 인스턴스가 라이터 인스턴스와 독립적으로 조정되어 추가 로드를 처리할 수 있도록 클러스터를 구성할 수 있습니다. 사용량이 줄어들면 리더 인스턴스는 라이터 인스턴스의 용량에 맞게 스케일 다운됩니다.
- 용량 계획 - 클러스터에 있는 모든 인스턴스의 인스턴스 클래스를 수정하여 일반적으로 데이터베이스 용량을 조정하거나 워크로드에 가장 적합한 데이터베이스 용량을 확인한다고 가정해 보겠습니다. DocumentDB Serverless를 사용하면 이런 관리 오버헤드 발생을 방지할 수 있습니다. 워크로드를 실행하고 인스턴스가 실제로 얼마나 조정되는지 확인하여 적절한 최소 및 최대 용량을 결정할 수 있습니다.
  - 기존 인스턴스를 프로비저닝된 인스턴스에서 DocumentDB Serverless로 또는 DocumentDB Serverless에서 프로비저닝된 인스턴스로 수정할 수 있습니다. 이런 경우 새 클러스터나 새 인스턴스를 생성할 필요가 없습니다.

- 개발 및 테스트 - 가장 까다로운 애플리케이션을 실행하는 것 외에도 개발 및 테스트 환경에 DocumentDB Serverless를 사용할 수 있습니다. DocumentDB Serverless를 사용하면 버스트 가능한 db.t* 인스턴스 클래스를 사용하는 대신 최소 용량이 낮은 인스턴스를 생성할 수 있습니다. 최대 용량을 충분히 높게 설정하여 해당 인스턴스에서 메모리가 부족하지 않게 상당한 워크로드를 실행할 수 있습니다. 데이터베이스를 사용하지 않을 때는 불필요한 요금을 피하기 위해 모든 인스턴스가 스케일 다운됩니다.

기존 프로비저닝된 워크로드에 Amazon DocumentDB Serverless 사용

프로비저닝된 클러스터에서 이미 DocumentDB 애플리케이션이 실행 중이라고 가정합시다. 기존 클러스터에 리더 인스턴스로 하나 이상의 DocumentDB Serverless 인스턴스를 추가하여 애플리케이션이 DocumentDB Serverless와 어떻게 작동하는지 확인할 수 있습니다. 리더 인스턴스가 얼마나 자주 스케일 업 및 스케일 다운되는지 확인할 수 있습니다. DocumentDB 장애 조치 메커니즘을 사용하여 DocumentDB Serverless 인스턴스가 라이터가 되도록 승격하고 읽기/쓰기 워크로드를 어떻게 처리하는지 확인합니다. 그러면 클라이언트 애플리케이션이 사용하는 엔드포인트를 변경하지 않고도 가동 중지 시간을 최소화하면서 전환할 수 있습니다. 기존 클러스터를 DocumentDB Serverless로 변환하는 절차에 대한 자세한 내용은 Amazon DocumentDB Serverless로 마이그레이션 섹션을 참조하세요.

Amazon DocumentDB Serverless의 장점

DocumentDB Serverless는 가변 또는 “급증” 워크로드를 위한 것입니다. 예측할 수 없는 워크로드로 인해 데이터베이스 용량을 변경하는 시기를 계획하는 데 어려움이 있을 수 있습니다. 또한 인스턴스 추가 또는 인스턴스 클래스 변경과 같은 익숙한 메커니즘을 사용하여 용량을 빠르게 변경하는 데 문제가 있을 수 있습니다. DocumentDB Serverless는 이러한 사용 사례에 도움이 되도록 다음과 같은 이점을 제공합니다.

- 프로비저닝된 클러스터보다 간편한 용량 관리 - DocumentDB Serverless를 사용하면 워크로드 변경에 따라 인스턴스 크기를 계획하고 인스턴스의 크기를 조정하기 위한 노력이 줄어듭니다. 또한 클러스터에 속한 모든 인스턴스에 일관된 용량을 유지하기 위한 노력도 줄여줍니다.

- 활동이 많은 기간 동안 더 빠르고 쉽게 조정 - DocumentDB Serverless는 클라이언트 트랜잭션이나 전체 워크로드에 영향을 주지 않고 필요에 따라 컴퓨팅 및 메모리 용량을 조정합니다. 리더 인스턴스를 DocumentDB Serverless와 함께 사용할 수 있으므로 수직적 스케일링뿐만 아니라 수평적 스케일링도 활용할 수 있습니다. DocumentDB 글로벌 클러스터를 사용할 수 있다는 것은 DocumentDB 서버리스 읽기 워크로드를 여러에 분산할 수 있음을 의미합니다 AWS 리전. 이 기능은 프로비저닝된 클러스터의 크기 조정 메커니즘보다 편리합니다.

- 활동이 적은 기간 동안의 비용 효율성 - DocumentDB Serverless는 인스턴스의 과다 프로비저닝을 방지하는 데 도움을 줍니다. DocumentDB Serverless는 인스턴스를 스케일 업할 때 리소스를 세분화된 단위로 추가합니다. 사용한 데이터베이스 리소스에 대해서만 요금을 지불합니다. DocumentDB Serverless 리소스 사용량은 초 단위로 측정됩니다. 따라서 인스턴스가 스케일 다운되면 리소스 사용량 감소가 즉시 등록됩니다.

- 프로비저닝과 동등한 기능 패리티 - DocumentDB Serverless에서 모든 DocumentDB 기능을 사용할 수 있습니다. 예를 들어 DocumentDB 서버리스에서는 리더 인스턴스, AWS Identity and Access Management (IAM) 데이터베이스 인증 및 성능 개선 도우미를 사용할 수 있습니다.

  특히 DocumentDB Serverless를 사용하면 프로비저닝된 클러스터로부터 다음과 같은 기능을 활용할 수 있습니다.

  - 리더 인스턴스 - DocumentDB Serverless는 리더 인스턴스를 활용하여 수평으로 조정할 수 있습니다. 클러스터에 하나 이상의 리더 DB 인스턴스가 포함된 경우 라이터 DB 인스턴스에 문제가 발생하면 클러스터가 즉시 장애 조치를 할 수 있습니다.
  - Multi-AZ 클러스터 - 한 클러스터의 DocumentDB Serverless 인스턴스를 여러 개의 가용 영역(AZ)에 배포할 수 있습니다. 다중 AZ 클러스터를 설정하면 드물게 전체 AZ에 영향을 주는 경우가 발생해도 비즈니스 연속성을 보장할 수 있습니다.
  - 글로벌 클러스터 - DocumentDB 서버리스를 DocumentDB 글로벌 클러스터와 함께 사용하여 재해 복구를 AWS 리전 위해 다른에 클러스터의 읽기 전용 복사본을 추가로 생성할 수 있습니다.

Amazon DocumentDB Serverless 작동 방식

주제

- 개요
- Amazon DocumentDB 클러스터에 대한 구성
- Amazon DocumentDB Serverless 규모 조정 용량
- Amazon DocumentDB Serverless 규모 조정
- 유휴 상태(0.5DCU)

개요

Amazon DocumentDB Serverless는 가장 까다롭고 매우 가변적인 워크로드에 적합합니다. 예를 들어 데이터베이스 사용량이 짧은 시간 동안 높게 나타나고 오랜 시간 동안 작업이 적거나 전혀 작업이 없을

수 있습니다. 정기 프로모션 이벤트를 진행하는 소매점, 게임 또는 스포츠 웹 사이트와 필요한 경우 보고서를 생성하는 데이터베이스가 예로 들 수 있습니다. 이외에도 개발 및 테스트 환경과 사용량이 빠르게 증가할 수 있는 새로운 애플리케이션도 있습니다. 이 경우와 다른 많은 경우에는 프로비저닝된 모델에서 용량을 미리 적절하게 구성하지 못할 수도 있습니다. 또한 초과 프로비저닝하여 사용하지 않는 용량이 있는 경우 비용이 더 많이 발생할 수 있습니다.

대조적으로, DocumentDB 프로비저닝된 클러스터는 안정적인 워크로드에 적합합니다. 프로비저닝된 클러스터에서는 미리 정의된 메모리 양, CPU 성능, I/O 대역폭 등이 있는 인스턴스 클래스를 선택합니다. 워크로드가 변경되면 라이터 및 리더의 인스턴스 클래스를 수동으로 수정합니다. 프로비저닝된 모델은 예상 소비 패턴에 앞서 용량을 조정할 수 있는 경우에 적합하며 클러스터에서 라이터 및 리더의 인스턴스 클래스를 변경하는 동안 발생하는 잠시 중단을 허용할 때 잘 작동합니다.

DocumentDB Serverless는 처음부터 즉시 확장 가능한 Serverless 클러스터를 지원하도록 설계되었습니다. DocumentDB Serverless는 프로비저닝된 라이터 및 리더와 동일한 수준의 보안 및 격리를 제공하도록 설계되었습니다. 이러한 측면은 멀티테넌트 서버리스 클라우드 환경에서 매우 중요합니다. 동적 크기 조정 메커니즘은 오버헤드가 매우 적기 때문에 데이터베이스 워크로드의 변화에 신속하게 대응할 수 있습니다. 또한 처리 수요의 급격한 증가에 대응할 수 있을 만큼 강력합니다.

DocumentDB Serverless를 사용하면 각 라이터 및 리더에 대한 특정 데이터베이스 용량에 얽매이지 않고 DocumentDB 클러스터를 생성할 수 있습니다. 최소 및 최대 용량 범위만 지정합니다. DocumentDB는 해당 용량 범위 내에서 클러스터에 있는 각 DocumentDB Serverless 라이터 또는 리더를 조정합니다. 각 라이터 또는 리더가 동적으로 확장할 수 있는 다중 AZ 클러스터를 사용하면 동적 확장 및 고가용성을 활용할 수 있습니다.

DocumentDB Serverless는 최소 및 최대 용량 사양에 따라 데이터베이스 리소스의 크기를 자동으로 조정합니다. 대부분의 크기 조정 이벤트 작업은 라이터 또는 리더를 동일한 호스트에 유지하기 때문에 확장이 빠릅니다. 드문 경우이지만 DocumentDB Serverless 라이터 또는 리더가 한 호스트에서 다른 호스트로 이동하는 경우 DocumentDB Serverless가 연결을 자동으로 관리합니다. 데이터베이스 클라이언트 애플리케이션 코드나 데이터베이스 연결 문자열을 변경할 필요가 없습니다.

DocumentDB Serverless의 경우 프로비저닝된 클러스터와 마찬가지로 스토리지 용량과 컴퓨팅 용량이 분리됩니다. DocumentDB Serverless 용량 및 규모 조정을 언급할 때 항상 컴퓨팅 용량이 증가하거나 감소하는 것을 의미합니다. 따라서 CPU 및 메모리 용량이 낮은 수준으로 축소되더라도 클러스터에 수 테라바이트의 데이터가 포함될 수 있습니다.

데이터베이스 서버를 프로비저닝 및 관리하는 대신 데이터베이스 용량을 지정합니다. 각 DocumentDB Serverless 라이터 또는 리더의 실제 용량은 워크로드를 바탕으로 시간이 지남에 따라 달라집니다. 이러한 메커니즘에 대한 자세한 내용은 Amazon DocumentDB Serverless 규모 조정의 내용을 참조하세요.

Amazon DocumentDB 클러스터에 대한 구성

각 DocumentDB Serverless 클러스터에 대해 DocumentDB Serverless 용량 또는 프로비저닝된 용량을 선택하거나 두 가지 모두를 조합해서 선택할 수 있습니다.

혼합 구성 클러스터라고 하는 DocumentDB Serverless 및 프로비저닝된 용량을 모두 포함하는 클러스터를 설정할 수 있습니다. 예를 들어, DocumentDB Serverless 라이터에서 사용할 수 있는 것보다 많은 읽기/쓰기 용량이 필요하다고 가정하겠습니다. 이 경우 대규모로 프로비저닝된 라이터를 사용하여 클러스터를 설정할 수 있습니다. 그런 다음 리더에 DocumentDB Serverless를 계속 사용할 수 있습니다. 또는 클러스터의 쓰기 워크로드는 다양하지만 읽기 워크로드는 일정하다고 가정합니다. 이 경우 DocumentDB Serverless 라이터와 하나 이상의 프로비저닝된 리더로 클러스터를 설정할 수 있습니다.

모든 용량을 DocumentDB Serverless에서 관리하는 클러스터를 설정할 수도 있습니다. 이렇게 하려면 새 클러스터를 생성하고 처음부터 DocumentDB Serverless를 사용할 수 있습니다. 또는 기존 클러스터에서 프로비저닝된 모든 용량을 DocumentDB Serverless로 교체할 수 있습니다. DocumentDB Serverless로 새 클러스터를 생성하거나 기존 클러스터를 DocumentDB Serverless로 전환하는 절차는 Amazon DocumentDB Serverless를 사용하는 클러스터 생성 및 Amazon DocumentDB Serverless로 마이그레이션의 내용을 참조하세요.

클러스터에서 DocumentDB Serverless를 전혀 사용하지 않으면 클러스터의 모든 라이터와 리더가 프로비저닝됩니다. 이것은 대부분의 사용자에게 친숙하고 가장 일반적인 클러스터입니다. 프로비저닝된 용량은 일정합니다. 요금은 비교적 쉽게 예측할 수 있습니다. 그러나 얼마나 많은 용량이 필요한지는 미리 예측해야 합니다. 경우에 따라 예측이 부정확하거나 용량 요구 사항이 변경될 수 있습니다. 이러한 경우 클러스터가 언더프로비저닝(기대치보다 더 느림) 또는 오버프로비저닝(기대치보다 더 비쌈)될 수 있습니다.

Amazon DocumentDB Serverless 규모 조정 용량

Amazon DocumentDB Serverless의 측정 단위는 DocumentDB 용량 단위(DCU)입니다. DocumentDB Serverless 조정 용량은 프로비저닝된 클러스터에 사용하는 인스턴스 클래스에 종속되지 않습니다.

각 DCU는 약 2기가바이트(GB)의 메모리로 해당 CPU 및 네트워킹의 합한 용량입니다. 이 측정 단위를 사용하여 데이터베이스 용량 범위를 지정합니다. `ServerlessDatabaseCapacity` 및 `DCUUtilization` CloudWatch 지표는 데이터베이스가 실제로 사용하고 있는 용량과 해당 용량이 지정된 범위에 속하는 위치를 결정하는 데 도움이 됩니다.

언제든지 각 DocumentDB Serverless 라이터 또는 리더에는 용량이 있습니다. 용량은 DCU를 나타내는 부동 소수점 숫자로 표시됩니다. 라이터 또는 리더가 확장될 때마다 용량이 증가하거나 감소합니

다. 이 값은 매초마다 측정됩니다. DocumentDB Serverless를 사용하려는 각 클러스터에 대해 용량 범위, 즉 각 DocumentDB Serverless 라이터 또는 리더가 조정할 수 있는 최소 및 최대 용량 값을 정의합니다. 용량 범위는 클러스터의 각 DocumentDB Serverless 라이터 또는 리더에 대해 동일합니다. 각 DocumentDB Serverless 라이터 또는 리더는 해당 범위의 어딘가에 해당하는 고유한 용량을 가지고 있습니다.

DocumentDB 서버리스는 용량 범위가 0.5~256DCU인 DocumentDB 5.0 및 8.0에서 지원됩니다. DCUs

정의할 수 있는 최소 DocumentDB Serverless 용량은 0.5DCU입니다. 지원되는 최대 용량 값보다 작거나 같은 경우 더 높은 숫자를 지정할 수 있습니다. 최소 용량을 작은 수로 설정하면 로드가 적은 클러스터가 최소한의 컴퓨팅 리소스를 소비할 수 있습니다. 동시에 연결을 즉시 수락하고 사용량이 많을 때 확장할 준비가 되어 있습니다.

최소값을 각 라이터 또는 리더가 버퍼 풀에 애플리케이션의 작업 세트를 보유할 수 있는 값으로 설정합니다. 이렇게 하면 유휴 기간 동안 버퍼 풀의 내용이 삭제되지 않습니다. 규모 조정 범위를 선택할 때 모든 고려 사항은 DocumentDB Serverless 클러스터의 규모 조정 용량 범위 선택 섹션을 참조하세요.

다중 AZ 배포에서 리더를 구성하는 방법에 따라 리더의 용량은 라이터의 용량에 연결되거나 별도로 구성될 수 있습니다. 이를 수행하는 방법에 대한 자세한 내용은 서버리스 리더의 승격 계층 보기 및 수정 페이지를 참조하세요.

DocumentDB Serverless 모니터링에는 시간 경과에 따른 DB 클러스터의 라이터 및 리더 용량 값 측정이 포함됩니다. 데이터베이스가 최소 용량으로 축소되지 않는 경우 최소값을 조정하고 데이터베이스 애플리케이션을 최적화하는 등의 작업을 수행할 수 있습니다. 데이터베이스가 지속적으로 최대 용량에 도달하면 최대값을 늘리는 등의 작업을 수행할 수 있습니다. 또한 데이터베이스 애플리케이션을 최적화하고 더 많은 리더에 쿼리 로드를 분산시킬 수 있습니다.

DocumentDB Serverless 용량에 대한 요금은 DCU 시간으로 측정됩니다. DocumentDB Serverless 요금 계산 방법에 대한 자세한 내용은 Amazon DocumentDB 요금을 참조하세요. 클러스터의 총 라이터 및 리더 수가 n개라고 가정해 보겠습니다. 이 경우 클러스터는 데이터베이스 작업을 실행하지 않을 때 대략적으로 n x 최소 DCU를 사용합니다. DocumentDB 자체는 약간의 로드를 유발하는 모니터링 또는 유지 관리 작업을 실행할 수 있습니다. 해당 클러스터는 데이터베이스가 전체 용량으로 실행 중일 때 n x 최대 DCU만 소비합니다.

적절한 최소 및 최대 DCU 값 선택에 대한 자세한 내용은 DocumentDB Serverless 클러스터의 규모 조정 용량 범위 선택 페이지를 참조하세요. 지정하는 최소 및 최대 DCU 값도 일부 Amazon DocumentDB 인스턴스 제한에 영향을 미칩니다. 용량 범위와 인스턴스 제한 간의 상호 작용에 대한 자세한 내용은 Amazon DocumentDB Serverless 인스턴스 제한 섹션을 참조하세요.

Amazon DocumentDB Serverless 규모 조정

각 DocumentDB Serverless 라이터 또는 리더에 대해 Amazon DocumentDB는 CPU, 메모리 및 네트워크와 같은 리소스 사용률을 지속적으로 추적합니다. 이러한 측정값을 통칭하여 로드라고 합니다. 로드에는 애플리케이션에서 수행한 데이터베이스 작업이 포함됩니다. 또한 데이터베이스 서버 및 Amazon DocumentDB 관리 작업에 대한 백그라운드 처리가 포함됩니다. 이 중 하나로 인해 용량이 제한되면 DocumentDB Serverless가 스케일 업됩니다. DocumentDB Serverless는 이를 통해 해결할 수 있는 성능 문제를 감지하면 스케일 업됩니다. Amazon DocumentDB Serverless 모니터링의 절차를 사용하여 리소스 사용률과 DocumentDB Serverless가 스케일링에 미치는 영향을 모니터링할 수 있습니다.

로드는 클러스터의 라이터와 리더에 따라 다를 수 있습니다. 라이터는 읽기 및 쓰기 작업을 모두 지원하고, 클러스터 볼륨의 모든 데이터 수정을 실행합니다. 리더는 읽기 전용 요청을 처리할 수 있습니다.

규모 조정은 데이터베이스의 DocumentDB Serverless 용량을 늘리거나 줄이는 작업입니다. DocumentDB Serverless를 사용하면 각 라이터와 리더는 각 현재의 용량 값을 DCU 단위로 보유하게 됩니다. DocumentDB Serverless는 현재 용량이 너무 낮아 로드를 처리할 수 없는 경우 라이터 또는 리더를 더 높은 용량으로 스케일 업합니다. 현재 용량이 필요 이상으로 높으면 라이터 또는 리더를 더 낮은 용량으로 축소합니다.

DocumentDB Serverless는 용량을 점진적으로 늘릴 수 있습니다. 워크로드 수요가 라이터 또는 리더의 현재 데이터베이스 용량에 도달하기 시작하면 DocumentDB Serverless는 해당 라이터 또는 리더의 DCU를 늘립니다. DocumentDB Serverless는 사용된 리소스에 대해 최상의 성능을 제공하는 데 필요한 증가분만큼 용량을 조정합니다. 스케일링은 0.5DCU의 작은 단위로 발생합니다. 현재 용량이 클수록 크기 확장분이 커지므로 크기 조정이 빨라질 수 있습니다.

DocumentDB Serverless 스케일링은 매우 빈번하고 세부적이며 중단되지 않기 때문에 AWS Management Console에서처럼 개별 이벤트를 일으키지 않습니다. 대신 `serverlessDatabaseCapacity` 및 `DCUUtilization`과 같은 Amazon CloudWatch 지표를 측정하고 시간 경과에 따른 최소값, 최대값 및 평균값을 추적할 수 있습니다. DocumentDB Serverless 모니터링에 대한 자세한 내용은 Amazon DocumentDB Serverless 모니터링 섹션을 참조하세요.

스케일 업이나 스케일 다운은 다음과 같은 원인으로 발생할 수 있습니다.

- 메모리 사용률
- CPU 사용률
- 네트워크 사용률
- 스토리지 사용률

DocumentDB Serverless 인스턴스에서 이러한 스케일 업/다운의 원인을 모니터링할 수 있습니다. 자세한 내용은 Amazon DocumentDB Serverless 모니터링 단원을 참조하십시오.

연결된 라이터와 동시에 또는 라이터와 독립적으로 리더를 확장하도록 선택할 수 있습니다. 해당 리더에 대한 프로모션 티어를 지정하면 됩니다.

- 승격 티어 0 및 1의 DocumentDB Serverless 리더는 라이터와 동시에 규모가 조정됩니다. 이러한 확장 동작은 우선 순위 티어 0 및 1의 리더가 가용성에 이상적입니다. 장애 조치(failover)의 경우 라이터의 워크로드를 인계할 수 있도록 적절한 용량으로 항상 크기가 조정되기 때문입니다.
- 프로모션 티어 2~15의 리더는 라이터와 독립적으로 확장됩니다. 각 리더는 클러스터에 대해 지정한 최소 및 최대 DCU 값 내에 유지됩니다. 리더가 연결된 라이터 DB와 독립적으로 확장되면 라이터가 계속해서 많은 양의 트랜잭션을 처리하는 동안 유휴 상태가 되어 축소될 수 있습니다. 더 낮은 프로모션 티어에서 사용할 수 있는 다른 판독기가 없는 경우 장애 조치 대상으로 계속 사용할 수 있습니다. 그러나 라이터로 프로모션되는 경우 라이터의 전체 워크로드를 처리하기 위해 확장해야 할 수 있습니다.

서버리스 인스턴스의 승격 티어 보기 및 변경에 대한 자세한 내용은 서버리스 리더의 승격 계층 보기 및 수정 섹션을 참조하세요.

DocumentDB Serverless의 규모 조정은 데이터베이스 연결이 열려 있는 동안, 트랜잭션이 처리 중인 동안, 그리고 그 밖의 상황에서 일어날 수 있습니다. DocumentDB Serverless는 규모 조정을 시작할 조용한 지점이 될 때까지 기다리지 않습니다. 확장은 진행 중인 데이터베이스 작업을 방해하지 않습니다.

워크로드에 단일 라이터 및 단일 리더에서 사용할 수 있는 것보다 더 많은 읽기 용량이 필요한 경우 클러스터에 여러 DocumentDB Serverless 리더를 추가할 수 있습니다. 각 DocumentDB Serverless 리더는 클러스터에 대해 지정한 최소 및 최대 용량 값 범위 내에서 조정할 수 있습니다. 클러스터의 리더 엔드포인트를 사용하여 읽기 전용 세션을 리더로 보내고 라이터의 로드를 줄일 수 있습니다.

DocumentDB Serverless가 규모 조정을 수행하는지 여부와 규모 조정이 시작된 후 일어나는 속도는 클러스터의 최소 및 최대 DCU 설정에 따라 달라집니다. 또한 리더가 라이터와 함께 확장되도록 구성되었는지 또는 독자적으로 구성되었는지에 따라 다릅니다. 규모 조정 구성에 대한 자세한 내용은 Amazon DocumentDB Serverless 규모 조정 구성 섹션을 참조하세요.

유휴 상태(0.5DCU)

Amazon DocumentDB Serverless 라이터 또는 리더가 유휴 상태에 있을 때 클러스터의 MinCapacity가 0.5로 구성되어 있으면 DocumentDB Serverless 인스턴스는 0.5DCU의 유휴 상태로의 스케일 다운을 지원합니다.

유휴 상태에서 DocumentDB Serverless 인스턴스는 대부분의 프로덕션 워크로드를 지원하기에 충분한 CPU 컴퓨팅 용량이 없지만 새 워크로드를 지원하기 위해 빠르게 스케일 업할 준비가 되어 있는 상태입니다. 유휴 상태가 아닌 상태에서 DocumentDB Serverless 인스턴스는 일반적으로 최소 1.0~2.5DCU가 필요합니다. 따라서 DocumentDB Serverless 인스턴스가 유휴 상태에서 유휴 상태가 아닌 상태로 스케일 업되면 1.0~2.5DCU(또는 더 낮은 경우 MaxCapacity 값)로 바로 스케일 업됩니다.

유휴 시 0.5DCU의 스케일 다운을 지원하기 위해 MinCapacity를 1.0DCU 이하로 구성한 경우 인스턴스에 제한이 걸립니다. MinCapacity 구성이 제한에 어떻게 영향을 미치는지에 대한 자세한 내용은 Amazon DocumentDB Serverless 인스턴스 제한 섹션을 참조하세요.

DocumentDB Serverless 요구 사항 및 제한 사항

기능 요구 사항

리전 가용성

Amazon DocumentDB 서버리스 인스턴스 유형을 사용할 수 AWS 리전 있는 위치에 대한 자세한 내용은의 서버리스 열을 참조하세요리전별 지원되는 인스턴스 클래스.

다음 AWS CLI 명령을 사용하여 특정 리전에서 제공되는 정확한 DocumentDB 서버리스 인스턴스 옵션을 확인할 수 있습니다.

```
aws docdb describe-orderable-db-instance-options \
        --region my_region \
        --db-instance-class db.serverless \
        --engine docdb
```

엔진 버전 가용성

DocumentDB 서버리스는 Amazon DocumentDB 5.0 및 8.0 엔진 버전에서 지원됩니다. 엔진 버전 3.6 또는 4.0에서는 사용할 수 없습니다.

> **Note**
>
> DocumentDB Serverless는 최신 5.0.0 패치 버전에서만 지원됩니다. 클러스터가 최신 엔진 패치 버전으로 업데이트되었는지 확인합니다. Amazon DocumentDB 클러스터 유지 관리에 대한 자세한 내용은 Amazon DocumentDB 유지 관리 섹션을 참조하세요.

클러스터 구성

Amazon DocumentDB 클러스터에 Amazon DocumentDB Serverless 인스턴스를 추가하려면 이에 앞서 클러스터에 `ServerlessV2ScalingConfiguration` 파라미터 세트도 있어야 합니다. 이 파라미터 세트는 클러스터 내에 있는 DocumentDB Serverless 인스턴스의 용량 범위를 정의합니다. 규모 조정 구성에 대한 자세한 내용은 Amazon DocumentDB Serverless 규모 조정 구성 섹션을 참조하세요.

특정 Amazon DocumentDB 기능에 대한 최소 규모 조정 용량 범위 설정

일부 Amazon DocumentDB 기능은 DocumentDB Serverless에서 작동하지만, 특정 워크로드가 있는 해당 기능의 메모리 요구 사항에 필요한 것보다 용량 범위가 낮은 경우 문제가 발생할 수 있습니다. 이 경우 데이터베이스가 평소와 같이 제대로 작동되지 않거나 메모리 부족 오류가 발생할 수 있습니다.

다음 기능은 최상의 작동을 위해 더 높은 `MinCapacity` 및/또는 `MaxCapacity` 값을 구성해야 합니다.

- 성능 개선 도우미
- 글로벌 클러스터(기본 클러스터에만 적용됨 AWS 리전)
- 데이터 볼륨이 큰 클러스터에서 서버리스 인스턴스 생성

  여기에는 클러스터 복원의 일부인 서버리스 인스턴스 생성이 포함됩니다.

이 기능을 사용할 경우 적절한 용량 범위 설정에 대한 권장 사항은 DocumentDB Serverless 클러스터의 규모 조정 용량 범위 선택 섹션을 참조하세요. 잘못 구성된 용량 범위로 인해 데이터베이스에 메모리 부족 오류가 발생하는 경우 문제 해결 정보는 메모리 부족 오류 방지의 내용을 참조하세요.

Amazon DocumentDB Serverless를 사용하는 클러스터 생성

Amazon DocumentDB Serverless 클러스터 생성

Amazon DocumentDB Serverless를 사용하면 클러스터를 프로비저닝된 클러스터와 교환할 수 있습니다. 일부 인스턴스는 Serverless를 사용하고 일부 인스턴스는 프로비저닝되도록 클러스터를 구성할 수 있습니다.

원하는 리전 및 엔진 버전이 DocumentDB 서버리스를 지원하는지 확인합니다. DocumentDB Serverless 요구 사항 및 제한 사항 섹션을 참조하세요.

서버리스 인스턴스를 추가할 수 있는 Amazon DocumentDB 클러스터를 생성하려면 Amazon DocumentDB 클러스터 생성 섹션과 동일한 절차를 따릅니다. 유일한 차이점은 `ServerlessV2ScalingConfiguration` 인수도 제공해야 한다는 것입니다.

`ServerlessV2ScalingConfiguration` 인수는 DocumentDB Serverless 인스턴스의 규모 조정 용량 범위를 지정합니다. 클러스터의 모든 DocumentDB Serverless 인스턴스에 적용되는 최소 및 최대 DocumentDB 용량 단위(DCU) 값으로 구성됩니다.

- `MinCapacity` 값은 최소 규모 조정 용량을 지정합니다.
- `MaxCapacity` 값은 최대 규모 조정 용량을 지정합니다.

조정에 대한 자세한 내용은 Amazon DocumentDB Serverless 규모 조정 구성 섹션을 참조하세요.

Using the AWS Management Console

다음 AWS Management Console 구성 예제에서는 DocumentDB 서버리스 클러스터를 생성하는 방법을 보여줍니다.

1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

   클러스터 테이블이 표시됩니다.
3. 생성을 선택합니다.

   Amazon DocumentDB 클러스터 생성 페이지가 표시됩니다.
4. Amazon DocumentDB 클러스터 생성 페이지의 클러스터 유형 섹션에서 인스턴스 기반 클러스터(기본 옵션)를 선택합니다.
5. 클러스터 구성 섹션에서:

   a. 클러스터 식별자 에 **myserverlesscluster**와 같은 고유한 이름을 입력합니다. 콘솔은 모든 클러스터 이름을 입력 방식에 관계없이 소문자로 변경합니다.

b. 엔진 버전에는 5.0.0(기본 옵션)을 선택합니다.

6. 클러스터 스토리지 구성 섹션에서 Amazon DocumentDB 표준(기본 옵션입니다)을 선택합니다.

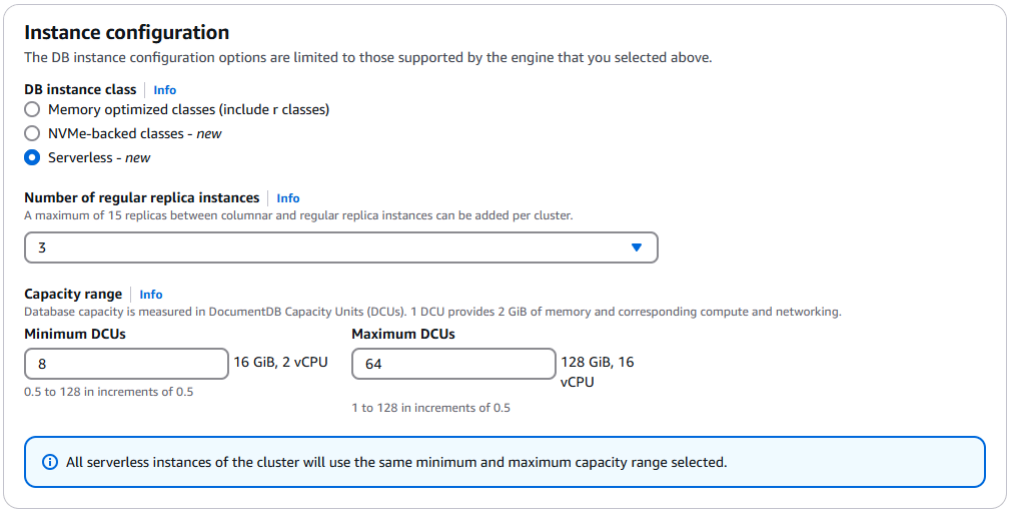
> **Note**
>
> 이 범주의 다른 옵션은 Amazon DocumentDB I/O 최적화입니다. 옵션에 대해 자세히 알아보려면 Amazon DocumentDB 클러스터 스토리지 구성 섹션을 참조하세요.

7. 인스턴스 구성 섹션에서:

   a. DB 인스턴스 클래스에서 서버리스를 선택합니다.

   b. 일반 복제본 인스턴스 수에서 3(기본 옵션)을 선택합니다.

   c. 용량 범위 섹션에서 최소 DCU 및 최대 DCU의 기본값을 그대로 둡니다. 이러한 파라미터의 설정에 대한 자세한 내용은 Amazon DocumentDB Serverless 인스턴스 제한 섹션을 참조하세요.



8. 연결 섹션에서 EC2 컴퓨팅 리소스에 연결 안 함을 기본 설정으로 둡니다.

9. 인증 섹션에 기본 사용자의 사용자 이름을 입력한 후 자체 관리형을 선택합니다. 암호를 입력한 다음 확인합니다.

   대신 에서 관리를 AWS Secrets Manager 선택한 경우 자세한 내용은 Amazon DocumentDB 및를 사용한 암호 관리 AWS Secrets Manager 섹션을 참조하세요.

10. 다른 모든 옵션을 기본값으로 두고 클러스터 생성을 선택합니다.

Using the AWS CLI

다음은 각각의 *### ## ## ###*를 자신의 정보 또는 구성 파라미터로 변경하는 예제입니다.

를 사용하여 DocumentDB 서버리스 인스턴스와 호환되는 클러스터를 생성하려면 섹션을 AWS CLI참조하세요를 사용하여 클러스터 생성 AWS CLI.

`create-db-cluster` 명령에 다음과 같은 추가 파라미터를 포함시킵니다.

```
--serverless-v2-scaling-configuration
     MinCapacity=minimum_capacity,MaxCapacity=maximum_capacity
```

예제:

```
aws docdb create-db-cluster \
      --db-cluster-identifier sample-cluster \
      --engine docdb \
      --engine-version 5.0.0 \
      --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16 \
      --master-username user-name \
      --master-user-password password
```

Amazon DocumentDB Serverless 인스턴스 추가

DocumentDB Serverless 인스턴스를 추가하려면 클러스터에 Amazon DocumentDB 인스턴스 추가 섹션과 동일한 절차에 따라 인스턴스 클래스로 db.serverless를 지정해야 합니다.

AWS Management Console을 사용하여 서버리스 인스턴스 추가

콘솔을 사용하여 DocumentDB Serverless 인스턴스를 추가하려면 클러스터에 Amazon DocumentDB 인스턴스 추가 섹션을 참조하여 AWS Management Console사용 탭을 선택합니다.

를 사용하여 서버리스 인스턴스 추가 AWS CLI

를 사용하여 DocumentDB 서버리스 인스턴스를 추가하려면 AWS CLI섹션을 참조클러스터에 Amazon DocumentDB 인스턴스 추가하고 사용 탭을 AWS CLI 선택합니다.

다음 인스턴스 클래스 CLI 인수를 사용합니다.

```
--db-instance-class db.serverless
```

예제:

```
aws docdb create-db-instance \
      --db-cluster-identifier sample-cluster \
      --db-instance-identifier sample-instance \
      --db-instance-class db.serverless \
      --engine docdb
```

Amazon DocumentDB Serverless로 마이그레이션

주제

- 기존 DocumentDB 클러스터를 서버리스로 마이그레이션
- MongoDB에서 DocumentDB Serverless로 마이그레이션

기존 DocumentDB 클러스터를 서버리스로 마이그레이션

클러스터의 엔진 버전 업그레이드

프로비저닝된 클러스터가 DocumentDB Serverless를 지원하지 않는 하위 엔진 버전을 실행하는 경우 먼저 지원되는 엔진 버전으로 클러스터를 업그레이드해야 합니다. 자세한 내용은 Amazon DocumentDB 인플레이스 주요 버전 업그레이드 단원을 참조하십시오.

프로비저닝된 클러스터를 DocumentDB Serverless로 마이그레이션

DocumentDB Serverless를 사용하도록 프로비저닝된 클러스터를 전환하려면 다음 단계를 따르세요.

1. DocumentDB Serverless와 함께 사용하기 위해 프로비저닝된 클러스터의 엔진 버전을 업그레이드해야 하는지 확인합니다. DocumentDB Serverless 요구 사항 및 제한 사항을(를) 참조하세요.

   > **Note**
   >
   > 프로비저닝된 클러스터가 DocumentDB Serverless에 사용할 수 없는 엔진 버전을 실행 중인 경우 클러스터의 엔진 버전을 업그레이드하세요. Amazon DocumentDB 인플레이스 주요 버전 업그레이드을(를) 참조하세요.

2. 클러스터에 대한 크기 조정 구성을 구성합니다. 규모 조정 구성의 선택에 대한 자세한 내용은 DocumentDB Serverless 클러스터의 규모 조정 용량 범위 선택 섹션을 참조하세요. 클러스터의 규

모 조정 구성을 수정하려면 클러스터의 규모 조정 용량 범위 구성 보기 및 수정 섹션을 참조하세요.

3. DocumentDB Serverless 요구 사항 및 제한 사항의 DocumentDB Serverless 요구 사항 및 제한 사항을 고려하도록 다른 클러스터 속성을 구성합니다.
4. 클러스터에 하나 이상의 DocumentDB Serverless 인스턴스를 추가합니다. Amazon DocumentDB Serverless 인스턴스 추가의 프로시저를 따르세요.

> **Note**
>
> 경우에 따라 클러스터에 이미 하나 이상의 프로비저닝된 리더 인스턴스가 있을 수 있습니다. 그렇다면 새 인스턴스를 생성하는 대신 리더 중 하나를 DocumentDB Serverless 인스턴스로 변환할 수 있습니다. 이 작업을 수행하려면 인스턴스의 클래스 변경의 프로시저를 따르세요.

5. (선택 사항) 장애 조치 작업을 수행하여 DocumentDB Serverless 인스턴스를 클러스터 라이터로 만듭니다. Amazon DocumentDB 장애 조치을(를) 참조하세요.
6. (선택 사항) 나머지 프로비저닝된 Amazon DocumentDB 인스턴스를 DocumentDB Serverless 인스턴스로 변환하거나(인스턴스의 클래스 변경 참조) 클러스터에서 제거합니다(Amazon DocumentDB 인스턴스 삭제 참조).

Using the AWS Management Console

다음 AWS Management Console 구성 예제는 Amazon DocumentDB 서버리스 사용을 시작하기 위해 엔진 버전 업그레이드가 필요하지 않은 Amazon DocumentDB 5.0.0을 실행하는 Amazon DocumentDB 프로비저닝 클러스터를 사용하는 마이그레이션 프로세스를 보여줍니다. 클러스터의 이름은 `sample`이고 `sample`, `sample2`, `sample3`라는 프로비저닝된 인스턴스 3개로 시작합니다. 이 예제에서는 이 세 인스턴스를 세 개의 서버리스 인스턴스로 대체합니다.

1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

클러스터 테이블이 표시됩니다.

3. 클러스터 테이블에서 서버리스 인스턴스를 추가할 클러스터의 확인란을 선택합니다.
4. 작업을 선택한 다음 인스턴스 추가를 선택합니다.
5. 인스턴스 추가 대화 상자에서 생성하려는 각각의 새 서버리스 인스턴스마다 DB 인스턴스 클래스 섹션에서 서버리스를 선택합니다.
6. 서버리스 용량 설정에서는 대화 상자의 용량 설명에 따라 규모 조정 구성을 설정합니다.
7. (선택 사항) 다른 인스턴스를 추가하려면 인스턴스 추가를 선택합니다. 원하는 새 인스턴스 수에 도달할 때까지 인스턴스를 계속 추가합니다.

   이 예제에서는 세 개의 새 서버리스 인스턴스가 생성됩니다.
8. 생성(Create)을 선택합니다.

   인스턴스를 생성하는 데 몇 분 정도 걸립니다. 콘솔 또는를 사용하여 인스턴스의 상태를 AWS CLI 볼 수 있습니다. 자세한 내용은 Amazon DocumentDB 클러스터 상태 모니터링 단원을 참조하십시오.
9. 클러스터 테이블로 돌아가서 원래 프로비저닝된 인스턴스 세 개 모두에 대해 확인란을 선택합니다.
10. 작업을 선택한 후 삭제를 선택합니다.

삭제하는 동안 장애 조치가 자동으로 수행되어 나머지 인스턴스 중 하나를 라이터로 승격합니다. 몇 분 후 삭제 프로세스가 완료됩니다. 이제 기존 클러스터에 DocumentDB Serverless 인스턴스 3개가 포함됩니다(크기 열에 정의됨).

Using the AWS CLI

다음 AWS CLI 구성 예제는 Amazon DocumentDB 서버리스 사용을 시작하기 위해 엔진 버전 업그레이드가 필요하지 않은 Amazon DocumentDB 5.0.0을 실행하는 Amazon DocumentDB 프로비저닝 클러스터를 사용하는 마이그레이션 프로세스를 보여줍니다. 클러스터의 이름은 `sample`이고 `sample`, `sample2`, `sample3`라는 프로비저닝된 인스턴스 3개로 시작합니다. 이 예제에서는 이 세 인스턴스를 세 개의 서버리스 인스턴스로 대체합니다. 클러스터의 이름은 `sample-cluster`이고, 라이터와 리더 인스턴스인 `sample-provisioned-instance-1`과 `sample-provisioned-instance-2`라는 두 개의 프로비저닝된 DB 인스턴스로 시작합니다.

다음은 각각의 `### ## ## ###`를 자신의 정보 또는 구성 파라미터로 변경하는 예제입니다.

`aws docdb describe-db-clusters` 작업을 사용하여 클러스터의 상태를 확인합니다. 다음 코드는 클러스터 `sample-cluster`의 상태를 찾고 결과를 테이블에 출력합니다.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterMembers' \
    --output table
```

```
----------------------------------------------------------------------------------------------------
|                                          DescribeDBClusters
                  |
+---------------------------------+---------------------------------
+------------------+----------------+
|  DBClusterParameterGroupStatus |      DBInstanceIdentifier       | IsClusterWriter
  | PromotionTier  |
+---------------------------------+---------------------------------
+------------------+----------------+
|  in-sync                        |  sample-provisioned-instance-2  |  False
  |  1             |
|  in-sync                        |  sample-provisioned-instance-1  |  True
  |  1             |
+---------------------------------+---------------------------------
+------------------+----------------+
```

클러스터에 대한 규모 조정 구성을 추가합니다.

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16
```

서버리스 인스턴스를 추가합니다. 이 예제에서는 `sample-serverless-instance-1` 및 `sample-serverless-instance-2`라는 새 서버리스 인스턴스를 추가합니다.

```
aws docdb create-db-instance \
    --db-cluster-identifier sample-cluster \
    --db-instance-identifier sample-serverless-instance-1 \
    --db-instance-class db.serverless \
    --engine docdb

aws docdb create-db-instance \
    --db-cluster-identifier sample-cluster \
```

```
    --db-instance-identifier sample-serverless-instance-2 \
    --db-instance-class db.serverless \
    --engine docdb
```

다음을 입력하여 서버리스 인스턴스를 사용할 수 있을 때까지 기다린 후 계속 진행합니다.

```
aws docdb wait db-instance-available \
    --db-instance-identifier sample-serverless-instance-1

aws docdb wait db-instance-available \
    --db-instance-identifier sample-serverless-instance-2
```

장애 조치 작업을 수행하여 새 `sample-serverless-instance-1` 인스턴스를 클러스터의 새 라이터로 만듭니다.

```
aws docdb failover-db-cluster \
    --db-cluster-identifier sample-cluster \
    --target-db-instance-identifier sample-serverless-instance-1
```

장애 조치를 완료하는 데 몇 초가 걸리며, 이후 sample-serverless-instance-1이 클러스터 라이터가 됩니다. 다음을 입력해서 이를 확인합니다.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterMembers' \
    --output table
```

```
-------------------------------------------------------------------------------------------
|                                        DescribeDBClusters
                  |
+---------------------------------+---------------------------------
+------------------+----------------+
|  DBClusterParameterGroupStatus |      DBInstanceIdentifier       | IsClusterWriter
  | PromotionTier  |
+---------------------------------+---------------------------------
+------------------+----------------+
|  in-sync                       |  sample-provisioned-instance-2  |  False
  |  1             |
|  in-sync                       |  sample-provisioned-instance-1  |  False
  |  1             |
```

```
|  in-sync                      |  sample-serverless-instance-2   |  False
  |  1             |
|  in-sync                      |  sample-serverless-instance-1   |  True
  |  1             |
+-------------------------------+---------------------------------
+------------------+----------------+
```

마지막으로 원래 프로비저닝된 인스턴스를 삭제합니다.

```
aws docdb delete-db-instance \
    --db-instance-identifier sample-provisioned-instance-1

aws docdb delete-db-instance \
    --db-instance-identifier sample-provisioned-instance-2
```

MongoDB에서 DocumentDB Serverless로 마이그레이션

프로비저닝된 Amazon DocumentDB와 마찬가지로 MongoDB 데이터베이스를 DocumentDB Serverless로 마이그레이션할 수 있습니다. 자세한 내용은 Amazon DocumentDB 마이그레이션 및 업그레이드 단원을 참조하십시오.

Amazon DocumentDB Serverless 관리

클러스터의 규모 조정 용량 범위 구성 보기 및 수정

`ServerlessV2ScalingConfiguration` 인수는 DocumentDB Serverless 인스턴스의 규모 조정 용량 범위를 지정합니다. 클러스터의 모든 DocumentDB Serverless 인스턴스에 적용되는 최소 및 최대 DocumentDB 용량 단위(DCU) 값으로 구성됩니다.

- **`MinCapacity`** - 클러스터 내에서 DocumentDB Serverless 인스턴스의 최소 규모 조정 용량입니다.
- **`MaxCapacity`** - 클러스터 내에서 DocumentDB Serverless 인스턴스의 최대 규모 조정 용량입니다.

> **Note**
>
> 다음 규모 조정 구성 수정 사항은 새 `MinCapacity` 및 `MaxCapacity` 값을 반영하기 위해 인스턴스를 다시 시작해야 합니다.
>
> - `MaxCapacity` 값에 대한 모든 변경

- `MinCapacity` 값을 더 높은 값에서 1.0 이하로 변경
- `MinCapacity` 값을 더 낮은 값에서 1.0보다 크게 변경

규모 조정 구성 및 적절한 규모 조정 용량 제한을 선택하는 방법에 대한 자세한 내용은 Amazon DocumentDB Serverless 규모 조정 구성 섹션을 참조하세요.

Using the AWS Management Console

다음 AWS Management Console 구성 예제에서는 DocumentDB 서버리스 클러스터의 조정 구성 설정을 보고 편집하는 방법을 보여줍니다.

1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

   > **Tip**
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

   클러스터 테이블이 표시됩니다.
3. 클러스터 테이블에서 규모 조정 용량을 수정하려는 클러스터의 확인란을 선택합니다.
4. 작업을 선택한 후 수정을 선택합니다.

   클러스터 수정 대화 상자가 표시 됩니다.
5. 서버리스 용량 설정 섹션을 찾은 다음 대화 상자의 용량 설명에 따라 규모 조정 구성(용량 범위)을 설정합니다.

   규모 조정 및 용량 범위에 대한 자세한 내용은 Amazon DocumentDB Serverless 규모 조정 구성 섹션을 참조하세요.
6. Continue(계속)를 선택합니다.
7. 수정 사항 예약에 대해 즉시 적용을 선택합니다.
8. 클러스터 수정을 선택합니다.
9. 수정이 완료되면 각 서버리스 인스턴스를 재부팅해야 합니다. 라이터 사용 불가를 최소화하려면 다음 일련의 작업을 수행합니다.

a. 각 서버리스 리더 인스턴스를 재부팅합니다.

   i. 리더 인스턴스를 선택하고 작업을 선택한 다음 재부팅을 선택합니다.
   ii. 인스턴스 상태가 사용 가능으로 돌아갈 때까지 기다립니다.

b. 재부팅된 서버리스 인스턴스에 장애 조치를 수행합니다.

   i. 클러스터를 선택하고 작업을 선택한 다음 장애 조치를 선택합니다.
   ii. 장애 조치 작업이 완료될 때까지 기다립니다.

c. 나머지 서버리스 인스턴스를 재부팅합니다.

   i. 나머지 인스턴스를 선택하고 작업을 선택한 다음 재부팅을 선택합니다.
   ii. 인스턴스 상태가 사용 가능으로 돌아갈 때까지 기다립니다.

Using the AWS CLI

다음 AWS CLI 구성 예제는 현재 조정 구성을 보여줍니다.

다음은 각각의 *### ## ## ###*를 자신의 정보 또는 구성 파라미터로 변경하는 예제입니다.

다음 `describe-db-clusters` AWS CLI 명령을 사용하여 클러스터의 현재 조정 구성을 볼 수 있습니다.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[0].ServerlessV2ScalingConfiguration'
```

다음은 이 명령의 출력입니다.

```
{
    "MinCapacity": 0.5,
    "MaxCapacity": 16.0
}
```

다음과 같이 `modify-db-cluster` 명령을 사용하여 클러스터의 규모 조정 구성을 수정할 수 있습니다.

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
```

```
    --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=32
```

완료되면 각 서버리스 인스턴스를 재부팅해야 합니다. 라이터 사용 불가를 최소화하기 위해 다음 일련의 작업을 수행할 수 있습니다.

```
aws docdb reboot-db-instance \
    --db-instance-identifier sample-serverless-instance-reader

aws docdb wait db-instance-available \
    --db-instance-identifier sample-serverless-instance-reader

aws docdb failover-db-cluster \
   --db-cluster-identifier sample-cluster \
   --target-db-instance-identifier sample-serverless-instance-reader

aws docdb reboot-db-instance \
    --db-instance-identifier sample-serverless-instance-writer

aws docdb wait db-instance-available \
    --db-instance-identifier sample-serverless-instance-writer

aws docdb failover-db-cluster \
   --db-cluster-identifier sample-cluster \
   --target-db-instance-identifier sample-serverless-instance-writer
```

서버리스 리더의 승격 계층 보기 및 수정

여러 개의 DocumentDB Serverless 인스턴스를 포함하는 클러스터 또는 프로비저닝된 인스턴스와 서버리스 인스턴스가 혼합된 클러스터의 경우 각 서버리스 인스턴스의 승격 티어 설정을 유의해야 합니다. 이 설정은 프로비저닝된 인스턴스보다 서버리스 인스턴스에 대해 더 많은 동작을 제어합니다.

프로비저닝된 인스턴스의 경우 티어 0~15를 선택하면 Amazon DocumentDB가 장애 조치 작업 중에 라이터로 승격할 리더 인스턴스를 선택하는 순서만 결정됩니다. 그러나 서버리스 인스턴스의 경우 티어 번호에 따라 인스턴스를 라이터 인스턴스의 용량에 맞춰 스케일 업할지 아니면 자체 워크로드를 기준으로 독립적으로 규모 조정을 할지도 결정됩니다. 티어 0 또는 1의 Serverless 리더 인스턴스는 최소한 라이터 인스턴스만큼 높은 최소 용량으로 유지됩니다. 따라서 장애 조치가 발생할 경우 라이터 인스턴스에서 인계할 준비가 됩니다. 라이터 인스턴스가 프로비저닝된 인스턴스인 경우 Amazon DocumentDB는 동등한 DocumentDB Serverless 용량을 추정합니다. 이 추정치를 서버리스 리더 인스턴스의 최소 용량으로 사용합니다.

티어 2~15의 DocumentDB Serverless 리더 인스턴스는 최소 용량에 대해 이러한 제약 조건을 가지고 있지 않습니다. 유휴 상태일 때 클러스터의 용량 범위에 지정된 최소 DocumentDB 용량 단위(DCU) 값으로 스케일 다운할 수 있습니다.

Using the AWS Management Console

다음 AWS Management Console 구성 예제에서는 DocumentDB 서버리스 인스턴스 리더의 프로모션 계층 설정을 보고 수정하는 방법을 보여줍니다.

1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.

   각 인스턴스의 승격 티어는 AWS Management Console의 승격 티어 열에 표시됩니다.
2. 탐색 창에서 클러스터를 선택합니다.

   > **Tip**
   > 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

   클러스터 테이블이 표시됩니다.
3. 클러스터 테이블에서 승격 티어를 수정하려는 인스턴스의 확인란을 선택합니다.
4. 작업을 선택한 후 수정을 선택합니다.

   인스턴스 수정 대화 상자가 표시됩니다.
5. 장애 조치 섹션을 찾아서 승격 티어를 원하는 수준으로 설정합니다.
6. Continue(계속)를 선택합니다.
7. 수정 사항 예약에 대해 즉시 적용을 선택합니다.
8. Modify Instance(인스턴스 수정)를 선택합니다.

Using the AWS CLI

다음 AWS CLI 구성 예제는 현재 조정 구성을 보여줍니다.

다음은 각각의 *### ## ## ###*를 자신의 정보 또는 구성 파라미터로 변경하는 예제입니다.

다음 `describe-db-clusters` AWS CLI 명령을 사용하여 클러스터에 있는 모든 인스턴스의 승격 티어를 볼 수 있습니다.

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[0].DBClusterMembers' \
    --output table
```

다음은 이 명령의 출력입니다.

```
-------------------------------------------------------------------------------------------------
|                                          DescribeDBClusters
                  |
+---------------------------------+---------------------------------
+------------------+----------------+
|  DBClusterParameterGroupStatus |      DBInstanceIdentifier       | IsClusterWriter
  | PromotionTier  |
+---------------------------------+---------------------------------
+------------------+----------------+
|  in-sync                       |  sample-serverless-instance-2   |  False
  |  1             |
|  in-sync                       |  sample-serverless-instance-1   |  True
  |  1             |
+---------------------------------+---------------------------------
+------------------+----------------+
```

특정 인스턴스의 승격 티어는 다음과 같이 `modify-db-instance` 명령을 사용하여 수정할 수 있습니다.

```
aws docdb modify-db-instance \
    --db-instance-identifier sample-serverless-instance-2 \
    --promotion-tier 3
```

Amazon DocumentDB Serverless 인스턴스 제한

DocumentDB Serverless 인스턴스의 경우 다음 인스턴스당 제한은 인스턴스의 현재 규모 조정 용량에 따라 달라집니다.

- 인스턴스 메모리(GiB)
- 연결(모두)

- 커서 제한
- 트랜잭션 열기
- 연결(활성)

다음 표에서는 인스턴스의 현재 규모 조정 용량과 관련하여 서버리스 인스턴스의 인스턴스당 제한이 어떻게 조정되는지 설명합니다.

제한 값은 클러스터의 `MinCapacity` 규모 조정 구성이 2보다 크게 설정되었는지 여부에 따라 다릅니다. `MinCapacity`가 2 이하로 설정된 경우 커서 제한, 미결 트랜잭션, 연결(활성)에 대한 제한 값은 더 낮은 최대값으로 제한됩니다. (이 한도의 기술적 이유는 '유휴' 상태로의 스케일 다운을 지원하기 위한 것입니다. 자세한 내용은 유휴 상태(0.5DCU) 단원을 참조하십시오.

MinCapacity가 1 이하인 경우 db.serverless 인스턴스 제한

현재 용량(DCU)	인스턴스 메모리(GiB)	연결(모두)	커서 제한	트랜잭션 열기	연결(활성)
0.5	1	250	6	12	39
1	2	500	12	24	79
2	4	1000	24	48	173
4	8	2000	48	96	416
8	16	4000	96	192	1071
16	32	8000	132	264	1550
32	64	16000	132	264	1550
64	128	32000	132	264	1550
128	256	60000	132	264	1550
256	512	60000	132	264	1550

MinCapacity가 1보다 클 경우 db.serverless 인스턴스 제한

현재 용량(DCU)	인스턴스 메모리(GiB)	연결(모두)	커서 제한	트랜잭션 열기	연결(활성)
1.5	3	750	18	36	124
2	4	1000	24	48	173
4	8	2000	48	96	416
8	16	4000	96	192	1071
16	32	8000	192	384	2709
32	64	16000	384	768	4500
64	128	32000	768	1536	4500
128	256	60000	1536	3072	4500
256	512	60000	3072	6144	4500

다음 CloudWatch 지표를 사용하여 인스턴스당 제한을 모니터링하고 경보를 생성할 수 있습니다. Amazon DocumentDB CloudWatch 지표에 관한 더 많은 내용은 CloudWatch를 사용하여 Amazon DocumentDB 모니터링를 참조하세요.

Resource	CloudWatch 제한 지표	CloudWatch 사용량 지표(최대 1분)	CloudWatch 사용량 지표
인스턴스 메모리	-	-	FreeableMemory
연결(모두)	DatabaseConnectionsLimit	DatabaseConnectionsMax	DatabaseConnections
커서	DatabaseCursorsLimit	DatabaseCursorsMax	DatabaseCursors
트랜잭션	TransactionsOpenLimit	TransactionsOpenMax	TransactionsOpen

Amazon DocumentDB Serverless 규모 조정 구성

주제

- DocumentDB Serverless 클러스터의 규모 조정 용량 범위 선택
- DocumentDB Serverless 클러스터에 대한 MinCapacity 설정 선택
- DocumentDB Serverless 클러스터에 대한 MaxCapacity 설정 선택
- 메모리 부족 오류 방지
- 서버리스 인스턴스가 스케일 다운되지 않는 이유는 무엇인가요?

DocumentDB Serverless 클러스터의 규모 조정 용량 범위 선택

Amazon DocumentDB 클러스터에 DocumentDB Serverless 인스턴스를 추가하려면 이에 앞서 클러스터에 `ServerlessV2ScalingConfiguration` 파라미터 세트도 있어야 합니다.

`ServerlessV2ScalingConfiguration` 파라미터는 클러스터 내 모든 서버리스 인스턴스의 서버리스 규모 조정 용량 범위를 정의하는 두 가지 값으로 구성됩니다.

- **`MinCapacity`** - 클러스터 내에서 DocumentDB Serverless 인스턴스의 최소 규모 조정 용량입니다.
- **`MaxCapacity`** - 클러스터 내에서 DocumentDB Serverless 인스턴스의 최대 규모 조정 용량입니다.

DocumentDB Serverless 클러스터에 대한 **`MinCapacity`** 설정 선택

항상 `MinCapacity`에 0.5를 선택하는 것이 매력적으로 보입니다. 이 값을 사용하면 인스턴스가 완전히 유휴 상태일 때 활성 상태로 유지하면서도 최소 용량으로 스케일 다운할 수 있기 때문입니다. 그러나 해당 클러스터를 사용하는 방법과 구성하는 기타 설정에 따라 가장 효과적인 최소 용량은 다를 수도 있습니다. 최소 용량 설정을 선택할 때는 다음 요소를 고려하세요.

- DocumentDB Serverless 인스턴스의 규모 조정 비율은 현재 용량에 따라 다릅니다. 현재 용량이 높을수록 더 빠르게 확장할 수 있습니다. 인스턴스를 매우 높은 용량으로 빠르게 스케일 업해야 하는 경우 규모 조정 속도가 요구 사항을 충족하는 값으로 최소 용량을 설정하는 것이 좋습니다.
- 일반적으로 특히 높거나 낮은 워크로드를 예상하여 인스턴스의 인스턴스 클래스를 수정하는 경우 해당 경험을 사용하여 동등한 DocumentDB Serverless 용량 범위를 대략적으로 추정할 수 있습니다. 프로비저닝된 Amazon DocumentDB 인스턴스 유형의 메모리 크기를 확인하려면 인스턴스 할당량 섹션을 참조하세요.

예를 들어 클러스터의 워크로드가 낮은 경우 db.r6g.xlarge 인스턴스 클래스를 사용한다고 가정해보겠습니다. 해당 인스턴스 클래스에는 32GiB의 메모리가 있습니다. 따라서 MinCapacity를 16으로 지정하여 거의 동일한 용량으로 스케일 다운할 수 있는 서버리스 인스턴스를 설정할 수 있습니다. 이는 각 DCU가 약 2GiB의 메모리에 해당하기 때문입니다. db.r6g.xlarge 인스턴스가 때때로 충분히 활용되지 않는 경우에 대비하여 인스턴스가 더 스케일 다운되도록 약간 더 낮은 값을 지정할 수 있습니다.

- 인스턴스의 버퍼 캐시에 일정량의 데이터가 있을 때 애플리케이션이 가장 효율적으로 작동하는 경우 메모리가 자주 액세스하는 데이터를 저장할 수 있을 만큼 충분히 큰 최소 DCU 설정을 지정하는 것이 좋습니다. 그렇지 않으면 서버리스 인스턴스가 더 낮은 메모리 크기로 스케일 다운될 때 버퍼 캐시에서 일부 데이터가 제거됩니다. 그런 다음 인스턴스가 다시 스케일 업되면 시간이 지남에 따라 정보가 버퍼 캐시로 다시 읽혀집니다. 데이터를 버퍼 캐시로 다시 가져오기 위한 I/O 양이 많은 경우 최소 DCU 값을 높이는 것이 더 효과적일 수 있습니다. 자세한 내용은 인스턴스 크기 조정 단원을 참조하십시오.
- DocumentDB 인스턴스가 대부분 특정 용량에서 실행되는 경우 해당 기준보다 낮으면서도 너무 모자라지는 않는 최소 용량 설정을 지정하는 것이 좋습니다. 서버리스 인스턴스는 현재 용량이 필요한 용량보다 크게 낮지 않은 경우 스케일 업할 양과 속도를 가장 효과적으로 추정할 수 있습니다.
- 프로비저닝된 워크로드에 T3 또는 T4g와 같은 소규모 인스턴스 클래스에 비해 너무 높은 메모리 요구 사항이 있는 경우 R5 또는 R6g 인스턴스에 필적하는 메모리를 제공하는 최소 DCU 설정을 선택합니다.
- 지정된 기능에 MinCapacity 대해 다음 최소값을 사용합니다(변경될 수 있음).
  - 성능 개선 도우미- 2 DCU
  - 글로벌 클러스터 - DCUs(기본 클러스터에만 적용됨 AWS 리전)
- Amazon DocumentDB에서 복제는 스토리지 계층에서 발생하므로 리더 용량은 복제에 직접적인 영향을 미치지 않습니다. 그러나 독립적으로 규모가 조정되는 DocumentDB Serverless 리더 인스턴스의 경우, 쿼리 지연 시간을 방지하기 위해 쓰기 집약적인 기간 동안 워크로드를 처리하기에 최소 용량이 충분한지 확인합니다. 승격 티어 2~15의 리더 인스턴스에 성능 문제가 발생하는 경우 클러스터의 최소 용량을 늘리는 것이 좋습니다. 리더 인스턴스 규모가 라이터와 함께 또는 독자적으로 조정되는지를 변경하는 방법에 대한 자세한 내용은 서버리스 리더의 승격 계층 보기 및 수정 섹션을 참조하세요.

  DocumentDB Serverless 리더 인스턴스가 있는 클러스터가 있는 경우, 리더의 승격 티어가 0 또는 1이 아니면 리더는 라이터 인스턴스와 함께 조정되지 않습니다. 이 경우 최소 용량을 낮게 설정하면 과도한 복제 지연이 발생할 수 있습니다. 데이터베이스가 사용 중일 때 리더의 용량이 라이터의 변경 사항을 적용하기에 충분하지 않을 수 있기 때문입니다. 최소 용량을 라이터 인스턴스와 비슷한 양의 메모리 및 CPU를 나타내는 값으로 설정합니다.

- DocumentDB Serverless 인스턴스가 최소 용량에서 최대 용량으로 조정되는 데 걸리는 시간은 최소 DCU 값과 최대 DCU 값의 차이에 따라 다릅니다. 인스턴스의 현재 용량이 클 때 DocumentDB Serverless는 인스턴스가 작은 용량에서 시작할 때보다 더 큰 증분으로 스케일 업합니다. 따라서 상대적으로 큰 최대 용량을 지정하고 인스턴스가 해당 용량 근처에서 대부분의 시간을 보내는 경우 최소 DCU 설정을 높이는 것이 좋습니다. 이렇게 하면 유휴 인스턴스가 더 빠르게 최대 용량까지 다시 스케일 업할 수 있습니다.
- 연결 제한, 커서 제한, 미결 트랜잭션 제한과 같은 특정 인스턴스 제한은 서버리스 인스턴스의 현재 용량에 따라 결정됩니다. 인스턴스의 현재 용량이 작으면 그에 따라 제한도 작아집니다. 서버리스 인스턴스를 `MinCapacity` 값으로 스케일 다운할 때 이러한 제한이 문제가 되는 경우 더 높은 값으로 `MinCapacity`를 늘리는 것이 좋습니다. 자세한 내용은 Amazon DocumentDB Serverless 인스턴스 제한 단원을 참조하십시오.
- 또한 `MinCapacity`가 1.0DCU 이하로 설정되면 활성 연결 제한, 커서 제한, 미결 트랜잭션 제한과 같은 특정 인스턴스 제한이 더 낮은 최대값으로 한정됩니다. 이러한 제한 한도가 워크로드에 충분하지 않은 경우 1.5DCUs 이상의 `MinCapacity` 값을 사용합니다. 자세한 내용은 Amazon DocumentDB Serverless 인스턴스 제한 단원을 참조하십시오.

클러스터의 규모 조정 구성을 수정하는 방법에 대한 지침은 Amazon DocumentDB Serverless 관리 섹션을 참조하세요.

DocumentDB Serverless 클러스터에 대한 **`MaxCapacity`** 설정 선택

최대 DocumentDB Serverless 용량 설정에 대해 항상 높은 값을 선택하는 것이 좋습니다. 최대 용량이 크면 인스턴스가 집약적인 워크로드를 실행할 때 가장 많이 스케일 업할 수 있습니다. 낮은 값은 예기치 않은 요금의 가능성을 방지합니다. 해당 클러스터를 사용하는 방법과 구성하는 기타 설정에 따라 가장 효과적인 값은 원래 생각했던 것보다 높거나 낮을 수 있습니다. 최대 용량 설정을 선택할 때는 다음 요소를 고려하세요.

- 최대 용량은 최소 용량보다는 커야 합니다. 최대 및 최소 용량은 동일하게 설정할 수 있습니다. 그러나 이 경우 용량은 절대 확장하거나 축소되지 않습니다. 따라서 최소 및 최대 용량에 대해 동일한 값을 사용하는 것은 테스트 상황 외에는 적절하지 않습니다.
- 최대 용량은 최소 1.0DCU여야 하며 최대 256DCU여야 합니다.
- 서버리스 인스턴스의 규모 조정 및 리소스 사용량을 모니터링합니다. 서버리스 인스턴스가 최대 용량으로 자주 확장되고 리소스 제약 조건에 도달하는 경우(예: 지표가 `DCUUtilization` 100.0인 경우) 더 높은 `MaxCapacity` 값을 선택합니다. 자세한 내용은 Amazon DocumentDB Serverless 모니터링 단원을 참조하십시오.

- 일반적으로 특히 높거나 낮은 워크로드를 예상하여 프로비저닝된 인스턴스의 인스턴스 클래스를 수정하는 경우 해당 경험을 사용하여 동등한 DocumentDB Serverless 용량 범위를 추정할 수 있습니다. 프로비저닝된 Amazon DocumentDB 인스턴스의 메모리 크기를 확인하려면 인스턴스 할당량 섹션을 참조하세요.

  예를 들어 클러스터의 워크로드가 높은 경우 `db.r6g.4xlarge` 인스턴스 클래스를 사용한다고 가정해보겠습니다. 해당 인스턴스 클래스에는 128GiB의 메모리가 있습니다. 따라서 최대 DCU 설정을 64로 지정하여 거의 동일한 용량으로 스케일 업할 수 있는 서버리스 인스턴스를 설정할 수 있습니다. 이는 각 DCU가 약 2GiB의 메모리에 해당하기 때문입니다. `db.r6g.4xlarge` 인스턴스에 워크로드를 효과적으로 처리할 수 있는 용량이 충분하지 않은 경우에 대비하여 인스턴스를 더 스케일 업할 수 있도록 다소 높은 값을 지정할 수 있습니다.
- 데이터베이스 사용량에 대한 예산 상한이 있는 경우 모든 서버리스 인스턴스가 항상 최대 용량으로 실행되더라도 해당 상한 내에서 유지되는 값을 선택하세요. 클러스터에 n개의 서버리스 인스턴스가 있는 경우 클러스터가 언제든지 사용할 수 있는 이론상 최대 서버리스 용량은 n에 클러스터의 최대 DCU 설정을 곱한 것입니다. (예를 들어 일부 리더가 라이터와 독자적으로 확장하는 경우 실제 소비량은 더 적을 수 있습니다.)
- 서버리스 리더 인스턴스를 사용하여 라이터 인스턴스에서 읽기 전용 워크로드의 일부를 오프로드하는 경우 더 낮은 최대 용량 설정을 선택할 수 있습니다. 클러스터에 단일 인스턴스만 포함되어 있는 것처럼 각 리더 인스턴스를 높게 조정할 필요가 없음을 반영하기 위해 이 작업을 수행합니다.
- 애플리케이션에서 잘못 구성된 데이터베이스 파라미터 또는 비효율적인 쿼리로 인한 과도한 사용을 방지하는 경우를 가정해 보겠습니다. 이 경우 설정할 수 있는 절대 최대값보다 낮은 최대 용량 설정을 선택하여 우발적인 남용을 방지할 수 있습니다.
- 실제 사용자 활동으로 인한 스파이크가 드물지만 발생하는 경우 최대 용량 설정을 선택할 때 이러한 경우를 고려할 수 있습니다. 애플리케이션이 전체 성능과 확장성을 유지하여 계속 실행하는 것이 우선 순위인 경우 정상 사용에서 관찰되는 것보다 더 높은 최대 용량 설정을 지정할 수 있습니다. 활동이 극도로 급증하는 동안 애플리케이션이 감소된 처리량으로 실행되는 것이 괜찮다면 약간 더 낮은 최대 용량 설정을 선택할 수 있습니다. 애플리케이션을 계속 실행하기에 충분한 메모리와 CPU 리소스가 있는 설정을 선택했는지 확인하세요.
- 클러스터에서 각 인스턴스의 메모리 사용량을 늘리는 설정을 켜면 최대 DCU 값을 결정할 때 해당 메모리를 고려합니다. 이러한 설정에는 성능 개선 도우미 및 글로벌 클러스터에 대한 설정이 포함됩니다. 최대 DCU 값이 서버리스 인스턴스가 해당 기능이 사용 중일 때 워크로드를 처리할 수 있을 만큼 충분히 스케일 업할 수 있도록 허용하는지 확인하세요. 낮은 최대 DCU 설정과 메모리 오버헤드를 부과하는 Amazon DocumentDB 기능의 조합으로 인해 발생하는 문제 해결에 대한 정보는 아래의 메모리 부족 오류 방지 섹션을 참조하세요.
- 지정된 기능에 `MaxCapacity` 대해 다음 최소값을 사용합니다(변경될 수 있음).

  - 큰 데이터 볼륨이 있는 클러스터에서 서버리스 인스턴스 생성 - 2DCU(클러스터 복원의 일부인 서버리스 인스턴스 생성 포함)
- 연결 제한, 커서 제한, 미결 트랜잭션 제한과 같은 특정 인스턴스 제한은 인스턴스의 현재 용량에 따라 결정됩니다. 워크로드 `MaxCapacity` 값을 선택할 때는 이러한 제한 중 하나에 병목 현상이 발생하지 않도록 이러한 인스턴스 제한을 염두에 두어야 합니다. 자세한 내용은 Amazon DocumentDB Serverless 인스턴스 제한 단원을 참조하십시오.

클러스터의 규모 조정 구성을 수정하는 방법에 대한 지침은 Amazon DocumentDB Serverless 관리 섹션을 참조하세요.

메모리 부족 오류 방지

DocumentDB Serverless 인스턴스 중 하나가 지속적으로 최대 용량 한도에 도달하는 경우 Amazon DocumentDB는 인스턴스를 incompatible-parameters 상태로 설정하여 이 조건을 나타냅니다. 인스턴스가 incompatible-parameters 상태인 동안에는 일부 작업이 차단됩니다. 예를 들어 엔진 버전을 업그레이드할 수 없습니다. Amazon DocumentDB 인스턴스의 상태에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB 인스턴스 상태 모니터링.

일반적으로 메모리 부족 오류로 인해 인스턴스가 자주 다시 시작되면 이 상태가 됩니다. Amazon DocumentDB는 이러한 유형의 재시작이 발생할 때 이벤트를 기록합니다. 리소스 이벤트를 보려면 Amazon DocumentDB 이벤트 보기 섹션을 참조하세요. 성능 개선 도우미와 같은 설정을 켜서 발생하는 오버헤드로 인해 비정상적으로 높은 메모리 사용량이 발생할 수 있습니다. 또한 인스턴스의 과중한 워크로드 또는 많은 수의 스키마 객체와 연결된 메타데이터 관리로 인해 발생할 수 있습니다.

메모리 압력이 낮아져 인스턴스가 최대 용량에 자주 도달하지 않는 경우 Amazon DocumentDB는 자동으로 인스턴스 상태를 다시 사용 가능 상태로 변경합니다.

이 조건에서 복구하려면 다음 작업 중 일부 또는 전부를 수행할 수 있습니다.

- 클러스터의 최소 DocumentDB 용량 단위(DCU) 값을 변경하여 서버리스 인스턴스의 용량 하한을 늘립니다. 이렇게 하면 유휴 데이터베이스가 클러스터에서 켜진 기능에 필요한 것보다 적은 메모리를 가진 용량으로 축소되는 문제를 피할 수 있습니다. 클러스터의 DCU 설정을 변경한 후 서버리스 인스턴스를 재부팅합니다. 이렇게 하면 Amazon DocumentDB가 상태를 다시 사용 가능으로 재설정할 수 있는지 여부를 평가합니다.
- 클러스터의 최대 DCU 값을 변경하여 서버리스 인스턴스의 용량 상한을 늘립니다. 이렇게 하면 사용량이 많은 데이터베이스가 클러스터 및 데이터베이스 워크로드에서 켜진 기능을 위한 충분한 메모리로 용량을 확장할 수 없는 문제를 피할 수 있습니다. 클러스터의 DCU 설정을 변경한 후 서버리스

인스턴스를 재부팅합니다. 이렇게 하면 Amazon DocumentDB가 상태를 다시 사용 가능으로 재설정할 수 있는지 여부를 평가합니다.

- 메모리 오버헤드가 필요한 구성 설정을 끕니다. 예를 들어 성능 개선 도우미와 같은 기능이 켜져 있지만 사용하지 않는다고 가정해보겠습니다. 이 경우 해당 기능을 끌 수 있습니다. 또는 클러스터의 최소 및 최대 용량 값을 더 높게 조정하여 그러한 유형의 기능에서 사용하는 메모리를 고려할 수 있습니다. 최소 및 최대 용량 설정 선택에 대한 지침은 DocumentDB Serverless 클러스터의 규모 조정 용량 범위 선택 페이지를 참조하세요.
- 인스턴스의 워크로드를 줄입니다. 예를 들어 클러스터에 리더 인스턴스를 추가하여 읽기 전용 쿼리의 로드를 더 많은 인스턴스에 분산할 수 있습니다.

서버리스 인스턴스가 스케일 다운되지 않는 이유는 무엇인가요?

데이터베이스에 로드가 없는데도 DocumentDB serverless가 최소값으로 스케일 다운되지 않는 경우가 있습니다. 원인은 다음과 같습니다.

- 성능 개선 도우미는 리소스 사용량을 늘리고, 데이터베이스가 최소 용량으로 스케일 다운되는 것을 방지할 수 있습니다. 이러한 특성은 다음과 같습니다.
- 리더 인스턴스가 최소 용량으로 스케일 다운되지 않고 라이터 인스턴스와 같거나 더 높은 용량을 유지하는 경우, 리더 인스턴스의 우선 순위 등급을 확인합니다. 티어 0 또는 1의 DocumentDB Serverless 리더 인스턴스는 최소한 라이터 인스턴스만큼 높은 최소 용량으로 유지됩니다. 리더의 우선 순위 등급을 2 이상으로 변경하여 라이터와 별개로 크기를 늘리거나 줄이십시오. 자세한 내용은 Amazon DocumentDB Serverless 규모 조정 단원을 참조하십시오.
- 데이터베이스 워크로드가 많으면 리소스 사용량이 증가할 수 있습니다.
- 데이터베이스 볼륨이 크면 리소스 사용량이 증가할 수 있습니다. Amazon DocumentDB는 클러스터 관리에 메모리와 CPU 리소스를 사용합니다. 데이터베이스 볼륨이 큰 클러스터를 관리하려면 Amazon DocumentDB에 더 많은 CPU와 메모리가 필요합니다. 클러스터의 최소 용량이 클러스터 관리에 필요한 최소 용량보다 작으면 클러스터가 최소 용량으로 스케일 다운되지 않습니다.
- 백그라운드 유지 관리 활동은 주기적으로 리소스 사용량을 늘릴 수 있습니다.

데이터베이스가 여전히 구성된 최소 용량으로 축소되지 않는 경우 데이터베이스를 중지했다가 다시 시작하여 시간이 지남에 따라 누적되었을 수 있는 메모리 조각을 회수합니다.

> **Note**
>
> 데이터베이스를 중지하고 시작하면 가동 중지가 발생합니다. 필요한 경우에만 이 작업을 수행합니다.

Amazon DocumentDB Serverless 모니터링

Amazon DocumentDB 모니터링에 대해 자세히 알아보려면 Amazon DocumentDB에서 모니터링 및 로깅 섹션을 참조하세요.

주제

- 메모리 부족: 호환되지 않는 파라미터 상태
- DocumentDB Serverless에 대한 Amazon CloudWatch 지표
- 성능 개선 도우미로 DocumentDB Serverless 성능 모니터링

메모리 부족: 호환되지 않는 파라미터 상태

Serverless 인스턴스 중 하나가 지속적으로 최대 용량 한도에 도달하는 경우 Amazon DocumentDB는 인스턴스를 incompatible-parameters 상태로 설정하여 이 조건을 나타냅니다. 자세한 내용은 메모리 부족 오류 방지 단원을 참조하십시오.

DocumentDB Serverless에 대한 Amazon CloudWatch 지표

Amazon DocumentDB에 CloudWatch를 사용하는 방법에 대해 자세히 알아보려면 CloudWatch를 사용하여 Amazon DocumentDB 모니터링 섹션을 참조하세요.

CloudWatch에서 서버리스 인스턴스를 보고 `ServerlessDatabaseCapacity` 지표를 통해 각 인스턴스에서 소비한 용량을 모니터링할 수 있습니다. `DatabaseConnections` 및 Queries와 같은 모든 표준 DocumentDB CloudWatch 지표를 모니터링할 수도 있습니다. Amazon DocumentDB에 대해 모니터링할 수 있는 CloudWatch 지표의 전체 목록은 CloudWatch를 사용하여 Amazon DocumentDB 모니터링 섹션을 참조하세요. 다음 CloudWatch 인스턴스 수준 지표는 DocumentDB Serverless 인스턴스가 스케일 업 및 스케일 다운되는 방식을 이해하기 위해 모니터링하는 데 중요합니다. 이러한 모든 지표는 매초마다 계산됩니다. 그렇게 하면 서버리스 인스턴스의 현재 상태를 모니터링할 수 있습니다. 서버리스 인스턴스가 용량과 관련된 지표 임계값에 도달하면 알림을 받도록 경보를 설정할 수 있습니다. 최소 및 최대 용량 설정이 적절한지 또는 조정이 필요한지 결정할 수 있습니다. 데이터베이스의 효율성을 최적화하기 위해 집중할 부분을 결정할 수 있습니다.

- **ServerlessDatabaseCapacity** - 인스턴스 수준 지표로서 현재 인스턴스 용량이 나타내는 DCU 숫자를 보고합니다. 클러스터 수준의 지표로 클러스터 내 모든 DocumentDB Serverless 인스턴스의 ServerlessDatabaseCapacity 값 평균을 나타냅니다.
- **DCUUtilization.** - 이 지표는 DocumentDB Serverless의 새로운 지표입니다. 이 값은 백분율로 표시됩니다. ServerlessDatabaseCapacity 지표의 값을 클러스터의 최대 DCU 값으로 나눈 값으로 계산됩니다. 이 지표를 해석하고 조치를 취하려면 다음 지침을 고려하세요.
  - 이 지표가 100.0 값에 가까워지면 인스턴스가 가능한 한 높게 스케일 업된 것입니다. 클러스터에 대한 최대 DCU 설정을 늘리는 것이 좋습니다. 이렇게 하면 라이터와 리더 인스턴스 모두 더 높은 용량으로 조정할 수 있습니다.
  - 읽기 전용 워크로드로 인해 리더 인스턴스가 100.0의 DCUUtilization에 접근하는 반면 라이터 인스턴스는 최대 용량에 근접하지 않는다고 가정해보겠습니다. 이 경우 클러스터에 리더 인스턴스를 추가하는 것이 좋습니다. 이렇게 하면 워크로드의 읽기 전용 부분을 더 많은 인스턴스에 분산시켜 각 리더 인스턴스의 로드를 줄일 수 있습니다.
  - 성능과 확장성이 주요 고려 사항인 프로덕션 애플리케이션을 실행하고 있다고 가정합니다. 이 경우 클러스터의 최대 DCU 값을 높은 값으로 설정할 수 있습니다. 목표는 DCUUtilization 지표가 항상 100.0 아래에 있도록 하는 것입니다. 최대 DCU 값이 높으면 데이터베이스 활동이 예기치 않게 급증하는 경우에 충분한 공간이 확보되어 있다는 확신을 가질 수 있습니다. 실제로는 사용한 데이터베이스 용량에 대해서만 요금이 청구됩니다.
- **CPUUtilization** - 이 지표는 프로비저닝된 인스턴스와 DocumentDB Serverless에서 다르게 해석됩니다. DocumentDB Serverless의 경우 이 값은 현재 사용 중인 CPU의 양을 클러스터의 최대 DCU 값 아래에서 사용 가능한 CPU 용량으로 나눈 값입니다. Amazon DocumentDB는 이 값을 자동으로 모니터링하고 인스턴스가 CPU 용량의 높은 비율을 지속적으로 사용하는 경우 서버리스 인스턴스를 스케일 업합니다.

  이 지표가 100.0 값에 가까워지면 인스턴스가 최대 CPU 용량에 도달한 것입니다. 클러스터에 대한 최대 DCU 설정을 늘리는 것이 좋습니다. 이 지표가 리더 인스턴스에서 100.0 값에 가까워지는 경우 클러스터에 리더 인스턴스를 추가하는 것이 좋습니다. 이렇게 하면 워크로드의 읽기 전용 부분을 더 많은 인스턴스에 분산시켜 각 리더 인스턴스의 로드를 줄일 수 있습니다.
- **FreeableMemory** - 이 값은 DocumentDB Serverless 인스턴스가 최대 용량으로 조정될 때 사용 가능한 미사용 메모리의 양을 나타냅니다. 현재 용량이 최대 용량 미만인 모든 DCU에 대해 이 값은 약 2GiB씩 증가합니다. 따라서 인스턴스가 가능한 한 높게 스케일 업될 때까지 이 지표는 0에 접근하지 않습니다.

  이 지표가 0 값에 접근하면 인스턴스가 가능한 한 스케일 업되었으며 사용 가능한 메모리 한도에 근접한 것입니다. 클러스터에 대한 최대 DCU 설정을 늘리는 것이 좋습니다. 이 지표가 리더 인스턴스에서 0 값에 가까워지는 경우 클러스터에 리더 인스턴스를 추가하는 것이 좋습니다. 이렇게 하면 워

크로드의 읽기 전용 부분을 더 많은 인스턴스에 분산하여 각 리더 인스턴스의 메모리 사용량을 줄일 수 있습니다.

- **`TempStorageIops`** - 인스턴스에 연결된 로컬 스토리지에서 수행된 IOPS 수입니다. 여기에는 읽기와 쓰기에 모두 대한 IOPS가 포함됩니다. 이 지표는 개수를 나타내며 초당 한 번 측정됩니다. DocumentDB Serverless에 대한 새로운 지표입니다. 자세한 내용은 CloudWatch를 사용하여 Amazon DocumentDB 모니터링을 참조하세요.
- **`TempStorageThroughput`** - 인스턴스와 연결된 로컬 스토리지에서 전송되는 데이터의 양입니다. 이 지표는 개수를 나타내며 초당 한 번 측정됩니다. DocumentDB Serverless에 대한 새로운 지표입니다. 자세한 내용은 CloudWatch를 사용하여 Amazon DocumentDB 모니터링을 참조하세요.

일반적으로 DocumentDB Serverless 인스턴스에 대한 대부분의 확장은 메모리 사용량과 CPU 활동으로 인해 발생합니다. `TempStorageIops` 및 `TempStorageThroughput` 지표는 인스턴스와 로컬 스토리지 디바이스 간의 전송에 대한 네트워크 활동이 예상치 못한 용량 증가의 원인이 되는 드문 경우를 진단하는 데 도움이 될 수 있습니다. 다른 네트워크 활동을 모니터링하려면 다음과 같은 기존 지표를 사용할 수 있습니다.

- `NetworkReceiveThroughput`
- `NetworkThroughput`
- `NetworkTransmitThroughput`
- `StorageNetworkReceiveThroughput`
- `StorageNetworkThroughput`
- `StorageNetworkTransmitThroughput`

DocumentDB 서버리스 CloudWatch 지표가 AWS 청구서에 적용되는 방법

AWS 청구서의 DocumentDB 서버리스 요금은 모니터링할 수 있는 것과 동일한 `ServerlessDatabaseCapacity` 지표를 기반으로 계산됩니다. DocumentDB Serverless 용량을 한 시간의 일부만 사용하는 경우 청구 메커니즘은 이 지표에 대해 계산된 CloudWatch 평균과 다를 수 있습니다. 시스템 문제로 인해 CloudWatch 지표를 짧은 기간 동안 사용할 수 없는 경우에도 다를 수 있습니다. 따라서 청구서에 DCU-시간 값이 `ServerlessDatabaseCapacity` 평균값에서 직접 계산하는 경우와 약간 다를 수 있습니다.

DocumentDB Serverless 지표에 대한 Amazon CloudWatch CLI 명령의 예

다음 AWS CLI 예제에서는 DocumentDB 서버리스와 관련된 가장 중요한 CloudWatch 지표를 모니터링하는 방법을 보여줍니다. 각 경우에 `--dimensions` 파라미터의 `Value=` 문자열을 고유한 DocumentDB Serverless 인스턴스의 식별자로 바꿉니다.

다음 Linux 예제는 1시간 동안 10분마다 측정된 인스턴스의 최소, 최대 및 평균 용량 값을 표시합니다. Linux date 명령은 현재 날짜 및 시각을 기준으로 시작 및 종료 시각을 지정합니다. `--query` 파라미터의 `sort_by` 함수는 `Timestamp` 필드를 기준으로 결과를 시간순으로 정렬합니다.

```
aws cloudwatch get-metric-statistics \
    --metric-name "ServerlessDatabaseCapacity" \
    --start-time "$(date -d '1 hour ago')" \
    --end-time "$(date -d 'now')" \
    --period 600 \
    --namespace "AWS/DocDB" \
    --statistics Minimum Maximum Average \
    --dimensions Name=DBInstanceIdentifier,Value=my_instance \
    --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' \
    --output table
```

다음 Linux 예제는 클러스터에 있는 인스턴스의 용량 모니터링을 보여줍니다. 인스턴스의 최소, 최대 및 평균 용량 사용률을 측정합니다. 측정은 3시간 동안 매시간에 한 번씩 측정됩니다. 이 예에서는 고정된 수의 DCU를 나타내는 `ServerlessDatabaseCapacity` 대신 DCU에 대한 상한의 백분율을 나타내는 `DCUUtilization` 지표를 사용합니다. 이렇게 하면 용량 범위에서 최소 및 최대 DCU 값의 실제 수치를 알 필요가 없습니다. 0에서 100 사이의 백분율을 볼 수 있습니다.

```
aws cloudwatch get-metric-statistics \
    --metric-name "DCUUtilization" \
    --start-time "$(date -d '3 hours ago')" \
    --end-time "$(date -d 'now')" \
    --period 3600 \
    --namespace "AWS/DocDB" \
    --statistics Minimum Maximum Average \
    --dimensions Name=DBInstanceIdentifier,Value=my_instance \
    --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' \
    --output table
```

다음 Linux 예제는 이전 예제와 유사한 측정을 수행합니다. 이 경우 측정은 CPUUtilization 지표에 대한 것입니다. 측정은 1시간 동안 10분마다 수행됩니다. 이 수치는 인스턴스의 최대 용량 설정까지 사용 가능한 CPU 리소스를 기준으로 사용된 사용 가능한 CPU의 백분율을 나타냅니다.

```
aws cloudwatch get-metric-statistics \
    --metric-name "CPUUtilization" \
    --start-time "$(date -d '1 hour ago')" \
    --end-time "$(date -d 'now')" \
    --period 600 \
    --namespace "AWS/DocDB" \
    --statistics Minimum Maximum Average \
    --dimensions Name=DBInstanceIdentifier,Value=my_instance \
    --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' \
    --output table
```

다음 Linux 예제는 이전 예제와 유사한 측정을 수행합니다. 이 경우 측정은 FreeableMemory 지표에 대한 것입니다. 측정은 1시간 동안 10분마다 수행됩니다.

```
aws cloudwatch get-metric-statistics \
    --metric-name "FreeableMemory" \
    --start-time "$(date -d '1 hour ago')" \
    --end-time "$(date -d 'now')" \
    --period 600 \
    --namespace "AWS/DocDB" \
    --statistics Minimum Maximum Average \
    --dimensions Name=DBInstanceIdentifier,Value=my_instance \
    --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' \
    --output table
```

성능 개선 도우미로 DocumentDB Serverless 성능 모니터링

성능 개선 도우미를 사용하여 DocumentDB Serverless 인스턴스의 성능을 모니터링할 수 있습니다. 성능 개선 도우미 프로시저는 성능 개선 도우미로 모니터링 페이지를 참조하세요.

다음과 같은 새로운 성능 개선 도우미 카운터가 DocumentDB Serverless 인스턴스에 적용됩니다.

- **os.general.serverlessDBCapacity** - 인스턴스의 현재 용량(DCU 단위)입니다. 인스턴스에 대한 ServerlessDatabaseCapacity CloudWatch 지표에 해당하는 값입니다.

- **os.general.dcuUtilization** - 구성된 최대 용량에서 현재 용량의 백분율입니다. 인스턴스에 대한 DCUUtilization CloudWatch 지표에 해당하는 값입니다.
- **os.general.maxConfiguredDcu** - 이 DocumentDB Serverless 인스턴스에 대해 구성한 최대 용량입니다. DCU로 측정됩니다.
- **os.general.minConfiguredDcu** - 이 DocumentDB Serverless 인스턴스에 대해 구성한 최소 용량입니다. DCU로 측정됩니다.

성능 개선 도우미 카운터의 전체 목록은 카운터 지표에 대한 성능 개선 도우미 페이지를 참조하세요.

성능 개선 도우미에서 DocumentDB Serverless 인스턴스에 대해 vCPU 값이 표시되는 경우 해당 값은 인스턴스에 대한 DCU 값을 기반으로 한 추정치를 나타냅니다. 기본 간격인 1분 간격으로 모든 분수 vCPU 값은 가장 가까운 정수로 반올림됩니다. 더 긴 간격의 경우 표시된 vCPU 값은 분당 정수 vCPU 값의 평균입니다.

Amazon DocumentDB 탄력적 클러스터 사용

Amazon DocumentDB 탄력적 클러스터는 초당 수백만 읽기/쓰기 및 페타바이트의 스토리지 용량을 갖춘 워크로드를 지원합니다. 또한 탄력적 클러스터는 인스턴스를 선택, 관리 또는 업그레이드할 필요가 없으므로 개발자가 Amazon DocumentDB와 상호 작용하는 방식을 단순화합니다.

> **Note**
>
> 탄력적 클러스터는 MongoDB 5.0 호환 와이어 프로토콜을 사용합니다. Amazon DocumentDB 엔진 버전 8.0에서는 지원되지 않으며 중국 리전 또는 AWS GovCloud에서는 사용할 수 없습니다.

Amazon DocumentDB 탄력적 클러스터는 다음과 같이 생성되었습니다.

- 풍부한 쿼리 기능과 MongoDB API 호환성을 통해 사실상 무한한 규모를 제공하는 데이터베이스를 찾는 고객에게 솔루션을 제공합니다.
- 고객에게 더 높은 연결 제한을 제공하고 패치로 인한 가동 중지 시간을 줄입니다.
- JSON 워크로드를 위한 클라우드 네이티브, 엘라스틱 및 최고 등급의 아키텍처에 지속적으로 투자하세요.

주제

- 탄력적 클러스터 사용 사례
- 탄력적 클러스터의 장점
- 탄력적 클러스터 리전 및 버전 가용성
- 제한 사항
- Amazon DocumentDB 탄력적 클러스터: 작동 방식
- Amazon DocumentDB 엘라스틱 클러스터 시작하기
- Amazon DocumentDB 탄력적 클러스터 모범 사례
- Amazon DocumentDB 탄력적 클러스터 관리
- Amazon DocumentDB 탄력적 클러스터에 대한 저장 데이터 암호화
- 탄력적 클러스터에서 서비스 연결 역할

탄력적 클러스터 사용 사례

문서 데이터베이스는 빠르고 반복적인 개발을 위해 유연한 스키마가 필요한 워크로드에 유용합니다. Amazon DocumentDB 사용 사례의 예는 문서 데이터베이스 사용 사례을 참조하세요.

다음은 탄력적 클러스터가 중요한 이점을 제공하는 일부 사용 사례의 예입니다.

사용자 프로필

문서 데이터베이스에는 유연한 스키마가 있으므로 규모에서 속성과 데이터 값이 다른 문서를 저장할 수 있습니다. 탄력적 클러스터는 서로 다른 사용자가 다양한 유형의 정보를 제공하는 온라인 프로필에 대한 실용적인 솔루션입니다. 내 애플리케이션이 수억 개의 사용자 프로필을 지원한다고 가정해 보겠습니다. 탄력적 클러스터를 사용하여 이러한 애플리케이션을 지원하실 수 있습니다. 사용자 프로필에 대한 수백만 건의 쓰기 및 읽기를 지원하도록 규모를 확장 및 축소할 수 있기 때문입니다. 피크 이외의 시간에도 확장하여 비용을 절감할 수 있습니다.

콘텐츠 관리 및 이력 기록

콘텐츠를 효과적으로 관리하려면 다양한 소스에서 콘텐츠를 수집 및 집계한 다음 고객에게 제공할 수 있어야 합니다. 문서 데이터베이스는 유연한 스키마로 인해 모든 유형의 데이터 수집과 저장에 적합합니다. 이미지, 코멘트 및 비디오와 같은 사용자 생성 콘텐츠를 포함하여 새로운 유형의 콘텐츠를 생성하고 통합하는 데 사용할 수 있습니다. 시간이 지남에 따라 데이터베이스에 더 많은 스토리지가 필요할 수 있습니다. 엘라스틱 클러스트를 사용하면 더 많은 스토리지 볼륨에 데이터를 분산하여 단일 클러스터에 페타바이트 규모의 데이터를 저장할 수 있습니다.

탄력적 클러스터의 장점

AWS 서비스 통합

Amazon DocumentDB 탄력적 클러스터는 Amazon DocumentDB와 동일한 방식으로 다른 AWS 서비스와 통합됩니다.

- 마이그레이션 - AWS Database Migration Service (DMS)를 사용하여 MongoDB 및 기타 관계형 데이터베이스에서 Amazon DocumentDB 탄력적 클러스터로 마이그레이션할 수 있습니다.
- 모니터링 - Amazon CloudWatch를 사용하여 탄력적 클러스터의 상태와 성능을 모니터링할 수 있습니다.

- 보안 - AWS Identity and Access Management (IAM)을 통해 인증 및 권한 부여를 설정하여 탄력적 클러스터 생성 및 업데이트와 같은 탄력적 클러스터 리소스를 관리하고 보안 VPC 전용 연결에 Amazon VPC를 사용할 수 있습니다. 탄력적 클러스터에 대한 IAM 인증 로그인은 지원되지 않습니다.
- 데이터 관리 - AWS Glue 를 사용하여 Amazon S3, Amazon Redshift 및 Amazon OpenSearch Service와 같은 다른 AWS 서비스에서 데이터를 가져오고 내보낼 수 있습니다.

탄력적 클러스터 리전 및 버전 가용성

탄력적 클러스터의 리전 가용성

다음 표에는 Amazon DocumentDB 탄력적 클러스터를 현재 사용할 수 AWS 리전 있는와 각 리전의 엔드포인트가 나와 있습니다.

리전 이름	리전	가용 영역
미국 동부(버지니아 북부)	us-east-1	5
미국 동부(오하이오)	us-east-2	3
미국 서부(오리건)	us-west-2	3
아시아 태평양(홍콩)	ap-east-1	3
아시아 태평양(뭄바이)	ap-south-1	3
아시아 태평양(서울)	ap-northeast-2	3
아시아 태평양(싱가포르)	ap-southeast-1	3
아시아 태평양(시드니)	ap-southeast-2	3
아시아 태평양(도쿄)	ap-northeast-1	3
캐나다(중부)	ca-central-1	3
남아메리카(상파울루)	sa-east-1	3
유럽(프랑크푸르트)	eu-central-1	3

리전 이름	리전	가용 영역
유럽(아일랜드)	eu-west-1	3
유럽(런던)	eu-west-2	3
유럽(밀라노)	eu-south-1	3
유럽(파리)	eu-west-3	3

버전 가용성

탄력적 클러스터는 MongoDB 5.0과 호환되는 와이어 프로토콜을 지원합니다. Amazon DocumentDB 4.0 인스턴스 기반 클러스터 및 탄력적 클러스터 간의 차이점은 Amazon DocumentDB 4.0과 탄력적 클러스터 간의 기능적 차이 섹션을 참조하세요. 참고: Amazon DocumentDB 8.0에서는 Elastic Clusters가 지원되지 않습니다.

제한 사항

탄력적 클러스터 관리

이번 릴리스에서는 다음의 클러스터 관리 기능이 지원되지 않습니다.

- 사용자 지정 포트(27017만 지원됨)
- IAM 사용자 및 역할을 사용하여 데이터베이스에 인증할 수 있는 기능
- 글로벌 클러스터를 생성할 수 있는 기능.
- 기존 Amazon DocumentDB 이벤트와 이벤트 구독
- 레인지 샤딩
- 기존 컬렉션 샤드
- 멀티필드 샤드 키
- 샤드 키 변경하기
- 특정 시점 복원
- 복제

- 성능 개선 도우미

> **Note**
>
> 탄력적 클러스터 할당량에 대한 자세한 내용은 섹션을 참조하세요Amazon DocumentDB 할당량.

쿼리 및 쓰기 작업

이번 릴리스에서는 다음과 같은 쿼리 및 쓰기 작업 명령과 기능이 지원되지 않습니다.

- 규모 조정 작업 중의 DDL 명령
- 프로파일러
- 파라미터 그룹
- AWS Config
- AWS Backup

컬렉션 및 인덱스 관리

이번 릴리스에서는 다음과 같은 컬렉션 및 인덱스 관리 기능이 지원되지 않습니다.

- 고유 인덱스
- 부분 인덱스
- 텍스트 인덱스
- 벡터 인덱스
- 문서 압축

관리 및 진단

이번 릴리스에서는 다음과 같은 관리 및 진단 명령과 기능이 지원되지 않습니다.

- AWS Secrets Manager
- 역할 기반 액세스 제어(RBAC) 사용자 정의 역할.

- 연결 시 0의 쓰기 문제는 지원되지 않습니다.
- 기존 탄력적 클러스터에 현재 할당되지 않은 VPC에 속하는 서브넷을 변경.

옵트인 기능

이번 릴리스에서는 다음과 같은 Amazon DocumentDB 옵트인 기능이 지원되지 않습니다.

- ACID 트랜잭션
- DDL/DML 감사
- Change streams
- 세션 명령

Amazon DocumentDB 탄력적 클러스터: 작동 방식

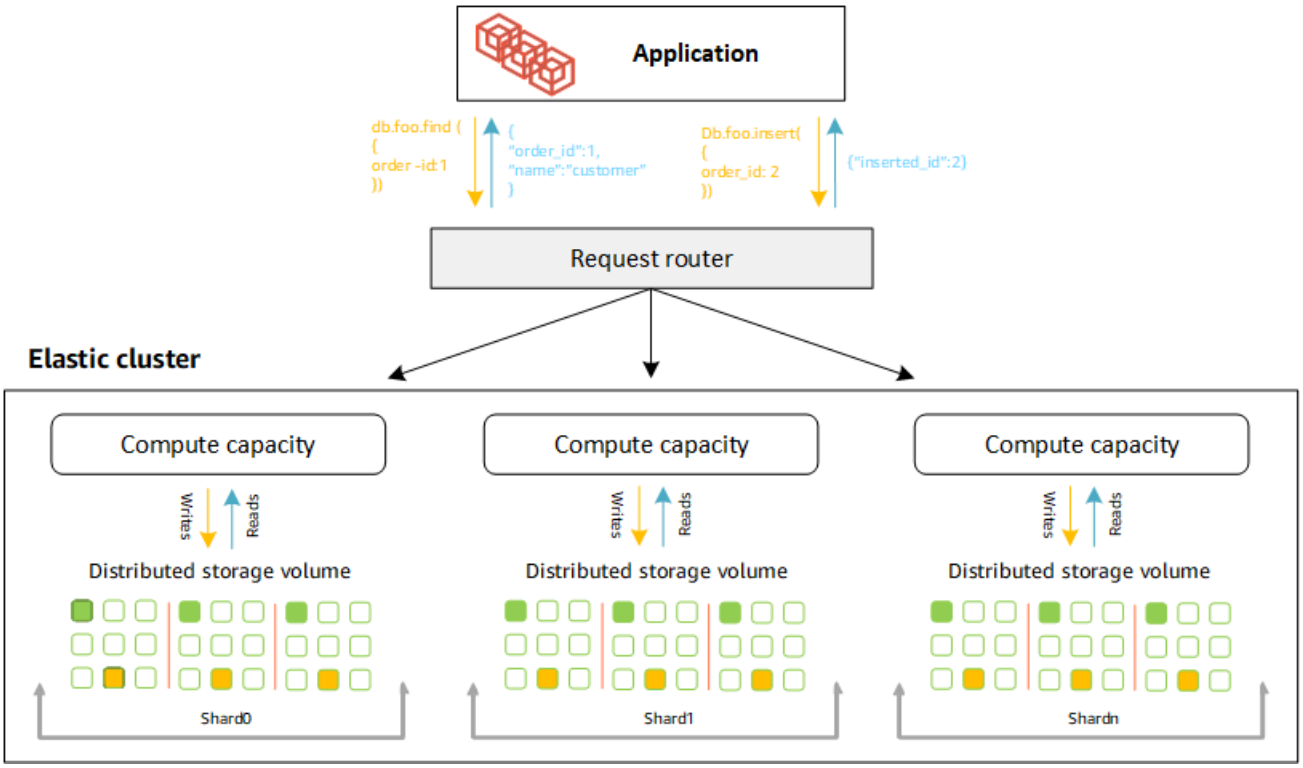
이 섹션의 주제에서는 Amazon DocumentDB 탄력적 클러스터를 지원하는 메커니즘 및 기능에 대한 정보를 제공합니다.

주제

- Amazon DocumentDB 탄력적 클러스터 샤딩
- 탄력적 클러스터 마이그레이션
- 탄력적 클러스터 규모 조정
- 탄력적 클러스터 안정성
- 탄력적 클러스터 스토리지 및 가용성
- Amazon DocumentDB 4.0과 탄력적 클러스터 간의 기능적 차이

Amazon DocumentDB 탄력적 클러스터 샤딩

Amazon DocumentDB 탄력적 클러스터는 해시 기반 샤딩을 사용하여 분산 스토리지 시스템에서 데이터를 분할합니다. 파티셔닝이라고도 하는 샤딩은 대규모 데이터 세트를 여러 노드에 걸쳐 작은 데이터 세트로 분할하므로 수직 규모 조정 한도 이상으로 데이터베이스를 규모 조정할 수 있습니다. 탄력적 클러스터는 Amazon DocumentDB의 컴퓨팅과 스토리지를 구분 또는 "분리"하므로 서로 독립적으로 규모를 조정할 수 있습니다. 탄력적 클러스터는 컴퓨팅 노드 간에 작은 데이터 청크를 이동하여 컬렉션을 다시 분할하는 대신 분산 스토리지 시스템 내에서 데이터를 효율적으로 복사합니다.



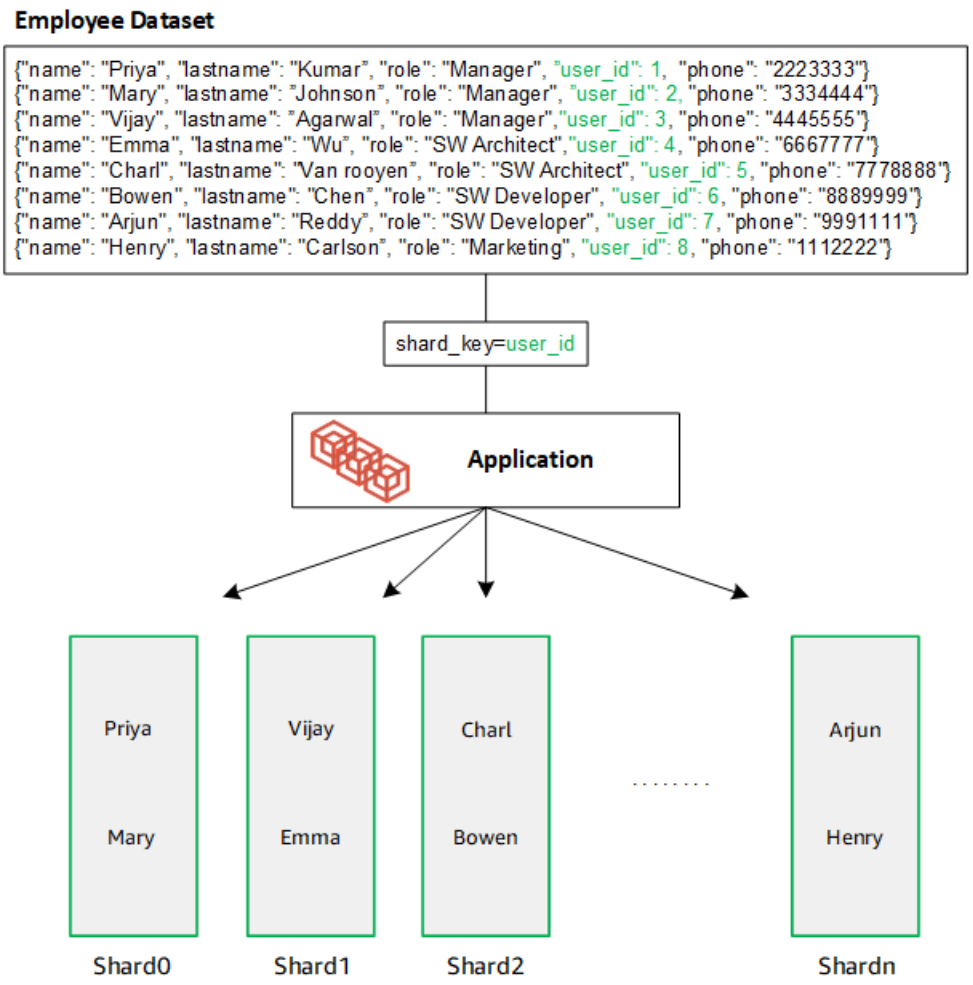
샤드 정의

샤드 명명법의 정의:

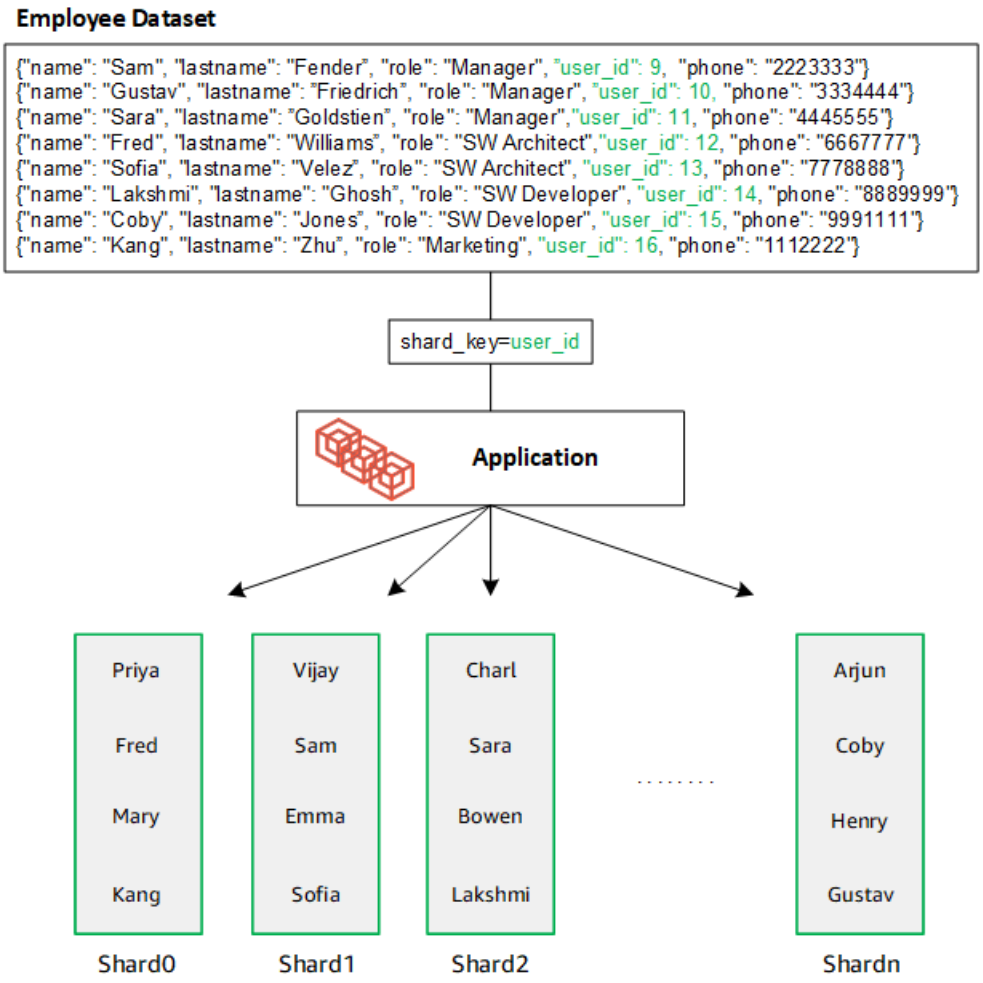
- 샤드 - 샤드는 탄력적 클러스터를 위한 컴퓨팅을 제공합니다. 단일 라이터 인스턴스와 0~15개의 읽기 전용 복제본이 있습니다. 기본적으로 샤드에는 라이터와 단일 읽기 전용 복제본이라는 두 개의 인스턴스가 있습니다. 샤드는 최대 32개까지 구성할 수 있으며 각 샤드 인스턴스는 최대 64개의 vCPU를 포함할 수 있습니다.
- 샤드 키 - 샤드 키는 탄력적 클러스터가 읽기 및 쓰기 트래픽을 일치하는 샤드에 분배하는 데 사용하는 샤드 컬렉션의 JSON 문서의 필수 필드입니다.
- 샤드 컬렉션 - 샤드 컬렉션은 데이터가 데이터 파티션의 탄력적 클러스터 전체에 분산되어 있는 컬렉션입니다.
- 파티션 - 파티션은 샤딩된 데이터의 논리적 부분입니다. 샤딩된 컬렉션을 만들면 샤드 키를 기반으로 데이터가 각 샤드 내의 파티션으로 자동 구성됩니다. 각 샤드에는 여러 개의 파티션이 있습니다.

구성된 샤드에 데이터 배포

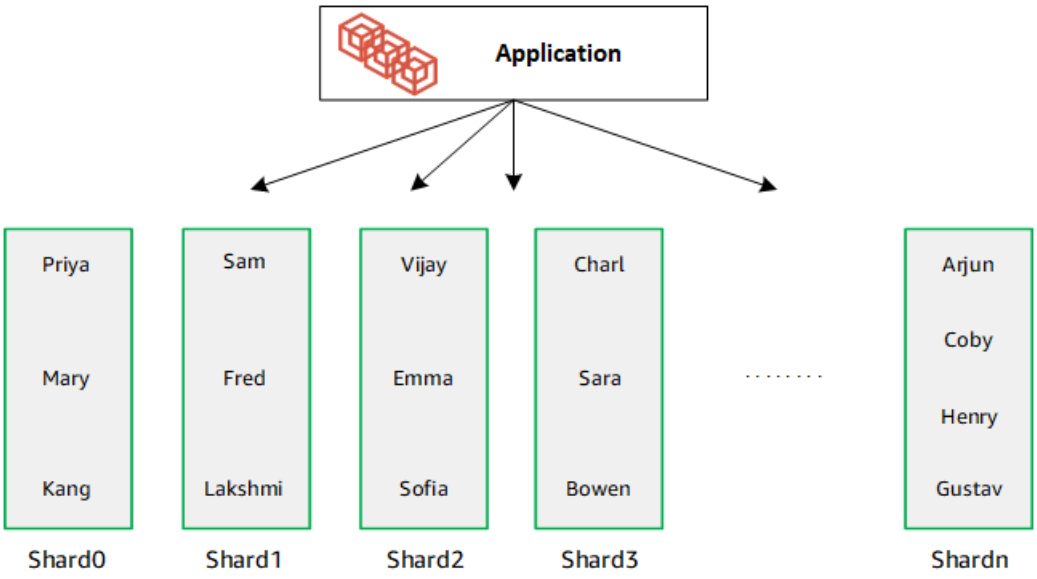
고유한 값이 많은 샤드 키를 생성하세요. 좋은 샤드 키는 데이터를 기본 분할된 데이터베이스로 균등하게 분할하여 워크로드에 최상의 처리량과 성능을 제공합니다. 다음은 "user_id"라는 샤드 키를 사용하는 직원 이름 데이터를 예로 들 수 있습니다.



DocumentDB는 해시 샤딩을 사용하여 데이터를 기본 샤드로 분할합니다. 추가 데이터는 다음과 같은 방식으로 삽입 및 배포됩니다.



샤드를 추가하여 데이터베이스를 규모 조정하면 Amazon DocumentDB가 데이터를 자동으로 재배포합니다.



탄력적 클러스터 마이그레이션

Amazon DocumentDB는 MongoDB 샤딩 데이터를 탄력적 클러스터로 마이그레이션하는 것을 지원합니다. 오프라인, 온라인 및 하이브리드 마이그레이션 방법이 지원됩니다. 자세한 내용은 Amazon DocumentDB 마이그레이션 및 업그레이드 단원을 참조하십시오.

탄력적 클러스터 규모 조정

Amazon DocumentDB 탄력적 클러스터는 탄력적 클러스터의 샤드 수를 늘리고(스케일 아웃) 각 샤드에 적용되는 vCPU의 수를 늘릴 수 있는 기능(스케일 업)을 제공합니다. 또한 필요에 따라 샤드 수와 컴퓨팅 용량(vCPU)을 줄일 수 있습니다.

규모 조정 모범 사례는 탄력적 클러스터 크기 조정을 참조하세요.

> **Note**
>
> 클러스터 수준 규모 조정도 가능합니다. 자세한 내용은 Amazon DocumentDB 클러스터 규모 조정 단원을 참조하십시오.

탄력적 클러스터 안정성

Amazon DocumentDB는 안정성, 내구성 및 내결함성을 고려하여 설계되었습니다. 가용성을 개선하기 위해 탄력적 클러스터는 여러 가용 영역에 배치된 샤드당 두 개의 노드를 배포합니다. Amazon DocumentDB에는 신뢰할 수 있는 데이터베이스 솔루션으로 만들어 주는 몇 가지 자동 기능이 포함되어 있습니다. 자세한 내용은 Amazon DocumentDB 신뢰성 단원을 참조하십시오.

탄력적 클러스터 스토리지 및 가용성

Amazon DocumentDB 데이터는 SSD(Solid State Drive)를 사용하는 단일 가상 볼륨인 클러스터 볼륨에 저장됩니다. 클러스터 볼륨은 단일 AWS 리전의 여러 가용 영역에 자동으로 복제되는 6개의 데이터 복사본으로 구성됩니다. 이 복제를 통해 데이터의 내구성을 높이고 데이터 손실 가능성을 줄일 수 있습니다. 또한 다른 가용 영역에 데이터 복사본이 이미 있어 장애 조치가 이루어지는 동안 클러스터 가용성이 높아집니다. 스토리지, 고가용성 및 복제에 대한 자세한 내용은 Amazon DocumentDB: 작동 방식 섹션을 참조하세요.

Amazon DocumentDB 4.0과 탄력적 클러스터 간의 기능적 차이

Amazon DocumentDB 4.0과 탄력적 클러스터 간에는 다음과 같은 기능적 차이가 있습니다.

- top 및 collStats의 결과는 샤드로 분할됩니다. 샤딩된 컬렉션의 경우 데이터는 여러 파티션에 분산되고 collStats 보고서는 파티션collScans에서 집계됩니다.
- 클러스터 샤드 수가 변경되면 샤딩된 컬렉션의 컬렉션 top 통계와 collStats 샤딩된 컬렉션의 컬렉션 통계가 재설정됩니다.
- 이제 백업 빌트인 역할이 지원됩니다. serverStatus 작업 - 백업 역할을 가진 개발자 및 응용프로그램은 Amazon DocumentDB 클러스터의 상태에 대한 통계를 수집할 수 있습니다.
- replSetGetConfig 출력에서 SecondaryDelaySecs 필드는 slaveDelay를 대체합니다.
- hello 명령은 isMaster를 대체합니다 - hello는 탄력적 클러스터의 역할을 설명하는 문서를 반환합니다.
- 탄력적 클러스터의 $elemMatch 연산자는 배열의 첫 번째 중첩 수준에 있는 문서만 일치시킵니다. Amazon DocumentDB 4.0에서는 작업자가 모든 수준을 순회한 후 일치하는 문서를 반환합니다. 예제:

```
db.foo.insert(
[
    {a: {b: 5}},
    {a: {b: [5]}},
    {a: {b: [3, 7]}},
    {a: [{b: 5}]},
    {a: [{b: 3}, {b: 7}]},
    {a: [{b: [5]}]},
    {a: [{b: [3, 7]}]},
    {a: [[{b: 5}]]},
    {a: [[{b: 3}, {b: 7}]]},
    {a: [[{b: [5]}]]},
    {a: [[{b: [3, 7]}]]}
]);
// Elastic clusters
> db.foo.find({a: {$elemMatch: {b: {$elemMatch: {$lt: 6, $gt: 4}}}}}, {_id: 0})
{ "a" : [ { "b" : [ 5 ] } ] }

// Docdb 4.0: traverse more than one level deep
> db.foo.find({a: {$elemMatch: {b: {$elemMatch: {$lt: 6, $gt: 4}}}}}, {_id: 0})
{ "a" : [ { "b" : [ 5 ] } ] }
{ "a" : [ [ { "b" : [ 5 ] } ] ] }
```

- Amazon DocumentDB 4.0의 "$" 프로젝션은 모든 필드가 포함된 모든 문서를 반환합니다. 탄력적 클러스터의 경우 "$" 프로젝션을 사용하는 `find` 명령은 "$" 프로젝션과 일치하는 필드만 포함하는 쿼리 파라미터와 일치하는 문서를 반환합니다.
- 탄력적 클러스터에서 `$regex` 및 `$options` 쿼리 매개 변수를 사용하는 `find` 명령은 "$regex 및 $options 모두에서 옵션을 설정할 수 없습니다." 라는 오류를 반환합니다.

- 탄력적 클러스터를 사용하면 `$indexOfCP`은 이제 다음과 같은 경우 "-1"을 반환합니다.
  - `string expression`에서 하위 문자열을 찾을 수 없거나
  - `start`이 `end`보다 큰 숫자이거나
  - `start`이 문자열의 바이트 길이보다 큰 수인 경우입니다.

  Amazon DocumentDB 4.0에서 `$indexOfCP`는 `start` 위치가 `end`이나 문자열의 바이트 길이보다 크면 "0"을 반환합니다.
- 탄력적 클러스터를 사용하면 `_id fields`의 프로젝션 작업(예: `{"_id.nestedField" : 1}`)이 프로젝션된 필드만 포함하는 문서를 반환합니다. 단, Amazon DocumentDB 4.0에서는 중첩 필드 프로젝션 명령이 문서를 필터링하지 않습니다.

Amazon DocumentDB 엘라스틱 클러스터 시작하기

이 시작하기 섹션에서는 첫 번째 엘라스틱 클러스터를 생성하고 쿼리하는 방법을 안내합니다.

Amazon DocumentDB에 연결하고 이를 통해 시작하는 방법에는 여러 가지가 있습니다. 다음 절차는 사용자가 강력한 도큐먼트 데이터베이스를 사용하여 시작할 수 있는 가장 빠르고 간단하며 쉬운 방법입니다. 이 가이드에서는 AWS CloudShell을 사용하여 AWS Management Console에서 직접 Amazon DocumentDB 클러스터를 연결하고 쿼리합니다. AWS 프리 티어 자격이 있는 신규 고객은 Amazon DocumentDB 및 CloudShell을 무료로 사용할 수 있습니다. AWS CloudShell 환경 또는 Amazon DocumentDB 탄력적 클러스터가 프리 티어 이외의 리소스를 사용하는 경우 해당 리소스에 대한 일반 AWS 요금이 부과됩니다. 이 안내서를 통해 5분 이내에 Amazon DocumentDB를 시작할 수 있습니다.

주제

- 사전 조건
- 1단계: 엘라스틱 클러스터 생성
- 2단계: 탄력적 클러스터에 연결
- 3단계: 컬렉션 샤드, 데이터 삽입 및 쿼리

- 4단계: 탐색

사전 조건

첫 번째 Amazon DocumentDB 클러스터를 작성하기 전에 다음 작업을 수행해야 합니다:

Amazon Web Services(AWS) 계정 만들기

Amazon DocumentDB 사용을 시작하기 전에 Amazon Web Services(AWS) 계정이 있어야 합니다. AWS 계정은 무료입니다. 사용하는 서비스 및 리소스에 대해서만 비용을 지불하는 것입니다.

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://signin.aws.amazon.com/signup?request_type=register를 엽니다.
2. 온라인 지시 사항을 따르세요.

   등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

필요한 AWS Identity and Access Management (IAM) 권한을 설정합니다.

클러스터, 인스턴스 및 클러스터 파라미터 그룹과 같은 Amazon DocumentDB 리소스를 관리하려면가 요청을 인증하는 데 사용할 AWS 수 있는 자격 증명이 필요합니다. 자세한 내용은 Amazon DocumentDB의 ID 및 액세스 관리 섹션을 참조하세요.

1. 의 검색 창에 드롭다운 메뉴에서 IAM을 AWS Management Console**IAM** 입력하고 선택합니다.
2. IAM 콘솔에 들어가면 탐색 창에서 사용자를 선택합니다.
3. 사용자 이름을 선택합니다.
4. 권한 추가를 선택합니다.
5. 정책 직접 연결을 선택합니다.

6. 검색 창에 `AmazonDocDBElasticFullAccess`을 입력하고 검색 결과에 표시되면 선택합니다.
7. 다음(Next)을 선택합니다.
8. 권한 추가를 선택합니다.

> **Note**
>
> AWS 계정에는 각 리전의 기본 VPC가 포함됩니다. Amazon VPC를 사용하기로 선택한 경우 Amazon VPC 사용 설명서에서 Amazon VPC 생성 주제의 단계를 완료합니다.

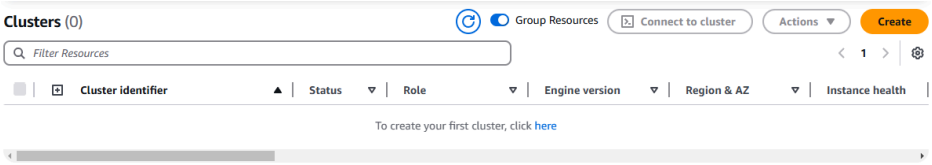
1단계: 엘라스틱 클러스터 생성

이 섹션에서는 AWS CLI 다음 지침에 따라 AWS Management Console 또는를 사용하여 새로운 탄력적 클러스터를 생성하는 방법을 설명합니다.

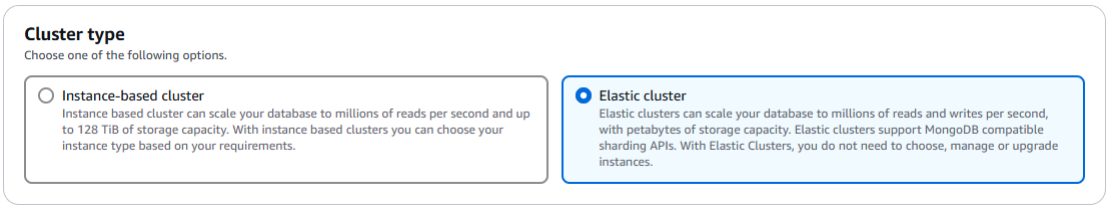
Using the AWS Management Console

AWS Management Console을 사용하여 엘라스틱 클러스터 구성을 생성하려면:

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. Amazon DocumentDB 관리 콘솔의 클러스터에서 생성을 선택합니다.



3. Amazon DocumentDB 클러스터 생성 페이지의 클러스터 유형 섹션에서 탄력적 클러스터를 선택합니다.



4. 구성 섹션에서 다음을 구성합니다.

a. 클러스터 이름 필드에 고유한 클러스터 식별자를 입력합니다(필드 아래의 이름 지정 요구 사항 준수).

b. 샤드 수 필드에서, 클러스터에 포함하려는 샤드 수를 입력합니다. 클러스터당 최대 샤드 수는 32개입니다.

> **Note**
>
> 각 샤드마다 두 개의 노드가 배포됩니다. 두 노드의 샤드 용량은 동일합니다.

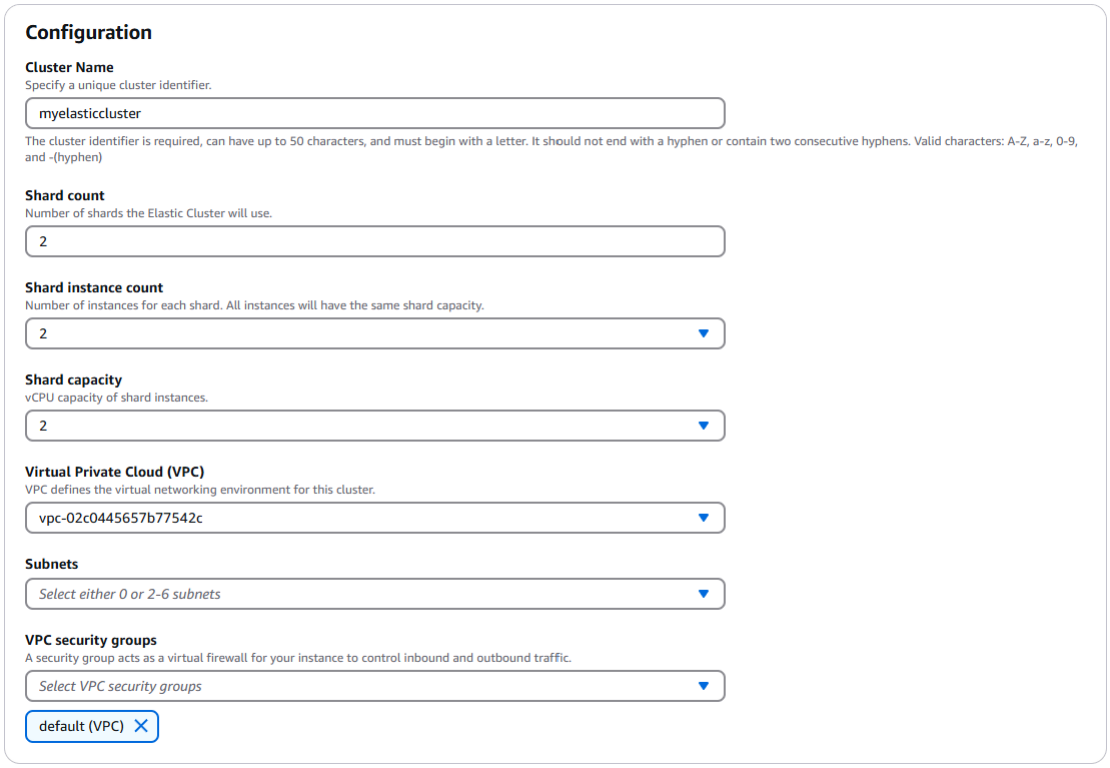
c. 샤드 인스턴스 수 필드에서 각 샤드와 연결할 복제본 인스턴스 수를 선택합니다. 최대 샤드 인스턴스 수는 16개이며 1씩 증가합니다. 모든 복제본 인스턴스는 다음 필드에 정의된 것과 동일한 샤드 용량을 갖습니다. 테스트 목적으로는 기본값인 2로 충분합니다.

> **Note**
>
> 복제본 인스턴스의 수는 탄력적 클러스터의 모든 샤드에 적용됩니다. 샤드 인스턴스 수의 값이 1이면 쓰기 인스턴스가 하나라는 의미이며, 추가 인스턴스는 읽기에 사용할 수 있고 가용성을 개선하는 복제본입니다. 테스트 목적으로는 기본값인 2로 충분합니다.

d. 샤드 용량 필드에서 각 샤드 인스턴스에 연결할 가상 CPU(vCPU) 수를 선택합니다. 샤드 인스턴스당 최대 vCPU 수는 64개입니다. 허용되는 값은 2, 4, 8, 16, 32, 64입니다. 테스트 목적으로는 기본값인 2로 충분합니다.

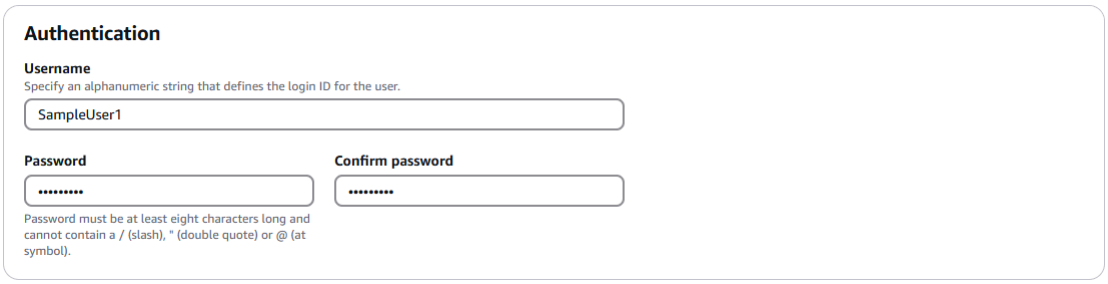
e. 가상 프라이빗 클라우드(VPC) 필드에서 드롭다운 목록 중 VPC를 선택합니다.

f. 서브넷 및 VPC 보안 그룹의 경우, 기본값을 사용하거나 원하는 서브넷 3개와 VPC 보안 그룹 최대 3개(최소 1개)를 선택할 수 있습니다.



5. 인증 섹션의 사용자 이름 필드에 기본 사용자의 로그인 이름을 식별하는 문자열을 입력합니다.

   암호 필드에 지침을 준수하는 고유한 암호를 입력한 후 확인합니다.



6. 암호화 섹션에서 기본 설정(기본 키)을 유지합니다.

   선택적으로 생성한 AWS KMS key ARN을 입력할 수 있습니다. 자세한 내용은 Amazon DocumentDB 탄력적 클러스터에 대한 저장 데이터 암호화 섹션을 참조하세요.

> **Important**
>
> 엘라스틱 클러스터에 대해 암호화를 활성화해야 합니다.

7. 백업 섹션에서 백업 요구 사항에 따라 필드를 편집합니다. 테스트 목적으로 기본 설정을 유지할 수 있습니다.

   a. 백업 보존 기간 - 목록에서 이 클러스터의 자동 백업을 삭제하지 않고 유지하는 일 수를 선택합니다.

   b. 백업 기간 - Amazon DocumentDB에서 이 클러스터를 백업하는 일별 시간과 기간을 설정합니다.

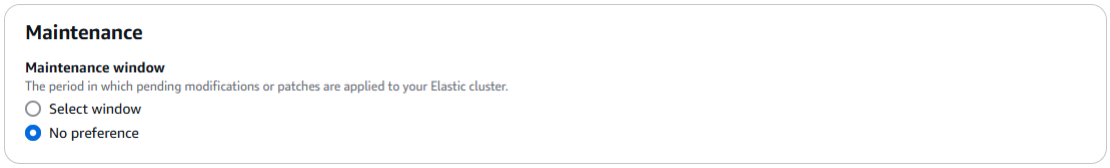
      i. 백업이 생성되는 시간과 기간을 구성하려면 창 선택을 선택합니다.

         시작 시간 - 첫 번째 목록에서 자동 백업 시작 시간(UTC)을 선택합니다. 두 번째 목록에서 자동 백업 시작 시간의 분을 선택합니다.

         기간 - 목록에서 자동 백업 생성에 할당된 시간 수를 선택합니다.

      ii. Amazon DocumentDB가 백업이 생성되는 시간과 기간을 선택하도록 하려면 기본 설정 없음을 선택합니다.

8. 유지 관리 섹션에서 클러스터에 수정 사항 또는 패치를 적용할 날짜, 시간, 기간을 선택합니다. 테스트 목적으로 기본 설정을 유지할 수 있습니다.



9. 클러스터 생성을 선택합니다.

이제 엘라스틱 클러스터를 프로비저닝하는 중입니다. 이 과정은 완료하는 데 최대 수분 소요될 수 있습니다. 클러스터 목록에서 탄력적 클러스터 상태가 사용 가능으로 표시되면 클러스터에 연결할 수 있습니다.

Using the AWS CLI

를 사용하여 탄력적 클러스터를 생성하려면 다음 파라미터와 함께 `create-cluster` 작업을 AWS CLI사용합니다.

- --cluster-name - 필수입니다. 생성시 입력했거나 마지막으로 수정한 엘라스틱 스케일 클러스터의 현재 이름.
- --shard-capacity - 필수입니다. 각 샤드에 할당된 vCPU 개수입니다. 최대값은 64입니다. 허용되는 값은 2, 4, 8, 16, 32, 64입니다.
- --shard-count - 필수입니다. 클러스터에 할당된 샤드 개수입니다. 최대 값은 32입니다.
- --shard-instance-count - 선택 사항. 이 클러스터의 모든 샤드에 적용되는 복제본 인스턴스 수입니다. 최대 값은 16입니다.
- --admin-user-name - 필수입니다. 관리자 사용자와 연결된 사용자 이름입니다.
- --admin-user-password - 필수입니다. 관리자와 연결된 암호입니다.
- --auth-type - 필수입니다. 엘라스틱 클러스터에 액세스하는 데 사용되는 암호를 가져올 위치를 결정하는 데 사용되는 인증 유형입니다. 유효한 형식은 `PLAIN_TEXT` 또는 `SECRET_ARN`입니다.
- --vpc-security-group-ids - 선택 사항. 이 클러스터와 연결할 EC2 VPC 보안 그룹 목록을 구성합니다.
- --preferred-maintenance-window - 선택 사항. 시스템 유지보수가 수행될 수 있는 주간 시간 범위를 UTC(Universal Coordinated Time)로 구성합니다.

  형식은 `ddd:hh24:mi-ddd:hh24:mi`입니다. 유효한 요일(ddd): 월, 화, 수, 목, 금, 토, 일

  기본값은 각 Amazon Web Services Region(아마존 웹 서비스 리전)의 8시간 블록에서 임의로 선택된 30분 창으로, 주 마다 임의의 요일에 발생합니다.

  최소 30분의 시간.
- --kms-key-id - 선택 사항. 암호화된 클러스터의 KMS 키 식별자를 구성하세요.

  KMS 키 식별자는 AWS KMS 암호화 키의 Amazon 리소스 이름(ARN)입니다. 새 클러스터를 암호화하는 데 사용되는 KMS 암호화 키를 소유한 동일한 Amazon Web Services 계정을 사용하여 클러스터를 생성하는 경우, KMS 암호화 키에 대한 ARN 대신 KMS 키 별칭을 사용할 수 있습니다.

  암호화 키가 KmsKeyId에 지정되어 있지 않고 `StorageEncrypted` 파라미터가 참이면 Amazon DocumentDB는 기본 암호화 키를 사용합니다.
- --preferred-backup-window - 선택 사항. 자동 백업이 생성되는 일일 선호 시간 범위입니다. 기본값은 각각 8시간 블록 시간 중에서 임의로 선택한 30분 기간입니다 AWS 리전.
- --backup-retention-period - 선택 사항. 자동 백업이 보존되는 일수입니다. 기본값은 1입니다.
- --storage-encrypted - 선택 사항. 클러스터의 암호화 여부를 구성합니다.

`--no-storage-encrypted`은 클러스터가 암호화되지 않도록 지정합니다.

- --subnet-ids - 선택 사항. 네트워크 서브넷 ID를 구성합니다.

다음 예제에서는 자신의 정보로 각각의 *### ## ## ###*를 바꿉니다.

> **Note**
>
> 다음 예에는 특정 KMS 키 생성이 포함됩니다. 기본 KMS 키를 사용하려면 `--kms-key-id` 파라미터를 포함하지 마세요.

Linux, macOS 또는 Unix의 경우:

```
aws docdb-elastic create-cluster \
     --cluster-name sample-cluster-123 \
     --shard-capacity 8 \
     --shard-count 4 \
     --shard-instance-count 3 \
     --auth-type PLAIN_TEXT \
     --admin-user-name testadmin \
     --admin-user-password testPassword \
     --vpc-security-group-ids ec-65f40350 \
     --kms-key-id arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 \
     --subnet-ids subnet-9253c6a3, subnet-9f1b5af9 \
     --preferred-backup-window 18:00-18:30 \
     --backup-retention-period 7
```

Windows의 경우:

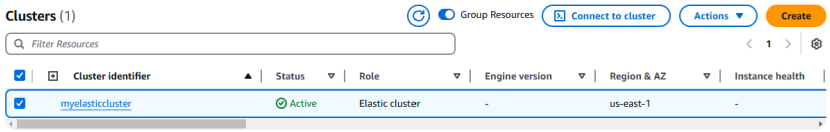
```
aws docdb-elastic create-cluster ^
     --cluster-name sample-cluster-123 ^
     --shard-capacity 8 ^
     --shard-count 4 ^
     --shard-instance-count 3 ^
     --auth-type PLAIN_TEXT ^
     --admin-user-name testadmin ^
     --admin-user-password testPassword ^
     --vpc-security-group-ids ec-65f40350 ^
```

```
    --kms-key-id arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 ^
    --subnet-ids subnet-9253c6a3, subnet-9f1b5af9 \
    --preferred-backup-window 18:00-18:30 \
    --backup-retention-period 7
```

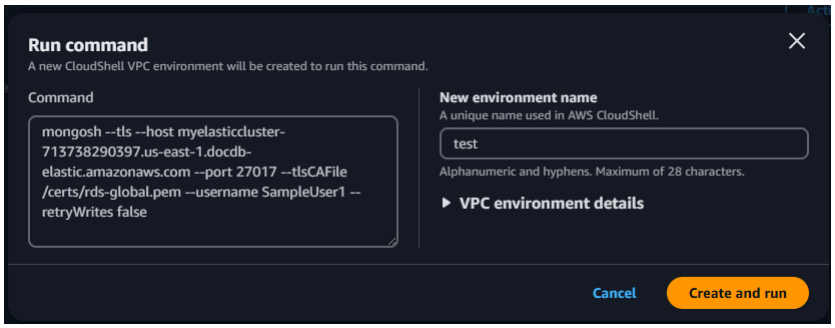
2단계: 탄력적 클러스터에 연결

를 사용하여 Amazon DocumentDB 탄력적 클러스터에 연결합니다 AWS CloudShell.

1. Amazon DocumentDB 관리 콘솔의 클러스터에서 생성한 탄력적 클러스터를 찾습니다. 클러스터 옆의 확인란을 클릭하여 클러스터를 선택합니다.



2. 클러스터에 연결을 선택합니다(작업 드롭다운 메뉴 옆에 있음. 이 버튼은 클러스터 옆의 확인란을 클릭한 후에만 활성화되며 클러스터의 상태는 사용 가능으로 표시됩니다. CloudShell 명령 실행 화면이 나타납니다.
3. 새 환경 이름 필드에 '테스트'와 같은 고유한 이름을 입력하고 생성 및 실행을 선택합니다. VPC 환경 세부 정보는 Amazon DocumentDB 데이터베이스에 대해 자동으로 구성됩니다.



4. 메시지가 표시되면 1단계: Amazon DocumentDB 탄력적 클러스터 생성(하위 5단계)에서 생성한 암호를 입력합니다.

암호를 입력한 후 프롬프트에 `direct: mongos] <env-name>>`이 표시되면 Amazon DocumentDB 클러스터에 성공적으로 연결된 것입니다.

> **Note**
>
> 흐름 문제 해결에 대한 자세한 내용은 Amazon DocumentDB 문제 해결을 참조하십시오.

3단계: 컬렉션 샤드, 데이터 삽입 및 쿼리

엘라스틱 클러스터는 Amazon DocumentDB에서 샤딩에 대한 지원을 추가합니다. 이제 클러스터에 연결되었으므로 클러스터를 샤딩, 데이터를 삽입하고 몇 가지 쿼리를 실행할 수 있습니다.

1. 컬렉션을 샤드하려면 다음을 입력합니다.

```
sh.shardCollection("db.Employee1" , { "Employeeid" : "hashed" })
```

2. 단일 문서를 삽입하려면 다음을 입력합니다:

```
db.Employee1.insertOne({"Employeeid":1, "Name":"Joe", "LastName": "Bruin", "level":
 1 })
```

   다음 결과가 표시됩니다:

```
WriteResult({ "nInserted" : 1 })
```

3. 작성한 문서를 읽으려면 `findOne()` 명령을 입력합니다(단일 문서만 반환함).

```
db.Employee1.findOne()
```

   다음 결과가 표시됩니다:

```
{
"_id" : ObjectId("61f344e0594fe1a1685a8151"),
"EmployeeID" : 1,
"Name" : "Joe",
"LastName" : "Bruin",
"level" : 1
```

```
}
```

4. 쿼리를 몇 개 더 수행하려면 게임 프로필 사용 사례를 고려하세요. 먼저 "직원"이라는 제목이 붙은 컬렉션에 몇 개의 항목을 삽입합니다. 다음을 입력합니다.

```
db.profiles.insertMany([ { "_id": 1, "name": "Matt", "status": "active", "level":
 12, "score": 202 },
     { "_id": 2, "name": "Frank", "status": "inactive", "level": 2, "score": 9 },
     { "_id": 3, "name": "Karen", "status": "active", "level": 7, "score": 87 },
     { "_id": 4, "name": "Katie", "status": "active", "level": 3, "score": 27 }
])
```

다음 결과가 표시됩니다:

```
{ acknowledged: true,
     insertedIds: {
        '0': ObjectId('679d02cd6b5a0581be78bcbd'),
        '1': ObjectId('679d02cd6b5a0581be78bcbe'),
        '2': ObjectId('679d02cd6b5a0581be78bcbf'),
        '3': ObjectId('679d02cd6b5a0581be78bcc0')
    }
}
```

5. find() 명령을 사용하여 프로필 컬렉션의 모든 문서를 반환합니다.

```
db.Employee.find()
```

4단계에서 입력한 데이터가 표시됩니다.

6. 단일 문서를 쿼리하려면 필터(예: "Katie")를 포함하세요. 다음을 입력합니다.

```
db.Employee.find({name: "Katie"})
```

다음 결과가 표시됩니다:

```
[

   {

     _id: ObjectId('679d02cd6b5a0581be78bcc0'),
```

```
    Employeeid: 4,

    name: 'Katie',

    lastname: 'Schaper',

    level: 3

  }

]
```

7. 프로필을 찾아 수정하려면 findAndModify 명령을 입력합니다. 이 예에서는 직원 "Matt"에게 더 높은 레벨인 "14"가 지정되었습니다.

```
db.Employee.findAndModify({
    query: { "Employeeid" : 1, "name" : "Matt"},
    update: { "Employeeid" : 1, "name" : "Matt", "lastname" : "Winkle", "level" :
 14 }
})
```

다음 결과가 표시됩니다(레벨은 아직 변경되지 않음).

```
{

   _id: ObjectId('679d02cd6b5a0581be78bcbd'),

   Employeeid: 1,

   name: 'Matt',

   lastname: 'Winkle',

   level: 12

}
```

8. 레벨 증가를 확인하려면 다음 쿼리를 입력합니다:

```
db.Employee.find({name: "Matt"})
```

다음 결과가 표시됩니다:

```
[
   {

   _id: ObjectId('679d02cd6b5a0581be78bcbd'),

   Employeeid: 1,

   name: 'Matt',

   lastname: 'Winkle',

   level: 14

   }
]
```

4단계: 탐색

축하합니다! Amazon DocumentDB 탄력적 클러스터에 대한 시작하기 절차를 성공적으로 완료했습니다.

다음 단계? 몇 가지 인기 있는 기능을 통해 이 데이터베이스를 최대한 활용하는 방법을 알아보십시오:

- Amazon DocumentDB 탄력적 클러스터 모범 사례
- Amazon DocumentDB 탄력적 클러스터 관리

> **Note**
>
> 이 시작 절차를 통해 생성한 탄력적 클러스터는 삭제하지 않는 한 계속 비용이 누적됩니다. 지침은 엘라스틱 클러스터 삭제 섹션을 참조하세요.

Amazon DocumentDB 탄력적 클러스터 모범 사례

Amazon DocumentDB 탄력적 클러스터 작업 모범 사례에 대해서 알아봅니다. 모든 인스턴스 기반 Amazon DocumentDB 클러스터 모범 사례는 탄력적 클러스터에도 적용됩니다. 이 섹션은 새로운 모범 사례가 확인되는 대로 지속적으로 업데이트됩니다.

주제

- 샤드 키 선택
- 연결 관리
- 샤딩되지 않은 컬렉션
- 탄력적 클러스터 크기 조정
- 탄력적 클러스터 모니터링

샤드 키 선택

다음 목록은 분할 키를 만들기 위한 지침을 설명합니다.

- 균등하게 분산된 해시 키를 사용하여 클러스터의 모든 샤드에 데이터를 분산합니다(핫키는 제외).
- 분산형 수집 쿼리를 방지하려면 모든 읽기/업데이트/삭제 요청에 샤드 키를 사용하세요.
- 읽기/업데이트/삭제 작업을 수행할 때는 분할된 키가 중첩되지 않도록 하세요.
- 일괄 작업을 수행할 때 모든 샤드가 병렬로 실행되고 지연 시간을 개선할 수 있도록 `ordered`를 false로 설정하세요.

연결 관리

다음 목록은 데이터베이스 연결 관리에 대한 지침을 설명합니다.

- 연결 수와 새 연결이 열리고 닫히는 빈도를 모니터링하세요.
- 애플리케이션 구성의 모든 서브넷에 연결을 분산하세요. 클러스터가 여러 서브넷으로 구성되어 있지만 서브넷의 일부만 사용하는 경우 최대 연결 수에 병목 현상이 발생할 수 있습니다.

샤딩되지 않은 컬렉션

다음은 샤딩되지 않은 컬렉션에 대한 가이드라인을 설명합니다.

- 샤딩되지 않은 컬렉션을 사용할 때는 부하를 분산하기 위해 활용도가 높은 샤딩되지 않은 컬렉션을 여러 데이터베이스에 보관해 보세요. Amazon DocumentDB 탄력적 클러스터는 데이터베이스를 여러 샤드에 배치하고 샤딩되지 않은 동일한 데이터베이스에 대한 샤딩되지 않은 컬렉션을 동일한 샤드에 배치합니다.

탄력적 클러스터 크기 조정

다음 목록은 탄력적 클러스터를 규모 조정하기 위한 지침을 설명합니다.

- 조정 작업으로 인해 잠시 동안 데이터베이스 및 네트워크 오류가 간헐적으로 발생할 수 있습니다. 가능하면 사용량이 많은 시간대에는 규모를 조정하지 마세요. 유지 관리 기간 중에 크기 조정을 시도해 보세요.
- 컴퓨팅 성능을 높이기 위해 샤드 용량을 늘리거나 줄이는 것(샤드당 vCPU 수 변경)이 샤드 수를 늘리거나 줄이는 것보다 더 빠르고 간헐적으로 발생하는 데이터베이스 및 네트워크 오류의 지속 시간이 짧기 때문에 선호됩니다.
- 용량 증가를 예상할 때는 샤드 용량을 규모 조정하는 대신 샤드 수를 늘리는 것이 좋습니다. 이를 통해 빠르게 규모 조정해야 하는 시나리오에서 샤드 용량을 늘려 클러스터를 규모 조정할 수 있습니다.
- 클라이언트측 재시도 정책을 모니터링하고 지수 백오프 및 지터로 재시도하여 규모 조정 중에 오류가 발생할 경우 데이터베이스 과부하를 방지하세요.
- 데이터를 한 샤드에서 다른 샤드로 안전하게 이동해야 하므로 샤드 수를 변경하는 데 시간이 걸릴 수 있습니다. 오류 위험을 최소화하려면 변경 사항이 진행되는 동안 클러스터 상태를 중지하거나 다른 변경하지 마십시오.

탄력적 클러스터 모니터링

다음 목록은 탄력적 클러스터를 모니터링하기 위한 지침을 설명합니다.

- 샤드당 지표의 피크 대 평균 비율을 추적하여 고르지 않은 트래픽(핫키/핫스팟이 있음)을 유도하고 있는지 확인하세요. 피크 대 평균 비율을 추적하는 주요 지표는 다음과 같습니다.
  - `PrimaryInstanceCPUUtilization`
    - 이는 샤드별 수준에서 모니터링할 수 있습니다.
    - 클러스터 수준에서 평균 p99 스큐를 모니터링할 수 있습니다.
  - `PrimaryInstanceFreeableMemory`
    - 이는 샤드별 수준에서 모니터링할 수 있습니다.
    - 클러스터 수준에서 평균 p99 스큐를 모니터링할 수 있습니다.
  - `DatabaseCursorsMax`
    - 샤드별 수준에서 모니터링하여 스큐를 확인해야 합니다.
  - `Documents-Inserted/Updated/Returned/Deleted`
    - 샤드별 수준에서 모니터링하여 스큐를 확인해야 합니다.

Amazon DocumentDB 탄력적 클러스터 관리

Amazon DocumentDB 엘라스틱 클러스터를 관리하려면 적절한 Amazon DocumentDB 컨트롤 플레인 권한이 있는 IAM 정책이 있어야 합니다. 이러한 권한은 클러스터를 생성, 수정 및 삭제하도록 허용합니다. Amazon DocumentDBFullAccess 정책은 Amazon DocumentDB 엘라스틱 클러스터를 관리하는 데 필요한 모든 권한을 제공합니다.

다음 항목에서는 Amazon DocumentDB 엘라스틱 클러스터로 작업할 때 다양한 작업을 수행하는 방법을 보여줍니다.

주제

- 엘라스틱 클러스터 구성 수정
- 엘라스틱 클러스터 모니터링
- 엘라스틱 클러스터 삭제
- 엘라스틱 클러스터 스냅샷 관리
- Amazon DocumentDB 탄력적 클러스터 중지 및 시작
- Amazon DocumentDB 탄력적 클러스터 유지 관리

엘라스틱 클러스터 구성 수정

이 섹션에서는 다음 지침에 따라 AWS Management Console 또는를 사용하여 탄력 AWS CLI 적 클러스터를 수정하는 방법을 설명합니다.

클러스터를 수정하는 주된 용도는 샤드 카운트 및/또는 샤드 컴퓨팅 용량을 늘리거나 줄임으로써 샤드를 확장하는 것입니다.

Using the AWS Management Console

를 사용하여 탄력적 클러스터 구성을 수정하려면 AWS Management Console:

1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 보이지 않으면 탐색 창의 왼쪽 상단 모서리에 있는 메뉴 아이콘을 선택합니다.

3. 클러스터 식별자 열에서 수정할 클러스터의 이름을 선택합니다.
4. 수정을 선택합니다.
5. 변경하려는 필드를 편집한 다음 클러스터 수정을 선택합니다.

Configuration

Cluster identifier

SampleCluster

Shard count

Number of shards the Elastic Cluster will use.

2

Shard instance count

Number of instances for each shard. All instances will have the same shard capacity.

2

Shard capacity

vCPU capacity of each shard.

2

Maintenance

Maintenance window

The period in which pending modifications or patches are applied to your Elastic cluster.

- ○ Select window
- ◉ No preference

Authentication

Username

SampleUser

New password

Confirm new password

Password must be at least eight characters long and cannot contain a / (slash), " (double quote) or @ (at symbol).

Network settings

Subnets

Select either 0 or 2-6 subnets

subnet-0b2962f92a0f5a8fb ✕ subnet-08c6d849efd4dfe96 ✕

VPC security groups

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic.

Select VPC security groups

sg-021d234a0a3a2c2fe ✕ sg-0824c1992d6009b52 ✕

> **Note**
>
> 또는 클러스터 페이지로 이동하여 클러스터 옆의 확인란을 선택한 다음 작업, 수정을 선택하여 클러스터 수정 대화 상자에 액세스할 수 있습니다.

Using the AWS CLI

를 사용하여 탄력적 클러스터 구성을 수정하려면 다음 파라미터와 함께 `update-cluster` 작업을 AWS CLI사용합니다.

- **`--cluster-arn`** - 필수입니다. 수정하려는 클러스터의 ARN 식별자입니다.
- **`--shard-capacity`** - 선택 사항. 각 샤드에 할당된 vCPU 개수입니다. 최대값은 64입니다. 허용되는 값은 2, 4, 8, 16, 32, 64입니다.
- **`--shard-count`** - 선택 사항. 클러스터에 할당된 샤드 개수입니다. 최대 값은 32입니다.
- **`--shard-instance`**-수 - 선택 사항입니다. 이 클러스터의 모든 샤드에 적용되는 복제본 인스턴스 수입니다. 최대 값은 16입니다.
- **`--auth-type`** - 선택 사항. 엘라스틱 클러스터에 액세스하는 데 사용되는 암호를 가져올 위치를 결정하는 데 사용되는 인증 유형입니다. 유효한 형식은 `PLAIN_TEXT` 또는 `SECRET_ARN`입니다.
- **`--admin-user-password`** - 선택 사항. 관리자 사용자와 연결된 암호입니다.
- **`--vpc-security-group-ids`** - 선택 사항. 클러스터와 연결할 Amazon EC2 및 Amazon Virtual Private Cloud(Amazon VPC) 보안 그룹의 목록을 구성합니다.
- **`--preferred-maintenance-window`** - 선택 사항. 시스템 유지보수가 수행될 수 있는 주간 시간 범위를 UTC(Universal Coordinated Time)로 구성합니다

  형식은 `ddd:hh24:mi-ddd:hh24:mi`입니다. 유효한 요일(ddd): 월, 화, 수, 목, 금, 토, 일

  기본값은 각 Amazon Web Services Region(아마존 웹 서비스 리전)의 8시간 블록에서 임의로 선택된 30분 창으로, 주 마다 임의의 요일에 발생합니다.

  최소 30분의 시간.
- **`--subnet-ids`** - 선택 사항. 네트워크 서브넷 ID를 구성합니다.

다음 예제에서는 자신의 정보로 각각의 *`### ## ## ###`*를 바꿉니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb-elastic update-cluster \
    --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 \
    --shard-capacity 8 \
    --shard-count 4 \
    --shard-instance-count 3 \
    --admin-user-password testPassword \
    --vpc-security-group-ids ec-65f40350 \
    --subnet-ids subnet-9253c6a3, subnet-9f1b5af9
```

Windows의 경우:

```
aws docdb-elastic update-cluster ^
    --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 ^
    --shard-capacity 8 ^
    --shard-count 4 ^
    --shard-instance-count 3 ^
    --admin-user-password testPassword ^
    --vpc-security-group-ids ec-65f40350 ^
    --subnet-ids subnet-9253c6a3, subnet-9f1b5af9
```

수정 후 엘라스틱 클러스터의 상태를 모니터링하려면 엘라스틱 클러스터 모니터링을 참조하세요.

엘라스틱 클러스터 모니터링

이 섹션에서는 다음 지침에 따라 AWS Management Console 또는를 사용하여 탄력 AWS CLI 적 클러스터를 모니터링하는 방법을 설명합니다.

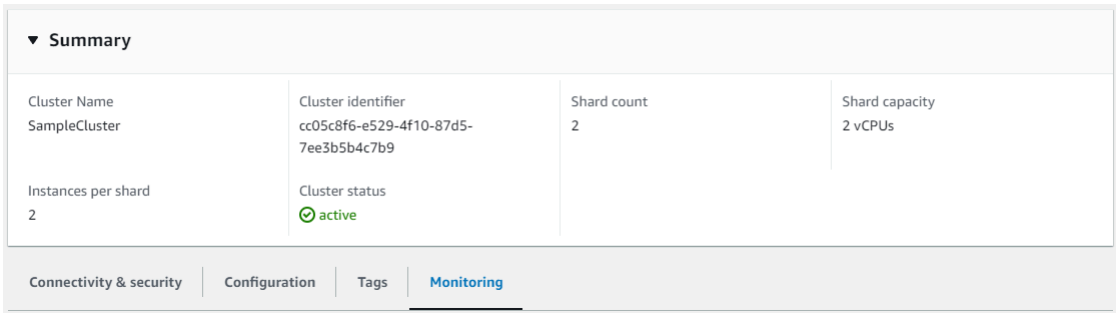
Using the AWS Management Console

를 사용하여 탄력적 클러스터 구성을 모니터링하려면 AWS Management Console:

1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 보이지 않으면 탐색 창의 왼쪽 상단 모서리에 있는 메뉴 아이콘을 선택합니다.

3. 클러스터 식별자 열에서 모니터링할 클러스터의 이름을 선택합니다.
4. 모니터링 탭을 선택합니다.



Amazon CloudWatch의 여러 차트가 다음과 같은 모니터링 카테고리에 대해 표시됩니다.

- 리소스 사용률
- 처리량
- 운영
- 시스템

를 통해 Amazon CloudWatch에 액세스 AWS Management Console 하여 탄력적 클러스터에 대한 자체 모니터링 환경을 설정할 수도 있습니다.

Using the AWS CLI

를 사용하여 특정 탄력적 클러스터 구성을 모니터링하려면 다음 파라미터와 함께 `get-cluster` 작업을 AWS CLI사용합니다.

- **`--cluster-arn`** - 필수입니다. 정보를 원하는 클러스터의 ARN 식별자입니다.

다음 예제에서는 자신의 정보로 각각의 *### ## ## ###*를 바꿉니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb-elastic get-cluster \
    --cluster-arn arn:aws:docdb-elastic:us-west-2:123456789012:cluster:/68ffcdf8-e3af-40a3-91e4-24736f2dacc9
```

Windows의 경우:

```
aws docdb-elastic get-cluster ^
    --cluster-arn arn:aws:docdb:-elastic:us-west-2:123456789012:cluster:/68ffcdf8-e3af-40a3-91e4-24736f2dacc9
```

이 작업의 출력은 다음과 같이 표시됩니다.

```
"cluster": {
        ...
        "clusterArn": "arn:aws:docdb-elastic:us-west-2:123456789012:cluster:/68ffcdf8-e3af-40a3-91e4-24736f2dacc9",
        "clusterEndpoint": "stretch-11-477568257630.us-east-1.docdb-elastic.amazonaws.com",
        "readerEndpoint": "stretch-11-477568257630-ro.us-east-1.docdb-elastic.amazonaws.com",
        "clusterName": "stretch-11",
        "shardCapacity": 2,
        "shardCount": 3,
        "shardInstanceCount: 5,
        "status": "ACTIVE",
        ...
 }
```

자세한 내용은 Amazon DocumentDB 리소스 관리 API 참조에서 `DescribeClusterSnapshot` 섹션을 참조하세요.

를 사용하여 모든 탄력적 클러스터의 세부 정보를 보려면 다음 파라미터와 함께 `list-clusters` 작업을 AWS CLI사용합니다.

- **`--next-token`** - 선택 사항. 출력 항목 수(`--max-results`)가 기본 API 호출에서 반환하는 전체 항목 수보다 적을 경우 사용자가 다음 항목 세트를 검색하기 위해 후속 명령에 전달할 수 있도록 출력에 `NextToken`이 포함됩니다.
- **`--max-results`** - 선택 사항. 명령의 출력에서 반환되는 항목의 총 수입니다. 지정된 `max-results` 값보다 레코드 수가 많으면 페이지 매김 토큰(`next-token`)을 응답에 포함시켜 나머지 결과를 검색할 수 있도록 합니다.
  - 기본값: 100
  - 최소: 20 최대: 100

다음 예제에서는 자신의 정보로 각각의 *### ## ## ###*를 바꿉니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb-elastic list-clusters \
    --next-token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ== \
    --max-results 2
```

Windows의 경우:

```
aws docdb-elastic list-clusters ^
    --next-token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ== ^
    --max-results 2
```

이 작업의 출력은 다음과 같습니다.

```
{
   "Clusters": [
      {
         "ClusterIdentifier":"mycluster-1",
         "ClusterArn":"arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster"
         "Status":"available",
         "ClusterEndpoint":"sample-cluster.sharded-cluster-corcjozrlsfc.us-
west-2.docdb.amazonaws.com"
       }
       {
         "ClusterIdentifier":"mycluster-2",
         "ClusterArn":"arn:aws:docdb:us-west-2:987654321098:sharded-cluster:sample-
cluster"
         "Status":"available",
         "ClusterEndpoint":"sample-cluster2.sharded-cluster-corcjozrlsfc.us-
west-2.docdb.amazonaws.com"
       }
   ]
}
```

엘라스틱 클러스터 삭제

이 섹션에서는 AWS CLI 다음 지침에 따라 AWS Management Console 또는를 사용하여 탄력적 클러스터를 삭제하는 방법을 설명합니다.

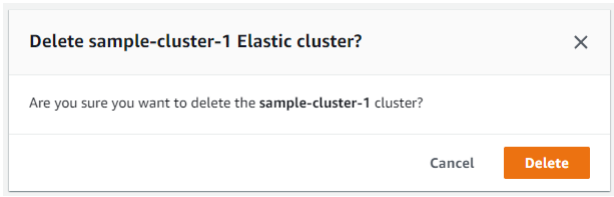
Using the AWS Management Console

AWS Management Console을 사용하여 엘라스틱 클러스터 구성을 삭제하려면:

1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 보이지 않으면 탐색 창의 왼쪽 상단 모서리에 있는 메뉴 아이콘을 선택합니다.

3. 클러스터 목록 테이블에서 삭제할 클러스터 이름의 왼쪽에 있는 확인란을 선택한 다음 작업을 선택합니다. 드롭다운 메뉴에서 삭제를 선택합니다.
4. "클러스터 이름" 엘라스틱 클러스터를 삭제하시겠습니까? 대화 상자에서 삭제를 선택합니다.



클러스터를 삭제하는 데 몇 분 정도 걸립니다. 클러스터 상태를 모니터링하려면 Amazon DocumentDB 클러스터 상태 모니터링 섹션을 참조하세요.

Using the AWS CLI

를 사용하여 엘라스틱 클러스터를 삭제하려면 다음 파라미터와 함께 `delete-cluster` 작업을 AWS CLI사용합니다.

- **`--cluster-arn`** - 필수입니다. 삭제하려는 클러스터의 ARN 식별자입니다.
- **`--no-skip-final-backup`** - 선택 사항. 최종 백업을 원하는 경우, 최종 백업의 이름과 함께 이 파라미터를 포함해야 합니다. `--final-backup-identifier` 또는 `--skip-final-backup`를 포함시켜야 합니다.
- **`--skip-final-backup`** - 선택 사항. 클러스터를 삭제하기 전에 최종 백업을 수행하지 않을 경우에만 이 파라미터를 사용합니다. 기본 설정은 최종 스냅샷을 생성하는 것입니다.

다음 AWS CLI 코드 예제에서는 ARN이 arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-cluster이고 최종 백업이 있는 클러스터를 삭제합니다.

다음 예제에서는 자신의 정보로 각각의 *### ## ## ###*를 바꿉니다.

Linux, macOS, Unix의 경우:

```
aws docdb-elastic delete-cluster \
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster \
    --no-skip-final-backup \
    --final-backup-identifier finalArnBU-arn:aws:docdb:us-
west-2:123456789012:sharded-cluster:sample-cluster
```

Windows의 경우:

```
aws docdb-elastic delete-cluster ^
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster ^
    --no-skip-final-backup ^
    --final-backup-identifier finalArnBU-arn:aws:docdb:us-
west-2:123456789012:sharded-cluster:sample-cluster
```

다음 AWS CLI 코드 예제에서는 ARN이 arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-cluster인 클러스터를 최종 백업 없이 삭제합니다.

다음 예제에서는 자신의 정보로 각각의 *### ## ## ###*를 바꿉니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb-elastic delete-cluster \
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster \
    --skip-final-backup \
```

Windows의 경우:

```
aws docdb-elastic delete-cluster ^
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster ^
    --skip-final-backup ^
```

`delete-cluster` 작업의 출력은 삭제할 클러스터의 표시입니다.

클러스터를 삭제하는 데 몇 분 정도 걸립니다. 클러스터 상태를 모니터링하려면 Amazon DocumentDB 클러스터 상태 모니터링 섹션을 참조하세요.

엘라스틱 클러스터 스냅샷 관리

수동 스냅샷은 엘라스틱 클러스터를 생성한 후에 만들 수 있습니다. 자동 백업은 탄력적 클러스터 스냅샷이 생성되는 즉시 생성됩니다.

> **Note**
>
> 수동 스냅샷을 생성하려면 엘라스틱 클러스터가 `Available` 상태에 있어야 합니다.

이 섹션에서는 엘라스틱 클러스터 스냅샷을 생성하고, 보고, 복원하고, 삭제하는 방법을 설명합니다.

다음 항목에서는 Amazon DocumentDB 엘라스틱 클러스터 스냅샷으로 작업할 때 다양한 작업을 수행하는 방법을 보여줍니다.

주제

- 수동 클러스터 스냅샷 생성
- 엘라스틱 클러스터 스냅샷 보기
- 스냅샷에서 엘라스틱 클러스터 복원
- 탄력적 클러스터 스냅샷 보기
- 탄성 클러스터 스냅샷 삭제
- 탄력적 클러스터 스냅샷 자동 백업 관리

수동 클러스터 스냅샷 생성

이 섹션에서는 다음 지침과 AWS CLI 함께 AWS Management Console 또는를 사용하여 수동 탄력적 클러스터 스냅샷을 생성하는 방법을 설명합니다.

Using the AWS Management Console

AWS Management Console을 사용하여 수동 엘라스틱 클러스터 스냅샷을 만들려면:

1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.

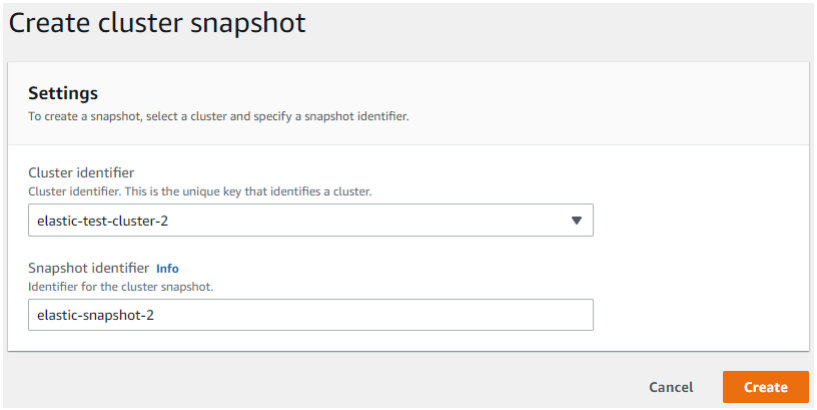
2. 탐색 창에서 스냅샷을 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 보이지 않으면 탐색 창의 왼쪽 상단 모서리에 있는 메뉴 아이콘을 선택합니다.

3. 스냅샷 페이지에서 생성을 선택합니다.
4. 클러스터 스냅샷 생성 페이지의 클러스터 식별자 필드에 있는 드롭다운 목록에서 탄력적 클러스터를 선택합니다.

   스냅샷 식별자 필드에 엘라스틱 클러스터의 고유 식별자를 입력합니다.

   생성(Create)을 선택합니다.



> **Note**
>
> 또는 클러스터 페이지로 이동하여 클러스터 옆의 확인란을 선택한 다음 작업, 스냅샷 찍기를 선택하여 클러스터 스냅샷 생성 대화 상자에 액세스할 수 있습니다.

이제 엘라스틱 클러스터 스냅샷이 프로비저닝 중입니다. 이 과정은 완료하는 데 최대 수 분 소요될 수 있습니다. 스냅샷 목록에 상태가 `Available`로 표시될 때 스냅샷을 보고 복원할 수 있습니다.

Using the AWS CLI

를 사용하여 수동 탄력적 클러스터 스냅샷을 생성하려면 다음 파라미터와 함께 `create-cluster-snapshot` 작업을 AWS CLI사용합니다.

- **`--snapshot-name`** - 필수입니다. 생성할 클러스터 스냅샷의 이름입니다.

- **`--cluster-arn`** - 필수입니다. 스냅샷을 생성하고자 하는 클러스터의 ARN 식별자입니다.

다음 예제에서는 자신의 정보로 각각의 *### ## ## ###*를 바꿉니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb-elastic create-cluster-snapshot \
    --snapshot-name sample-snapshot-1 \
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster
```

Windows의 경우:

```
aws docdb-elastic create-cluster-snapshot ^
    --snapshot-name sample-snapshot-1 ^
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster
```

엘라스틱 클러스터 스냅샷 보기

이 섹션에서는 다음 지침에 따라 AWS Management Console 또는를 사용하여 탄력 AWS CLI 적 클러스터 스냅샷 정보를 보는 방법을 설명합니다.

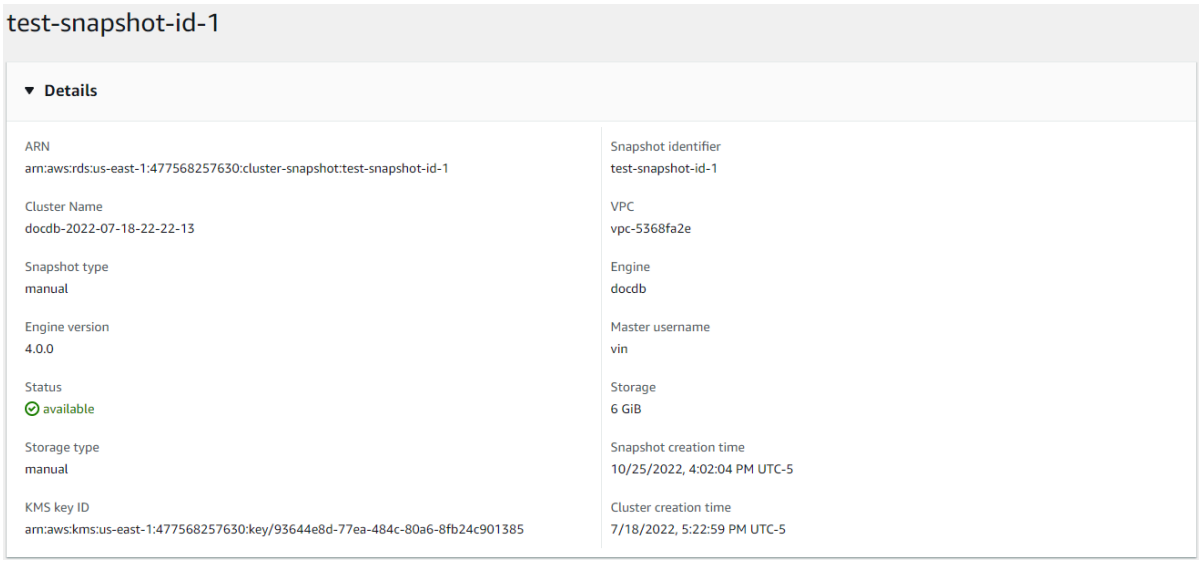
Using the AWS Management Console

를 사용하여 특정 탄력적 클러스터 스냅샷에 대한 정보를 보려면 AWS Management Console:

1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 스냅샷을 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 보이지 않으면 탐색 창의 왼쪽 상단 모서리에 있는 메뉴 아이콘을 선택합니다.

3. 스냅샷 페이지에서 스냅샷 식별자 열에 있는 이름을 클릭하여 목록에서 스냅샷을 선택합니다.
4. 세부 정보에서 스냅샷 정보를 확인하세요.



Using the AWS CLI

를 사용하여 특정 탄력적 클러스터 스냅샷에 대한 정보를 보려면 다음 파라미터와 함께 `get-cluster-snapshot` 작업을 AWS CLI사용합니다.

- **`--snapshot-arn`** - 필수입니다. 정보를 원하는 스냅샷의 ARN 식별자입니다.

다음 예제에서는 자신의 정보로 각각의 *### ## ## ###*를 바꿉니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb-elastic get-cluster-snapshot \
    --snapshot-arn sampleResourceName
```

Windows의 경우:

```
aws docdb-elastic get-cluster-snapshot ^
    --snapshot-arn sampleResourceName
```

를 사용하여 특정 탄력적 클러스터 스냅샷에 대한 정보를 보려면 다음 파라미터와 함께 `get-cluster-snapshot` 작업을 AWS CLI사용합니다.

- **`--snapshot-arn`** - 필수입니다. 정보를 원하는 스냅샷의 ARN 식별자입니다.

다음 예제에서는 자신의 정보로 각각의 *### ## ## ###*를 바꿉니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb-elastic get-cluster-snapshot \
    --snapshot-arn sampleResourceName
```

Windows의 경우:

```
aws docdb-elastic get-cluster-snapshot ^
    --snapshot-arn sampleResourceName
```

를 사용하여 모든 탄력적 클러스터 스냅샷에 대한 정보를 보려면 다음 파라미터와 함께 `list-cluster-snapshots` 작업을 AWS CLI사용합니다.

- **`--snapshot-type`** - 선택 사항. 반환되는 클러스터 스냅샷의 유형입니다. 다음 값 중 하나를 지정할 수 있습니다.
  - `automated` - Amazon DocumentDB가 AWS 계정에 대해 자동으로 생성한 모든 클러스터 스냅샷을 반환합니다.
  - `manual` - AWS 계정에 대해 수동으로 생성한 모든 클러스터 스냅샷을 반환합니다.
  - `shared` - AWS 계정에 공유된 모든 수동 클러스터 스냅샷을 반환합니다.
  - `public` - 퍼블릭으로 표시된 모든 클러스터 스냅샷을 모두 반환합니다.
- **`--next-token`** - 선택 사항. 이전의 요청에서 제공된 선택적 페이지 매김 토큰입니다. 이 파라미터를 지정한 경우, 이 토큰 이후부터 `max-results`에 지정된 값까지의 레코드만 응답에 포함됩니다.
- **`--max-results`** - 선택 사항. 응답에 포함되는 최대 결과 수입니다. 지정된 `max-results` 값보다 레코드 수가 많으면 페이지 매김 토큰(`next-token`)을 응답에 포함시켜 나머지 결과를 검색할 수 있도록 합니다.
  - 기본값: 100
  - 최소: 20 최대: 100

다음 예제에서는 자신의 정보로 각각의 *### ## ## ###*를 바꿉니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb-elastic list-cluster-snapshots \
```

```
    --snapshot-type  value \
    --next-token  value \
    --max-results 50
```

Windows의 경우:

```
aws docdb-elastic list-cluster-snapshots ^
    --snapshot-type  value ^
    --next-token  value ^
    --max-results 50
```

스냅샷에서 엘라스틱 클러스터 복원

이 섹션에서는 AWS CLI 다음 지침에 따라 AWS Management Console 또는를 사용하여 스냅샷에서 탄력적 클러스터를 복원하는 방법을 설명합니다.

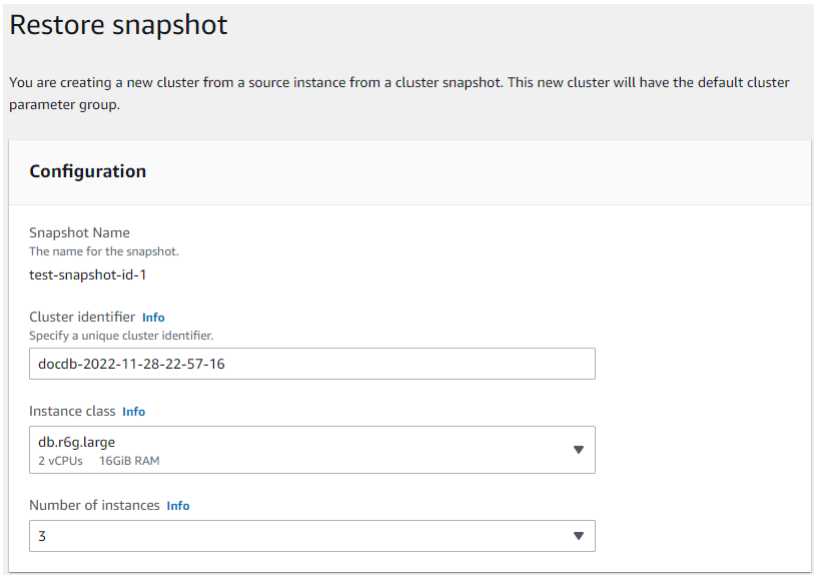
Using the AWS Management Console

AWS Management Console를 사용하여 스냅샷에서 엘라스틱 클러스터 복원:

1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 스냅샷을 선택합니다.

   > **Tip**
   >
   > 화면 왼쪽에 탐색 창이 보이지 않으면 탐색 창의 왼쪽 상단 모서리에 있는 메뉴 아이콘을 선택합니다.

3. 스냅샷 식별자 열에서 클러스터 복원에 사용하려는 스냅샷 왼쪽에 있는 버튼을 선택합니다.
4. 작업을 선택한 다음, 복원을 선택합니다.



5. 스냅샷 복원 페이지의 클러스터 식별자 필드에 새 클러스터의 이름을 입력합니다.

> **Note**
>
> 수동 스냅샷 복원의 경우 새 클러스터를 생성해야 합니다.

6. 가상 프라이빗 클라우드(VPC) 필드에서 드롭다운 목록 중 VPC를 선택합니다.
7. 서브넷 및 VPC 보안 그룹의 경우 기본값을 사용하거나 원하는 서브넷 3개와 VPC 보안 그룹 최대 3개 (최소 1개) 를 선택할 수 있습니다.
8. 클러스터 구성이 마음에 들면 클러스터 복원을 선택하고 클러스터가 복원되는 동안 기다립니다.

Using the AWS CLI

를 사용하여 스냅샷에서 탄력적 클러스터를 복원하려면 다음 파라미터와 함께 `restore-cluster-from-snapshot` 작업을 AWS CLI사용합니다.

- **`--cluster-name`** - 필수입니다. 생성시 입력했거나 마지막으로 수정한 엘라스틱 클러스터의 현재 이름.
- **`--snapshot-arn`** - 필수입니다. 클러스터를 복원하는 데 사용되는 스냅샷의 ARN 식별자입니다.
- **`--vpc-security-group-ids`** - 선택 사항. 클러스터와 연결할 Amazon EC2 및 and Amazon Virtual Private Cloud(Amazon VPC) 보안 그룹의 목록을 구성합니다.
- **`--kms-key-id`** - 선택 사항. 암호화된 클러스터의 KMS 키 식별자를 구성하세요.

KMS 키 식별자는 AWS KMS 암호화 키의 Amazon 리소스 이름(ARN)입니다. 새 클러스터를 암호화하는 데 사용되는 KMS 암호화 키를 소유한 동일한 Amazon Web Services 계정을 사용하여 클러스터를 생성하는 경우, KMS 암호화 키에 대한 ARN 대신 KMS 키 별칭을 사용할 수 있습니다.

암호화 키가 KmsKeyId에 지정되어 있지 않고 `StorageEncrypted` 파라미터가 참이면 Amazon DocumentDB는 기본 암호화 키를 사용합니다.

- **`--subnet-ids`** - 선택 사항. 네트워크 서브넷 ID.

다음 예제에서는 자신의 정보로 각각의 *### ## ## ###*를 바꿉니다.

Linux, macOS, Unix의 경우:

```
aws docdb-elastic restore-cluster-from-snapshot \
     --cluster-name elastic-sample-cluster \
     --snapshot-arn sampleResourceName \
     --vpc-security-group-ids value ec-65f40350 \
     --kms-key-id arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 \
     --subnet-ids subnet-9253c6a3, subnet-9f1b5af9
```

Windows의 경우:

```
aws docdb-elastic restore-cluster-from-snapshot ^
     --cluster-name elastic-sample-cluster ^
     --snapshot-arn sampleResourceName ^
     --vpc-security-group-ids value ec-65f40350 ^
     --kms-key-id arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2 ^
     --subnet-ids subnet-9253c6a3, subnet-9f1b5af9
```

탄력적 클러스터 스냅샷 보기

Amazon DocumentDB에서는 동일한 리전과 동일한 계정 내에서 수동 및 자동 탄력적 클러스터 스냅샷을 복사할 수 있습니다. 이 섹션에서는 AWS Management Console 또는를 사용하여 탄력적 클러스터 스냅샷을 복사하는 방법을 설명합니다 AWS CLI.

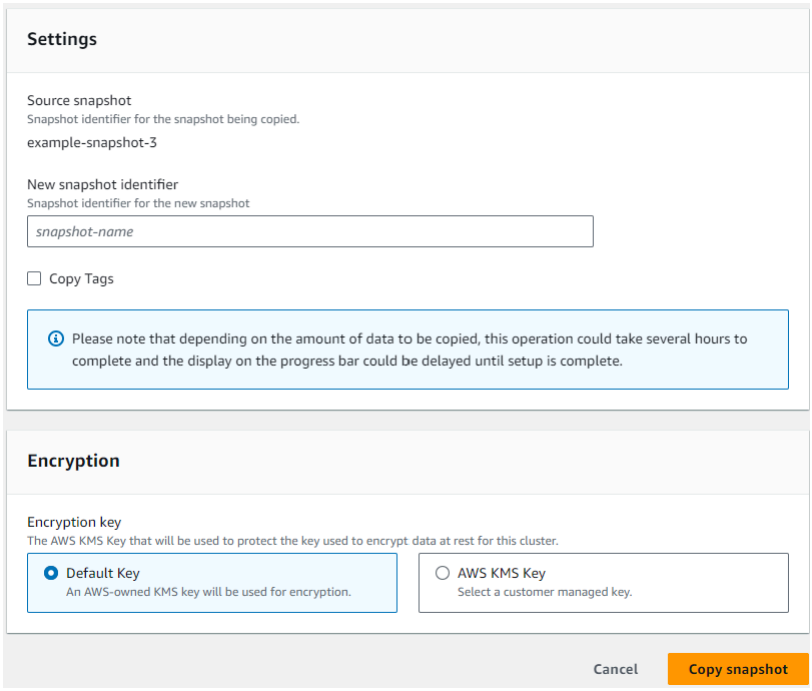
Using the AWS Management Console

를 사용하여 탄력적 클러스터 스냅샷을 복사하려면 AWS Management Console:

1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 스냅샷을 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 보이지 않으면 탐색 창의 왼쪽 상단 모서리에 있는 메뉴 아이콘을 선택합니다.

3. 스냅샷 식별자 열에서 복사하려는 스냅샷 왼쪽에 있는 버튼을 선택합니다.
4. 작업, 복사 순으로 선택합니다.



5. 새 스냅샷 식별자의 경우 새 스냅샷의 이름을 입력합니다.
6. 태그 복사에서 소스 탄력적 클러스터 스냅샷의 모든 태그를 대상 탄력적 클러스터 스냅샷으로 복사하려면 확인란을 선택합니다.
7. 암호화에서 기본 AWS KMS 키 또는 원하는 KMS 키를 선택합니다. 두 번째 옵션을 사용하면 이미 생성한 기존 KMS 키를 선택하거나 새 키를 생성할 수 있습니다.
8. 완료되면 스냅샷 복사를 선택합니다.

Using the AWS CLI

를 사용하여 탄력적 클러스터 스냅샷을 복사하려면 다음 파라미터와 함께 `copy-cluster-snapshot` 작업을 AWS CLI사용합니다.

- **`--source-db-cluster-snapshot-identifier`** - 필수입니다. 복사되는 기존 탄력적 클러스터 스냅샷의 식별자. 탄력적 클러스터 스냅샷이 있어야 하며 사용 가능한 상태여야 합니다. 이 파라미터는 대소문자를 구분하지 않습니다.
- **`--target-db-cluster-snapshot-identifier`** - 필수입니다. 기존의 탄력적 클러스터 스냅샷에서 생성할 새 탄력적 클러스터 스냅샷의 식별자입니다. 이 파라미터는 대소문자를 구분하지 않습니다.

  대상 스냅샷 명명 제약 조건:

  - 기본 스냅샷의 이름일 수 없습니다.
  - 길이는 [1~63]글자, 숫자 또는 하이픈입니다.
  - 첫 번째 문자는 글자이어야 합니다.
  - 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.

다음 예제에서는 자신의 정보로 각각의 *### ## ## ###*를 바꿉니다.

Linux, macOS 또는 Unix의 경우:

```
aws docdb-elastic copy-cluster-snapshot \
     --source-cluster-snapshot-arn <sample ARN> \
     --target-cluster-snapshot-name my-target-copied-snapshot
```

Windows의 경우:

```
aws docdb-elastic copy-cluster-snapshot ^
     --source-cluster-snapshot-arn <sample ARN> ^
     --target-cluster-snapshot-name my-target-copied-snapshot
```

탄성 클러스터 스냅샷 삭제

이 섹션에서는 AWS Management Console 또는를 사용하여 탄력적 클러스터 스냅샷을 삭제하는 방법을 설명합니다 AWS CLI.

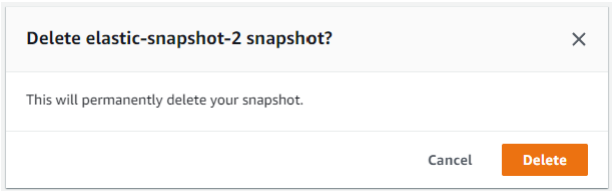
Using the AWS Management Console

AWS Management Console를 사용하여 스냅샷에서 엘라스틱 클러스터 복원:

1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 스냅샷을 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 보이지 않으면 탐색 창의 왼쪽 상단 모서리에 있는 메뉴 아이콘을 선택합니다.

3. 스냅샷 식별자 열에서 클러스터 복원에 사용하려는 스냅샷 왼쪽에 있는 버튼을 선택합니다.
4. 작업을 선택한 다음 삭제를 선택합니다.



5. "스냅샷 이름" 스냅샷 삭제 대화 상자에서 삭제를 선택합니다.

Using the AWS CLI

를 사용하여 탄력적 클러스터 스냅샷을 삭제하려면 다음 파라미터와 함께 `delete-cluster-snapshot` 작업을 AWS CLI사용합니다.

- **`--snapshot-arn`** - 필수입니다. 클러스터를 복원하는 데 사용되는 스냅샷의 ARN 식별자입니다.

다음 예제에서는 자신의 정보로 각각의 *`### ## ## ###`*를 바꿉니다.

Linux, macOS, Unix의 경우:

```
aws docdb-elastic delete-cluster-snapshot \
     --snapshot-arn sampleResourceName
```

Windows의 경우:

```
aws docdb-elastic delete-cluster-snapshot ^
```

```
    --snapshot-arn sampleResourceName
```

탄력적 클러스터 스냅샷 자동 백업 관리

Amazon DocumentDB는 탄력적 클러스터의 일일 스냅샷을 생성합니다. 새 탄력적 클러스터 또는 기존 탄력적 클러스터 스냅샷 구성에서 기본 백업 기간과 백업 보존 기간을 지정할 수 있습니다. 이 섹션에서는 AWS Management Console 또는를 사용하여 탄력적 클러스터 스냅샷에서 자동 백업 파라미터를 설정하는 방법을 설명합니다 AWS CLI.

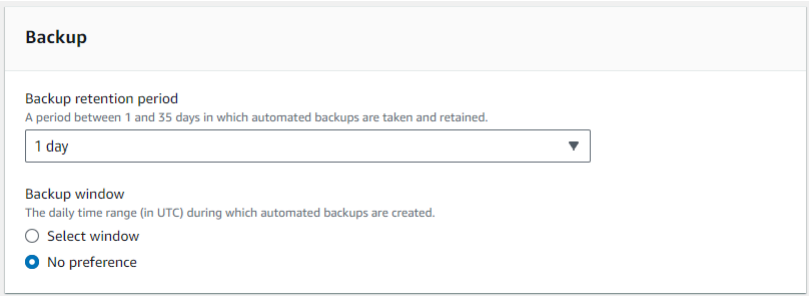
Using the AWS Management Console

를 사용하여 새 탄력적 클러스터 스냅샷에 대한 자동 백업을 설정하려면 AWS Management Console:

1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 보이지 않으면 탐색 창의 왼쪽 상단 모서리에 있는 메뉴 아이콘을 선택합니다.

3. 클러스터 식별자 열에서 백업 설정을 변경할 클러스터 왼쪽의 버튼을 선택합니다.
4. 작업(Actions), 수정(Modify)을 선택합니다.
5. 백업 섹션에서 백업 요구 사항에 따라 필드를 편집합니다.



   a. 백업 보존 기간 - 목록에서 이 클러스터의 자동 백업을 삭제하지 않고 유지하는 일 수를 선택합니다.

   b. 백업 기간 - Amazon DocumentDB에서 이 클러스터를 백업하는 일별 시간과 기간을 설정합니다.

i. 백업이 생성되는 시간과 기간을 구성하려면 창 선택을 선택합니다.

시작 시간 - 첫 번째 목록에서 자동 백업 시작 시간(UTC)을 선택합니다. 두 번째 목록에서 자동 백업 시작 시간의 분을 선택합니다.

기간 - 목록에서 자동 백업 생성에 할당된 시간 수를 선택합니다.

ii. Amazon DocumentDB가 백업이 생성되는 시간과 기간을 선택하도록 하려면 기본 설정 없음을 선택합니다.

6. 완료되면 클러스터 수정을 선택합니다.

Using the AWS CLI

를 사용하여 새 탄력적 클러스터 스냅샷에 대한 자동 백업을 설정하려면 다음 파라미터와 함께 `create-cluster-snapshot` 작업을 AWS CLI사용합니다.

- **`--preferred-backup-window`** - 선택 사항. 자동 백업이 생성되는 일일 선호 시간 범위입니다. 기본값은 각각에 대해 8시간 블록 시간 중에서 임의로 선택한 30분 기간입니다 AWS 리전.

  제약 조건:

  - `hh24:mi-hh24:mi` 형식이어야 합니다.
  - 협정 세계시(UTC)여야 합니다.
  - 원하는 유지 관리 기간과 충돌하지 않아야 합니다.
  - 30분 이상이어야 합니다.
- **`--backup-retention-period`** - 선택 사항. 자동 백업이 보존되는 일수입니다. 기본값은 1입니다.

  제약 조건:

  - 최소 1 이상의 값을 지정해야 합니다.
  - 범위는 1~35여야 합니다.

> **Note**
>
> 자동 백업은 클러스터가 '활성' 상태인 경우에만 수행됩니다.

> **Note**
>
> `aws docdb-elastic update-cluster` 명령을 사용하여 기존 탄력적 클러스터의 `preferred-backup-window` 및 `backup-retention-period` 파라미터를 수정할 수도 있습니다.

다음은 자신의 정보를 각각의 *### ## ## ###*로 변경하는 예제입니다.

다음 `create-cluster` 예제에서는 자동 백업을 위한 보존 기간이 *7*일이고 기본 백업 기간이 *18:00~18:30 UTC*인 Amazon DocumentDB 탄력적 클러스터 *## ####*를 생성합니다.

Linux, macOS, Unix의 경우:

```
aws docdb-elastic create-cluster \
     --cluster-name sample-cluster \
     --shard-capacity 2 \
     --shard-count 2 \
     --admin-user-name SampleAdmin \
     --auth-type PLAIN_TEXT \
     --admin-user-password SamplePass123! \
     --preferred-backup-window 18:00-18:30 \
     --backup-retention-period 7
```

Windows의 경우:

```
aws docdb-elastic create-cluster ^
     --cluster-name sample-cluster ^
     --shard-capacity 2 ^
     --shard-count 2 ^
     --admin-user-name SampleAdmin ^
     --auth-type PLAIN_TEXT ^
     --admin-user-password SamplePass123! ^
     --preferred-backup-window 18:00-18:30 ^
     --backup-retention-period 7
```

Amazon DocumentDB 탄력적 클러스터 중지 및 시작

Amazon DocumentDB 탄력적 클러스터를 중지하고 시작하면 개발 및 테스트 환경 비용을 관리하는 데 도움이 됩니다. 필요하지 않을 경우 Amazon DocumentDB를 사용할 때마다 탄력적 클러스터를 생성하

고 삭제하는 대신 클러스터를 일시적으로 중지할 수 있습니다. 그런 다음 테스트를 재개할 때 다시 시작할 수 있습니다.

주제

- 탄력적 DB 클러스터의 중지 및 시작 개요
- 중지된 탄력적 클러스터에서 수행할 수 있는 작업

탄력적 DB 클러스터의 중지 및 시작 개요

Amazon DocumentDB 탄력적 클러스터가 필요하지 않은 기간에는 이 클러스터를 중지할 수 있습니다. 그런 다음 사용해야 할 때는 언제든지 클러스터를 다시 시작할 수 있습니다. 시작 및 중지를 사용하면 개발, 테스트 또는 연속 가용성을 필요로 하지 않는 유사한 활동에 사용되는 탄력적 클러스터의 설정 및 해제 프로세스가 간소화됩니다. 단일 작업으로 AWS Management Console 또는를 사용하여 탄력 AWS CLI 적 클러스터를 중지하고 시작할 수 있습니다.

탄력적 클러스터가 중지되는 동안 클러스터 스토리지 볼륨은 변경되지 않습니다. 지정된 보존 기간 내에는 스토리지, 수동 스냅샷 및 자동 백업 스토리지에 대한 비용만 청구됩니다. Amazon DocumentDB는 필요한 유지 관리 업데이트에 뒤처지지 않도록 7일 후에 탄력적 클러스터를 자동으로 시작합니다. 7일 후에 클러스터가 시작되면 탄력적 클러스터의 이용 비용이 다시 청구됩니다. 클러스터가 중지된 동안에는 쿼리하려면 클러스터가 사용 가능한 상태여야 하므로 스토리지 볼륨을 쿼리할 수 없습니다.

Amazon DocumentDB 탄력적 클러스터가 중지되면 클러스터는 어떤 식으로든 수정할 수 없습니다. 여기에는 클러스터 삭제가 포함됩니다.

Using the AWS Management Console

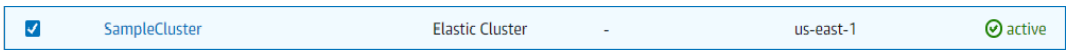
다음 절차에서는 사용 가능한 상태인 탄력적 클러스터를 중지하거나, 중지된 탄력적 클러스터를 시작하는 방법을 보여 줍니다.

Amazon DocumentDB 탄력적 클러스터 중지 및 시작 방법

1. 에 로그인 AWS Management Console하고 https://console.aws.amazon.com/docdb Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.

> **Tip**
>
> 화면 왼쪽에 탐색 창이 표시되지 않으면 페이지 왼쪽 상단 모서리에서 메뉴 아이콘(≡)을 선택합니다.

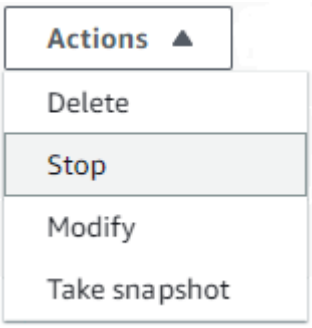
3. 클러스터 목록에서 중지하거나 시작하려는 클러스터 이름의 왼쪽에 있는 버튼을 선택합니다.



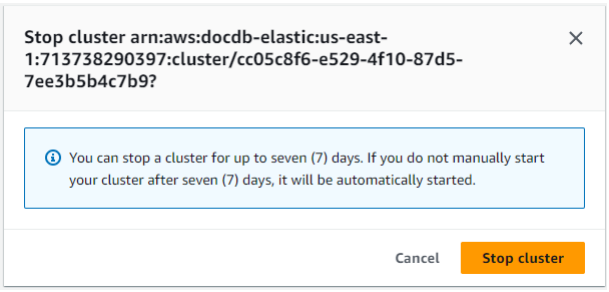
4. 작업을 선택한 다음 클러스터에서 수행하려는 작업을 선택합니다.

   - 클러스터를 중지하고 다음 작업을 수행합니다.

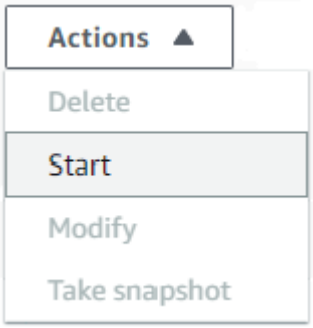
     a. 중지를 선택합니다.



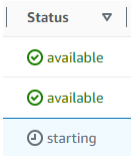
     b. 확인 대화 상자에서 클러스터 중지를 선택하여 탄력적 클러스터 중지를 확인하거나, 클러스터를 계속 실행하려면 취소를 선택합니다.



   - 클러스터를 시작하려면 클러스터가 중지된 상태에서 시작을 선택합니다.



5. Elastic 클러스터의 상태를 모니터링합니다. 클러스터를 시작한 경우 클러스터를 사용 가능할 때 클러스터 사용을 다시 시작할 수 있습니다. 자세한 내용은 클러스터 상태 확인 단원을 참조하십시오.



Using the AWS CLI

다음 절차에서는 활성 상태이거나 사용 가능한 상태인 탄력적 클러스터를 중지하거나, 중지된 탄력적 클러스터를 시작하는 방법을 보여 줍니다.

를 사용하여 엘라스틱 클러스터를 중지하려면 `stop-cluster` 작업을 AWS CLI사용합니다. 중지된 클러스터를 시작하려면 `start-cluster` 작업을 사용합니다. 두 작업은 `--cluster-arn` 파라미터를 사용합니다.

파라미터:

- **`--cluster-arn`** - 필수입니다. 중단하거나 시작하려는 탄력적 클러스터의 ARN 식별자입니다.

Example-를 사용하여 엘라스틱 클러스터를 중지하려면 AWS CLI

다음은 자신의 정보를 각각의 *### ## ## ###*로 변경하는 예제입니다.

다음 코드는 ARN이 `arn:aws:docdb-elastic:us-east-1:477568257630:cluster/b9f1d489-6c3e-4764-bb42-da62ceb7bda2`인 탄력적 클러스터를 중지합니다.

> **Note**
>
> 탄력적 클러스터가 활성 상태거나 사용 가능 상태여야 합니다.

Linux, macOS, Unix의 경우:

```
aws docdb-elastic stop-cluster \
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

Windows의 경우:

```
aws docdb-elastic stop-cluster ^
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

Example-를 사용하여 엘라스틱 클러스터를 시작하려면 AWS CLI

다음은 자신의 정보를 각각의 *### ## ## ###*로 변경하는 예제입니다.

다음 코드는 ARN이 arn:aws:docdb-elastic:us-east-1:477568257630:cluster/b9f1d489-6c3e-4764-bb42-da62ceb7bda2인 탄력적 클러스터를 시작합니다.

> **Note**
>
> 탄력적 클러스터는 현재 중지된 상태여야 합니다.

Linux, macOS, Unix의 경우:

```
aws docdb-elastic start-cluster \
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

Windows의 경우:

```
aws docdb-elastic start-cluster ^
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

중지된 탄력적 클러스터에서 수행할 수 있는 작업

클러스터가 중지되는 동안 Amazon DocumentDB 탄력적 클러스터의 구성을 수정할 수 없습니다. 이러한 관리 작업을 수행하기 전에 클러스터를 시작해야 합니다.

Amazon DocumentDB는 다시 시작한 후에만 중지된 탄력적 클러스터에 예약 유지 관리를 적용합니다. 7일 후에 Amazon DocumentDB는 중지된 탄력적 클러스터를 자동으로 시작하므로 유지 관리 상태에서 너무 늦어지지 않습니다. 탄력적 클러스터가 다시 시작되면 클러스터의 샤드 비용이 다시 청구됩니다.

탄력적 클러스터가 중지되는 동안 Amazon DocumentDB는 자동화된 백업을 수행하지 않으며 백업 보존 기간도 연장하지 않습니다.

Amazon DocumentDB 탄력적 클러스터 유지 관리

주제

- 보류 중인 탄력적 클러스터 유지 관리 작업 보기
- 탄력적 클러스터 엔진 업데이트
- 탄력적 클러스터 운영 체제 업데이트

Amazon DocumentDB는 Amazon DocumentDB 탄력적 클러스터 리소스를 정기적으로 유지 관리합니다. 이러한 유지 관리에는 데이터베이스 엔진(탄력적 클러스터 유지 관리) 또는 탄력적 클러스터의 기본 운영 체제(OS 업데이트)가 포함되는 경우가 많습니다. 데이터베이스 엔진 업데이트는 필수 패치이며 데이터베이스 엔진에 대한 보안 수정, 버그 수정 및 개선 사항을 포함합니다. 대부분의 운영 체제 패치는 선택 사항이지만 한동안 패치를 적용하지 않으면 보안 태세를 유지하기 위해 패치가 필요할 수 있고 자동으로 적용될 수 있습니다. Amazon DocumentDB 탄력적 클러스터를 사용할 수 있게 되는 즉시 운영 체제 업데이트를 적용합니다.

데이터베이스 엔진 패치를 사용하려면 Amazon DocumentDB 탄력적 클러스터를 잠시 오프라인으로 전환해야 합니다. 사용 가능하게 되면 Amazon DocumentDB 탄력적 클러스터의 예정된 유지 관리 기간 동안 이러한 패치가 자동으로 적용되도록 예약됩니다.

탄력적 클러스터에는 고유한 유지 관리 기간이 있습니다. 즉시 적용되지 않도록 선택한 탄력적 클러스터 수정 사항은 유지 관리 기간에도 적용됩니다. 기본적으로 탄력적 클러스터를 생성할 때 Amazon DocumentDB가 탄력적 클러스터에 대해 유지 관리 기간을 지정합니다. 탄력적 클러스터를 생성할 때 유지 관리 기간을 선택할 수 있습니다. 또한 언제라도 비즈니스 일정 또는 절차에 맞춰 유지 관리 기간을 수정할 수 있습니다. 일반적으로 애플리케이션에 대 유지 관리의 영향을 최소화하는 유지 관리 기간을 서택하는 것이 좋습니다(예: 야간 또는 주말).

보류 중인 탄력적 클러스터 유지 관리 작업 보기

AWS CLI를 사용하여 유지 관리 업데이트를 탄력적 클러스터에서 사용할 수 있는지 여부를 확인할 수 있습니다.

업데이트가 있을 경우에는 다음 중 한 가지를 선택할 수 있습니다.

- 현재 다음 유지 관리 기간에 예약된 유지 관리 작업을 연기합니다(OS 패치만 해당).
- 유지 관리 작업을 즉시 적용합니다.
- 다음 유지 관리 기간 중 시작할 유지 관리 작업을 예약합니다.
- 선택한 신청 기간 중 유지 관리 작업을 시작하도록 예약합니다.

유지 관리 기간에 따라 대기 중인 작업의 시작 시기가 결정되지만 이러한 작업의 전체 실행 시간이 제한되지는 않습니다.

다음 AWS CLI 작업을 사용하여 보류 중인 유지 관리 작업을 확인합니다. 보류 중인 모든 유지 관리 작업을 나열합니다.

```
aws docdb-elastic list-pending-maintenance-actions
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
'ResourcePendingMaintenanceActions': [
    {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'ENGINE_UPDATE',
                'AutoAppliedAfterDate': 'string',
                'ForcedApplyDate': 'string',
                'OptInStatus': 'string',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            },
        ]
    },
],
'NextToken': 'string'
}
```

지정된 `resourceArn`에 대해 보류 중인 유지 관리 작업(있는 경우)을 가져옵니다.

```
aws docdb-elastic get-pending-maintenance-action --resource-arn string-arn
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    'ResourcePendingMaintenanceAction': {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'ENGINE_UPDATE',
                'AutoAppliedAfterDate': 'string',
                'ForcedApplyDate': 'string',
                'OptInStatus': 'string',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            }
        ]
    }
}
```

파라미터:

- ResourceArn - 보류 중인 유지 관리 작업이 적용되는 리소스의 Amazon DocumentDB Amazon 리소스 이름(ARN)입니다.
- Action – 리소스에 적용 중인 보류 중인 유지 관리 작업입니다.

  유효한 값:

  - ENGINE_UPDATE
  - ENGINE_UPGRADE
  - SECURITY_UPDATE
  - OS_UPDATE
  - MASTER_USER_PASSWORD_UPDATE
- AutoAppliedAfterDate - 이 날짜 이후의 첫 번째 유지 관리 기간입니다. 이 경우 NEXT_MAINTENANCE OPT_IN은 무시됩니다.
- ForcedApplyDate - 유지 관리 기간에 관계없이 적용됩니다. 이 경우 IMMEDIATE OPT_IN은 무시됩니다.
- OptInStatus - 옵트인 요청의 유형을 지정하거나 옵트인 요청을 실행 취소하는 값입니다. IMMEDIATE 유형의 옵트인 요청은 실행 취소할 수 없습니다.

  유효한 값:

  - IMMEDIATE - 유지 관리 작업을 즉시 적용합니다.
  - NEXT_MAINTENANCE - 리소스에 대한 다음 유지 관리 기간 중에 유지 관리 작업을 적용합니다.
  - APPLY_ON - 리소스의 다음 유지 관리 기간에 관계없이 지정된 적용 날짜에 유지 관리 작업을 적용합니다.
  - UNDO_OPT_IN - 기존 NEXT_MAINTENANCE 또는 APPLY_ON 옵트인 요청을 취소합니다.
- CurrentApplyDate - opt-in-type이 APPLY_ON인 경우 표시됩니다.
- Description - 유지 관리 작업에 대한 옵션 설명입니다.

탄력적 클러스터 엔진 업데이트

Amazon DocumentDB를 사용하여 유지 관리 작업을 적용하는 시기를 선택할 수 있습니다. AWS CLI를 사용하여 Amazon DocumentDB에서 업데이트를 적용하는 시기를 결정할 수 있습니다.

보류 중인 유지 관리 작업 적용:

```
aws docdb-elastic apply-pending-maintenance-action
--resource-arn string-arn
--apply-action string-enum
--opt-in-type string-enum
[--apply-on string-date-range]
```

파라미터:

- **--resource-arn** - 보류 중인 유지 관리 작업이 적용되는 리소스의 Amazon DocumentDB Amazon 리소스 이름(ARN)입니다.
- **--apply-action** - 이 리소스에 적용할 대기 중인 유지 관리 작업입니다.

  유효한 값:

  - ENGINE_UPDATE
  - ENGINE_UPGRADE
  - SECURITY_UPDATE
  - OS_UPDATE
  - MASTER_USER_PASSWORD_UPDATE
- **--opt-in-type** - 옵트인 요청의 유형을 지정하거나 옵트인 요청을 실행 취소하는 값입니다. IMMEDIATE 유형의 옵트인 요청은 실행 취소할 수 없습니다.

유효한 값:

- IMMEDIATE - 유지 관리 작업을 즉시 적용합니다.
- NEXT_MAINTENANCE - 리소스에 대한 다음 유지 관리 기간 중에 유지 관리 작업을 적용합니다.
- APPLY_ON - 리소스의 다음 유지 관리 기간에 관계없이 지정된 적용 날짜에 유지 관리 작업을 적용합니다.
- UNDO_OPT_IN - 기존 NEXT_MAINTENANCE 또는 APPLY_ON 옵트인 요청을 취소합니다.

- **[--apply-on]** - opt-in-type이 APPLY_ON인 경우 필수입니다. 형식: yyyy/MM/dd HH:mm-yyyy/MM/dd HH:mm(이 옵션은 UTC 시간을 사용합니다. 시작 시간은 최소 30분에서 최대 14일 또는 보류 중인 작업에 대한 강제 적용 날짜 중 먼저 도래하는 날짜부터 향후 언제든지 가능합니다. 시작부터 종료까지의 기간은 최소 30분에서 최대 8시간입니다.)

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
 'ResourcePendingMaintenanceAction': {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'SECURITY_UPDATE',
                'AutoAppliedAfterDate': 'string',
                'ForcedApplyDate': 'string',
                'OptInStatus': 'IMMEDIATE',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            },
        ]
 }
}
```

파라미터:

- ResourceArn - 보류 중인 유지 관리 작업이 적용되는 리소스의 Amazon DocumentDB Amazon 리소스 이름(ARN)입니다.
- Action – 리소스에 적용 중인 보류 중인 유지 관리 작업입니다.

  유효한 값:

  - ENGINE_UPDATE

  - ENGINE_UPGRADE
  - SECURITY_UPDATE
  - OS_UPDATE
  - MASTER_USER_PASSWORD_UPDATE
- AutoAppliedAfterDate - 이 날짜 이후의 첫 번째 유지 관리 기간입니다. 이 경우 NEXT_MAINTENANCE OPT_IN은 무시됩니다.
- ForcedApplyDate - 유지 관리 기간에 관계없이 적용됩니다. 이 경우 IMMEDIATE OPT_IN은 무시됩니다.
- OptInStatus - 옵트인 요청의 유형을 지정하거나 옵트인 요청을 실행 취소하는 값입니다. IMMEDIATE 유형의 옵트인 요청은 실행 취소할 수 없습니다.

  유효한 값:

  - IMMEDIATE - 유지 관리 작업을 즉시 적용합니다.
  - NEXT_MAINTENANCE - 리소스에 대한 다음 유지 관리 기간 중에 유지 관리 작업을 적용합니다.
  - APPLY_ON - 리소스의 다음 유지 관리 기간에 관계없이 지정된 적용 날짜에 유지 관리 작업을 적용합니다.
  - UNDO_OPT_IN - 기존 NEXT_MAINTENANCE 또는 APPLY_ON 옵트인 요청을 취소합니다.
- CurrentApplyDate - opt-in-type이 APPLY_ON인 경우 표시됩니다.
- Description - 유지 관리 작업에 대한 옵션 설명입니다.

적용 날짜

각 유지 관리 작업에는 대기 중인 유지 관리 작업을 설명할 때 찾을 수 있는 적용 날짜가 있습니다. 에서 보류 중인 유지 관리 작업의 출력을 읽으면 세 AWS CLI가지 날짜가 나열됩니다.

- CurrentApplyDate - 유지 관리 작업이 즉시 또는 다음 유지 관리 기간에 적용되는 날짜입니다. 해당 유지 관리가 선택 항목일 경우 이 값은 Null입니다.
- ForcedApplyDate - 유지 관리가 사용자의 유지 관리 기간과 상관없이 자동으로 적용되는 날짜입니다.
- AutoAppliedAfterDate - 유지 관리가 클러스터의 유지 관리 기간 동안 이 날짜 이후에 적용됩니다.

사용자가 생성한 유지 관리 작업

Amazon DocumentDB 탄력적 클러스터 사용자는 클러스터 구성에 대한 업데이트를 시작할 수 있습니다.

클러스터 기본 암호 업데이트

```
aws docdb-elastic update-cluster
--cluster-arn string-arn
[--admin-user-password string]
[--auth-type string-enum]
[--apply-method string-enum]
[--apply-on string-date-range]
#... other parameters of the API that follow here are not relevant for this
 configuration
```

파라미터:

- **`--cluster-arn`** - 유지 관리 작업이 적용되는 리소스의 Amazon DocumentDB Amazon 리소스 이름(ARN)입니다.
- **`[--admin-user-password]`** - 관리자 사용자와 연결된 암호입니다.
- **`[--auth-type]`** - 탄력적 클러스터에 액세스하는 데 사용되는 암호를 가져올 위치를 결정하는 데 사용되는 인증 유형입니다. 유효한 형식은 `PLAIN_TEXT` 및 `SECRET_ARN`입니다.
- **`[--apply-method]`** - 적용할 메서드 유형을 지정하는 값입니다. 허용되는 값은 `IMMEDIATE` 및 `APPLY_ON`입니다. 기본값은 `IMMEDIATE`입니다.
- **`[--apply-on]`** - `apply-method`가 `APPLY_ON`인 경우 필수입니다. 형식: `yyyy/MM/dd HH:mm-yyyy/MM/dd HH:mm`(이 옵션은 UTC 시간을 사용합니다. 시작 시간은 최소 30분에서 최대 14일 부터 향후 언제든지 가능합니다. 시작부터 종료까지의 기간은 최소 30분에서 최대 8시간입니다.)

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
 'ResourcePendingMaintenanceAction': {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'MASTER_USER_PASSWORD_UPDATE',
                'OptInStatus': 'APPLY_ON',
                'CurrentApplyDate': 'string',
                'Description': 'string'
```

```
            },
        ]
 }
}
```

Amazon DocumentDB 유지 관리 윈도우 변경

유지 관리 기간은 사용률이 가장 낮은 시간에 할당되어야 하므로 수시로 변경되어야 할 수 있습니다. 시스템 변경 사항(규모 조정 스토리지 작업 변경 등)을 적용 중이고 가동 중단이 필요한 경우에만 인 이 기간 동안 탄력적 클러스터를 사용할 수 없습니다. 필수 변경 사항을 적용하는 데 필요한 최소 시간 동안만 사용이 불가능합니다.

기본값은 각 Amazon Web Services Region(아마존 웹 서비스 리전)의 8시간 블록에서 임의로 선택된 30분 창으로, 주 마다 임의의 요일에 발생합니다.

유지 관리 기간을 변경하려면 엘라스틱 클러스터 구성 수정 섹션을 참조하세요.

탄력적 클러스터 운영 체제 업데이트

Amazon DocumentDB 탄력적 클러스터에는 때때로 운영 체제 업데이트가 필요합니다. Amazon DocumentDB는 운영 체제를 최신 버전으로 업그레이드하여 데이터베이스 성능과 고객의 전반적인 보안 태세를 개선합니다. 운영 체제 업데이트는 Amazon DocumentDB 탄력적 클러스터의 클러스터 엔진 버전을 변경하지 않습니다.

Amazon DocumentDB 탄력적 클러스터에 대한 대부분의 운영 체제 업데이트는 선택 사항이며 이를 적용할 설정된 날짜가 없습니다. 그러나 한동안 이러한 업데이트를 적용하지 않으면 결국 클러스터의 유지 관리 기간 동안 업데이트가 필요해지고 자동으로 적용될 수 있습니다. 이는 데이터베이스의 보안 태세를 유지하는 데 도움이 됩니다. 예기치 않은 가동 중지를 방지하려면 운영 체제 업데이트를 사용 가능한 즉시 Amazon DocumentDB 탄력적 클러스터에 적용하고 비즈니스 요구 사항에 맞는 시간에 클러스터 유지 관리 기간을 설정합니다.

Amazon DocumentDB 탄력적 클러스터에 대한 저장 데이터 암호화

다음 주제는 Amazon DocumentDB 탄력적 클러스터의 AWS Key Management Service 암호화 키를 학습, 생성 및 모니터링하는 데 도움이 됩니다.

주제

- Amazon DocumentDB 탄력적 클러스터가에서 권한 부여를 사용하는 방법 AWS KMS
- 고객 관리형 키 생성
- Amazon DocumentDB 탄력적 클러스터에 대한 암호화 키 모니터링

- 자세히 알아보기

Amazon DocumentDB 탄력적 클러스터는 키 관리를 위해 AWS Key Management Service (AWS KMS)와 자동으로 통합되며 봉투 암호화라는 메서드를 사용하여 데이터를 보호합니다. 봉투 암호화에 대한 자세한 내용은 AWS Key Management Service 개발자 가이드의 봉투 암호화를 참조하십시오.

AWS KMS key 는 키의 논리적 표현입니다. KMS 키에는 키 ID, 생성 날짜, 설명 및 키 상태와 같은 메타데이터가 포함됩니다. 또한 KMS 키에는 데이터를 암호화 및 복호화하는 데 사용되는 키 재료도 포함됩니다. KMS 키에 대한 자세한 내용은 AWS Key Management Service 개발자 가이드의 AWS KMS keys 섹션을 참조하십시오.

Amazon DocumentDB 탄력적 클러스터는 두 가지 유형의 키를 사용한 암호화를 지원합니다:

- AWS 소유 키 - Amazon DocumentDB 탄력적 클러스터는 기본적으로 이러한 키를 사용하여 개인 식별 데이터를 자동으로 암호화합니다. AWS소유 키를 확인, 관리 또는 사용하거나 사용을 감사할 수 없습니다. 하지만 데이터를 암호화하는 키를 보호하기 위해 어떤 작업을 수행하거나 어떤 프로그램을 변경할 필요가 없습니다. 자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS 소유 키를 참조하세요.
- 고객 관리형 키 - 사용자가 생성, 소유 및 관리하는 대칭 키 AWS KMS keys 입니다. 이 암호화 계층을 완전히 제어할 수 있으므로 다음과 같은 작업을 수행할 수 있습니다.
  - 키 정책 수립 및 유지
  - IAM 정책 및 권한 부여 수립 및 유지
  - 키 정책 활성화 및 비활성화
  - 키 암호화 자료 교체
  - 태그 추가
  - 키 별칭 만들기
  - 삭제를 위한 스케줄 키

  자세한 내용은 AWS Key Management Service 개발자 가이드의 고객 관리형 키를 참조하십시오.

> **Important**
>
> Amazon DocumentDB는 대칭 암호화 KMS 키만 지원하므로 클러스터를 암호화하려면 대칭 암호화 KMS 키를 사용해야 합니다. 비대칭 KMS 키를 사용하여 Amazon DocumentDB 탄력적 클러스터의 데이터를 암호화하지 마십시오. 자세한 내용은 AWS Key Management Service 개발자 가이드의 AWS KMS의 비대칭 키를 참조하십시오.

예를 들어, 키에 대한 액세스가 취소된 경우 Amazon DocumentDB가 더 이상 클러스터의 암호화 키에 액세스할 수 없는 경우 암호화된 클러스터는 터미널 상태가 됩니다. 이러한 경우에는 백업 파일에서만 클러스터를 복원할 수 있습니다. Amazon DocumentDB의 경우 항상 1일 동안 백업이 실행됩니다. 또한 암호화된 Amazon DocumentDB 클러스터에 대한 키를 비활성화하면 해당 클러스터에 대한 읽기 및 쓰기 액세스 권한이 손실됩니다. Amazon DocumentDB는 접근 권한이 없는 키로 암호화된 클러스터를 만나면 클러스터를 터미널 상태로 만듭니다. 이러한 상태에서는 클러스터를 더 이상 사용하지 못하기 때문에 데이터베이스의 현재 상태를 복구할 수 없습니다. 클러스터를 복원하려면 Amazon DocumentDB의 암호화 키에 대한 액세스를 다시 실행한 다음 백업에서 클러스터를 복원해야 합니다.

Important

암호화된 클러스터의 KMS 키는 이미 생성한 후에는 변경할 수 없습니다. 암호화된 탄력적 클러스터를 생성하기 전에 암호화 키 요구 사항을 결정해야 합니다.

Amazon DocumentDB 탄력적 클러스터가에서 권한 부여를 사용하는 방법 AWS KMS

Amazon DocumentDB 탄력적 클러스터는 고객 관리형 키를 사용하려면 권한이 필요합니다.

고객 관리형 키로 암호화된 클러스터를 생성하면 Amazon DocumentDB 탄력적 클러스터는 `CreateGrant` 요청을 전송하여 사용자를 대신하여 권한을 생성합니다 AWS KMS. 의 권한 부여 AWS KMS 는 Amazon DocumentDB 탄력적 클러스터에 고객 계정의 KMS 키에 대한 액세스 권한을 부여하는 데 사용됩니다.

Amazon DocumentDB 탄력적 클러스터는 다음 내부 작업에 고객 관리형 키를 사용하려면 권한이 필요합니다:

- AWS KMS 에 `DescribeKey` 요청을 보내 탄력적 클러스터를 생성할 때 입력한 대칭 고객 관리형 키 ID가 유효한지 확인합니다.
- AWS KMS 에 `GenerateDataKey` 요청을 보내 고객 관리형 키로 암호화된 데이터 키를 생성합니다.
- AWS KMS 에 `Decrypt` 요청을 보내 암호화된 데이터 키를 복호화하여 데이터를 암호화하는 데 사용할 수 있도록 합니다.

- 언제든지 권한 부여에 대한 액세스 권한을 취소하거나 고객 관리형 키에 대한 서비스 액세스를 제거할 수 있습니다. 이렇게 하면 Amazon DocumentDB 탄력적 클러스터는 고객 관리형 키에 의해 암호화된 데이터에 액세스할 수 없게 되어 해당 데이터에 의존하는 작업에 영향을 미칩니다.

고객 관리형 키 생성

AWS Management Console 또는 AWS KMS API를 사용하여 대칭 고객 관리형 키를 생성할 수 있습니다.

대칭 고객 관리형 키

AWS Key Management Service 개발자 안내서의 대칭 고객 관리형 키 생성 단계를 따릅니다.

키 정책

키 정책에서는 고객 관리형 키에 대한 액세스를 제어합니다. 모든 고객 관리형 키에는 키를 사용할 수 있는 사람과 키를 사용하는 방법을 결정하는 문장이 포함된 정확히 하나의 키 정책이 있어야 합니다. 고객 관리형 키를 만들 때 키 정책을 지정할 수 있습니다. 자세한 내용은 AWS Key Management Service 개발자 가이드의 AWS Key Management Service 개요에 있는 KMS 키 액세스 정보를 참조하십시오.

고객 관리형 키를 Amazon DocumentDB 탄력적 클러스터 리소스와 함께 사용하려면 키 정책에서 다음과 같은 API 작업을 허용해야 합니다:

- `kms:CreateGrant` - 고객 관리형 키에 권한 부여를 추가합니다. Amazon DocumentDB 탄력적 클러스터에 필요한 권한 부여 작업에 대한 액세스를 허용하는 지정된 KMS 키에 대한 제어 액세스 권한을 부여합니다. 권한 부여에 대한 자세한 내용은 AWS Key Management Service 개발자 가이드의 AWS KMS의 권한 부여를 참조하십시오.
- `kms:DescribeKey` - Amazon DocumentDB 탄력적 클러스터가 키를 검증할 수 있도록 고객 관리형 키 세부 정보를 제공합니다.
- `kms:Decrypt` - Amazon DocumentDB 탄력적 클러스터가 저장된 암호화된 데이터 키를 사용하여 암호화된 데이터에 액세스할 수 있도록 허용합니다.
- `kms:GenerateDataKey` - 데이터 키가 암호화에 즉시 사용되지 않으므로 Amazon DocumentDB 탄력적 클러스터가 암호화된 데이터 키를 생성하고 저장할 수 있습니다.

자세한 내용은 AWS Key Management Service 개발자 안내서의 키 정책의 AWS 서비스에 대한 권한 및 키 액세스 문제 해결을 참조하세요.

IAM 정책을 통한 고객 관리형 키 액세스 제한

KMS 키 정책 외에도 IAM 정책에서 KMS 키 권한을 제한할 수도 있습니다.

다양한 방법으로 IAM 정책을 더 엄격하게 설정할 수 있습니다. 예를 들어 고객 관리형 키를 Amazon DocumentDB 탄력적 클러스터에서 시작되는 요청에만 사용할 수 있도록 하려면 `kms:ViaService` 조건 키를 `docdb-elastic.<region-name>.amazonaws.com` 값과 함께 사용할 수 있습니다.

자세한 내용은 AWS Key Management Service 개발자 가이드의 다른 계정의 사용자가 CMK를 사용하도록 허용을 참조하십시오.

Amazon DocumentDB 탄력적 클러스터에 대한 암호화 키 모니터링

Amazon DocumentDB 탄력적 클러스터 리소스와 함께 고객 관리형 키를 사용하는 경우 AWS CloudTrail 또는 Amazon CloudWatch Logs를 사용하여 Amazon DocumentDB 탄력적 클러스터가 보내는 요청을 추적할 수 있습니다 AWS KMS.

다음 예제는 Amazon DocumentDB 탄력적 클러스터가 고객 관리형 키로 암호화된 데이터에 액세스`DescribeKey`하기 위해 호출하는 AWS KMS key 작업을 모니터링하기 위한 `CreateGrant`, `GenerateDataKeyWithoutPlainText`, `Decrypt`, 및 AWS CloudTrail 이벤트입니다.

CreateGrant

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
```

```
                "creationDate": "2023-05-09T23:04:20Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "docdb-elastic.amazonaws.com"
    },
    "eventTime": "2023-05-09T23:55:48Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "docdb-elastic.amazonaws.com",
    "userAgent": "docdb-elastic.amazonaws.com",
    "requestParameters": {
        "retiringPrincipal": "docdb-elastic.us-east-1.amazonaws.com",
        "granteePrincipal": "docdb-elastic.us-east-1.amazonaws.com",
        "operations": [
            "Decrypt",
            "Encrypt",
            "GenerateDataKey",
            "GenerateDataKeyWithoutPlaintext",
            "ReEncryptFrom",
            "ReEncryptTo",
            "CreateGrant",
            "RetireGrant",
            "DescribeKey"
        ],
        "keyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": {
        "grantId":
 "0ab0ac0d0b000f00ea00cc0a0e00fc00bce000c000f0000000c0bc0a0000aaafSAMPLE",
        "keyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": false,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
```

```
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

GenerateDataKey

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-05-10T18:02:59Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "docdb-elastic.amazonaws.com"
    },
    "eventTime": "2023-05-10T18:03:25Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "docdb-elastic.amazonaws.com",
    "userAgent": "docdb-elastic.amazonaws.com",
    "requestParameters": {
        "keySpec": "AES_256",
```

```
        "keyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Decrypt

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-05-10T18:05:49Z",
                "mfaAuthenticated": "false"
```

```
            }
        },
        "invokedBy": "docdb-elastic.amazonaws.com"
    },
    "eventTime": "2023-05-10T18:06:19Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "docdb-elastic.amazonaws.com",
    "userAgent": "docdb-elastic.amazonaws.com",
    "requestParameters": {
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


DescribeKey


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE3",
        "sessionContext": {
            "sessionIssuer": {
```

```
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "Sampleuser01"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-05-09T23:04:20Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "docdb-elastic.amazonaws.com"
    },
    "eventTime": "2023-05-09T23:55:48Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "docdb-elastic.amazonaws.com",
    "userAgent": "docdb-elastic.amazonaws.com",
    "requestParameters": {
        "keyId": "alias/SampleKmsKey"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-123456SAMPLE"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

자세히 알아보기

다음 리소스에서 저장 데이터 암호화에 대한 추가 정보를 확인할 수 있습니다:

- AWS KMS 개념에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS Key Management Service 기본 개념을 참조하세요.
- AWS KMS 보안에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의에 대한 보안 모범 사례를 AWS Key Management Service 참조하세요.

탄력적 클러스터에서 서비스 연결 역할

Amazon DocumentDB 탄력적 클러스터는 AWS Identity and Access Management (IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 Amazon DocumentDB 탄력적 클러스터에 직접 연결되는 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon DocumentDB 탄력적 클러스터에 의해 사전 정의되며 서비스가 사용자를 대신하여 다른 AWS 서비스를 호출하는 데 필요한 모든 권한을 포함합니다.

서비스 연결 역할을 사용하면 필요한 권한을 수동으로 추가할 필요가 없으므로 Amazon DocumentDB 탄력적 클러스터를 보다 쉽게 사용할 수 있습니다. Amazon DocumentDB 탄력적 클러스터는 서비스 연결 역할의 권한을 정의하며, 달리 정의되지 않는 한 Amazon DocumentDB 탄력적 클러스터만 역할을 맡을 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔터티에 연결할 수 없습니다. 먼저 역할의 관련 리소스를 삭제해야만 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 실수로 제거할 수 없기 때문에 Amazon DocumentDB 탄력적 클러스터 리소스를 보호할 수 있습니다.

서비스 연결 역할을 지원하는 다른 서비스에 대한 내용은 IAM과 함께 작동하는AWS 서비스를 참조하고 서비스 연결 역할 열에서 예로 표시된 서비스를 확인하세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예(Yes) 링크를 선택합니다.

탄력적 클러스터에 대한 서비스 연결 역할 권한

Amazon DocumentDB 탄력적 클러스터는 라는 서비스 연결 역할을 `AWS ServiceRoleForDocDB-Elastic` 사용하여 Amazon DocumentDB 탄력적 클러스터가 클러스터를 대신하여 AWS 서비스를 호출하도록 허용합니다.

이 서비스 연결 역할에는 계정에서 운영할 수 있는 권한을 부여하는 `AmazonDocDB-ElasticServiceRolePolicy`라는 권한 정책이 연결되어 있습니다. 역할 권한 정책을 통해 Amazon DocumentDB 탄력적 클러스터는 지정된 리소스에 대해 다음 작업을 완료할 수 있습니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": [
                        "AWS/DocDB-Elastic"
                    ]
                }
            }
        }
    ]
}
```

> **Note**
>
> IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 연결 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 다음 오류 메시지가 표시되는 경우: “리소스를 생성할 수 없습니다. 서비스 연결 역할을 생성할 권한이 있는지 확인합니다. 그렇지 않은 경우 기다렸다가 나중에 다시 시도하십시오.”, 다음 권한이 활성화되어 있는지 확인하십시오.

```
{
"Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/docdb-elastic.amazonaws.com/
AWSServiceRoleForDocDB-Elastic",
    "Condition": {
"StringLike": {
```

```
"iam:AWSServiceName":"docdb-elastic.amazonaws.com"
        }
    }
}
```

자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한을 참조하세요.

Amazon DocumentDB 탄력적 클러스터를 위한 서비스 연결 역할 생성

서비스 연결 역할은 수동으로 생성할 필요가 없습니다. DB 인스턴스를 생성할 때 Amazon DocumentDB 탄력적 클러스터는 서비스 연결 역할을 생성합니다.

Amazon DocumentDB 탄력적 클러스터에 대한 서비스 연결 역할 편집

Amazon DocumentDB 탄력적 클러스터에서는 `AWS ServiceRoleForDocDB-Elastic` 서비스 연결 역할을 편집할 수 없습니다. 서비스 연결 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 편집을 참조하세요.

Amazon DocumentDB 탄력적 클러스터의 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스를 더 이상 사용할 필요가 없는 경우 해당 역할을 삭제합니다. 그렇게 하면 적극적으로 모니터링하거나 유지 관리하지 않은 미사용 엔터티가 없습니다. 그러나 서비스 연결 역할을 삭제하려면 먼저 모든 클러스터를 삭제해야 합니다.

서비스 연결 역할 정리

IAM을 사용하여 서비스 연결 역할을 삭제하기 전에 먼저 역할에 활성 세션이 없는지 확인하고 역할에서 사용되는 리소스를 모두 제거해야 합니다.

IAM 콘솔에서 서비스 연결 역할에 활성 세션이 있는지 확인하려면

1. AWS Management Console에 로그인하고 IAM 콘솔을 엽니다.
2. IAM 콘솔의 탐색 창에서 역할을 선택합니다. 그런 다음 `AWS ServiceRoleForDocDB-Elastic` 역할의 이름(확인란 아님)을 선택합니다.
3. 선택한 역할의 요약 페이지에서 Access Advisor 탭을 선택합니다.

> **Note**
>
> Amazon DocumentDB 탄력적 클러스터가 `AWS ServiceRoleForDocDB-Elastic` 역할을 사용하고 있는지 확신할 수 없으면 역할을 삭제할 수 있습니다. 서비스가 역할을 사용하는 경우 삭제가 실패하고 역할이 사용 중인 AWS 리전 를 볼 수 있습니다. 역할이 사용 중인 경우에는 세션이 종료될 때까지 기다렸다가 역할을 삭제해야 합니다. 서비스 연결 역할에 대한 세션은 취소할 수 없습니다.
> `AWS ServiceRoleForDocDB-Elastic` 역할을 제거하려면 먼저 모든 클러스터를 삭제해야 합니다.

모든 클러스터 삭제

Amazon DocumentDB 콘솔에서 클러스터를 삭제하려면

1. AWS Management Console에 로그인하고 Amazon DocumentDB 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 삭제할 클러스터를 선택합니다.
4. 작업에 대해 삭제를 선택합니다.
5. 최종 스냅샷을 생성할까요?를 묻는 메시지가 나타나면 예 또는 아니오를 선택합니다.
6. 이전 단계에서 예를 선택한 경우 최종 스냅샷 이름에 최종 스냅샷 이름을 입력합니다.
7. Delete(삭제)를 선택합니다.

> **Note**
>
> IAM 콘솔, IAM CLI 또는 IAM API를 사용하여 `AWS ServiceRoleForDocDB-Elastic` 서비스 연결 역할을 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서에서 서비스 연결 역할 삭제를 참조하세요.

Amazon DocumentDB 할당량

이 주제에서는 Amazon DocumentDB(MongoDB 호환)의 리소스 할당량 및 이름 지정 제약 조건에 대해 설명합니다.

일부 관리 기능의 경우 Amazon DocumentDB는 Amazon Relational Database Service(RDS) 및 Amazon Neptune과 공유되는 운영 기술을 사용합니다.

주제

- 지원되는 인스턴스 유형
- 지원되는 리전:
- 리전별 할당량
- 집계 할당량
- 클러스터 할당량
- 인스턴스 할당량
- 명명 제약 조건
- TTL 제약 조건
- 탄력적 클러스터 할당량
- 탄력적 클러스터 샤드 할당량
- 샤드당 탄력적 클러스터 CPU, 메모리, 연결 및 커서 할당량

지원되는 인스턴스 유형

Amazon DocumentDB는 온디맨드 인스턴스와 다음의 인스턴스 유형을 지원합니다.

- NVMe 지원:
  - R6GD 인스턴스 유형: `db.r6gd.xlarge`, `db.r6gd.2xlarge`, `db.r6gd.4xlarge`, `db.r6gd.8xlarge`, `db.r6gd.12xlarge`, `db.r6gd.16xlarge`.
- 메모리 최적화:
  - R8G 인스턴스 유형: `db.r8g.large`, `db.r8g.xlarge`, `db.r8g.2xlarge`, , `db.r8g.4xlarge`, `db.r8g.8xlarge`, `db.r8g.12xlargedb.r8g.16xlarge`, `db.r8g.24xlarge`, . `db.r8g.48xlarge`

  - R6G 인스턴스 유형: db.r6g.large, db.r6g.xlarge, db.r6g.2xlarge, db.r6g.4xlarge, db.r6g.8xlarge, , db.r6g.12xlarge. db.r6g.16xlarge
  - R5 인스턴스 유형: db.r5.large, db.r5.xlarge, db.r5.2xlarge, db.r5.4xlarge, db.r5.8xlarge, db.r5.12xlarge, db.r5.16xlarge.
  - R4 인스턴스 유형: db.r4.large, db.r4.2xlarge, db.r4.4xlarge, db.r4.8xlarge, db.r4.16xlarge.
- 버스트 가능한 성능:
  - T4G 인스턴스 유형: db.t4g.medium.
  - T3 인스턴스 유형: db.t3.medium.

지원되는 인스턴스 유형 및 사양에 대한 자세한 내용은 인스턴스 클래스 사양 섹션을 참조하십시오.

지원되는 리전:

Amazon DocumentDB는 AWS 리전다음에서 사용할 수 있습니다.

리전 이름	리전	가용 영역(컴퓨팅)
미국 동부(오하이오)	us-east-2	3
미국 동부(버지니아 북부)	us-east-1	6
미국 서부(오리건)	us-west-2	4
아프리카(케이프타운)	af-south-1	3
남아메리카(상파울루)	sa-east-1	3
아시아 태평양(홍콩)	ap-east-1	3
아시아 태평양(하이데라바드)	ap-south-2	3
아시아 태평양(말레이시아)	ap-southeast-5	3
아시아 태평양(뭄바이)	ap-south-1	3
아시아 태평양(오사카)	ap-northeast-3	3
아시아 태평양(서울)	ap-northeast-2	4
아시아 태평양(싱가포르)	ap-southeast-1	3
아시아 태평양(시드니)	ap-southeast-2	3
아시아 태평양(자카르타)	ap-southeast-3	3
아시아 태평양(멜버른)	ap-southeast-4	3

리전 이름	리전	가용 영역(컴퓨팅)
아시아 태평양(태국)	ap-southeast-7	3
아시아 태평양(도쿄)	ap-northeast-1	3
캐나다(중부)	ca-central-1	3
캐나다 서부(캘거리)	ca-west-1	3
중국(베이징) 리전	cn-north-1	3
중국(닝샤)	cn-northwest-1	3
유럽(프랑크푸르트)	eu-central-1	3
유럽(취리히)	eu-central-2	3
유럽(아일랜드)	eu-west-1	3
유럽(런던)	eu-west-2	3
유럽(밀라노)	eu-south-1	3
유럽(파리)	eu-west-3	3
유럽(스페인)	eu-south-2	3
유럽(스톡홀름)	eu-north-1	3
멕시코(중부)	mx-central-1	3
중동(UAE)	me-central-1	3
이스라엘(텔아비브)	il-central-1	3
AWS GovCloud(미국 서부)	us-gov-west-1	3
AWS GovCloud(미국 동부)	us-gov-east-1	3

리전별 할당량

일부 관리 기능의 경우 Amazon DocumentDB는 Amazon Relational Database Service(RDS)와 공유되는 운영 기술을 사용합니다. 다음 표에는 Amazon DocumentDB와 Amazon RDS 간에 공유되는 리전 할당량이 포함되어 있습니다.

> **Note**
>
> 위에서 설명한 Amazon RDS 공유 기술은 Amazon DocumentDB 인스턴스 기반 클러스터에만 적용됩니다. Amazon DocumentDB 탄력적 클러스터는 Amazon RDS와 기술을 공유하지 않습니다.

다음 할당량은 Amazon DocumentDB 인스턴스 기반 클러스터에 적용되며 리전 AWS 계정 당 입니다.

Resource	AWS 기본 할당량
클러스터	40
클러스터 파라미터 그룹	50
파라미터 그룹당 클러스터	100
파라미터 그룹당 인스턴스	300
이벤트 구독	20
인스턴스	40
수동 클러스터 스냅샷 수	100
클러스터당 읽기 전용 복제본	15
서브넷 그룹 수	50
서브넷 그룹 1개당 서브넷 수	20
리소스당 태그	50
인스턴스당 VPC 보안 그룹	5

다음 할당량은 Amazon DocumentDB 탄력적 클러스터에 적용되며 리전 AWS 계정 당 입니다.

Resource	AWS 기본 할당량
엘라스틱 클러스터	20
탄력적 클러스터 vCPU	1024
수동 탄력적 클러스터 스냅샷	20

할당량을 조정할 수 있는 경우 Service Quotas를 사용하여 할당량 증가를 요청할 수 있습니다. 일부 요청은 자동으로 해결되고 다른 요청은 제출됩니다 지원. 제출된 할당량 증가 요청의 상태를 추적할 수 있습니다 지원. Service Quotas 증가 요청은 우선 순위 지원을 받지 못합니다. 긴급 요청이 있는 경우에 문의하십시오지원. Service Quotas에 대한 자세한 내용은 Service Quotas는 무엇입니까?를 참조하십시오.

Amazon DocumentDB에 대한 할당량 증가를 요청하려면

1. https://console.aws.amazon.com/servicequotas에서 Service Quotas 콘솔을 열고 필요한 경우 로그인합니다.
2. 탐색 창에서 AWS 서비스를 선택합니다.
3. 목록에서 Amazon DocumentDB(MongoDB 호환) 또는 Amazon DocumentDB 탄력적 클러스터를 선택하거나 검색 필드에 어느 하나를 입력합니다.
4. 할당량을 조정할 수 있는 경우 해당 라디오 버튼 또는 이름을 선택한 다음 페이지 오른쪽 상단에 있는 할당량 증가 요청를 선택할 수 있습니다.
5. 할당량 값 변경에 새 값을 입력합니다. 새 값은 현재 값보다 커야 합니다.
6. 요청을 선택합니다. 요청이 해결되면 할당량에 대한 적용된 할당량 값이 새 값으로 설정됩니다.
7. 보류 중이거나 최근에 해결된 요청을 보려면 탐색 창에서 대시보드를 선택합니다. 보류 중인 요청의 경우, 요청 상태를 선택하여 요청 접수증을 엽니다. 요청의 초기 상태는 Pending입니다. 상태가 로 변경되면 사례 번호가 Quota requested표시됩니다 지원. 이 사례 번호를 선택하여 요청의 티켓을 엽니다.

집계 할당량

다음 표에서는 Amazon DocumentDB의 집계 할당량을 설명합니다.

Resource	할당량
지원되는 스테이지의 최대 개수	500

클러스터 할당량

다음 표에서는 Amazon DocumentDB 인스턴스 기반 클러스터 할당량을 설명합니다.

Resource	할당량
클러스터 크기(모든 컬렉션 및 인덱스의 합계)	엔진 버전 8.0 이상의 경우 256TiB(이전 엔진 버전의 경우 128TiB)
컬렉션 크기(모든 컬렉션의 합계는 클러스터 할당량을 초과할 수 없음) - 인덱스 크기는 포함되지 않음	32TiB
클러스터당 모음 수	100,000건
클러스터당 데이터베이스 수	100,000건
데이터베이스 크기(모든 데이터베이스의 합계는 클러스터 할당량을 초과할 수 없음)	엔진 버전 8.0 이상의 경우 256TiB(이전 엔진 버전의 경우 128TiB)
문서 중첩 깊이	200개 수준
문서 크기	16MiB
인덱스 키 크기	2,048바이트
모음당 인덱스 수	64
복합 인덱스의 키	32

Resource	할당량
단일 배치 명령 내 최대 읽기 수	100,000건
클러스터당 사용자 수	1000

인스턴스 할당량

다음 표에서는 인스턴스당 Amazon DocumentDB 할당량을 설명합니다.

Amazon DocumentDB 8.0

인스턴스 유형	인스턴스 메모리(GiB)	연결(모두)	커서 할당량	트랜잭션 열기	연결(활성)
T3.medium	4	1000	30	50	102
T4G.medium	4	1000	30	50	102
R5.large	16	3400	450	200	1100
R5.xlarge	32	7000	450	400	2700
R5.2xlarge	64	14200	450	800	4500
R5.4xlarge	128	28400	760	1600	4500
R5.8xlarge	256	60000	1520	3200	4500
R5.12xlarge	384	60000	2280	4800	4500
R5.16xlarge	512	60000	3040	6400	4500
R6G.large*	16	3400	450	200	1100
R6G.xlarge*	32	7000	450	400	2700
R6G.2xlarge*	64	14200	450	800	4500
R6G.4xlarge*	128	28400	760	1600	4500
R6G.8xlarge*	256	60000	1520	3200	4500

인스턴스 유형	인스턴스 메모리(GiB)	연결(모두)	커서 할당량	트랜잭션 열기	연결(활성)
R6G.12xlarge*	384	60000	2280	4800	4500
R6G.16xlarge*	512	60000	3040	6400	4500
R8G.large	16	3400	450	200	1100
R8G.xlarge	32	7000	450	400	2700
R8G.2xlarge	64	14200	450	800	4500
R8G.4xlarge	128	28400	760	1600	4500
R8G.8xlarge	256	60000	1520	3200	4500
R8G.12xlarge	384	60000	2280	4800	4500
R8G.16xlarge	512	60000	3040	6400	4500
R8G.24xlarge	768	60000	4560	9600	4500
R8G.48xlarge	1536	60000	9120	9600	4500

* R6GD 포함

Amazon DocumentDB 5.0

인스턴스 유형	인스턴스 메모리(GiB)	연결(모두)	커서 할당량	트랜잭션 열기	연결(활성)
T3.medium	4	1000	30	50	102

인스턴스 유형	인스턴스 메모리(GiB)	연결(모두)	커서 할당량	트랜잭션 열기	연결(활성)
T4G.medium	4	1000	30	50	102
R5.large	16	3400	450	200	1100
R5.xlarge	32	7000	450	400	2700
R5.2xlarge	64	14200	450	800	4500
R5.4xlarge	128	28400	760	1600	4500
R5.8xlarge	256	60000	1520	3200	4500
R5.12xlarge	384	60000	2280	4800	4500
R5.16xlarge	512	60000	3040	6400	4500
R6G.large*	16	3400	450	200	1100
R6G.xlarge*	32	7000	450	400	2700
R6G.2xlarge*	64	14200	450	800	4500
R6G.4xlarge*	128	28400	760	1600	4500
R6G.8xlarge*	256	60000	1520	3200	4500
R6G.12xlarge*	384	60000	2280	4800	4500
R6G.16xlarge*	512	60000	3040	6400	4500
R8G.large	16	3400	450	200	1100
R8G.xlarge	32	7000	450	400	2700

인스턴스 유형	인스턴스 메모리(GiB)	연결(모두)	커서 할당량	트랜잭션 열기	연결(활성)
R8G.2xlarge	64	14200	450	800	4500
R8G.4xlarge	128	28400	760	1600	4500
R8G.8xlarge	256	60000	1520	3200	4500
R8G.12xlarge	384	60000	2280	4800	4500
R8G.16xlarge	512	60000	3040	6400	4500
R8G.24xlarge	768	60000	4560	9600	4500
R8G.48xlarge	1536	60000	9120	9600	4500

* R6GD 포함

Amazon DocumentDB 4.0

인스턴스 유형	인스턴스 메모리(GiB)	연결(모두)	커서 할당량	트랜잭션 열기	연결(활성)
T3.medium	4	1000	30	50	102
T4G.medium	4	1000	30	50	102
R5.large	16	3400	450	200	1100
R5.xlarge	32	7000	450	400	2700
R5.2xlarge	64	14200	450	800	4500
R5.4xlarge	128	28400	760	1600	4500

인스턴스 유형	인스턴스 메모리(GiB)	연결(모두)	커서 할당량	트랜잭션 열기	연결(활성)
R5.8xlarge	256	60000	1520	3200	4500
R5.12xlarge	384	60000	2280	4800	4500
R5.16xlarge	512	60000	3040	6400	4500
R6G.large*	16	3400	450	200	1100
R6G.xlarge*	32	7000	450	400	2700
R6G.2xlarge*	64	14200	450	800	4500
R6G.4xlarge*	128	28400	760	1600	4500
R6G.8xlarge*	256	60000	1520	3200	4500
R6G.12xlarge*	384	60000	2280	4800	4500
R6G.16xlarge*	512	60000	3040	6400	4500

* R6GD 포함

Amazon DocumentDB 3.6

인스턴스 유형	인스턴스 메모리(GiB)	연결(모두)	커서 할당량	트랜잭션 열기	연결(활성)
T3.medium	4	500	30	해당 사항 없음	102

인스턴스 유형	인스턴스 메모리(GiB)	연결(모두)	커서 할당량	트랜잭션 열기	연결(활성)
R4.large	15.25	1700	450	해당 사항 없음	1100
R4.xlarge	30.5	3400	450	해당 사항 없음	2700
R4.2xlarge	61	6800	450	해당 사항 없음	4500
R4.4xlarge	122	13600	725	해당 사항 없음	4500
R4.8xlarge	288	27200	1450	해당 사항 없음	4500
R4.16xlarge	488	30000	2900	해당 사항 없음	4500
R5.large	16	1700	450	해당 사항 없음	1100
R5.xlarge	32	3500	450	해당 사항 없음	2700
R5.2xlarge	64	7100	450	해당 사항 없음	4500
R5.4xlarge	128	14200	760	해당 사항 없음	4500
R5.8xlarge	256	28400	1520	해당 사항 없음	4500
R5.12xlarge	384	30000	2280	해당 사항 없음	4500

인스턴스 유형	인스턴스 메모리(GiB)	연결(모두)	커서 할당량	트랜잭션 열기	연결(활성)
R5.16xlarge	512	30000	3040	해당 사항 없음	4500

다음 CloudWatch 지표를 사용하여 인스턴스당 제한을 모니터링하고 경보를 생성할 수 있습니다. Amazon DocumentDB CloudWatch 지표에 관한 더 많은 내용은 CloudWatch를 사용하여 Amazon DocumentDB 모니터링를 참조하세요.

Resource	CloudWatch 할당량 지표	CloudWatch 사용량 지표(최대 1분)	CloudWatch 사용량 지표
인스턴스 메모리	-	-	FreeableMemory
연결(모두)	DatabaseConnectionsLimit	DatabaseConnectionsMax	DatabaseConnections
커서	DatabaseCursorsLimit	DatabaseCursorsMax	DatabaseCursors
트랜잭션	TransactionsOpenLimit	TransactionsOpenMax	TransactionsOpen

명명 제약 조건

다음 표에서는 Amazon DocumentDB의 명명 제약 조건을 설명합니다.

Resource	기본 할당량
클러스터 식별자(Cluster identifier)	• 길이는 [1~63] 글자, 숫자 또는 하이픈입니다. • 첫 번째 문자는 글자이어야 합니다. • 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다. • 리전별로 AWS 계정당 모든 클러스터(Amazon RDS, Amazon Neptune 및 Amazon DocumentDB 전체)에 대해 고유해야 합니다.
인스턴스 식별자	• 길이는 [1~63] 글자, 숫자 또는 하이픈입니다. • 첫 번째 문자는 글자이어야 합니다 • 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다 • 리전별로 AWS 계정당 모든 인스턴스(Amazon RDS, Amazon Neptune 및 Amazon DocumentDB)에 대해 고유해야 합니다.
컬렉션 이름: `<col>`	길이는 [1~255]자입니다(Amazon DocumentDB 5.0 이상). 길이는 [1~57]자입니다(Amazon DocumentDB 4.0 이하).

Resource	기본 할당량
데이터베이스 이름: <db>	길이는 [1~63]자입니다.
정규화된 컬렉션 이름: <db>.<col>	길이는 [3~255]자입니다(Amazon DocumentDB 5.0 이상). 길이는 [3~120]자입니다(Amazon DocumentDB 4.0 이하).
정규화된 인덱스 이름: <db>.<col>$<index>	길이는 [6~511]자입니다(Amazon DocumentDB 5.0 이상). 길이는 [6~127]자입니다(Amazon DocumentDB 4.0 이하).
인덱스 이름	길이는 [1~255]자입니다(Amazon DocumentDB 5.0 이상). 길이는 결합된 <col>$<index> (Amazon DocumentDB 4.0 이하)의 경우 [3~63]자입니다.

기본 암호	• 길이는 [8-100]자의 인쇄 가능한 ASCII 문자입니다.
	• 다음을 제외한 인쇄 가능한 ASCII 문자를 사용할 수 있습니다.
	  • **/**(슬래시)
	  • **"**(큰 따옴표)
	  • **@**(at 기호)

Resource	기본 할당량
기본 사용자 이름	• 길이는 [1-63]자의 영숫자 문자입니다. • 첫 번째 문자는 글자이어야 합니다. • 데이터베이스 엔진에 포함된 단어는 사용할 수 없습니다.
파라미터 그룹 이름	• [1-255]자 길이의 영숫자 문자입니다. • 첫 번째 문자는 글자이어야 합니다. • 하이픈으로 끝나거나 하이픈이 2개 연속으로 포함되어서는 안 됩니다.

TTL 제약 조건

TTL 인덱스에서의 삭제는 특정 기간 내에 삭제된다고 보장할 수 없으며 가능한 한 빠른 시간 내에 수행됩니다. 인스턴스 리소스 사용률, 문서 크기 및 전체 처리량과 같은 요소가 TTL 삭제 소요 시간에 영향을 줄 수 있습니다.

탄력적 클러스터 할당량

다음 표에서는 Amazon DocumentDB 탄력적 클러스터의 최대 할당량을 설명합니다.

Resource	할당량
리전당 탄력적 클러스터	20
리전당 모든 탄력적 클러스터에서 합산된 vCPU	1024
리전당 수동 클러스터 스냅샷	20
클러스터당 샤드 수	32

Resource	할당량
클러스터당 스토리지(샤드 키로 데이터가 균등하게 분산되는 경우)	4PiB
클러스터에 연결	300,000의 더 낮은 값 또는 샤드 수 x 샤드당 vCPU와 연결된 연결 할당량
샤딩되지 않은 컬렉션 크기	32TiB
샤딩된 컬렉션 크기(샤드 키로 데이터가 균등하게 배포되는 경우)	1PB
클러스터당 데이터베이스 수	10,000
클러스터당 샤딩되지 않은 컬렉션	100,000건
클러스터당 샤딩된 컬렉션	1000
클러스터 당 사용자	100
단일 배치 명령 내 읽기	100,000건
모음당 인덱스 수	64
문서 중첩 깊이	100개 수준
문서 크기	16MB
인덱스 키 크기	2048 바이트
복합 인덱스의 키	32

탄력적 클러스터 샤드 할당량

다음 표에서는 Amazon DocumentDB 탄력적 클러스터의 최대 샤드 할당량을 설명합니다.

Resource	할당량
샤드 인스턴스당 vCPU	64
샤드당 인스턴스 수	16
샤드당 스토리지	128TiB
샤드당 컬렉션별 스토리지	32TiB

샤드당 탄력적 클러스터 CPU, 메모리, 연결 및 커서 할당량

다음 표에서는 Amazon DocumentDB 탄력적 클러스터 샤드의 최대 CPU, 메모리, 연결 및 커서 제한을 설명합니다.

샤드당 vCPU	인스턴스 메모리(GiB)	연결 할당량	커서 할당량
2	16	1700	450
4	32	3500	450
8	64	7100	450
16	128	14200	760
32	256	28400	1520
48	384	30000	2280
64	512	30000	3040

Amazon DocumentDB 문제 해결

다음 섹션에는 Amazon DocumentDB(MongoDB 호환) 사용 시 발생할 수 있는 문제를 해결하는 방법에 대한 정보가 나와 있습니다.

주제

- 연결 문제 해결
- 인덱스 문제 해결
- 성능 및 리소스 사용률 문제 해결
- Amazon DocumentDB의 폐영역 회수
- 오류 코드

연결 문제 해결

연결에 문제가 있으신가요? 다음에서는 몇 가지 일반적인 시나리오와 이에 대한 해결 방법에 대해 설명합니다.

주제

- Amazon DocumentDB 엔드포인트에 연결할 수 없습니다.
- Amazon DocumentDB 인스턴스 연결 테스트
- 유효하지 않은 엔드포인트에 연결
- 연결 수에 영향을 미치는 드라이버 구성

Amazon DocumentDB 엔드포인트에 연결할 수 없습니다.

다음은 Amazon DocumentDB에 연결하려고 할 때 표시될 수 있는 가장 일반적인 오류 메시지 중 하나입니다.

```
connecting to: mongodb://docdb-2018-11-08-21-47-27.cluster-ccuszbx3pn5e.us-east-
1.docdb.amazonaws.com:27017/
2018-11-14T14:33:46.451-0800 W NETWORK [thread1] Failed to connect to
172.31.91.193:27017 after 5000ms milliseconds, giving up.
2018-11-14T14:33:46.452-0800 E QUERY [thread1] Error: couldn't connect to server
docdb-2018-11-08-21-47-27.cluster-ccuszbx3pn5e.us-east-1.docdb.amazonaws.com:27017,
connection attempt failed :
```

```
connect@src/mongo/shell/mongo.js:237:13
@(connect):1:6
exception: connect failed
```

이 오류 메시지의 일반적인 의미는 클라이언트(이 예제에서는 mongo 쉘)가 Amazon DocumentDB 엔드포인트에 액세스할 수 없다는 것입니다. 다음과 같이 여러 가지 원인이 있을 수 있습니다.

주제

- 퍼블릭 엔드포인트로부터 연결
- 교차 리전 연결
- 다른 Amazon VPC에서 연결
- 보안 그룹이 인바운드 연결을 차단
- Java Mongo 드라이버 읽기 기본 설정 문제

퍼블릭 엔드포인트로부터 연결

노트북 또는 로컬 개발 머신에서 직접 Amazon DocumentDB 클러스터에 연결하려고 시도하는 중입니다.

노트북 또는 로컬 개발 머신과 같은 퍼블릭 엔드포인트에서 직접 Amazon DocumentDB 클러스터에 연결하려고 시도하는 것은 실패하게 됩니다. Amazon DocumentDB는 Virtual Private Cloud(VPC) 전용이며 현재 퍼블릭 엔드포인트를 지원하지 않습니다. 따라서 VPC 외부의 노트북 또는 개발 환경에서 Amazon DocumentDB 클러스터에 직접 연결할 수 없습니다.

Amazon VPC 외부에서 Amazon DocumentDB 클러스터에 연결하려면 SSH 터널을 사용할 수 있습니다. 자세한 내용은 Amazon VPC 외부에서 Amazon DocumentDB 클러스터에 연결 섹션을 참조하세요. 또한 개발 환경이 다른 Amazon VPC에 있는 경우 VPC 피어링을 사용하고 동일한 리전 또는 다른 리전의 다른 Amazon VPC에서 Amazon DocumentDB 클러스터에 연결할 수도 있습니다.

교차 리전 연결

다른 리전의 Amazon DocumentDB 클러스터에 연결하려고 합니다.

클러스터 지역이 아닌 지역의 Amazon EC2 인스턴스에서 Amazon DocumentDB 클러스터에 연결하려고 시도할 경우(예: 미국 서부(오레곤) 지역(us-west-2)에서 미국 동부(버지니아 북부) 지역(us-east-1)의 클러스터에 연결하려고 하면 연결이 실패합니다.

Amazon DocumentDB 클러스터의 리전을 확인하려면 다음 명령을 실행합니다. 리전이 엔드포인트에 있습니다.

```
aws docdb describe-db-clusters \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].Endpoint'
```

이 작업의 출력은 다음과 같습니다.

```
[
    "sample-cluster.node.us-east-1.docdb.amazonaws.com"
]
```

EC2 인스턴스의 리전을 확인하려면 다음 명령을 실행합니다.

```
 aws ec2 describe-instances \
     --query 'Reservations[*].Instances[*].Placement.AvailabilityZone'
```

이 작업의 출력은 다음과 같습니다.

```
[
    [
        "us-east-1a"
    ]
]
```

다른 Amazon VPC에서 연결

클러스터가 배포된 Amazon VPC가 아닌 다른 Amazon DocumentDB 클러스터에 연결하려고 합니다.

Amazon DocumentDB 클러스터와 Amazon EC2 인스턴스가 동일한에 AWS 리전있지만 동일한 Amazon VPC에는 없는 경우 두 Amazon VPC 간에 VPC 피어링이 활성화되지 않는 한 Amazon DocumentDB 클러스터에 직접 연결할 수 없습니다 VPCs.

Amazon DocumentDB 인스턴스의 Amazon VPC를 확인하려면 다음 명령을 실행합니다.

```
aws docdb describe-db-instances \
   --db-instance-identifier sample-instance \
   --query 'DBInstances[*].DBSubnetGroup.VpcId'
```

Amazon EC2 인스턴스의 Amazon VPC를 확인하려면 다음 명령을 실행합니다.

```
aws ec2 describe-instances \
```

```
   --query 'Reservations[*].Instances[*].VpcId'
```

보안 그룹이 인바운드 연결을 차단

Amazon DocumentDB 클러스터에 연결하려고 하는데 클러스터의 보안 그룹이 클러스터 포트(기본 포트: 27017)의 인바운드 연결을 허용하지 않습니다.

Amazon DocumentDB 클러스터와 Amazon EC2 인스턴스가 모두 동일한 리전 및 Amazon VPC에 있고 동일한 Amazon VPC 보안 그룹을 사용한다고 가정합니다. Amazon DocumentDB 클러스터에 연결할 수 없는 경우 가능성이 높은 원인은 클러스터에 대한 보안 그룹(예: 방화벽)이 Amazon DocumentDB 클러스터에 대해 선택한 포트(기본 포트: 27017)의 인바운드 연결을 허용하지 않기 때문입니다.

Amazon DocumentDB 클러스터의 포트를 확인하려면 다음 명령을 실행합니다.

```
aws docdb describe-db-clusters \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[DBClusterIdentifier,Port]'
```

클러스터에 대한 Amazon DocumentDB 보안 그룹을 가져오려면 다음 명령을 실행합니다.

```
aws docdb describe-db-clusters \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[VpcSecurityGroups[*],VpcSecurityGroupId]'
```

보안 그룹에 대한 인바운드 규칙을 확인하려면 Amazon EC2 설명서에서 다음 주제를 참조하십시오.

- Linux 인스턴스의 인바운드 트래픽 권한 부여
- Windows 인스턴스의 인바운드 트래픽 권한 부여

Java Mongo 드라이버 읽기 기본 설정 문제

클라이언트의 읽기 기본 설정이 적용되지 않으며 일부 클라이언트는 재부팅하지 않는 한 장애 조치 후 Amazon DocumentDB에 쓸 수 없습니다.

Java Mongo Driver 3.7.x에서 처음 발견된 이 문제는 클라이언트가 특히 `MongoClientSettings` 메서드를 사용하여 Amazon DocumentDB에 연결할 때, 특히 `applyToClusterSettings` 메서드를 체인으로 연결할 때 발생합니다. MongoClient 클러스터 설정은 `hosts()`,

requiredReplicaSetName(), mode()와(과) 같은 몇 가지 다른 방법을 사용하여 정의할 수 있습니다.

클라이언트가 hosts() 메서드에서 호스트를 하나만 지정하면 모드가 ClusterConnectionMode.SINGLE 대신 ClusterConnectionMode.MULTIPLE으로 설정됩니다. 이렇게 하면 클라이언트는 읽기 기본 설정을 무시하고 hosts()에서 구성된 서버에만 연결합니다. 따라서 아래와 같이 클라이언트 설정을 초기화하더라도 모든 읽기는 여전히 보조 서버 대신 기본 서버로 이동합니다.

```
final ServerAddress serverAddress0 = new ServerAddress("cluster-endpoint", 27317));
    final MongoCredential credential = MongoCredential.createCredential("xxx",
            "admin", "xxxx".toCharArray());
    final MongoClientSettings settings = MongoClientSettings.builder()
            .credential(credential)
            .readPreference(ReadPreference.secondaryPreferred())
            .retryWrites(false)
            .applyToSslSettings(builder -> builder
                    .enabled(false))
            .applyToClusterSettings(builder -> builder.hosts(
                            Arrays.asList(serverAddress0
                            ))
                    .requiredReplicaSetName("rs0"))
            .build();
    MongoClient mongoClient = MongoClients.create(settings);
```

장애 조치 사례

위의 클라이언트 연결 설정을 사용하면 클러스터 작성기 엔드포인트에 대한 장애 조치 및 지연된 DNS 레코드 업데이트가 발생하는 경우 클라이언트는 여전히 이전 작성기(이제는 장애 조치 후 리더)에 쓰기를 시도합니다. 이로 인해 서버 측 오류(기본 아님)가 발생하는데, 이 오류는 Java 드라이버에서 적절하게 처리되지 않습니다(이 내용은 아직 조사 중임). 따라서 예를 들어 애플리케이션 서버가 재부팅될 때까지 클라이언트는 잘못된 상태로 남아 있을 수 있습니다.

이에 대한 두 가지 해결 방법이 있습니다.

- 연결 문자열을 통해 Amazon DocumentDB에 연결하는 클라이언트는 읽기 기본 설정을 지정할 때 ClusterConnectionMode가 MULTIPLE로 설정되므로 이 문제가 발생하지 않습니다.

  ```
  MongoClientURI mongoClientURI = new MongoClientURI("mongodb://usr:pass:cluster-
  endpoint:27317/test?ssl=false&replicaSet=rs0&readpreference=secondaryPreferred");
  MongoClient mongoClient = MongoClients.create(mongoClientURI.getURI());
  ```

applyConnectionString 메서드와 함께 MongoClientSettings 빌더를 사용할 수도 있습니다.

```
final MongoClientSettings settings = MongoClientSettings.builder()
        .credential(credential)
        .applyConnectionString(new ConnectionString("usr:pass:cluster-endpoint:27317/
test?ssl=false&replicaSet=rs0&readpreference=secondaryPreferred"))
        .retryWrites(false)
        .applyToSslSettings(builder # builder
                .enabled(false))
        .build();
MongoClient mongoClient = MongoClients.create(settings);
```

- 명시적으로 ClusterConnectionMode을 MULTIPLE으로 설정합니다. 이는 applyToClusterSettings 및 hosts().size() == 1를 사용할 때만 필요합니다.

```
final ServerAddress serverAddress0 = new ServerAddress("cluster-endpoint", 27317));
final MongoCredential credential = MongoCredential.createCredential("xxx","admin",
 "xxxx".toCharArray());
final MongoClientSettings settings = MongoClientSettings.builder()
    .credential(credential)
    .readPreference(ReadPreference.secondaryPreferred())
    .retryWrites(false)
    .applyToSslSettings(builder # builder
    .enabled(false))
    .applyToClusterSettings(builder # builder
                .hosts(Arrays.asList(serverAddress0))
                .requiredReplicaSetName("rs0"))
                .mode(ClusterConnectionMode.MULTIPLE))
    .build();
MongoClient mongoClient = MongoClients.create(settings);
```

Amazon DocumentDB 인스턴스 연결 테스트

공통 Linux 또는 Windows 도구를 사용하여 클러스터에 대한 연결을 테스트할 수 있습니다.

Linux 또는 Unix 터미널에서 다음을 입력하여 연결을 테스트할 수 있습니다(cluster-endpoint를 해당 엔드포인트로 바꾸고 port를 인스턴스의 포트로 바꿈).

```
nc -zv cluster-endpoint port
```

다음은 샘플 작업 및 반환 값의 예입니다.

```
nc -zv docdbTest.d4c7nm7stsfc0.us-west-2.docdb.amazonaws.com 27017

Connection to docdbTest.d4c7nm7stsfc0.us-west-2.docdb.amazonaws.com 27017 port [tcp/*]
 succeeded!
```

유효하지 않은 엔드포인트에 연결

Amazon DocumentDB 클러스터에 연결하고 유효하지 않은 클러스터 엔드포인트를 사용할 때 다음과 유사한 오류가 표시됩니다.

```
mongo --ssl \
   --host sample-cluster.node.us-east-1.docdb.amazonaws.com:27017 \
   --sslCAFile global-bundle.pem \
   --username <user-name> \
   --password <password>
```

출력은 다음과 같습니다.

```
MongoDB shell version v3.6
connecting to: mongodb://sample-cluster.node.us-east-1.docdb.amazonaws.com:27017/
2018-11-14T17:21:18.516-0800 I NETWORK [thread1] getaddrinfo("sample-cluster.node.us-
east-1.docdb.amazonaws.com") failed:
nodename nor servname provided, or not known 2018-11-14T17:21:18.537-0800 E QUERY
 [thread1] Error: couldn't initialize
connection to host sample-cluster.node.us-east-1.docdb.amazonaws.com, address is
 invalid :
connect@src/mongo/shell/mongo.js:237:13@(connect):1:6
exception: connect failed
```

클러스터에 대해 유효한 엔드포인트를 가져오려면 다음 명령을 실행합니다.

```
aws docdb describe-db-clusters \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[Endpoint,Port]'
```

인스턴스에 대해 유효한 엔드포인트를 가져오려면 다음 명령을 실행합니다.

```
aws docdb describe-db-instances \
```

```
    --db-instance-identifier sample-instance \
    --query 'DBInstances[*].[Endpoint.Address,Endpoint.Port]'
```

자세한 내용은 Amazon DocumentDB 엔드포인트에 대한 이해 단원을 참조하십시오.

연결 수에 영향을 미치는 드라이버 구성

클라이언트 드라이버를 사용하여 Amazon DocumentDB 클러스터에 연결할 때는 `maxPoolSize` 구성 파라미터를 고려하는 것이 중요합니다. `maxPoolSize` 설정은 클라이언트 드라이버가 연결 풀에서 유지할 최대 연결 수를 결정합니다.

인덱스 문제 해결

다음 주제에서는 인덱스 또는 배경 인덱스 빌드에 실패한 경우 취해야 할 조치를 설명합니다.

주제

- 인덱스 빌드 실패
- 백그라운드 인덱스 빌드 지연 문제 및 실패
- 데이터베이스 인덱스 팽창

인덱스 빌드 실패

Amazon DocumentDB는 인덱스 생성 프로세스의 일부로써 인스턴스의 로컬 스토리지를 활용합니다. FreeLocalStorage CloudWatch 지표(`CloudWatch -> Metrics -> DocDB -> Instance Metrics`)를 사용하여 이 디스크 사용량을 모니터링할 수 있습니다. 인덱스 빌드에 로컬 디스크 전체가 사용되고 실패할 경우 오류가 표시됩니다. 데이터를 Amazon DocumentDB로 마이그레이션할 경우 먼저 인덱스를 생성한 다음 데이터를 삽입하는 것이 좋습니다. 마이그레이션 전략 및 인덱스 생성에 대한 자세한 내용은 Amazon DocumentDB 설명서의 Amazon DocumentDB 마이그레이션 및 업그레이드 섹션과 오프라인 방법을 사용한 MongoDB에서 Amazon DocumentDB로의 마이그레이션 블로그를 참조하십시오.

기존 클러스터에서 인덱스를 생성할 때 인덱스 빌드가 예상보다 오래 걸리거나 실패하는 경우 인스턴스를 확장하여 인덱스를 생성한 다음 인덱스가 생성된 후 다시 축소합니다. Amazon DocumentDB를 사용하면 AWS Management Console 또는를 사용하여 인스턴스 크기를 몇 분 만에 빠르게 조정할 수 있습니다 AWS CLI. 자세한 내용은 인스턴스 클래스 관리 단원을 참조하십시오. 초당 요금이 청구되는 경우, 초 단위로 계산하여 사용한 리소스에 대해서만 지불하면 됩니다.

백그라운드 인덱스 빌드 지연 문제 및 실패

Amazon DocumentDB의 백그라운드 인덱스 빌드는 인덱스 빌드가 시작되기 전에 시작된 기본 인스턴스의 모든 쿼리가 실행될 때까지 시작되지 않습니다. 쿼리가 오래 실행되는 경우 쿼리가 완료될 때까지 백그라운드 인덱스 빌드가 차단되므로 완료하는 데 예상보다 시간이 오래 걸릴 수 있습니다. 컬렉션이 비어 있는 경우에도 마찬가지입니다.

포그라운드 인덱스 빌드는 동일한 차단 동작을 나타내지 않습니다. 대신 포그라운드 인덱스 빌드는 인덱스 빌드가 완료될 때까지 컬렉션을 독점적으로 잠급니다. 빈 컬렉션에 인덱스를 생성하고 장기 실행 쿼리를 차단하지 않으려면 포그라운드 인덱스 빌드를 사용합니다.

> **Note**
>
> Amazon DocumentDB는 특정 시간에 모음에서 발생하는 배경 인덱스 빌드를 하나만 허용합니다. 배경 인덱스 빌드 중에 동일한 컬렉션에 `createIndex()` 또는 `dropIndex()`와 같은 DDL(Data Definition Language) 연산이 발생하면 배경 인덱스 빌드가 실패합니다.

데이터베이스 인덱스 팽창

Amazon DocumentDB는 다중 버전 동시성 제어(MVCC)를 사용하여 동시 트랜잭션을 관리합니다. 문서가 삭제되거나 업데이트되면 이전 버전은 컬렉션 및 인덱스에 '데드' 버전으로 유지됩니다. 폐영역 회수 프로세스는 향후 작업을 위해 이러한 데드 버전에서 공간을 자동으로 회수합니다.

인덱스 팽창은 데드 버전 또는 더 이상 사용되지 않는 인덱스 항목이 누적되거나 페이지 내에 조각화되면서 컬렉션의 인덱스가 커질 때 발생합니다. 보고된 백분율은 향후 인덱스 항목에서 사용할 수 있는 인덱스 공간의 양을 나타냅니다. 이 팽창은 버퍼 캐시와 스토리지 모두에서 공간을 소비합니다. 팽창을 제거하려면 인덱스를 다시 빌드해야 합니다.

Example예제

다음 명령을 실행하여 인덱스의 미사용 스토리지를 확인합니다.

```
db.coll.aggregate({$indexStats:{}});
```

다음과 비슷한 결과가 반환됩니다.

```
{
    "name" : "_id_",
    "key" : {
```

```
        "_id" : 1
    },
    "host" : "devbox-test.localhost.a2z.com:27317",
    "size" : NumberLong(827392),
    "accesses" : {
        "ops" : NumberLong(40000),
        "docsRead" : NumberLong(46049),
        "since" : ISODate("2025-04-03T21:44:51.251Z")
    },
    "cacheStats" : {
        "blksRead" : NumberLong(264),
        "blksHit" : NumberLong(140190),
        "hitRatio" : 99.8121
    },
    "unusedStorageSize" : {
        "unusedSizeBytes" : 409600,
        "unusedSizePercent" : 49.51
    }
}
```

전체 컬렉션을 스캔해야 하는 reIndex 명령을 사용하여 가동 중지 없이 인덱스를 다시 빌드할 수 있습니다. reIndex를 사용한 인덱스 유지 관리을(를) 참조하세요.

성능 및 리소스 사용률 문제 해결

이 섹션에서는 Amazon DocumentDB 배포 시 일반적인 진단 문제에 대한 질문과 솔루션을 제공합니다. 제공된 예제는 mongo 쉘을 사용하며 개별 인스턴스에 적용됩니다. 인스턴스 엔드포인트를 찾으려면 Amazon DocumentDB 엔드포인트에 대한 이해을 참조하세요.

주제

- Mongo API를 통해 내 컬렉션에 대해 수행된 삽입, 업데이트 및 삭제 작업 수를 확인하려면 어떻게 해야 하나요?
- 캐시 성능을 분석하려면 어떻게 해야 하나요?
- 장시간 실행 중이거나 차단된 쿼리를 찾아서 종료하려면 어떻게 해야 하나요?
- 쿼리 계획을 보고 쿼리를 최적화하려면 어떻게 해야 하나요?
- 엘라스틱 클러스터에서 쿼리 계획을 확인하려면 어떻게 해야 하나요?
- 인스턴스에서 실행 중인 작업을 모두 나열하려면 어떻게 해야 하나요?
- 쿼리 진행 상황을 어떻게 알 수 있나요?
- 시스템이 갑자기 느리게 실행되는 이유를 어떻게 알 수 있나요?

- 하나 이상의 클러스터 인스턴스에서 높은 CPU 사용률의 원인을 어떻게 확인하나요?
- 인스턴스에서 열려 있는 커서를 어떻게 확인하나요?
- 현재 Amazon DocumentDB 엔진 버전은 어떻게 확인하나요?
- 인덱스 사용량을 분석하고 사용하지 않는 인덱스를 식별하려면 어떻게 해야 하나요?
- 누락된 인덱스는 어떻게 식별하나요?
- 데이터베이스 수집 팽창을 확인하려면 어떻게 해야 하나요?
- 유용한 쿼리 요약

Mongo API를 통해 내 컬렉션에 대해 수행된 삽입, 업데이트 및 삭제 작업 수를 확인하려면 어떻게 해야 하나요?

특정 컬렉션에서 수행된 삽입, 업데이트 및 삭제 작업 수를 보려면 해당 컬렉션에서 다음 명령을 실행합니다.

```
db.collection.stats()
```

명령에 대한 출력은 opCounters 필드에서 다음과 같습니다.

- numDocsIns - 이 컬렉션에 삽입된 문서 수입니다. 여기에는 `insert` 및 `insertMany` 명령을 사용하여 삽입한 문서와 업서트로 삽입한 문서가 포함됩니다.
- numDocsUpd - 이 컬렉션에서 업데이트된 문서 수입니다. 여기에는 `update` 및 `findAndModify` 명령을 사용하여 업데이트한 문서가 포함됩니다.
- numDocsDel - 이 컬렉션에서 삭제된 문서 수입니다. 여기에는 `deleteOne`, `deleteMany`, `remove`, 및 `findAndModify` 명령을 사용하여 삭제된 문서가 포함됩니다.
- lastReset - 이 카운터를 마지막으로 재설정한 시간입니다. 이 명령으로 제공된 통계는 클러스터를 시작/중지하거나 인스턴스를 스케일 업/다운할 때 재설정됩니다.

`db.collection.stats()`를 실행한 예제 출력은 다음과 같습니다.

```
{
    "ns" : "db.test",
    "count" : ...,
    "size" : ...,
    "avgObjSize" : ...,
    "storageSize" : ...,
```

```
    "capped" : false,
    "nindexes" : ...,
    "totalIndexSize" : ...,
    "indexSizes" : {
        "_id_" : ...,
        "x_1" : ...
    },
    "collScans" : ...,
    "idxScans" : ...,
    "opCounter" : {
        "numDocsIns" : ...,
        "numDocsUpd" : ...,
        "numDocsDel" : ...
    },
    "cacheStats" : {
        "collBlksHit" : ...,
        "collBlksRead" : ..,
        "collHitRatio" : ...,
        "idxBlksHit" : ...,
        "idxBlksRead" : ...,
        "idxHitRatio" : ...
    },
    "lastReset" : "2022-09-02 19:41:40.471473+00",
    "ok" : 1,
    "operationTime" : Timestamp(1662159707, 1)
}
```

이 stats 명령은 Mongo API를 통해 삽입, 업데이트 및 삭제 작업에 대한 컬렉션별 카운터를 볼 때 사용해야 합니다. 컬렉션별 작업 카운터를 보는 또 다른 방법은 DML 감사를 활성화하는 것입니다. 1분 간격 동안의 모든 컬렉션에 대한 삽입, 업데이트 및 삭제 작업 수는 CloudWatch를 사용하여 Amazon DocumentDB 모니터링에서 확인할 수 있습니다.

캐시 성능을 분석하려면 어떻게 해야 하나요?

캐시 성능을 분석하면 캐시와 비교하여 디스크에서 읽은 데이터의 양을 기반으로 데이터 검색 효율성과 시스템 성능에 대한 통찰력을 얻을 수 있습니다. 캐시 성능에 대한 통찰력을 제공하기 위해 캐시 적중 수(캐시에서 읽은 데이터) 및 캐시 실패(캐시에 없고 디스크에서 읽은 데이터) 에 대한 캐시 통계를 제공합니다. 특정 컬렉션에 대한 캐시 통계는 해당 컬렉션에서 다음 명령을 실행하여 확인할 수 있습니다.

```
db.collection.stats()
```

이 명령 출력의 cacheStats 필드 값은 컬렉션에 대한 캐시 통계는 물론 컬렉션에 생성된 인덱스의 전체 캐시 통계도 제공합니다. 이러한 통계는 다음과 같습니다.

- **collBlksHit** - 이 컬렉션에 대한 작업 중에 캐시에서 읽은 블록 수입니다.
- **collBlksRead** - 이 컬렉션에 대한 작업 중에 디스크에서 읽은 블록 수(캐시 누락)입니다.
- **collHitRatio** - 이 컬렉션의 캐시 적중률(`100 * [collBlksHit / (collBlksHit + collBlksRead)]`).
- **idxBlksHit** - 이 컬렉션에서 생성된 인덱스에 대해 캐시에서 읽은 블록 수입니다.
- **idxBlksRead** - 이 컬렉션에서 생성된 인덱스에 대해 디스크에서 읽은 블록 수(캐시 누락)입니다.
- **idxHitRatio** - 이 컬렉션에서 생성된 인덱스의 캐시 적중률(`100 * [idxBlksHit / (idxBlksHit + idxBlksRead)]`).
- **lastReset** - 이 통계가 마지막으로 재설정된 시간. db.collection.stats()에 의해 제공된 통계는 클러스터를 시작/중지하거나 인스턴스를 스케일 업/다운할 때 재설정됩니다.

indexStats 명령을 사용하여 각 인덱스의 idxBlksHit 및 idxBlksRead 필드의 분류를 확인할 수도 있습니다. 인덱스별 캐시 통계는 다음 명령을 실행하여 확인할 수 있습니다.

```
db.collection.aggregate([{$indexStats:{}}]).pretty()
```

각 인덱스에 대해 cacheStats 필드 아래에서 다음과 같은 캐시 통계를 찾을 수 있습니다.

- **blksHit** - 이 인덱스에 대해 캐시에서 읽은 블록 수입니다.
- **blksRead** - 이 인덱스에 대해 디스크에서 읽은 블록 수입니다.
- **blksHitRatio** - `100 * [blksHit / (blksHit + blksRead)]`에서 캐시 적중률을 소수점 네 자리로 반올림하여 계산합니다.

장시간 실행 중이거나 차단된 쿼리를 찾아서 종료하려면 어떻게 해야 하나요?

사용자 쿼리는 최적화되지 않은 쿼리 계획으로 인해 실행 속도가 느려지거나 리소스 경합으로 인해 차단될 수 있습니다.

최적이지 않은 쿼리 계획으로 인해 느려지고 오래 실행되는 쿼리나, 속도가 느려지거나 리소스 경합으로 인해 차단된 쿼리를 찾으려면 currentOp 명령을 사용하세요. 명령을 필터링하면 종료할 쿼리의 목록을 좁힐 수 있습니다. 쿼리를 종료하려면 장시간 실행 중인 쿼리와 연결된 opid가 필요합니다.

다음 쿼리는 currentOp 명령을 사용하여 차단되었거나 10초 이상 실행 중인 모든 쿼리를 나열합니다.

```
db.adminCommand({
    aggregate: 1,
    pipeline: [
        {$currentOp: {}},
        {$match:
            {$or: [
                {secs_running: {$gt: 10}},
                {WaitState: {$exists: true}}]}},
        {$project: {_id:0, opid: 1, secs_running: 1}}],
    cursor: {}
});
```

다음으로, 쿼리 결과를 좁혀 10초를 초과하여 실행 중인 쿼리의 opid를 찾아 종료할 수 있습니다.

10초를 초과하여 실행 중인 쿼리를 찾아 종료하려면

1. 쿼리의 opid를 찾습니다.

   ```
   db.adminCommand({
       aggregate: 1,
       pipeline: [
           {$currentOp: {}},
           {$match:
               {$or:
                   [{secs_running: {$gt: 10}},
                    {WaitState: {$exists: true}}]}}],
       cursor: {}
   });
   ```

   이 작업의 출력은 다음과 같습니다(JSON 형식).

   ```
   {
       "waitedMS" : NumberLong(0),
       "cursor" : {
           "firstBatch" : [
               {
                   "opid" : 24646,
                   "secs_running" : 12
               }
   ```

```
        ],
        "id" : NumberLong(0),
        "ns" : "admin.$cmd"
    },
    "ok" : 1
}
```

2. killOp 작업을 사용하여 쿼리를 종료합니다.

```
db.adminCommand({killOp: 1, op: 24646});
```

쿼리 계획을 보고 쿼리를 최적화하려면 어떻게 해야 하나요?

쿼리의 실행 속도가 느린 경우 이 상황은 쿼리 실행 시 모음 전체 검색하여 관련 문서를 선택해야 하기 때문에 발생했을 수 있습니다. 경우에 따라 적절한 인덱스를 생성하면 쿼리 실행 속도가 빨라질 수도 있습니다. 이 시나리오를 찾아내어 인덱스를 생성할 기준 필드를 결정하려면 explain 명령을 사용합니다.

> **Note**
>
> Amazon DocumentDB는 분산, 내결함성, 자가 치유 스토리지 시스템을 활용하는 목적으로 만들어진 목적별 데이터베이스 엔진에서 MongoDB 3.6 API를 에뮬레이션합니다. 그 결과, 쿼리 계획과 explain()의 출력은 Amazon DocumentDB와 MongoDB 간에 다를 수 있습니다. 쿼리 계획을 제어하려는 고객은 $hint 연산자를 사용하여 기본 인덱스를 선택할 수 있습니다.

다음과 같이 explain 명령에서 개선하려는 쿼리를 실행합니다.

```
db.runCommand({explain: {<query document>}})
```

예는 다음과 같습니다.

```
db.runCommand({explain:{
    aggregate: "sample-document",
    pipeline: [{$match: {x: {$eq: 1}}}],
    cursor: {batchSize: 1}}
});
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "db.test",
        "winningPlan" : {
            "stage" : "COLLSCAN"
        }
    },
    "serverInfo" : {
        "host" : "...",
        "port" : ...,
        "version" : "..."
    },
    "ok" : 1
}
```

위의 출력은 $match 단계에서 전체 모음을 검색하고 각 문서의 "x" 필드가 1인지 확인해야 함을 나타냅니다. 모음에 여러 문서가 있는 경우 모음 검색 및 전체 쿼리 성능이 매우 느려질 수 있습니다. 그러므로 explain 명령의 출력에 "COLLSCAN"이 있는 경우 이는 적정 인덱스를 생성하여 쿼리 성능을 향상시킬 수 있음을 나타냅니다.

이 예제에서 쿼리는 모든 문서에서 "x" 필드가 1인지 확인합니다. 따라서 "x" 필드에서 인덱스를 작성하면 쿼리가 전체 컬렉션 스캔을 피하고 인덱스를 사용하여 관련 문서를 더 빨리 반환할 수 있습니다.

"x" 필드에 인덱스를 생성한 후 explain 출력은 다음과 같습니다.

```
{
    "queryPlanner" : {
         "plannerVersion" : 1,
         "namespace" : "db.test",
         "winningPlan" : {
             "stage" : "IXSCAN",
             "indexName" : "x_1",
             "direction" : "forward"
         }
    },
    "serverInfo" : {
        "host" : "...",
        "port" : ...,
        "version" : "..."
    },
```

```
    "ok" : 1
}
```

그러므로 "x" 필드에 대한 인덱스를 생성하면 $match 단계에서 인덱스 검색을 통해 조건자 "x = 1"을 평가해야 하는 문서 수를 줄일 수 있습니다.

작은 컬렉션에서는 성능 이득이 무시할 수준인 경우 Amazon DocumentDB 쿼리 프로세서는 인덱스를 사용하지 않도록 결정할 수 있습니다.

엘라스틱 클러스터에서 쿼리 계획을 확인하려면 어떻게 해야 하나요?

엘라스틱 클러스터의 쿼리 계획을 검사하려면 explain 명령을 사용합니다. 다음은 샤딩된 컬렉션을 대상으로 하는 찾기 쿼리의 예제 explain 작업입니다.

```
db.runCommand(
   {
     explain: { find: "cities", filter: {"name": "Seoul"}}
   }
)
```

> **Note**
>
> Amazon DocumentDB는 특별히 구축된 목적별 데이터베이스 엔진에서 MongoDB를 에뮬레이션합니다. 그 결과, 쿼리 계획과 explain()의 출력은 Amazon DocumentDB와 MongoDB 간에 다를 수 있습니다. $hint 연산자를 사용하여 기본 인덱스를 선택하도록 하여 쿼리 계획을 제어할 수 있습니다.

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
  "queryPlanner" : {
    "elasticPlannerVersion" : 1,
    "winningPlan" : {
      "stage" : "SINGLE_SHARD",
      "shards" : [
        {
          "plannerVersion" : 1,
          "namespace" : "population.cities",
          "winningPlan" : {
            "stage" : "SHARD_MERGE",
```

```
            "shards" : [
              {
                "shardName" : "f2cf5cfd-fe9c-40ca-b4e5-298ca0d11111",
                "plannerVersion" : 1,
                "namespace" : "population.cities",
                "winningPlan" : {
                  "stage" : "PARTITION_MERGE",
                  "inputStages" : [
                    {
                      "stage" : "COLLSCAN",
                      "partitionCount" : 21
                    }
                  ]
                }
              },
              {
                "shardName" : "8f3f80e2-f96c-446e-8e9d-aab8c7f22222",
                "plannerVersion" : 1,
                "namespace" : "population.cities",
                "winningPlan" : {
                  "stage" : "PARTITION_MERGE",
                  "inputStages" : [
                    {
                      "stage" : "COLLSCAN",
                      "partitionCount" : 21
                    }
                  ]
                }
              },
              {
                "shardName" : "32c5a06f-1b2b-4af1-8849-d7c4a033333",
                "plannerVersion" : 1,
                "namespace" : "population.cities",
                "winningPlan" : {
                  "stage" : "PARTITION_MERGE",
                  "inputStages" : [
                    {
                      "stage" : "COLLSCAN",
                      "partitionCount" : 22
                    }
                  ]
                }
              }
            ]
```

```
          },
          "shardName" : "32c5a06f-1b2b-4af1-8849-d7c4a0f3fb58"
        }
      ]
    }
  },
  "serverInfo" : {
    "host" : "example-4788267630.us-east-1.docdb-elastic.amazonaws.com:27017",
    "version" : "5.0.0"
  },
  "ok" : 1,
  "operationTime" : Timestamp(1695097923, 1)
}
```

위 출력은 3개 샤드 클러스터의 find 쿼리에 대한 쿼리 계획을 보여줍니다. 각 샤드에는 입력 단계가 서로 다를 수 있는 데이터 파티션이 여러 개 있습니다. 이 예시에서는 결과가 각 샤드의 'PARTITION_MERGE' 단계에서 병합되기 전에 모든 파티션에서 'COLLSCAN'(컬렉션 스캔)이 실행됩니다. 그런 다음 샤드 전체의 결과가 'SHARD_MERGE' 단계에서 병합된 후 클라이언트로 다시 전송됩니다.

인스턴스에서 실행 중인 작업을 모두 나열하려면 어떻게 해야 하나요?

사용자 또는 기본 사용자로서, 진단 및 문제 해결을 위해 인스턴스에서 실행 중인 모든 현재 작업을 나열해야 하는 경우가 흔히 있습니다. (사용자 관리에 대한 자세한 내용은 Amazon DocumentDB 사용자 관리 섹션을 참조하세요.)

mongo 쉘을 사용하여 다음 쿼리를 통해 Amazon DocumentDB 인스턴스에서 실행 중인 모든 작업을 나열할 수 있습니다.

```
db.adminCommand({currentOp: 1, $all: 1});
```

이 쿼리는 현재 인스턴스에서 실행 중인 모든 사용자 쿼리 및 내부 시스템 작업의 전체 목록을 반환합니다.

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "inprog" : [
        {
            "desc" : "INTERNAL"
        },
        {
```

```
            "desc" : "TTLMonitor",
            "active" : false
        },
        {
            "client" : ...,
            "desc" : "Conn",
            "active" : true,
            "killPending" : false,
            "opid" : 195,
            "ns" : "admin.$cmd",
            "command" : {
                "currentOp" : 1,
                "$all" : 1
            },
            "op" : "command",
            "$db" : "admin",
            "secs_running" : 0,
            "microsecs_running" : NumberLong(68),
            "clientMetaData" : {
            "application" : {
                "name" : "MongoDB Shell"
            },
            "driver" : {
                ...
            },
            "os" : {
                ...
            }
        }
    },
    {
        "desc": "GARBAGE_COLLECTION",
        "garbageCollection": {
            "databaseName": "testdb",
            "collectionName": "testCollectionA"
        },
        "secs_running": 3,
        "microsecs_running": NumberLong(3123456)
    },
    {
        "desc": "GARBAGE_COLLECTION",
        "garbageCollection": {
            "databaseName": "testdb",
            "collectionName": "testCollectionB"
```

```
            },
            "secs_running": 4,
            "microsecs_running": NumberLong(4123456)
        }
    ],
    "ok" : 1
}
```

"desc" 필드의 유효한 값은 다음과 같습니다.

- **INTERNAL** - 커서 정리 또는 기한 경과 사용자 정리 작업 같은 내부 시스템 작업입니다.
- **TTLMonitor** - 유지 시간(TTL) 모니터 스레드입니다. 실행 중 상태가 "active" 필드에 반영됩니다.
- **GARBAGE_COLLECTION** - 내부 가비지 수집기 스레드입니다.
- **CONN** - 사용자 쿼리입니다.
- **CURSOR** - 이 작업은 사용자가 'getMore' 명령을 호출하여 다음 일괄 결과를 가져오기를 기다리는 유휴 상태의 커서입니다. 이 상태에서는 커서가 메모리를 소비하지만 컴퓨팅은 소비하지 않습니다.

이전 출력에는 시스템에서 실행 중인 모든 사용자 쿼리도 나열됩니다. 각 사용자 쿼리는 데이터베이스 및 모음의 컨텍스트에서 실행되며 이들의 결합을 네임스페이스라고 합니다. 각 사용자 쿼리의 네임스페이스는 "ns" 필드에서 사용할 수 있습니다.

때로는 특정 네임스페이스에서 실행 중인 모든 사용자 쿼리를 나열해야 합니다. 따라서 "ns" 필드에서 이전 출력을 필터링해야 합니다. 필터링할 출력을 얻기 위한 예제 쿼리는 다음과 같습니다. 쿼리는 데이터베이스 "db" 및 "test" 컬렉션(즉, "db.test" 네임스페이스)에서 현재 실행 중인 모든 사용자 쿼리를 나열합니다.

```
db.adminCommand({aggregate: 1,
    pipeline: [{$currentOp: {allUsers: true, idleConnections: true}},
               {$match: {ns: {$eq: "db.test"}}}],
    cursor: {}
});
```

시스템의 기본 사용자는 모든 사용자의 쿼리와 및 모든 내부 시스템 작업을 볼 수 있습니다. 다른 모든 사용자는 각자의 쿼리만 볼 수 있습니다.

총 쿼리 수와 내부 시스템 작업 수가 기본 배치 커서 크기를 초과하는 경우 mongo 쉘에서 반복자 객체 'it'이 자동으로 생성되어 나머지 결과를 표시합니다. 모든 결과가 표시될 때까지 'it' 명령을 계속 실행합니다.

쿼리 진행 상황을 어떻게 알 수 있나요?

사용자 쿼리는 최적화되지 않은 쿼리 계획으로 인해 실행 속도가 느려지거나 리소스 경합으로 인해 차단될 수 있습니다. 이러한 쿼리를 디버깅하는 작업은 다단계 프로세스로 경우에 따라서는 동일 단계를 여러 번 실행해야 할 수도 있습니다.

디버깅의 첫 번째 단계는 장시간 실행 중이거나 차단된 모든 쿼리를 나열하는 것입니다. 다음 쿼리는 실행 시간이 10초를 넘었거나 리소스 대기 중인 모든 사용자 쿼리를 나열합니다.

```
db.adminCommand({aggregate: 1,
                 pipeline: [{$currentOp: {}},
                            {$match: {$or: [{secs_running: {$gt: 10}},
                                            {WaitState: {$exists: true}}]}},
                            {$project: {_id:0,
                                        opid: 1,
                                        secs_running: 1,
                                        WaitState: 1,
                                        blockedOn: 1,
                                        command: 1}}],
                 cursor: {}
                });
```

위의 쿼리를 주기적으로 반복 실행하여 쿼리 목록의 변경 여부를 확인하고 장시간 실행 중이거나 차단된 쿼리를 찾습니다.

해당 쿼리의 출력 문서에 `WaitState` 필드가 있는 경우 이는 쿼리 실행 속도가 느리거나 쿼리 실행이 차단된 이유가 리소스 경합 때문임을 나타냅니다. 리소스 경합은 I/O, 내부 시스템 작업 또는 다른 사용자 쿼리가 원인일 수 있습니다.

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "waitedMS" : NumberLong(0),
    "cursor" : {
        "firstBatch" : [
            {
                "opid" : 201,
                "command" : {
                    "aggregate" : ...
                },
                "secs_running" : 208,
```

```
                "WaitState" : "IO"
            }
        ],
        "id" : NumberLong(0),
        "ns" : "admin.$cmd"
    },
    "ok" : 1
}
```

동일 인스턴스에 대해 동시에 여러 모음에 대한 여러 쿼리가 있거나 쿼리가 실행 중인 데이터 세트에 비해 인스턴스 크기가 너무 작은 경우 IO가 병목 요인일 수 있습니다. 쿼리가 읽기 전용 쿼리인 경우 이전 상황에 대한 완화 방법은 여러 복제본에 대해 각 모음별로 쿼리를 분리하는 것입니다. 다양한 모음을 동시에 업데이트하거나 인스턴스 크기가 데이터 세트에 비해 너무 작은 경우 완화 방법은 인스턴스 크기를 스케일 업하는 것입니다.

리소스 경합이 다른 사용자 쿼리로 인해 발생하는 경우 출력 문서의 "blockedOn" 필드에 이 쿼리에 영향을 주는 쿼리의 "opid"가 있습니다. 모든 쿼리의 "WaitState" 및 "blockedOn" 필드 체인 뒤에 "opid"를 사용하여 체인의 헤드에서 쿼리를 찾습니다.

체인의 헤드에 있는 작업이 내부 작업이면 이 경우 유일한 완화 방법은 쿼리를 종료했다가 잠시 후에 다시 실행하는 방법뿐입니다.

아래에는 다른 작업이 소유하는 모음 잠금에서 찾기 쿼리가 차단된 샘플 출력이 나와 있습니다.

```
{
    "inprog" : [
        {
            "client" : "...",
            "desc" : "Conn",
            "active" : true,
            "killPending" : false,
            "opid" : 75,
            "ns" : "...",
            "command" : {
                "find" : "...",
                "filter" : {

                }
            },
            "op" : "query",
            "$db" : "test",
            "secs_running" : 9,
```

```
            "microsecs_running" : NumberLong(9449440),
            "threadId" : 24773,
            "clientMetaData" : {
                "application" : {
                   "name" : "MongoDB Shell"
                },
                "driver" : {
                    ...
                },
                "os" : {
                    ...
                }
            },
            "WaitState" : "CollectionLock",
            "blockedOn" : "INTERNAL"
        },
        {
            "desc" : "INTERNAL"
        },
        {
            "client" : "...",
            ...
            "command" : {
                "currentOp" : 1
            },
            ...
        }
    ],
    "ok" : 1
}
```

"WaitState"의 값이 "Latch", "SystemLock", "BufferLock", "BackgroundActivity" 또는 "Other"이면 리소스 경합의 원인은 내부 시스템 작업입니다. 이 상황이 장시간 지속되는 경우 유일한 완화 방법은 쿼리를 종료했다가 나중에 다시 실행하는 방법뿐입니다.

시스템이 갑자기 느리게 실행되는 이유를 어떻게 알 수 있나요?

다음은 시스템 속도가 느려지는 몇 가지 일반적인 이유입니다.

- 동시 쿼리 간 과도한 리소스 경합
- 시간이 지남에 따라 증가하는 활성 동시 쿼리 수
- "GARBAGE_COLLECTION" 같은 내부 시스템 작업

시간 경과에 따른 시스템 사용량을 모니터링하려면 다음 "currentOp" 쿼리를 주기적으로 실행하고 그 결과를 외부 저장소로 출력합니다. 이 쿼리는 시스템의 각 네임스페이스에서 쿼리 및 작업 수를 셉니다. 그 다음에는 시스템 사용량 결과를 분석하여 시스템에 대한 부하를 파악하고 그에 따른 적절한 조치를 취할 수 있습니다.

```
db.adminCommand({aggregate: 1,
                 pipeline: [{$currentOp: {allUsers: true, idleConnections: true}},
                            {$group: {_id: {desc: "$desc", ns: "$ns", WaitState:
 "$WaitState"}, count: {$sum: 1}}}],
                 cursor: {}
                });
```

이 쿼리는 각 네임스페이스에서 실행 중인 모든 쿼리, 모든 내부 시스템 작업, 그리고 네임스페이스별 고유한 대기 상태 수의 집계 값을 반환합니다.

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{
    "waitedMS" : NumberLong(0),
    "cursor" : {
        "firstBatch" : [
            {
                "_id" : {
                    "desc" : "Conn",
                    "ns" : "db.test",
                    "WaitState" : "CollectionLock"
                },
               "count" : 2
            },
            {
                "_id" : {
                    "desc" : "Conn",
                    "ns" : "admin.$cmd"
                },
                "count" : 1
            },
            {
                "_id" : {
                    "desc" : "TTLMonitor"
                },
                "count" : 1
            }
```

```
        ],
        "id" : NumberLong(0),
        "ns" : "admin.$cmd"
    },
    "ok" : 1
}
```

이전 출력에서, 모음 잠금에 대해 차단된 네임스페이스 "db.test"에 사용자 쿼리 2개가 있고, 네임스페이스 "admin.$cmd"의 쿼리 1개, 그리고 내부 "TTLMonitor" 작업 1개가 있습니다.

출력에 차단 대기 상태인 쿼리가 여러 개 있으면 장시간 실행 중이거나 차단된 쿼리를 찾아서 종료하려면 어떻게 해야 하나요? 섹션을 참조합니다.

하나 이상의 클러스터 인스턴스에서 높은 CPU 사용률의 원인을 어떻게 확인하나요?

다음 섹션은 인스턴스 CPU 사용률이 높은 원인을 식별하는 데 도움이 될 수 있습니다. 워크로드에 따라 결과가 달라질 수 있습니다.

- 인스턴스가 갑자기 느리게 실행되는 이유를 확인하려면 시스템이 갑자기 느리게 실행되는 이유를 어떻게 알 수 있나요? 섹션을 참조하세요.
- 특정 인스턴스에서 장기 실행 쿼리를 식별하고 종료하려면 장시간 실행 중이거나 차단된 쿼리를 찾아서 종료하려면 어떻게 해야 하나요? 섹션을 참조합니다.
- 쿼리가 진행 중인지 알아보려면 쿼리 진행 상황을 어떻게 알 수 있나요? 섹션을 참조하세요.
- 쿼리를 실행하는 데 시간이 오래 걸리는 이유를 확인하려면 쿼리 계획을 보고 쿼리를 최적화하려면 어떻게 해야 하나요? 섹션을 참조하세요.
- 시간의 경과에 따라 장기 실행 쿼리를 추적하려면 Amazon DocumentDB 작업 프로파일링 섹션을 참조합니다.

높은 인스턴스 CPU 사용률의 이유에 따라 다음 중 하나 이상을 수행하면 도움이 될 수 있습니다.

- 기본 인스턴스의 CPU 사용률이 높지만 복제본 인스턴스는 그렇지 않은 경우 클라이언트 읽기 기본 설정(예: `secondaryPreferred`)을 통해 복제본에 읽기 트래픽을 분산시키는 것을 고려하세요. 자세한 내용은 Amazon DocumentDB에 복제본 세트로 연결 단원을 참조하십시오.

  읽기에 복제본을 사용하면 기본 인스턴스가 더 많은 쓰기 트래픽을 처리할 수 있게 하여 클러스터 리소스를 더 잘 활용할 수 있습니다. 복제본에서의 읽기는 최종적으로 일관됩니다.

- 높은 CPU 사용률이 쓰기 워크로드의 결과인 경우 클러스터 인스턴스의 크기를 더 큰 인스턴스 유형으로 변경하면 워크로드를 처리하는 데 사용할 수 있는 CPU 코어 수가 증가합니다. 자세한 내용은 Amazon DocumentDB 인스턴스 관리 및 인스턴스 클래스 사양 섹션을 참조하세요.
- 모든 클러스터 인스턴스의 CPU 사용률이 높고 워크로드에서 읽기 전용 복제본을 사용하는 경우 클러스터에 복제본을 더 추가하면 읽기 트래픽에 사용할 수 있는 리소스가 늘어납니다. 자세한 내용은 클러스터에 Amazon DocumentDB 인스턴스 추가 단원을 참조하십시오.

인스턴스에서 열려 있는 커서를 어떻게 확인하나요?

Amazon DocumentDB 인스턴스에 연결되면 `db.runCommand("listCursors")` 명령을 사용하여 해당 인스턴스에서 열려 있는 커서를 나열할 수 있습니다. 인스턴스 유형에 따라 지정된 Amazon DocumentDB 인스턴스에서 언제든지 최대 4,560개의 활성 커서가 열려 있는 할당량이 있습니다. 커서는 인스턴스의 리소스를 활용하므로 더 이상 사용되지 않는 커서를 닫습니다. 자세한 내용은 Amazon DocumentDB 할당량 단원을 참조하십시오.

```
db.runCommand("listCursors")
```

현재 Amazon DocumentDB 엔진 버전은 어떻게 확인하나요?

현재 Amazon DocumentDB 엔진 버전을 확인하려면 다음 명령을 실행합니다.

```
db.runCommand({getEngineVersion: 1})
```

이 작업의 출력은 다음과 같습니다(JSON 형식).

```
{ "engineVersion" : "2.x.x", "ok" : 1 }
```

> **Note**
>
> Amazon DocumentDB 3.6의 엔진 버전은 1.x.x이고, Amazon DocumentDB 4.0의 엔진 버전은 2.x.x이고, Amazon DocumentDB 5.0의 엔진 버전은 3.x.x이고, Amazon DocumentDB 8.0의 엔진 버전은 4.x.x입니다.

인덱스 사용량을 분석하고 사용하지 않는 인덱스를 식별하려면 어떻게 해야 하나요?

지정된 컬렉션에 대한 인덱스를 식별하려면 다음 명령을 실행합니다.

```
db.collection.getIndexes()
```

`collStats` 및 `indexStats` 명령을 사용하여 컬렉션에서 작업을 수행하는 동안 사용되는 인덱스의 양을 분석할 수 있습니다. 인덱스를 사용하여 수행한 총 스캔 수(인덱스 스캔)를 인덱스 없이 수행한 스캔 수(컬렉션 스캔)와 비교하여 보려면 다음 명령을 실행합니다.

```
db.collection.stats()
```

이 명령의 출력에는 다음과 같은 값이 포함됩니다.

- **`idxScans`**- 인덱스를 사용하여 이 컬렉션에 대해 수행한 스캔 횟수.
- **`collScans`**- 인덱스를 사용하지 않고 이 컬렉션에 대해 수행한 스캔 횟수. 이러한 스캔에서는 컬렉션에 있는 문서를 한 번에 하나씩 살펴봐야 했을 것입니다.
- **`lastReset`** - 이 카운터를 마지막으로 재설정한 시간. 이 명령으로 제공된 통계는 클러스터를 시작/중지하거나 인스턴스를 스케일 업/다운할 때 재설정됩니다.

각 인덱스의 사용량에 대한 분석은 다음 명령의 출력에서 확인할 수 있습니다. 사용하지 않는 인덱스를 정기적으로 식별 및 제거하여 성능을 개선하고 비용을 절감하는 것이 가장 좋습니다. 인덱스를 유지 관리하는 데 사용되는 불필요한 컴퓨팅, 스토리지 및 I/O를 제거하기 때문입니다.

```
db.collection.aggregate([{$indexStats:{}}]).pretty()
```

이 명령의 출력은 컬렉션에 생성된 각 인덱스에 대해 다음 값을 제공합니다.

- **`ops`** - 인덱스를 사용한 작업 수. 작업 로드가 충분히 오랫동안 실행 중이고 작업 로드가 정상 상태에 있다고 확신하는 경우 `ops` 값이 0이면 인덱스가 전혀 사용되지 않음을 나타냅니다.
- **`numDocsRead`**- 이 인덱스를 사용하여 작업을 수행하는 동안 읽은 문서 수.
- **`since`** - Amazon DocumentDB에서 인덱스 사용량에 대한 통계를 수집하기 시작한 이후의 시간으로서, 일반적으로 마지막 데이터베이스 재시작 또는 유지 관리 작업 이후의 값.
- **`size`** - 이 인덱스의 크기(바이트).

다음은 위 명령을 실행한 출력 예제입니다.

```
{
    "name" : "_id_",
    "key" : {
        "_id" : 1
    },
    "host" : "example-host.com:12345",
    "size" : NumberLong(...),
    "accesses" : {
        "ops" : NumberLong(...),
        "docsRead" : NumberLong(...),
        "since" : ISODate("...")
    },
    "cacheStats" : {
        "blksRead" : NumberLong(...),
        "blksHit" : NumberLong(...),
        "hitRatio" : ...
    }
}
{
    "name" : "x_1",
    "key" : {
        "x" : 1
    },
    "host" : "example-host.com:12345",
    "size" : NumberLong(...),
    "accesses" : {
        "ops" : NumberLong(...),
        "docsRead" : NumberLong(...),
        "since" : ISODate("...")
    },
    "cacheStats" : {
        "blksRead" : NumberLong(...),
        "blksHit" : NumberLong(...),
        "hitRatio" : ...
    }
}
```

컬렉션의 전체 인덱스 크기를 확인하려면 다음 명령을 실행합니다.

```
db.collection.stats()
```

사용되지 않는 인덱스를 삭제하려면 다음 명령을 실행합니다.

```
db.collection.dropIndex("indexName")
```

누락된 인덱스는 어떻게 식별하나요?

Amazon DocumentDB 프로파일러를 사용하여 느린 쿼리를 로깅할 수 있습니다. 느린 쿼리 로그에 반복적으로 나타나는 쿼리는 해당 쿼리의 성능을 향상하기 위해 추가 인덱스가 필요함을 나타낼 수 있습니다.

하나 이상의 단계를 수행하는 단계가 하나 이상 있는 장기 실행 쿼리를 찾아 유용한 인덱스의 기회를 식별할 수 있습니다. 즉COLLSCAN, 쿼리 단계는 쿼리에 대한 응답을 제공하기 위해 컬렉션의 모든 문서를 읽어야 합니다.

다음 예제에서는 대규모 컬렉션에서 실행된 택시 타기 컬렉션에 대한 쿼리를 보여 줍니다.

```
db.rides.count({"fare.totalAmount":{$gt:10.0}}))
```

이 예제를 실행하려면 `fare.totalAmount` 필드에 인덱스가 없으므로 쿼리는 컬렉션 스캔(즉, 컬렉션의 모든 단일 문서 읽기)을 수행해야 했습니다. 이 쿼리에 대한 Amazon DocumentDB 프로파일러의 출력은 다음과 같습니다.

```
{
    ...
    "cursorExhausted": true,
    "nreturned": 0,
    "responseLength": 0,
    "protocol": "op_query",
    "millis": 300679,
    "planSummary": "COLLSCAN",
    "execStats": {
        "stage": "COLLSCAN",
        "nReturned": "0",
        "executionTimeMillisEstimate": "300678.042"
    },
    "client": "172.31.5.63:53878",
    "appName": "MongoDB Shell",
    "user": "example"
}
```

이 예에서 쿼리 속도를 높이기 위해 아래 그림과 같이 fare.totalAmount 색인을 작성하려고 합니다.

```
db.rides.createIndex( {"fare.totalAmount": 1}, {background: true} )
```

> **Note**
>
> 포그라운드에서 생성된 인덱스(인덱스를 만들 때 {background:true} 옵션이 제공되지 않은 경우)는 독점적 인 쓰기 잠금을 통해 인덱스 빌드가 완료될 때까지 애플리케이션이 컬렉션에 데이터를 쓰지 못하게 합니다. 프로덕션 클러스터에서 인덱스를 만들 때 이러한 잠재적인 영향을 고려해야 합니다. 인덱스를 생성할 때를 설정합니다{background:true}.

일반적으로 카디널리티가 높은 필드(예: 많은 수의 고유 값)에 인덱스를 만드는 경향이 있습니다. 카디널리티가 낮은 필드에 인덱스를 만들면 사용되지 않는 큰 인덱스가 발생할 수 있습니다. Amazon DocumentDB 쿼리 최적화 프로그램은 쿼리 계획을 만들 때 컬렉션의 전체 크기와 인덱스의 선택성을 고려합니다. 인덱스가 있는 경우에도 쿼리 프로세서가 COLLSCAN을 선택하는 경우가 있습니다. 이는 인덱스를 사용하는 경우 전체 컬렉션을 스캔하는 데 비해 성능 이점이 없을 것으로 쿼리 프로세서가 예측할 때 발생합니다. 특정 인덱스를 활용하기 위해 쿼리 프로세서를 강제하려면 아래 그림과 같이 hint() 연산자를 사용할 수 있습니다.

```
db.collection.find().hint("indexName")
```

데이터베이스 수집 팽창을 확인하려면 어떻게 해야 하나요?

컬렉션 팽창은 데드 버전 또는 더 이상 사용되지 문서가 누적되거나 데이터베이스 페이지 내에 조각화되면서 컬렉션의 크기가 커질 때 발생합니다. 보고된 백분율은 향후 문서에서 사용할 수 있는 문서 공간의 양을 나타냅니다. 이 팽창은 버퍼 캐시와 스토리지 모두에서 공간을 소비합니다. 팽창을 제거하려면 덤프/복원을 통해 또는 유지 관리 기간 동안 마이그레이션 루프백 및 전환을 사용하여 컬렉션을 다시 로드해야 합니다.

Example예제

다음 명령을 실행하여 컬렉션의 미사용 스토리지를 확인합니다.

```
db.runCommand({collStats:'coll'})
```

다음과 비슷한 결과가 반환됩니다.

```
{
        "ns" : "test.coll",
        "count" : 7500,
        "size" : 23250,
        "avgObjSize" : 31,
        "storageSize" : 106496,
        "unusedStorageSize" : {
                "unusedBytes" : 16384,
                "unusedPercent" : 25.12
        },
        "compression" : {
                "enable" : false
        },
        "capped" : false,
        "nindexes" : 1,
        "totalIndexSize" : 57344,
        "indexSizes" : {
                "_id_" : 57344
        },
        "collScans" : 4,
        "idxScans" : 10000,
        "opCounter" : {
                "numDocsIns" : 1000,
                "numDocsUpd" : 0,
                "numDocsDel" : 250
        },
        "cacheStats" : {
                "collBlksHit" : 3570,
                "collBlksRead" : 8,
                "collHitRatio" : 99.7765,
                "idxBlksHit" : 12293,
                "idxBlksRead" : 6,
                "idxHitRatio" : 99.9513
        },
        "lastReset" : "2024-12-18 00:30:21.552019+00",
        "ok" : 1,
        "operationTime" : Timestamp(1734632375, 1)
}
```

유용한 쿼리 요약

다음 쿼리는 Amazon DocumentDB의 성능 및 리소스 사용률을 모니터링하는 데 유용할 수 있습니다.

- 다음 명령을 사용하여 작업 카운터, 캐시 통계, 액세스 통계, 크기 통계를 비롯한 특정 컬렉션에 대한 통계를 볼 수 있습니다.

```
db.collection.stats()
```

- 다음 명령을 사용하여 인덱스 크기, 인덱스별 캐시 통계, 인덱스 사용 통계를 포함하여 컬렉션에 생성된 각 인덱스에 대한 통계를 볼 수 있습니다.

```
db.collection.aggregate([{$indexStats:{}}]).pretty()
```

- 다음 쿼리를 사용하여 모든 활동을 나열하세요.

```
db.adminCommand({currentOp: 1, $all: 1});
```

- 다음 코드는 오래 실행되거나 차단된 모든 쿼리를 나열합니다.

```
db.adminCommand({aggregate: 1,
                 pipeline: [{$currentOp: {}},
                            {$match: {$or: [{secs_running: {$gt: 10}},
                                            {WaitState: {$exists: true}}]}},
                            {$project: {_id:0,
                                        opid: 1,
                                        secs_running: 1,
                                        WaitState: 1,
                                        blockedOn: 1,
                                        command: 1}}],
                 cursor: {}
                });
```

- 다음 코드는 쿼리를 종료합니다.

```
db.adminCommand({killOp: 1, op: <opid of running or blocked query>});
```

- 다음 코드를 사용하여 시스템 상태를 종합적으로 볼 수 있습니다.

```
db.adminCommand({aggregate: 1,
                 pipeline: [{$currentOp: {allUsers: true, idleConnections: true}},
                            {$group: {_id: {desc: "$desc", ns: "$ns", WaitState:
 "$WaitState"}, count: {$sum: 1}}}],
                 cursor: {}
                });
```

Amazon DocumentDB의 폐영역 회수

Amazon DocumentDB는 모든 업데이트 작업에 대한 문서 및 인덱스 항목의 새 버전을 생성하는 다중 버전 동시성 제어(MVCC) 데이터베이스 아키텍처를 구현합니다. 이 아키텍처는 트랜잭션 격리를 활성화하여 한 트랜잭션의 변경 사항이 다른 트랜잭션에 나타나지 않도록 합니다.

주제

- Amazon DocumentDB의 가비지 수집 이해
- 폐영역 회수 프로세스
- 스토리지 아키텍처 및 확장 스토리지
- 폐영역 회수 모니터링
- collStats 출력 예제
- 자주 묻는 질문(FAQ)

Amazon DocumentDB의 가비지 수집 이해

폐영역 회수(GC)는 Amazon DocumentDB에서 최적의 시스템 성능과 가용성을 유지하는 자동화된 백그라운드 프로세스입니다. 여러 최신 데이터베이스와 마찬가지로 Amazon DocumentDB의 MVCC 아키텍처는 업데이트마다 새 문서 및 인덱스 버전을 생성합니다. 각 쓰기 작업은 유한 카운터에서 고유한 MVCC ID를 사용합니다. 이러한 IDs 문서 버전이 속한 트랜잭션과 커밋 또는 롤백 여부를 식별합니다. 시간이 지남에 따라 이러한 이전 버전과 해당 MVCC IDs 누적되므로 성능 저하를 방지하기 위해 정리가 필요합니다.

폐영역 회수의 함수

폐영역 회수기는 세 가지 필수 함수를 제공합니다.

- 스토리지 공간 회수 - 활성 쿼리에 더 이상 필요하지 않은 더 이상 사용되지 않는 문서 및 인덱스 버전을 제거하여 향후 쓰기 작업을 위한 공간을 확보합니다.
- MVCC ID 오버플로 방지 - MVCC ID의 유한 카운터를 관리하여 MVCC ID 오버플로를 방지합니다. 이 관리가 없으면 카운터가 결국 한도에 도달하여 데이터베이스를 ID가 재활용될 때까지 임시 읽기 전용 모드로 강제 적용합니다,
- 쿼리 성능 유지 - 쿼리 처리가 누적되고 느려지는 데드 문서 버전을 제거하여 최적의 쿼리 성능을 유지합니다.

폐영역 회수 프로세스

GC 프로세스는 컬렉션별로 작동하며 여러 컬렉션에서 여러 프로세스를 동시에 실행할 수 있습니다. 이 프로세스는 4개의 순차적 단계로 구성됩니다.

1. 식별 - 시스템은 활성 트랜잭션 또는 쿼리에서 더 이상 참조하지 않는 문서 및 인덱스 버전을 식별합니다.
2. 메모리 로드 - 문서와 인덱스 항목이 아직 없는 경우 이전 문서와 인덱스 항목이 메모리에 로드됩니다.
3. 삭제 - 더 이상 사용되지 않는 버전이 영구적으로 삭제되어 스토리지 공간을 회수합니다.
4. MVCC ID 재활용 - 시스템은 새 작업을 위해 삭제된 버전에서 MVCC ID를 재활용합니다.

폐영역 회수가 이전 문서 버전 처리를 완료하면 시스템에서 가장 오래된 MVCC ID가 제거됩니다. 이 정리는 MVCC ID를 재활용하여 클러스터 전체에서 새 쓰기 작업에 사용할 수 있도록 하여 MVCC ID 오버플로를 방지하는 데 매우 중요합니다. 이 재활용 프로세스가 없으면 시스템은 결국 유한 MVCC ID 카운터를 소진하고 읽기 전용 상태가 됩니다.

폐영역 회수 일정

폐영역 회수는 주기적으로 백그라운드에서 자동으로 실행됩니다. 타이밍 및 빈도는 시스템 로드, 사용 가능한 리소스, 쓰기 볼륨 및 MVCC ID 소비 수준에 따라 동적으로 조정됩니다. 쓰기 작업이 많은 동안에는 GC 프로세스가 더 자주 실행되어 문서 버전의 수가 증가합니다.

스토리지 아키텍처 및 확장 스토리지

Amazon DocumentDB는 문서 스토리지를 두 개의 개별 세그먼트로 분리하는 정교한 스토리지 아키텍처를 사용합니다.

기본 스토리지 세그먼트

기본 스토리지 세그먼트에는 기본 문서 데이터와 메타데이터가 포함되어 있습니다. 이 세그먼트는 다음을 저장합니다.

- 표준 페이지 크기(8KB)에 맞는 콘텐츠를 문서화합니다.
- 메타데이터 및 구조 정보를 문서화합니다.
- 기본 인덱스 및 해당 항목.
- 컬렉션 수준 통계 및 구성.

확장 스토리지 세그먼트

확장 스토리지 세그먼트는 표준 스토리지 페이지 크기를 초과하는 문서를 처리하도록 설계된 특수 대형 문서 객체 스토어를 활용합니다. 이 세그먼트는 다음을 제공합니다.

- 효율적인 대용량 문서 처리 - 기본 스토리지 임계값보다 큰 문서는 확장 스토리지 세그먼트로 자동으로 이동됩니다.
- 최적화된 스토리지 레이아웃 - 세그먼트는 큰 객체에 최적화된 다른 스토리지 형식을 사용하여 조각화를 줄이고 액세스 패턴을 개선합니다.
- 독립 가비지 수집 - 확장 스토리지 세그먼트에는 기본 스토리지 정리와 독립적으로 실행할 수 있는 자체 가비지 수집 프로세스가 있습니다.
- 투명한 액세스 - 애플리케이션이 데이터가 포함된 스토리지 세그먼트를 알 필요 없이 대용량 문서에 원활하게 액세스합니다.

확장 스토리지 세그먼트는 다음과 같은 경우에 특히 유용합니다.

- 대용량 임베디드 배열이 포함된 문서가 포함된 컬렉션입니다.
- 광범위한 중첩 구조가 있는 문서입니다.
- 바이너리 데이터 또는 큰 텍스트 필드를 저장하는 컬렉션입니다.
- 일부 문서가 평균 크기를 크게 초과하는 혼합 문서 크기가 있는 애플리케이션.

폐영역 회수 모니터링

클러스터 수준 지표

`AvailableMVCCIds`

- 위치 - Amazon CloudWatch
- 설명 - 최대 한도 18억에서 사용 가능한 나머지 쓰기 작업 수를 보여주는 카운터입니다. 이 카운터가 0에 도달하면 ID가 회수되고 재활용될 때까지 클러스터가 읽기 전용 모드로 전환됩니다. 카운터는 각 쓰기 작업에 따라 감소하고 폐영역 회수가 이전 MVCC ID를 재활용함에 따라 증가합니다.
- 권장 사항 - Amazon DocumentDB 5.0 이상에서 값이 9억 개 미만으로 떨어지거나 Amazon DocumentDB 3.6 및 4.0에서 13억 개 미만으로 떨어지면 경보를 설정합니다. 이 조기 경고를 사용하면 나중에 설명하는 권장 단계를 수행할 수 있습니다.

`LongestActiveGCRuntime`

- 위치 - Amazon CloudWatch
- 설명 - 가장 긴 활성 폐영역 회수 프로세스의 초 단위 기간입니다. 1분마다 업데이트하고 1분 내에 완료되는 프로세스를 제외한 활성 작업만 추적합니다.
- 권장 사항 - `gcRuntimeStats` 과거 데이터와 비교하여 대량 삭제 중 연장된 런타임과 같은 비정상적인 가비지 수집 동작을 식별합니다.

컬렉션 수준 지표

`MVCCIDStats: MVCCIdScale`

- 위치 - 데이터베이스 collStats 명령
- 설명 - 0~1의 척도로 MVCC ID 수명을 측정합니다. 여기서 1은 클러스터가 읽기 전용 상태가 되기 전의 최대 수명을 나타냅니다. 이 지표를 `AvailableMVCCIds`와 함께 사용하여 클러스터를 노후화하는 가장 오래된 MVCC ID가 포함된 컬렉션을 식별합니다.
- 권장 사항 - 각 컬렉션에 대해 0.3 미만의 값을 유지합니다.

`gcRuntimeStats`

- 위치 - 데이터베이스 collStats 명령
- 설명 - 총 실행 횟수, 평균 기간, 최대 기간을 포함한 폐영역 회수 지표의 2개월 기록을 제공합니다. 의미 있는 통계를 보장하기 위해 5분 이상 지속되는 폐영역 회수 작업만 포함합니다.

`storageSizeStats`

- 위치 - 데이터베이스 collStats 명령
- 설명 - 다양한 스토리지 세그먼트의 스토리지 사용률을 자세히 분석합니다.
  - `storageSegmentBase` - 표준 문서의 기본 스토리지 세그먼트에서 사용하는 스토리지
  - `storageSegmentExtended` - 대용량 문서의 확장 스토리지 세그먼트에서 사용하는 스토리지
- 사용량 - 상당히 큰 문서 스토리지가 있는 컬렉션을 식별하고 스토리지 배포 패턴을 이해하는 데 도움이 됩니다.

`unusedStorageSize`(수집 수준)

- 위치 - 데이터베이스 collStats 명령

- 설명 - 샘플링된 통계를 기반으로 컬렉션의 미사용 스토리지 공간을 추정합니다. 여기에는 삭제된 문서와 빈 세그먼트의 공간이 포함됩니다. 지표는 결합된 합계와 세그먼트별 분석을 모두 제공합니다.
  - 모든 스토리지 세그먼트unusedPercent에서 unusedBytes 및 통합
  - storageSegmentBase - 기본 스토리지 세그먼트에서 특별히 사용되지 않는 공간
  - storageSegmentExtended - 확장 스토리지 세그먼트에서 특별히 사용되지 않는 공간

documentFragmentStats

- 위치 - 데이터베이스 collStats 명령
- 설명 - 모음 내의 문서 조각 및 데드 데이터에 대한 자세한 정보를 제공합니다. 문서 조각은 데이터베이스 엔진에서 사용하는 내부 스토리지 단위를 나타내며, 데드 조각은 더 이상 액세스할 수 없지만 아직 회수되지 않은 데이터를 나타냅니다. 이 지표에는 다음이 포함됩니다.
  - totalDocFragmentsCount - 컬렉션의 총 문서 조각 수
  - deadDocFragmentsCount - 데드(액세스할 수 없음) 데이터가 포함된 조각 수
  - deadDocFragmentsPercent - 데드 데이터가 포함된 조각의 백분율
  - deadDocFragmentBytes - 배달 못한 문서 조각에서 소비되는 예상 바이트
  - storageSegmentBase 및에 대한 세그먼트별 분석 storageSegmentExtended
- 사용량 -이 지표를 모니터링하여 가비지 수집의 효과를 이해하고 유지 관리 작업의 이점을 얻을 수 있는 컬렉션을 식별합니다. 데드 조각의 비율이 높으면 가비지 수집이 뒤처지거나 수집이 최적화의 이점을 누릴 수 있음을 나타냅니다.

인덱스 수준 지표

unusedStorageSize(인덱스 수준)

- 위치 - 데이터베이스 indexStats 명령
- 설명 - 샘플링된 통계를 기반으로 인덱스의 미사용 스토리지 공간을 추정합니다. 여기에는 더 이상 사용되지 않는 인덱스 항목과 빈 세그먼트의 공간이 포함됩니다.
- 권장 사항 - reIndex 명령을 사용하여 가동 중지 없이 인덱스를 다시 빌드하고 미사용 공간을 회수합니다. 자세한 내용은 인덱스 관리를 참조하세요.

collStats 출력 예제

다음 예제에서는 가비지 수집 및 스토리지 지표가 포함된 일반적인 collStats 출력을 보여줍니다.

```
{
    "ns" : "Mvcc_consumption_test_db.mvcc_test_collection",
    "MVCCIdStats" : {
        "MVCCIdScale" : 0.03
    },
    "gcRuntimeStats" : {
        "numRuns" : 1,
        "historicalAvgRuntime" : 3295,
        "historicalMaxRuntime" : 3295,
        "lastRuntime" : 3295,
        "lastRuntimeStart" : ISODate("2025-06-24T08:47:14Z")
    },
    "documentFragmentStats" : {
        "totalDocFragmentsCount" : 45000000,
        "deadDocFragmentsCount" : 2250000,
        "deadDocFragmentsPercent" : 5.0,
        "deadDocFragmentBytes" : 98304000,
        "storageSegmentBase" : {
            "totalDocFragmentsCount" : 30000000,
            "deadDocFragmentsCount" : 1500000,
            "deadDocFragmentsPercent" : 5.0,
            "deadDocFragmentBytes" : 65536000
        },
        "storageSegmentExtended" : {
            "totalDocFragmentsCount" : 15000000,
            "deadDocFragmentsCount" : 750000,
            "deadDocFragmentsPercent" : 5.0,
            "deadDocFragmentBytes" : 32768000
        }
    },
    "collScans" : 14,
    "count" : 30000000,
    "size" : 1320000000,
    "avgObjSize" : 44,
    "storageSize" : 6461497344,
    "storageSizeStats" : {
        "storageSegmentBase" : 4307664896,
        "storageSegmentExtended" : 2153832448
    },
    "capped" : false,
    "nindexes" : 2,
    "totalIndexSize" : 9649553408,
    "indexSizes" : {
```

```
        "_id_" : 1910661120,
        "c_1" : 7738892288
    },
    "unusedStorageSize" : {
        "unusedBytes" : 4201881600,
        "unusedPercent" : 65.05,
        "storageSegmentBase" : {
            "unusedBytes" : 2801254400,
            "unusedPercent" : 65.05
        },
        "storageSegmentExtended" : {
            "unusedBytes" : 1400627200,
            "unusedPercent" : 65.05
        }
    },
    "cacheStats" : {
        "collBlksHit" : 171659016,
        "collBlksRead" : 754061,
        "collHitRatio" : 99.5627,
        "idxBlksHit" : 692563636,
        "idxBlksRead" : 1177921,
        "idxHitRatio" : 99.8303
    },
    "idxScans" : 41823984,
    "opCounter" : {
        "numDocsIns" : 0,
        "numDocsUpd" : 20911992,
        "numDocsDel" : 0
    },
    "lastReset" : "2025-06-24 05:57:08.219711+00",
    "ok" : 1,
    "operationTime" : Timestamp(1750968826, 1)
}
```

자주 묻는 질문(FAQ)

폐영역 회수의 효율적 작동 여부를 확인하려면 어떻게 해야 하나요?

비효율적인 폐영역 회수를 나타내는 다음 경고 신호를 모니터링합니다.

- 과도한 수집 팽창 - 특히 큰 인덱스의 경우 쓰기가 많거나 대량 삭제 시 `unusedStorageSize` 지표를 지속적으로 증가시킵니다.

- 높은 데드 조각 비율 - 일관되게 높은 deadDocFragmentsPercent 값(10~15% 이상)을 documentFragmentStats 보여줍니다.
- 성능 저하된 쿼리 지연 시간 - 누적된 배달 못한 문서로 인해 쿼리 지연 시간이 늘어납니다.
- 연장된 GC 기간 -의 과거 평균보다 더 오래 걸리는 가비지 수집 작업입니다gcRuntimeStats.
- GC 처리 기능 향상 - 가비지 수집기가 시스템 요구 사항을 따라잡을 수 없음을 LongestActiveGCRuntime 나타냅니다.

폐영역 회수가 데이터베이스 성능에 영향을 미치나요?

정상적인 조건에서는 폐영역 회수가 성능에 미치는 영향을 최소화합니다. 그러나 폐영역 회수가 뒤처지면 다음과 같은 상황이 발생할 수 있습니다.

- 누적된 배달 못한 문서로 인해 스토리지 비용이 증가합니다.
- 더 이상 사용되지 않는 인덱스 항목으로 인해 쿼리 성능이 느려집니다.
- MVCC IDs가 고갈된 경우 임시 읽기 전용 모드입니다.
- 특히 더 작은 인스턴스에서 집약적인 수집 실행 중 리소스 사용량이 증가합니다.
- 대용량 문서에 대한 확장 스토리지 세그먼트 작업의 효율성이 감소했습니다.

폐영역 회수를 수동으로 트리거할 수 있나요?

아니요. Amazon DocumentDB의 폐영역 회수는 수동으로 트리거할 수 없습니다. 시스템은 내부 유지 관리 작업의 일부로 폐영역 회수를 자동으로 관리합니다.

운영 모범 사례로 설정해야 하는 경보는 무엇인가요?

클러스터 및 컬렉션 수준 모두에서 모니터링을 설정하여 Amazon DocumentDB 시스템의 성능을 최적화합니다.

클러스터 수준 모니터링의 경우 먼저 Amazon DocumentDB 5.0 이상에서는 9억, Amazon DocumentDB 3.6 및 4.0에서는 13억의 임계값으로 지표에 대한 Amazon CloudWatch 경보를 생성합니다. AvailableMVCCIds Amazon DocumentDB 이렇게 하면 지표가 0에 도달하기 전에 조치를 취할 수 있는 충분한 시간을 확보할 수 있습니다. 그러면 클러스터가 읽기 전용 모드로 전환됩니다. 이 지표는 특정 사용 패턴에 따라 변동할 수 있습니다. 일부 고객은 가비지 수집이 작업을 완료하면 임계값 아래로 떨어졌다가 15억 개 이상으로 복구되는 것을 볼 수 있습니다.

Amazon CloudWatch를 통해 LongestActiveGCRuntime 지표를 모니터링하는 것도 중요합니다. 이 지표는 gcRuntimeStats와 함께 시스템 전체에서 폐영역 회수가 얼마나 효율적으로 수행되고 있는지 이해하는 데 도움이 됩니다.

컬렉션 수준 모니터링의 경우 다음 주요 지표에 집중하세요.

- MVCCIdScale - MVCC IDs 오래되어 주의가 필요할 수 있음을 나타내는 값이 증가하는지 확인합니다.
- gcRuntimeStats - 비정상적으로 오래 걸리거나 며칠에 걸쳐 연장되는 가비지 수집 프로세스를 식별합니다.
- documentFragmentStats - deadDocFragmentsPercent 값 모니터링 - 지속적으로 높은 비율(10~15% 이상)은 가비지 수집이 뒤처지고 있음을 나타낼 수 있습니다.
- storageSizeStats 및 unusedStorageSize- 스토리지 사용률 패턴을 추적하고 두 스토리지 세그먼트에서 상당한 미사용 공간이 있는 컬렉션을 식별합니다.

쓰기 작업이 빈번한 컬렉션은 폐영역 회수기에 더 많은 작업을 생성하므로 각별한 주의가 필요합니다. 가비지 수집이 워크로드를 따라잡을 수 있도록 쓰기 활동이 많은 컬렉션에 대해 이러한 지표를 더 자주 확인하세요.

이러한 모니터링 권장 사항은 시작점 역할을 합니다. 시스템 동작에 더 익숙해지면 특정 사용 패턴 및 요구 사항에 더 잘 맞게 이러한 임계값을 조정하는 것이 좋습니다.

경보 임계값 아래로 **AvailableMVCCIds** 떨어지면 어떻게 해야 하나요?

AvailableMVCCIds 지표가 경보 임계값 아래로 떨어지면 클러스터가 읽기 전용 모드로 전환되지 않도록 즉각적인 조치를 취합니다. 먼저 인스턴스 크기를 확장하여 가비지 수집기에 더 많은 컴퓨팅 리소스를 제공합니다. 이렇게 하면 애플리케이션이 폐영역 회수기에 따라 잡는 데 필요한 추가 전력을 제공하면서 정상적인 작업을 계속할 수 있습니다.

확장만으로는 상황이 개선되지 않는 경우 쓰기 작업을 줄이는 것이 좋습니다. MVCCIdScale 지표를 사용하여 주의가 필요한 이전 MVCC ID가 포함된 특정 컬렉션을 식별합니다. 또한를 모니터링documentFragmentStats하여 가비지 수집 비효율성에 영향을 미칠 수 있는 데드 조각 비율이 높은 컬렉션을 식별합니다.

이러한 컬렉션을 식별한 후에는 가비지 수집이 따라잡을 수 있도록 컬렉션에 대한 쓰기 작업을 일시적으로 줄여야 할 수 있습니다. 복구 기간 동안 AvailableMVCCIds 지표를 면밀히 모니터링하여 작업에 원하는 효과가 있는지 확인합니다. AvailableMVCCIds 값이 15억 이상으로 반환되면 클러스터가 정상으로 간주됩니다.

이러한 단계는 시스템이 중요한 상태에 도달하기 전에 복구하는 데 도움이 되는 예방 조치입니다. 지표가 임계값 아래로 떨어지는 것을 확인한 후 더 빨리 조치를 취할수록 쓰기 작업에 미치는 영향을 피할 가능성이 높아집니다.

배치 작업이 MVCC ID 소비율에 영향을 미칩니까?

예. 단일 원자성 트랜잭션 내에서 실행된 배치는 하나의 MVCC ID만 사용하기 때문에 배치로 쓰기를 그룹화하면 MVCC IDs 사용되는 속도가 낮아집니다. 가장 좋은 방법은 updateMany 가능하면 `insertMany` 및를 사용하여 MVCC ID 소비를 최소화하고 가비지 수집에 대한 부하를 줄이는 것입니다.

가 특정 임계값에 도달할 때까지 **`AvailableMVCCIds`** 복구되지 않는 이유는 무엇입니까?

가비지 수집 재생 주기를 트리거하기에 충분한 MVCC IDs를 아직 소비하지 않은 쓰기 중심 워크로드의 경우 복구 `AvailableMVCCIds` 없이가 일시적으로 감소할 것으로 예상됩니다. 가비지 수집은 컬렉션 내 문서의 특정 부분이 더 이상 사용되지 않거나, 새 문서의 특정 부분이 삽입되거나, 재생 임계값 아래로 `AvailableMVCCIds` 떨어질 때 트리거됩니다.

가 재생 임계값 아래로 `AvailableMVCCIds` 떨어질 때까지 강제 재생은 발생하지 않습니다. 그러면 재생성이 트리거되고 복구`AvailableMVCCIds`됩니다. 그러나 장기 실행 트랜잭션과 열린 커서는 회수 기간을 고정하고 임계값에 도달한 후에도 재생을 방지할 수 있습니다.

오류 코드

명령이 실패하면 Amazon DocumentDB(MongoDB 호환)에서 숫자 오류`code`와 설명 메시지()가 포함된 오류가 발생합니다`errmsg`. 다음 표를 사용하여 Amazon DocumentDB 8.0이 클라이언트 애플리케이션에 반환할 수 있는 오류 코드를 식별합니다. 오류 코드가 단일 조건에 해당하는 경우 설명은 서버에서 반환한 메시지 텍스트와 일치합니다. 여러 조건을 포함하는 코드에는 일반적인 설명이 있습니다.

> **Important**
>
> Amazon DocumentDB가 메이저 버전 간에 오류 코드를 변경할 수 있으므로 특정 숫자 오류 코드에 의존하는 애플리케이션 로직을 빌드하지 마세요.

오류 코드	설명
1	항목을 찾을 수 없음
2	명령 인수, 옵션 또는 필드 값이 유효하지 않거나 지원되지 않습니다.
4	필수 필드가 누락되었거나 구성 값의 형식이 올바르지 않습니다.
5	뷰 정의는 자체를 참조하여 주기를 생성하거나 파일 시스템 오류가 발생했습니다.
9	명령, 필드, 힌트 또는 재개 토큰을 구문 분석할 수 없거나 형식이 잘못되었습니다.
10	대상 컬렉션을 찾을 수 없거나 컬렉션이 지정되지 않았습니다.
11	참조된 사용자를 찾을 수 없거나 대상 컬렉션이 이미 있습니다.
13	이 명령을 실행할 권한이 없거나 인증이 유효하지 않습니다.
14	필드 또는 값의 데이터 형식이 잘못되었습니다.
15	값이 평가 중에 허용된 범위를 초과했습니다.
16	쓰기 배치가 비어 있거나 형식이 잘못되었거나 허용된 작업 수를 초과합니다.
17	모든 객체의 유형이 동일한 것은 아닙니다.
18	자격 증명이 유효하지 않거나 사용자가 존재하지 않아 인증에 실패했습니다.
19	데이터 처리 기본 작업을 찾을 수 없거나 기본 유형을 찾을 수 없음
20	이 컨텍스트에서는 요청된 작업이 허용되지 않습니다.
26	대상 네임스페이스 또는 컬렉션이 존재하지 않습니다.
27	요청된 인덱스가 없거나 적합한 인덱스를 사용할 수 없습니다.
37	파이프라인 또는 집계 단계 인수가 누락되었거나 비어 있거나 유효하지 않습니다.
38	데이터베이스 이름을 찾을 수 없음
39	maxTimeMS를 초과했거나, 너무 오래 실행되었거나, 사용 가능한 메모리를 초과하여 작업이 취소되었습니다.

오류 코드	설명
40	업데이트가 다른 필드와 충돌하거나 가비지 수집에 의해 쓰기가 제한되거나 비활성화됩니다.
41	JavaScript 오류
43	지정된 커서가 더 이상 존재하지 않거나 만료되었습니다.
44	배열 인덱스가 범위를 벗어남
48	해당 이름의 컬렉션, 뷰 또는 네임스페이스가 이미 있습니다.
51	잘못된 업데이트 연산자
52	필드 이름은 스토리지에 유효하지 않은 예약된 '$' 접두사를 사용합니다.
53	_id 필드에 배열 또는 정규식과 같이 유효하지 않거나 지원되지 않는 유형이 있습니다.
54	는 설정할 쿼리 필드를 유추할 수 없습니다.
55	트랜잭션이 이미 커밋됨
56	요청의 필드 이름이 비어 있어 허용되지 않습니다.
57	다른 트랜잭션에 의해 무효화된 트랜잭션
58	필드 값이 너무 깁니다.
59	요청된 명령을 알 수 없거나 지원되지 않습니다.
61	잘못된 컬렉션
62	컬렉션을 변경할 수 없음
65	배치 운영 오류 발생
66	변경 불가능한 _id 필드를 수정하려는 업데이트가 시도되었습니다.
67	유효하지 않거나 지원되지 않는 사양으로 인해 인덱스를 생성할 수 없습니다.

오류 코드	설명
68	속성이 충돌하는 인덱스가 이미 있습니다.
70	$snippet과 함께 사용되는 필드가 snippet 열로 구성되지 않음
71	인덱스가 이미 다른 옵션으로 존재합니다.
72	명령 또는 인덱스에 지정된 하나 이상의 옵션이 잘못되었습니다.
73	네임스페이스, 데이터베이스 또는 컬렉션 이름이 유효하지 않거나이 작업에 허용되지 않습니다.
74	컬렉션에 텍스트 인덱스가 정의되어 있지 않습니다.
75	텍스트 인덱스는이 컬렉션에 대해 정의되지만이 필드에는 정의되지 않습니다.
77	정렬에 스냅샷을 사용할 수 없음
78	문서를 찾을 수 없음
80	쿼리가 허용되는 최대 크기를 초과합니다.
82	기간 형식이 잘못되었습니다. 유효한 형식: '1m' 또는 '3y' '<number> h/d/w/s/m/y'
85	해당 이름의 인덱스가 이미 존재하지만 옵션이 다릅니다.
86	요청된 인덱스가 이름이 같은 기존 인덱스와 충돌합니다.
87	$sort 키 순서는 1(오름차순) 또는 -1(내림차순)이어야 합니다.
92	작업을 위한 공간이 충분하지 않음
93	사용자가 이미 존재합니다.
94	대상 컬렉션, 클러스터 또는 리소스 권한이 유효하지 않거나 사용할 수 없습니다.
95	데이터베이스가 존재하지 않음
96	'writeConcern' 파라미터가 잘못되었습니다.
97	암호는 비워둘 수 없습니다.

오류 코드	설명
99	정의되지 않은 역할 또는 권한
102	이미 내장된 역할을 추가할 수 없음
103	컬렉션에서 유효한 역할을 추출할 수 없습니다.
105	사용자가 유효하지 않거나 권한 부여 라이브러리를 초기화할 수 없습니다.
107	데이터베이스에 다른 데이터베이스의 역할을 부여할 수 없음
111	업데이트 문서를 제공할 때 _id를 정확하게 지정해야 합니다.
112	쓰기 충돌
113	지정된 역할이 존재하지 않습니다.
114	역할이 이미 있음
115	anyResource는 지정된 경우 true여야 합니다.
116	리소스에는 db 및 컬렉션 세트, 클러스터 세트 또는 anyResource 세트가 있어야 합니다.
117	작업은 활성 트랜잭션과 같이 진행 중인 다른 작업과 충돌합니다.
120	모든 사용자를 표시하므로 'showPrivileges'를 지정할 수 없습니다.
121	문서 검증 실패
122	업데이트 오류
124	스키마 ID 처리 오류
125	종종 모호한 참조 또는 누락된 세션으로 인해 명령이 실패했습니다.
126	데이터베이스가 이미 있음
127	데이터베이스 이름이 이미 다른 대/소문자 이름에 있음
128	필드가 너무 많음

오류 코드	설명
129	데이터베이스 삭제 대기
130	잘못된 파라미터 값
136	커서 위치가 손실되었거나 필요한 스트림 변경 기능이 활성화되지 않았습니다.
139	JS 인터프리터 실패
145	쿼리에 정의되지 않은 별칭
147	마스터 아님
148	쿼리를 실행할 리소스 부족
149	쿼리에 이미 정의된 별칭
154	인수에 잘못된 값 또는 잘못된(숫자가 아닌) 유형이 있습니다.
155	NULL 값 발생
156	UNDEFINED 값 발생
157	NAN 값 발생
158	컬렉션이 이미 다른 용도로 사용 중입니다.
159	지속 시간 값이 오버플로되었거나 네트워크 작업이 실패했습니다.
161	요청된 데이터 정렬 버전이 잘못되었거나 사용할 수 없습니다.
163	필드 또는 이름이 스토리지에 유효하지 않습니다.
164	쿼리 연산자가 허용되지 않는 방식으로 중첩되었습니다.
165	보기가 최대 중첩 깊이인 20을 초과하거나 한 주기에서 자신을 참조합니다.
166	뷰에서 작업이 지원되지 않거나 너무 많은 위치 요소가 사용되었습니다.
167	요청된 작업 또는 단계는 뷰에서 지원되지 않습니다.

오류 코드	설명
168	표현식 또는 파이프라인 연산자가 인식되지 않거나 잘못된 인수가 있습니다.
169	BadValue $maxTimeMS에 통합되지 않은 값이 있음
170	BadValue $maxTimeMS가 범위를 벗어났습니다.
171	인덱스는 여러 배열 필드에 빌드할 수 없습니다.
173	건너뛰기 값은 양수여야 합니다.
174	숫자로 예상되는 로그 세부 정보
181	키 패턴은 여러 인덱스와 일치합니다. 대신 이름으로 인덱스를 지정합니다.
197	인덱스 사양에 유효하지 않거나 지원되지 않는 필드가 포함되어 있습니다.
207	uuid는 UUID(4) 하위 유형이 있는 16바이트 바이너리 필드여야 합니다.
224	현재 컨텍스트에서는이 쿼리 기능이 허용되지 않습니다.
225	세션에서 최신 트랜잭션이 시작되었으므로이 이전 트랜잭션은 진행할 수 없습니다.
241	$convert에서 값을 요청된 유형으로 변환할 수 없습니다.
250	잘못된 뷰
251	지정된 트랜잭션 번호가 활성 트랜잭션과 일치하지 않습니다.
252	재귀 한도에 도달함
253	뷰에 항목을 삽입할 수 없음
254	보기에서 항목을 삭제할 수 없음
256	$having 절로 필터링된 문서
257	트랜잭션에 있는 모든 작업의 총 크기가 허용 한도를 초과합니다.
263	이 작업은 다중 문서 트랜잭션 내에서 실행할 수 없습니다.

오류 코드	설명
291	$geoNear 쿼리에 대한 인덱스를 찾을 수 없음
300	SASL 핸드셰이크 또는 세션 오류로 인해 인증에 실패했습니다.
301	인증 메커니즘 또는 요청된 작업은 지원되지 않습니다.
302	SASL 교환을 완료하려면 인증에 추가 데이터가 필요합니다.
303	요청된 단계, 옵션 또는 역할 구성은 지원되지 않습니다.
304	요청된 집계 단계는 지원되지 않습니다.
305	집계 프로젝션 연산자는 지원되지 않습니다.
306	지정된 집계 그룹 누적기 연산자는 지원되지 않습니다.
502	열이 이미 있음
505	여기에 삽입이 허용되지 않음
601	좌표가 범위를 벗어남
602	인덱스 오류
700	라이브 모드에서는이 파라미터에 대한 변경이 허용되지 않습니다.
701	이 파라미터의 값이 잘못되었습니다.
703	이 파라미터의 잘못된 상태 전환
999	잘못된 트랜잭션 ID
4570	문자열 값을 예상하는 $lookup 파라미터
4572	$lookup에 필수 필드가 누락되었습니다.
10065	객체(예: vars 또는 $rand)여야 하는 파라미터에 객체가 아닌 값이 지정되었습니다.
10107	마스터 아님

오류 코드	설명
11000	중복 키는 고유한 인덱스 제약 조건을 위반합니다.
11601	작업이 중단되었습니다.
11603	클라이언트 요청이 손상되었거나 잘못된 구문입니다.
13103	너무 많은 복합 키
13297	이름이 같은 데이터베이스가 이미 존재하지만 문자만 다릅니다.
13388	인식할 수 없는 파이프라인 단계 이름: $collStats
13435	마스터가 아님 및 slaveOk=false
14031	디스크 공간이 충분하지 않음
15888	는 생성할 컬렉션 이름을 전달해야 합니다.
15947	그룹의 필드는 객체에 지정되어야 합니다.
15950	그룹 집계 필드 이름은 연산자가 될 수 없습니다.
15952	알 수 없는 $group 연산자
15955	그룹 집계 연산자의 구문 오류: _id 필드를 지정해야 합니다.
15956	$skip 인수는 음수이며 음수가 아니어야 합니다.
15957	제한은 숫자로 지정해야 합니다.
15958	$limit 값은 양수여야 합니다.
15959	일치 필터는 객체의 표현식이어야 합니다.
15969	구문 오류: $project 연산자의 값은 문서여야 합니다.
15972	$skip에 대한 인수는 숫자여야 합니다.
15973	$sort 키 사양은 객체여야 합니다.

오류 코드	설명
15974	숫자를 사용하여 $sort 키 순서를 지정해야 합니다.
15975	$sort 키 순서는 1(오름차순) 또는 -1(내림차순)이어야 합니다.
15976	$sort 스테이지에는 정렬 키가 하나 이상 있어야 합니다.
15981	쿼리 필드가 문자열이 아님
15983	표현식을 나타내는 객체에는 정확히 하나의 필드가 있어야 합니다.
15992	객체 표현식에서 허용되지 않는 필드 유형 NULL
15998	필드 경로 이름은 빈 문자열일 수 없습니다.
16006	유효한 날짜 유형이 아니므로 값을 날짜로 변환할 수 없습니다.
16007	문자열이 예상되었기 때문에 값을 문자열로 변환할 수 없습니다.
16020	잘못된 수의 인수로 표현식 연산자가 호출되었습니다.
16034	$substrBytes 시작 인덱스는 숫자 형식이어야 합니다.
16035	$substrBytes 길이 인수는 숫자 형식이어야 합니다.
16395	$match 집계 표현식 내에서 $where가 허용되지 않음
16400	해당 필드 또는 해당 하위 필드 중 하나에 대한 표현식이 이미 있으므로는 필드에 대한 표현식을 추가할 수 없습니다.
16401	전체 필드에 적용되는 표현식이 이미 있으므로는 하위 필드에 표현식을 추가할 수 없습니다.
16404	이 컨텍스트에서는 $ 표현식이 허용되지 않습니다.
16406	_id 이외의 필드는 제외할 수 없습니다.
16410	$project 필드 이름 또는 경로는 로 불법적으로 시작하거나 '$'를 포함합니다.
16411	필드 이름에는 null 문자(들)를 포함할 수 없습니다.

오류 코드	설명
16419	필드 경로에는 포함된 null 문자가 포함되어서는 안 됩니다.
16424	$match 집계 표현식 내에서 $near가 허용되지 않음
16426	$nearSphere/$near는 $match 집계 표현식 내에서 허용되지 않습니다.
16435	파이프라인 구성 요소에는 하나의 요소가 포함되어야 합니다.
16493	16MB보다 긴 문자열을 생성하려고 했습니다.
16554	$add는 숫자 또는 날짜 유형만 지원합니다.
16605	$geoNear에는 유효한 'near' 필드와 올바르게 입력된 geoNear 옵션이 필요합니다.
16606	$geoNear는 문자열로 'distanceField' 옵션이 필요합니다.
16607	$geoNear를 사용하려면 주어진 옵션을 문자열로 지정해야 합니다.
16608	0으로 나눕니다.
16610	$mod에 0의 제곱이 부여되었습니다.
16611	$mod는 두 피연산자에 대한 숫자 유형만 지원합니다.
16612	$add 표현식에 허용되는 날짜는 하나뿐입니다.
16613	작업이 날짜에서 숫자가 아닌 날짜 이외의 값을 빼려고 시도했습니다.
16702	$concat은 문자열 인수만 지원합니다.
16730	textIndexVersion이 범위를 벗어남
16755	엄격한 권선 순서로 지오메트리를 인덱싱할 수 없음
16800	에는 2개의 지역 필드가 있을 수 없습니다.
16801	2d는 인덱스에서 첫 번째여야 합니다.
16804	위치 객체 예상, 위치 배열이 올바른 형식이 아님

오류 코드	설명
16810	$mod에는 파라미터로 정확히 두 값의 배열이 필요합니다.
16837	업데이트 경로는 동일한 필드를 두 번 이상 수정할 수 없습니다.
16868	잘못된 변수 이름
16869	잘못된 변수 이름
16870	잘못된 변수 이름
16872	필드 경로가 비어 있거나 '$'만 $project의 필드 경로로 사용됩니다.
16873	Fieldpath는 $로 시작하지 않습니다.
16874	$let은 객체만 인수로 지원합니다.
16875	$let에 인식할 수 없는 파라미터가 지정되었습니다.
16876	$let에 'vars' 파라미터 누락
16877	$let에 'in' 파라미터 누락
16878	$map은 객체만 인수로 지원합니다.
16879	$map에는 인식할 수 없는 파라미터가 지정됩니다.
16880	$map에 필수 '입력' 파라미터가 없습니다.
16881	필수 필드 누락
16882	$map에 'in' 파라미터 누락
16883	$map에 대한 입력은 배열이어야 합니다.
16898	$minDistance는 음수가 아니어야 합니다.
16899	$maxDistance는 숫자여야 합니다.
16900	$maxDistance는 음수가 아니어야 합니다.

오류 코드	설명
16945	$group 또는 $lookup의 메모리 제한을 초과했습니다. allowDiskUse:true를 전달하여 옵트인합니다.
16949	집계 옵션이 잘못되었습니다. allowDiskUse는 부울 값이어야 합니다.
16990	$out에는 출력 대상의 이름을 지정하는 문자열 또는 객체 인수가 필요합니다.
16996	$out에 대한 삽입 실패
16997	처리 중에 대상 컬렉션의 인덱스가 변경되어 $out에 실패했습니다.
17040	$allElementsTrue의 인수는 배열이어야 합니다.
17041	$anyElementTrue의 인수는 배열이어야 합니다.
17042	$setIsSubset의 두 피연산자는 모두 배열이어야 합니다.
17043	$setUnion의 모든 피연산자는 배열이어야 합니다.
17044	$setEquals의 피연산자는 배열이 아닙니다.
17045	$setEquals에는 인수가 두 개 이상 필요합니다.
17046	$setIsSubset의 두 피연산자는 모두 배열이어야 합니다.
17047	$setIntersection의 모든 피연산자는 배열이어야 합니다.
17048	$setDifference의 두 피연산자는 모두 배열이어야 합니다.
17049	$setDifference의 두 피연산자는 모두 배열이어야 합니다.
17053	$redact의 표현식은 $$KEEP, $$DESCEND 및 $$PRUNE 변수를 제외한 어떤 것도 반환해서는 안 됩니다.
17080	cond에 'if' 파라미터 누락
17081	cond에 'then' 파라미터 누락
17082	cond에 'else' 파라미터 누락

오류 코드	설명
17083	$cond에 대한 인식할 수 없는 파라미터
17124	$size에 대한 인수는 배열이어야 합니다.
17275	ROOT을 재정의할 수 없음
17276	변수를 찾을 수 없음
17280	인덱스 키 값이 너무 커서 인덱싱할 수 없습니다.
17287	$merge는 동일한 대상 행을 두 번 이상 수정할 수 없습니다.
17293	텍스트 인덱스 옵션 'textIndexVersion'은 숫자여야 합니다.
17302	이 인수 값은 0보다 크거나 같아야 합니다.
17304	$geoNear에는 유효한 '근접' 포인트 및 지리 쿼리 인수가 필요합니다.
17308	$meta에는 지원되지 않는 인수가 부여됩니다.
17310	$setIsSubset의 두 피연산자는 모두 배열이어야 합니다.
17313	$text가 있는 $match는 첫 번째 파이프라인 단계로만 허용됩니다.
17320	createCollection은 예약된 이름이 포함된 네임스페이스에서 실행할 수 없습니다.
17385	특수 컬렉션으로 $out할 수 없음
17390	$group은 포함 스타일 표현식을 지원하지 않습니다.
17419	업데이트 후 결과 문서가 허용되는 최대 크기를 초과합니다.
17420	업서트할 문서가 16777216보다 큼
18533	형식 필드는 문자열이어야 합니다.
18534	$dateToString에 대한 인식할 수 없는 필드
18535	후행 백분율 기호로 인해 날짜 형식 문자열이 잘못되었습니다.

오류 코드	설명
18536	잘못된 $dateToString 시간 형식
18537	$dateToString은 0~9999 범위를 벗어나는 날짜 구성 요소의 형식을 지정할 수 없습니다.
18627	$dateToString에 'format' 파라미터 누락
18628	$dateToString에 'date' 파라미터 누락
18629	잘못된 인수 유형
25278	$geoNear에는 'distanceField' 인수가 필요합니다.
26823	사용 가능한 비트 내에서는 작업 ID를 표시할 수 없습니다.
28539	고유 명령의 첫 번째 인수는 문자열이어야 합니다.
28646	$Filter는 객체를 파라미터로 예상합니다.
28647	예상치 못한 필드를 $filter로
28648	$Filter에는 '입력' 필드가 필요합니다.
28649	$Filter에는 'as' 필드가 필요합니다.
28650	$Filter에는 'cond' 필드가 필요합니다.
28651	배열 입력 필요
28656	$substrBytes: 잘못된 범위, 시작 인덱스는 UTF-8 연속 바이트입니다.
28657	$substrBytes: 잘못된 범위, 끝 인덱스가 UTF-8 문자의 중간에 있습니다.
28664	$concatArrays는 배열 입력만 지원합니다.
28667	$slice, $range 또는 $indexOfArray와 같은 표현식에 잘못된 인수 수가 발생했습니다.
28680	$abs는 부정할 수 없는 최소 길이와 같은 값에 적용됩니다.

오류 코드	설명
28689	연산자 인수는 배열이어야 하지만 다른 유형이 지정되었습니다.
28690	arrayElemAt에 대한 두 번째 인수는 숫자여야 합니다.
28691	arrayElemAt에 대한 두 번째 인수는 32비트 정수로 표시되어야 합니다.
28714	$sqrt의 인수는 0보다 크거나 같아야 합니다.
28724	$slice에 대한 첫 번째 인수는 배열이어야 합니다.
28725	$slice에 대한 두 번째 인수는 숫자 값이어야 합니다.
28726	$slice에 대한 두 번째 인수는 32비트 정수여야 합니다.
28727	$slice에 대한 세 번째 인수는 숫자여야 합니다.
28728	$slice에 대한 세 번째 인수는 32비트 정수여야 합니다.
28729	$slice의 세 번째 인수는 양수 32비트 정수여야 합니다.
28745	$sample 단계 사양은 객체여야 합니다.
28746	$sample에 대한 크기 인수는 숫자여야 합니다.
28747	$sample에 대한 크기 인수는 음수가 아니어야 합니다.
28748	$sample이 인식할 수 없는 옵션을 수신했습니다.
28749	$sample 스테이지는 크기를 지정해야 합니다.
28756	$log의 인수는 숫자여야 합니다.
28757	$log를 사용하려면 기본 인수가 숫자 값이어야 합니다.
28758	$log에 대한 인수는 양수여야 합니다.
28759	$log의 기본값은 1이 아닌 양수여야 합니다.
28761	$log10의 인수는 양수여야 합니다.

오류 코드	설명
28762	$pow의 기본값은 숫자 값이어야 합니다.
28763	$pow의 지수는 숫자여야 합니다.
28764	$pow는 0의 기본값과 음의 지수를 사용할 수 없습니다.
28765	$ceil, $exp, $floor, $ln 또는 $log10은 숫자가 아닌 유형을 수신합니다.
28766	$ln에는 양수 인수가 필요합니다.
28812	필드 경로가 비어 있음
28818	Fieldpath는 '$'로 시작하지 않습니다.
31022	이 연산자를 지정하려면 '입력' 파라미터를 지정해야 합니다.
31023	이 연산자에는 'regex' 파라미터가 필요합니다.
31024	연산자에게 알 수 없는 인수가 부여됩니다.
31032	필드 이름에는 포함된 null 바이트가 포함될 수 없습니다.
31034	값이 [-32768, 32767]의 허용 범위를 벗어납니다.
31095	'isoWeekYear'는 1에서 9999 사이의 정수로 평가되어야 합니다.
31138	정렬에 사용되는 $meta 값은 유효한 메타데이터 키워드가 아닙니다.
31170	빈 객체가 예상되었지만 비어 있지 않은 값이 제공되었습니다.
31249	프로젝션 경로가 프로젝션의 다른 필드와 충돌합니다.
31254	제외 프로젝션 내에서는 포함을 지정할 수 없습니다.
31393	$bsonSize에는 문서가 입력으로 필요합니다.
31394	위치 프로젝션 연산자는 필드 경로 중간에 나타날 수 없습니다.
34413	지리 근거리 쿼리에 잘못된 연산자가 사용되었습니다.

오류 코드	설명
34435	$reverseArray에 대한 인수는 배열이어야 합니다.
34443	$range에는 숫자 시작 값이 필요합니다.
34444	$range에는 32비트 정수로 표현할 수 있는 시작 값이 필요합니다.
34445	$range를 사용하려면 종료 값이 숫자여야 합니다.
34446	$range 종료 값은 32비트 정수여야 합니다.
34447	$range에는 숫자 단계 값이 필요합니다.
34448	$range에는 32비트 정수로 표현할 수 있는 단계 값이 필요합니다.
34449	$range에는 0이 아닌 단계 값이 필요합니다.
34450	$substrCP의 시작 인덱스는 숫자 유형이어야 합니다.
34451	$substrCP 시작 인덱스는 32비트 정수로 표시할 수 없습니다.
34452	$substrCP에 대한 길이 인수는 숫자 유형이 아닙니다.
34453	$substrCP 길이는 32비트 정수로 표시할 수 없습니다.
34454	$substrCP: 길이는 음수가 아닌 정수여야 합니다.
34455	$substrCP: 시작 인덱스는 음수가 아닌 정수여야 합니다.
34460	$zip은 객체만 인수로 지원합니다.
34461	$zip 입력은 표현식 배열이어야 합니다.
34462	'defaults' 옵션은 표현식 배열이어야 합니다.
34463	$zip의 useLongestLength 옵션은 부울이어야 합니다.
34464	$zip에 알 수 없는 인수가 지정되었습니다.
34465	$zip에는 하나 이상의 입력 배열이 필요합니다.

오류 코드	설명
34466	useLongestLength가 true가 아닌 한는 기본값을 지정할 수 없습니다.
34467	기본값과 입력의 길이는 같아야 합니다.
34468	$zip은 입력에서 배열이 아닌 표현식을 수신합니다.
34471	$strLenCP에는 문자열 인수가 필요합니다.
34473	$strLenBytes에는 문자열 인수가 필요합니다.
40060	$스위치에는 객체가 인수로 필요합니다.
40061	$스위치를 사용하려면 '브랜치'가 배열이어야 합니다.
40062	$스위치를 사용하려면 각 브랜치가 객체여야 합니다.
40063	브랜치 내에 알 수 없는 인수가 $스위치에 할당됩니다.
40064	$switch를 사용하려면 각 브랜치에 'case' 표현식이 있어야 합니다.
40065	$스위치를 사용하려면 각 브랜치에 'then' 표현식이 있어야 합니다.
40066	$switch가 입력에 대해 일치하는 브랜치를 찾을 수 없으며 기본값이 지정되지 않았습니다.
40067	$switch 표현식에 알 수 없는 인수가 포함되어 있습니다.
40068	$switch에는 하나 이상의 브랜치가 필요합니다.
40075	$reduce에 객체 필요
40076	$reduce에 대한 알 수 없는 인수
40077	$reduce에는 'input'이 필요합니다.
40078	$reduce에는 'initialValue'가 필요합니다.
40079	$reduce에는 'in'이 필요합니다.
40080	$reduce에 대한 입력은 배열이어야 합니다.

오류 코드	설명
40081	연산자는 두 번째 인수로 배열이 필요합니다.
40085	$split을 문자열로 평가하려면 첫 번째 인수가 필요합니다.
40086	$split의 두 번째 인수는 문자열로 평가되어야 합니다.
40087	$split에는 비어 있지 않은 구분자가 필요합니다.
40090	$indexOfArray는 첫 번째 인수로 배열이 필요합니다.
40091	$indexOfBytes에 대한 첫 번째 인수는 문자열이 아닙니다.
40092	$indexOfBytes에는 문자열이 두 번째 인수로 필요합니다.
40093	$indexOfCP에는 첫 번째 인수로 문자열이 필요합니다.
40094	$indexOfCP에는 문자열이 두 번째 인수로 필요합니다.
40096	$indexOfArray에는 필수 시작 및 종료 인덱스 값이 필요합니다.
40097	$indexOfArray에는 음수가 아닌 숫자 시작 및 종료 인덱스가 필요합니다.
40148	sortByCount 필드는 $ 접두사 경로 또는 객체 내의 표현식으로 정의되어야 합니다.
40149	sortByCount 필드는 문자열 또는 객체로 지정해야 합니다.
40156	개수 필드는 비어 있지 않은 문자열이어야 합니다.
40157	개수 필드는 비어 있지 않은 문자열이어야 합니다.
40158	개수 필드는 $ 접두사 경로일 수 없습니다.
40159	개수 필드에는 null 바이트가 포함될 수 없습니다.
40160	개수 필드에는 '.'를 포함할 수 없습니다.
40169	$facet 사양은 비어 있지 않은 객체여야 합니다.
40170	$facet에 대한 인수는 배열이어야 합니다.

오류 코드	설명
40171	$facet 사양의 배열 요소는 비어 있지 않은 객체여야 합니다.
40172	$facet 단계의 하위 파이프라인은 비워둘 수 없습니다.
40176	충돌하는 필드 이름이 지정됨
40177	$project 및 $addFields에는 하나 이상의 출력 필드가 필요합니다.
40178	포함 프로젝션에서 '_id'를 제외한 필드는 제외할 수 없습니다.
40179	제외 프로젝션에 필드를 포함하거나 필드를 재설정하거나 계산된 필드를 추가할 수 없습니다.
40180	$project는 빈 객체를 필드 값으로 허용하지 않습니다.
40181	잘못된 $project :: ::로 인해 표현식 사양에 정확히 하나의 필드가 포함되어야 합니다.
40183	점선 필드 이름은 하위 객체 내에서 사용할 수 없습니다.
40192	$bucket 'boundary' 필드에는 2개 이상의 값이 있어야 합니다.
40193	$bucket에 대한 'boundary' 옵션의 모든 값은 동일한 유형이어야 합니다.
40194	$bucket에 대한 '경계' 옵션은 정렬해야 하지만 요소는 오름차순으로 정렬되지 않습니다.
40195	$bucket 'default' 필드는 상수 표현식이어야 합니다.
40196	$bucket 'output' 필드는 객체여야 합니다.
40197	$bucket에 대한 인식할 수 없는 옵션
40198	$bucket을 사용하려면 'groupBy' 및 'boundaries'를 지정해야 합니다.
40199	$bucket 'default' 필드는 가장 낮은 경계보다 작거나 가장 높은 경계보다 크거나 같아야 합니다.
40200	$bucket 'boundaries' 필드는 배열이어야 합니다.

오류 코드	설명
40201	$bucket 스테이지에 대한 인수는 객체여야 합니다.
40202	$bucket 'groupBy' 필드는 $ 접두사 경로 또는 표현식으로 정의되어야 합니다.
40203	필드는 누적기 객체여야 합니다.
40204	필드 이름은 ''를 포함할 수 없습니다.
40205	필드 이름은 연산자 이름이 될 수 없습니다.
40206	누적기는 단항 연산자입니다.
40218	쿼리에는 텍스트 점수 메타데이터가 필요하지만 사용할 수 없음
40228	$replaceRoot 'newRoot' 표현식은 객체로 평가되어야 합니다.
40229	$replaceRoot에는 객체가 사양으로 필요합니다.
40230	$replaceRoot는 'newRoot' 옵션만 허용합니다.
40234	$group 필드는 누적기 객체여야 합니다.
40235	필드 이름에는 '.' 문자를 사용할 수 없습니다.
40236	필드 이름은 '$'로 시작할 수 없습니다.
40237	이 누적기는 단항 연산자이며 단일 인수를 사용합니다.
40239	$bucketAuto 'groupBy' 필드는 $ 접두사 경로 또는 표현식 객체로 정의되어야 합니다.
40240	$bucketAuto에 대한 인수는 객체여야 합니다.
40241	$bucketAuto 'buckets' 필드는 숫자 값이어야 합니다.
40242	$bucketAuto 'buckets' 필드는 32비트 정수로 표현할 수 있어야 합니다.
40243	$bucketAuto 'buckets' 필드는 0보다 커야 합니다.
40244	$bucketAuto 'output' 필드는 객체여야 합니다.

오류 코드	설명
40245	$bucketAuto에 대한 인식할 수 없는 옵션
40246	$bucketAuto를 사용하려면 'groupBy' 및 'buckets'를 지정해야 합니다.
40257	알 수 없는 반올림 세부 수준
40258	$bucketAuto는 숫자 경계만 있는 '세분성'을 지정할 수 있습니다.
40260	$bucketAuto는 음수가 아닌 숫자로만 '세분성'을 지정할 수 있지만 음수가 발견되었습니다.
40261	$bucketAuto '세분성' 필드는 문자열이어야 합니다.
40272	$addFields 스테이지 사양은 객체여야 합니다.
40319	$lookup 단계 사양은 객체여야 합니다.
40320	$lookup 단계 사양에 'from' 옵션 누락
40321	$lookup에 대한 'from' 옵션은 문자열이어야 합니다.
40322	잘못된 $lookup 네임스페이스
40323	파이프라인 단계 사양 객체에는 정확히 하나의 필드가 포함되어야 합니다.
40324	파이프라인에는 인식할 수 없는 단계 이름이 포함되어 있습니다.
40331	이 집계 단계는 $facet 파이프라인 내에서 허용되지 않습니다.
40352	$project 필드 경로는 빈 문자열일 수 없습니다.
40353	필드 경로는 '.'로 끝나지 않아야 합니다.
40386	$arrayToObject에는 배열 입력이 필요합니다.
40390	$objectToArray에는 문서가 입력으로 필요합니다.
40391	$arrayToObject에는 일관된 입력 형식이 필요합니다. 요소는 모두 배열이거나 모두 객체여야 합니다. 객체가 감지되었습니다. 이제 찾았습니다. 배열

오류 코드	설명
40392	$arrayToObject에는 정확히 키가 'k' 및 'v'인 객체가 필요합니다.
40393	$arrayToObject에는 키가 'k' 및 'v'인 객체가 필요합니다. 두 키 중 하나 또는 둘 다 누락
40394	$arrayToObject에는 문자열 'k' 키와 'v' 값이 있는 객체가 필요합니다.
40395	$arrayToObject에는 키가 문자열인 키-값 페어가 필요합니다.
40396	$arrayToObject에는 일관된 입력 형식이 필요합니다. 요소는 모두 배열이거나 모두 객체여야 합니다. 배열이 감지되었습니다. 이제 객체가 발견되었습니다.
40397	$arrayToObject에는 두 요소 배열의 배열이 필요합니다.
40398	$arrayToObject에는 인식할 수 없는 유형 또는 형식의 입력이 제공됩니다.
40400	$mergeObjects를 사용하려면 모든 입력이 객체여야 합니다.
40414	필수 명령 필드가 누락되었거나 검증에 실패했습니다.
40415	명령에 인식할 수 없는 필드가 포함되어 있습니다.
40422	BSON 배열 필드 이름에 잘못된 값이 있습니다.
40485	인식할 수 없는 시간대 식별자가 제공되었습니다.
40489	$dateFromParts는 정규 일정 필드를 ISO 주 날짜 필드와 혼합하는 것을 허용하지 않습니다.
40515	$dateFromParts 인수는 정수로 평가되지 않습니다.
40516	$dateFromParts를 사용하려면 'year' 또는 'isoWeekYear'가 있어야 합니다.
40517	'시간대' 옵션은 날짜 연산자의 문자열로 평가되어야 합니다.
40518	$dateFromParts 또는 $dateToParts에 인식되지 않거나 잘못 입력된 파라미터가 있습니다.
40519	$dateFromParts는 객체만 인수로 지원합니다.

오류 코드	설명
40520	$dateToParts에 인식할 수 없는 파라미터가 지정되었습니다.
40522	$dateToParts에는 'date' 파라미터가 있는 객체 인수가 필요합니다.
40523	$dateFromParts 'year'는 1~9999 범위의 정수여야 합니다.
40525	$dateFromParts ISO 주/일 값이 범위를 벗어났거나 필수 필드가 누락되었습니다.
40533	시간대 인수는 문자열이어야 합니다.
40535	단계 또는 표현식에 인식할 수 없는 옵션이 제공되었습니다.
40536	표현식은 배열을 지정할 때 정확히 하나의 인수를 허용합니다.
40539	날짜 연산자에 대한 필수 '날짜' 인수가 누락되었습니다.
40540	$dateFromString은 객체만 인수로 지원합니다.
40541	$dateFromString이 인식할 수 없는 인수를 수신했습니다.
40542	$dateStringdateFromString' 파라미터 누락
40543	$dateFromString을 사용하려면 'dateString'이 문자열이어야 합니다.
40545	날짜/시간 문자열이 불완전하고 필수 요소가 누락되었습니다.
40551	시간대 인수와 함께 시간대 정보가 포함된 날짜/시간 문자열을 전달할 수 없습니다.
40553	$dateFromString은 지정된 날짜 문자열을 구문 분석할 수 없습니다.
40554	시간대 인수와 함께 GMT 오프셋이 있는 날짜/시간 문자열을 전달할 수 없습니다.
40575	$changeStream 'fullDocument'는 'default' 또는 'updateLookup'이어야 합니다.
40600	$out은 $facet 단계 내에서 사용할 수 없습니다.
40601	$out 및 $merge는 파이프라인의 마지막 단계일 수 있습니다.

오류 코드	설명
40602	$changeStream 또는 $geoNear와 같은 단계는 파이프라인의 첫 번째 단계가 아닙니다.
40603	$geoNear는 최적화 후 파이프라인의 첫 번째 단계가 아니었습니다. 최적화가 비활성화되거나 금지되나요?
40647	변경 스트림 재개 토큰에 누락되었거나 잘못된 유형의 _data 필드가 있습니다.
40674	한 가지 유형의 재개 옵션만 허용되지만 여러 개가 발견되었습니다.
50001	autoIndexId:false는 지정된 컬렉션에 허용되지 않습니다.
50694	이 연산자에게 알 수 없는 인수가 부여되었습니다.
50695	이 연산자에는 '입력' 필드가 필요합니다.
50696	연산자는 객체만 인수로 지원합니다.
50699	이 표현식 연산자는 입력이 문자열이어야 합니다.
50700	'chars' 인수는 문자열이어야 합니다.
50736	getMore는 세션에서 생성되지 않은 커서의 세션에서 실행할 수 없습니다.
50737	세션 생성 커서에서 getMore를 실행하려면 lsid를 제공해야 합니다.
50752	$substrBytes 시작 인덱스가 음수입니다.
50808	$changeStream 단계에서는 문서를 인수로 예상합니다.
51024	데이터 정렬 '강도' 값은 5 이하여야 합니다.
51047	이 연산자 또는 단계는 $lookup 하위 파이프라인 내에서 허용되지 않습니다.
51075	$regex 및 $options 모두에서 옵션을 설정할 수 없음
51076	지정된 목록 수에 대한 벡터 인덱스를 빌드할 메모리가 부족합니다.
51078	인덱스 이름을 찾을 수 없습니다.

오류 코드	설명
51079	지연 시간 통계는 지원되지 않습니다.
51091	정규식이 잘못되었거나 잘못 구성되었습니다.
51103	연산자는 명명된 인수의 객체를 예상하지만 다른 유형을 수신했습니다.
51104	연산자를 사용하려면 'input' 인수가 문자열이어야 합니다.
51105	연산자의 'regex' 인수는 문자열 또는 정규식 유형이 아닙니다.
51106	연산자를 사용하려면 'options' 파라미터가 문자열이어야 합니다.
51107	$regex 옵션은 'regex' 및 'options' 필드 모두에서 지정할 수 없습니다.
51108	정규 표현식 옵션에는 잘못된 플래그 또는 포함된 null 바이트가 포함되어 있습니다.
51109	정규식에는 포함된 null 바이트가 포함될 수 없습니다.
51110	정규 표현식 옵션에는 포함된 null 바이트가 포함될 수 없습니다.
51111	이 연산자에 제공된 정규식이 잘못되었습니다.
51151	출력을 저장할 버퍼 크기가 데이터 형식 제한을 초과했습니다.
51156	평가 중에 $regex에서 정규식을 실행하지 못했습니다.
51270	빈 하위 프로젝션 객체는 유효한 프로젝션 값이 아닙니다.
51272	구문 오류: $projection에는 하나 이상의 출력 필드가 필요합니다.
51744	연산자를 문자열로 사용하려면 'replacement'가 필요합니다.
51745	연산자를 문자열로 사용하려면 'find' 인수가 필요합니다.
51746	연산자의 'input' 인수는 문자열이 아닙니다.
51747	연산자에는 'replacement' 파라미터가 필요합니다.
51748	이 연산자에는 '찾기' 파라미터가 필요합니다.

오류 코드	설명
51749	이 연산자에는 '입력' 파라미터가 필요합니다.
51750	집계 연산자가 알 수 없는 인수를 수신했습니다.
100003	최적화된 쿼리 경로를 처리하는 동안 내부 시스템 오류 발생
100005	컬렉션, 사용자 지정 역할 또는 권한과 같이 구성된 제한을 초과했습니다.
100006	허용된 최대 사용자 수를 초과했습니다.
3040501	$rand는 현재 인수를 허용하지 않습니다.
3041701	$getField가 $project에서 알 수 없는 인수를 수신했습니다.
3041702	잘못된 $project :: :: $getField에서 'field'를 지정해야 함
3041703	잘못된 $project :: :: $getField에 의해 발생하려면 '입력'을 지정해야 합니다.
3041704	$getField를 문자열로 평가하려면 '필드' 인수가 필요합니다.
4161100	$setField 및 $unsetField는 객체를 인수로만 지원합니다.
4161101	$setField 또는 $unsetField에 알 수 없는 인수가 부여되었습니다.
4161102	$setField 및 $unsetField를 지정하려면 '필드'를 지정해야 합니다.
4161103	잘못된 $project :: :: $setField에서 'value'를 지정해야 함
4161105	$setField 또는 $unsetField를 객체로 평가하려면 '입력'이 필요합니다.
4161106	$setField 및 $unsetField를 상수로 평가하려면 '필드'가 필요합니다.
4161107	$setField 또는 $unsetField를 문자열로 평가하려면 '필드'가 필요합니다.
4161108	$project는 여기서 필드 경로 참조를 허용하지 않습니다. 대신 {$literal: ...}를 사용합니다.
4161109	$setField 및 $unsetField를 사용하려면 '입력' 필드를 지정해야 합니다.
4662500	명령에 인식할 수 없는 필드가 포함되어 있습니다.

오류 코드	설명
4862100	정규화된 네임스페이스가 최대 허용 길이를 초과했습니다.
5166301	$dateDiff는 객체만 인수로 지원합니다.
5166302	$dateDiff에 인식할 수 없는 인수가 부여되었습니다.
5166303	$dateDiff에 'startDate' 파라미터 누락
5166304	$dateDiff에 'endDate' 파라미터 누락
5166305	$dateDiff에 '단위' 파라미터 누락
5166307	$dateDiff 인수는 날짜가 아닙니다.
5166308	날짜 차이를 계산하는 동안 $dateDiff가 오버플로되었습니다.
5166400	$dateAdd 및 $dateSubtract를 사용하려면 인수가 객체여야 합니다.
5166401	$dateAdd 또는 $dateSubtract에 인식할 수 없는 인수가 발생했습니다.
5166402	$dateAdd 또는 $dateSubtract를 사용하려면 startDate, 단위 및 금액이 있어야 합니다.
5166403	$dateAdd를 사용하려면 startDate를 날짜로 변환할 수 있어야 합니다.
5166404	$dateAdd는 시간 단위를 정의하는 문자열을 예상합니다.
5166405	$dateAdd는 시간 단위의 정수 양을 예상합니다.
5166406	dateAdd 오버플로
5166503	알 수 없는 시간 단위 값이 제공되었습니다.
5439002	잘린 날짜를 계산하는 동안 $dateTrunc가 오버플로되었습니다.
5439006	dateTrunc 지원되지 않는 binSize 값
5439007	$dateTrunc는 객체만 인수로 지원합니다.
5439008	인식할 수 없는 $dateTrunc 인수, 예상 날짜, 단위, binSize, 시간대, startOfWeek.

오류 코드	설명
5439009	$dateTrunc에 'date' 파라미터 누락
5439010	$dateTrunc에 'unit' 파라미터 누락
5439012	$dateTrunc를 사용하려면 'date'가 날짜여야 합니다.
5439013	$dateDiff 및 $dateTrunc는 '단위'가 문자열이어야 합니다.
5439014	$dateDiff 및 $dateTrunc 'unit' 값은 인식된 시간 단위가 아닙니다.
5439015	$dateDiff 또는 $dateTrunc를 사용하려면 'startOfWeek'가 문자열이어야 합니다.
5439016	$dateDiff 또는 $dateTrunc 'startOfWeek' 값은 유효한 요일이 아닙니다.
5439017	$dateTrunc를 사용하려면 'binSize'가 64비트 정수여야 합니다.
5439018	$dateTrunc를 사용하려면 'binSize'가 0보다 커야 합니다.
5447000	$collStats 단계는 중첩 객체여야 합니다.
5626500	이 컨텍스트에서는 $geoNear, $near 및 $nearSphere가 허용되지 않습니다. 이러한 연산자는 지리 공간 데이터를 정렬해야 하기 때문입니다. 정렬이 필요하지 않은 경우 대신 $geoWithin을 사용하는 것이 좋습니다.
5654600	$project는이 컨텍스트에서 필드 경로 참조를 허용하지 않습니다.
5654601	잘못된 $project :: :: $getField에서 상수로 평가하려면 '필드'가 필요하지만 상수가 아닌 인수를 가져옴
5654602	$getField는 문자열로 평가하려면 '필드'가 필요합니다.
5733415	$mod를 0으로 설정할 수 없음
5860400	$geoNear에는 'near' 인수가 필요합니다.
5887502	$setEquals의 모든 피연산자는 배열이어야 합니다.
5911200	집계 중 PlanExecutor 오류:: 로 인해 발생:: $mergeObjects는 객체만 지원합니다.
5976500	$dateAdd에 잘못된 'amount' 파라미터 값이 부여되었습니다.

오류 코드	설명
6045000	$dateSubtract에 잘못된 'amount' 파라미터 값이 부여되었습니다.
7158303	$switch가 입력과 일치하는 브랜치를 찾지 못했으며 기본값이 지정되지 않았습니다.

릴리스 노트

이 릴리스 정보에서는 메이저 엔진 버전별로 구성된 Amazon DocumentDB 기능, 개선 사항 및 버그 수정에 대해 설명합니다. 각 버전 섹션 내에서 항목은 날짜별로 테이블 형식으로 나열됩니다.

엔진 버전 및 패치 버전의 번호 지정 방법에 대한 자세한 내용은(엔진 패치 버전 접두사와 Amazon DocumentDB 엔진 버전 간의 매핑 포함) 섹션을 참조하세요엔진 버전 번호 지정.

현재 Amazon DocumentDB 엔진 패치 버전을 확인하려면 다음 명령을 실행합니다.

```
db.runCommand({getEngineVersion: 1})
```

> **Note**
>
> Amazon DocumentDB 5.0.2 및 8.0.2부터이 명령은 엔진 버전과 엔진 패치 버전을 형식의 단일 문자열로 반환합니다*engine-version*+*engine-patch-version*. 예를 들어 Amazon DocumentDB 5.0.2의 클러스터는를 반환하고 `5.0.2+3.0.123458.0.2`의 클러스터는를 반환합니다`8.0.2+4.0.12345`. 이전 버전의 클러스터는와 같이 엔진 패치 버전을 자체적으로 반환합니다`3.0.12345`. 자세한 내용은 엔진 버전 번호 지정 섹션을 참조하세요.

클러스터가 최신 버전의 엔진을 사용하지 않는 경우, 엔진을 업그레이드하기 위해 보류 중인 유지 관리가 있을 수 있습니다. 자세한 내용은 개발자 가이드의 Amazon DocumentDB 유지 관리 섹션을 참조하세요.

주제

- Amazon DocumentDB 엔진 버전 릴리스 정보
- 일반 Amazon DocumentDB 릴리스 정보
- Amazon DocumentDB 탄력적 클러스터 릴리스 정보

Amazon DocumentDB 엔진 버전 릴리스 정보

메이저 버전별로 구성된 Amazon DocumentDB 엔진 버전의 릴리스 정보입니다.

Amazon DocumentDB 8.0

Amazon DocumentDB 엔진 버전 8.0(MongoDB 8.0 호환성)에 대한 릴리스 정보입니다.

Amazon DocumentDB 8.0.2

Date	릴리스 노트
2026년 9월 28일	Amazon DocumentDB 8.0.2(엔진 버전: 4.0.12042) 새로운 기능 및 개선 사항 1. 플래너 버전 3에서 `$setWindowFields` 집계 단계 및 창 연산자에 대한 지원이 추가되었습니다. 2. 플래너 버전 3의 `$bucketAuto` 집계 단계에 대한 지원이 추가되었습니다. 3. 플래너 버전 3의 `$facet` 집계 단계에 대한 지원이 추가되었습니다. 4. 플래너 버전 3의 `$graphLookup` 집계 단계에 대한 지원이 추가되었습니다. 5. 플래너 버전 3의 파이프라인 단계 하위 집합과 함께 간결한 상관 하위 쿼리 구문을 `$lookup` 포함하여 상관 하위 쿼리에 대한 지원이 추가되었습니다. 6. `$changeStreamSplitLargeEvent` 집계 단계에 대한 지원이 추가되었습니다. 7. 이제 변경 스트림이 `createCollection` 작업에 대한 이벤트를 내보냅니다. 8. 이제 변경 스트림이 `createIndex` 작업에 대한 이벤트를 내보냅니다. 9. 변경 스트림 쓰기 성능이 개선되었습니다. 10.재시도 가능한 쓰기에 대한 지원이 추가되었습니다. 11.부분 인덱스의 `reIndex` 명령에 대한 지원이 추가되었습니다. 12.플래너 버전 3은 이제 `$expr` 조건자에 대한 인덱스 스캔을 지원합니다. 13.플래너 버전 3에는 찾기 쿼리를 위한 다중 필드 인덱스 전용 스캔(IXONLYSCAN)이 도입되었습니다. 자세한 내용은 인덱스 전용 스캔(적용된 쿼리) 섹션을 참조하세요.

Date	릴리스 노트
	14.플래너 버전 3에는와 같은 집계 단계에 대한 인덱스 전용 스캔(IXONLYSCAN)이 도입되었습니다$group. 자세한 내용은 인덱스 전용 스캔(적용된 쿼리) 섹션을 참조하세요. 15.플래너 버전 3countDocuments() 의 $count 및에 대한 성능 최적화. 16.플래너 버전 3의 증분 정렬에 대한 지원이 추가되었습니다. 자세한 내용은 증분 정렬 섹션을 참조하세요. 17.텍스트 인덱스 쿼리에 대한 계획 선택을 개선했습니다. 18.이제 병합 $lookup 및 $unwind 단계에 대한 설명 계획에 includeArrayIndex 파라미터가 반영됩니다. 19모든 명령에 인덱스 필터 지원이 추가되었습니다. 20.쿼리 플래너는 이제 등식($eq) 조건자를 위한 고유 인덱스의 우선 순위를 지정합니다. 21.currentOp 출력의 집계 명령에 대한 잘림 제한을 늘렸습니다. 버그 수정 1. 버전 1(기본값) 쿼리 플래너의 버그를 수정했습니다. 2. 메이저 버전 업그레이드(MVU) 중에 부분 인덱스 표현식 텍스트 왕복 처리를 수정했습니다. 3. 빠른 수 쿼리에 대한 프로파일러 문제를 수정했습니다. 4. collStats 명령의 보안 수정입니다.

Amazon DocumentDB 8.0.1

Date	릴리스 노트
2026년 9월 15일	보안 및 안정성 개선 Amazon DocumentDB 8.0.1(엔진 버전: 4.0.11638) [진행 중]

Date	릴리스 노트
	Amazon DocumentDB는 다음 유지 관리 기간 동안이 패치를 자동으로 적용합니다.
2026년 8월 31일	새로운 기능 Amazon DocumentDB는 이제 엔진 버전 3.6 및 4.0에서 버전 8.0으로 직접 인플레이스 메이저 버전 업그레이드(MVU)를 지원합니다. 따라서 중간 버전 업그레이드가 필요하지 않으며 기존 데이터, 구성 및 클러스터 설정을 유지하면서 3.6 또는 4.0 클러스터를 업그레이드할 수 있습니다. Amazon DocumentDB 3.6 및 4.0에서 8.0으로의 메이저 버전 업그레이드는 현재 이러한 버전이 지원되는 모든 AWS 리전에서 사용할 수 있습니다. 자세한 내용은 Amazon DocumentDB 인플레이스 주요 버전 업그레이드 섹션을 참조하세요.
2026년 8월 18일	보안 및 안정성 개선 Amazon DocumentDB 8.0.1(엔진 버전: 4.0.9938) 이 엔진 패치는 다음과 같은 개선 사항을 제공합니다. • 요청 처리 개선 Amazon DocumentDB는 이제 연결 및 메시지 구문 분석 경로를 따라 수신 클라이언트 요청을 더 철저하게 검증합니다. 이렇게 하면 광범위한 입력 패턴에서 복원력과 안정성이 향상됩니다. • 인증 핸드셰이크 강화 Amazon DocumentDB는 이제 인증 교환을 더 철저하게 검증합니다. 이렇게 하면 예기치 않거나 잘못된 형식의 입력을 처리할 때 견고성이 향상됩니다. Amazon DocumentDB는 다음 유지 관리 기간 동안이 패치를 자동으로 적용합니다.

Date	릴리스 노트
2026년 8월 11일	새로운 기능 Amazon DocumentDB 8.0.1(엔진 버전: 4.0.8864) • SCRAM-SHA-256 인증에 대한 지원을 활성화했습니다. • Amazon DocumentDB 글로벌 클러스터에 대해 튜닝 가능한 RPO를 활성화했습니다. • 백그라운드 가비지 수집 중 스토리지 재생성을 개선하여 시간이 지남에 따라 미사용 스토리지 공간을 줄였습니다.
2026년 7월 22일	새로운 기능 엔진 버전 8.0 이상을 실행하는 Amazon DocumentDB 클러스터는 이제 최대 스토리지 제한인 256TiB를 지원하여 이전 제한인 128TiB의 두 배를 지원합니다. 자세한 내용은 Amazon DocumentDB 할당량 섹션을 참조하세요.

Date	릴리스 노트
2026년 7월 13일	Amazon DocumentDB 8.0.1(엔진 버전: 4.0.8864) 새로운 기능 및 개선 사항 1. 기본 플래너(v3)에서 13개의 누적기 연산자 지원: $top, $bottom, $topN, $bottomN, $count, $firstN, $lastN, $minN, $maxN, $median, $percentile , $stdDevPop , . $stdDevSamp 2. 기본 플래너(v3)에서 36개의 표현식 연산자 지원: $median, $percentile , $stdDevPop , , $stdDevSamp , $round, $sigmoid, $trunc$firstN, $lastN, $maxN, $minN $bitAnd $bitNot$bitOr$bitXor, $binarySize , $bsonSize , , $tsIncrement , , $tsSecond , $cos, $acos, $acosh, $cosh, , , sinasin$sinh$asinh$tan, $atan, $tanh, $atan2, , , $degreesToRadians , $atanh, $isNumber , $radiansToDegrees . $toUUID 3. 기본 플래너(v3)$sampleRate 의 $match에서를 지원합니다. 4. 기본 플래너(v3)$sortByCount 에서를 지원합니다. 5. 기본 플래너(v3)에 $listSearchIndexes 집계 단계에 대한 지원이 추가되었습니다. 컬렉션의 텍스트 및 벡터 검색 인덱스에 대한 검색 인덱스 메타데이터(이름, 상태, 쿼리 가능, 버전)를 반환합니다. 6. OP_COMPRESSED 메시지 유형을 사용하여 스냅 압축을 지원합니다. 7. 기본 플래너(v3)의 찾기 명령에서 min 및 max 커서 메서드를 지원합니다. 8. 플래너 버전 3에는에 대한 인덱스 전용 스캔(IXONLYSCAN)이 도입되었습니다countDocuments() . 9. 모든 인덱스 유형(플래너 버전 2 이상)에 대한 숨겨진 인덱스 지원이 추가되었습니다. 버그 수정 1. 밑줄 문자가 포함된 $text 검색 쿼리가 잔차 필터로 평가될 때 잘못된 결과를 반환할 수 있는 문제를 수정했습니다.

Amazon DocumentDB 8.0.0

Date	릴리스 노트
2026년 9월 15일	보안 및 안정성 개선 Amazon DocumentDB 8.0.0(엔진 버전: 4.0.11440) [진행 중] 버그 수정 이 릴리스에는 다음 버그 수정이 포함되어 있습니다. 1. 밑줄 문자가 포함된 `$text` 검색 쿼리가 잔차 필터로 평가될 때 잘못된 결과를 반환할 수 있는 문제를 수정했습니다. Amazon DocumentDB는 다음 유지 관리 기간 동안이 패치를 자동으로 적용합니다.
2026년 8월 18일	보안 및 안정성 개선 Amazon DocumentDB 8.0.0(엔진 버전: 4.0.9912) 이 엔진 패치는 다음과 같은 개선 사항을 제공합니다. • 요청 처리 개선 Amazon DocumentDB는 이제 연결 및 메시지 구문 분석 경로를 따라 수신 클라이언트 요청을 더 철저하게 검증합니다. 이렇게 하면 광범위한 입력 패턴에서 복원력과 안정성이 향상됩니다. • 인증 핸드셰이크 강화 Amazon DocumentDB는 이제 인증 교환을 더 철저하게 검증합니다. 이렇게 하면 예기치 않거나 잘못된 형식의 입력을 처리할 때 견고성이 향상됩니다. 이 패치를 더 빨리 적용하지 않으면 Amazon DocumentDB는 릴리스 후 30일이 지난 다음 유지 관리 기간에이 패치를 자동으로 적용합니다.
2026년 7월 22일	새로운 기능 엔진 버전 8.0 이상을 실행하는 Amazon DocumentDB 클러스터는 이제 최대 스토리지 제한인 256TiB를 지원하여 이전 제한인 128TiB의 두 배

Date	릴리스 노트
	를 지원합니다. 자세한 내용은 Amazon DocumentDB 할당량 섹션을 참조하세요.
2026년 5월 13일	새로운 기능 이제 Amazon DocumentDB 서버리스를 Amazon DocumentDB 8.0에서 사용할 수 있습니다. 자세한 내용은 Amazon DocumentDB Serverless 사용 섹션을 참조하세요.
2026년 5월 6일	버그 수정 및 기타 변경 사항 Amazon DocumentDB 8.0(엔진 버전: 4.0.6940) 이 업데이트는 다음과 같은 개선 사항이 적용된 선택적 엔진 패치를 제공합니다. 1. 엣지 케이스에서 독자가 충돌할 수 있는 가비지 컬렉션의 버그를 수정했습니다.
2026년 4월 20일	새로운 기능 이제 Amazon DocumentDB 5.0에서 Amazon DocumentDB 8.0으로의 인플레이스 메이저 버전 업그레이드(MVU)를 사용할 수 있습니다. 자세한 내용은 Amazon DocumentDB 인플레이스 주요 버전 업그레이드 섹션을 참조하세요.

Date	릴리스 노트
2026년 4월 20일	새로운 기능 및 개선 사항 Amazon DocumentDB 8.0(엔진 버전: 4.0.5995, 4.0.6473) 이는 다음과 같은 새로운 기능, 개선 사항 및 수정 사항이 포함된 필수 엔진 패치입니다. 1. `$sortArray` 의 연산자`$project`: • 플래너 버전 3에 대한 집계 파이프라인의 `$project` 단계 내 `$sortArray` 운영자 지원. `$sortArray` 연산자는 요소(스칼라 값의 경우) 또는 지정된 필드(문서 배열의 경우)를 기준으로 배열을 정렬하여 정렬된 배열을 오름차순(1) 또는 내림차순(-1)으로 반환합니다. 2. 플래너 v3를 `$geoNear` 사용하여 `$collStats` , `$out``$indexStats` ,를 지원합니다. 플래너 V3에서는 `$out` 여러 데이터베이스의 컬렉션에 쓰기도 지원합니다. 3. 집계 파이프라인은 이제 `$unwind` 스테이지에 `includeArrayIndex` 파라미터가 포함된 경우에도 `$lookup` 및 `$unwind` 스테이지를 병합합니다. 이전에는가 있을 때이 최적화`includeArrayIndex` 를 건너뛰었습니다. 4. I/O 최적화 구성에서 실행되는 db.r8g.8xlarge 이상의 인스턴스에 대한 쓰기 처리량을 개선하기 위한 성능 향상. 5. 이제 인스턴스에 대한 `Connections (active)` 지표를 모니터링할 수 있습니다. 자세한 내용은 인스턴스 할당량을 참조하세요. 6. `$dateToParts` 및 `$dateFromParts` 집계 연산자를 지원합니다. 7. 이제 CloudWatch에서 명령 수준 성능 지표를 CRUD 작업(`find`, , `insert`, `findAndModify` , `updatedelete`, `aggregate` , `count`, `distinct`, `getMore`, `abortTransaction` , `commitTransaction` ), 노출`AvgDuration` , `P100Duration` , `MaxConcurrent` 차원에 사용할 수 있어 운영 모니터링이 개선되었습니다. 8. 출력 문서에서 부울 값 `true`/`false` 또는 숫자 값 `0`/`1`가 포함된 `newRoot` 필드가 누락된 `$replaceRoot` 동작의 버그를 수정했습니다.

Date	릴리스 노트
	9. 유효한 쿼리$indexOfArray 가 유효하지 않음을 나타내는 오류를 반환하는 버그를 수정했습니다. 10.이제 설명 계획에 executionStats 출력에 docsExamined 지표가 포함됩니다. 11.SASL 인증 성능이 개선되었습니다. 12.인덱스 값이 변경되지 않고 문서가 기존 페이지에 적합할 때 스토리지 오버헤드를 줄이도록 문서 업데이트가 최적화되었습니다. 13bypassDocumentValidation 에서에 대한 지원$merge. 모든 Amazon DocumentDB 8.0 클러스터는 다음 유지 관리 기간 동안 이 패치 버전으로 자동 업그레이드됩니다.
2026년 3월 26일	새로운 기능 Amazon DocumentDB는 이제 Arm-based AWS Graviton4 프로세서로 구동되는 최신 메모리 최적화 인스턴스 db.r8g를 지원하여 R6G 인스턴스보다 최대 30% 더 나은 성능을 제공합니다. 자세한 내용은 이 가이드 인스턴스 클래스 관리의 섹션을 참조하세요.

Date	릴리스 노트
2025년 11월 11일	새로운 기능 Amazon DocumentDB는 이제 MongoDB 8.0과의 전체 유선 프로토콜 호환성을 제공합니다. Amazon DocumentDB 8.0은 쿼리 성능을 개선하여 더 저렴한 비용으로 고성능 애플리케이션을 구축할 수 있습니다. Amazon DocumentDB 8.0(엔진 패치 버전 4.0.4448) • MongoDB 8.0 API 드라이버에 대한 지원이 추가되었습니다. • 집계 파이프라인 최적화 및 개별 명령 지원과 함께 집계 단계 연산자의 성능 개선을 확장하는 플래너 Version3에 대한 지원이 추가되었습니다. • Zstandard 압축 알고리즘을 통한 사전 기반 압축에 대한 지원을 추가하여 압축 비율을 5배 개선했습니다. • 새로운 기능 추가: Amazon DocumentDB 8.0은 데이터 정렬 및 뷰를 지원합니다. • $replaceWith, $vectorSearch, $merge, $set, $unset, $bucket 및 3개의 새 집계 연산자 $pow, $rand, $dateTrunc에 대한 지원이 추가되었습니다. • 새로운 버전의 텍스트 인덱스에 대한 지원 추가: Amazon DocumentDB 8.0의 텍스트 인덱스 v2에는 추가 토큰이 도입되어 텍스트 검색 기능이 향상되었습니다. • 병렬 벡터 인덱스 빌드를 통한 벡터 검색 개선 사항을 포함하여 인덱스 빌드 시간을 최대 30배 단축합니다. • TLS 1.0 및 TLS 1.1에 대한 지원이 제거되었습니다. 이제 최소 TLS 버전은 TLS 1.2입니다.

Amazon DocumentDB 5.0

Amazon DocumentDB 엔진 버전 5.0(MongoDB 5.0 호환성)에 대한 릴리스 정보입니다.

Amazon DocumentDB 5.0.2

Date	릴리스 노트
2026년 9월 28일	Amazon DocumentDB 5.0.2(엔진 버전: 3.0.23065) 새로운 기능 및 개선 사항 1. 모든 인덱스 유형(플래너 버전 2 이상)에 대한 숨겨진 인덱스 지원이 추가되었습니다. 2. find 명령에서 `min()` 및 `max()` 커서 메서드를 지원합니다. 3. 모든 명령에 인덱스 필터 지원이 추가되었습니다. 4. 쿼리 플래너는 이제 등식(`$eq`) 조건자를 위한 고유 인덱스의 우선 순위를 지정합니다. 5. `currentOp` 출력의 집계 명령에 대한 잘림 제한을 늘렸습니다. 버그 수정 1. 버전 1(기본값) 쿼리 플래너의 버그를 수정했습니다. 2. 메이저 버전 업그레이드(MVU) 중에 부분 인덱스 표현식 텍스트 왕복 처리를 수정했습니다. 3. 빠른 수 쿼리에 대한 프로파일러 문제를 수정했습니다.

Amazon DocumentDB 5.0.1

Date	릴리스 노트
2026년 8월 18일	보안 및 안정성 개선 Amazon DocumentDB 5.0.1(엔진 버전: 3.0.22319) [진행 중] 이 엔진 패치는 다음과 같은 개선 사항을 제공합니다. • 요청 처리 개선 Amazon DocumentDB는 이제 연결 및 메시지 구문 분석 경로를 따라 수신 클라이언트 요청을 더 철저하게 검증합니다. 이렇게 하면 광범위한 입력 패턴에서 복원력과 안정성이 향상됩니다.

Date	릴리스 노트
	• 인증 핸드셰이크 강화 Amazon DocumentDB는 이제 인증 교환을 더 철저하게 검증합니다. 이렇게 하면 예기치 않거나 잘못된 형식의 입력을 처리할 때 견고성이 향상됩니다. 이 패치를 더 빨리 적용하지 않으면 Amazon DocumentDB는 릴리스 후 30일이 지난 다음 유지 관리 기간에이 패치를 자동으로 적용합니다.
2026년 8월 11일	새로운 기능 Amazon DocumentDB 5.0.1(엔진 버전: 3.0.21138) SCRAM-SHA-256 인증에 대한 지원을 활성화했습니다.

Date	릴리스 노트
2026년 6월 8일	새 마이너 버전 Amazon DocumentDB 5.0.1(엔진 버전 3.0.21138) Amazon DocumentDB 5.0.1은 Amazon DocumentDB 5.0 메이저 버전의 첫 번째 마이너 버전 릴리스입니다. 마이너 버전에는 새로운 기능, 보안 수정 및 버그 수정과 같은 개선 사항이 포함되어 있습니다. 마이너 버전 업그레이드를 사용하여 5.0.0에서 5.0.1로 업그레이드할 수 있습니다. 자세한 내용은 Amazon DocumentDB 마이너 버전 업그레이드 섹션을 참조하세요. 이 마이너 버전에는 다음과 같은 새로운 기능과 개선 사항이 포함되어 있습니다. 1. 가비지 수집 프로세스의 전체 상태에 대한 추가 정보를 표시collstats 하기 위해 및 gcRuntimeStats storageSegmentExtended 에 수집 수준 지표의 documentFragmentStats , storageSegmentBase 및 분류를 추가했습니다. 세부 정보는 Amazon DocumentDB의 폐영역 회수 섹션을 참조하세요. 2. $rand 및 $pow 집계 연산자를 지원합니다. 3. 이제 인스턴스에 대한 Connections (active) 지표를 모니터링할 수 있습니다. 자세한 내용은 인스턴스 할당량을 참조하세요. 4. Global Survivable Read Only를 사용하여 보조 클러스터 재부팅을 제거하여 글로벌 클러스터 리전 간 복원력을 개선했습니다. 5. $dateToParts 및 $dateFromParts 집계 연산자를 지원합니다. 6. 이제 CloudWatch에서 명령 수준 성능 지표를 CRUD 작업(find, , insert, findAndModify , updatedelete, aggregate , count, distinct, getMore, abortTransaction , commitTransaction), 노출AvgDuration , P100Duration , MaxConcurrent 차원에 사용할 수 있어 운영 모니터링이 개선되었습니다. 7. $replaceRoot 는 부울 값(true/false)과 숫자 값(0/)이 포함된 newRoot 필드를 출력합니다1.

Date	릴리스 노트
	8. `$indexOfArray` 는 더 광범위한 유효한 쿼리 표현식을 지원합니다. 9. 설명 계획에는 `executionStats` 출력에 `docsExamined` 지표가 포함됩니다. 10.SASL 인증 성능이 개선되었습니다.

Amazon DocumentDB 5.0.0

Date	릴리스 노트
2026년 8월 18일	보안 및 안정성 개선 Amazon DocumentDB 5.0.0(엔진 버전: 3.0.22303) [진행 중] 이 엔진 패치는 다음과 같은 개선 사항을 제공합니다. • 요청 처리 개선 Amazon DocumentDB는 이제 연결 및 메시지 구문 분석 경로를 따라 수신 클라이언트 요청을 더 철저하게 검증합니다. 이렇게 하면 광범위한 입력 패턴에서 복원력과 안정성이 향상됩니다. • 인증 핸드셰이크 강화 Amazon DocumentDB는 이제 인증 교환을 더 철저하게 검증합니다. 이렇게 하면 예기치 않거나 잘못된 형식의 입력을 처리할 때 견고성이 향상됩니다. • 액세스 제어 적용 Amazon DocumentDB는 이제 중첩 쿼리 작업에 대해 모든 중첩 수준에서 필요한 권한을 일관되게 확인합니다. 이 패치를 더 빨리 적용하지 않으면 Amazon DocumentDB는 릴리스 후 30일이 지난 다음 유지 관리 기간에이 패치를 자동으로 적용합니다.
2026년 5월 6일	버그 수정 및 기타 변경 사항 Amazon DocumentDB 5.0(엔진 버전: 3.0.20421) 이 릴리스는 필수 보안 패치용입니다.
2026년 4월 20일	새로운 기능

Date	릴리스 노트
	이제 Amazon DocumentDB 5.0에서 Amazon DocumentDB 8.0으로의 인플레이스 메이저 버전 업그레이드(MVU)를 사용할 수 있습니다. 자세한 내용은 Amazon DocumentDB 인플레이스 주요 버전 업그레이드 섹션을 참조하세요.
2026년 4월 13일	버그 수정 및 기타 변경 사항 Amazon DocumentDB 5.0(엔진 버전: 3.0.19917) 이는 다음과 같은 새로운 기능, 개선 사항 및 수정 사항이 포함된 필수 엔진 패치입니다. 1. 이제 `DatabaseConnectionsActiveMax` 지표가 인스턴스`Connections (active)` 를 추적합니다. 자세한 내용은 인스턴스 할당량을 참조하세요. 2. SASL 인증 성능이 개선되었습니다. 모든 Amazon DocumentDB 5.0 클러스터는 다음 유지 관리 기간 동안 이 패치 버전으로 자동 업그레이드됩니다.

Date	릴리스 노트
2025년 10월 22일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.17983) Amazon DocumentDB는 이제 Arm-based AWS Graviton4 프로세서로 구동되는 최신 메모리 최적화 인스턴스 db.r8g를 지원하여 R6G 인스턴스보다 최대 30% 더 나은 성능을 제공합니다. 자세한 내용은 이 설명서의 인스턴스 클래스 관리 섹션 및 이 블로그 게시물을 참조하세요. 새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.15902) • 더 긴 컬렉션 이름(네임스페이스 길이 최대 255자)에 대한 지원이 추가되었습니다. • 새 쿼리 플래너(플래너 버전 2.0)를 도입했습니다. 자세한 내용은 이 안내서의 쿼리 플래너 v2 섹션을 참조하세요. • 폐영역 회수 및 노후화 컬렉션과 관련된 컬렉션의 전체 상태에 대한 정보를 제공하는 'MVCCIDStats' 및 'gcRuntimeStats'에 대한 `collstats` 의 새 필드가 추가되었습니다.
2025년 10월 16일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.15902) Amazon DocumentDB는 이제 클러스터가 가상 프라이빗 클라우드(VPC)에 있을 때 이중 스택 모드(IPv4/IPv6)를 지원합니다. 자세한 내용은 이 블로그 게시물과 Amazon DocumentDB 개발자 안내서의 Amazon VPC 및 Amazon DocumentDB를 참고하시기 바랍니다.
2025년 9월 15일	버그 수정 및 기타 변경 사항 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.17983) 쿼리에 후속 `$replaceRoot` 및 `$lookup` 단계가 포함된 경우 엔진 충돌을 수정했습니다.

Date	릴리스 노트
2025년 7월 29일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.15902) • `$regexFindAll` 집계 연산자에 대한 지원이 추가되었습니다. • 시작된 인덱스 팽창 및 컬렉션 기반 팽창 지표 • 폐영역 회수 및 노후화 컬렉션과 관련된 클러스터의 전체 상태에 대한 정보를 제공하는 CloudWatch 지표 `AvailableMVCCIds` 및 `LongestActiveGCRuntime` 가 추가되었습니다. • `serverStatus` 명령에는 이제 NVMe 지원 인스턴스에서 NVMe 캐시에 기록되거나 기록되지 않은 페이지 수를 보여주는 'nvme_cache' 필드가 포함됩니다. • `$group` 및 `$project`의 필드 수 제한 제거 • `$trim`, `$rtrim`, `$ltrim`, `$regexFindAll` , `$replaceOne` , 및 `$replaceAll` 문자열 연산자에 대한 지원 • `$first` 및 `$last` 배열 연산자에 대한 지원 • `$collStats` 스테이지 연산자에 대한 지원 버그 수정 및 기타 변경 사항 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.15902) 인덱스 생성 중에 대규모 컬렉션에 영향을 미치는 문제를 수정했습니다.
2025년 7월 28일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.14228) Amazon DocumentDB 인스턴스 기반 클러스터는 이제 온디맨드 오토 스케일링 구성인 Amazon DocumentDB Serverless를 지원합니다. 자세한 내용은 Amazon DocumentDB Serverless 사용 및 이 새로운 소식 블로그 게시물을 참조하세요.

Date	릴리스 노트
2025년 3월 24일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.11800) 변경 스트림에서 `postBatchResumeToken` 에 대한 지원이 추가되었습니다. 자세한 내용은 `postBatchResumeToken` 에서 변경 스트림 재개 섹션을 참조하세요.
2025년 1월 15일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.11800) Amazon DocumentDB는 이제 라이터 인스턴스 재시작을 통해 읽기 가용성을 유지합니다. 이제 리더 인스턴스는 라이터 인스턴스를 다시 시작하는 동안 읽기 요청을 계속 처리합니다. 버그 수정 및 기타 변경 사항 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.11800) • 대량 삽입의 특수 사례를 처리하도록 `killOp` 명령을 수정했습니다. • Amazon DocumentDB 인스턴스에서 네트워크 I/O 관련 메모리 사용량을 개선했습니다. • `$text` 필터 쿼리에 대한 `count` 명령을 수정했습니다.
2024년 12월 18일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.5721) Amazon DocumentDB 인스턴스 기반 클러스터는 이제 NVMe 지원 R6GD 인스턴스를 지원합니다. 자세한 내용은 NVMe 지원 인스턴스 및 이 새로운 소식 블로그 게시물을 참조하세요.

Date	릴리스 노트
2024년 11월 6일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.11051) • 최소 TLS 버전 `tls1.2+` 및 `tls1.3+`에 대한 지원이 `tls` 클러스터 파라미터에 추가되었습니다. • 필드 이름에서 달러($) 및 점(.) 문자에 대한 지원이 활성화되었습니다. 기능적 차이점은 필드 이름의 달러($) 및 점(.) 섹션을 참조하세요.
2024년 9월 17일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.10696) 이제 Amazon DocumentDB는 `runCommand` 에서 `reIndex`를 사용하여 인덱스 재구축을 지원합니다. 자세한 내용은 `reIndex`를 사용한 인덱스 유지 관리 섹션을 참조하세요. Note `reIndex`는 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.10696 이상)에서만 지원됩니다. 버그 수정 및 기타 변경 사항 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.10696) • 이제 `$setOnInsert` 는 업데이트 작업으로 인한 삽입 중에 `_id` 필드를 지원합니다. • Amazon DocumentDB 5.0으로 메이저 버전을 업그레이드한 후 스토리지 공간을 회수하지 못하는 문제를 수정했습니다.

Date	릴리스 노트
2024년 8월 22일	새로운 기능 Amazon DocumentDB 5.0(모든 엔진 패치 버전) Amazon DocumentDB 글로벌 클러스터는 이제 클러스터 전환 및 관리형 클러스터 장애 조치를 지원합니다. 자세한 내용은 Amazon DocumentDB 글로벌 클러스터에 대한 전환 수행 및 Amazon DocumentDB 글로벌 클러스터에 대한 관리형 장애 조치 수행 섹션을 참조하세요. **Note** 글로벌 클러스터 전환 및 장애 조치는 Amazon DocumentDB 4.0 및 5.0에서만 지원됩니다.

<table>
<tr><th>Date</th><th>릴리스 노트</th></tr>
<tr><td>2024년 7월 23일</td><td>
새로운 기능

Amazon DocumentDB 5.0(엔진 패치 버전 3.0.8126)

• 더 긴 인덱스 이름(최대 255자)에 대한 지원이 추가되었습니다. 자세한 내용은 명명 제약 조건 섹션을 참조하세요.
• 최대 연결 한도를 두 배로 늘렸습니다.

<table>
<tr><th>인스턴스 유형</th><th>원래 한도</th><th>신규 한도</th></tr>
<tr><td>t3.medium</td><td>500</td><td>1000</td></tr>
<tr><td>t4g.medium</td><td>500</td><td>1000</td></tr>
<tr><td>r5.large</td><td>1700</td><td>3400</td></tr>
<tr><td>r5.xlarge</td><td>3500</td><td>7000</td></tr>
<tr><td>r5.2xlarge</td><td>7100</td><td>14200</td></tr>
<tr><td>r5.4xlarge</td><td>14200</td><td>28400</td></tr>
<tr><td>r5.8xlarge</td><td>28400</td><td>60000</td></tr>
<tr><td>r5.12xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r5.16xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r5.24xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r6g.large</td><td>1700</td><td>3400</td></tr>
<tr><td>r6g.xlarge</td><td>3500</td><td>7000</td></tr>
<tr><td>r6g.2xlarge</td><td>7100</td><td>14200</td></tr>
<tr><td>r6g.4xlarge</td><td>14200</td><td>28400</td></tr>
<tr><td>r6g.8xlarge</td><td>28400</td><td>60000</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Date</th><th>릴리스 노트</th></tr>
<tr><td></td><td>
<table>
<tr><th>인스턴스 유형</th><th>원래 한도</th><th>신규 한도</th></tr>
<tr><td>r6g.12xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r6g.16xlarge</td><td>30000</td><td>60000</td></tr>
</table>
자세한 내용은 인스턴스 할당량 섹션을 참조하세요.

버그 수정 및 기타 변경 사항

Amazon DocumentDB 5.0(엔진 패치 버전 3.0.8126)

리더의 변경 스트림에 대해 <code>CurrentTime</code> 및 <code>ResumeToken</code> 을 동기화하도록 로직을 개선했습니다.
</td></tr>
<tr><td>2024년 7월 22일</td><td>
새로운 기능

Amazon DocumentDB 5.0(엔진 패치 버전 3.0.6742)

• DML 감사 필터링 지원이 추가되었습니다. 이제 모든 DML 쿼리를 로깅하는 대신 특정 요구 사항에 따라 DML 감사 로그를 필터링하도록 필터 조건을 설정할 수 있습니다. 자세한 내용은 DML 감사 이벤트 필터링 섹션을 참조하세요.
• 다음에 대한 문서 압축 지원이 추가되었습니다.
  • 최소 압축 임계값 설정
  • 기존 컬렉션에 대한 압축 활성화(새 문서에 적용 가능)
  • 클러스터 수준에서 기본 압축 설정 허용

  자세한 내용은 컬렉션 수준 문서 압축 관리 섹션을 참조하세요.
• 리더 인스턴스에서 변경 스트림 소비에 대한 지원이 추가되었습니다. 자세한 내용은 보조 인스턴스에서 변경 스트림 사용 섹션을 참조하세요.
</td></tr>
</table>

Date	릴리스 노트
2024년 7월 9일	새로운 기능 Amazon DocumentDB 5.0(최소 엔진 패치 버전 3.0.4780)의 경우 TLS CA 인증서 업데이트 시 더 이상 시스템을 재부팅할 필요가 없습니다. 자세한 내용은 Amazon DocumentDB TLS 인증서 업데이트 섹션을 참조하세요.
2024년 5월 29일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.6742) • `$regexMatch` 및 `$regexFind` 연산자에 대한 지원이 추가되었습니다. • 큰 정수를 처리할 때 감사 로그의 전체 정밀도를 보장하기 위한 지원이 추가되었습니다. 이제 감사 로그는 모든 숫자에 대해 정확한 숫자 표현을 유지하여 정밀도 손실을 방지합니다.
2024년 4월 3일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.5721) • `bypassDocumentValidation` 에 대한 지원 및 `$jsonSchema` 에 대한 세분화된 오류 메시지가 추가되었습니다. `bypassDocumentValidation` 에 대한 자세한 내용은 bypassDocumentValidation 섹션을 참조하세요. • `$expr`의 지원이 추가되었습니다. • `$lookup`에 상관 없는 조인에 대한 지원이 추가되었습니다. • `$out` 집계 단계에서 검증 규칙을 유지하기 위한 지원이 추가되었습니다.

Date	릴리스 노트
2024년 1월 10일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.4574, 3.0.4780, 3.0.4960) • HNSW 벡터 인덱스에 대한 지원이 추가되었습니다. 자세한 내용은 Amazon DocumentDB에 대한 벡터 검색 섹션을 참조하세요. • 부분 인덱스에 대한 지원이 추가되었습니다. 자세한 내용은 부분 인덱스 섹션을 참조하세요. • `currentOp` 명령 내 컬렉션에서 GC 런타임에 대한 지원이 추가되었습니다. • Amazon DocumentDB의 기본 텍스트 검색에 대한 텍스트 인덱스 지원이 추가되었습니다. 자세한 내용은 Amazon DocumentDB를 사용하여 텍스트 검색 수행 섹션을 참조하세요. • `$jsonSchema` 스키마 키워드 `type`, `allOf`, `oneOf`, `anyOf`, `not`, `maxItems`, `minItems`, `maxProperties` , `minProperties` , `pattern`, `patternProperties` , `multipleOf` , `dependencies` 및 `uniqueItems` 에 대한 지원이 추가되었습니다. 자세한 내용은 JSON 스키마 검증 사용 섹션을 참조하세요. • 산술 연산자 `$ceil`, `$floor`, `$ln`, `$log`, `$log10`, `$sqrt` 및 `$exp`에 대한 지원이 추가되었습니다. 자세한 내용은 산술 연산자 섹션을 참조하세요. • 조건식 연산자 `$switch`에 대한 지원이 추가되었습니다. • 병렬 `IVFFLAT` 벡터 인덱스 빌드에 대한 지원이 추가되었습니다. 개발자 안내서에서 병렬 `IVFFLAT` 벡터 인덱스 빌드 제한을 제거하여 설명서가 업데이트되었습니다. 버그 수정 및 기타 변경 사항 • `db.runCommand("dbstats")` 호출 시 대/소문자를 구분하지 않는 기능이 추가되었습니다. 3.0.4960 또는 2.0.10221 이전 엔진 패치

Date	릴리스 노트
	버전의 Amazon DocumentDB 5.0고객은 이러한 최신 엔진 패치를 적용해야 합니다. • mongo 쉘 버전 1.7 이상에서 `db.coll.stats()` 를 호출할 때 발생하는 오류를 수정했습니다. 개발자 안내서에서 mongo 쉘 `db.coll.stats()` 문제 해결 팁을 제거하여 설명서가 업데이트되었습니다.
2023년 11월 29일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.3727) 벡터 검색에 대한 지원이 추가되었습니다. 자세한 내용은 이 블로그 게시물과 Amazon DocumentDB 개발자 안내서의 Amazon DocumentDB에 대한 벡터 검색를 참고하시기 바랍니다.
2023년 11월 21일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.3727) I/O 최적화 스토리지에 대한 지원이 추가되었습니다. 자세한 내용은 Amazon DocumentDB 개발자 안내서의 Amazon DocumentDB 클러스터 스토리지 구성를 참조하세요. SageMaker Canvas와 코드 없는 기계 학습을 위한 통합이 추가되었습니다. 자세한 내용은 Amazon DocumentDB 개발자 안내서의 Amazon SageMaker AI Canvas를 사용한 코드 없는 기계 학습를 참조하세요.

Date	릴리스 노트
2023년 11월 6일	Amazon DocumentDB 5.0(엔진 패치 버전 3.0.3727) 새로운 기능 • `$jsonSchema` 스키마 키워드`maxLength` , `minLength` , `maximum`, `minimum`, `exclusiveMaximum` , `exclusive Minimum` , `items`, 및 `additionalItems` 에 대한 지원이 추가되었습니다. JSON 스키마 검증은 인스턴스 기반 클러스터에서만 지원됩니다. • `$convert`집계 파이프라인 연산자 및 속기 파생 연산자 `$toBool`, `$toInt`, `$toLong`, `$toDouble` , `$toString` , `$toDecimal` , `$toObjectId` 및 `$toDate`에 대한 지원이 추가되었습니다. • 집합 표현식 연산자`$setDifference` , `$anyElementTrue` 및 `$allElementTrue` 에 대한 지원이 추가되었습니다. 버그 수정 및 기타 변경 사항 `-NaN`에서 `NaN`로 변경 스트림 업데이트가 표시되지 않던 문제가 수정되었습니다.
2023년 9월 15일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.3140) 인스턴스 기반 클러스터에서만 $jsonSchema 스키마 검사기에 대한 지원이 추가되었습니다. 자세한 내용은 JSON 스키마 검증 사용 섹션을 참조하세요.

Date	릴리스 노트
2023년 7월 13일	새로운 기능 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.1948) • 문서 압축에 대한 지원이 추가되었습니다. • 병렬 인덱스 빌드에 대한 지원이 추가되었습니다. • 인덱스 빌드 상태에 대한 지원이 추가되었습니다. 버그 수정 및 기타 변경 사항 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.1948) • 사용자가 시스템 컬렉션에 액세스할 수 없을 때 Amazon DocumentDB Elastic 클러스터의 `createCollection` 에서 발생하는 인증 문제를 수정했습니다. • 보조 리전 인스턴스가 동일한 기본 리전 인스턴스 이름을 사용할 수 없는 문제를 수정했습니다.
2023년 6월 7일	버그 수정 및 기타 변경 사항 Amazon DocumentDB 5.0 • 이제 Amazon DocumentDB 5.0에서 r5 및 t3.medium 인스턴스가 지원됩니다. • `engineVersion` 옵션 기본값은 AWS SDK AWS CLI, 및 `5.0.0` 입니다 CloudFormation.
2023년 5월 10일	버그 수정 및 기타 변경 사항 Amazon DocumentDB 5.0(엔진 패치 버전 3.0.1361) • createIndex 명령에서 `ignoreunknownindexoptions` 에 대한 지원을 추가했습니다. • 감사 로그에 내부 모니터링 쿼리를 추가하는 것을 중단했습니다. • 사용자 정의 로컬 변수 이름은 이제 `$let` 및 `$filter`와 같은 프로젝션 연산자에 대해 “_”(밑줄)을 지원합니다.

Date	릴리스 노트

2023년 3월 1일	새로운 기능
	
	Amazon DocumentDB 5.0(엔진 패치 버전 3.0.775)
	
	• Amazon DocumentDB 5.0 출시
	  • 몽고DB 5.0 호환성(MongoDB 5.0 API 드라이버 지원)
	  • 클라이언트측 필드 레벨 암호화 FLE) 지원 이제 Amazon DocumentDB 클러스터에 데이터를 쓰기 전에 클라이언트 측에서 필드를 암호화할 수 있습니다. 자세한 내용은 클라이언트 측 필드 수준 암호화를 참조하세요.
	  • 새 집계 연산자: `$dateAdd`, `$dateSubtract`
	• 모든 인스턴스 기반 Amazon DocumentDB 클러스터와 샤드 기반 Elastic 클러스터의 스토리지 한도를 128TiB로 늘렸습니다.
	• Amazon DocumentDB 5.0은 이제 첫 번째 중첩 수준에서 `$elemMatch` 운영자를 사용한 인덱스 스캔을 지원합니다. 쿼리 전용 필터에 한 레벨의 `$elemMatch` 필터가 있는 경우 인덱스 스캔이 지원되고 중첩된 `$elemMatch` 쿼리는 인덱스 스캔을 지원하지 않습니다.
	
	인덱스 스캔을 지원하는 쿼리 셰이프:
	
	`db.foo.find( { "a": {$elemMatch: { "b": "xyz", "c": "abc"} } })`
	
	인덱스 스캔을 지원하지 않는 쿼리 셰이프:
	
	`db.foo.find( { "a": {$elemMatch: { "b": {$elemMatch: { "d": "xyz", "e": "abc"} }} } })`

Amazon DocumentDB 4.0

Amazon DocumentDB 엔진 버전 4.0(MongoDB 4.0 호환성)에 대한 릴리스 정보입니다.

Date	릴리스 노트
2026년 8월 31일	새로운 기능 Amazon DocumentDB는 이제 엔진 버전 4.0에서 버전 8.0으로 인플레이스 메이저 버전 업그레이드(MVU)를 직접 지원합니다. 따라서 중간 버전 업그레이드가 필요하지 않으며 기존 데이터, 구성 및 클러스터 설정을 유지하면서 4.0 클러스터를 업그레이드할 수 있습니다. 이 업그레이드는 현재 이러한 버전이 지원되는 모든 AWS 리전에서 사용할 수 있습니다. 자세한 내용은 Amazon DocumentDB 인플레이스 주요 버전 업그레이드 섹션을 참조하세요.
2026년 8월 18일	보안 및 안정성 개선 Amazon DocumentDB 4.0(엔진 버전: 2.0.14287) [진행 중] 이 엔진 패치는 다음과 같은 개선 사항을 제공합니다. • 요청 처리 개선 Amazon DocumentDB는 이제 연결 및 메시지 구문 분석 경로를 따라 수신 클라이언트 요청을 더 철저하게 검증합니다. 이렇게 하면 광범위한 입력 패턴에서 복원력과 안정성이 향상됩니다. • 인증 핸드셰이크 강화 Amazon DocumentDB는 이제 인증 교환을 더 철저하게 검증합니다. 이렇게 하면 예기치 않거나 잘못된 형식의 입력을 처리할 때 견고성이 향상됩니다. • 액세스 제어 적용 Amazon DocumentDB는 이제 중첩 쿼리 작업에 대해 모든 중첩 수준에서 필요한 권한을 일관되게 확인합니다. 이 패치를 더 빨리 적용하지 않으면 Amazon DocumentDB는 릴리스 후 30일이 지난 다음 유지 관리 기간에이 패치를 자동으로 적용합니다.
2026년 4월 27일	버그 수정 및 기타 변경 사항 Amazon DocumentDB 4.0(엔진 버전: 2.0.14278) 이는 다음 문제를 해결하는 필수 엔진 패치입니다. 1. 드문 경우지만 point-in-time 복구(PITR) 및 메이저 버전 업그레이드(MVU)가 응답하지 않을 수 있는 문제를 수정했습니다.

Date	릴리스 노트
	모든 Amazon DocumentDB 4.0 클러스터는 다음 유지 관리 기간 동안 이 패치 버전으로 자동 업그레이드됩니다.
2026년 3월 9일	버그 수정 및 기타 변경 사항 Amazon DocumentDB 4.0(엔진 버전: 2.0.14165) 이 업데이트는 다음과 같은 개선 사항이 적용된 선택적 엔진 패치를 제공합니다. 1. 이제 `DatabaseConnectionsActiveMax` 지표가 인스턴스`Connections (active)` 를 추적합니다. 자세한 내용은 인스턴스 할당량을 참조하세요. 2. SASL 인증 성능이 개선되었습니다.
2025년 10월 22일	새로운 기능 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.11747) 폐영역 회수 및 노후화 컬렉션과 관련된 컬렉션의 전체 상태에 대한 정보를 제공하는 'MVCCIDStats' 및 'gcRuntimeStats'에 대한 `collstats` 의 새 필드가 추가되었습니다.
2025년 10월 16일	새로운 기능 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.11747) Amazon DocumentDB는 이제 클러스터가 가상 프라이빗 클라우드(VPC)에 있을 때 이중 스택 모드(IPv4/IPv6)를 지원합니다. 자세한 내용은 이 블로그 게시물과 Amazon DocumentDB 개발자 안내서의 Amazon VPC 및 Amazon DocumentDB를 참고하시기 바랍니다.

Date	릴리스 노트
2025년 7월 29일	새로운 기능 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.11747) • 시작된 인덱스 팽창 및 컬렉션 기반 팽창 지표 • 폐영역 회수 및 노후화 컬렉션과 관련된 클러스터의 전체 상태에 대한 정보를 제공하는 CloudWatch 지표 `AvailableMVCCIds` 및 `LongestActiveGCRuntime` 가 추가되었습니다. • `$trim`, `$rtrim`, 및 `$ltrim` 문자열 연산자에 대한 지원 • `$collStats` 스테이지 연산자에 대한 지원 버그 수정 및 기타 변경 사항 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.11747) 인덱스 생성 중에 대규모 컬렉션에 영향을 미치는 문제를 수정했습니다.
2025년 1월 15일	버그 수정 및 기타 변경 사항 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.11096) • 대량 삽입의 특수 사례를 처리하도록 `killOp` 명령을 수정했습니다. • Amazon DocumentDB 인스턴스에서 네트워크 I/O 관련 메모리 사용량을 개선했습니다.
2024년 11월 6일	새로운 기능 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.10980) • 최소 TLS 버전 `tls1.2+` 및 `tls1.3+`에 대한 지원이 `tls` 클러스터 파라미터에 추가되었습니다.

Date	릴리스 노트
2024년 9월 17일	버그 수정 및 기타 변경 사항 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.10898) • 이제 `$setOnInsert` 는 업데이트 작업으로 인한 삽입 중에 `_id` 필드를 지원합니다. • Amazon DocumentDB 5.0으로 메이저 버전을 업그레이드한 후 스토리지 공간을 회수하지 못하는 문제를 수정했습니다.
2024년 8월 22일	새로운 기능 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.5704) Amazon DocumentDB 글로벌 클러스터는 이제 클러스터 전환 및 관리형 클러스터 장애 조치를 지원합니다. 자세한 내용은 Amazon DocumentDB 글로벌 클러스터에 대한 전환 수행 및 Amazon DocumentDB 글로벌 클러스터에 대한 관리형 장애 조치 수행 섹션을 참조하세요. **Note** 글로벌 클러스터 전환 및 장애 조치는 Amazon DocumentDB 4.0 및 5.0에서만 지원됩니다.

<table>
<tr><th>Date</th><th>릴리스 노트</th></tr>
<tr><td>2024년 7월 23일</td><td>새로운 기능

Amazon DocumentDB 4.0(엔진 패치 버전 2.0.10709)

• 더 긴 인덱스 이름(최대 255자)에 대한 지원이 추가되었습니다. 자세한 내용은 명명 제약 조건 섹션을 참조하세요.
• 최대 연결 한도를 두 배로 늘렸습니다.

<table>
<tr><th>인스턴스 유형</th><th>원래 한도</th><th>신규 한도</th></tr>
<tr><td>t3.medium</td><td>500</td><td>1000</td></tr>
<tr><td>t4g.medium</td><td>500</td><td>1000</td></tr>
<tr><td>r5.large</td><td>1700</td><td>3400</td></tr>
<tr><td>r5.xlarge</td><td>3500</td><td>7000</td></tr>
<tr><td>r5.2xlarge</td><td>7100</td><td>14200</td></tr>
<tr><td>r5.4xlarge</td><td>14200</td><td>28400</td></tr>
<tr><td>r5.8xlarge</td><td>28400</td><td>60000</td></tr>
<tr><td>r5.12xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r5.16xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r5.24xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r6g.large</td><td>1700</td><td>3400</td></tr>
<tr><td>r6g.xlarge</td><td>3500</td><td>7000</td></tr>
<tr><td>r6g.2xlarge</td><td>7100</td><td>14200</td></tr>
<tr><td>r6g.4xlarge</td><td>14200</td><td>28400</td></tr>
<tr><td>r6g.8xlarge</td><td>28400</td><td>60000</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Date</th><th>릴리스 노트</th></tr>
<tr><td></td><td>
<table>
<tr><th>인스턴스 유형</th><th>원래 한도</th><th>신규 한도</th></tr>
<tr><td>r6g.12xlarge</td><td>30000</td><td>60000</td></tr>
<tr><td>r6g.16xlarge</td><td>30000</td><td>60000</td></tr>
</table>
자세한 내용은 인스턴스 할당량 섹션을 참조하세요.
</td></tr>
<tr><td>2024년 7월 22일</td><td>새로운 기능

Amazon DocumentDB 4.0(엔진 패치 버전 2.0.10593)

• DML 감사 필터링 지원이 추가되었습니다. 이제 모든 DML 쿼리를 로깅하는 대신 특정 요구 사항에 따라 DML 감사 로그를 필터링하도록 필터 조건을 설정할 수 있습니다. 자세한 내용은 DML 감사 이벤트 필터링 섹션을 참조하세요.</td></tr>
<tr><td>2024년 7월 9일</td><td>새로운 기능

Amazon DocumentDB 4.0(최소 엔진 패치 버전 2.0.10179)의 경우 TLS CA 인증서 업데이트 시 더 이상 시스템을 재부팅할 필요가 없습니다. 자세한 내용은 Amazon DocumentDB TLS 인증서 업데이트 섹션을 참조하세요.</td></tr>
<tr><td>2024년 5월 29일</td><td>새로운 기능

Amazon DocumentDB 4.0(엔진 패치 버전 2.0.10593)

• 큰 정수를 처리할 때 감사 로그의 전체 정밀도를 보장하기 위한 지원이 추가되었습니다. 이제 감사 로그는 모든 숫자에 대해 정확한 숫자 표현을 유지하여 정밀도 손실을 방지합니다.</td></tr>
</table>

Date	릴리스 노트
2024년 4월 3일	새로운 기능 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.10392) • `$jsonSchema` 에 대한 `bypassDocumentValidation` 지원을 추가했습니다. `bypassDocumentValidation` 에 대한 자세한 내용은 bypassDocumentValidation 섹션을 참조하세요. • `$expr`의 지원이 추가되었습니다. • `$lookup`에 상관 없는 조인에 대한 지원이 추가되었습니다. • `$out` 집계 단계에서 검증 규칙을 유지하기 위한 지원이 추가되었습니다.
2024년 1월 10일	새로운 기능 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.10124, 2.0.10179, 2.0.10221) • `currentOp` 명령 내 컬렉션에서 GC 런타임에 대한 지원이 추가되었습니다. • `$jsonSchema` 스키마 키워드 `type`, `allOf`, `oneOf`, `anyOf`, `not`, `maxItems`, `minItems`, `maxProperties` , `minProperties` , `pattern`, `patternProperties` , `multipleOf` , `dependencies` 및 `uniqueItems` 에 대한 지원이 추가되었습니다. 자세한 내용은 JSON 스키마 검증 사용 섹션을 참조하세요. • 산술 연산자 `$ceil`, `$floor`, `$ln`, `$log`, `$log10`, `$sqrt` 및 `$exp`에 대한 지원이 추가되었습니다. 자세한 내용은 산술 연산자 섹션을 참조하세요. • 조건식 연산자 `$switch`에 대한 지원이 추가되었습니다.
2023년 12월 20일	기타 변경사항 Amazon DocumentDB 4.0에서 인플레이스 메이저 버전 업그레이드에 대한 지원이 활성화되었습니다. 자세한 내용은 Amazon DocumentDB 인플레이스 주요 버전 업그레이드 섹션을 참조하세요.

Date	릴리스 노트
2023년 11월 6일	Amazon DocumentDB 4.0(엔진 패치 버전 2.0.9876) 새로운 기능 • `$jsonSchema` 스키마 키워드`maxLength` , `minLength` , `maximum`, `minimum`, `exclusiveMaximum` , `exclusive Minimum` , `items`, 및 `additionalItems` 에 대한 지원이 추가되었습니다. JSON 스키마 검증은 인스턴스 기반 클러스터에서만 지원됩니다. • `$convert`집계 파이프라인 연산자 및 속기 파생 연산자 `$toBool`, `$toInt`, `$toLong`, `$toDouble` , `$toString` , `$toDecimal` , `$toObjectId` 및 `$toDate`에 대한 지원이 추가되었습니다. • 집합 표현식 연산자`$setDifference` , `$anyElementTrue` 및 `$allElementTrue` 에 대한 지원이 추가되었습니다. 버그 수정 및 기타 변경 사항 `-NaN`에서 `NaN`로 변경 스트림 업데이트가 표시되지 않던 문제가 수정되었습니다.
2023년 9월 20일	새로운 기능 Amazon DocumentDB 4.0의 인플레이스 메이저 버전 업그레이드에 대한 지원이 추가되었습니다. 자세한 내용은 Amazon DocumentDB 인플레이스 주요 버전 업그레이드 섹션을 참조하세요.
2023년 9월 15일	새로운 기능 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.9686) 인스턴스 기반 클러스터에서만 $jsonSchema 스키마 검사기에 대한 지원이 추가되었습니다. 자세한 내용은 JSON 스키마 검증 사용 섹션을 참조하세요.

Date	릴리스 노트
2023년 7월 13일	새로운 기능 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.9259) • 병렬 인덱스 빌드에 대한 지원이 추가되었습니다. 버그 수정 및 기타 변경 사항 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.9259) • 감사 로그에 내부 모니터링 쿼리를 추가하는 것을 중단했습니다.
2023년 4월 4일	버그 수정 및 기타 변경 사항 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.8934) • 워크로드가 진행 중인 동안 DML 감사가 활성화된 경우 발생하는 문제가 해결되었습니다. • 힌트가 포함된 집계 명령에 문자열 값이 전달될 때 발생하는 DML 감사 문제가 해결되었습니다. • readwriteanydatabase 역할을 사용하는 사용자가 authorizedCollections와 nameOnly 옵션을 모두 true로 설정한 경우 `listCollections` 명령이 작동하지 않는 문제가 해결되었습니다. • 필드 이름의 숫자 문자열을 제대로 구문 분석하기 위한 문제가 해결되었습니다. • 장시간 실행되는 커서가 가비지 수집에 영향을 줄 경우, 해당 커서를 취소하세요. • 사용자 정의 로컬 변수 이름은 이제 `$let` 및 `$filter`와 같은 프로젝션 연산자에 대해 “_”(밑줄)을 지원합니다.
2023년 2월 27일	버그 수정 및 기타 변경 사항 Amazon DocumentDB 4.0 에 대한 지원이 추가되었습니다 AWS Lambda. 자세한 내용은 변경 스트림과 AWS Lambda 함께 사용을 참조하세요.

Date	릴리스 노트
2022년 8월 9일	새로운 기능 4.0 • Decimal128 데이터 유형에 대한 지원이 추가되었습니다. Decimal128은 DocumentDB를 사용할 수 있는 모든 리전에서 지원되는 BSON 데이터 형식입니다. 자세한 정보는 데이터 형식을 참조하십시오. • Amazon CloudWatch Logs에 DML 쿼리 감사에 대한 지원이 추가되었습니다. 이제 Amazon DocumentDB는 데이터 조작 언어(DML) 이벤트 및 데이터 정의 언어(DDL) 이벤트를 Amazon CloudWatch Logs에 기록할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오. 버그 수정 및 기타 변경 사항 4.0 • 이제 `changeOwnPassword` 권한이 있는 자체 비밀번호로 비밀번호를 변경할 수 있습니다.
2022년 7월 25일	새로운 기능 Amazon DocumentDB 4.0 이제 동일한 DocumentDB 클러스터 볼륨을 사용하고 원본 클러스터와 동일한 데이터를 갖는 클론을 생성할 수 있어 클러스터를 더 빠르게 생성할 수 있습니다. 자세한 내용은 Amazon DocumentDB 클러스터 관리를 참조하십시오.

Date	릴리스 노트
2022년 6월 27일	새로운 기능 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.7509) Amazon DocumentDB는 사용 패턴에 따라 데이터베이스 크기를 동적으로 조정합니다. 데이터를 더 추가하면 스페이스가 최대 64 테비바이트(TiB)까지 늘어나고 데이터를 삭제하면 할당된 공간이 줄어듭니다.
2022년 4월 7일	새로운 기능 4.0(엔진 패치 버전 2.0.6142 및 2.0.6948) Amazon DocumentDB Performance Insights는 현재 미리 보기 단계에 있습니다. 이제 추가 비용 없이 7일간의 성능 기록을 롤링 윈도우에 저장할 수 있습니다. 자세한 내용은 성능 인사이트를 사용한 모니터링을 참조하세요.

2022년 1월 24일	새로운 기능
	
	4.0(엔진 패치 버전 2.0.5170)
	
	• Amazon DocumentDB는 이제 무료 평가판을 제공합니다. 자세한 내용은 Amazon DocumentDB 무료 평가판 페이지를 참조하십시오.
	• 이제 지리 공간 쿼리에서 다음 API를 포함한 향상된 기능을 사용할 수 있습니다.
	  • `$geoWithin`
	  • `$geoIntersects`
	• 다음 MongoDB 연산자에 대한 지원이 추가되었습니다.
	  • `$mergeObjects`
	  • `$reduce`
	
	자세한 내용은 Amazon DocumentDB에 지리 공간 데이터 쿼리를 참조하십시오.

Date	릴리스 노트

2022년 1월 21일	새로운 기능
	
	Amazon DocumentDB 4.0(엔진 패치 버전 2.0.5706)
	
	• 이제 Amazon DocumentDB Graviton2(r6g.large, r6g.2xlarge, r6g.4xlarge, r6g.8xlarge, r6g.12xlarge, r6g.16xlarge, t4g.medium) 인스턴스가 지원됩니다.
	
	새로운 기능
	
	4.0(엔진 패치 버전 2.0.5706)
	
	• 다음 MongoDB API에 대한 지원이 추가되었습니다.
	  • `$reduce`
	  • `$mergeObjects`
	  • `$geoWithin`
	  • `$geoIntersects`

Date	릴리스 노트

2021년 10월 25일	새로운 기능
	
	4.0(엔진 패치 버전 2.0.5704)
	
	• 다음 MongoDB API에 대한 지원이 추가되었습니다.
	  • `$literal`
	  • `$map`
	  • `$$ROOT`
	• 지리 공간 쿼리 기능 지원 자세한 내용은 이 블로그 게시물을 참조하십시오.
	• 사용자 정의 역할을 통한 액세스 제어 지원 자세한 내용은 이 블로그 게시물을 참조하십시오.
	• Amazon DocumentDB JDBC 드라이버를 사용하면 Tableau와 같은 BI 도구와 SQL Workbench와 같은 쿼리 도구에서 연결할 수 있습니다.
	
	버그 수정 및 기타 변경 사항
	
	4.0(엔진 패치 버전 2.0.5704)
	
	• 다음과 같이 명시적인 `.sort()`와 `$natural`이 있을 때 올바르게 정렬되도록 `$natural` 버그 수정
	• `$redact`에 사용할 변경 스트림에 대한 버그 수정
	• 빈 배열과 함께 작동하기 위한 `$ifNull` 버그 수정
	• 현재 로그인한 사용자를 삭제하거나 진행 중인 활동에 대한 해당 사용자의 권한이 취소될 때 과도한 리소스 소비/서버 충돌이 발생하는 버그 수정
	• `listDatabase` 의 버그 수정 및 `listCollection` 권한 확인
	• 버그 수정 다중 키 요소에 대한 중복 제거 로직

Date	릴리스 노트
2021년 6월 24일	새로운 기능 4.0(엔진 패치 버전 2.0.3371) • 이제 r5.8xlarge 및 r5.16xlarge 인스턴스가 지원됩니다. Amazon DocumentDB는 이제 r5.8xlarge 및 r5.16xlarge 인스턴스를 지원합니다 블로그 게시물에서 자세히 알아보십시오. • 이제 글로벌 클러스터가 지원되어 리전 전체의 중단으로부터 재해 복구를 제공하고 가장 가까운 Amazon DocumentDB 클러스터에서 읽기를 허용하여 지연 시간이 짧은 글로벌 읽기를 사용할 수 있습니다. 글로벌 클러스터는 남아메리카(상파울루), 유럽(밀라노), 중국(베이징), 중국(닝샤) 리전에서 현재 지원하지 않습니다.

Date	릴리스 노트
2021년 5월 4일	새로운 기능 4.0(엔진 패치 버전 2.0.3371) • `renameCollection` • `$zip` • `$indexOfArray` • `$reverseArray` • `$natural` • `$hint`업데이트 지원 • `distinct`의 인덱스 스캔 버그 수정 및 기타 변경 사항 4.0(엔진 패치 버전 2.0.3371) • `$in` 쿼리의 메모리 사용량 감소 • 다중 키 인덱스의 메모리 누수가 수정되었습니다. • `$out`에 대한 설명 계획 및 프로파일러 출력을 수정했습니다. • 신뢰성을 개선하기 위해 내부 모니터링 시스템의 작업 제한 시간을 추가했습니다. • 다중 키 인덱스에 전달되는 쿼리 조건자에 영향을 미치는 결함을 수정했습니다.
2021년 1월 15일	새로운 기능 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.722) • 없음

Date	릴리스 노트
2020년 11월 9일	새로운 기능 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.722) • MongoDB 4.0 호환성 • ACID 트랜잭션 • cluster(client.watch() 또는 mongo.watch()) 및 데이터베이스 수준 (db.watch()) 변경 스트림 지원 • startAtOperationTime 을 사용하여 변경 스트림을 시작하거나 재개할 수 있습니다. • 변경 스트림 보존 기간을 7일로 연장(이전에는 24시간) • AWS DMS Amazon DocumentDB 4.0 대상 • CloudWatch 지표: TransactionsOpen , TransactionsOpenMax , TransactionsAborted , TransactionsStarted 및 TransactionsCommitted • currentOp , ServerStatus 및 profiler의 트랜잭션을 위한 새 필드. • $lookup 집계 단계에서 인덱스를 사용할 수 있는 능력 • find() 프로젝션이 있는 쿼리는 인덱스의 방향에 따라 제공될 수 있습니다(대상 쿼리) • findAndModify 와 함께 hint() 가용성 • $addToSet 운영자를 위한 성능 최적화 • 전체 인덱스 크기를 줄이기 위한 개선. • 새 집계 연산자: $ifNull, $replaceRoot , $setIsSubset , $setInstersection , $setUnion , 및 $setEquals • 이제 ListCollection 및 ListDatabase 명령을 사용하면 선택적으로 authorizedCollections 및 authorizedDatabases 매개변수를 사용하여 사용자가 각각 listCollections 및 listDatabase 역할을 요구하지 않고도 액세스 권한이 있는 컬렉션과 데이터베이스를 나열할 수 있습니다. • 또한 사용자는 KillCursor 역할 없이도 자신의 커서를 끝낼 수 있습니다.

Date	릴리스 노트
	• 이제 하위 문서의 숫자 유형을 비교하는 것이 첫 번째 수준 문서의 숫자 유형을 비교하는 것과 동일합니다. Amazon DocumentDB 4.0에서의 동작은 이제 MongoDB와 호환됩니다. 버그 수정 및 기타 변경 사항 Amazon DocumentDB 4.0(엔진 패치 버전 2.0.722) • `$setOnInsert` 위치 연산자를 사용할 때는 더 이상 업데이트를 허용하지 않습니다 $. Amazon DocumentDB 4.0에서의 동작은 이제 MongoDB와 호환됩니다. • `$createCollection` 의 문제를 해결하고 `autoIndexId` 를 설정했습니다. • 중첩된 문서에 대한 프로젝션 • 작업 메모리의 기본 설정을 인스턴스 메모리 크기에 맞게 규모를 조정하도록 변경했습니다. • 가비지 수집 개선 • 경로에 빈 키를 사용한 조회, mongo와의 동작 차이 • 시간대 동작의 `dateToString` 버그 수정 • 정렬 순서를 준수하도록 $push (집계) 수정 • 집계 관련 `$currentOp` 버그 수정 • 보조의 `readPreference` 에 발생하는 문제가 해결되었습니다. • `$createIndex` 이 명령이 실행된 것과 동일한 데이터베이스인지 확인하는 문제가 해결되었습니다. • `minKey`, `maxKey` 조회 실패에 대한 일관되지 않은 동작이 수정되었습니다. • `$size` 연산자가 복합 배열을 사용하지 않는 문제가 해결되었습니다. • regex `$in` 부정 관련 문제가 해결되었습니다. • 뷰에 대한 `$distinct` 명령 실행 문제가 해결되었습니다. • 누락된 필드를 다르게 정렬하는 집계 및 찾기 명령 관련 문제가 해결되었습니다. • 정규 표현식이 `$eq` 유형을 확인하지 않던 문제를 수정했습니다.

Date	릴리스 노트
	• 타임스탬프 서수 위치 동작의 `$currentDate` 버그 수정 • `$currentDate` 에 대한 밀리초 단위 수정

Amazon DocumentDB 3.6

Amazon DocumentDB 엔진 버전 3.6(MongoDB 3.6 호환성)에 대한 릴리스 정보입니다.

Date	릴리스 노트
2026년 8월 31일	새로운 기능 Amazon DocumentDB는 이제 엔진 버전 3.6에서 버전 8.0으로 직접 인플레이스 메이저 버전 업그레이드(MVU)를 지원합니다. 따라서 중간 버전 업그레이드가 필요하지 않으며 기존 데이터, 구성 및 클러스터 설정을 유지하면서 3.6 클러스터를 업그레이드할 수 있습니다. 이 업그레이드는 현재 이러한 버전이 지원되는 모든 AWS 리전에서 사용할 수 있습니다. 자세한 내용은 Amazon DocumentDB 인플레이스 주요 버전 업그레이드 섹션을 참조하세요.
2026년 8월 18일	보안 및 안정성 개선 Amazon DocumentDB 3.6(엔진 버전: 1.0.208702) [진행 중] 이 엔진 패치는 다음과 같은 개선 사항을 제공합니다. • 요청 처리 개선 Amazon DocumentDB는 이제 연결 및 메시지 구문 분석 경로를 따라 수신 클라이언트 요청을 더 철저하게 검증합니다. 이렇게 하면 광범위한 입력 패턴에서 복원력과 안정성이 향상됩니다. • 인증 핸드셰이크 강화 Amazon DocumentDB는 이제 인증 교환을 더 철저하게 검증합니다. 이렇게 하면 예기치 않거나 잘못된 형식의 입력을 처리할 때 견고성이 향상됩니다. 이 패치를 더 빨리 적용하지 않으면 Amazon DocumentDB는 릴리스 후 30일이 지난 다음 유지 관리 기간에이 패치를 자동으로 적용합니다.
2026년 4월 27일	새로운 기능

Date	릴리스 노트
	Amazon DocumentDB 3.6(엔진 버전: 1.0.208692) 이 업데이트는 다음과 같은 개선 사항이 포함된 선택적 엔진 패치를 제공합니다. 1. 변경 스트림에서 `startAtOperationTime` 에 대한 지원이 추가되었습니다.
2026년 2월 12일	버그 수정 및 기타 변경 사항 Amazon DocumentDB 3.6(엔진 버전: 1.0.208675) 1. 이제 `DatabaseConnectionsActiveMax` 지표가 인스턴스`Connections (active)` 를 추적합니다. 자세한 내용은 인스턴스 할당량을 참조하세요. 2. SASL 인증 성능이 개선되었습니다. 3. $indexStats 명령에 대한 버그 수정.
2024년 8월 20일	새로운 기능 Amazon DocumentDB 3.6(최소 엔진 패치 버전 1.0.208662)의 경우 TLS CA 인증서 업데이트 시 더 이상 시스템을 재부팅할 필요가 없습니다. 자세한 내용은 Amazon DocumentDB TLS 인증서 업데이트 섹션을 참조하세요.
2024년 7월 22일	새로운 기능 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.208662) Amazon DocumentDB MVU의 db.r5.* 및 db.r6.* 인스턴스에 대한 인덱스 제한을 제거했습니다. 자세한 내용은 사전 조건 섹션을 참조하세요. 버그 수정 및 기타 변경 사항 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.208662) Amazon DocumentDB는 이제 -NaN을 유효한 JSON 토큰으로 인식합니다.

Date	릴리스 노트
2023년 12월 20일	기타 변경사항 Amazon DocumentDB 3.6에서 인플레이스 메이저 버전 업그레이드에 대한 지원을 활성화했습니다. 자세한 내용은 Amazon DocumentDB 인플레이스 주요 버전 업그레이드 섹션을 참조하세요.
2023년 11월 17일	버그 수정 및 기타 변경 사항 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.208570) 사용자 정의 로컬 변수 이름은 이제 `$let` 및 `$filter`와 같은 프로젝션 연산자에 대해 "_"(밑줄)을 지원합니다.
2023년 9월 20일	새로운 기능 Amazon DocumentDB 3.6의 인플레이스 메이저 버전 업그레이드에 대한 지원이 추가되었습니다. 자세한 내용은 Amazon DocumentDB 인플레이스 주요 버전 업그레이드 섹션을 참조하세요.
2023년 2월 2일	버그 수정 및 기타 변경 사항 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.208432) • 워크로드가 진행 중인 동안 DML 감사가 활성화된 경우 발생하는 문제가 해결되었습니다. • 힌트가 포함된 집계 명령에 문자열 값이 전달될 때 발생하는 DML 감사 문제가 해결되었습니다. • readwriteanydatabase 역할을 사용하는 사용자가 authorizedCollections와 nameOnly 옵션을 모두 true로 설정한 경우 `listCollections` 명령이 작동하지 않는 문제가 해결되었습니다. • 필드 이름의 숫자 문자열을 제대로 구문 분석하기 위한 문제가 해결되었습니다. • 장시간 실행되는 커서가 가비지 수집에 영향을 줄 경우, 해당 커서를 취소하세요.

Date	릴리스 노트
2022년 8월 9일	새로운 기능 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.208152) • Decimal128 데이터 유형에 대한 지원이 추가되었습니다. Decimal128은 DocumentDB를 사용할 수 있는 모든 리전에서 지원되는 BSON 데이터 형식입니다. 자세한 정보는 데이터 형식을 참조하십시오. • Amazon CloudWatch Logs에 DML 쿼리 감사에 대한 지원이 추가되었습니다. 이제 Amazon DocumentDB는 데이터 조작 언어(DML) 이벤트 및 데이터 정의 언어(DDL) 이벤트를 Amazon CloudWatch Logs에 기록할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오. 버그 수정 및 기타 변경 사항 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.208152) • 이제 `changeOwnPassword` 권한이 있는 자체 비밀번호로 비밀번호를 변경할 수 있습니다.
2022년 4월 7일	새로운 기능 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.207836 및 1.0.208015) Amazon DocumentDB Performance Insights는 현재 미리 보기 단계에 있습니다. 이제 추가 비용 없이 7일간의 성능 기록을 롤링 윈도우에 저장할 수 있습니다. 자세한 내용은 성능 인사이트를 사용한 모니터링을 참조하세요.

Date	릴리스 노트

2022년 1월 24일	새로운 기능
	
	Amazon DocumentDB 3.6(엔진 패치 버전 1.0.207684)
	
	• Amazon DocumentDB는 이제 무료 평가판을 제공합니다. 자세한 내용은 Amazon DocumentDB 무료 평가판 페이지를 참조하십시오.
	• 이제 지리 공간 쿼리에서 다음 API를 포함한 향상된 기능을 사용할 수 있습니다.
	  • `$geoWithin`
	  • `$geoIntersects`
	• 다음 MongoDB 연산자에 대한 지원이 추가되었습니다.
	  • `$mergeObjects`
	  • `$reduce`
	
	자세한 내용은 Amazon DocumentDB에 지리 공간 데이터 쿼리를 참조하십시오.

2022년 1월 21일	새로운 기능
	
	Amazon DocumentDB 3.6(엔진 패치 버전 1.0.207781)
	
	• 다음 MongoDB API에 대한 지원이 추가되었습니다.
	  • `$reduce`
	  • `$mergeObjects`
	  • `$geoWithin`
	  • `$geoIntersects`

Date	릴리스 노트

2021년 10월 25일	새로운 기능
	
	Amazon DocumentDB 3.6(엔진 패치 버전 1.0.207780)
	
	• 다음 MongoDB API에 대한 지원이 추가되었습니다.
	  • `$literal`
	  • `$map`
	  • `$$ROOT`
	• 지리 공간 쿼리 기능 지원 자세한 내용은 이 블로그 게시물을 참조하십시오.
	• 사용자 정의 역할을 통한 액세스 제어 지원 자세한 내용은 이 블로그 게시물을 참조하십시오.
	• Amazon DocumentDB JDBC 드라이버를 사용하면 Tableau와 같은 BI 도구와 SQL Workbench와 같은 쿼리 도구에서 연결할 수 있습니다.
	
	버그 수정 및 기타 변경 사항
	
	Amazon DocumentDB 3.6(엔진 패치 버전 1.0.207780)
	
	• 다음과 같이 명시적인 `.sort()`와 `$natural`이 있을 때 올바르게 정렬되도록 `$natural` 버그 수정
	• `$redact`에 사용할 변경 스트림에 대한 버그 수정
	• 빈 배열과 함께 작동하기 위한 `$ifNull` 버그 수정
	• 현재 로그인한 사용자를 삭제하거나 진행 중인 활동에 대한 해당 사용자의 권한이 취소될 때 과도한 리소스 소비/서버 충돌이 발생하는 버그 수정
	• `listDatabase` 의 버그 수정 및 `listCollection` 권한 확인
	• 버그 수정 다중 키 요소에 대한 중복 제거 로직

Date	릴리스 노트
2021년 6월 24일	새로운 기능 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.207117) • 이제 r5.8xlarge 및 r5.16xlarge 인스턴스가 지원됩니다. Amazon DocumentDB는 이제 r5.8xlarge 및 r5.16xlarge 인스턴스를 지원합니다 블로그 게시물에서 자세히 알아보십시오. • 이제 글로벌 클러스터가 지원되어 리전 전체의 중단으로부터 재해 복구를 제공하고 가장 가까운 Amazon DocumentDB 클러스터에서 읽기를 허용하여 지연 시간이 짧은 글로벌 읽기를 사용할 수 있습니다. 글로벌 클러스터는 남아메리카(상파울루), 유럽(밀라노), 중국(베이징), 중국(닝샤) 리전에서 현재 지원하지 않습니다.

Date	릴리스 노트
2021년 5월 4일	새로운 기능 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.207117) • `renameCollection` • `$zip` • `$indexOfArray` • `$reverseArray` • `$natural` • `$hint`업데이트 지원 • `distinct`의 인덱스 스캔 버그 수정 및 기타 변경 사항 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.207117) • `$in` 쿼리의 메모리 사용량 감소 • 다중 키 인덱스의 메모리 누수가 수정되었습니다. • `$out`에 대한 설명 계획 및 프로파일러 출력을 수정했습니다. • 신뢰성을 개선하기 위해 내부 모니터링 시스템의 작업 제한 시간을 추가했습니다. • 다중 키 인덱스에 전달되는 쿼리 조건자에 영향을 미치는 결함을 수정했습니다.

Date	릴리스 노트
2021년 1월 15일	새로운 기능 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.206295) • `$lookup` 집계 단계에서 인덱스를 사용할 수 있는 능력 • `find()` 프로젝션이 있는 쿼리는 인덱스의 방향에 따라 제공될 수 있습니다(대상 쿼리) • `findAndModify` 와 함께 `hint()` 가용성 • `$addToSet` 운영자를 위한 성능 최적화 • 전체 인덱스 크기를 줄이기 위한 개선 • 새 집계 연산자: `$ifNull`, `$replaceRoot` , `$setIsSubset` , `$setInstersection` , `$setUnion` , 및 `$setEquals` • 또한 사용자는 `KillCursor` 역할 없이도 자신의 커서를 끝낼 수 있습니다.
2020년 11월 9일	새로운 기능 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.206295) • 없음 버그 수정 및 기타 변경 사항 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.206295) • 없음

Date	릴리스 노트
2020년 10월 30일	새로운 기능 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.206295) • 클러스터 수준 (`client.watch()` 또는 `mongo.watch()`) 및 데이터베이스 (`db.watch()`) 에서 변경 스트림 커서를 여는 기능이 추가되었습니다. • 변경 스트림 보존 기간을 7일로 연장(이전에는 24시간) 버그 수정 및 기타 변경 사항 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.206295) • 다양한 일반 케이스 성능 개선 • 표적 보안 개선 • 복합 인덱스의 두 번째 필드에서 정렬을 건너뛰는 문제가 해결되었습니다. • 다중 키 인덱스의 단일 필드에서 일반 인덱스가 같도록 활성화(복합 아님) • 인증 경쟁 조건이 수정되었습니다. • 가비지 수집 충돌이 간헐적으로 발생하던 문제가 해결되었습니다. • RBAC 보안 개선 • `databaseConnectionsMax` 메트릭 추가 • `r5.24xlarge` 인스턴스의 특정 워크로드에 대한 성능 개선

Date	릴리스 노트
2020년 9월 22일	새로운 기능 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.206295) • `$out` 집계 단계 • 인스턴스당 최대 연결 및 커서 수를 10배까지 늘렸습니다. 버그 수정 및 기타 변경 사항 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.206295) • 없음
2020년 7월 10일	새로운 기능 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.206295) • 리전 간 스냅샷 복제 버그 수정 및 기타 변경 사항 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.206295) • 없음

Date	릴리스 노트
2020년 6월 30일	새로운 기능 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.206295) • T3 medium 인스턴스 버그 수정 및 기타 변경 사항 Amazon DocumentDB 3.6(엔진 패치 버전 1.0.206295) • t3 인스턴스의 유휴 메모리 회수 • 인증 개선 • 향상된 SASL 인증 성능 • 가능한 최대 작업을 초과할 때 발생하는 `currentOp` 문제가 해결되었습니다. • 대량 업데이트 및 삭제 관련 `killOps` 문제가 해결되었습니다. • `$match`에서의 `$sample` 성능 개선 • 수정 단계의 $$ cond 사례의 지원이 수정되었습니다. • 여러 가지 반복되는 크래시 근본 원인을 수정했습니다. • I/O 및 지연 시간을 줄이기 위한 TTL 스위핑 개선 • `$unwind`에 대한 메모리 사용률 최적화 • 드롭 인덱스를 이용한 수집 통계 경쟁 조건을 수정했습니다. • 동시 인덱스 빌드 중 경쟁 상태가 수정되었습니다. • 인덱스의 `hash_search` 에서 가끔 충돌이 발생하는 문제를 수정했습니다.

일반 Amazon DocumentDB 릴리스 정보

리전 가용성, 도구 통합 및 교차 버전 기능을 포함한 일반 서비스 업데이트.

Date	릴리스 노트
2026년 9월 15일	새로운 기능 이제 Amazon DocumentDB 서버리스 인스턴스 유형을 다음 리전의 Amazon DocumentDB 5.0 및 8.0 엔진 버전에서 사용할 수 있습니다. • 아시아 태평양(오사카) • 아시아 태평양(자카르타) • 아시아 태평양(말레이시아) • 아시아 태평양(멜버른) • 아시아 태평양(태국) • 캐나다 서부(캘거리) • 유럽(취리히) • 유럽(스톡홀름) • 이스라엘(텔아비브) • 멕시코(중부) 자세한 내용은 리전별 지원되는 인스턴스 클래스 및 Amazon DocumentDB Serverless 사용 섹션을 참조하세요.
2026년 4월 29일	새로운 기능 이제 캐나다 서부(캘거리) 리전에서 Amazon DocumentDB를 사용할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오.
2026년 4월 17일	새로운 기능 이제 DocumentDB 서버리스에서 더 빠른 조정을 사용할 수 있습니다.
2026년 2월 11일	새로운 기능 이제 아시아 태평양(멜버른) 리전에서 Amazon DocumentDB를 사용할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오. 이제 유럽(취리히) 리전에서 Amazon DocumentDB를 사용할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오.

Date	릴리스 노트
2026년 1월 8일	새로운 기능 이제 아시아 태평양(자카르타) 리전에서 Amazon DocumentDB를 사용할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오.
2025년 10월 13일	새로운 기능 Amazon DocumentDB는 이제 아시아 태평양(하이데라바드) 리전에서 R6G 및 T4G 인스턴스를 지원합니다. 자세한 내용은 리전별 지원되는 인스턴스 클래스 섹션을 참조하세요.
2025년 10월 7일	새로운 기능 이제 다음 리전에서 Amazon DocumentDB를 사용할 수 있습니다. • 아시아 태평양(말레이시아) • 아시아 태평양(오사카) • 아시아 태평양(태국) • 멕시코(중부) 자세한 내용은 이 블로그 게시물을 참조하십시오.
2025년 9월 26일	새로운 기능 Amazon DocumentDB는 이제 교차 리전 및 교차 계정 스냅샷 복사 및 스냅샷 공유를 지원합니다. 자세한 내용은 에서 스냅샷 복사 AWS 리전 및 Amazon DocumentDB 클러스터 스냅샷 공유 섹션을 참조하세요.
2025년 6월 18일	새로운 기능 이제 이스라엘(텔아비브) 리전에서 Amazon DocumentDB를 사용할 수 있습니다.

Date	릴리스 노트
2025년 5월 8일	새로운 기능 이제 유럽(스톡홀름) 리전에서 Amazon DocumentDB를 사용할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오.
2025년 4월 2일	버그 수정 및 기타 변경 사항 백그라운드 인덱스가 있는 벡터 인덱스 생성의 버그를 수정했습니다.
2025년 2월 6일	새로운 기능 이제 Amazon DocumentDB가와 통합됩니다 AWS Toolkit for Visual Studio Code. 자세한 내용은 이 새로운 소식 블로그 게시물을 참조하고 AWS Toolkit for Visual Studio Code 사용 설명서의 Toolkit에서 Amazon DocumentDB 작업을 참조하세요.
2024년 10월 22일	새로운 기능 Amazon DocumentDB는 이제 스토리지 네트워크 처리량 지표 `StorageNetworkReceiveThroughput` , `StorageNetworkTransmitThroughput` 및 `StorageNetworkThroughput` 을 지원합니다. 자세한 내용은 CloudWatch 지표를 사용한 Amazon DocumentDB 인스턴스 사용량 평가 섹션을 참조하세요.
2024년 9월 18일	새로운 기능 이제 아프리카(케이프타운) 리전에서 Amazon DocumentDB를 사용할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오. 이제 유럽(스페인) 리전에서 Amazon DocumentDB를 사용할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오.
2024년 6월 25일	새로운 기능 AWS IAM ARNs 사용한 인증은 지원되는 모든 리전의 Amazon DocumentDB 인스턴스 기반 5.0 클러스터에서 사용할 수 있습니다. 자세한 내용은 IAM 자격 증명을 사용한 인증 섹션을 참조하세요.

Date	릴리스 노트
2024년 4월 3일	새로운 기능 이제 중동(UAE) 리전에서 Amazon DocumentDB를 사용할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오. 버그 수정 및 기타 변경 사항 • mongo 쉘 버전 1.7 이상에서 `db.coll.stats()` 를 호출할 때 발생하는 오류를 수정했습니다. • 동일한 집계 파이프라인의 일부로 `$regex`가 포함된 변경 스트림 쿼리에 대한 메모리 누출 문제를 수정했습니다.
2024년 1월 30일	새로운 기능 Amazon DocumentDB 글로벌 클러스터 이제 AWS GovCloud(미국 동부) 및 GovCloud(미국 AWS GovCloud 서부) AWS GovCloud (US) 리전 모두에서 글로벌 클러스터를 사용할 수 있습니다.
2023년 12월 13일	새로운 기능 원클릭 EC2 연결에 대한 지원이 추가되었습니다. 자세한 내용은 Amazon EC2를 사용하여 연결 섹션을 참조하세요.
2023년 11월 17일	새로운 기능 이제 AWS GovCloud(미국 동부) 리전에서 Amazon DocumentDB를 사용할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오.
2023년 9월 25일	새로운 기능 이제 아시아 태평양(홍콩) 리전에서 Amazon DocumentDB를 사용할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오.

Date	릴리스 노트
2023년 9월 11일	새로운 기능 이제 아시아 태평양(하이데라바드) 리전에서 Amazon DocumentDB를 사용할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오.
2022년 4월 29일	새로운 기능 이제 중국(베이징) 리전에서 Amazon DocumentDB를 사용할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오.
2022년 3월 16일	새로운 기능 이제 유럽(밀라노) 리전에서 Amazon DocumentDB를 사용할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오.
2022년 2월 8일	새로운 기능 Amazon DocumentDB R6G 및 T4G 인스턴스는 이제 아시아 태평양, 남미 및 유럽에서 사용할 수 있습니다. 자세한 내용은 이 블로그 게시물을 참조하십시오.
2021년 5월 4일	새로운 기능 이 블로그 게시물에서 모든 새로운 기능을 확인하십시오.
2020년 11월 9일	새로운 기능 이 블로그 게시물에서 모든 새로운 기능을 확인하십시오.
2020년 10월 30일	새로운 기능 이 블로그 게시물에서 모든 새로운 기능을 확인하십시오.
2020년 9월 22일	새로운 기능 이 블로그 게시물에서 모든 새로운 기능을 확인하십시오.
2020년 7월 10일	새로운 기능 이 블로그 게시물에서 모든 새로운 기능을 확인하십시오.

Date	릴리스 노트
2020년 6월 30일	새로운 기능 이 블로그 게시물에서 모든 새로운 기능을 확인하십시오.

Amazon DocumentDB 탄력적 클러스터 릴리스 정보

Amazon DocumentDB 탄력적 클러스터에 대한 릴리스 정보입니다.

Date	릴리스 노트
2025년 1월 28일	새로운 기능 원클릭 연결을 위해 이제 Amazon DocumentDB가 인스턴스 기반 및 탄력적 클러스터 AWS CloudShell 와 통합됩니다. 자세한 내용은 다음 중 일부 또는 전부를 참조하세요. • Amazon DocumentDB로 시작하기 • Amazon DocumentDB 엘라스틱 클러스터 시작하기 • 새로운 소식 블로그 게시물 • 기술적 방법 블로그 게시물
2024년 11월 12일	새로운 기능 Amazon DocumentDB 탄력적 클러스터는 이제 백그라운드 인덱스를 지원합니다.
2024년 11월 1일	새로운 기능 Amazon DocumentDB는 이제 탄력적 클러스터 유지 관리 작업을 지원합니다. 자세한 내용은 Amazon DocumentDB 탄력적 클러스터 유지 관리 섹션을 참조하세요.
2024년 8월 8일	새로운 기능

Date	릴리스 노트
	이제 아시아 태평양(홍콩), 캐나다(중부) 및 유럽(파리) 리전에서 Amazon DocumentDB 탄력적 클러스터를 사용할 수 있습니다. 자세한 내용은 탄력적 클러스터의 리전 가용성 섹션을 참조하세요.
2024년 7월 8일	새로운 기능 이제 유럽(밀라노) 리전에서 Amazon DocumentDB 탄력적 클러스터를 사용할 수 있습니다. 자세한 내용은 탄력적 클러스터의 리전 가용성 섹션을 참조하세요.
2024년 2월 22일	새로운 기능 Amazon DocumentDB 탄력적 클러스터 Amazon DocumentDB 탄력적 클러스터는 이제 다음 기능을 지원합니다. • 읽기 가능한 보조 샤드 인스턴스 복제본 - 자세한 내용은 1단계: 엘라스틱 클러스터 생성의 5b단계를 참조하세요. • 클러스터 시작/중지 - 자세한 내용은 Amazon DocumentDB 탄력적 클러스터 중지 및 시작 섹션을 참조하세요. • 구성 가능한 샤드 인스턴스 - 자세한 내용은 1단계: 엘라스틱 클러스터 생성의 5b단계를 참조하세요. • 스냅샷 자동 백업 - 자세한 내용은 탄력적 클러스터 스냅샷 자동 백업 관리 섹션을 참조하세요. • 스냅샷 복사 - 자세한 내용은 탄력적 클러스터 스냅샷 보기 섹션을 참조하세요.

Date	릴리스 노트
2024년 1월 30일	새로운 기능 Amazon DocumentDB 탄력적 클러스터 이제 Amazon DocumentDB 탄력적 클러스터를 다음 리전에서 사용할 수 있습니다. • 아시아 태평양(뭄바이) • 아시아 태평양(서울) • 남아메리카(상파울루) • 유럽(런던) 자세한 내용은 탄력적 클러스터 리전 및 버전 가용성 섹션을 참조하세요.

2023년 8월 3일	새로운 기능
	
	Amazon DocumentDB Elastic Clusters
	
	• Amazon DocumentDB Elastic 클러스터는 이제 다음 작업을 지원합니다.
	  • `top`
	  • `collStats`
	  • `hint`
	  • `dataSize`
	
	  지원되는 명령 및 작업의 전체 목록은 Amazon DocumentDB에서 지원되는 MongoDB API, 작업 및 데이터 형식을 참조하세요.
	• 이제 TTL(Time to Live) 인덱스가 지원됩니다.
	• 이제 인덱스 표현식에서도 인덱스 `hints`이 지원됩니다.

Date	릴리스 노트
2023년 3월 22일	새로운 기능 이제 아시아 태평양(싱가포르), 아시아 태평양(시드니) 및 아시아 태평양(도쿄) 리전에서 Amazon DocumentDB 탄력적 클러스터를 사용할 수 있습니다. 자세한 내용은 탄력적 클러스터 리전 및 버전 가용성 섹션을 참조하세요.
2022년 11월 30일	새로운 기능 Amazon DocumentDB Elastic Clusters Amazon DocumentDB Elastic 클러스터는 사용자가 MongoDB 샤딩 API를 활용하여 클러스터를 스케일 아웃할 수 있는 새로운 유형의 Amazon DocumentDB 클러스터입니다. 엘라스틱 클러스터는 데이터와 컴퓨팅을 여러 기본 컴퓨팅 인스턴스 및 볼륨에 분산하여 페타바이트의 스토리지 용량으로 거의 모든 수의 읽기 및 쓰기를 처리합니다. 자세한 내용은 Amazon DocumentDB 엘라스틱 클러스터 사용법을 참조하십시오.

Amazon DocumentDB 개발자 안내서용 문서 기록

- API 버전: 2014-10-31
- 최종 설명서 업데이트: 2026년 5월 27일

다음 표에서는 Amazon DocumentDB 개발자 안내서의 이번 릴리스를 소개합니다.

변경 사항	설명	날짜
새로운 Amazon DocumentDB 개발자 안내서	이제 Amazon DocumentDB 개발자 안내서가 새 URL에 게시됩니다. API 참조는 Amazon DocumentDB API 참조에서 별도로 사용할 수 있습니다.	2026년 5월 27일
AWS 관리형 정책 업데이트 - 정책 변경	Amazon DocumentDB는 엘라스틱 클러스터에 대한 전체 액세스 정책을 업데이트합니다.	2025년 2월 11일
AWS 관리형 정책 업데이트 - 정책 변경	Amazon DocumentDB는 엘라스틱 클러스터에 대한 전체 액세스 정책을 업데이트합니다.	2024년 2월 21일
AWS 관리형 정책 업데이트 - 정책 변경	Amazon DocumentDB는 엘라스틱 클러스터에 대한 읽기 전용 및 전체 액세스 정책을 업데이트합니다.	2023년 6월 21일
AWS 관리형 정책 업데이트 - 새 정책	Amazon DocumentDB는 엘라스틱 클러스터에 대한 새로운 읽기 전용 정책을 도입합니다.	2023년 6월 8일
AWS 관리형 정책 업데이트 - 새 정책	Amazon DocumentDB는 엘라스틱 클러스터에 대한 새로운 액세스 정책을 도입합니다.	2023년 6월 5일

MongoDB 5.0 호환성	Amazon DocumentDB은 이제 MongoDB의 5.0 버전과 호환됩니다.	2023년 3월 1일
정책 업데이트	Amazon DocumentDB 엘라스틱 클러스터 기능을 지원하기 위해, AmazonDocDBConsoleFullAccess 정책이 업데이트되고 AmazonDocDB-ElasticServiceRolePolicy 정책이 도입되었습니다.	2022년 11월 30일
탄력적 클러스터	Amazon DocumentDB의 분산 스토리지 시스템에서 데이터의 해시 기반 파티셔닝(샤딩)을 지원하는 새로운 엘라스틱 클러스터 기능이 추가되었습니다.	2022년 11월 30일
글로벌 클러스터	글로벌 클러스터 사용 방법에 대한 설명서가 추가되었습니다.	2021년 6월 2일
이벤트 구독	이벤트 구독 설명서가 추가되었습니다.	2021년 3월 26일
버전 3.6 업그레이드	역할 기반 액세스 제어, 집계 연산자 및 성능에서 버전 3.6에 대한 개선 사항을 문서화했습니다.	2021년 1월 15일
MongoDB 4.0 호환성	Amazon DocumentDB은 이제 MongoDB의 4.0 버전과 호환됩니다.	2020년 11월 9일

시작하기 가이드	, Amazon EC2, Robo3T 또는 Studio3T를 사용하여 Amazon DocumentDB를 시작하기 위한 새로운 시작 안내서입니다. AWS Cloud9 Amazon EC2 Robo3T Studio3T	2020년 8월 15일
추가 가용 영역 지원	Amazon DocumentDB에는 아시아 태평양(서울) (ap-northeast-2)의 새로운 가용 영역이 추가되었습니다.	2020년 7월 14일
리전 간의 스냅샷 복사 지원이 추가되었습니다.	Amazon DocumentDB에 AWS 리전간의 클러스터 스냅샷을 복사하기 위한 지원이 추가되었습니다. 자세한 내용은 리전 간 스냅샷 복사를 참조하십시오.	2020년 7월 10일
T3 인스턴스 클래스에 대한 지원이 추가되었습니다.	Amazon DocumentDB를 지원하는 모든 리전에 T3 인스턴스 유형에 대한 지원이 추가되었습니다. 자세한 내용은 리전별 지원되는 인스턴스 클래스 및 인스턴스 클래스 사양을 참조하십시오.	2020년 6월 30일
에 대한 지원이 추가되었습니다 AWS GovCloud (US).	이제 AWS GovCloud (US) 리전(us-gov-west-1)에서 Amazon DocumentDB를 사용할 수 있습니다.	2020년 6월 29일

16개의 새로운 CloudWatch 지표가 추가되었습니다.	Amazon DocumentDB에 16개의 새로운 Amazon CloudWatch 지표에 대한 지원이 추가되었습니다. 자세한 내용은 CloudWatch를 사용하는 Amazon DocumentDB 모니터링을 참조하십시오.	2020년 6월 23일
null 문자 및 $regex 연산자에 대한 지원이 추가되었습니다.	Amazon DocumentDB에 문자열에서 null 문자에 대한 지원과 $regex에 대한 인덱스 사용 기능이 추가되었습니다. Amazon DocumentDB에 대해 지원되는 MongoDB API 및 집계 파이프라인 기능을 보려면 MongoDB와의 기능적 차이를 참조하십시오.	2020년 6월 22일
향상된 다중 키 인덱싱 기능에 대한 지원이 추가되었습니다.	Amazon DocumentDB에 2,048바이트보다 큰 배열 인덱싱을 포함한 다중 키로 개선된 다중 키 인덱스를 생성하는 기능 지원과 동일한 배열의 다중 키가 있는 복합 다중 키 인덱싱 기능에 대한 지원이 추가되었습니다. 자세한 내용은 MongoDB와의 기능 차이점을 참조하십시오.	2020년 4월 23일
Amazon DocumentDB CloudFormation 스택의 삭제 방지에 대한 지원이 추가되었습니다.	Amazon DocumentDB 스택을 생성할 때 삭제 방지 활성화에 대한 지원이 Amazon DocumentDB CloudFormation에 추가되었습니다.	2020년 4월 20일

역할 기반 액세스 제어에 대한 지원이 추가되었습니다.	Amazon DocumentDB에 기본 제공 역할을 사용하여 역할 기반 액세스를 제어하는 데 대한 지원이 추가되었습니다.	2020년 3월 26일
캐나다(중부) (ca-central-1)의 추가 가용 영역에 대한 지원이 추가되었습니다.	Amazon DocumentDB를 이제 R5 클래스 인스턴스와 3개의 가용 영역이 있는 캐나다(중부) 리전(ca-central-1)에서 사용할 수 있습니다.	2020년 3월 26일
두 가지 추가 MongoDB API에 대한 지원이 추가되었습니다.	Amazon DocumentDB에 `$dateFromString` 및 `executionStats` MongoDB API에 대한 지원이 추가되었습니다.	2020년 3월 23일
다섯 가지 추가 MongoDB API에 대한 지원이 추가되었습니다.	Amazon DocumentDB에 `$objectToArray` , `$arrayToObject` , `$slice`, `$mod` 및 `$range` MongoDB API에 대한 지원이 추가되었습니다.	2020년 2월 6일
캐나다(중부)에 대한 지원이 추가되었습니다.	Amazon DocumentDB를 이제 R5 클래스 인스턴스가 있는 캐나다(중부) 리전(ca-central-1)에서 사용할 수 있습니다.	2019년 12월 11일
ChangeStreamLogSize에 대한 지원이 추가되었습니다.	Amazon DocumentDB에 CloudWatch 지표에 대한 `ChangeStreamLogSize` 지원이 추가되었습니다.	2019년 11월 22일
유럽(파리) 리전에 대한 지원 추가	이제 R5 클래스 인스턴스가 있는 유럽(파리) 리전(eu-west-3)에서 Amazon DocumentDB를 사용할 수 있습니다.	2019년 10월 30일

아시아 태평양(뭄바이) 리전에 대한 지원 추가	이제 R5 클래스 인스턴스가 있는 아시아 태평양(뭄바이) 리전(ap-south-1)에서 Amazon DocumentDB를 사용할 수 있습니다.	2019년 10월 17일
세 가지 추가 MongoDB API에 대한 지원이 추가되었습니다	Amazon DocumentDB에 `$addFields` , `$concatArrays` 및 `$lookup` MongoDB API에 대한 지원이 추가되었습니다.	2019년 10월 16일
아시아 태평양(싱가포르) 리전에 대한 지원 추가	이제 R5 클래스 인스턴스가 있는 아시아 태평양(싱가포르) 리전(ap-southeast-1)에서 Amazon DocumentDB를 사용할 수 있습니다.	2019년 10월 14일
TLS 인증서 업데이트를 위한 새 문서가 추가되었습니다	새 CA 인증서를 사용하여 TLS 연결을 생성하기 위한 CA 인증서 업데이트 지침이 추가되었습니다.	2019년 10월 2일
인증서에 대한 API 지원이 추가되었습니다	Amazon DocumentDB는 인스턴스에 대한 새 인증서 데이터 유형입니다. 자세한 내용은 DBInstance를 참조하십시오.	2019년 10월 1일
쿼리 프로파일링 지원	Amazon DocumentDB에 클러스터의 인스턴스와 데이터베이스에서 지원되는 작업을 프로파일링할 수 있는 기능이 추가되었습니다.	2019년 8월 19일
아시아 태평양(도쿄)에 세 번째 AZ가 추가되었습니다	Amazon DocumentDB에 아시아 태평양(도쿄)의 컴퓨팅 인스턴스를 위한 세 번째 가용 영역(AZ)이 추가되었습니다.	2019년 8월 9일

추가 Mongo API 지원	`$in`, `$isoWeek`, `$isoWeekYear` , `$isoDayOfWeek` 및 `$dateToString` 집계 연산자와 `$addToSet` 집계 단계를 포함하는 추가 집계 파이프라인 기능에 대한 지원이 추가되었습니다. 또한 Amazon DocumentDB에 컬렉션 수준 진단을 위한 `top()` 명령에 대한 지원과 `collMod()` 명령을 사용하여 TTL 인덱스의 `expireAfterSeconds` 파라미터를 수정할 수 있는 기능이 추가되었습니다.	2019년 7월 31일
유럽(런던)에 대한 지원이 추가되었습니다	Amazon DocumentDB를 이제 R5 클래스 인스턴스가 있는 유럽(런던) 리전(eu-west-2)에서 사용할 수 있습니다.	2019년 7월 18일
코드 샘플이 추가됨	프로그래밍 방식으로 Amazon DocumentDB에 연결하기 위한 R 및 Ruby의 코드 예제가 추가되었습니다.	2019년 7월 17일
모범 사례가 추가됨	Amazon DocumentDB 비용 관리에 도움이 되는 모범 사례가 추가되었습니다.	2019년 7월 17일
클러스터의 중지 및 시작 지원	Amazon DocumentDB에 클러스터를 중지하고 시작하여 개발 및 테스트 환경 비용을 관리할 수 있는 지원 기능이 추가되었습니다.	2019년 7월 1일

클러스터 삭제 방지 지원	실수로 인한 삭제로부터 클러스터를 보호하기 위해, Amazon DocumentDB에 삭제 방지가 추가되었습니다. 자세한 내용은 Amazon DocumentDB 클러스터 생성, Amazon DocumentDB 클러스터 수정, Amazon DocumentDB 클러스터 삭제 및 DBCluster의 API 주제의 `DeletionProtection` 를 참조하십시오.	2019년 7월 1일
기능적 차이 업데이트	기능적 차이에 암시적 트랜잭션이 추가되었습니다.	2019년 6월 26일
기능적 차이 추가	Amazon DocumentDB에 스토리지 및 인덱스 압축 관련 참고 사항이 추가되었습니다.	2019년 6월 13일
지원되는 추가 리전	Amazon DocumentDB를 이제 R5 클래스 인스턴스가 있는 아시아 태평양(시드니) (ap-southeast-2)에서 사용할 수 있습니다.	2019년 6월 5일
추가 리전에서 지원되는 R5 인스턴스 클래스	미국 동부(오하이오), 미국 동부(버지니아 북부), 미국 서부(오레곤), EU(아일랜드) 등 4개의 추가 리전에 대한 R5 인스턴스 클래스 지원이 추가되었습니다. 이 변경으로 R5 인스턴스는 Amazon DocumentDB를 지원하는 모든 리전에서 지원됩니다.	2019년 5월 17일

지원되는 추가 리전	R5 인스턴스 클래스를 사용하는 아시아 태평양(도쿄)(ap-northeast-1) 및 아시아 태평양(서울)(ap-northeast-2) 리전 2개에 대한 지원이 추가되었습니다. 자세한 내용은 리전별 지원되는 인스턴스 클래스 및 인스턴스 클래스 사양을 참조하십시오.	2019년 5월 8일
연결 코드 예제 추가	Java 및 C#의 Amazon DocumentDB 연결 코드 예제가 추가되었습니다.	2019년 4월 24일
추가 Mongo API 지원	7개 집계 문자열 연산자(`$indexOfBytes`, `$indexOfCP`, `$strLenBytes`, `$strLenCP`, `$toLower`, `$toUpper`, `$split`), 9개 날짜-시간 연산자(`$dayOfYear`, `$dayOfMonth`, `$dayOfWeek`, `$year`, `$month`, `$hour`, `$minute`, `$second`, `$millisecond`) 및 `$sample` 집계 파이프라인 단계에 대한 지원이 추가되었습니다.	2019년 4월 4일
연결 코드 예제 추가	Python, Node.js, PHP, Go의 Amazon DocumentDB 연결 코드 예제가 추가되었습니다.	2019년 3월 21일

프랑크푸르트 리전 및 R5 인스턴스에 대한 지원	R5 인스턴스 클래스를 사용하는 유럽(프랑크푸르트) 리전(eu-central-1)에 대한 지원이 추가되었습니다. 자세한 내용은 리전별 지원되는 인스턴스 클래스 및 인스턴스 클래스 사양을 참조하십시오.	2019년 3월 13일
집계 파이프라인 연산자 지원	새로운 집계 문자열 연산자($concat, $substr, $substrBytes , $substrCP , $strcasecmp), 배열 집계 연산자($size), 집계 그룹 누적기 연산자($push) 및 집계 단계($redact 및$indexStats)에 대한 지원이 추가되었습니다. 또한 위치 배열 연산자($[] 및 $[<identifier>]) 및 hint()에 대한 지원도 추가되었습니다.	2019년 2월 28일
엔진 업그레이드	대기 중인 클러스터 수정 사항 확인 및 클러스터 엔진 버전 업그레이드에 대한 설명서가 추가되었습니다.	2019년 2월 15일
이벤트 감사	Amazon CloudWatch Logs를 사용한 데이터베이스 이벤트 감사에 대한 지원이 추가되었습니다.	2019년 2월 12일
빠른 시작	를 사용하여 Amazon DocumentDB를 쉽게 시작하는 데 도움이 되는 퀵 스타트 주제가 추가되었습니다 CloudFormation.	2019년 1월 11일

공개 릴리스	Amazon DocumentDB(MongoDB 호환)의 첫 공개 릴리스입니다. 이 릴리스에는 개발자 안내서 및 통합 리소스 관리 API 참조가 포함됩니다.	2019년 1월 9일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.