aws

开发人员指南

Amazon DocumentDB



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

Amazon DocumentDB: 开发人员指南

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

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

Amazon DocumentDB 是什么 1
概述 1
集群 3
实例 3
区域和可用区 6
Regions 7
可用区 7
定价 9
免费试用 10
监控 10
接口 10
AWS 管理控制台 10
AWS CLI 10
MongoDB 驱动程序 11
接下来做什么？ 11
工作原理 11
亚马逊 DocumentDB 版本 12
Amazon DocumentDB 端点 14
TLS Support 16
Amazon DocumentDB 存储 17
Amazon DocumentDB 复制 18
Amazon DocumentDB 可靠性 18
读取首选项选项 19
TTL 删除 22
可计费资源 23
什么是文档数据库？ 26
使用案例 26
了解文档 27
使用文档 32
入门指南 44
先决条件 44
步骤 1：创建集群 45
步骤 2：Connect 连接到您的集群 48
步骤 3：插入和查询数据 49

第 4 步：探索 51
功能和配置 52
发动机状态 52
集群类型 52
功能 53
实例类型 54
AWS 集成 55
快速开始使用 CloudFormation 56
先决条件 56
所需的 IAM 权限 56
Amazon EC2 密钥对 57
启动亚马逊 DocumentDB CloudFormation 堆栈 57
访问 Amazon DocumentDB 集群 62
终止保护和删除保护 62
MongoDB 兼容性 63
兼容 MongoDB 8.0 63
亚马逊 DocumentDB 8.0 的新增功能 63
开始使用 Amazon DocumentDB 64
升级或迁移到亚马逊 DocumentDB 5.0 或 8.0 64
功能差异 65
MongoDB 5.0 兼容性 65
Amazon DocumentDB 5.0 中有什么新内容 65
亚马逊 DocumentDB 5.0.1 的新增功能 66
Amazon DocumentDB 5.0 入门 64
升级或迁移到亚马逊 DocumentDB 5.0 64
功能差异 65
与 MongoDB 4.0 的兼容性 69
Amazon DocumentDB 4.0 特征 69
Amazon DocumentDB 4.0 入门 70
升级或迁移到 Amazon DocumentDB 4.0 71
功能差异 71
事务 72
要求 72
最佳实践 72
限制 73
监控和诊断 73

事务隔离级别 74
使用案例 74
Multi-Statement 交易 75
Multi-collection 交易 76
回调 API 的事务 API 示例 78
核心 API 的事务 API 示例 78
支持的 命令 112
不支持的功能 112
会话 113
因果一致性 113
可重试写入 114
事务错误 114
最佳实践 116
基本操作指导 116
实例大小调整 117
使用索引 118
建立索引 118
索引选择性 118
索引对数据写入的影响 119
识别缺失的索引 119
识别未使用的索引 119
安全最佳实践 119
成本优化 120
使用指标确定性能问题 120
查看性能指标 121
设置 CloudWatch 闹铃 121
评估性能指标 121
使用指标评估亚马逊 DocumentDB 实例使用情况 CloudWatch 123
优化查询 124
TTL 和时间序列工作负载 124
迁移 125
使用集群参数组 125
聚合管道查询 125
######### 125
与 MongoDB 之间的功能差异 126
Amazon DocumentDB 的功能优势 126

隐式事务 126
更新的功能差异 127
数组索引 128
Multi-key 索引 128
字符串中的 Null 字符 129
Role-based 访问控制 129
$##表达式索引 129
嵌套文档投影 130
与 MongoDB 之间的功能差异 130
$####### 131
0pCountersCommand 131
管理数据库和集合 131
cursormaxTimems 131
explain() 132
索引构建 132
在路径中用空键查找 132
MongoDB API、操作和数据类型 132
mongodump # mongorest ore ## 133
结果排序 133
可重试写入 133
稀疏索引 134
$all ###### $elemMatch 134
字段名称中的美元符号（$）和句点（.） 135
$lookup 135
$###序和反向排序 138
支持的 MongoDB API、操作和数据类型 139
数据库命令 139
管理命令 140
聚合 142
身份验证 142
诊断命令 142
查询和写入操作 143
角色管理命令 144
会话命令 145
User management 146
分片命令 147

查询和投影运算符 148
数组运算符 149
按位运算符 149
注释运算符 149
比较运算符 150
元素运算符 150
评估查询运算符 150
逻辑运算符 151
投影运算符 151
更新运算符 151
数组运算符 152
按位运算符 152
字段运算符 152
更新修改器 153
地理空间 153
几何说明符 153
查询选择器 154
游标方法 155
聚合管道运算符 157
累加器表达式 158
算术运算符 159
数组运算符 160
布尔运算符 161
比较运算符 161
条件表达式运算符 162
数据类型运算符 162
数据大小运算符 162
日期运算符 163
文字运算符 164
合并运算符 164
自然运算符 164
集合运算符 165
阶段运算符 165
字符串运算符 167
系统变量 169
文本搜索运算符 169

类型转换运算符 169
变量运算符 170
其他运算符 170
数据类型 171
索引和索引属性 172
索引 172
索引属性 173
聚合管道运算符 173
$$DESCEND 178
$$KEEP 181
$$PRUNE 184
$ROOT 186
$abs 190
$add 193
$add ToSet 195
$addFields 197
$all ElementsTrue 200
$and 202
$any ElementTrue 205
$array ElemAt 207
$array ToObject 210
$avg 212
$bucket 214
$ceil 217
$changeStream 219
$cmp 222
$collStats 225
$concat 228
$concatArrays 230
$cond 233
$convert 236
$count 239
$currentOp 241
$dateAdd 243
$dateDiff 247
$date FromString 250

$dateSubtract 253
$date ToString 256
$dateTrunc 260
$day OfMonth 263
$day OfWeek 265
$day OfYear 267
$divide 270
$eq 272
$exp 274
$filter 277
$first 280
$floor 283
$geoNear 285
$group 288
$gt 290
$gte 293
$hour 295
$ifNull 299
$in 301
$index OfArray 303
$index OfBytes 306
$indexOfCP 308
$indexStats 310
$isArray 312
$iso DayOfWeek 315
$isoWeek 318
$iso WeekYear 320
$last 322
$let 326
$limit 329
$literal 332
$ln 335
$log 337
$log10 339
$lt 342
$lte 344

$lookup 347
$ltrim 350
$map 353
$match 355
$max 357
$meta 359
merge 362
$mergeObjects 365
$millisecond 367
$min 370
$minute 372
$mod 375
$month 377
$multiply 380
$natural 383
$ne 385
$not 388
$object ToArray 390
$or 394
$out 397
$pow 400
$push 402
$project 404
$rand 406
$range 409
$redact 412
$reduce 414
$regexFind 417
$正则表达式 FindAll 420
$regexMatch 423
$replaceAll 426
$replaceOne 430
$replaceRoot 433
$replaceWith 436
$reverseArray 439
$rtrim 441

$sample 444
$search 447
$second 449
$set 451
$setDifference 453
$setEquals 456
$setIntersection 459
$set IsSubset 461
$setUnion 463
$skip 466
$slice 468
$size 470
$sort 473
$split 475
$sqrt 476
$str LenBytes 479
$strLenCP 482
$strcasecmp 484
$substr 487
$substrBytes 489
$substrCP 491
$subtract 494
$sum 497
$switch 500
$toBool 503
$toDate 505
$toDecimal 508
$toDouble 510
$toInt 513
$toLong 515
$toLower 517
$to ObjectId 519
$toString 521
$toUpper 523
$trim 525
$type 527

$unset ... 530
$unwind ... 532
$向量搜索 ... 534
$week ... 538
$year ... 540
$zip ... 542
地理空间 ... 545
$geometry ... 546
$geoIntersects ... 550
$geoWithin ... 554
$maxDistance ... 557
$minDistance ... 560
$near ... 563
$nearSphere ... 567
查询和投影运算符 ... 569
$ ... 571
$all ... 573
$and ... 575
$bits AllClear ... 577
$bits AllSet ... 579
$bits AnyClear ... 581
$bits AnySet ... 583
$comment ... 585
$elemMatch ... 587
$eq ... 590
$exists ... 591
$expr ... 593
$gt ... 596
$gte ... 597
$in ... 599
$jsonSchema ... 601
$lt ... 604
$mod ... 605
$lte ... 607
$meta ... 609
$ne ... 611

$nin ... 613
$nor ... 616
$not ... 618
$or ... 620
$regex ... 623
$slice ... 625
$size ... 627
$text ... 628
$type ... 631
更新运算符 ... 633
$ ... 634
$[] ... 636
$ [<标识符>] ... 638
$add ToSet ... 640
$bit ... 642
$currentDate ... 644
$each ... 647
$inc ... 649
$max ... 651
$min ... 653
$mul ... 655
$pop ... 657
$position ... 659
$pull ... 662
$pullAll ... 665
$push ... 667
$rename ... 669
$set ... 672
$set OnInsert ... 674
$slice ... 677
$sort ... 680
$unset ... 683
生成式人工智能 ... 686
SageMaker 帆布 ... 686
如何使用 SageMaker AI Canvas 构建无代码机器学习模型 ... 686
配置 A SageMaker I 域和用户配置文件 ... 687

为亚马逊 DocumentDB 和 SageMaker AI Canvas 配置 IAM 访问权限 687
为 SageMaker AI Canvas 创建数据库用户和角色 687
可用区 688
向量搜索 688
插入向量 689
创建向量索引 689
获取索引定义 693
查询向量 694
特征和限制 697
最佳实践 698
迁移到 Amazon DocumentDB 700
快速入门指南 700
准备 DMS 源 700
设置 DMS 701
启用 DocumentDB 压缩 701
创建复制任务 701
监控进度 701
附加信息 702
迁移运行手册 702
兼容性 703
工作负载发现 705
索引迁移 708
用户迁移 709
数据迁移 711
监控 722
验证 724
从 Couchbase 服务器迁移 729
简介 729
与亚马逊 DocumentDB 的比较 729
Discovery 731
规划 736
迁移 739
验证 747
升级 DocumentDB 751
引擎版本支持日期 751
亚马逊 DocumentDB 主要版本的发布日历 751

亚马逊 DocumentDB 次要版本的发布日历 752
升级主引擎版本 752
支持的升级途径 753
先决条件 753
最佳实践 754
从 Amazon DocumentDB 5.0 升级到 8.0 后会发生什么变化 757
Post-upgrade 从 3.6 或 4.0 升级的集群的注意事项 758
执行升级 758
问题排查 761
升级次要引擎版本 761
在执行次要版本升级之前 761
执行次要版本升级 762
验证次要版本升级 762
使用升级 AWS DMS 763
步骤 1：启用变更流 763
步骤 2：修改变更流保留期限 764
步骤 3：迁移您的索引 764
步骤 4：创建一个 AWS DMS 复制实例 765
步骤 5：创建一个 AWS DMS 源端点 767
步骤 6：创建一个 AWS DMS 目标端点 769
步骤 7：创建并运行迁移任务 771
步骤 8：将应用程序端点更改为目标 Amazon DocumentDB 集群 772
使用长期支持（LTS）版本 772
当前的 LTS 版本 772
谁应该使用 LTS 773
如何创建新的 LTS 集群 773
如何从引擎版本 3.6.0 或 4.0.0 升级到 5.0.0 的 LTS 集群 774
如何检查我现有的引擎版本 5.0.0 集群是否在 LTS 上 774
扩展支持 775
扩展支持概述 776
扩展支持费用 776
责任 776
安全性 778
密码管理 AWS Secrets Manager 779
Secrets Manager 与 Amazon DocumentDB 集成的限制 779
使用管理主用户密码的概述 AWS Secrets Manager 779

强制执行 Amazon DocumentDB 对主用户密码的管理 AWS Secrets Manager 780
使用 Secrets Manager 管理集群的主用户密码 781
数据保护 781
Client-side 字段级加密 782
加密静态数据 789
加密传输中数据 795
密钥管理 806
身份和访问管理 806
受众 807
使用身份进行身份验证 807
使用策略管理访问 808
Amazon DocumentDB 如何配合 IAM 工作 809
Identity-based 策略示例 815
问题排查 818
管理对 Amazon DocumentDB 资源的访问权限 819
使用基于身份的策略（IAM 策略） 824
AWS 亚马逊 DocumentDB 的托管政策 828
Amazon DocumentDB API 权限参考 846
Cross-service 混乱的副手预防 854
使用 IAM 身份进行身份验证 855
开始使用 IAM 用户和角色 855
配置计算类型 858
监控身份验证请求 859
使用 IAM 身份验证 860
支持 IAM 的驱动程序 860
IAM 身份验证的常见问题解答 860
管理 Amazon DocumentDB 用户 861
主用户和####### 861
创建其他用户 861
自动轮换密码 863
Role-Based 访问控制 864
RBAC 概念 865
RBAC 内置角色入门 866
RBAC 用户定义角色入门 870
以用户身份连接到 Amazon DocumentDB 874
通用命令 876

功能差异 ... 881
限制 ... 881
使用访问控制 Role-Based 访问数据库 ... 881
日志记录和监控 ... 890
更新证书 ... 890
更新您的应用程序和 Amazon DocumentDB 集群 ... 891
自动服务器证书轮换 ... 895
问题排查 ... 896
常见问题 ... 896
合规性验证 ... 901
恢复能力 ... 902
基础结构安全性 ... 903
VPC 端点（AWS PrivateLink） ... 904
VPC 端点注意事项 ... 905
区域可用性 ... 905
为 Amazon DocumentDB API 创建接口 VPC 端点 ... 906
为 Amazon DocumentDB API 创建 VPC 端点策略 ... 906
安全最佳实践 ... 908
审核事件 ... 908
支持的事件 ... 909
启用审核 ... 913
禁用审核 ... 919
访问您的事件 ... 921
筛选 DML 事件 ... 922
Amazon VPC 和 Amazon DocumentDB ... 927
VPC 中的 DocumentDB 集群 ... 927
访问 VPC 中的集群 ... 938
为集群创建 IPv4-only VPC ... 940
为集群创建双堆栈 VPC ... 946
备份和还原 ... 955
备份和还原：概念 ... 956
了解 备份存储使用量 ... 957
转储、还原、导入和导出数据 ... 958
mongodump ... 958
mongorestor ... 959
... 960

mongoimpor ... 960
教程 ... 961
集群快照注意事项 ... 963
备份存储 ... 964
备份时段 ... 964
备份保留期 ... 966
复制集群快照加密 ... 966
比较自动快照和手动快照 ... 967
创建手动集群快照 ... 968
复制集群快照 ... 971
复制共享快照 ... 971
跨复制快照 AWS 区域 ... 972
限制 ... 972
处理加密 ... 972
参数组注意事项 ... 972
复制集群快照 ... 973
共享集群快照 ... 982
共享加密的快照 ... 982
共享快照 ... 985
从集群快照还原 ... 989
还原到某个时间点 ... 995
删除集群快照 ... 1001
管理 Amazon DocumentDB ... 1003
操作任务概述 ... 1003
向 Amazon DocumentDB 集群添加副本 ... 1004
描述集群和实例 ... 1005
创建集群快照 ... 1007
从快照还原 ... 1008
从集群中删除实例 ... 1009
删除集群 ... 1009
全局集群 ... 1010
什么是全局集群？ ... 1010
全局集群有何用处？ ... 1010
目前全局集群的局限性有哪些？ ... 1010
快速入门指南 ... 1011
管理全局集群 ... 1023

连接全局集群 1029
监控全局集群指标 1030
灾难恢复 1030
管理 集群 1043
了解集群 1044
集群设置 1046
集群存储配置 1048
确定集群的状态 1051
集群生命周期 1052
扩展集群 1089
克隆集群卷 1091
了解集群容错能力 1104
管理实例 1105
确定实例的状态 1105
实例生命周期 1105
管理实例类 1127
NVMe-backed 实例 1139
管理子网组 1141
创建子网组 1142
描述子网组 1147
修改子网组 1149
删除子网组 1152
高可用性和复制 1154
读取扩展 1154
高可用性 1154
添加 副本 1155
失效转移 1156
复制滞后 1160
管理索引 1161
Amazon DocumentDB 索引创建 1161
维护 Amazon DocumentDB 索引 1166
管理文档压缩 1168
管理文档压缩 1169
监控文档压缩 1170
管理基于字典的压缩 1171
性能注意事项 1171

启用基于字典的压缩 1172
开始使用 1173
监控 1173
管理事件 1174
查看事件类别 1174
查看 Amazon DocumentDB 事件 1177
选择区域和可用区 1179
区域可用性 1180
管理集群参数组 1183
描述集群参数组 1183
创建集群参数组 1190
修改集群参数组 1192
修改集群以使用自定义的集群参数组 1197
复制集群参数组 1198
重置集群参数组 1200
删除集群参数组 1202
集群参数参考 1205
了解端点 1222
查找集群的端点 1223
查找实例的端点 1225
连接到端点 1228
了解 Amazon DocumentDB ARN 1230
构建 ARN 1230
查找 ARN 1234
标注资源 1236
资源标签概述 1236
标签约束 1237
添加或更新标签 1237
列出标签 1239
移除标签 1240
维护 Amazon DocumentDB 1242
版本编号 1243
管理维护窗口 1244
引擎补丁通知 1246
查看待处理维护 1247
引擎更新 1250

次要版本 1253
操作系统更新 1253
User-initiated 更新 1256
全局集群 1257
了解服务相关角色 1258
Service-linked 角色权限 1258
创建服务相关角色 1260
修改服务相关角色 1260
删除服务相关角色 1260
Amazon DocumentDB 服务相关角色支持的区域 1261
使用变更流 1261
支持的操作 1262
计费 1262
限制 1263
启用变更流 1263
在 Python 中使用变更流 1265
完整文档查找 1268
恢复变更流 1268
使用 startAtOperationTime 恢复 1270
使用 postBatchResumeToken 恢复 1272
事务 1273
修改日志保留期 1273
辅助实例上的变更流 1276
使用排序规则 1277
限制 1278
使用视图 1278
最佳实践 1279
聚合器运算符兼容性 1279
利用索引和视图 1279
使用 AWS Lambda 有变更流 1280
限制 1280
使用弹性集群 1281
弹性集群用例 1282
用户资料 1282
内容管理和历史记录 1282
弹性集群的优势 1282

AWS 服务集成 1282
区域和版本可用性 1283
弹性集群的区域可用性 1283
版本可用性 1284
限制 1284
弹性集群管理 1284
查询和写入操作 1284
集合和索引管理 1285
管理和诊断 1285
Opt-in features 1285
工作原理 1286
Amazon DocumentDB 弹性集群分片 1286
弹性集群迁移 1290
弹性集群扩展 1290
弹性集群可靠性 1290
弹性集群存储和可用性 1290
Amazon DocumentDB 4.0 与弹性集群之间的功能差异 1290
开始使用 1292
先决条件 1292
第 1 步：创建弹性集群 1294
步骤 2：连接到您的弹性集群 1299
步骤 3：对您的集合分片，插入和查询数据 1300
第 4 步：探索 1303
最佳实践 1304
选择分片键 1304
连接管理 1304
未分片的集合 1305
扩展弹性集群 1305
监控弹性集群 1305
管理弹性集群 1306
修改弹性集群配置 1306
监控弹性集群 1310
删除弹性集群 1313
管理弹性集群快照 1315
停止和启动弹性集群 1328
维护弹性集群 1332

静态数据加密 1339
Amazon DocumentDB 弹性集群如何在中使用授权 AWS KMS 1340
创建客户托管密钥 1341
监控您的 Amazon DocumentDB 弹性集群加密密钥 1342
了解详情 1347
Service-linked 角色 1348
Service-linked 弹性集群的角色权限 1348
监控 Amazon DocumentDB 1352
监控集群的状态 1353
集群状态值 1353
监控集群的状态 1354
监控实例的状态 1356
实例状态值 1357
使用监控实例状态 AWS 管理控制台 或者 AWS CLI 1358
实例运行状况值 1360
使用监控实例的运行状况 AWS 管理控制台 1360
查看 Amazon DocumentDB 推荐 1362
事件订阅 1364
订阅活动 1365
管理订阅 1367
类别和消息 1370
使用以下方式监控亚马逊 DocumentDB CloudWatch 1373
Amazon DocumentDB 指标 1373
查看 CloudWatch 数据 1384
Amazon DocumentDB 维度 1390
监控 Opcounter 指标 1390
监控数据库连接 1391
使用记录亚马逊 DocumentDB API 调用 CloudTrail 1391
中的亚马逊 DocumentDB 信息 CloudTrail 1391
分析操作 1392
支持的操作 1393
限制 1393
启用分析器 1394
禁用分析器 1398
禁用分析器日志导出 1398
访问分析器日志 1401

常见查询 1401
使用 Performance Insights 进行监控 1402
Performance Insights 概念 1403
启用和禁用 Performance Insights 1405
为 Performance Insights 配置访问策略 1408
使用 Performance Insights 控制面板分析指标 1412
使用 Performance Insights API 检索指标 1430
Performance Insights 的亚马逊 CloudWatch 指标 1444
Performance Insights 的计数器指标 1446
OpenSearch 整合 1448
以亚马逊 OpenSearch 服务为目的地 1448
步骤 1：创建 Amazon OpenSearch 服务域名或 OpenSearch 无服务器集合 1448
步骤 2：在 Amazon DocumentDB 集群上启用变更流 1449
步骤 3：设置管道角色，使其拥有写入 Amazon S3 存储桶和目标域或集合的权限 1449
步骤 4：添加要创建的管道角色所需的权限 X-ENI 1450
步骤 5：创建管道 1451
限制 1451
使用 DocumentDB 无服务器 1452
无服务器应用场景 1452
使用 Amazon DocumentDB 无服务器处理现有的预置工作负载 1453
Amazon DocumentDB 无服务器的优势 1454
无服务器的工作原理 1455
概述 1455
Amazon DocumentDB 集群的配置 1456
Amazon DocumentDB 无服务器扩缩容量 1456
Amazon DocumentDB 无服务器扩缩 1457
空闲状态（0.5 个 DCU） 1459
无服务器要求和限制 1459
功能要求 1459
创建使用无服务器的集群 1461
创建 Amazon DocumentDB 无服务器集群 1461
添加 Amazon DocumentDB 无服务器实例 1464
迁移到无服务器 1464
将现有的 DocumentDB 集群迁移到无服务器 1465
从 MongoDB 迁移到 DocumentDB 无服务器 1469
管理无服务器 1469

查看和修改集群的扩缩容量范围配置 1469
查看和修改无服务器读取器的提升层 1472
无服务器实例限制 1474
无服务器扩缩配置 1476
为 DocumentDB 无服务器集群选择扩缩容量范围 1477
为 DocumentDB 无服务器集群选择 MinCapacity 设置 1477
为 DocumentDB 无服务器集群选择 MaxCapacity 设置 1478
避免内存不足错误 1480
为什么我的无服务器实例无法缩减？ 1481
监控无服务器 1481
内存不足：incompatible-parameters 状态 1481
DocumentDB 无服务器的亚马逊 CloudWatch 指标 1482
使用性能详情监控 DocumentDB 无服务器性能 1485
使用 Amazon DocumentDB 开发 1487
连接到 DocumentDB 1487
以编程方式连接 1487
从 Amazon VPC 外部连接 1514
作为副本集进行连接 1515
使用 Studio 3T 连接 1519
使用 Connect DataGrip 1529
使用 Amazon EC2 进行连接 1534
使用 JDBC 驱动程序进行连接 1566
使用 ODBC 驱动程序进行连接 1581
使用 DocumentDB 进行编程 1595
DocumentDB Java 编程指南 1595
使用 JSON 架构验证 1623
限额和限制 1633
支持的实例类型 1633
支持的区域 1635
区域配额 1637
聚合限制 1639
集群限制 1639
实例限制 1640
命名约束 1646
TTL 约束 1647
弹性集群限制 1647

弹性集群分片限制 1648
每个分片的弹性集群 CPU、内存、连接和光标限制 1649
查询 1650
查询文档 1650
检索所有文档 1651
匹配字段值 1651
嵌入文档 1651
嵌入文档中的字段值 1651
匹配数组 1652
匹配数组中的值 1652
使用运算符 1652
查询计划 1653
查询计划 1653
查询计划缓存 1654
解释结果 1655
扫描和过滤阶段 1656
索引交集 1657
索引并集 1657
多索引 intersection/union 1658
复合索引 1659
排序阶段 1659
小组阶段 1659
查询计划器 v2 1660
先决条件 1660
选择计划程序版本 2.0 作为默认查询计划程序 1661
最佳实践 1661
限制 1661
对##和##运算符的改进 1661
计划缓存筛选条件 API 1664
计划程序版本 1.0、2.0 和 MongoDB 之间的潜在行为差异 1666
计划程序版本 2.0 弥合了与 MongoDB 的行为差距 1673
查询计划器 v3 1675
先决条件 1676
选择计划器版本 3.0 作为默认查询计划器 1676
最佳实践 1676
限制 1676

对##和##运算符的改进 1677
计划器版本 1.0、3.0 和 MongoDB 之间的潜在行为差异 1678
地理空间数据 1679
概述 1
索引化和存储地理空间数据 1680
查询地理空间数据 1681
限制 1685
部分索引 1685
创建部分索引 1685
支持的运算符 1686
使用部分索引进行查询 1686
部分索引功能 1687
部分索引限制 1691
文本搜索 1692
支持的功能 1692
使用 Amazon DocumentDB 文本索引 1693
与 MongoDB 的差异 1698
最佳实践和准则 1698
文本索引 V2 1699
限制 1699
索引 1700
索引管理 1700
索引创建 1700
维护索引 1706
使用索引的最佳实践 1708
索引类型 1709
单字段索引 1709
复合索引 1711
Multi-key 索引 1712
文本索引 1714
地理空间索引 1715
向量索引 1716
索引属性 1717
主题 1717
name 1717
unique 1721

稀疏 1723
局部的 FilterExpression 1727
到期 AfterSeconds 1731
性能优化 1734
常见性能问题 1734
高 CPU 使用率 1734
连接问题 1736
查询运行缓慢 1742
Anti-patterns 1746
复合索引反模式超过 3 个属性 1746
长时间运行的查询 1748
低使用率和冗余索引 1750
Multi-key 包含大型数组的索引 1753
性能改进技巧 1754
1. 使用 $match 作为聚合管道的第一阶段 1754
2. 在聚合管道中使用 $project 来最小化管道数据大小 1754
3. 启用文档压缩以降低存储 I/O 成本、成本并提高查询性能 1754
4. 利用索引实现最佳查询性能 1755
5. 根据查询模式优化数据模型 1756
查询计划分析 1758
总体时机 1760
执行阶段 1760
在 “解释计划执行统计信息” 中了解已检查的文档 1762
问题排查 1770
连接问题 1770
无法连接到 Amazon DocumentDB 端点 1770
测试与 Amazon DocumentDB 实例的连接 1775
连接到无效终端节点 1776
影响连接数的驱动程序配置 1777
索引 1777
索引构建失败 1777
后台索引构建延迟问题和失败 1777
索引膨胀 1778
性能和资源利用率 1779
查看插入、更新和删除统计信息 1779
分析缓存性能 1781

查找并终止长时间运行或受阻的查询 1782
查看查询计划和优化查询 1783
如何在弹性集群中查看查询计划？ 1785
列出实例上正在运行的所有操作 1788
知道查询何时取得进展 1790
确定系统突然运行缓慢的原因 1793
确定 CPU 使用率过高的原因 1794
在实例上找到打开的光标 1795
查看当前的 Amazon DocumentDB 引擎版本 1795
分析索引使用情况并识别未使用的索引 1796
识别缺失的索引 1798
如何确定数据库集合膨胀？ 1799
有用查询的摘要 1800
垃圾回收 1801
了解 Amazon DocumentDB 中的垃圾收集 1802
垃圾回收过程 1802
存储架构和扩展存储 1803
监控垃圾回收 1804
collStats 输出示例 1806
常见问题 1808
发行说明 1810
亚马逊 DocumentDB 引擎版本发行说明 1810
亚马逊 DocumentDB 通用发行说明 1855
亚马逊 DocumentDB 弹性集群发行说明 1860
文档历史记录 1864
..... mdccclxxiii

Amazon DocumentDB（兼容 MongoDB）是什么

Amazon DocumentDB（与 MongoDB 兼容）是一种快速、可靠、完全托管的数据库服务。Amazon DocumentDB 可以轻松地在云中设置、操作和扩展 MongoDB-compatible 数据库。借助 Amazon DocumentDB，您可以运行与 MongoDB 相同的应用程序代码，并使用与 MongoDB 相同的驱动程序和工具。

在使用 Amazon DocumentDB 之前，您应查看 工作原理 中所述的概念和功能。之后，完成 入门指南 中的步骤。

主题

- Amazon DocumentDB 概述
- 集群
- 实例
- 区域和可用区
- Amazon DocumentDB 定价
- 监控
- 接口
- 接下来做什么？
- Amazon DocumentDB：工作方式
- 什么是文档数据库？

Amazon DocumentDB 概述

以下是 Amazon DocumentDB 的一些高级功能：

- Amazon DocumentDB 支持两个类型的集群：基于实例的集群和弹性集群。弹性集群支持 reads/writes 每秒数百万和数 PB 存储容量的工作负载。有关弹性集群的更多信息，请参阅 Amazon DocumentDB 弹性集群。以下内容涉及 Amazon DocumentDB 基于实例的集群。
- Amazon DocumentDB 会随着您的数据库存储需求的增长而自动提高您的存储卷大小。您的存储卷以 10 GB 为增量增加，最大可扩展到 128 TiB。您无需为了满足未来增长需求而为集群预置任何多余存储空间。
- 借助 Amazon DocumentDB，您可以通过创建多达 15 个副本实例来增加读取吞吐量以支持高容量应用程序请求。Amazon DocumentDB 副本共享相同的底层存储，从而降低成本并且避免在副本节点

上执行写入的需要。此功能将释放更多的处理能力来提供读取请求并降低副本延迟时间 通常可降低到几毫秒。无论存储卷大小如何，您都可以在几分钟内添加副本。Amazon DocumentDB 还提供读取器端点，从而无需在添加和删除副本时跟踪副本应用程序即可连接。

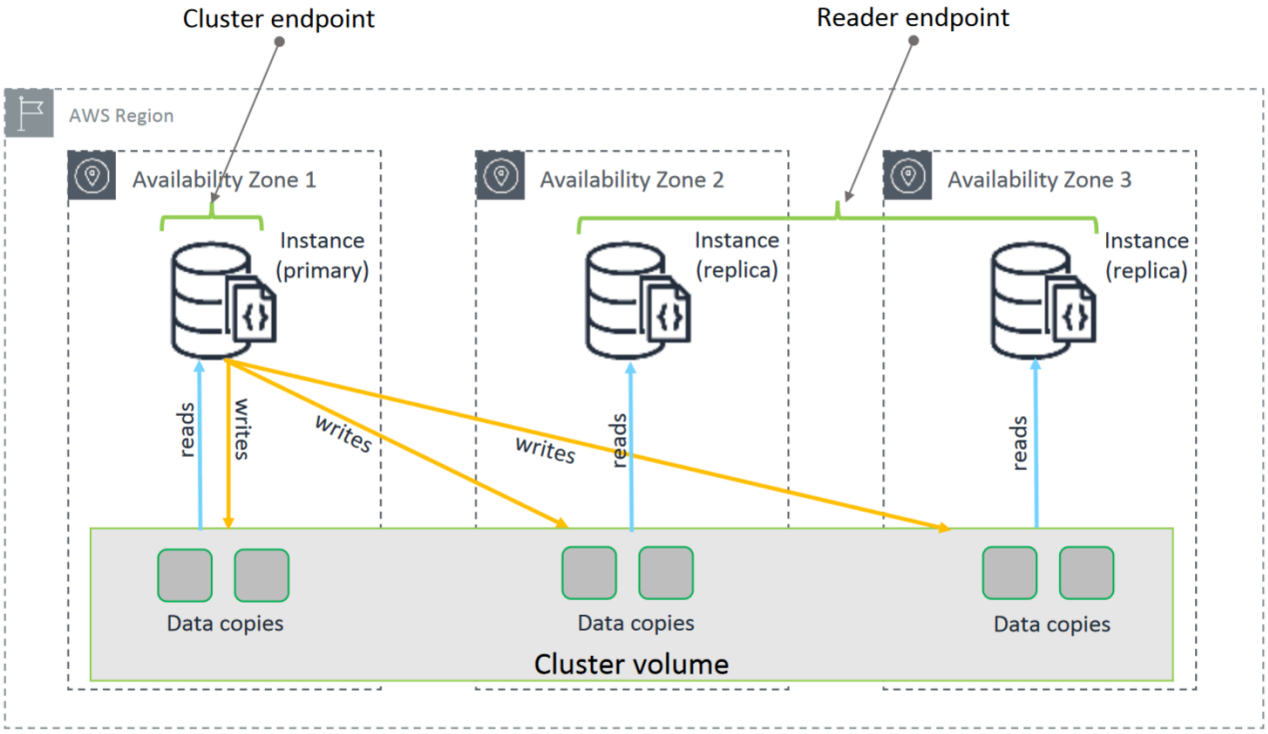
- Amazon DocumentDB 允许您为每个实例增加或减少计算和内存资源。计算扩展操作通常可在几分钟之内完成。
- Amazon DocumentDB 在 Amazon Virtual Private Cloud（Amazon VPC）中运行，因此您可以在自己的虚拟网络中隔离您的数据库。您还可以配置防火墙设置以控制网络访问集群。
- Amazon DocumentDB 持续监控集群的运行状况。发生实例故障时，Amazon DocumentDB 自动重启该实例和相关进程。Amazon DocumentDB 不需要数据库重做日志的崩溃恢复重播，这大大缩短重启时间。Amazon DocumentDB 还将数据库缓存与数据库进程隔离，使缓存能够在实例重启后继续运行。
- 发生实例故障时，Amazon DocumentDB 自动将故障转移到最多 15 个 Amazon DocumentDB 副本中的一个，而这些副本是您在其他可用区中创建的。如果未预配置副本，当发生故障时，Amazon DocumentDB 尝试为您自动创建一个新的 Amazon DocumentDB 实例。
- 借助 Amazon DocumentDB 的备份功能，您可以对集群进行时间点恢复。此功能允许您将集群恢复到保留期内（最多是近 5 分钟内）任何一秒钟的状态。可将自动备份保留期配置为最长 35 天。自动备份存储在 Amazon Simple Storage Service (Amazon S3) 中，该服务专为 99.999999999% 持久性设计。Amazon DocumentDB 备份是自动、增量和连续的，不影响您的集群性能。
- 借助 Amazon DocumentDB，您可以使用自己创建和控制的密钥对数据库进行加密 AWS Key Management Service ()AWS KMS。在通过 Amazon DocumentDB 加密运行的数据库集群上，静态存储于底层存储的数据都将加密。在同一个集群中的自动备份、快照和副本也会被加密。
- Amazon DocumentDB 已获得联邦风险与授权管理计划 (FedRAMP) 的授权。它拥有（美国）地区的 FedRAMP 高级授权， AWS GovCloud 对美国地区有 FedRAMP 中级授权。 AWS East/West 有关合规工作的详细信息 AWS 以及合规工作，请参阅合规性计划范围内的AWS 服务。

如果您不熟悉 AWS 服务，请使用以下资源了解更多信息：

- AWS 为计算、数据库、存储、分析和其他功能提供服务。有关所有 AWS 服务的概述，请参阅使用 Amazon Web Services 进行云计算。
- AWS 提供了许多数据库服务。有关最适合您环境的服务的指南，请参阅 AWS上的数据库。

集群

一个集群包含 0 到 16 个实例和一个管理这些实例的数据的集群存储卷。所有写入操作都通过主实例完成。所有实例（主实例和副本实例）都支持读取。集群的数据存储在集群卷中，副本存储在三个不同的可用区中。



基于 Amazon DocumentDB 5.0 实例的集群支持数据库集群的两种存储配置：亚马逊 DocumentDB 标准配置和亚马逊 DocumentDB。 I/O-optimized有关更多信息，请参阅 Amazon DocumentDB 集群存储配置。

实例

Amazon DocumentDB 实例是在云中运行的独立数据库环境。一个实例可以包含多个由用户创建的数据库。您可以使用 AWS 管理控制台 或创建和修改实例 AWS CLI。

实例的计算和内存容量由其实例类决定。您可以选择最能满足您需求的实例。如果一段时间后您的需求出现了变化，可以选择其他实例类。有关实例类的规格，请参阅实例类规格。

Amazon DocumentDB 实例仅在 Amazon VPC 环境中运行。Amazon VPC 允许您控制您的虚拟联网环境：可以选择自己的 IP 地址范围、创建子网以及配置路由和访问控制列表 (ACL)。

在创建 Amazon DocumentDB 实例之前，您必须创建一个集群以包含实例。

并非每个区域都支持所有实例类。下表显示了每个区域支持的实例类。

> **Note**
>
> 有关每个实例类中 Amazon DocumentDB 支持的实例类型的完整列表，请参阅 实例类规格。

不同区域支持的实例类

Region	R8G	R6GD	R6G	R5	R4	T4G	T3	Serverless
美国东部（俄亥俄州）	支持	支持	支持	支持	支持	支持	支持	支持
美国东部（弗吉尼亚州北部）	支持	支持	支持	支持	支持	支持	支持	支持
美国西部（俄勒冈州）	支持	支持	支持	支持	支持	支持	支持	支持
非洲（开普敦）			支持	支持		支持	支持	支持
南美洲（圣保罗）		支持	支持	支持		支持	支持	支持
亚太地区（香港）			支持	支持		支持	支持	支持
亚太地区（海得拉巴）			支持	支持		支持	支持	支持
亚太地区（马来西亚）			支持			支持	支持	
亚太地区（孟买）	支持	支持	支持	支持		支持	支持	支持
亚太地区（大阪）		支持	支持	支持		支持	支持	

Region	R8G	R6GD	R6G	R5	R4	T4G	T3	Serverless
亚太地区（首尔）	支持	支持	支持	支持		支持	支持	支持
亚太地区（悉尼）	支持	支持	支持	支持		支持	支持	支持
亚太地区（雅加达）	支持	支持	支持	支持		支持	支持	
亚太地区（墨尔本）			支持	支持		支持	支持	
亚太地区（新加坡）	支持	支持	支持	支持		支持	支持	支持
亚太地区（泰国）			支持			支持	支持	
亚太地区（东京）	支持	支持	支持	支持		支持	支持	支持
加拿大（中部）	支持	支持	支持	支持		支持	支持	支持
加拿大西部（卡尔加里）			支持	支持		支持	支持	
欧洲地区（法兰克福）	支持	支持	支持	支持		支持	支持	支持
欧洲（苏黎世）		支持	支持	支持		支持	支持	
欧洲地区（爱尔兰）	支持	支持	支持	支持	支持	支持	支持	支持
欧洲地区（伦敦）		支持	支持	支持		支持	支持	支持

Region	R8G	R6GD	R6G	R5	R4	T4G	T3	Serverless
欧洲地区（米兰）			支持	支持		支持	支持	支持
欧洲地区（巴黎）		支持	支持	支持		支持	支持	支持
欧洲（西班牙）	支持	支持	支持	支持		支持	支持	支持
欧洲地区（斯德哥尔摩）	支持	支持	支持	支持		支持	支持	
墨西哥（中部）			支持			支持	支持	
中东（阿联酋）：			支持	支持		支持	支持	支持
中国（北京）		支持	支持	支持		支持	支持	支持
中国（宁夏）			支持	支持		支持	支持	支持
以色列（特拉维夫）			支持	支持		支持	支持	
AWS GovCloud (US-West)	支持	支持	支持	支持			支持	支持
AWS GovCloud (US-East)		支持	支持	支持		支持	支持	支持

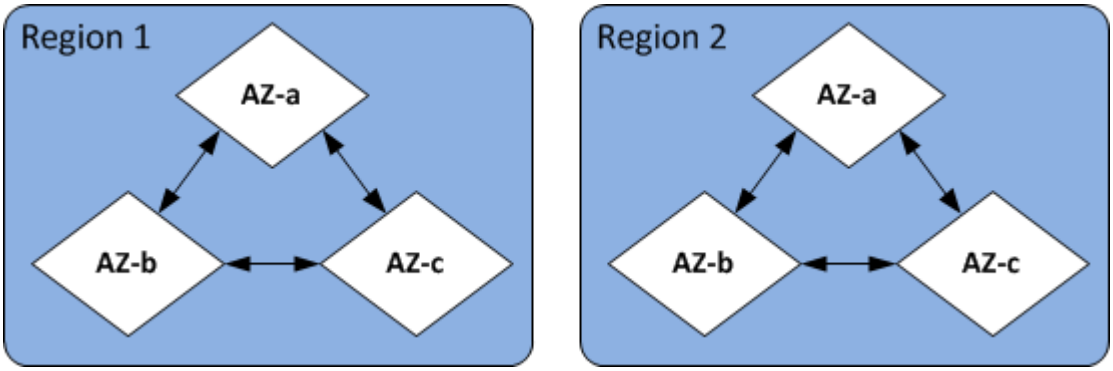
区域和可用区

区域和可用区定义集群和实例的物理位置。

Regions

AWS 云计算资源存放在世界不同地区（例如北美、欧洲或亚洲）的高可用性数据中心设施中。每个数据中心位置称为一个地区。

每个 AWS 区域都设计为与其他 AWS 区域完全隔离。每个区域内有多个可用区。在不同的可用区内启动节点，可以实现可能的最大容错。下图显示了 AWS 区域和可用区如何运作的高级视图。



可用区

每个 AWS 区域都包含多个不同的位置，称为可用区。每个可用区都被设计成不受其他可用区故障的影响，并提供低价、低延迟的网络连接，以连接到同一地区的其他可用区。通过在多个可用区中启动给定集群的实例，您可以防止应用程序出现可用区故障，尽管这种情况很少发生。

Amazon DocumentDB 架构将存储和计算分开。在存储层方面，Amazon DocumentDB 会在三个 AWS 可用区域中复制六份数据副本。例如，如果您在仅支持两个可用区的区域中启动 Amazon DocumentDB 集群，则系统会在三个可用区中以六种方式复制您的数据存储，但只有两个可用区中会有计算实例。

下表列出了在给 AWS 区域 定可用区中可用于为集群配置计算实例的数量。

区域名称	Region	可用区（计算）
美国东部（俄亥俄州）	`us-east-2`	3
美国东部（弗吉尼亚州北部）	`us-east-1`	6
美国西部（俄勒冈州）	`us-west-2`	4

区域名称	Region	可用区（计算）
非洲（开普敦）	af-south-1	3
南美洲（圣保罗）	sa-east-1	3
亚太地区（香港）	ap-east-1	3
亚太地区（海得拉巴）	ap-south-2	3
亚太地区（马来西亚）	ap-southeast-5	3
亚太地区（孟买）	ap-south-1	3
亚太地区（大阪）	ap-northeast-3	3
亚太地区（首尔）	ap-northeast-2	4
亚太地区（新加坡）	ap-southeast-1	3
亚太地区（悉尼）	ap-southeast-2	3
亚太地区（雅加达）	ap-southeast-3	3
亚太地区（墨尔本）	ap-southeast-4	3
亚太地区（泰国）	ap-southeast-7	3
亚太地区（东京）	ap-northeast-1	3
加拿大（中部）	ca-central-1	3
加拿大西部（卡尔加里）	ca-west-1	3
中国（北京）区域	cn-north-1	3
中国（宁夏）	cn-northwest-1	3

区域名称	Region	可用区（计算）
欧洲地区（法兰克福）	eu-central-1	3
欧洲（苏黎世）	eu-central-2	3
欧洲地区（爱尔兰）	eu-west-1	3
欧洲地区（伦敦）	eu-west-2	3
欧洲地区（米兰）	eu-south-1	3
欧洲地区（巴黎）	eu-west-3	3
欧洲（西班牙）	eu-south-2	3
欧洲地区（斯德哥尔摩）	eu-north-1	3
墨西哥（中部）	mx-central-1	3
中东（阿联酋）：	me-central-1	3
以色列（特拉维夫）	il-central-1	3
AWS GovCloud (US-West)	us-gov-west-1	3
AWS GovCloud (US-East)	us-gov-east-1	3

Amazon DocumentDB 定价

Amazon DocumentDB 集群根据以下组件进行计费：

- 实例小时数（按小时）：根据实例的实例类（例如，db.r5.xlarge）。定价以每小时为单位列出，但账单向下计算至秒，并以十进制形式显示时间。Amazon DocumentDB 使用量以一秒的增量进行计费，时长最少 10 分钟。有关更多信息，请参阅 管理实例类。

- I/O 请求（每月每 100 万个请求）-您在账单周期内发出的存储 I/O 请求总数。
- 备份存储（每月每 GiB）：备份存储是指与自动数据库备份和拍摄的有效数据库快照相关联的存储。延长备份保留期或增加快照创建数量，将增加数据库所消耗的备份存储。Backup 存储按流量计费 GB-months，每秒不适用。有关更多信息，请参阅 在 Amazon DocumentDB 中进行备份和还原。
- 数据传输（每 GB）— 从您的实例传入和传出互联网或其他 AWS 地区的数据。

有关详细信息，请参阅Amazon DocumentDB 定价。

免费试用

你可以使用 1 个月免费试用期免费试用 Amazon DocumentDB。有关更多信息，请参阅 Amazon DocumentDB 定价中的免费试用期或参阅 Amazon DocumentDB 免费试用期常见问题解答。

监控

您可以使用多种方法对实例性能和运行状况进行跟踪。您可以使用免费的 Amazon CloudWatch 服务来监控实例的性能和运行状况。您可以查找 Amazon DocumentDB 控制台上的性能图表。您可以订阅 Amazon DocumentDB 事件，以便在实例、快照、参数组或者安全组发生更改时收取通知。

有关更多信息，请参阅下列内容：

- 使用以下方式监控亚马逊 DocumentDB CloudWatch
- 使用记录亚马逊 DocumentDB API 调用 AWS CloudTrail

接口

您可以通过多种方式与 Amazon DocumentDB 进行交互，包括 AWS 管理控制台 和。 AWS CLI

AWS 管理控制台

AWS 管理控制台 这是一个简单的基于 Web 的用户界面。您可以从控制台中管理集群和实例，而无需进行任何编程。要访问亚马逊 DocumentDB 控制台，请登录 AWS 管理控制台 并打开亚马逊文档数据库控制台，网址为。https://console.aws.amazon.com/docdb

AWS CLI

您可以使用 AWS Command Line Interface (AWS CLI) 来管理您的 Amazon DocumentDB 集群和实例。只需极少配置即可从您喜爱的终端程序开始使用 Amazon DocumentDB 控制台提供的所有功能。

- 要安装 AWS CLI，请参阅安装 AWS 命令行界面。
- 要开始使用 AWS CLI 适用于亚马逊 DocumentDB 的，请参阅亚马逊 DocumentD B 的AWS 命令行界面参考。

MongoDB 驱动程序

要针对 Amazon DocumentDB 集群开发和编写应用程序，您还可以将 MongoDB 驱动程序与 Amazon DocumentDB 结合使用。有关更多信息，请参阅 启用了 TLS 的情况下的连接 或 禁用了 TLS 的情况下的连接 中的 MongoDB Shell 选项卡。

接下来做什么？

上面章节为您介绍了 Amazon DocumentDB 提供的基本基础设施组件。您下一步该做什么？ 根据您的情况，请参阅以下主题之一了解其用法：

- 使用创建集群和实例，开始使用 Amazon DocumentDB。 CloudFormation 亚马逊 DocumentDB 快速入门使用 CloudFormation
- 通过使用 入门指南 中的说明创建集群和实例开始使用 Amazon DocumentDB。
- 通过使用 开始使用 Amazon DocumentDB 弹性集群 中的说明创建弹性集群开始使用 Amazon DocumentDB。
- 使用 迁移到 Amazon DocumentDB 中的指南将您的 MongoDB 实现迁移到 Amazon DocumentDB

Amazon DocumentDB：工作方式

亚马逊 DocumentDB（兼容 MongoDB）是一项完全托管的数据库服务。 MongoDB-compatible 借助 Amazon DocumentDB，您可以运行与 MongoDB 相同的应用程序代码，并使用与 MongoDB 相同的驱动程序和工具。亚马逊 DocumentDB 与 MongoDB 3.6、4.0、5.0 和 8.0 兼容。

主题

- 亚马逊 DocumentDB 版本
- Amazon DocumentDB 端点
- TLS Support
- Amazon DocumentDB 存储
- Amazon DocumentDB 复制

- Amazon DocumentDB 可靠性
- 读取首选项选项
- TTL 删除
- 可计费资源

亚马逊 DocumentDB 版本

> **Note**
>
> 次要版本编号适用于亚马逊 DocumentDB 5.0 及更高版本。亚马逊 DocumentDB 3.6 和 4.0 仅使用带有引擎补丁的主要版本。

从亚马逊 DocumentDB 5.0 开始，引擎版本使用四级编号方案：major.major.minor.patch。

- 主要版本-由版本的前两个数字标识（例如，5.0）。主要版本代表了 MongoDB 的兼容级别。主要版本引入了新功能、性能改进和 MongoDB 兼容性更新。在主要版本（例如，从 4.0 到 5.0）之间进行升级需要主版本升级 (MVU)。有关更多信息，请参阅 Amazon DocumentDB 主版本就地升级。
- 次要版本-由版本的第三个数字标识（例如，5.0.1 1 中的）。次要版本包括相同主版本中的新功能、安全修复和错误修复等改进。您可以手动升级到新的次要版本。有关更多信息，请参阅 亚马逊 DocumentDB 次要版本升级。
- 补丁版本-引擎补丁应用于次要版本中，包含重要的安全修复和错误修复。补丁版本由引擎补丁版本（例如`3.0.17983`）跟踪，您可以通过运行来检查该版本`db.runCommand({getEngineVersion: 1})`。补丁将在集群的维护时段内应用。有关更多信息，请参阅 维护 Amazon DocumentDB。

创建新集群时，您可以指定任何可用的引擎版本。有关可用版本和支持日期的信息，请参阅Amazon DocumentDB 引擎版本支持日期。

使用 Amazon DocumentDB 时，首先要创建一个集群。集群由零个或多个数据库实例以及管理这些实例的数据的集群卷组成。Amazon DocumentDB 集群卷是一个跨多个可用区的虚拟数据库存储卷。每个可用区都有一个集群数据副本。

一个 Amazon DocumentDB 集群包含两个组件：

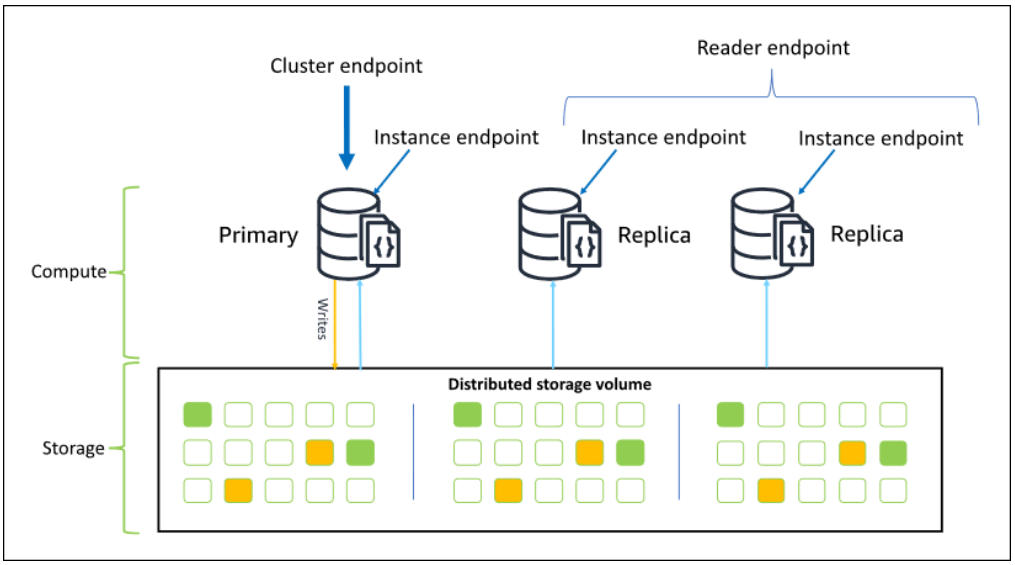
- 集群卷：使用云原生存储服务在三个可用区中复制数据六次，从而提供高度持久且可用的存储。一个 Amazon DocumentDB 集群只有一个集群卷，最多可存储 128 TiB 数据。

- 实例：提供数据库处理能力，以及向集群存储卷写入数据和从中读取数据的能力。一个 Amazon DocumentDB 集群可以有 0–16 个实例。

实例具有以下两种角色之一：

- 主实例：支持读写操作，并对集群卷执行所有数据修改。每个 Amazon DocumentDB 集群都有一个主实例。
- 副本实例：仅支持读取操作。除主实例之外，每个 Amazon DocumentDB 集群最多可拥有 15 个副本。拥有多个副本使您可以分配读取工作负载。此外，通过将副本置于单独的可用区中，您还可以提高集群可用性。

下图说明了 Amazon DocumentDB 集群中的集群卷、主实例和副本之间的关系：



集群实例无需为相同的实例类，可以根据需要预置和终止它们。此体系结构允许您独立于集群的存储扩展集群的计算容量。

当您的应用程序将数据写入主实例时，主实例会执行对集群卷的持久写入。然后，它将这个写入的状态（而非数据）复制到每个活动副本。Amazon DocumentDB 副本不参与处理写入，因此 Amazon DocumentDB 副本有利于读取扩展。来自 Amazon DocumentDB 副本的读取最终具有一致性，且副本滞后时间最短：通常少于主实例写入更新后的 100 毫秒。保证按照将数据写入主实例的顺序读取副本中的数据。副本滞后取决于数据更改的速率，高写入活动的时间段可能会增加副本滞后。有关更多信息，请参阅Amazon DocumentDB 指标中的 `ReplicationLag` 指标。

Amazon DocumentDB 端点

Amazon DocumentDB 提供多种连接选项，以便为各种使用案例提供服务。要连接到 Amazon DocumentDB 集群中的实例，您需要指定实例的端点。终端节点 是主机地址和端口号，用冒号分隔。

我们建议您使用集群终端节点以副本集模式（请参阅 作为副本集连接到 Amazon DocumentDB）连接到集群，除非您有用于连接读取器终端节点或实例终端节点的特定使用案例。要将请求路由到您的副本，请选择一项驱动程序读取首选项设置，以便在满足应用程序的读取一致性要求的同时，最大程度地提高读取扩展能力。`secondaryPreferred` 读取首选项将启用副本读取，并释放主实例以执行更多工作。

可从 Amazon DocumentDB 集群使用以下端点。

集群端点

集群终端节点 连接到集群的当前主实例。集群终端节点可用于读写操作。一个Amazon DocumentDB 集群只有一个集群端点。

集群终端节点为集群的读取/写入连接提供故障转移支持。如果集群的当前主实例失败并且您的集群至少有一个活动的只读副本，则集群终端节点会自动将连接请求重定向到新的主实例。在连接到 Amazon DocumentDB 集群时，我们建议您使用集群端点并以副本集模式（请参阅 作为副本集连接到 Amazon DocumentDB）连接到您的集群。

以下是示例 Amazon DocumentDB 集群端点：

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

以下是使用此集群终端节点的示例连接字符串：

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

有关查找集群终端节点的信息，请参阅查找集群的端点。

读取器端点

读取器终端节点 跨集群中的所有可用副本对只读连接进行负载平衡。如果通过 `replicaSet` 模式进行连接，则集群读取器端点将以集群端点发挥作用，这意味着在连接字符串中，副本集参数为 `&replicaSet=rs0`。在这种情况下，可以在主实例上执行写入操作。但是，如果连接指定

directConnection=true 的集群，则尝试通过与读取器端点的连接执行写入操作会导致错误。一个 Amazon DocumentDB 集群只有一个读取器端点。

如果集群只包含一个（主）实例，则读取器终端节点将连接到该主实例。将副本实例添加到 Amazon DocumentDB 集群时，读取器端点会在新副本处于活动状态后打开与新副本的只读连接。

以下是 Amazon DocumentDB 集群的示例读取器端点：

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

以下是使用读取器终端节点的示例连接字符串：

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

读取器终端节点对只读连接而不是读取请求进行负载平衡。如果某些读取器终端节点连接的使用频率高于其他连接，则读取请求可能无法在集群实例之间以相同方式取得平衡。建议以副本集形式连接到集群终端节点并利用 secondaryPreferred 读取首选项来分发请求。

有关查找集群终端节点的信息，请参阅查找集群的端点。

实例端点

实例终端节点 连接到集群中的特定实例。当前主实例的实例终端节点可用于读取和写入操作。但是，尝试对只读副本的实例终端节点执行写入操作会导致错误。Amazon DocumentDB 集群的每个活动实例有一个实例端点。

对于可能不适合使用集群终端节点或读取器终端节点的场景，实例终端节点提供对特定实例的连接的直接控制。示例使用案例是配置周期性只读分析工作负载。您可以配置大于普通副本实例的实例，使用其实例终端节点直接连接到新的较大实例，运行分析查询，然后终止实例。使用实例终端节点可防止分析流量影响其他集群实例。

下面是 Amazon DocumentDB 集群中单个实例的示例实例端点：

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

下面是使用此实例终端节点的示例连接字符串：

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

> **Note**
>
> 由于故障转移事件，实例的主要角色或副本角色可能会发生变化。您的应用程序永远不应该假设特定的实例终端节点是主实例。我们不建议连接到生产应用程序的实例终端节点。相反，我们建议您使用集群终端节点以副本集模式（请参阅 作为副本集连接到 Amazon DocumentDB）连接到您的集群。要想对实例故障转移优先级进行更高级的控制，请参阅了解 Amazon DocumentDB 集群容错能力。

有关查找集群终端节点的信息，请参阅查找实例的端点。

副本集模式

您可以通过指定副本集名称 `rs0` 以副本集模式连接到 Amazon DocumentDB 集群端点。在副本集模式下连接可以指定“Read Concern (读取问题)”、“Write Concern (写入问题)”和“Read Preference (读取首选项)”选项。有关更多信息，请参阅 读取一致性。

下面是以副本集模式连接的示例连接字符串：

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

在副本集模式下连接时，您的 Amazon DocumentDB 集群将作为副本集显示给您的驱动程序和客户端。在 Amazon DocumentDB 集群中添加和删除的实例会自动反映在副本集配置中。

每个 Amazon DocumentDB 集群均包含一个默认名称为 `rs0` 的副本集。该副本集名称无法修改。

在副本集模式下连接到集群终端节点是常规用途的推荐方法。

> **Note**
>
> Amazon DocumentDB 集群中的所有实例都在同一 TCP 端口上侦听连接。

TLS Support

有关使用传输层安全性协议（TLS）连接到 Amazon DocumentDB 的更多详细信息，请参阅 加密传输中数据。

Amazon DocumentDB 存储

Amazon DocumentDB 数据存储在集群卷中，该集群卷是使用固态驱动器 (SSD) 的单个虚拟卷。集群卷由六个数据副本组成，可在单个 AWS 区域的多个可用区之间自动复制。此复制有助于确保您的数据具有高持久性，减少数据丢失的可能性。它还有助于确保在故障转移期间您的集群具有更高可用性，因为您的数据副本已存在于其他可用区中。这些副本可以继续向 Amazon DocumentDB 集群中的实例提供数据请求服务。

数据存储的计费方式

随着数据量的增加，Amazon DocumentDB 会自动增加集群卷的大小。Amazon DocumentDB 集群卷可以增长到 128 TiB 的最大容量；但是，您只需为 Amazon DocumentDB 集群卷中您使用的空间付费。从 Amazon DocumentDB 4.0 开始，当移除数据（例如，通过丢弃集群或索引）时，整个分配的空间按相当的数量减少。因此，您可以通过删除不再需要的系列、索引、数据库等来减少存储费用。在 Amazon DocumentDB 版本 3.6 中，集群卷可以重复使用删除数据后释放的空间，但卷本身的大小从不会减小。因此，在版本 3.6 中，即使释放的空间被重复使用，在删除集合或索引后，您也可能不会看到存储空间的任何变化。

> **Note**
>
> 在 Amazon DocumentDB 3.6 中，存储成本基于存储“高水位线”（在任何时间点分配给 Amazon DocumentDB 集群的最大数量）。您可以通过避免创建大量临时信息或者在移除不需要的较旧数据之前加载大量新数据的 ETL 实践来管理成本。如果从 Amazon DocumentDB 集群中删除数据导致分配大量未使用的空间，则重置高水位需要使用 mongodump 或 mongorestore 之类的工具执行逻辑数据转储和还原以转储和还原到新集群。创建和还原快照不会 减少分配的存储，因为基础存储的物理布局在还原的快照中保持不变。

> **Note**
>
> 使用诸如mongodump和mongorestore之类的实用程序会根据正在读取和写入存储卷的数据的大小收 I/O 费。

有关亚马逊 DocumentDB 数据存储和 I/O 定价的信息，请参阅亚马逊 DocumentDB（兼容 MongoDB）定价和定价常见问题解答。

Amazon DocumentDB 复制

在 Amazon DocumentDB 集群中，每个副本实例都公开一个独立的端点。这些副本终端节点提供对集群卷中数据的只读访问。它们使您能够在多个复制实例上扩展数据的读取工作负载。它们还有助于在您的 Amazon DocumentDB 集群中改进数据读取性能并提高数据可用性。Amazon DocumentDB 副本也是失效转移目标，如果您的 Amazon DocumentDB 集群的主实例失效，则迅速升级这些副本。

Amazon DocumentDB 可靠性

Amazon DocumentDB 的设计具有可靠、持久和容错的特点。（为了改进可用性，您应配置您的 Amazon DocumentDB 集群，使其在不同的可用区拥有多个副本实例。）Amazon DocumentDB 包括多种自动功能，使其成为很可靠的数据库解决方案。

存储自动修复

Amazon DocumentDB 在三个可用区中维护数据的多个副本，从而大大降低了因存储故障而丢失数据的可能性。Amazon DocumentDB 会自动检测集群卷中的故障。如果集群卷的某个区段发生故障，Amazon DocumentDB 会立即修复该区段。它使用集群卷包含的其他卷中的数据以帮助确保已修复区段中的数据是最新的。因此，Amazon DocumentDB 将避免数据丢失，并减少执行时间点还原以从实例故障恢复的需求。

自动恢复缓存预热

Amazon DocumentDB 在与数据库不同的进程中管理其页面缓存，以便页面缓存可以独立于数据库而存在。万一发生数据库故障，页面缓存仍保留在内存中。这可确保在数据库重新启动时使用最新状态对缓冲池进行预热。

崩溃恢复

Amazon DocumentDB 设计为几乎可以立即从崩溃中恢复，并继续为您的应用程序数据提供服务。Amazon DocumentDB 在并行线程上异步执行崩溃恢复，以便在发生崩溃后使数据库能够打开并几乎立即恢复使用。

资源管理

Amazon DocumentDB 保护在服务中运行关键进程（例如运行状况检查）所需的资源。为做到这点且当实例正遭遇高内存压力时，Amazon DocumentDB 将对请求节流。因此，某些操作可能排队等候内存压力消退。如果内存压力持续存在，则排队的操作可能超时。您可以使用以下 CloudWatch 指标监控服务限制操作是否是由于内存不足所

致：LowMemThrottleQueueDepth、、LowMemThrottleMaxQueueDepth、LowMemNumOperationsTh
关更多信息，请参阅使用监控 Amazon DocumentDB。 CloudWatch如果 LowMem CloudWatch 指标导致您的实例持续承受内存压力，我们建议您扩大实例规模，为工作负载提供额外的内存。

读取首选项选项

Amazon DocumentDB 使用云原生共享存储服务，该服务跨三个可用区复制数据六次，以提供高水平持久性。Amazon DocumentDB 不依赖将数据复制到多个实例来实现持久性。无论集群是包含单个实例还是 15 个实例，集群的数据都具有持久性。

主题

- 写入持久性
- 读取隔离
- 读取一致性
- 高可用性
- 扩展读取

写入持久性

Amazon DocumentDB 使用一种独特的、分布式、容错、自我修复的存储系统。该系统跨三个 AWS 可用区复制六个数据副本 (V=6)，以提供高可用性和耐久性。在写入数据时，Amazon DocumentDB 在确认写入客户端之前，确保所有写入都已在大多数节点上持久记录。如果您运行的是三节点 MongoDB 副本集，则与 Amazon DocumentDB 相比，使用写入关注 `{w:3, j:true}` 将提供最佳配置。

对 Amazon DocumentDB 集群的写入必须由集群的写入器实例处理。尝试写入读取器会导致错误。来自 Amazon DocumentDB 主实例的已确认写入具有持久性，无法回滚。默认情况下，Amazon DocumentDB 有高度持久性并且不支持非持久写入选项。您无法修改持久性级别（即写关注）。Amazon DocumentDB 忽略 w=anything，并且有效的是 w: 3 和 j: true。您无法减少它。

在 Amazon DocumentDB 架构中，存储与计算是分离的，因此具有单个实例的集群拥有很高的持久性。持久性在存储层处理。因此，具有单个实例和具有三个实例的 Amazon DocumentDB 集群实现了相同级别的持久性。您可以根据特定用例配置集群，同时仍为数据提供高持久性。

对 Amazon DocumentDB 集群的写入操作在单个文档中是原子操作。

Amazon DocumentDB 不支持 `wtimeout` 选项，并且如果指定了值，将不会返回错误。对 Amazon DocumentDB 主实例的写入操作保证不会无限期阻塞。

读取隔离

从 Amazon DocumentDB 实例读取只返回查询开始之前已持久存在的数据。读取从不返回在查询开始执行后修改的数据，即，任何情况下都不会进行脏读取。

读取一致性

从 Amazon DocumentDB 集群读取的数据具有持久性，将不会回滚。您可以通过指定请求或连接的读取首选项来修改 Amazon DocumentDB读取的读取一致性。Amazon DocumentDB 不支持非持久读取选项。

从 Amazon DocumentDB 集群的主实例读取在正常操作条件下具有强一致性，并且具有先写后读一致性。如果在写入与后续读取之间发生故障转移事件，系统可能短暂返回不具有强一致性的读取。从只读副本的所有读取最终是一致的，以相同的顺序返回数据，并且通常具有小于 100 毫秒的副本滞后。

Amazon DocumentDB 读取首选项

仅当 Amazon DocumentDB 以副本集模式从集群端点读取数据时，才支持设置读取首选项。设置读取首选项会影响 MongoDB 客户端或驱动程序将读取请求路由到 Amazon DocumentDB 集群中的实例的方式。您可以为特定查询设置读取首选项，或者作为 MongoDB 驱动程序中的常规选项。（有关如何设置读取首选项的说明，请查阅客户端或驱动程序文档。）

如果您的客户端或驱动程序未在副本集模式下连接到 Amazon DocumentDB 集群端点，则指定读取首选项的结果并不明确。

Amazon DocumentDB 不支持将标记集设置为读取首选项。

支持的读取首选项

- **`primary`**：指定 `primary` 读取首选项有助于确保将所有读取请求路由到集群的主实例。如果主实例不可用，则读取操作将失败。`primary` 读取首选项可产生先写后读一致性，适用于先写后读一致性优先于高可用性和读取扩展的使用案例。

  以下示例指定 `primary` 读取首选项：

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

- **`primaryPreferred`**：指定 `primaryPreferred` 读取首选项会在正常操作期间将读取请求路由到主实例。如果发生主实例故障转移，则客户端会将请求路由到副本。`primaryPreferred` 读取首选项会在正常操作期间产生先写后读一致性，并在故障转移事件期间产生最终一致性读

取。primaryPreferred 读取首选项适用于先写后读一致性优先于读取扩展但仍需要高可用性的使用案例。

以下示例指定 primaryPreferred 读取首选项：

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

- **secondary**：指定 secondary 读取首选项可确保只将读取请求路由到副本，不会路由到主实例。如果集群中没有副本实例，则读取请求将失败。secondary 读取首选项会产生最终一致性读取，适用于主实例写入吞吐量优先于高可用性和先写后读一致性的使用案例。

  以下示例指定 secondary 读取首选项：

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

- **secondaryPreferred**：指定 secondaryPreferred 读取首选项可确保在一个或多个副本处于活动状态时将读取路由到只读副本。如果集群中没有活动的副本实例，则读取请求将路由到主实例。当读取由只读副本提供服务时，secondaryPreferred 读取首选项会产生最终一致性读取。当读取由主实例提供服务时，它会产生先写后读一致性（禁止故障转移事件）。secondaryPreferred 读取首选项适用于读取扩展和高可用性优先于先写后读一致性的使用案例。

  以下示例指定 secondaryPreferred 读取首选项：

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

- **nearest**：指定 nearest 读取首选项将仅基于测量的客户端与 Amazon DocumentDB 集群中所有实例之间的延迟来路由读取请求。当读取由只读副本提供服务时，nearest 读取首选项会产生最终一致性读取。当读取由主实例提供服务时，它会产生先写后读一致性（禁止故障转移事件）。nearest 读取首选项适用于实现尽可能低的读取延迟和高可用性优先于先写后读取一致性和读取扩展的使用案例。

  以下示例指定 nearest 读取首选项：

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

高可用性

Amazon DocumentDB 通过将副本用作主实例的失效转移目标来支持高度可用的集群配置。如果主实例失败，则一个 Amazon DocumentDB 副本将被提升为新的主实例，且出现短暂中断，在此期间，对主实例发出的读写请求将失败，并会出现异常。

如果您的 Amazon DocumentDB 集群不包含任何副本，则会在失败期间重新创建主实例。但是，提升 Amazon DocumentDB 副本要比重新创建主实例快得多。因此，我们建议您创建一个或多个 Amazon DocumentDB 副本作为失效转移目标。

旨在用作故障转移目标的副本应与主实例具有相同的实例类。它们应在不同于主实例的可用区中进行配置。您可以控制哪些副本作为首选故障转移目标。有关配置 Amazon DocumentDB 以实现高可用性的最佳实践，请参阅 了解 Amazon DocumentDB 集群容错能力。

扩展读取

Amazon DocumentDB 副本是读取扩展的理想选择。它们完全专用于集群卷上的读取操作，即副本不处理写入。数据复制发生在集群卷中，而不是在实例之间。因此，每个副本的资源都专用于处理查询，而不是复制和写入数据。

如果您的应用程序需要更多读取容量，则可以快速向集群添加副本（通常在不到十分钟的时间内完成）。如果您的读取容量要求减少，则可以删除不需要的副本。使用 Amazon DocumentDB 副本，您只需为所需的读取容量付费。

Amazon DocumentDB 通过使用“Read Preference (读取首选项)”选项支持客户端读取扩展。有关更多信息，请参阅 Amazon DocumentDB 读取首选项。

TTL 删除

在特定时间范围内无法保证从通过后台进程实现的 TTL 索引区域中进行删除，只能尽力而为。实例大小、实例资源利用率、文档大小和总体吞吐量等因素会影响 TTL 删除的时间。

TTL 监视器每次删除您的文档都会产生 IO 成本，这将增加您的账单费用。如果吞吐量和 TTL 删除率提高，您应预计到账单费用会因 IO 使用量的增加而增多。

您在现有集合上创建 TTL 索引时，必须删除所有过期文档，之后创建该索引。当前的 TTL 实现就删除集合中一小部分文档进行优化，如果从一开始就对集合启用 TTL，则这情况是常见的，并且如果需要一次性删除大量文档，则这可能导致比所需更高的 IOPS。

如果您不想创建 TTL 索引来删除文档，您也可以根据时间将文档归入各个集合中，并在不再需要文档时将这些集合删除。例如：您可以每周创建一个集合再删除，而不产生 IO 成本。这可能比使用 TTL 索引明显更具成本效益。

可计费资源

确定应计费的 Amazon DocumentDB 资源

作为一项完全托管的数据库服务，Amazon DocumentDB 会对实例、存储 I/Os、备份和数据传输收费。有关更多信息，请参阅 Amazon DocumentDB（与 MongoDB 兼容）定价。

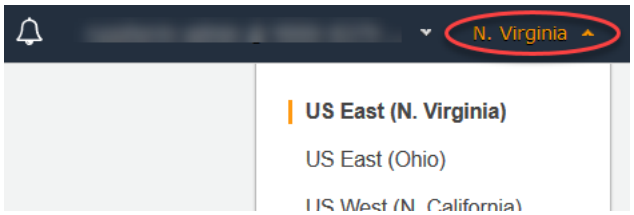
要发现您账户中的可计费资源并可能删除这些资源，您可以使用 AWS 管理控制台 或 AWS CLI。

使用 AWS 管理控制台

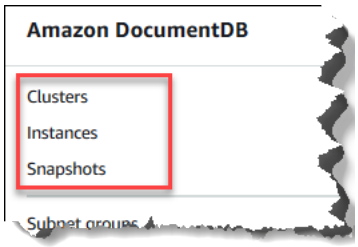
使用 AWS 管理控制台，您可以发现您为给定内容预配置的 Amazon DocumentDB 集群、实例和快照。 AWS 区域

查找集群、实例和快照

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 要查找默认区域以外的区域中的可计费资源，请在屏幕右上角选择要搜索 AWS 区域 的资源。



3. 在导航窗格中，选择您感兴趣的应计费资源类型：Clusters (集群)、Instances (实例) 或 Snapshots (快照)。



4. 右侧窗格中列出了该区域的所有已预置的集群、实例或快照。您将需要为集群、实例和快照付费。

使用 AWS CLI

使用 AWS CLI，您可以发现您为给定内容预配置的 Amazon DocumentDB 集群、实例和快照。 AWS 区域

查找集群和实例

以下代码将列出指定区域的所有集群和实例。如果要在默认区域中搜索集群和实例，可以省略 `--region` 参数。

Example

对于 Linux、macOS 或 Unix：

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

对于 Windows：

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

此操作的输出将类似于下文。

```
"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",
```

查找快照

以下代码将列出指定区域的所有快照。如果要在默认区域中搜索快照，可以省略 `--region` 参数。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-snapshots \
```

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

对于 Windows：

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

此操作的输出将类似于下文。

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

您只需要删除 `manual` 快照。`Automated` 快照会在您删除集群时删除。

删除不需要的应计费资源

要删除集群，必须先删除集群中的所有实例。

- 要删除实例，请参阅删除 Amazon DocumentDB 实例。

> **Important**
>
> 即使您删除集群中的实例，仍需要为与该集群关联的存储和备份用量付费。要停止所有计费，您还必须删除集群和手动快照。

- 要删除集群，请参阅删除 Amazon DocumentDB 集群。
- 要删除手动快照，请参阅删除集群快照。

什么是文档数据库？

一些开发人员不根据规范化行和列来考虑其数据模型。通常，在应用程序层中，数据表示为 JSON 文档，因为开发人员将其数据模型视为文档更为直观。

文档数据库的受欢迎程度不断提高，因为开发人员可以使用他们在其应用程序代码中使用的相同文档模型格式来保存数据库中的数据。文档数据库为灵活和敏捷的开发提供了强大直观的 API。

主题

- 文档数据库使用案例
- 了解文档
- 使用文档

文档数据库使用案例

无论您是否需要文档数据库或其他类型的数据库来管理数据，您的使用案例都将启动。对于需要一个灵活架构以实现快速迭代开发的工作负载来说，文档数据库很有用。以下是一些使用案例的示例，文档数据库可以为这些案例提供显著的优势：

主题

- 用户配置文件
- Real-time 大数据
- 内容管理

用户配置文件

由于文档数据库具有灵活架构，他们可以存储具有不同属性和数据值的文档。文档数据库是在线资料的实际解决方案，其中不同的用户提供不同类型的信息。使用文档数据库，通过仅存储特定于每个用户的属性，您可以高效地存储每个用户的资料。

假设用户选择添加或删除其资料中的信息。在这种情况下，其文档可轻易被一个更新的版本所取代，更新的版本包含任何最近添加的属性和数据，或者省略任何新省略的属性和数据。文档数据库轻松管理此级别的个人性和流动性。

Real-time 大数据

从历史上看，操作数据库和分析数据库是在不同的环境中维护的（分别是操作环境和分析数据库），这一事实阻碍了从操作数据中提取信息的能力。 business/reporting 在竞争激烈的商业环境中，能够实时提取运营信息至关重要。通过使用文档数据库，企业可以存储和管理任何来源的运营数据，并将数据并发到选择的 BI 引擎进行分析。无需具备两种环境。

内容管理

要有效地管理内容，您必须能够从各种来源收集和汇总内容，然后将其交付给客户。由于其灵活的架构，文档数据库非常适合用于收集和存储任何类型的数据。您可以使用它们来创建和引入新类型的内容，包括用户生成的内容，如图像、评论和视频。

了解文档

文档数据库用于将半结构化数据存储为文档，而不是像关系数据库那样在多个表之间对数据进行规范化，每个表都有唯一的固定结构。存储在文档数据库中的文档使用嵌套键值对来提供文档的结构或架构。不过，不同类型的文档可以存储在同一文档数据库中，从而满足了处理不同格式的类似数据的要求。例如，由于每个文档都是自描述性的 JSON-encoded ，因此主题中描述的在线商店文档文档数据库中的示例文档可以存储在同一个文档数据库中。

主题

- SQL 和非关系数据库对比
- 简单文档
- 嵌入文档
- 文档数据库中的示例文档
- 了解文档数据库中的规范化

SQL 和非关系数据库对比

下表对文档数据库 (MongoDB) 使用的术语与 SQL 数据库使用的术语进行了比较。

SQL	MongoDB
表	集合
行	文档

SQL	MongoDB
列	字段
主键	ObjectId
索引	索引
视图	视图
嵌套表或对象	嵌入文档
数组	数组

简单文档

文档数据库中的所有文档都是自描述的。尽管您可以使用其他编码方式，但本文档使用 JSON-like 格式化文档。

简单文档具有一个或多个字段，这些字段在文档中都处于同一级别。在以下示例中，字段 SSN、LName、FName、DOB、Street、City、State-Province、PostalCode 和 Country 在文档中属于同级。

```
{
   "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"
}
```

在简单文档中组织信息时，每个字段将被单独管理。要检索个人的地址，必须作为单个数据项检索 Street、City、State-Province、PostalCode 和 Country。

嵌入文档

复杂文档通过在文档中创建嵌入文档来组织数据。嵌入文档帮助管理分组中作为单个数据项的数据，这些数据项在特定情况下效率更高。使用上述示例，您可以在主文档中嵌入 Address 文档。这样做会生成以下文档结构：

```
{
   "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"
   }
}
```

现在，您可以作为单个字段 ("SSN":)、嵌入文档 ("Address":) 或嵌入文档的成员 ("Address": {"Street":}) 对文档中的数据进行访问。

文档数据库中的示例文档

如上所述，因为数据库中的每个文档是自描述的，文档数据库中的文档结构可能彼此不同。以下两个文档，一个是图书文档，另一个是为期刊文档，在结构上有所不同。不过，它们可以位于同一个文档数据库中。

下面是一个示例图书文档：

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

```
    },
    "Title": "Understanding Document Databases"
}
```

以下是带有两篇文章的示例期刊文档：

```
{
    "_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"
}
```

比较这两个文档的结构。借助关系数据库，您需要单独的“periodical”和“books”表或未使用字段的单个表作为 null 值，例如“发布”、“问题”、“文章”和“MI”。由于文档数据库是半结构化的，每个文档都定义了自己的结构，因此这两个文档可以在同一个文档数据库中共存，而没有 null 字段。文档数据库善于处理稀疏数据。

针对文档数据库进行开发，可以实现快速、迭代的开发。这是因为您可以动态地更改文档的数据结构，而不必更改整个集合的架构。文档数据库非常适合敏捷开发和动态变化的环境。

了解文档数据库中的规范化

文档数据库未规范化；在一个文档中发现的数据可以在另一个文档中重复。此外，文档之间可能存在一些数据差异。例如，假设您在网上商店购物，所有购物的详细信息都存储在一个文档中。文档应类似于以下 JSON 文档：

```
{
    "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"
}
```

所有这些信息都作为文档存储在交易集合中。后来，您意识到忘记购买一件物品。因此，您再次登录同一家商店，进行另一次购买，这也是作为另一个文档存储在交易集合中。

```
{
    "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"
}
```

请注意这两个文档之间的冗余：您的姓名和购物者 ID（此外，如果您使用相同的信用卡，您的信用卡信息）。但这没关系，因为存储成本低廉，每个文档都完整记录了单个交易，只需简单的键值查询就可以快速检索，不需要连接。

两个文档之间也存在明显差异：您的信用卡信息。这只是一个明显差异，因为可能您每次购买都使用不同的信用卡。每个文档对其记录的交易都是准确的。

使用文档

作为文档数据库，Amazon DocumentDB 可以轻松存储、查询和索引 JSON 数据。在 Amazon DocumentDB 中，除了没有对所有文档强制实施单一架构外，文档数据库集合类似于关系数据库中的表。集合允许您将类似的文档分组在一起，同时将它们保存在同一数据库中，而不需要它们在结构上相同。

使用前面部分的示例文档，您可能会有针对 `reading_material` 和 `office_supplies` 的集合。您的软件有责任强制执行文档属于哪个集合。

以下示例使用 MongoDB API 来展示如何添加、查询、更新和删除文档。

主题

- 添加文档
- 查询文档

- 更新文档
- 删除文档

添加文档

在 Amazon DocumentDB 中，首次向集合添加文档时会创建数据库。在此示例中，您将在 test 数据库中创建名为 example 的集合，该集合是连接到集群时的默认数据库。由于插入第一个文档时隐式创建连接，因此集合名称没有错误检查。因此，集合名称中的拼写错误（例如 eexample 而不是 example）将创建文档并将其添加到 eexample 集合，而不是预期集合中。错误检查必须由您的应用程序进行处理。

以下示例使用 MongoDB API 来添加文档。

主题

- 添加单个文档
- 添加多个文档

添加单个文档

要将单个文档添加至集合，请使用带有您想添加至集合的文档的 insertOne({}) 操作。

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

此操作的输出将类似于下文（JSON 格式）。

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

添加多个文档

要将多个文档添加至集合，请使用带有您想添加至集合的文档列表的 `insertMany( [{},..., {}] )` 操作。虽然此特定列表中的文档具有不同的架构，但它们都可以添加至同一个集合。

```
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
            }
        }
    ]
)
```

此操作的输出将类似于下文（JSON 格式）。

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

```
    ]
}
```

查询文档

有时，您可能需要查看在线商店的库存，这样客户就能看到并购买您销售的物品。查询集合相对容易，无论您想要集合中的所有文档，还是仅需要那些满足特定标准的文档。

要查询文档，请使用 `find()` 操作。`find()` 命令具有单个文档参数，该参数定义了在选择要返回的文档时要使用的标准。`find()` 的输出是一个文档，其格式为一行文本，不含换行符。要格式化输出文档，从而更加轻松地读取，请使用 `find().pretty()`。本主题中的所有示例都使用 `.pretty()` 设置输出的格式。

使用您在前两个练习（即 `insertOne()` 和 `insertMany()`）中插入到 `example` 集合中的四个文档。

主题

- 检索集合中的所有文档
- 检索与字段值匹配的文档
- 检索与嵌入文档匹配的文档
- 检索与嵌入文档中的字段值匹配的文档
- 检索与数组匹配的文档
- 检索与数组中的值匹配的文档
- 使用运算符检索文档

检索集合中的所有文档

要检索集合中的所有文档，请将 `find()` 操作和空查询文档结合使用。

以下查询返回 `example` 集合中的所有文档。

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

检索与字段值匹配的文档

要检索与字段和值匹配的所有文档，请将 `find()` 操作和查询文档（标识要匹配的字段和值）结合使用。

通过使用前述文档，此查询将返回其中“Item”字段等于“Pen”的所有文档。

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

检索与嵌入文档匹配的文档

要查找与嵌入文档匹配的所有文档，请将 `find()` 操作和查询文档（指定嵌入文档名称和嵌入文档的所有字段和值）结合使用。

在与嵌入文档匹配时，该文档的嵌入文档的名称必须与查询中的名称相同。此外，嵌入文档中的字段和值必须与查询匹配。

以下查询仅返回“Poster Paint”文档。这是因为“Pen”具有不同的“`OnHand`”和“`MinOnHand`”值，并且“Spray Paint”比查询文档多一个字段 (`OrderQnty`)。

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

检索与嵌入文档中的字段值匹配的文档

要查找与嵌入文档匹配的所有文档，请将 `find()` 操作和查询文档（指定嵌入文档名称和嵌入文档的所有字段和值）结合使用。

考虑到上述文档，以下查询使用“点表示法”来指定嵌入文档和感兴趣的字段。将返回所有与这些内容匹配的文档，而不管嵌入文档中可能存在哪些其他字段。此查询将返回“Poster Paint”和“Spray Paint”，因为它们与指定的字段和值匹配。

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

检索与数组匹配的文档

要查找所有与数组匹配的文档，请将 `find()` 操作和您感兴趣的数组名称以及数组中的所有值结合使用。此查询将返回所有包含带该名称的数组（其中数组值和顺序与查询中的完全相同）的文档。

以下查询仅返回“Pen”，因为“Poster Paint”具有其他颜色 (White)，并且“Spray Paint”具有顺序不同的颜色。

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

检索与数组中的值匹配的文档

要查找所有具有特定数组值的文档，请将 `find()` 操作与您感兴趣的数组名称和值结合使用。

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

上述操作将返回所有三个文档，因为它们都有一个名为 Colors 的数组，并且此数组中的某个位置具有“Red”值。如果您指定值“White”，则查询将仅返回“Poster Paint”。

使用运算符检索文档

以下查询返回“Inventory.OnHand”值小于 50 的所有文档。

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

有关支持的查询运算符的列表，请参阅 查询和投影运算符。

更新文档

通常，您的文档不是静态的，而是作为应用程序工作流的一部分进行更新。以下示例展示了更新文档的一些方法。

要更新现有文档，请使用 update() 操作。update() 操作具有两个文档参数。第一个文档标识要更新的文档。第二个文档指定要进行的更新。

在更新现有字段时，无论该字段是简单字段、数组还是嵌入文档 ，都请指定字段名及其值。在操作结束时，旧文档中的字段似乎已替换为新的字段和值。

主题

- 更新现有字段的值
- 添加新字段
- 替换嵌入文档
- 将新字段插入嵌入文档
- 从文档中删除字段
- 从多个文档中删除字段

更新现有字段的值

在下面的更新操作中，使用您之前添加的以下四个文档。

```
{
```

```
    "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
    }
}
```

更新简单字段

要更新简单字段，请将 `update()` 和 `$set` 结合使用以指定字段名和新值。以下示例将 `Item` 从“Pen”更改为“Gel Pen”。

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

此操作的结果将类似于下文。

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

更新数组

以下示例将现有颜色数组替换为新数组（其中包括 Orange）并从颜色列表中删除 White。新的颜色列表的顺序是在 update() 操作中指定的。

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

此操作的结果将类似于下文。

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

添加新字段

要通过添加一个或多个新字段修改文档，请使用带有查询文档的 update() 操作，该查询文档使用 $set 运算符标识要插入的文档及新字段和值。

以下示例将值为 3.99 的字段 UnitPrice 添加到 Spray Paints 文档。请注意，值 3.99 是数字，而不是字符串。

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

```
    { "Item": "Spray Paint" },
    { $set: { "UnitPrice": 3.99 } }
)
```

此操作的结果将类似于下文。

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

替换嵌入文档

要通过替换嵌入文档来修改文档，请将 update() 操作和文档（使用 $set 运算符标识嵌入文档及其新字段和值）结合使用。

给定以下文档。

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

替换嵌入文档

以下示例将当前 Date 文档替换为新文档，后者仅具有 Month 和 Day 字段；并且已消除 Year。

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

此操作的结果将类似于下文。

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

将新字段插入嵌入文档

将字段添加到嵌入文档

要通过向嵌入文档添加一个或多个新字段来修改文档，请将 update() 操作与文档（标识嵌入文档）结合使用，并使用“点表示法”通过 $set 运算符指定嵌入文档以及要插入的新字段和值。

给定下面的文档，以下代码使用“点表示法”将 Year 和 DoW 字段插入嵌入式 Date 文档中，并将 Words 插入父文档中。

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

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

此操作的结果将类似于下文。

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

```
        "Day": 18,
        "Year": 1987,
        "DoW": "Saturday"
    },
    "Words": 2482
}
```

从文档中删除字段

要通过删除文档中的字段来修改文档，请将 update() 操作与查询文档（标识要从中删除字段的文档）和 $unset 运算符（指定要删除的字段）结合使用。

以下示例从前述文档中删除 Words 字段。

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

此操作的结果将类似于下文。

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

从多个文档中删除字段

要通过从多个文档中删除字段来修改文档，请使用具有 $unset 运算符和 multi 选项的 update() 操作设置为 true。

以下示例从示例集合中的所有文档中删除 Inventory 字段。如果文档没有 Inventory 字段，则不对该文档采取任何操作。如果省略了 multi: true，则仅只在符合标准的第一个文档上执行此操作。

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

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

删除文档

要从您的数据库删除文档，请使用 `remove()` 操作，指定要删除哪个文档。以下代码将从 `example` 集合中删除“Gel Pen”。

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

要删除数据库中的所有文档，请将 `remove()` 操作和空查询结合使用。

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

开始使用 Amazon DocumentDB

有许多连接和开始使用 Amazon DocumentDB 的方式。本指南是用户开始使用我们强大文档数据库的最快捷、最简便的方法。本指南使用 AWS CloudShell 直接从 AWS 管理控制台连接和查询 Amazon DocumentDB 集群。符合 AWS 免费套餐条件的新客户可以免费使用亚马逊 DocumentDB CloudShell。如果您的 AWS CloudShell 环境或 Amazon DocumentDB 集群使用了超出免费套餐范围的资源，则您需要为这些资源支付正常 AWS 费率。本指南将让您在五分钟内入门 Amazon DocumentDB。

> **Note**
>
> 本指南中的说明专门用于创建和连接可用 Amazon DocumentDB 和 Amazon DocumentDB 的基于实例的集群。 AWS CloudShell
>
> - 如果您想要创建并连接到 Amazon DocumentDB 弹性集群，请参阅 开始使用 Amazon DocumentDB 弹性集群。
> - 如果您位于 AWS 中国地区，自动连接 Amazon EC2请参阅。

主题

- 先决条件
- 步骤 1：创建集群
- 步骤 2：Connect 连接到您的集群
- 步骤 3：插入和查询数据
- 第 4 步：探索

先决条件

在创建第一个 Amazon DocumentDB 集群之前，您必须执行以下操作：

已创建 Amazon Web Services（AWS）账户

在开始使用 Amazon DocumentDB 之前，您必须拥有 Amazon Web Services（AWS）账户。该 AWS 账户是免费的。您只需为使用的服务和资源付费。

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明操作。

   在注册时，将接到电话或收到短信，要求使用电话键盘输入一个验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

设置所需的 AWS Identity and Access Management (IAM) 权限。

要管理集群、实例和集群参数组等 Amazon DocumentDB 资源，需要 AWS 能够对您的请求进行身份验证的证书。有关更多信息，请参阅 适用于 Amazon DocumentDB 的 Identity and Access Management。

1. 在的搜索栏中 AWS 管理控制台，键入 IAM，然后在出现的下拉菜单中选择 IAM。
2. 一旦您进入 IAM 控制台，就从导航窗格中选择用户。
3. 选择您的用户名。
4. 选择添加权限。
5. 选择直接附加策略。
6. 在搜索栏中键入 `AmazonDocDBFullAccess`，并且一旦它出现在搜索结果中就选择之。
7. 选择下一步。
8. 选择添加权限。

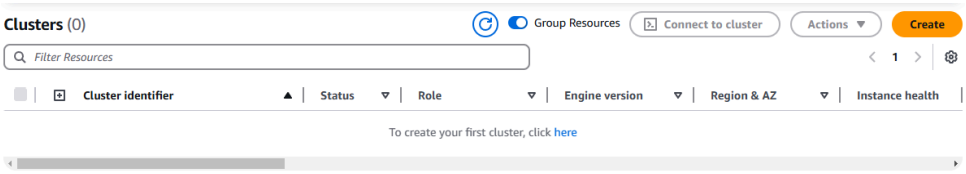
> **Note**
>
> 您的 AWS 账户在每个区域都包含一个默认 VPC。如果您选择使用 Amazon VPC，请按照《Amazon VPC 用户指南》的创建 Amazon VPC 主题中的步骤进行操作。

步骤 1：创建集群

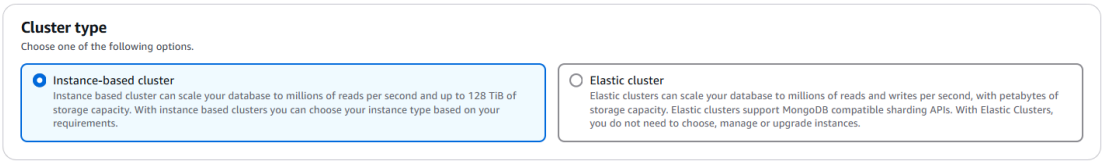
在这一步骤中，您将创建 Amazon DocumentDB 集群。

1. 登录并打开亚马逊文档数据库控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb

2. 在 Amazon DocumentDB 管理控制台上，集群下，选择创建。



3. 在创建 Amazon DocumentDB 集群页面的集群类型部分，选择Instance-based 集群（这是默认选项）。

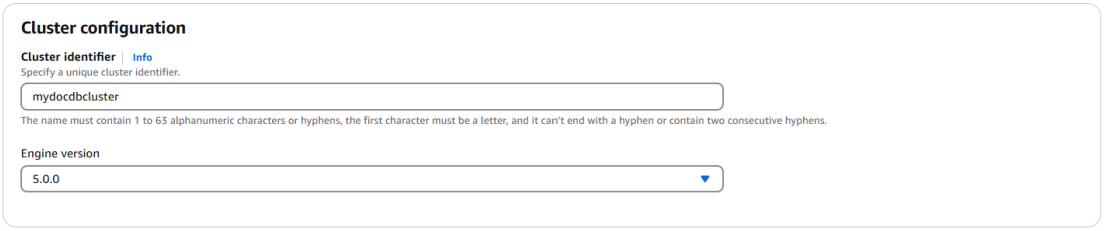


> **Note**
>
> 此类别中的另一个选项是弹性集群。要了解有关 Amazon DocumentDB 弹性集群的更多信息，请参阅 Amazon DocumentDB 弹性集群。

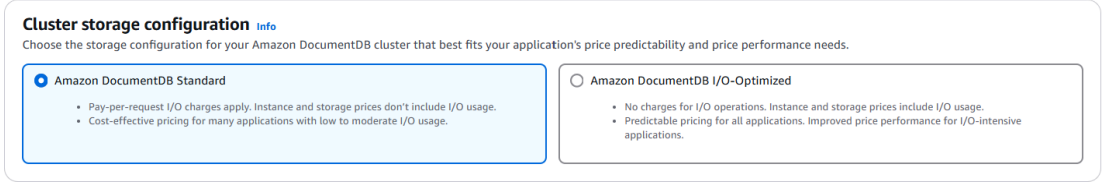
4. 在集群配置部分中：

   a. 在集群标识符中，输入唯一名称，例如 **`mydocdbcluster`**。请注意，无论如何输入，控制台都会将所有集群的名称更改为小写。

   b. 对于主引擎版本，选择 5.0.0。



5. 在集群存储配置部分，选择 Amazon DocumentDB 标准（此为默认选项）。



> **Note**
>
> 此类别中的另一个选项是 Amazon Doc I/O-Optimized umentDB。要了解有关任一选项的更多信息，请参阅 Amazon DocumentDB 集群存储配置

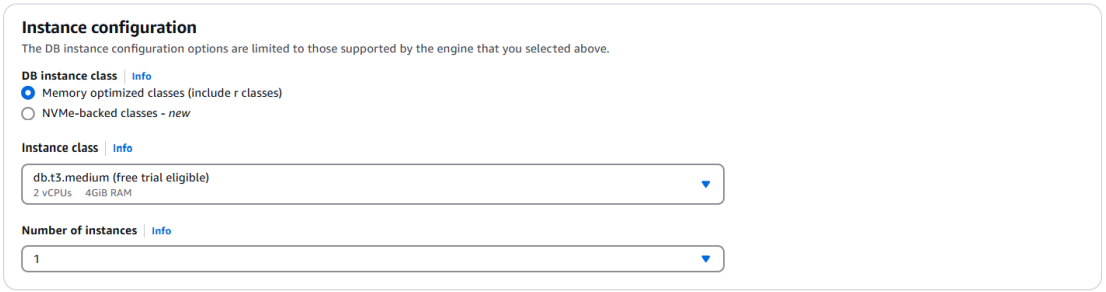
6. 在实例配置部分：

   a. 对于数据库实例类，选择内存优化类（包括 r 类）（这是默认值）。

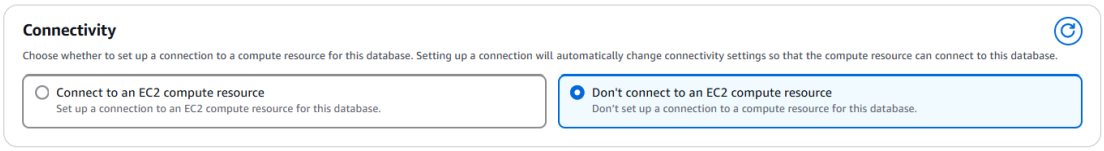
      另一个实例选项是NVMe-backed 类。要了解更多信息，请参阅NVMe-backed 实例。

   b. 对于实例类，请选择 db.t3.medium。这有资格 AWS 免费试用。

   c. 对于实例数，选择 1 个实例。选择一个实例有助于成本最小化。如果这是生产系统，我们建议您预配置三个实例以便可用性高。

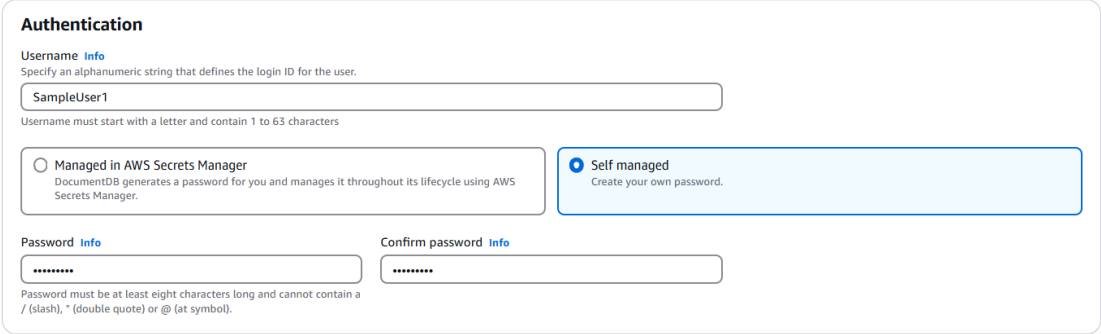


7. 在连接部分中，保留默认设置：不连接到 EC2 计算资源。



8. 在身份验证部分中，输入主要用户的用户名，然后选择自行管理。输入密码，然后确认密码。

   如果您改为在中选择 “托管” AWS Secrets Manager，使用 Amazon DocumentDB 进行密码管理以及 AWS Secrets Manager请参阅，了解更多信息。



9. 使其他所有选项保持默认，并选择创建集群。

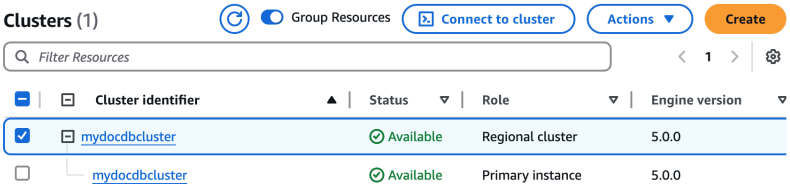
Amazon DocumentDB 现在正配置您的集群，这可能耗时长达数分钟完成。

> **Note**
>
> 有关集群状态值的信息，请参阅“监控 Amazon DocumentDB”一章中的 集群状态值。

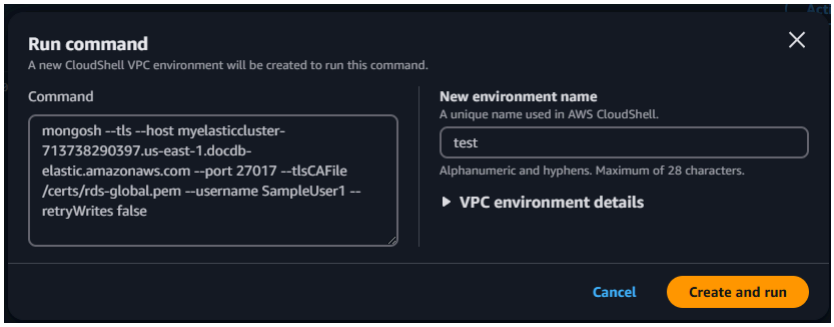
步骤 2：Connect 连接到您的集群

使用连接到您的亚马逊文档数据库集群。 AWS CloudShell

1. 在 Amazon DocumentDB 管理控制台上的集群下，找到您创建的集群。单击集群旁边的复选框，选择您的集群。



2. 选择 Connect to cluster（位于 “操作” 下拉菜单旁边）。只有在您单击集群旁边的复选框并且区域集群和主实例的状态都显示为可用后，才会启用此按钮。将出现 “ CloudShell 运行命令” 屏幕。
3. 在新环境名称字段中，输入唯一的名称，例如“test”，然后选择创建并运行。将自动为您的 Amazon DocumentDB 数据库配置 VPC 环境详细信息。



4. 出现提示时，输入您在“步骤 1：创建 Amazon DocumentDB 集群”（子步骤 7）中创建的密码。



在您输入密码并且提示符变成 `rs0 [direct: primary] <env-name>>` 后，表示您已成功连接到您的 Amazon DocumentDB 集群。

> **Note**
>
> 有关对 流进行问题排查的更多信息，请参阅 Amazon DocumentDB 问题排查。

步骤 3：插入和查询数据

现在，您已连接到自己的集群，您可以运行几个查询来熟悉如何使用文档数据库。

1. 要插入单个文档，请输入以下内容：

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

您会得到以下输出：

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

```
}
```

2. 您可以读取您用 findOne() 命令编写过的文档（因为它只返回单个文档）。输入以下：

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

您会得到以下输出：

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

3. 要执行若干更多查询，请考虑游戏个人资料用例。首先，将几个条目插入标题为 profiles 的集合。输入以下：

```
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 }
])
```

您会得到以下输出：

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

4. 使用 find() 命令返回个人资料集合中的所有文档。输入以下：

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

您将获得与您在步骤 3 中键入的数据相匹配的输出。

5. 利用筛选器对单个文档使用查询。输入以下：

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

您会得到以下输出：

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

6. 现在，让我们尝试查找个人资料并使用 findAndModify 命令修改它。我们将用以下代码向用户 Matt 给予额外的 10 分：

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

你得到以下输出（请注意，他的分数尚未增加）：

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

7. 你可以借助以下查询验证他的分数是否已变化：

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

   您会得到以下输出：

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

第 4 步：探索

恭喜您！您已成功完成基于 Amazon DocumentDB 实例的集群入门指南。

接下来做什么？ 了解如何充分利用这个数据库及其热门功能：

- 管理 Amazon DocumentDB
- 扩展
- 备份和还原

> **Note**
>
> 您通过此入门练习创建的集群将持续产生费用，除非您将其删除。有关说明，请参阅删除 Amazon DocumentDB 集群。

亚马逊 DocumentDB 的功能和配置

下表比较了 Amazon DocumentDB 引擎版本和弹性集群的特性、功能和支持的实例类型。使用这些表来确定哪个引擎版本最适合您的工作负载要求。

发动机状态

按版本划分的引擎状态

特性/能力	v3.6	v4.0	v5.0	v8.0	
Support 状态	EOL：2026 年 3 月	活动	已激活（LTS 可用）	活跃（GA 2025 年 11 月）	
提供延长 Support	是（已付款，直到 2029 年 3 月）	N/A	N/A	N/A	
LTS 版本已上线	否	否	是 (v5.0 LTS)	否	
弹性集群	否	否	是	否	

集群类型

按版本划分的集群类型

集群类型	v3.6	v4.0	v5.0	v8.0	弹性集群
Instance-based 集群	支持	是	是	是	是
无服务器实例	否	否	是	是	否

功能

按引擎版本划分的功能和配置

功能	v3.6	v4.0	v5.0	v8.0	弹性集群
全局集群	否	是	是	是	否
Multi-AZ (3 个可用区)	支持	是	是	是	是（每个分片）
支持只读副本	15	15	15	15	每个分片 15
ACID 事务	否	是	是	是	否
变更流	支持	是	是	是	否
向量搜索	否	否	是	是	否
性能详情	支持	是	是	是	否
In-place 主要版本升级 (MVU)	是（到 5.0）	是（到 5.0）	是（到 8.0）	否（仅限目标）	否
Client-side 字段级加密 (FLE)	否	否	是	是	否
I/O-Optimized 存储	否	否	是	是	否
TLS 证书轮换（不重启）	是的（补丁 1.0.208662+）	是的（补丁 2.0.10179+）	是的（补丁 3.0.4780+）	支持	是
Cross-Region 快照副本	支持	是	是	是	是
Cross-account 快照共享	支持	是	是	是	否
审计和慢速查询日志	支持	是	是	是	是

功能	v3.6	v4.0	v5.0	v8.0	弹性集群
Dual-stack IPv4/IPv6	否	是的（补丁 2.0.11747 +）	是的（补丁 3.0.15902 +）	是	N/A
排序规则	否	否	否	是	否
观看次数	否	否	否	是	否
压缩支持	否	否	LZ4	LZ4/Ztsd	否
文本索引	否	否	V1	V2	否
FedRamp () GovCloud	否	否	是	是	否

实例类型

按版本划分的实例类型支持

实例类型	v3.6	v4.0	v5.0	v8.0
db.r4（上一代）	是	否	否	否
db.r5	支持	是	是	是
db.r6g (Graviton2)	否	是	是	是
db.r6gd (Graviton2) NVMe-backed	否	否	是	是
db.r8g (Graviton4)	否	否	是	是
db.t3.medium	是	是	是	是
db.t4g.medium (Graviton 2)	否	是	是	是
无服务器	否	否	是	是

AWS 集成

AWS 按版本划分的集成支持

AWS 集成/功能	v3.6	v4.0	v5.0	v8.0	弹性集群
AWS Backup	支持	是	是	是	否
AWS Secrets Manager	是	是	是	是	否
AWS Lambda （原生 ESM）	否	是	是	否	否
亚马逊 OpenSearch 服务摄取 () Zero-ETL	否	是	是	是	否
AWS DMS 迁移支持	支持	是	是	是（从 5.0 开始到 AWS DMS）	是
AWS Identity and Access Management authentication	否	否	是（基于 5.0 实例）	是	否
AWS CloudShell	否	否	是	是	否
AWS Key Management Service	是	是	是	是	是
AWS CloudTrail	是	是	是	是	是

亚马逊 DocumentDB 快速入门使用 CloudFormation

此节包含使用 CloudFormation 帮助您快速实现 Amazon DocumentDB（与 MongoDB 兼容）入门的步骤和其他信息。有关 Amazon DocumentDB 的一般信息，请参阅 Amazon DocumentDB（兼容 MongoDB）是什么。

这些说明使用 AWS CloudFormation 模板在您的默认 Amazon VPC 中创建集群和实例。有关自行创建这些资源的说明，请参阅开始使用 Amazon DocumentDB。

> **Important**
>
> 此模板创建的 CloudFormation 堆栈会创建多个资源，包括 Amazon DocumentDB（例如，集群和实例）和亚马逊弹性计算云（例如，子网组）中的资源。
> 其中一些资源并非免费套餐资源。有关定价信息，请参阅 Amazon DocumentDB 定价和 Amazon EC2 定价。在您用堆栈完成后，您可以删除它以终止任何收费。

此 CloudFormation 堆栈仅用于教程目的。如果您将此模板用于生产环境，建议您使用更严格的 IAM 策略和安全。有关安全资源的信息，请参阅 Amazon VPC 安全和 Amazon EC2 网络与安全。

主题

- 先决条件
- 启动亚马逊 DocumentDB CloudFormation 堆栈
- 访问 Amazon DocumentDB 集群
- 终止保护和删除保护

先决条件

在创建 Amazon DocumentDB 集群之前，必须执行以下操作：

- 默认 Amazon VPC
- 必备的 IAM 权限

所需的 IAM 权限

以下权限允许您为 CloudFormation 堆栈创建资源：

AWS 托管策略

- `AWSCloudFormationReadOnlyAccess`
- `AmazonDocDBFullAccess`

其他 IAM 权限

以下策略概述了创建和删除此 CloudFormation 堆栈所需的其他权限。

在以下示例中，将每个*`user input placeholder`*示例替换为您的资源信息。

Amazon EC2 密钥对

在要创建 CloudFormation 堆栈的区域中，您必须有可用的密钥对（和 PEM 文件）。如果您需要创建密钥对，请参阅《Amazon EC2 用户指南》中的使用 Amazon EC2 创建密钥对。

启动亚马逊 DocumentDB CloudFormation 堆栈

本节介绍如何启动和配置 Amazon DocumentDB CloudFormation 堆栈。

1. 登录到 a AWS 管理控制台 t https://console.aws.amazon.com/。
2. 下表列出每个 AWS 区域的 Amazon DocumentDB 堆栈模板。为 AWS 区域 要启动堆栈的选择启动堆栈。

Region	查看模板	在 Designer 中查看	启动
美国东部（俄亥俄州）	查看模板	在 Designer 中查看	Launch Stack
美国东部（弗吉尼亚州北部）	查看模板	在 Designer 中查看	Launch Stack
美国西部（俄勒冈州）	查看模板	在 Designer 中查看	Launch Stack
亚太地区（孟买）	查看模板	在 Designer 中查看	Launch Stack
亚太地区（首尔）	查看模板	在 Designer 中查看	Launch Stack

Region	查看模板	在 Designer 中查看	启动
亚太地区（新加坡）	查看模板	在 Designer 中查看	Launch Stack
亚太地区（悉尼）	查看模板	在 Designer 中查看	Launch Stack
亚太地区（东京）	查看模板	在 Designer 中查看	Launch Stack
加拿大（中部）	查看模板	在 Designer 中查看	Launch Stack
欧洲地区（法兰克福）	查看模板	在 Designer 中查看	Launch Stack
欧洲地区（爱尔兰）	查看模板	在 Designer 中查看	Launch Stack
欧洲地区（伦敦）	查看模板	在 Designer 中查看	Launch Stack
欧洲地区（巴黎）	查看模板	在 Designer 中查看	Launch Stack

3. Create stack(创建堆栈)：描述您选定的 Amazon DocumentDB 模板。每个堆栈都基于一个模板（JSON 或 YAML 文件），其中包含有关要包含在堆栈中的 AWS 资源的配置。由于您选择从上面提供的模板启动堆栈，因此您的模板已配置为 AWS 区域 为您选择的创建一个 Amazon DocumentDB 堆栈。

   当您启动 CloudFormation 堆栈时，Amazon DocumentDB 集群的删除保护在默认情况下处于禁用状态。如果要为集群启用删除保护，请完成以下步骤。否则，请选择 Next (下一步) 以继续执行下一步。

   要为 Amazon DocumentDB 集群启用删除保护，请执行以下操作：

   1. 从“创建堆栈”页面的右下角选择“设计器中查看”。
   2. 在控制台生成的 “ CloudFormation 设计器” 页面中，使用集成的 JSON 和 YAML 编辑器修改模板。滚动到 `Resources` 部分并修改它以包括 `DeletionProtection`，如下所示。有关使用 D CloudFormation esigner 的更多信息，请参阅什么是 CloudFormation 设计器？ 。

      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. 从页面左上角选择 Create Stack (创建堆栈) (



)

以保存更改并创建启用这些更改的堆栈。

4. 保存更改后，您将被重定向到 Create stack (创建堆栈) 页面。
5. 选择下一步以继续。

4. 指定堆栈详细信息：为您的模板输入堆栈名称和参数。参数在模板中定义，并允许您在创建或更新堆栈时输入自定义值。

   - 在 Stack name (堆栈名称) 下，输入堆栈的名称或接受提供的名称。堆栈名称可以包含字母（A–Z 和 a–z）、数字（0–9）和破折号（—）。

- 在 Parameters (参数) 下，输入以下详细信息：
  - DBClusterName— 输入您的 Amazon DocumentDB 集群的名称或接受提供的名称。

    集群命名约束：
    - 长度为 [1-63] 个字母、数字或连字符。
    - 第一个字符必须是字母。
    - 不能以连字符结尾或包含两个连续的连字符。
    - 每个区域的 Amazon RDS、Neptune 和 Amazon DocumentDB 中的所有集群都必须是唯一 AWS 账户的。
  - DBInstanceClass— 从下拉列表中，为您的亚马逊文档数据库集群选择实例类。
  - DBInstanceName— 输入您的 Amazon DocumentDB 实例的名称或接受提供的名称。

    实例命名约束：
    - 长度为 [1-63] 个字母、数字或连字符。
    - 第一个字符必须是字母。
    - 不能以连字符结尾或包含两个连续的连字符。
    - 每个区域的 Amazon RDS、Neptune 和 Amazon DocumentDB 中的所有实例都必须是唯一 AWS 账户的。
  - MasterPassword— 数据库管理员帐户密码。
  - MasterUser— 数据库管理员帐户用户名。 MasterUser 必须以字母开头，并且只能包含字母数字字符。

  选择 Next (下一步) 以保存您的更改并继续。

5. Configure stack options (配置堆栈选项)：配置您堆栈的标签、权限和其他选项。
   - Tags (标签)：指定要应用于您堆栈中资源的标签（键值）对。您可以为每个堆栈添加最多 50 个唯一标签。
   - Permissions (权限)：可选。选择 IAM 角色以明确定义 CloudFormation 如何在堆栈中创建、修改或删除资源。如果您不选择角色，则会根据您的用户凭据 CloudFormation 使用权限。在指定服务角色之前，请确保您具有传递该角色的权限 (`iam:PassRole`)。`iam:PassRole` 权限指定您可以使用哪些角色。

> **Note**
>
> 指定服务角色时，CloudFormation 始终使用该角色执行在该堆栈上执行的所有操作。拥有对此堆栈执行操作的权限的其他用户将可以使用该角色，即使他们无权传递该角色也是如此。如果该角色包含用户不应具有的权限，则您可能无意中提升了用户的权限。确保该角色授予最低权限。

- Advanced options (高级选项)：您可以设置以下高级选项：
  - Stack policy (堆栈策略)：可选。定义在堆栈更新期间要防止意外更新的资源。默认情况下，堆栈更新期间所有资源都可更新。

    您可以直接以 JSON 形式输入堆栈策略，也可以上传包含堆栈策略的 JSON 文件。有关更多信息，请参阅防止更新堆栈资源。
  - Rollback configuration (回滚配置)：可选。指定要在创建和更新堆栈时监控的 CloudWatch 日志警报。CloudFormation 如果操作突破了警报阈值，则将其回 CloudFormation 滚。
  - Notification options (通知选项)：可选。指定简单通知系统 (SNS) 的主题。
  - Stack creation options (堆栈创建选项)：可选。可以指定以下选项：
    - Rollback on failure (失败时回滚)：如果堆栈创建失败，堆栈是否应回滚。
    - Timeout (超时)：堆栈创建超时之前的分钟数。
    - Termination protection (终止保护)：防止意外删除堆栈。

> **Note**
>
> CloudFormation 终止保护不同于 Amazon DocumentDB 的删除保护概念。有关更多信息，请参阅 终止保护和删除保护。

选择下一步以继续。

6. Review <stack-name> (审核 <stack-name>)：审核您的堆栈模板、详细信息和配置选项。您还可以在页面底部打开 quick-create link (快速创建链接)，以创建与此页面具有相同基本配置的堆栈。
   - 选择 Create (创建) 以创建堆栈。
   - 或者，您也可以选择 Create change set (创建更改集)。更改集是对在创建堆栈之前如何配置此堆栈的预览。这允许您在执行更改集之前检查各种配置。

访问 Amazon DocumentDB 集群

CloudFormation 堆栈完成后，您可以使用 Amazon EC2 实例连接到您的 Amazon DocumentDB 集群。有关使用 SSH 连接到 Amazon EC2 实例的信息，请参阅《Amazon EC2 用户指南》中的连接到 Linux 实例。

在连接之后，请参阅以下各节，其中包含有关使用 Amazon DocumentDB 的信息。

- 步骤 3：插入和查询数据
- 管理 Amazon DocumentDB 资源
- 监控 Amazon DocumentDB

终止保护和删除保护

启用删除保护和终止保护是 Amazon DocumentDB 的最佳实践。 CloudFormation 终止保护功能与 Amazon DocumentDB 删除保护功能截然不同。

- 终止保护-您可以通过为堆栈启用终止保护来防止 CloudFormation 堆栈被意外删除。如果用户尝试删除已为其启用终止保护的堆栈，则删除操作会失败，并且堆栈将保持不变。使用创建堆栈时，默认情况下终止保护处于禁用状态 CloudFormation。可在创建堆栈时对其启用终止保护。有关更多信息，请参阅设置 CloudFormation 堆栈选项。
- 删除保护：Amazon DocumentDB 还提供对集群启用删除保护的能力。如果用户试图删除启用了删除保护的 Amazon DocumentDB 集群，则删除失败，集群保持不变。启用删除保护后，可防止意外删除 Amazon DocumentDB AWS 管理控制台 AWS CLI、和。 CloudFormation有关对 Amazon DocumentDB 集群启用和禁用删除保护的更过信息，请参阅 删除保护。

Amazon DocumentDB 与 MongoDB 兼容性

亚马逊 DocumentDB 支持兼容 MongoDB，包括 MongoDB 4.0、mongoDB 5.0 和 MongoDB 8.0。MongoDB 兼容性意味着，您目前配合 MongoDB 数据库使用的绝大多数应用程序、驱动程序和工具都可以配合 Amazon DocumentDB 一起使用，变化甚微或无变化。本节说明了您需要了解的关于 Amazon DocumentDB 与 MongoDB 兼容性的一切，包括新功能和新特征、入门、迁移路径和功能差异。

主题

- 兼容 MongoDB 8.0
- MongoDB 5.0 兼容性
- 与 MongoDB 4.0 的兼容性

兼容 MongoDB 8.0

主题

- 亚马逊 DocumentDB 8.0 的新增功能
- 开始使用亚马逊 DocumentDB 8.0
- 升级或迁移到亚马逊 DocumentDB 5.0 或 8.0
- 功能差异

亚马逊 DocumentDB 8.0 的新增功能

Amazon DocumentDB 8.0 将查询性能提高多达 7 倍，压缩率提高多达 5 倍，使您能够以更低的成本构建高性能应用程序。亚马逊 DocumentDB 8.0 与 MongoDB 8.0 完全兼容。以下摘要介绍了亚马逊 DocumentDB 8.0 中引入的一些主要功能。要查看新功能的完整列表，请参阅 发布说明。

- 通过添加对 MongoDB 8.0 API 驱动程序的支持，提供与 MongoDB 8.0 的兼容性。还支持使用 MongoDB API 版本 6.0 和 7.0 构建的应用程序。
- Amazon DocumentDB 8.0 中的新查询规划器将性能改进扩展到聚合阶段运算符，同时支持聚合管道优化和不同的命令。
- 亚马逊 DocumentDB 8.0 支持排序规则和视图。

- 提供 6 个新的聚合阶段：$replaceWith、$VectorSearch、$merge、$set、$unset、$unset、$bucket 和 3 个新的聚合运算符 $pow、$rand、$dateTrunc。
- 新版本的文本索引：Amazon DocumentDB 8.0 中的文本索引 v2 引入了额外的标记，增强了文本搜索功能。
- 通过并行向量索引构建，Amazon DocumentDB 8.0 最多可将索引构建时间缩短 30 倍。

开始使用亚马逊 DocumentDB 8.0

要开始使用 Amazon DocumentDB 8.0，请参阅《入门指南》。您可以使用或 AWS 软件开发工具包创建新的 Amazon DocumentDB 8.0 集群， AWS 管理控制台 AWS CLI或者。 CloudFormation连接到 Amazon DocumentDB 时，要求您使用与 MongoDB 5.0 或更高版本兼容的 MongoDB 驱动程序或实用工具。

> **Note**
>
> 使用 AWS SDK AWS CLI、或时 CloudFormation，引擎版本将默认为 5.0.0。要创建新的 Amazon DocumentDB 8.0 集群、创建新的亚马逊 DocumentDB 4.0 集群或`engineVersion = 4.0.0`创建新的 Amazon DocumentDB 3.6 集群`engineVersion = 3.6.0`，您必须明确指定参数`engineVersion = 8.0.0`。对于给定的 Amazon DocumentDB 集群，您可以使用调用`describe-db-clusters`或使用 Amazon DocumentDB 管理控制台来查看特定集群的引擎版本号来确定集群版本。 AWS CLI

Amazon DocumentDB 5.0 和 8.0 支持 Amazon EC2 `r8g` Graviton4 处理器和 Graviton2 处理器，例如您的集群的`t4.medium`实例类型，`r6g`并且适用于所有支持的区域（参见）。实例有关定价的更多信息，请参阅 Amazon DocumentDB（兼容 MongoDB）定价。

升级或迁移到亚马逊 DocumentDB 5.0 或 8.0

你可以使用、、和之类的或实用程序从 MongoDB 3.6 或 MongoDB 4.0 迁移到亚马逊 DocumentDB 5.0 或 Amazon DocumentDB 8.0。AWS DMS`mongodump``mongorestore``mongoimport``mongoexport`有关如何迁移的说明，请参阅 使用升级您的亚马逊文档数据库集群 AWS Database Migration Service。

功能差异

亚马逊 DocumentDB 5.0 和 8.0 之间的功能区别

随着亚马逊 DocumentDB 8.0 的发布，亚马逊 DocumentDB 5.0 和亚马逊 DocumentDB 8.0 之间存在功能差异：

- Planner v1 是亚马逊 DocumentDB 5.0 中的默认查询计划器，而性能更高的 Planner v3 是亚马逊 DocumentDB 8.0 中的默认查询计划器。
- 亚马逊 DocumentDB 8.0 中的新功能，包括视图、排序规则和 $merge 等运算符，仅与 Planner v3 兼容。
- 在 Amazon DocumentDB 8.0 中，压缩功能默认处于开启状态，并设置为使用 Zstandard 算法。此外，在亚马逊 DocumentDB 8.0 中，“启用” 不再是一个有效的选择；你可以从 Zstd、LZ4 中选择，也可以不选择。
- 亚马逊 DocumentDB 8.0 仅支持 TLS 1.2 及更高版本的数据库连接。

> **Important**
>
> 亚马逊 DocumentDB 8.0 需要 TLS 1.2 或更高版本。不再支持 TLS 1.0 和 TLS 1.1。

MongoDB 5.0 兼容性

主题

- Amazon DocumentDB 5.0 中有什么新内容
- 亚马逊 DocumentDB 5.0.1 的新增功能
- Amazon DocumentDB 5.0 入门
- 升级或迁移到亚马逊 DocumentDB 5.0
- 功能差异

Amazon DocumentDB 5.0 中有什么新内容

Amazon DocumentDB 5.0 引入了新特征和新功能，其中包括存储限值和客户端字段级加密。以下摘要介绍了Amazon DocumentDB 5.0 中引入的一些主要特征。要查看新功能的完整列表，请参阅 发布说明。

- 对于所有基于实例的 Amazon DocumentDB 集群和基于分片的弹性集群，将存储限值升高到 128TiB。
- 引入 Amazon DocumentDB 5.0 引擎（版本 3.0.775）
  - 支持 MongoDB 5.0 API 驱动程序
  - 支持 Client-side 字段级加密 (FLE)。现在，在写入数据至 Amazon DocumentDB 集群之前，您可以在客户端对字段加密。有关更多信息，请参阅Client-side 字段级加密。
  - 新聚合操作符：`$dateAdd`, `$dateSubtract`
  - 支持带 `$elemMatch` 操作符的索引。因此，具有 `$elemMatch` 的查询将导致索引扫描。

Amazon DocumentDB 并不支持 MongoDB 5.0 的所有特征。构建 Amazon DocumentDB 5.0 时，我们从客户最迫切要求我们构建的特征和功能逆向工作。我们将根据客户要求我们构建的内容，持续添加其他 MongoDB 5.0 功能。有关支持的 API 的最新列表，请参阅Amazon DocumentDB 中支持的 MongoDB API、操作和数据类型。

亚马逊 DocumentDB 5.0.1 的新增功能

亚马逊 DocumentDB 5.0.1 是亚马逊 DocumentDB 5.0 主版本的第一个次要版本。次要版本包括新功能、安全修复和错误修复等改进。您可以使用次要版本升级从 5.0.0 升级到 5.0.1。有关更多信息，请参阅 亚马逊 DocumentDB 次要版本升级。

- 使用新的运算符（`$rand`、、`$pow$dateToParts`、`$dateFromParts`）增强了聚合功能。
- 用于监控实例的活动连接指标。这些限制可以在我们的 Amazon DocumentDB 配额和限制页面上找到。
- CRUD 操作的精细命令级性能指标（`find`、、、、、、`insert`、、`findAndModify`、`update`、`delete`、、`aggregate`、`count`、`disti`公开和`MaxConcurrent`维度 `AvgDurationP100Duration`，以改善操作监控。 CloudWatch
- 亚马逊 DocumentDB 5.0.1 仅支持 TLS 1.2 及更高版本的数据库连接。
- 在垃圾收集过程的整体运行状况中添加了`gcRuntimeStatsdocumentFragmentStats`、`storageSegmentBase`和分解了收集级别的指标，`collstats`以便`storageSegmentExtended`在垃圾收集过程的整体运行状况中显示更多信息。
- 通过使用全局生存只读功能消除辅助集群重启，从而提高了全球集群的跨区域弹性。

Amazon DocumentDB 5.0 入门

要开始使用亚马逊 DocumentDB 5.0，请参阅《入门指南》。您可以使用或软件开发工具包、 AWS 管理控制台 AWS CLI或创建新的 Amazon Document AWS DB 5.0 集群。 CloudFormation连接到 Amazon DocumentDB 时，要求您使用与 MongoDB 5.0 或更高版本兼容的 MongoDB 驱动程序或实用工具。

> **Note**
>
> 使用 AWS SDK AWS CLI、或时 CloudFormation，引擎版本将默认为 5.0.0。您必须明确指定参数 `engineVersion = 4.0.0` 以创建新的 Amazon DocumentDB 4.0 集群 或参数 `engineVersion = 3.6.0` 以创建新的 Amazon DocumentDB 3.6 集群。对于给定的 Amazon DocumentDB 集群，您可以使用调用`describe-db-clusters`或使用 Amazon DocumentDB 管理控制台来查看特定集群的引擎版本号来确定集群版本。 AWS CLI

Amazon DocumentDB 5.0 和 8.0 支持 Amazon EC2 `r8g` Graviton4 处理器和 Graviton2 处理器，例如您的集群的`t4.medium`实例类型，`r6g`并且适用于所有支持的区域（参见）。实例有关定价的更多信息，请参阅 Amazon DocumentDB（兼容 MongoDB）定价。

升级或迁移到亚马逊 DocumentDB 5.0

您可以使用 AWS DMS 或实用工具如 `mongodump`、`mongorestore`、`mongoimport` 和 `mongoexport` 从 MongoDB 3.6 或 MongoDB 4.0 迁移到 Amazon DocumentDB 5.0。有关如何迁移的说明，请参阅 使用升级您的亚马逊文档数据库集群 AWS Database Migration Service。

功能差异

Amazon DocumentDB 4.0 与 5.0 之间的功能差异

随着 Amazon DocumentDB 5.0 发布，Amazon DocumentDB 4.0 与 Amazon DocumentDB 5.0 之间存在功能差异：

- 备份内置角色现在支持 `serverStatus`。操作：具有备份角色的开发人员和应用程序可以收集有关 Amazon DocumentDB 集群状态的统计量。
- `SecondaryDelaySecs` 字段替换 `replSetGetConfig`输出中的 `slaveDelay`。
- hello 命令替换：`isMaster`hello返回描述 Amazon DocumentDB 集群角色的文档。

- Amazon DocumentDB 5.0 现在支持在第一个嵌套级别采用 $elemMatch 操作符的索引扫描。如果仅查询筛选器具有一个级别的 $elemMatch 筛选器，则支持索引扫描，但如果包含一个嵌套 $elemMatch 查询，则不支持索引扫描。

  例如，在 Amazon DocumentDB 5.0 中，如果您在嵌套级别中包含 $elemMatch 操作符，将不会像在 Amazon DocumentDB 4.0 中那样返回数值：

```
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 4.0 中的“$”投影返回带所有字段的所有文档。在 Amazon DocumentDB 5.0 中，带“$”投影的 find 命令返回匹配查询参数的文档，这种参数仅含有匹配“$”投影的字段。
- 在 Amazon DocumentDB 5.0 中，带 $regex 和 $options 查询参数的 find 命令返回错误：“无法在 $regex 和 $options 同时设置选项”。
- 在 Amazon DocumentDB 5.0 中，目前 $indexOfCP 在以下情况时返回“-1”：
  - 字符串表达式中未找到子字符串，或
  - 起点是一个大于终点的数字，或
  - 起点是大于字符串字节长度的数字。

- 在 Amazon DocumentDB 4.0 中，当起点位置是大于终点或大于字符串字节长度的数字，`$indexOfCP` 返回 “0”。
- 在 Amazon DocumentDB 5.0 中，在 `_id fields` 中的投影操作，例如 `{"_id.nestedField" : 1}`，返回仅包含所投影字段的文档。而在 Amazon DocumentDB 4.0 中，嵌套字段投影命令不筛选出任何文档。

与 MongoDB 4.0 的兼容性

主题

- Amazon DocumentDB 4.0 特征
- Amazon DocumentDB 4.0 入门
- 升级或迁移到 Amazon DocumentDB 4.0
- 功能差异

Amazon DocumentDB 4.0 特征

Amazon DocumentDB 4.0 引入了许多新特征和新功能，包括 ACID 事务和对变更流的改进。以下摘要显示了 Amazon DocumentDB 4.0 中引入的一些主要特征。要查看功能的完整列表，请参阅 发布说明。

- ACID 事务：Amazon DocumentDB 现在支持跨多个文档、报表、集合和数据库执行事务的能力。事务通过使您能够在 Amazon DocumentDB 集群内部跨一个或多个文档执行原子操作、一致操作、隔离操作和持久操作（ACID），简化应用程序开发。有关更多信息，请参阅 Amazon DocumentDB 中的事务。
- 变更流：您现在有能力在集群层面（`client.watch()` 或 `mongo.watch()`）和在数据库（`db.watch()`）打开变更流，可以指定 `startAtOperationTime` 来打开变更流游标，并且最后，您现在可以将变更流保留期延长至 7 天（之前为 24 小时）。有关更多信息，请参阅 将变更流与 Amazon DocumentDB 结合使用。
- AWS Database Migration Service(AWS DMS)：您现在可以使用将 MongoDB 4.0 AWS DMS 工作负载迁移到亚马逊 DocumentDB。 AWS DMS 现在支持 MongoDB 4.0 源、亚马逊 DocumentDB 4.0 目标和亚马逊 DocumentDB 3.6 源，用于在亚马逊 DocumentDB 3.6 和 4.0 之间执行升级。有关更多信息，请参阅 AWS DMS 文档。
- 性能和索引化：现在，您可以利用带 `$lookup` 的索引，查找查询借助含一个字段或一个字段且 `_id` 字段可以从索引中直接提供而无需从集合（涵盖的查询）中读取，借助 `findAndModify` 进行

`hint()` 的能力、对 `$addToSet` 的性能优化和缩小索引总体大小的改进。有关更多信息，请参阅发布说明。

- 操作符：Amazon DocumentDB 4.0 现在支持许多新的聚合操作符：`$ifNull`、`$replaceRoot`、`$setIsSubset`、`$setIntersection`、`$setUnion`、`$setEquals`。您可以在 Amazon DocumentDB 中支持的 MongoDB API、操作和数据类型 查看我们支持的所有 MongoDB API、操作和数据类型。
- 基于角色的访问控制（RBAC）：同时使用命令 `ListCollection` 和 `ListDatabase` ，您现在可以选择使用 `authorizedCollections` 和 `authorizedDatabases` 参数允许用户列出他们有权访问的集合和数据库，而无需分别使用 `listCollections` 和 `listDatabase` 角色。您还可以杀死自己的游标，而无需使用 `KillCursor` 角色。

Amazon DocumentDB 并不支持 MongoDB 4.0 的所有特征。构建 Amazon DocumentDB 4.0 时，我们从客户最迫切要求我们构建的特征和功能逆向工作。我们将根据客户要求我们构建的内容，持续添加其他 MongoDB 4.0 功能。例如，Amazon DocumentDB 4.0 目前不支持 MongoDB 4.0 中引入的类型转换操作符或字符串操作符。有关支持的 API 的最新列表，请参阅Amazon DocumentDB 中支持的 MongoDB API、操作和数据类型。

Amazon DocumentDB 4.0 入门

要开始使用亚马逊 DocumentDB 4.0，请参阅《入门指南》。您可以使用或软件开发工具包创建新的 Amazon Document AWS DB 4.0 集群 AWS CLI， AWS 管理控制台 或。 CloudFormation连接到 Amazon DocumentDB 时，要求您使用与 MongoDB 4.0 或更高版本兼容的 MongoDB 驱动程序或实用工具。

> **Note**
>
> 使用 AWS SDK AWS CLI、或时 CloudFormation，引擎版本将默认为 5.0.0。您必须明确指定参数 `engineVersion = 4.0.0` 以创建新的 Amazon DocumentDB 4.0 集群 或参数 `engineVersion = 3.6.0` 以创建新的 Amazon DocumentDB 3.6 集群。对于给定的 Amazon DocumentDB 集群，您可以使用调用`describe-db-clusters`或使用 Amazon DocumentDB 管理控制台来查看特定集群的引擎版本号来确定集群版本。 AWS CLI

Amazon DocumentDB 4.0 支持您集群的 `r5`、`r6g`、`t3.medium` 和 `t4g.medium` 实例类型，并且可用于所有受支持区域。没有使用 Amazon DocumentDB 4.0 的额外成本。有关定价的更多信息，请参阅 Amazon DocumentDB（与 MongoDB 兼容）定价。

升级或迁移到 Amazon DocumentDB 4.0

您可以使用 AWS DMS 或实用工具如 mongodump、mongorestore、mongoimport 和 mongoexport 从 MongoDB 3.6 或 MongoDB 4.0 迁移到 Amazon DocumentDB 4.0。类似地，您可以使用相同工具从 Amazon DocumentDB 3.6 升级到 Amazon DocumentDB 4.0。有关如何迁移的说明，请参阅 使用升级您的亚马逊文档数据库集群 AWS Database Migration Service。

功能差异

Amazon DocumentDB 3.6 与 4.0 之间的功能差异

随着 Amazon DocumentDB 4.0 发布，Amazon DocumentDB 3.6 和Amazon DocumentDB 4.0 之间存在功能差异：

- 嵌套文档的投影：Amazon DocumentDB 3.6 在应用投影时考虑嵌套文档中的第一个字段。但是，Amazon DocumentDB 4.0 将解析子文档并将投影也应用于每个子文档。例如：如果投影是 `"a.b.c": 1`，则行为在两个版本中相同。但是，如果投影是 `{a:{b:{c:1}}}`，则 Amazon DocumentDB 3.6 将仅将该投影应用于“a”，而非“b”或“c”。
- 对 **`minKey`**、**`maxKey`**的行为：在 Amazon DocumentDB 4.0 中，对 `{x:{$gt:MaxKey}}` 的行为不返回任何内容，对 `{x:{$lt:MaxKey}}` 的行为返回所有内容。
- 文档比较差异：比较子文档（例如，`{"_id" :1, "a" :{"b":1}}` 中 b）中不同类型（double、int、 long）的数值现在跨不同数字数据类型并为文档的每个级别提供一致输出。

Amazon DocumentDB 4.0 与 MongoDB 4.0 之间的功能差异

以下是 Amazon DocumentDB 4.0 与 MongoDB 4.0 之间的功能差异。

- 在路径中借助空键查找：当集合包含数组内部带空键的文档（例如`{"x" : [ { "" : 10 }, { "b" : 20 } ]}`），且查询中所用的密钥以空字符串结尾（例如`x.`）时，则 Amazon DocumentDB 将返回该文档，因为其遍历该数组中的所有文档，而 MongoDB 将不返回该文档。
- 路径中的 **`$setOnInsert`** 同 **`$`** ：在 Amazon DocumentDB 中字段操作符 `$setOnInsert` 将与 `$` 在路径中共同发挥作用，这也与 MongoDB 4.0 一致。

Amazon DocumentDB 中的事务

亚马逊 DocumentDB（兼容 MongoDB）支持亚马逊 DocumentDB 4.0 及更高版本中的交易。您可以跨多个文档、报表、集合和数据库执行事务。事务通过使您能够在 Amazon DocumentDB 集群内部跨一个或多个文档执行原子操作、一致操作、隔离操作和持久操作 (ACID)，简化应用程序开发。常见的事务用例包括财务处理、履行和管理订单以及开发多人游戏。

启用事务无需其他成本。您只需为您消耗的作为事务组成部分的读写IO付费。

主题

- 要求
- 最佳实践
- 限制
- 监控和诊断
- 事务隔离级别
- 使用案例
- 支持的 命令
- 不支持的功能
- 会话
- 事务错误

要求

要使用事务特征，您需要满足以下要求：

- 您必须使用亚马逊 DocumentDB 4.0 或更高版本。
- 您必须使用与 MongoDB 4.0 或更高版本兼容的驱动程序。

最佳实践

以下是一些最佳实践，帮助您通过 Amazon DocumentDB 充分利用事务。

- 务必在事务完成后提交或中止事务。听任事务处于未完成状态占用数据库资源并可能导致写入冲突。

- 建议将事务保持在所需的最少命令数。如果您的事务包含多个可以划分成多个更小事务的语句，建议划分以降低超时可能性。始终以创建短事务而非长时间运行读取为目标。

限制

- Amazon DocumentDB 不支持事务内部的游标。
- Amazon DocumentDB 无法在交易中创建新的馆藏，也不能 query/update 针对不存在的馆藏。
- Document-level 写锁的超时时间为 1 分钟，用户无法对其进行配置。
- Amazon DocumentDB 不支持可重试写入、可重试提交和可重试中止命令。如果您使用旧版 mongo Shell（而非 mongosh），不要在任何代码字符串中包含 `retryWrites=false` 命令。默认情况下，禁用可重试写入。包含 `retryWrites=false` 可能导致正常读取命令失败。
- 每个 Amazon DocumentDB 实例对实例上同时打开的并发事务数目都有上限限值。有关限制，请参阅实例限制。
- 对于给定的事务，事务日志大小必须小于 32MB。
- Amazon DocumentDB 确实支持事务内部的 `count()`，但并非所有驱动程序都支持此功能。一种替代方法是使用 `countDocuments()` API，它将计数查询转换为客户端上的聚合查询。
- 事务有一分钟执行限值，会话有 30 分钟超时。如果事务超时，将被中止，并且会话内部对现有事务发出的任何后续命令都将产生以下错误：

```
WriteCommandError({
"ok" : 0,
"operationTime" : Timestamp(1603491424, 627726),
"code" : 251,
"errmsg" : "Given transaction number 0 does not match any in-progress transactions."
}
```

监控和诊断

借助 Amazon DocumentDB 4.0 中对交易的支持，我们添加了其他 CloudWatch 指标来帮助您监控交易。

新 CloudWatch 指标

- `DatabaseTransactions`：在一分钟时段进行的开放事务的数目。
- `DatabaseTransactionsAborted`：在一分钟时段进行的已中止事务的数目。

- DatabaseTransactionsMax：一分钟时段内开放事务的最大数目。
- TransactionsAborted：一分钟时段内对一个实例中止的事务数目。
- TransactionsCommitted：一分钟时段内对一个实例提交的事务数目。
- TransactionsOpen：在一分钟时段对一个实例开放的事务数目。
- TransactionsOpenMax：一分钟时段内对一个实例开放事务的最大数目。
- TransactionsStarted：一分钟时段内对一个实例启动的事务数目。

> **Note**
>
> 有关 Amazon DocumentDB 的更多 CloudWatch 指标，请访问。使用以下方式监控亚马逊 DocumentDB CloudWatch

另外，将新字段同时添加至 currentOp lsid、transactionThreadId ，“idle transaction”的新状态与 serverStatus 事务：currentActive、currentInactive、currentOpen、totalAborted、totalCommitted 和 totalStarted。

事务隔离级别

启动事务时，您可以同时指定readConcern和，writeConcern如以下示例所示：

```
mySession.startTransaction({readConcern: {level: 'snapshot'}, writeConcern: {w: 'majority'}});
```

对于readConcern，Amazon DocumentDB 默认支持快照隔离。如果指定了“本地”、“可用”或“多数”的 readConcern，则 Amazon DocumentDB 会将该readConcern级别升级成快照。Amazon DocumentDB 不支持可线性化 readConcern，而指定这样的读取问题会导致错误。

对于 writeConcern，Amazon DocumentDB 默认支持多数，当数据的四个副本维持在三个可用区(AZ) 时，则实现写入仲裁。如果指定更低的 writeConcern，则 Amazon DocumentDB 会将 writeConcern 升级成多数。此外，所有 Amazon DocumentDB 写入都被记录，并且日志功能无法禁用。

使用案例

本节将介绍两个事务用例：多语句和多集合。

Multi-Statement 交易

Amazon DocumentDB 事务有多语句性，这意味着您可以通过显式提交或回滚写入跨多个语句的事务。您可以将 `insert`、`update`、`delete` 和`findAndModify`操作分组为单一原子操作。

一个多语句事务的常见用例是借-贷事务。例如：您欠朋友购衣钱。因此，您需要从您的账户借记（取出）500美元，并贷记（存入） 500美元至您朋友的账户。要执行该操作，您需要在单一事务内部同时执行借记操作和贷记操作，以确保原子性。这样做防止出现从您的账户借记了 500美元但未存入您朋友账户的情况。以下是这个用例的样子：

```
// *** 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();
```

Multi-collection 交易

我们的事务也有多集合性，这意味着它们可用于执行单一事务内部且跨多个集合的多项操作。这提供一致的数据视图并维持数据完整性。将命令作为单一 <> 提交时，这些事务是全有或全无执行：既要么全部成功，要么全部失败。

以下是多集合事务的一个示例，其使用来自多语句事务示例的相同场景和数据。

```
// *** 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
```

回调 API 的事务 API 示例

回调 API 仅可用于 4.2 及以上驱动程序。

Javascript

以下代码演示如何配合 Javascript 使用 Amazon DocumentDB 事务 API。

```
// *** 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

以下代码演示了如何使用亚马逊 DocumentDB 交易 API。 Node.js

```
// 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#

以下代码演示如何配合 C# 使用 Amazon DocumentDB 事务 API。

```
// 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

以下代码演示如何配合 Ruby 使用 Amazon DocumentDB 事务 API。

```
// 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

以下代码演示如何配合 Go 使用 Amazon DocumentDB 事务 API。

```
// 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

以下代码演示如何配合 Java 使用 Amazon DocumentDB 事务 API。

```
// 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

以下代码演示如何配合 C 使用 Amazon DocumentDB 事务 API。

```
// 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

以下代码演示如何配合 Python 使用 Amazon DocumentDB 事务 API。

```
// 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
```

核心 API 的事务 API 示例

Javascript

以下代码演示如何配合 Javascript 使用 Amazon DocumentDB 事务 API。

```
// *** 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#

以下代码演示如何配合 C# 使用 Amazon DocumentDB 事务 API。

```
// 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

以下代码演示如何配合 Ruby 使用 Amazon DocumentDB 事务 API。

```
# 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

以下代码演示如何配合 Go 使用 Amazon DocumentDB 事务 API。

```
// 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

以下代码演示如何配合 Java 使用 Amazon DocumentDB 事务 API。

```
// 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

以下代码演示如何配合 C 使用 Amazon DocumentDB 事务 API。

```
// 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

以下代码演示如何配合 Scala 使用 Amazon DocumentDB 事务 API。

```
// 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

以下代码演示如何配合 Python 使用 Amazon DocumentDB 事务 API。

```
// 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
```

支持的 命令

命令	支持
abortTransaction	支持
commitTransaction	是
endSessions	是
killSession	是
killAllSession	是
killAllSessionsByPattern	否
refreshSessions	否
startSession	是

不支持的功能

方法	阶段或命令
db.collection.aggregate()	$collStats $currentOp $indexStats $listSessions $out
db.collection.count() db.collection.countDocuments()	$where $near $nearSphere

方法	阶段或命令
db.collection.insert()	如果 insert 不针对现有集合运行，则得不到支持。如果这种方法以预存在集合为目标，则得到支持。

会话

MongoDB 会话是用于支持可重试写入、因果一致性、事务及管理跨数据库操作的框架。创建会话时，逻辑会话标识符 (lsid) 由客户端生成，并且向服务器发送命令时，用于标记该会话内部的所有操作。

Amazon DocumentDB 支持使用会话启用事务，但不支持因果一致性或可重试写入。

在 Amazon DocumentDB 内部使用事务时，事务将使用 session.startTransaction() API 从会话内部启动，并且一个会话一次支持单一事务。类似地，使用提交 (session.commitTransaction()) 或终止 (session.abortTransaction()) API 完成事务。

因果一致性

因果一致性保证，在单个客户端会话中，客户端将观察到先写后读的一致性、单原子性 reads/writes，并且写入将在读取之后进行，这些保证适用于集群中的所有实例，而不仅仅是主实例。Amazon DocumentDB 不支持因果一致性，以下语句将导致错误。

```
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()
```

您可以在会话内部禁用因果一致性。请注意，这样做将使您能够使用会话框架，但不能为读取提供因果一致性保证。使用 Amazon DocumentDB 时，从主实例读取将为先写后读一致，而从副本实例读取将为最终一致。事务是利用会话的主要用例。

```
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 }
```

可重试写入

可重试写入是一种功能，当发生网络错误或客户端无法找到主实例时，客户端将尝试重试写入操作一次。Amazon DocumentDB 不支持且必须禁用可重试写入。您可以用连接字符串中的命令 (`retryWrites=false`) 禁用它。

> **Note**
>
> 如果您使用旧版 mongo Shell（而非 mongosh），不要在任何代码字符串中包含 `retryWrites=false` 命令。默认情况下，禁用可重试写入。包含 `retryWrites=false` 可能导致正常读取命令失败。

事务错误

使用事务时，存在可能导致出现错误的场景，这种错误指出事务编号不匹配任何进程中的事务。

该错误可能在至少两种不同场景下生成：

- 在一分钟的事务超时后。
- 实例（因为打补丁、崩溃恢复等）重启后，即使事务成功提交，也有可能收到此种错误。在实例重启期间，数据库无法区分成功完成的事务与中止事务之间的差异。换句话说，事务完成状态不明确。

处理此错误的最佳方法是使事务更新具有等性，例如，使用$set变异器代替操作。 increment/decrement 请参阅下面的：

```
{ "ok" : 0,
"operationTime" : Timestamp(1603938167, 1),
"code" : 251,
"errmsg" : "Given transaction number 1 does not match any in-progress transactions."
}
```

Amazon DocumentDB 的最佳实践

了解使用 Amazon DocumentDB（与 MongoDB 兼容）的最佳实践。随着新的最佳实践的确定，此节将不断更新。

主题

- 基本操作指导
- 实例大小调整
- 使用索引
- 安全最佳实践
- 成本优化
- 使用指标确定性能问题
- TTL 和时间序列工作负载
- 迁移
- 使用集群参数组
- 聚合管道查询
- 批量插入和批量更新

基本操作指导

以下是使用 Amazon DocumentDB 时每个人都应遵循的基本操作指导方针。Amazon DocumentDB 服务等级协议要求您遵循以下指导方针。

- 在两个 AWS 可用区部署由两个或更多 Amazon DocumentDB 实例组成的集群。对于生产工作负载，建议在三个可用区中部署包含三个或以上 Amazon DocumentDB 实例的集群。
- 在规定的服务限制内使用服务。有关更多信息，请参阅 Amazon DocumentDB 配额和限制。
- 监控您的内存、CPU、连接和存储使用情况。为了帮助您保持系统性能和可用性，请 CloudWatch 将 Amazon 设置为在使用模式发生变化或接近部署容量时通知您。
- 当接近容量限制时，可以向上扩展您的实例。您应为这些实例预配置足够的计算资源（即 RAM、CPU），以满足无法预测的应用程序需求增长。
- 设置您的备份保留期以与您的恢复点目标保持一致。
- 测试您的集群的失效转移，以了解对于您的使用案例而言，该过程需要多长时间。有关更多信息，请参阅 Amazon DocumentDB 失效转移。

- 使用集群端点（请参阅 Amazon DocumentDB 端点）以副本集模式（请参见 作为副本集连接到 Amazon DocumentDB）连接到 Amazon DocumentDB 集群，以尽可能减少失效转移对应用程序的影响。
- 选择一项驱动程序读取首选项设置，以便在满足应用程序的读取一致性要求的同时最大程度地提高读取扩展能力。`secondaryPreferred` 读取首选项将启用副本读取，并释放主实例以执行更多工作。有关更多信息，请参阅 读取首选项选项。
- 将您的应用程序设计为在出现网络和数据库错误时能够灵活应对。使用您的驱动程序的错误机制来区分临时错误和持久性错误。适当时使用指数回退机制重试临时错误。确保您的应用程序在实施重试逻辑时考虑数据一致性。
- 为所有生产集群或包含重要数据的任何集群启用集群删除保护。删除 Amazon DocumentDB 集群之前，请拍摄最终快照。如果您使用部署资源 CloudFormation，请启用终止保护。有关更多信息，请参阅 终止保护和删除保护。
- 在创建 Amazon DocumentDB 集群时，`--engine-version` 是一个默认为最新主引擎版本的可选参数。当前的默认引擎版本为 5.0.0（注意：亚马逊 DocumentDB 8.0 可用，但必须明确指定）。发布主引擎新版本时，`--engine-version` 的默认引擎版本将更新，以反映最近的主引擎版本。因此，对于生产工作负载，尤其是那些依赖脚本、自动化或 CloudFormation 模板的工作负载，我们建议您明确指定预期的主要版本。`--engine-version`

实例大小调整

选择 Amazon DocumentDB 中实例大小的最关键方面之一是缓存的 RAM 量。Amazon DocumentDB 为自身服务保留了三分之一的 RAM，这意味着只有三分之二的实例 RAM 可用于缓存。因此，为了发挥 Amazon DocumentDB 的最佳性能，最好的做法是选择具有足够 RAM 的实例类型，以便有足够的内存来容纳工作集（即数据和索引）。拥有适当大小的实例将有助于优化整体性能，并有可能最大限度地降低 I/O 成本。您可以使用第三方 Amazon DocumentDB 大小调整计算器估算特定工作负载的实例大小。

要确定您的应用程序的工作集是否适合内存，请监控集群中每个处于负载状态 CloudWatch 的实例的`BufferCacheHitRatio`使用 Amazon 的情况。

该`BufferCacheHitRatio` CloudWatch 指标衡量的是实例内存缓存提供的数据和索引的百分比（相对于存储量）。一般来说，`BufferCacheHitRatio` 的值应该尽可能高，因为从工作集内存读取数据比从存储卷读取数据更快、更具成本效益。虽然保证 `BufferCacheHitRatio` 尽可能接近 100% 是最理想的，但可实现的最佳值将取决于您的应用程序的访问模式和性能要求。为保证 `BufferCacheHitRatio` 尽可能高，建议为您集群中的实例配置足够 RAM，以便能够在内存中保存索引和工作数据集。

如果您的索引不在内存中，您就会看到一个较低的 BufferCacheHitRatio。持续从磁盘读取会产生额外 I/O的成本，而且并不比从内存读取好。如果您的 BufferCacheHitRatio 比率低于预期，请上调集群的实例大小，以提供更多 RAM，以便将工作集数据容纳在内存中。如果上调实例类大小会导致 BufferCacheHitRatio 大幅提高，就表明应用程序的工作集不在内存中。继续向上扩展，直至 BufferCacheHitRatio 在扩展操作后不再大幅上升。有关监控实例的指标的信息，请参阅 Amazon DocumentDB 指标。

根据您的工作负载和延迟要求，也许可以让您的应用程序在稳定状态使用期间具有较高 BufferCacheHitRatio 值，然后定期降低 BufferCacheHitRatio，因为分析实例需要扫描实例上运行的整个集合。定期降低 BufferCacheHitRatio 可能会很明显，因为后续查询需要将工作集数据从存储卷复制回缓冲区缓存，所以延迟时间较长。我们建议您首先在具有代表性生产工作负载的预生产环境中测试您的工作负载，以便了解性能特征和 **BufferCacheHitRatio**，然后再将工作负载部署到生产环境。

BufferCacheHitRatio 是特定于实例的指标，因此同一集群中不同实例可能具有不同的 BufferCacheHitRatio 值，具体取决于读取在主实例和副本实例之间的分配方式。如果运行工作负载无法处理因运行分析查询后重新填充工作集缓存而导致的定期延迟增加，应尝试将常规工作负载的缓冲缓存与分析查询的缓冲缓存隔离开。您可以通过将操作查询指向主实例，将分析查询指向仅指向副本实例，实现完全的 BufferCacheHitRatio 隔离。您还可以通过将分析查询定向到特定副本实例来实现部分隔离，但要知道有一定百分比的常规查询也将在该副本上运行，并且可能受到影响。

BufferCacheHitRatio 的值是否适当取决于您的使用案例和应用程序要求。此指标没有一个最佳值或最小值；只有您可以从成本和性能的角度决定是否可以接受 BufferCacheHitRatio 暂时较低的后果。

使用索引

建立索引

将数据导入 Amazon DocumentDB 时，最好在导入大型数据集之前先创建索引。您可以使用 Amazon DocumentDB 索引工具从正在运行的 MongoDB 实例或 mongodump 目录提取索引，然后在 Amazon DocumentDB 集群中创建这些索引。有关迁移的更多指导，请参阅迁移到 Amazon DocumentDB。

索引选择性

我们建议您将索引创建限制为其中的重复值数量低于集合中总文档数的 1% 的字段。例如，如果您的集合包含 100,000 个文档，则仅在相同值出现 1000 次或更少的字段上创建索引。

选择具有大量唯一值（即高基数）的索引可确保筛选操作返回少量文档，从而在索引扫描期间产生良好的性能。高基数索引的一个例子是一个唯一索引，它保证相等谓词最多返回单个文档。低基数的示例包括布尔字段上的索引和一周中某天的索引。由于性能差，数据库的查询优化程序不太可能选择低基数索引。同时，低基数索引继续消耗资源，例如磁盘空间和。 I/Os作为经验法则，应将索引定位于典型值频率为总集合大小的 1% 或更小的字段。

此外，建议仅在通常用作筛选条件的字段上创建索引，并定期查找未使用的索引。有关更多信息，请参阅 如何分析索引使用情况并识别未使用的索引？。

索引对数据写入的影响

虽然索引可以通过避免扫描集合中的每个文档来提高查询性能，但这种提高是有代价的。对于集合的每个索引，每次插入、更新或删除文档时，数据库都必须更新集合并将字段写入集合的每个索引。例如，如果某个集合有九个索引，则数据库必须执行十次写入操作，才能将操作通知给客户端。因此，每增加一个索引都会导致额外的写入延迟， I/O并增加总利用的存储空间。

所以应该适当调整集群实例的大小，以确保将所有工作集容纳在内存中。这就避免了从存储卷中持续读取索引页的需要，因为这会对性能产生负面影响并产生更高 I/O 的成本。有关更多信息，请参阅 实例大小调整。

为了获得最佳性能，请尽量减少集合中的索引数量，仅添加必要的索引来提高常用查询的性能。虽然工作负载会变化，但有一个很好的指导原则，就是将每个集合的索引数量保持在五个以内。

识别缺失的索引

我们建议您定期识别缺失的索引，这是一种很好的做法。有关更多信息，请参阅 如何识别缺失的索引？。

识别未使用的索引

我们建议您定期识别并删除未使用的索引，这是一种很好的做法。有关更多信息，请参阅 如何分析索引使用情况并识别未使用的索引？。

安全最佳实践

为了获得最佳安全实践，您必须使用 AWS Identity and Access Management (IAM) 账户来控制对亚马逊 DocumentDB API 操作的访问权限，尤其是创建、修改或删除亚马逊 DocumentDB 资源的操作。此类资源包括集群、安全组和参数组。此外，您还必须使用 IAM 来控制执行常见管理任务的操作，例如备份和还原集群。创建 IAM 角色时，请采用最小权限原则。

- 使用基于角色的访问控制强制执行最低权限。
- 为每个管理 Amazon DocumentDB 资源的人员分配个人 IAM 账户。请勿使用 AWS 账户 根用户来管理 Amazon DocumentDB 资源。为每个人（包括您自己）创建一个 IAM 用户。
- 授予每位 IAM 用户履行其职责所需的最小权限集。
- 使用 IAM 组有效地管理适用于多个用户的权限。有关 IAM 的更多信息，请参阅 IAM 用户指南。有关 IAM 最佳实践的信息，请参阅 IAM 最佳实践。
- 定期轮换 IAM 凭证。
- 将 S AWS ecrets Manager 配置为自动轮换 Amazon DocumentDB 的密钥。有关更多信息，请参阅 S AWS ecrets Manager 用户指南中的轮换您的 Sec rets Manager 密钥和轮换亚马逊 DocumentDB 的AWS 密钥。
- 授予每位 Amazon DocumentDB 用户履行其职责所需的最小权限集。有关更多信息，请参阅 使用访问控制 Role-Based 访问数据库。
- 使用传输层安全 (TLS) 对传输中的数据进行加密 AWS KMS ，并对静态数据进行加密。

成本优化

以下最佳实践可帮助您管理和最大程度地降低使用 Amazon DocumentDB 的成本。有关定价信息，请参阅 Amazon DocumentDB（与 MongoDB 兼容）定价和 Amazon DocumentDB（与 MongoDB 兼容）常见问题解答。

- 在阈值为该月预期账单的 50% 和 75% 时创建账单提醒。有关创建账单提醒的更多信息，请参阅创建账单提醒。
- Amazon DocumentDB 的架构将存储与计算分离，因此即使是单实例集群也具备高持久性。集群存储卷在三个可用区中从六个方向复制数据，提供极高的持久性，而不论集群中有多少个实例。典型的生产集群具有三个或更多实例，以提供高可用性。但是，您在不需要高可用性时可使用单个实例开发集群，以优化成本。
- 对于开发和测试场景，在不再需要时停止集群，并在恢复开发时启动集群。有关更多信息，请参阅停止和启动 Amazon DocumentDB 集群。
- 写入、读取和删除数据时，TTL 和变更流都会产 I/O生。如果您已启用这些功能，但未在应用程序中使用它们，则禁用这些功能有助于降低开销。

使用指标确定性能问题

主题

- 查看性能指标
- 设置 CloudWatch 闹铃
- 评估性能指标
- 使用指标评估亚马逊 DocumentDB 实例使用情况 CloudWatch
- 优化查询

要确定资源不足和其他常见瓶颈导致的性能问题，您可以监控可用于 Amazon DocumentDB 集群的指标。

查看性能指标

定期监控性能指标，以查看各种时间范围内的平均值、最大值和最小值。这可帮助您确定性能下降的时间。您还可以为特定的指标阈值设置 Amazon CloudWatch 警报，以便在达到这些阈值时收到提醒。

要排除性能问题，了解系统的基准性能十分重要。设置新集群并让它在典型工作负载下运行之后，您应按一些不同的间隔（例如，1 小时、24 小时、1 周、2 周）来捕获所有性能指标的平均值、最大值和最小值。这将使您能够了解运行状况。这有助于将操作的峰值时间与非峰值时间进行比较。您随后可以利用这些信息确定性能何时降到标准水平以下。

您可以使用 AWS 管理控制台 或查看性能指标 AWS CLI。有关更多信息，请参阅 查看 CloudWatch 数据。

设置 CloudWatch 闹铃

要设置 CloudWatch 警报，请参阅亚马逊 CloudWatch 用户指南中的使用亚马逊 CloudWatch 警报。

评估性能指标

一个实例具有多个不同类别的指标。确定可接受的值的方式取决于指标。

CPU

- CPU 利用率：使用的计算机处理容量的百分比。

内存

- 可用内存：实例上可用的 RAM 量。
- 交换区使用情况：实例使用的交换空间量（以兆字节为单位）。

Input/output 操作

- 读取 IOPS，写入 IOPS：每秒进行的磁盘读取或写入操作平均数。
- 读取延迟，写入延迟：读取或写入操作的平均时间（以毫秒为单位）。
- 读取吞吐量，写入吞吐量：每秒对磁盘读取或写入的平均兆字节数。
- 磁盘队列深度-等待写入磁盘或从磁盘读取的 I/O操作数。

网络流量

- 网络接收吞吐量，网络传输吞吐量：每秒出入实例的网络流量速率（以兆字节为单位）。

数据库连接

- 数据库连接：连接到实例的客户端会话数。

一般而言，性能指标的可接受值取决于您的基准性能以及应用程序执行的操作。应调查相对于基准性能的一致或趋势性变化。

以下是有关特定类型的指标的建议：

- 高 CPU 消耗：CPU 消耗值高可能是正常情况，只要它们符合您的应用程序目标（如吞吐量或并发度）并且符合预期。如果 CPU 消耗始终高于 80%，请考虑扩展您的实例。
- 高 RAM 消耗：如果 `FreeableMemory` 指标经常下降至总实例内存的 10% 以下，请考虑扩展您的实例。有关您的文档数据库实例正经历高内存压力时发生什么的更多信息，请参阅 Amazon DocumentDB 资源管理。
- 交换区使用情况：该指标应保持为零或接近零。如果交换区使用情况非常大，请考虑扩展您的实例。
- 网络流量：对于网络流量，请与系统管理员讨论，以了解域网络和互联网连接的预期吞吐量。如果吞吐量始终低于预期，则应调查网络流量。
- 数据库连接：如果发现用户连接数较高，同时实例性能下降并且响应时间延长，请考虑约束数据库连接。实例的最佳用户连接数因您的实例类所执行操作的复杂性而异。对于与性能指标有关的问题，提高性能的首要手段之一是优化最常使用和成本最高昂的查询，以了解这是否会减少对系统资源的压力。

如果您的查询经过调整但问题仍然存在，请考虑将您的 Amazon DocumentDB 实例类升级为具有更多与您遇到的问题相关的资源（CPU、RAM、磁盘空间、网络带宽、 I/O 容量）的实例类。

使用指标评估亚马逊 DocumentDB 实例使用情况 CloudWatch

您可以使用 CloudWatch 指标来观察您的实例吞吐量，并发现您的实例类是否为应用程序提供了足够的资源。有关实例类限制的信息，请访参阅 实例限制 并找到实例类的规格以了解您的网络性能。

如果您的实例使用情况接近实例类限制，则性能可能会开始变慢。这些 CloudWatch 指标可以证实这种情况，因此您可以计划手动扩展到更大的实例类别。

组合以下 CloudWatch 指标值以了解您是否已接近实例类限制：

- NetworkThroughput— Amazon DocumentDB 集群中每个实例的客户端接收和传输的网络吞吐量。此吞吐量值不包括集群中的实例与集群存储卷之间的网络流量。
- StorageNetworkThroughput— Amazon DocumentDB 集群中每个实例接收并发送到 Amazon DocumentDB 集群存储卷的网络吞吐量。

将添加到，NetworkThroughputStorageNetworkThroughput以查看 Amazon DocumentDB 集群中每个实例从亚马逊文档数据库集群存储卷接收和发送到该集群存储卷的网络吞吐量。您的实例的实例类限制应大于这两个组合指标的总和。

在发送和接收时，您可以使用以下指标来查看来自客户端应用程序的网络流量的更多详细信息：

- NetworkReceiveThroughput— Amazon DocumentDB 集群中每个实例从客户端接收的网络吞吐量。此吞吐量不包括集群中的实例与集群存储卷之间的网络流量。
- NetworkTransmitThroughput— Amazon DocumentDB 集群中每个实例发送给客户端的网络吞吐量。此吞吐量不包括集群中的实例与集群存储卷之间的网络流量。
- StorageNetworkReceiveThroughput— 集群中每个实例从 Amazon DocumentDB 集群存储卷接收的网络吞吐量。
- StorageNetworkTransmitThroughput— 集群中每个实例发送到 Amazon DocumentDB 集群存储卷的网络吞吐量。

将所有这些指标相加，以评估您的网络使用量与实例类限制的对比情况。实例类限制应大于这些组合指标的总和。

网络限制和实例的 CPU 利用率是相互的。当网络吞吐量增加时，CPU 利用率也会增加。监控 CPU 和网络使用情况可提供有关资源耗尽的方式和原因的信息。

为了帮助显著减少网络使用量，您可以考虑：

- 使用更大的实例类。
- 批量划分写入请求以减少总体事务量。
- 将只读工作负载定向到只读实例。
- 删除任何未使用的索引。

优化查询

提高集群性能的最佳方式之一是优化最常使用和占用最多资源的查询，以降低其运行成本。

您可以使用分析器（请参阅 分析 Amazon DocumentDB 操作）来记录在您集群上执行的操作的执行时间和详细信息。对于监控集群上速度最慢的操作以帮助您提高单个查询的性能和整体集群性能，分析器非常有用。

您还可以使用 `explain` 命令来了解如何分析特定查询的查询计划。您可以使用此信息修改查询或底层集合以提高查询性能（例如，添加索引）。

TTL 和时间序列工作负载

TTL 索引到期后的文档删除是一个工作量巨大的过程。无法保证在任何特定时间段内删除文档。有大量因素会影响 TTL 进程删除过期文档的时间，包括实例大小、实例资源利用率、文档大小、总吞吐量、索引数量以及索引和工作集是否位于内存中，等等。

当 TTL 监控器删除您的文档时，每次删除都会产生 I/O 费用，从而增加您的账单。如果吞吐量和 TTL 删除率增加，则由于 I/O 使用量增加，您应该会收到更高的账单。但是，如果您未创建 TTL 索引删除文档，而是将文档根据时间分成集合并且在不再需要这些集合时丢弃它们，则不会发生任何 IO 成本。这可能比使用 TTL 指数明显更有成本效益。

对于时间序列工作负载，您可以考虑创建滚动集合而不是 TTL 索引，因为滚动收集可能是删除数据的更好方法，而且密集度较低 I/O。如果您有大量馆藏（尤其是超过 1TB 的馆藏），或者 TTL 删除 I/O 成本是一个问题，我们建议您根据时间将文档分成集合，并在不再需要文档时删除馆藏。您可以每天或每周创建一个集合，具体取决于您的数据接收速率。虽然要求会因应用程序而异，但一个很好的经验法则是划分数量较多的小型集合，而不是数量较少的大型集合。删除这些集合不会产生 I/O 成本，而且比使用 TTL 索引更快、更具成本效益。

迁移

作为最佳做法，我们建议在将数据迁移到 Amazon DocumentDB 时，首先在 Amazon DocumentDB 中创建索引，然后再迁移数据。首先创建索引可以减少总时间并提高迁移速度。为此，您可以使用 Amazon DocumentDB 索引工具。有关迁移的更多信息，请参阅 Amazon DocumentDB 迁移指南。

我们还建议，在迁移您的生产数据库之前，最佳实践是考虑功能、性能、操作和成本，在 Amazon DocumentDB 上全面测试您的应用程序。

使用集群参数组

我们建议，在将集群参数组更改应用于生产集群前，您应在测试集群上试验这些更改。有关备份集群的信息，请参阅在 Amazon DocumentDB 中进行备份和还原。

聚合管道查询

创建具有多个阶段的聚合管道查询并且仅评估查询中的一个数据子集时，请将该 `$match` 阶段用作第一阶段或管道开头。首先使用 `$match` 将减少聚合管道查询中后续阶段需要处理的文档数量，从而提高查询性能。

#########

当执行高并发`batchInsert` and/or`batchUpdate`操作速率并且主实例上的`FreeableMemory`（CloudWatch 指标）量变为零时，您可以减少批量插入或更新工作负载的并发性，或者如果无法降低工作负载的并发性，则可以增加实例大小以增加数量。`FreeableMemory`

功能差异：Amazon DocumentDB 和 MongoDB

下面是 Amazon DocumentDB（与 MongoDB 兼容）与 MongoDB 之间的功能差异。

主题

- Amazon DocumentDB 的功能优势
- 更新的功能差异
- 与 MongoDB 之间的功能差异

Amazon DocumentDB 的功能优势

隐式事务

在 Amazon DocumentDB 中，所有 CRUD 语句（ `findAndModify`、`update`、`insert`、`delete` ）均保证原子性和一致性，即使对于修改多个文档的操作也是如此。随着 Amazon DocumentDB 4.0 推出，现在支持为多语句操作和多集合操作提供 ACID 属性的显式事务。有关在 Amazon DocumentDB 中使用交易的更多信息，请参阅。Amazon DocumentDB 中的事务

下面示例中介绍的 Amazon DocumentDB 中的操作用于修改可同时满足原子行为和一致行为的多个文档。

```
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 }]
})
```

组成批量操作（如 updateMany 和 deleteMany）的各个操作是原子操作，但整个批量操作不是原子操作。例如，如果各个插入操作成功执行而未出现错误，则整个 insertMany 操作是原子操作。如果 insertMany 操作遇到错误，insertMany 操作中每个单独的插入语句都将作为原子操作执行。如果您需要 insertMany、updateMany 和 deleteMany 操作的 ACID 属性，建议使用事务。

更新的功能差异

Amazon DocumentDB 通过从客户要求我们构建的功能中向后开发，持续改善与 MongoDB 的兼容性。本部分包含我们已在Amazon DocumentDB 中删除的功能差异，以便客户能够更轻松地迁移和构建应用程序。

主题

- 数组索引
- Multi-key 索引
- 字符串中的 Null 字符
- Role-based 访问控制
- $正则表达式索引

- 嵌套文档投影

数组索引

自 2020 年 4 月 23 日起，Amazon DocumentDB 现在支持对大于 2,048 个字节的数组编制索引的功能。数组中单个项的限制仍保持为 2,048 字节，这与 MongoDB 一致。

如果您正在创建新索引，则无需操作即可利用改进的功能。如果您有现有索引，则可以通过删除索引然后重新创建索引来利用改进的功能。具有改进功能的当前索引版本为 "v" : 3。

> **Note**
>
> 对于生产集群，删除索引可能会影响您的应用程序性能。我们建议您在更改生产系统时首先进行测试并谨慎行事。此外，重新创建索引所需的时间将是集合的总体数据大小的函数。

您可以使用以下命令查询索引的版本。

```
db.collection.getIndexes()
```

此操作的输出将类似于下文。在此输出中，索引的版本是 "v" : 3，这是最新的索引版本。

```
[
    {
        "v" : 3,
        "key" : {
        "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.test"
    }
]
```

Multi-key 索引

自 2020 年 4 月 23 日起，Amazon DocumentDB 现在支持在同一个数组中创建具有多个键的复合索引的功能。

如果您正在创建新索引，则无需操作即可利用改进的功能。如果您有现有索引，则可以通过删除索引然后重新创建索引来利用改进的功能。具有改进功能的当前索引版本为 "v" : 3。

> **Note**
>
> 对于生产集群，删除索引可能会影响您的应用程序性能。我们建议您在更改生产系统时首先进行测试并谨慎行事。此外，重新创建索引所需的时间将是集合的总体数据大小的函数。

您可以使用以下命令查询索引的版本。

```
db.collection.getIndexes()
```

此操作的输出将类似于下文。在此输出中，索引的版本是 "v" : 3，这是最新的索引版本。

```
[
    {
        "v" : 3,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.test"
    }
]
```

字符串中的 Null 字符

自 2020 年 6 月 22 日起，Amazon DocumentDB 现在支持字符串中的 null 字符（'\0'）。

Role-based 访问控制

自 2020 年 3 月 26 日起，Amazon DocumentDB 面向内置角色支持基于角色的访问控制 (RBAC)。要了解更多信息，请参阅Role-Based 访问控制。

$##表达式索引

自 2020 年 6 月 22 日起，Amazon DocumentDB 现在支持 $regex 运算符使用索引的功能。

要将索引与 $regex 运算符一起使用，您必须使用 hint() 命令。使用 hint() 时，您必须指定将 $regex 应用到的字段的名称。例如，如果您在字段 product 上有索引，且索

引名称为 p_1，则 db.foo.find({product: /^x.*/}).hint({product:1}) 将使用 p_1 索引，但 db.foo.find({product: /^x.*/}).hint("p_1") 不使用该索引。您可以通过使用 explain() 命令或使用分析器记录慢速查询来验证是否选择了索引。例如 db.foo.find({product: /^x.*/}).hint("p_1").explain()。

> **Note**
>
> hint() 方法一次只能与一个索引一起使用。

为 $regex 查询使用索引的功能，针对使用前缀但未指定 i、m 或 o 正则表达式选项的正则表达式查询进行了优化。

将索引与 $regex 结合使用时，建议您在具有高选择性的字段上创建索引，这些字段的重复值数量低于集合中总文档数的 1% 的字段。例如，如果您的集合包含 100,000 个文档，则仅在相同值出现 1000 次或更少的字段上创建索引。

嵌套文档投影

在 3.6 版本中，Amazon DocumentDB 与 MongoDB 之间存在带 $project 运算符的功能差异，该差异已在 Amazon DocumentDB 4.0 中得到解决，但在 Amazon DocumentDB 3.6 中将仍不予支持。

Amazon DocumentDB 3.6 仅在应用投影时才考虑嵌套文档中的第一个字段，而 MongoDB 3.6 将解析子文档并将投影应用于每个子文档。

例如：如果投影是 "a.b.c": 1，则该行为在 Amazon DocumentDB 和 MongoDB 中均按预期运行。但是，如果投影是 {a:{b:{c:1}}}，则 Amazon DocumentDB 3.6 仅将该投影应用于a，而非 b 或 c。在 Amazon DocumentDB 4.0 中，投影 {a:{b:{c:1}}} 将应用于 a、b 和 c。

与 MongoDB 之间的功能差异

主题

- $向量搜索运算符
- OpCountersCommand
- 管理数据库和集合
- cursormaxTimems
- explain()

- 索引构建
- 在路径中用空键查找
- MongoDB API、操作和数据类型
- mongodump 和 mongorest ore 工具
- 结果排序
- 可重试写入
- 稀疏索引
- 在 $all 表达式中使用 $elemMatch
- 字段名称中的美元符号（$）和句点（.）
- $lookup
- $自然排序和反向排序

$#######

Amazon DocumentDB 不支持 $vectorSearch 作为独立运算符。相反，我们支持 $search 运算符内的 vectorSearch。有关更多信息，请参阅 Amazon DocumentDB 向量搜索。

OpCountersCommand

Amazon DocumentDB 的OpCountersCommand行为偏离于 MongoDB 的opcounters.command 如下：

- MongoDB 的opcounters.command 计入除插入、更新和删除之外的所有命令，而 Amazon DocumentDB 的 OpCountersCommand 也排除 find 命令。
- Amazon DocumentDB 将一些内部命令计入 OpCountersCommand。

管理数据库和集合

Amazon DocumentDB 不支持管理或本地数据库，MongoDB system.* 或 startup_log 集合也不支持。

cursormaxTimems

在 Amazon DocumentDB 中，cursor.maxTimeMS 重置每个请求的计数器。getMore因此，如果指定了 3000MS maxTimeMS，则该查询耗时 2800MS，而每个后续getMore请求耗时 300MS，则

游标不会超时。游标仅在单个操作（无论是查询还是单个getMore请求）耗时超过指定值时才将超时maxTimeMS。此外，检查游标执行时间的扫描器以五 (5) 分钟间隔尺寸运行。

explain()

Amazon DocumentDB 在专门构建的数据库引擎上模拟 MongoDB 3.6、4.0、5.0 和 8.0 API，该引擎利用分布式、容错、自我修复的存储系统。因此，查询计划和 `explain()` 的输出在 Amazon DocumentDB 和 MongoDB 之间可能有所不同。希望控制其查询计划的客户可以使用 `$hint` 运算符强制选择首选索引。

索引构建

在任何给定时间，Amazon DocumentDB 只允许在一个集合中构建一个索引。或在前台，或在后台。如果当前正在构建索引，在同一集合上发生了 `createIndex()` 或 `dropIndex()` 之类的操作，则新尝试的操作将失败。

默认情况下，Amazon DocumentDB 和 MongoDB 版本 4.0 中的索引构建在前台进行。如果指定为 createIndexes 或其 Shell 助手 `createIndex()` 和 `createIndexes()`，则 MongoDB 版本 4.2 及更高版本将忽略后台索引构建选项。

在构建索引完成后，有效时间 (TTL) 索引开始使文档过期。

在路径中用空键查找

当您用包含空字符串作为路径一部分的键（例如`x.`、`x..b`）查找，并且该对象在数组内部有一个空字符串键路径（例如`{"x" : [ { "" : 10 }, { "b" : 20 } ]}`）时，Amazon DocumentDB 将返回与您在 MongoDB 中运行相同查找时不同的结果。

在 MongoDB 中，当空字符串键不在路径查找的末尾时，在数组内部查找空键路径如预期那样工作。但是，当空字符串键位于路径查找的末尾时，它不查看数组。

但是，在 Amazon DocumentDB 中，仅读取数组内部的第一个元素，因为 `getArrayIndexFromKeyString` 将空字符串转换成 `0`，从而字符串键查找作为数组索引查找处理。

MongoDB API、操作和数据类型

亚马逊 DocumentDB 与 MongoDB 3.6、4.0、5.0 和 8.0 API 兼容。有关支持的功能的最新列表，请参阅 Amazon DocumentDB 中支持的 MongoDB API、操作和数据类型。

mongodump # mongorest ore

Amazon DocumentDB 不支持管理数据库，因此在使用 mongodump 或 mongorestore 实用程序时不转储或还原管理数据库。当您在 Amazon DocumentDB 中使用 mongorestore创建新的数据库时，除了执行还原操作外，还需要重新创建用户角色。

> **Note**
>
> 我们对 Amazon DocumentDB 推荐高达且包括版本 100.6.1 的 MongoDB 数据库工具。您可以在此处下载 MongoDB 数据库工具。

结果排序

Amazon DocumentDB 不保证结果集的隐式结果排序顺序。要确保结果集的顺序，可使用 sort() 显式指定排序顺序。

以下示例根据库存字段按降序对清单集合中的项目排序。

```
db.inventory.find().sort({ stock: -1 })
```

使用 $sort 聚合阶段时，除非 $sort 阶段是聚合管道中的最后一个阶段，否则不保留排序顺序。将 $sort 聚合阶段结合 $group 聚合阶段结合使用时，$sort 聚合阶段仅适用于 $first 和 $last 累加器。在 Amazon DocumentDB 4.0 中，增加支持 $push 以遵守来自上个 $sort 阶段的排序顺序。

可重试写入

从 MongoDB 4.2 可兼容驱动程序开始，默认情况下可重试写入处于启用状态。但是，当前 Amazon DocumentDB 不支持可重试写入。该功能差异将显示在类似以下内容的错误消息中。

```
{"ok":0,"errmsg":"Unrecognized field: 'txnNumber'","code":9,"name":"MongoError"}
```

可以通过连接字符串（例如MongoClient("mongodb://my.mongodb.cluster/db?retryWrites=false")）或 MongoClient 构造函数的关键字参数（例如）禁用可重试写入。MongoClient("mongodb://my.mongodb.cluster/db", retryWrites=False)

下面是一个在连接字符串中禁用可重试写入的 Python 示例。

```
client =
 pymongo.MongoClient('mongodb://
<username>:<password>@docdb-2019-03-17-16-49-12.cluster-ccuszbx3pn5e.us-
east-1.docdb.amazonaws.com:27017/?
replicaSet=rs0',w='majority',j=True,retryWrites=False)
```

稀疏索引

要使用您在查询中创建的稀疏索引，必须在涵盖索引的字段中使用 `$exists` 子句。如果省略 `$exists`，Amazon DocumentDB 将不使用稀疏索引。

示例如下：

```
db.inventory.count({ "stock": { $exists: true }})
```

对于稀疏的多键索引，如果查找文档生成了一组值并且只缺少了部分索引字段，则 Amazon DocumentDB 不支持唯一键约束。例如，如果输入为 `"a" : [ { "b" : 2 }, { "c" : 1 } ]`，则 `createIndex({"a.b" : 1 }, { unique : true, sparse :true })` 不受支持，因为 `"a.c"` 存储在索引中。

$all ###### $elemMatch

当前，Amazon DocumentDB 不支持在 `$all`表达式中使用 `$elemMatch` 运算符。一种解决方法是，您可以将 `$and` 运算符与 `$elemMatch` 结合使用，如下所示。

原始运算：

```
db.col.find({
  qty: {
    $all: [
      { "$elemMatch": { part: "xyz", qty: { $lt: 11 } } },
      { "$elemMatch": { num: 40, size: "XL" } }
    ]
  }
})
```

更新后的运算：

```
db.col.find({
  $and: [
```

```
    { qty: { "$elemMatch": { part: "xyz", qty: { $lt: 11 } } } },
    { qty: { "$elemMatch": { qty: 40, size: "XL" } } }
  ]
})
```

字段名称中的美元符号（$）和句点（.）

Amazon DocumentDB 不支持在嵌套对象中查询 $in、$nin 和 $all 中以美元符号（$）为前缀的字段。例如，以下查询在 Amazon DocumentDB 中无效：

```
coll.find({"field": {"$all": [{ "$a": 1 }]}})
```

$lookup

Amazon DocumentDB 支持进行相等匹配（例如，左外联接），也支持不相关子查询，但不支持相关的子查询。

使用带有 $ lookup 的索引

现在，您可以配合 $lookup 阶段运算符使用索引。根据您的用例，存在您可以用来优化性能的多种索引化算法。这个部分将解释用于 $lookup 的不同索引化算法，并帮助您选择最适合自己工作负载的索引化算法。

默认情况下，Amazon DocumentDB 在使用allowDiskUse:false时将使用哈希算法，在使用allowDiskUse:true时将使用排序合并。

> **Note**
>
> find 命令目前不支持 allowDiskUse 选项。该选项仅支持作为聚合的一部分。建议将聚合框架与 allowDiskUse:true 配合使用，以处理可能超过内存限制的大型查询。

对于某些用例，可能需要强制查询优化器来使用一个不同算法。以下是 $lookup 聚合运算符可以使用的不同索引化算法：

- 嵌套循环：如果外部集合小于 1 GB 且外部集合中的字段具有索引，则嵌套循环计划通常有益于工作负载。如果正在使用嵌套循环算法，则解释计划将该阶段显示为NESTED_LOOP_LOOKUP。
- 排序合并：如果外部集合对查找中所用字段没有索引并且工作数据集不适合内存，则排序合并计划通常有益于工作负载。如果正在使用排序合并算法，则解释计划将该阶段显示为SORT_LOOKUP。

- 哈希：如果外部集合小于 1GB 且工作数据集适合内存，则哈希计划通常有益于工作负载。如果正在使用哈希算法，则解释计划将该阶段显示为HASH_LOOKUP。

您可以通过在查询上使用 explain，确定正用于 $lookup 运算符的索引化算法。以下是示例：

```
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
}
```

作为使用 explain() 方法的替代，您可以使用探查器查看在您使用 $lookup 运算符时所采用的算法。有关探查器的更多信息，请参阅分析 Amazon DocumentDB 操作。

使用 P **lanHin** t

如果您希望强制查询优化器配合$lookup使用不同的索引化算法，则您可以使用 planHint 。为此，请使用聚合阶段选项中的注释来强制执行一个不同计划。下面是该注释的语法示例：

```
comment : {
    comment :  "<string>",
    lookupStage : { planHint : "SORT" | "HASH" | "NESTED_LOOP" }
}
```

以下是使用planHint 强制查询优化器使用HASH索引化算法的示例：

```
db.foo.aggregate(
   [
      {
         $lookup:
            {
               from: "foo",
               localField: "_id",
               foreignField: "_id",
               as: "joined"
            },
      }
```

```
   ]
),
{
   comment : "{ \"lookupStage\" : { \"planHint\": \"HASH\" }}"
```

要测试哪种算法最适合您的工作负载，可以使用 explain 方法的executionStats参数计量 $lookup 阶段的执行时间，同时修改索引化算法（即HASH/SORT/NESTED_LOOP）。

以下示例说明使用 SORT 算法，如何使用 executionStats 计量 $lookup 阶段的执行时间。

```
db.foo.explain("executionStats").aggregate(
   [
      {
         $lookup:
            {
               from: "foo",
               localField: "_id",
               foreignField: "_id",
               as: "joined"
            },
      }
   ]
),
{
   comment : "{ \"lookupStage\" : { \"planHint\": \"SORT\" }}"
```

$###序和反向排序

Amazon DocumentDB 仅支持 $natural 进行正向集合扫描。反向集合扫描（{$natural: -1}）将导致 MongoServerError。

Amazon DocumentDB 中支持的 MongoDB API、操作和数据类型

Amazon DocumentDB（与 MongoDB 兼容）是一个快速、可扩展、高度可用且完全托管的文档数据库服务，它支持 MongoDB 工作负载。亚马逊 DocumentDB 与 MongoDB 3.6、4.0、5.0 和 8.0 API 兼容。本部分列出了支持的功能。要获得使用 MongoDB API 和驱动程序的支持，请访问 MongoDB 社区论坛。要获得使用 Amazon DocumentDB 服务的支持，请联系相应的 AWS 支持团队。有关 Amazon DocumentDB 和 MongoDB 之间的功能差异，请参阅。功能差异：Amazon DocumentDB 和 MongoDB

仅供内部使用或不适用于完全托管的服务的 MongoDB 命令和运算符不受支持，并且未包含在支持的功能列表中。

自发布以来，我们已添加 50 多种附加功能，并将继续向客户学习以提供他们所需的功能。有关最近发布的信息，请参阅 Amazon DocumentDB 公告。

如果您希望我们构建一项不受支持的功能，请向 Amazon DocumentDB 服务团队发送一封包含您的 accountID、请求的功能和使用案例的电子邮件来告知我们此情况。

主题

- 数据库命令
- 查询和投影运算符
- 更新运算符
- 地理空间
- 游标方法
- 聚合管道运算符
- 数据类型
- 索引

数据库命令

主题

- 管理命令

- 聚合
- 身份验证
- 诊断命令
- 查询和写入操作
- 角色管理命令
- 会话命令
- User management
- 分片命令

管理命令

命令	3.6	4.0	5.0	8.0	弹性集群
受限集合	否	否	否	否	否
克隆 CollectionAsCapped	否	否	否	否	否
collMod	部分	部分	部分	部分	部分
collMod：过期 AfterSeconds	支持	是	是	是	是
转换 ToCapped	否	否	否	否	否
copydb	否	否	否	否	否
创建	支持	是	是	是	是
createView	否	否	否	是	否
createIndexes	支持	是	是	是	是
currentOp	支持	是	是	是	是

命令	3.6	4.0	5.0	8.0	弹性集群
drop	支持	是	是	是	是
dropDatabase	支持	是	是	是	是
dropIndexes	支持	是	是	是	是
filemd5	否	否	否	否	否
得到 AuditConfig	否	是	是	是	否
killCursors	支持	是	是	是	是
killOp	支持	是	是	是	是
listCollections*	支持	是	是	是	是
listDatabases	支持	是	是	是	是
listIndexes	支持	是	是	是	是
reIndex	否	否	是	是	否
renameCollection	支持	是	是	是	否
设置 AuditConfig	否	是	是	是	否

* 不支持筛选选项中的 `type` 密钥。

聚合

命令	3.6	4.0	5.0	8.0	弹性集群
aggregate	支持	是	是	是	是
count	支持	是	是	是	是
区分	支持	是	是	是	是
mapReduce	否	否	否	是	否

身份验证

命令	3.6	4.0	5.0	8.0	弹性集群
authenticate	支持	是	是	是	是
注销	支持	是	是	是	是

诊断命令

命令	3.6	4.0	5.0	8.0	弹性集群
buildInfo	支持	是	是	是	是
collStats	支持	是	是	是	是
conn PoolStats	否	否	否	否	否
connectio nStatus	支持	是	是	是	是
dataSize	支持	是	是	是	是
dbHash	否	否	否	否	否

命令	3.6	4.0	5.0	8.0	弹性集群
dbStats	支持	是	是	是	是
explain	支持	是	是	是	是
解释：executionStats	支持	是	是	是	是
功能	否	否	否	否	否
hostInfo	支持	是	是	是	是
listCommands	支持	是	是	是	是
profiler	是	是	是	是	否
serverStatus	支持	是	是	是	是
top	支持	是	是	是	是

查询和写入操作

命令	3.6	4.0	5.0	8.0	弹性集群
变更流	是	是	是	是	否
删除	支持	是	是	是	是
查找	支持	是	是	是	是
找到AndModify	支持	是	是	是	是
得到LastError	否	否	否	否	否
getMore	支持	是	是	是	是

命令	3.6	4.0	5.0	8.0	弹性集群
得到 PrevError	否	否	否	否	否
GridFS	支持	是	是	是	否
insert	支持	是	是	是	是
parallel CollectionScan	否	否	否	否	否
resetError	否	否	否	否	否
更新	支持	是	是	是	是
ReplaceOne	是	是	是	是	是

角色管理命令

命令	3.6	4.0	5.0	8.0	弹性集群
createRole	支持	是	是	是	否
dropAllRoles FromDatabase	支持	是	是	是	否
dropRole	支持	是	是	是	否

命令	3.6	4.0	5.0	8.0	弹性集群
支持	是	是	是	否	
支持	是	是	是	否	
支持	是	是	是	否	
支持	是	是	是	否	
支持	是	是	是	否	

会话命令

命令	3.6	4.0	5.0	8.0	弹性集群
abortTransaction	否	是	是	是	否
commitTransaction	否	是	是	是	否
endSessions	否	否	否	否	否

命令	3.6	4.0	5.0	8.0	弹性集群
杀 AllSessions	否	是	是	是	否
杀 AllSessionsByPattern	否	否	否	否	否
killSessions	否	是	是	是	否
refreshSessions	否	否	否	否	否
startSession	否	是	是	是	否

User management

命令	3.6	4.0	5.0	8.0	弹性集群
createUser	支持	是	是	是	是
掉落 AllUsersFromDatabase	支持	是	是	是	是
dropUser	支持	是	是	是	是
授予 RolesToUser	支持	是	是	是	是
撤销 RolesFromUser	支持	是	是	是	是
updateUser	支持	是	是	是	是
usersInfo	支持	是	是	是	是

分片命令

命令	弹性集群
中止 ReshardCollection	否
addShard	否
添加 ShardToZone	否
平衡器 CollectionStatus	否
balancerStart	否
balancerStatus	否
balancerStop	否
检查 ShardingIndex	否
清除 JumboFlag	否
cleanupOrphaned	否
清理 ReshardCollection	否
提交 ReshardCollection	否
enableSharding	是
脸红 RouterConfig	否
得到 ShardMap	否
得到 ShardVersion	否
isdbgrid	否
listShards	否
medianKey	否

命令	弹性集群
moveChunk	否
movePrimary	否
mergeChunks	否
完善 CollectionShardKey	否
removeShard	否
移除 ShardFromZone	否
reshardCollection	否
设置 AllowMigrations	否
设置 ShardVersion	否
shardCollection	是
shardingState	否
split	否
splitVector	否
unsetSharding	否
更新 ZoneKeyRange	否

查询和投影运算符

主题

- 数组运算符
- 按位运算符
- 注释运算符
- 比较运算符

- 元素运算符
- 评估查询运算符
- 逻辑运算符
- 投影运算符

数组运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$all	支持	是	是	是	是
$elemMatch	是	是	是	是	是
$size	是	是	是	是	是

按位运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$bits AllSet	支持	是	是	是	是
$bits AnySet	是	是	是	是	是
$bits AllClear	是	是	是	是	是
$bits AnyClear	是	是	是	是	是

注释运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$comment	支持	是	是	是	是

比较运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$eq	支持	是	是	是	是
$gt	是	是	是	是	是
$gte	是	是	是	是	是
$in	是	是	是	是	是
$lt	是	是	是	是	是
$lte	是	是	是	是	是
$ne	是	是	是	是	是
$nin	是	是	是	是	是

元素运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$exists	支持	是	是	是	是
$type	是	是	是	是	是

评估查询运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$expr	否	是	是	是	否
$jsonSchema	否	是	是	是	否
$mod	是	是	是	是	是

命令	3.6	4.0	5.0	8.0	弹性集群
$regex	是	是	是	是	是
$text	否	否	是	是	否
$where	否	否	否	否	否

逻辑运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$and	支持	是	是	是	是
$nor	是	是	是	是	是
$not	是	是	是	是	是
$or	是	是	是	是	是

投影运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$	支持	是	是	是	是
$elemMatch	是	是	是	是	是
$meta	否	否	是	是	否
$slice	是	是	是	是	是

更新运算符

主题

- 数组运算符

- 按位运算符
- 字段运算符
- 更新修改器

数组运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$	支持	是	是	是	是
$[]	是	是	是	是	是
$ [<标识符>]	是	是	是	是	是
$addToSet	支持	是	是	是	是
$pop	支持	是	是	是	是
$pullAll	支持	是	是	是	是
$pull	支持	是	是	是	是
$push	支持	是	是	是	是

按位运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$bit	支持	是	是	是	是

字段运算符

运算符	3.6	4.0	5.0	8.0	弹性集群
$currentDate	支持	是	是	是	是

运算符	3.6	4.0	5.0	8.0	弹性集群
$inc	支持	是	是	是	是
$max	是	是	是	是	是
$min	是	是	是	是	是
$mul	支持	是	是	是	是
$rename	支持	是	是	是	是
$set	是	是	是	是	是
$setOnInsert	支持	是	是	是	是
$unset	是	是	是	是	是

更新修改器

运算符	3.6	4.0	5.0	8.0	弹性集群
$each	支持	是	是	是	是
$position	支持	是	是	是	是
$slice	是	是	是	是	是
$sort	是	是	是	是	是

地理空间

几何说明符

查询选择器	3.6	4.0	5.0	8.0	弹性集群
BOX	否	否	否	否	否

查询选择器	3.6	4.0	5.0	8.0	弹性集群
.center	否	否	否	否	否
$centerSphere	否	否	否	否	否
$geometry	是	是	是	是	是
$maxDistance	是	是	是	是	是
$minDistance	是	是	是	是	是
$nearSphere	是	是	是	是	是
$polygon	否	否	否	否	否
$uniqueDocs	否	否	否	否	否

查询选择器

命令	3.6	4.0	5.0	8.0	弹性集群
$geoIntersects	支持	是	是	是	是
$geoWithin	是	是	是	是	是
$near	是	是	是	是	是
$nearSphere	是	是	是	是	是
$polygon	否	否	否	否	否
$uniqueDocs	否	否	否	否	否

游标方法

命令	3.6	4.0	5.0	8.0	弹性集群
cursor.batchSize()	支持	是	是	是	是
cursor.close()	支持	是	是	是	是
cursor.collation()	否	否	否	是	否
cursor.comment()	支持	是	是	是	是
cursor.count()	支持	是	是	是	是
cursor.explain()	支持	是	是	是	否
cursor.forEach()	支持	是	是	是	是
cursor.hasNext()	支持	是	是	是	是
cursor.hint()	支持	是	是	是	是*
cursor.isClosed()	支持	是	是	是	是
cursor.isExhausted()	支持	是	是	是	否
cursor.itcount()	支持	是	是	是	否
cursor.limit()	支持	是	是	是	否
cursor.map()	支持	是	是	是	否

命令	3.6	4.0	5.0	8.0	弹性集群
cursor.max()	否	否	否	否	否
cursor.maxScan()	支持	是	是	是	否
cursor.maxTimeMS()	支持	是	是	是	否
cursor.min()	否	否	否	否	否
cursor.next()	支持	是	是	是	是
cursor.noCursorTimeout ()	否	否	否	否	否
cursor.objsLeftInBatch ()	支持	是	是	是	否
cursor.pretty()	支持	是	是	是	否
cursor.readConcern()	支持	是	是	是	否
cursor.readPref()	支持	是	是	是	否
cursor.returnKey()	否	否	否	否	否
cursor.showRecordId ()	否	否	否	否	否
cursor.size()	支持	是	是	是	否
cursor.skip()	支持	是	是	是	否

命令	3.6	4.0	5.0	8.0	弹性集群
cursor.sort()	支持	是	是	是	否
cursor.tailable()	否	否	否	否	否
cursor.toArray()	支持	是	是	是	否

* 索引 hint 表达式支持索引。例如 db.foo.find().hint({x:1})。

聚合管道运算符

主题

- 累加器表达式
- 算术运算符
- 数组运算符
- 布尔运算符
- 比较运算符
- 条件表达式运算符
- 数据类型运算符
- 数据大小运算符
- 日期运算符
- 文字运算符
- 合并运算符
- 自然运算符
- 集合运算符
- 阶段运算符
- 字符串运算符
- 系统变量
- 文本搜索运算符
- 类型转换运算符

- 变量运算符
- 其他运算符

累加器表达式

Expression	3.6	4.0	5.0	8.0	弹性集群
$accumulator	-	-	否	否	否
$add ToSet	是	是	是	是	是
$avg	是	是	是	是	是
$count	-	-	否	否	否
$covarian cePop	否	否	否	否	否
$covarian ceSamp	否	否	否	否	否
$denseRank	否	否	否	否	否
$derivative	否	否	否	否	否
$document Number	否	否	否	否	否
$exp MovingAvg	否	否	否	否	否
$first	是	是	是	是	是
$integral	否	否	否	否	否
$last	是	是	是	是	是
$max	是	是	是	是	是
$min	是	是	是	是	是

Expression	3.6	4.0	5.0	8.0	弹性集群
$push	是	是	是	是	是
$rank	否	否	否	否	否
$shift	否	否	否	否	否
$std DevPop	否	否	否	否	否
$std DevSamp	否	否	否	否	否
$sum	是	是	是	是	是

算术运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$abs	支持	是	是	是	是
$add	是	是	是	是	是
$ceil	否	是	是	是	是
$divide	是	是	是	是	是
$exp	否	是	是	是	是
$floor	否	是	是	是	是
$ln	否	是	是	是	是
$log	否	是	是	是	是
$log10	否	是	是	是	是
$mod	是	是	是	是	是
$multiply	是	是	是	是	是

命令	3.6	4.0	5.0	8.0	弹性集群
$pow	否	否	是的 (5.0.1+)	是	否
$round	-	-	否	否	否
$sqrt	否	是	是	是	是
$subtract	是	是	是	是	是
$trunc	否	否	否	否	否

数组运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$arrayElemAt	支持	是	是	是	是
$arrayToObject	是	是	是	是	是
$concatArrays	是	是	是	是	是
$filter	是	是	是	是	是
$first	-	-	是	是	否
$in	是	是	是	是	是
$indexOfArray	是	是	是	是	是
$isArray	是	是	是	是	是
$last	-	-	是	是	否

命令	3.6	4.0	5.0	8.0	弹性集群
$objectToArray	是	是	是	是	是
$range	是	是	是	是	是
$reverseArray	是	是	是	是	是
$reduce	是	是	是	是	是
$size	是	是	是	是	是
$slice	是	是	是	是	是
$zip	是	是	是	是	是

布尔运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$and	支持	是	是	是	是
$not	是	是	是	是	是
$or	是	是	是	是	是

比较运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$cmp	支持	是	是	是	是
$eq	是	是	是	是	是
$gt	是	是	是	是	是

命令	3.6	4.0	5.0	8.0	弹性集群
$gte	是	是	是	是	是
$lt	是	是	是	是	是
$lte	是	是	是	是	是
$ne	是	是	是	是	是

条件表达式运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$cond	支持	是	是	是	是
$ifNull	是	是	是	是	是
$switch	否	是	是	是	否

数据类型运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$type	支持	是	是	是	是

数据大小运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$binarySize	-	-	否	否	否
$bsonSize	-	-	否	否	否

日期运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$dateAdd	否	否	是	是	是
$dateDiff	-	-	是	是	否
$date FromParts	否	否	是的 (5.0.1+)	否	否
$date FromString	是	是	是	是	是
$dateSubt ract	否	否	是	是	是
$date ToParts	否	否	是的 (5.0.1+)	否	否
$date ToString	是	是	是	是	是
$dateTrunc	-	-	否	是	否
$day OfMonth	是	是	是	是	是
$day OfWeek	是	是	是	是	是
$day OfYear	是	是	是	是	是
$hour	是	是	是	是	是
$iso DayOfWeek	是	是	是	是	是
$isoWeek	是	是	是	是	是

命令	3.6	4.0	5.0	8.0	弹性集群
$isoWeekYear	是	是	是	是	是
$millisecond	是	是	是	是	是
$minute	是	是	是	是	是
$month	是	是	是	是	是
$second	是	是	是	是	是
$week	是	是	是	是	是
$year	是	是	是	是	是

文字运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$literal	支持	是	是	是	是

合并运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$mergeObjects	支持	是	是	是	是

自然运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$natural	支持	是	是	是	是

集合运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$allElementsTrue	否	是	是	是	是
$anyElementTrue	否	是	是	是	是
$setDifference	否	是	是	是	是
$setEquals	是	是	是	是	是
$setIntersection	是	是	是	是	是
$set IsSubset	是	是	是	是	是
$setUnion	是	是	是	是	是
$setWindowFields	否	否	否	否	否

阶段运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$addFields	支持	是	是	是	是
$bucket	否	否	否	是	否
$bucketAuto	否	否	否	否	

命令	3.6	4.0	5.0	8.0	弹性集群
$changeStream	是	是	是	是	否
$collStats	否	是	是	是	否
$count	是	是	是	是	是
$currentOp	是	是	是	是	是
$facet	否	否	否	否	否
$geoNear	是	是	是	是	是
$graphLookup	否	否	否	否	否
$group	是	是	是	是	是
$indexStats	是	是	是	是	是
$limit	是	是	是	是	是
$listLocalSessions	否	否	否	否	否
$listSessions	否	否	否	否	否
$lookup	是	是	是	是	是
$match	是	是	是	是	是
merge	-	-	否	是	否
$out	是	是	是	是	否
$planCacheStats	-	-	否	否	否

命令	3.6	4.0	5.0	8.0	弹性集群
$project	是	是	是	是	是
$redact	是	是	是	是	是
$replaceRoot	是	是	是	是	是
$sample	是	是	是	是	是
$set	-	-	否	是	否
$set WindowFields	-	-	否	否	否
$skip	是	是	是	是	是
$sort	是	是	是	是	是
$sort ByCount	否	否	否	否	否
$unionWith	-	-	否	否	否
$unset	-	-	否	是	否
$unwind	是	是	是	是	是
$replaceWith	否	否	否	是	否
$向量搜索	否	否	否	是	否

字符串运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$concat	支持	是	是	是	是

命令	3.6	4.0	5.0	8.0	弹性集群
$indexOfBytes	是	是	是	是	是
$indexOfCP	是	是	是	是	是
$ltrim	否	是	是	是	否
$regexFind	-	-	是	是	否
$正则表达式FindAll	-	-	是	是	否
$regexMatch	-	-	是	是	否
$replaceAll	-	-	是	是	否
$replaceOne	-	-	是	是	否
$rtrim	否	是	是	是	否
$split	是	是	是	是	是
$strcasecmp	是	是	是	是	是
$str LenBytes	是	是	是	是	是
$strLenCP	是	是	是	是	是
$substr	是	是	是	是	是
$substrBytes	是	是	是	是	是
$substrCP	是	是	是	是	是
$toLower	是	是	是	是	是
$toUpper	是	是	是	是	是
$trim	否	是	是	是	否

系统变量

命令	3.6	4.0	5.0	8.0	弹性集群
$$CURRENT	否	否	否	否	否
$$DESCEND	是	是	是	是	是
$$KEEP	是	是	是	是	是
$$PRUNE	是	是	是	是	是
$$REMOVE	否	否	否	否	否
$ROOT	是	是	是	是	是

文本搜索运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$meta	否	否	是	是	否
$search	否	否	是	是	否

类型转换运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$convert	否	是	是	是	是
$isNumber	-	-	否	否	否
$toBool	否	是	是	是	是
$toDate	否	是	是	是	是
$toDecimal	否	是	是	是	是

命令	3.6	4.0	5.0	8.0	弹性集群
$toDouble	否	是	是	是	是
$toInt	否	是	是	是	是
$toLong	否	是	是	是	是
$to ObjectId	否	是	是	是	是
$toString	否	是	是	是	是

变量运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$let	支持	是	是	是	是
$map	是	是	是	是	是

其他运算符

命令	3.6	4.0	5.0	8.0	弹性集群
$getField	-	-	否	否	否
$rand	-	-	是的 (5.0.1+)	是	否
$sampleRate	-	-	否	否	否

数据类型

命令	3.6	4.0	5.0	8.0	弹性集群
32-bit Integer (int)	支持	是	是	是	是
64-bit Integer (long	支持	是	是	是	是
数组	支持	是	是	是	是
二进制数据	支持	是	是	是	是
布尔值	支持	是	是	是	是
日期	支持	是	是	是	是
DBPointer	否	否	否	否	否
DBREF	否	否	否	否	否
Decimal128	支持	是	是	是	是
双精度	支持	是	是	是	是
JavaScript	否	否	否	否	否
JavaScript（带瞄准镜）	否	否	否	否	否
MaxKey	是	是	是	是	是
MinKey	是	是	是	是	是
Null	支持	是	是	是	是
对象	支持	是	是	是	是
ObjectId	是	是	是	是	是

命令	3.6	4.0	5.0	8.0	弹性集群
正则表达式	支持	是	是	是	是
字符串	是	是	是	是	是
符号	否	否	否	否	否
Timestamp	支持	是	是	是	是
未定义	否	否	否	否	否

索引和索引属性

主题

- 索引
- 索引属性

索引

命令	3.6	4.0	5.0	8.0	弹性集群
2dsphere	支持	是	是	是	是
2d 索引	否	否	否	否	否
复合索引	支持	是	是	是	是
哈希索引	否	否	否	否	否
多键索引	支持	是	是	是	是
单个字段索引	支持	是	是	是	是
文本索引	否	否	是	是	否
通配符	否	否	否	否	否

索引属性

命令	3.6	4.0	5.0	8.0	弹性集群
背景	支持	是	是	是	是
区分大小写	否	否	否	是	否
隐藏	否	否	否	否	否
部分	否	否	是	是	否
稀疏	支持	是	是	是	是
文本	否	否	是	是	否
TTL	支持	是	是	是	是
Unique	支持	是	是	是	是
向量	否	否	是	是	否

有关特定 MongoDB 运算符的详细信息，请参阅以下主题：

- 聚合管道运算符
- 地理空间
- 投影运算符
- 更新运算符

聚合管道运算符

本节提供有关 Amazon DocumentDB 支持的聚合管道运算符的详细信息。

主题

- $$DESCEND
- $$KEEP
- $$PRUNE

- $ROOT
- $abs
- $add
- $add ToSet
- $addFields
- $all ElementsTrue
- $and
- $any ElementTrue
- $array ElemAt
- $array ToObject
- $avg
- $bucket
- $ceil
- $changeStream
- $cmp
- $collStats
- $concat
- $concatArrays
- $cond
- $convert
- $count
- $currentOp
- $dateAdd
- $dateDiff
- $date FromString
- $dateSubtract
- $date ToString
- $dateTrunc
- $day OfMonth

- $day OfWeek
- $day OfYear
- $divide
- $eq
- $exp
- $filter
- $first
- $floor
- $geoNear
- $group
- $gt
- $gte
- $hour
- $ifNull
- $in
- $index OfArray
- $index OfBytes
- $indexOfCP
- $indexStats
- $isArray
- $iso DayOfWeek
- $isoWeek
- $iso WeekYear
- $last
- $let
- $limit
- $literal
- $ln
- $log

- $log10
- $lt
- $lte
- $lookup
- $ltrim
- $map
- $match
- $max
- $meta
- merge
- $mergeObjects
- $millisecond
- $min
- $minute
- $mod
- $month
- $multiply
- $natural
- $ne
- $not
- $object ToArray
- $or
- $out
- $pow
- $push
- $project
- $rand
- $range
- $redact

- $reduce
- $regexFind
- $正则表达式 FindAll
- $regexMatch
- $replaceAll
- $replaceOne
- $replaceRoot
- $replaceWith
- $reverseArray
- $rtrim
- $sample
- $search
- $second
- $set
- $setDifference
- $setEquals
- $setIntersection
- $set IsSubset
- $setUnion
- $skip
- $slice
- $size
- $sort
- $split
- $sqrt
- $str LenBytes
- $strLenCP
- $strcasecmp
- $substr

- $substrBytes
- $substrCP
- $subtract
- $sum
- $switch
- $toBool
- $toDate
- $toDecimal
- $toDouble
- $toInt
- $toLong
- $toLower
- $to ObjectId
- $toString
- $toUpper
- $trim
- $type
- $unset
- $unwind
- $向量搜索
- $week
- $year
- $zip

$$DESCEND

Amazon DocumentDB 中的$$DESCEND运算符是管道阶段中使用的特殊位置数组运算符。$redact它指示聚合管道进入当前文档并处理所有字段，无论其嵌套级别如何。

当$redact舞台遇到$$DESCEND操作员时，它将保持当前文档中的所有字段可见，并在管道中进一步处理它们。当你想根据条件有选择地编辑或修剪某些字段，同时保留文档的结构时，这很有用。

参数

无。

示例（MongoDB 外壳）

在此示例中，我们将使用带有$$DESCEND运算符的$redact舞台来有选择地显示code字段等于 “Reg” 的文档。

创建示例文档

```
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" }
]);
```

查询示例

```
db.patient.aggregate([
  { $redact: {
    $cond: {
      if: { $eq: ["Reg", "$code"] },
      then: "$$DESCEND",
      else: "$$PRUNE"
    }
  }}
]);
```

输出

```
{
  "_id": 2,
  "code": "Reg",
  "patient": "Jane Doe",
  "DOB": "3/27/1989",
  "Hospital": "Cherry Hill"
}
```

代码示例

要查看使用该$$DESCEND命令的代码示例，请选择要使用的语言的选项卡：

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系统变量与聚合管道中的$redact阶段一起使用，以保持当前文档或字段不变，并将其包含在输出中。

参数

无

示例（MongoDB 外壳）

以下示例演示了在 Amazon DocumentDB 聚合管道$$KEEP中的用法。只有当访问权限等于 “公开” 时，才会保留文档，否则它们将被删除。

创建示例文档

```
db.articles.insertMany([
  { title: "Article A", access: "public", content: "Visible content" },
  { title: "Article B", access: "private", content: "Hidden content" }
]);
```

查询示例

```
db.articles.aggregate([
  {
    $redact: {
      $cond: [
        { $eq: ["$access", "public"] },
```

```
        "$$KEEP",
        "$$PRUNE"
      ]
    }
  }
]);
```

输出

```
[
  {
    "_id" : ObjectId("..."),
    "title" : "Article A",
    "access" : "public",
    "content" : "Visible content"
  }
]
```

代码示例

要查看使用该$$KEEP命令的代码示例，请选择要使用的语言的选项卡：

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

$$PRUNE系统变量与聚合管道中的$redact阶段一起使用，从结果中排除文档或嵌入式文档级别。当条件计算为时$$PRUNE，当前文档或子文档将从输出中删除。它通常与$$DESCEND（用于保留和遍历文档）或$$KEEP（将文档保持在所有级别）一起使用。

参数

无。$$PRUNE系统变量不带任何参数，必须与一起使用$redact。

示例（MongoDB 外壳）

以下示例演示如何$$PRUNE使用和将 30 岁$redact以上的用户排除在结果之外。

创建示例文档

```
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" } }
])
```

查询示例

```
db.users.aggregate([
  {
    $redact: {
      $cond: {
        if: { $gt: ["$age", 30] },
        then: "$$PRUNE",
        else: "$$DESCEND"
      }
    }
  }
])
```

输出

```
[
```

```
  {
    "_id": 2,
    "name": "Saanvi Sarkar",
    "age": 28,
    "address": {
      "street": "456 Oak Rd",
      "city": "Someplace",
      "state": "NY"
    }
  }
]
```

代码示例

要查看使用该$$PRUNE命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$ROOT运算符用于引用聚合管道中的整个输入文档。它允许您访问和操作完整的文档，包括其所有嵌套字段和结构。

参数

无

示例（MongoDB 外壳）

此示例演示$ROOT如何使用创建审核日志，该日志可捕获完整的原始文档以及有关文档处理时间的元数据。

创建示例文档

```
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
  }
]);
```

查询示例

```
db.orders.aggregate([
  {
    $project: {
      processedAt: new Date(),
      originalDocument: "$$ROOT",
      summary: {
        $concat: [
          "Order ",
          "$orderId",
          " for ",
          "$customer",
          " - Total: $",
          { $toString: "$totalAmount" }
        ]
      }
```

```
    }
  }
]);
```

输出

```
[
  {
    _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'
  }
]
```

代码示例

要查看使用该$ROOT命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$abs运算符返回数字的绝对值。它可以在聚合管道中用于对数值字段执行数学运算。

参数

- number：将返回其绝对值的数值表达式。

示例（MongoDB 外壳）

此示例演示如何使用$abs运算符来查找数值字段的绝对值。

创建示例文档

```
db.numbers.insertMany([
  { "_id": 1, "value": -5 },
  { "_id": 2, "value": 10 },
  { "_id": 3, "value": -3.14 },
  { "_id": 4, "value": 0 }
]);
```

查询示例

```
db.numbers.aggregate([
  { $project: {
    "_id": 1,
    "absolute_value": { $abs: "$value" }
  }}
]);
```

输出

```
[
  { "_id": 1, "absolute_value": 5 },
  { "_id": 2, "absolute_value": 10 },
  { "_id": 3, "absolute_value": 3.14 },
  { "_id": 4, "absolute_value": 0 }
]
```

代码示例

要查看使用该$abs命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$add

Amazon DocumentDB 中的$add运算符用于将数字或日期相加。它可用于对数值字段执行算术运算，或者通过向日期字段添加多个时间单位来执行日期算术。

参数

- expression1：要添加的第一个数字或日期。
- expression2：要添加的第二个数字或日期。
- expression3:（可选）要添加的其他数字或日期。

示例（MongoDB 外壳）

以下示例演示如何使用$add运算符将两个数字相加。

创建示例文档

```
db.numbers.insertMany([
  { _id: 1, a: 5, b: 3 },
  { _id: 2, a: 10, b: 7 },
  { _id: 3, a: 2, b: 8 }
]);
```

查询示例

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    sum: { $add: ["$a", "$b"] }
  }}
])
```

输出

```
[
  { "_id" : 1, "sum" : 8 },
  { "_id" : 2, "sum" : 17 },
  { "_id" : 3, "sum" : 10 }
]
```

在此示例中，$add运算符用于为每个文档添加a和b字段的值，并将结果存储在sum字段中。

代码示例

要查看使用该$add命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$add ToSet

$addToSet聚合运算符从每个组的指定表达式中返回一个由唯一值组成的数组。它在$group舞台中用于累积不同的值，从而自动消除重复值。

参数

- expression：要为组中的每个文档计算的表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$addToSet运算符收集为每个客户下单的独特城市。

创建示例文档

```
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 }
]);
```

查询示例

```
db.orders.aggregate([
  {
    $group: {
      _id: "$customer",
      cities: { $addToSet: "$city" }
    }
  }
]);
```

输出

```
[
  { _id: 'Bob', cities: [ 'Seattle', 'Boston' ] },
  { _id: 'Alice', cities: [ 'Seattle', 'Portland' ] }
]
```

代码示例

要查看使用$addToSet聚合运算符的代码示例，请选择要使用的语言的选项卡：

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([
    {
      $group: {
        _id: "$customer",
        cities: { $addToSet: "$city" }
      }
    }
  ]).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([
        {
            '$group': {
                '_id': '$customer',
                'cities': { '$addToSet': '$city' }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$addFields

Amazon DocumentDB 聚合管道中的$addFields阶段允许您向文档添加新的计算字段。这对于向文档中添加派生或转换后的数据非常有用。

参数

- newField：要添加的新字段的名称。
- expression：解析为新字段值的表达式。

示例（MongoDB 外壳）

以下示例演示$addFields如何使用添加新字段TotalInventory，该字段根据Inventory.OnHand和Inventory.OrderQnty字段计算总库存。

创建示例文档

```
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
  }
]);
```

查询示例

```
db.example.aggregate([
  {
    $addFields: {
      TotalInventory: { $add: ["$Inventory.OnHand", "$Inventory.OrderQnty"] }
    }
  }
])
```

输出

```
[
  {
    "_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
  }
]
```

代码示例

要查看使用该$addFields命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$all ElementsTrue

4.0 版的新增内容

该$allElementsTrue运算符用于检查数组中所有元素的计算结果是否为真值。

参数

- expression: 计算结果为数组的表达式。

示例（MongoDB 外壳）

以下示例演示了检查数组中所有元素是否为真时的用法。$allElementsTrue

创建示例文档

```
db.collection.insert([
  { "name": "John", "scores": [100, 90, 80] },
  { "name": "Jane", "scores": [80, 85, 0] },
  { "name": "Bob", "scores": [90, 95, null] }
])
```

查询示例

```
db.collection.find({
  "scores": { "$allElementsTrue": [{ "$gt": 0 }] }
})
```

输出

```
[
  { "_id" : ObjectId("..."), "name" : "John", "scores" : [ 100, 90, 80 ] },
  { "_id" : ObjectId("..."), "name" : "Bob", "scores" : [ 90, 95, null ] }
]
```

在此示例中，查询检查scores数组中的所有元素是否都大于 0。之所以排除带的"name": "Jane"文档，是因为该scores数组包含一个 0，这是一个虚假值。

代码示例

要查看使用该$allElementsTrue命令的代码示例，请选择要使用的语言的选项卡：

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聚合运算符计算多个表达式，并且true仅当所有表达式的计算结果均为true时才返回。如果有任何表达式false，则返回false。

参数

- expressions：要计算的表达式数组。

示例（MongoDB 外壳）

以下示例演示如何使用$and运算符来检查产品是否满足多个标准。

创建示例文档

```
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 }
]);
```

查询示例

```
db.products.aggregate([
  {
    $project: {
      name: 1,
      price: 1,
      inStock: 1,
      affordable: {
        $and: [
          { $lt: ["$price", 100] },
          { $eq: ["$inStock", true] }
        ]
      }
    }
  }
]);
```

输出

```
[
  { _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 }
]
```

代码示例

要查看使用$and聚合运算符的代码示例，请选择要使用的语言的选项卡：

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()
```

$any ElementTrue

4.0 版的新增内容

该$anyElementTrue运算符用于确定数组中是否有任何元素为真。

参数

- field: 要评估的数组字段。

示例（MongoDB 外壳）

以下示例演示了检查数$anyElementTrue组中是否有任何元素为真值的用法。

创建示例文档

```
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] }
])
```

查询示例

```
db.grades.aggregate([
  {
    $project: {
```

```
      student: 1,
      isAnyTrue: { $anyElementTrue: ["$scores"] },
      _id: 0
    }
  }
])
```

输出

```
[
  { student: 'Tim', isAnyTrue: false },
  { student: 'Bob', isAnyTrue: false },
  { student: 'Ivy', isAnyTrue: true }
]
```

代码示例

要查看使用该$anyElementTrue命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$array ElemAt

Amazon DocumentDB 中的$arrayElemAt运算符允许您按索引位置从数组中检索元素。当您需要访问文档中数组字段中的特定元素时，这特别有用。

参数

- array: 要从中检索元素的输入数组。
- index：要检索的元素从零开始的索引位置。此值必须是非负整数。

示例（MongoDB 外壳）

在此示例中，我们将演示如何使用$arrayElemAt运算符从miles集合中的flight_miles数组中检索特定元素。

创建示例文档

```
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 ]}
]);
```

查询示例

```
db.miles.aggregate([
  { $project: {
    "_id": 1,
    "first_mile": { $arrayElemAt: [ "$flight_miles", 0 ] },
    "last_mile": { $arrayElemAt: [ "$flight_miles", -1 ] }
  }}
]);
```

输出

```
{ "_id" : 1, "first_mile" : 1205, "last_mile" : 880 }
{ "_id" : 2, "first_mile" : 1205, "last_mile" : 2780 }
{ "_id" : 3, "first_mile" : 1205, "last_mile" : 880 }
```

在此示例中，我们使用$arrayElemAt运算符为每个文档检索flight_miles数组的第一个和最后一个元素。

代码示例

要查看使用该$arrayElemAt命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$array ToObject

Amazon DocumentDB 中的$arrayToObject运算符与运算符相反。$objectToArray它接受一组键值对文档并将其转换为单个文档。当你需要将一组键值对转换回对象或文档结构时，这特别有用。

参数

- array expression: 解析为数组的表达式。数组元素必须是包含两个字段的文档：k（键）和v（值）。

示例（MongoDB 外壳）

以下示例演示$arrayToObject如何使用将键值对数组转换回文档。

创建示例文档

```
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 } }
]);
```

查询示例

```
db.videos.aggregate([
  { $project: { name: 1, videos: { $objectToArray: "$inventory" } } },
  { $project: { name: 1, inventory: { $arrayToObject: "$videos" } } }
]);
```

输出

```
{ "_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 } }
```

在此示例中，$objectToArray运算符用于将inventory对象转换为键值对数组。然后使用该$arrayToObject运算符将数组转换回文档，从而恢复原始对象结构。

代码示例

要查看使用该$arrayToObject命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$avg

Amazon DocumentDB 中的$avg聚合运算符计算输入到舞台的文档中指定表达式的平均值。此运算符可用于计算一组文档中数值字段或表达式的平均值。

参数

- expression：用于计算平均值的表达式。这可以是字段路径（例如"$field"）或表达式（例如{ $multiply: ["$field1", "$field2"] }）。

示例（MongoDB 外壳）

以下示例演示如何使用$avg运算符计算一组学生文档的平均分数。

创建示例文档

```
db.students.insertMany([
  { name: "John", score: 85 },
  { name: "Jane", score: 92 },
  { name: "Bob", score: 78 },
  { name: "Alice", score: 90 }
]);
```

查询示例

```
db.students.aggregate([
  { $group: {
    _id: null,
    avgScore: { $avg: "$score" }
  }}
]);
```

输出

```
[
  {
    "_id": null,
    "avgScore": 86.25
  }
]
```

代码示例

要查看使用该$avg命令的代码示例，请选择要使用的语言的选项卡：

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');
  const result = await db.collection('students').aggregate([
    { $group: {
      _id: null,
      avgScore: { $avg: '$score' }
    }}
  ]).toArray();
  console.log(result);
  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
    result = list(db.students.aggregate([
        { '$group': {
            '_id': None,
            'avgScore': { '$avg': '$score' }
        }}
    ]))
    print(result)
    client.close()

calculate_avg_score()
```

$bucket

8.0 版的新增内容

弹性集群不支持。

Amazon DocumentDB 中的$bucket聚合阶段允许您根据指定的表达式和存储桶边界将输入文档分组到存储桶中。这对于分析属于特定值范围或类别的数据非常有用。

参数

- groupBy（必填）：指定分组依据的值的表达式。
- boundaries（必填）：定义存储桶边界的双精度值数组。根据位于指定边界内的groupBy表达式值将文档分配给存储桶。
- default（可选）：为groupBy表达式值不属于任何指定边界的文档输出的文字值。
- output（可选）：一个对象，它指定要为每个存储桶输出的信息。您可以使用诸如$sum、$avg$min、和之类的累加器运算符$max来计算每个存储桶的聚合。

示例（MongoDB 外壳）

以下示例演示如何使用$bucket舞台按价格区间对销售数据进行分组。

创建示例文档

```
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") }
]);
```

查询示例

```
db.sales.aggregate([
  {
    $bucket: {
      groupBy: "$price",
      boundaries: [0, 10, 20, 30],
      default: "Other",
      output: {
        "count": { $sum: 1 },
        "totalQuantity": { $sum: "$quantity" }
      }
    }
  },
  {
    $sort: { _id: 1 }
  }
])
```

输出

```
[
  { _id: 0, count: 1, totalQuantity: 3 },
  { _id: 10, count: 2, totalQuantity: 4 },
  { _id: 20, count: 2, totalQuantity: 2 }
]
```

代码示例

要查看使用该$bucket命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$ceil

4.0 版的新增内容

与 MongoDB 一样，Amazon DocumentDB 中的$ceil运算符将数字向上舍入到最接近的整数。当您需要对数值字段执行数学运算并确保结果为整数时，这很有用。

参数

- expression: 要向上舍入的数值表达式。

示例（MongoDB 外壳）

此示例演示如何使用$ceil运算符向上舍入数值字段。

创建示例文档

```
db.numbers.insertMany([
  { "_id": 1, "value": 3.14 },
  { "_id": 2, "value": -2.7 },
  { "_id": 3, "value": 0 }
])
```

查询示例

```
db.numbers.aggregate([
  { $project: {
    "roundedUp": { $ceil: "$value" }
  }}
])
```

输出

```
{ "_id": 1, "roundedUp": 4 }
{ "_id": 2, "roundedUp": -2 }
{ "_id": 3, "roundedUp": 0 }
```

代码示例

要查看使用该$ceil命令的代码示例，请选择要使用的语言的选项卡：

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

弹性集群不支持。

$changeStream聚合阶段会打开更改流游标，以监视集合的实时更改。当进行插入、更新、替换或删除操作时，它会返回更改事件文档。

参数

- `fullDocument`：指定是否返回完整文档以进行更新操作。选项有 `default` 或 `updateLookup`。
- `resumeAfter`: 可选。恢复令牌以从变更流中的特定点继续。

- `startAtOperationTime`: 可选。开始变更流的时间戳。
- `allChangesForCluster`: 可选。布尔值。何时`true`，监视集群中的所有更改（对于管理员数据库）。当`false`（默认）时，仅监视指定的集合。

示例（MongoDB 外壳）

以下示例演示了如何使用`$changeStream`舞台来监视集合的更改。

查询示例

```
// 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()));
}
```

输出

```
{
  _id: { _data: '...' },
  operationType: 'insert',
  clusterTime: Timestamp(1, 1234567890),
  fullDocument: { _id: 1, item: 'Widget', qty: 10 },
  ns: { db: 'test', coll: 'inventory' },
  documentKey: { _id: 1 }
}
```

代码示例

要查看使用`$changeStream`聚合阶段的代码示例，请选择要使用的语言对应的选项卡：

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()
```

$cmp

Amazon DocumentDB 中的$cmp运算符用于比较两个值并返回一个表示其相对顺序的整数值。它是一个比较运算符，它比较两个表达式并分别返回 -1、0 或 1 的整数值，具体取决于第一个值是小于、等于还是大于第二个值。

参数

- expression1：第一个要比较的表达式。
- expression2：第二个要比较的表达式。

示例（MongoDB 外壳）

以下示例演示如何使用运$cmp算符比较两个数值。

创建示例文档

```
db.collection.insertMany([
  { _id: 1, value1: 10, value2: 20 },
  { _id: 2, value1: 15, value2: 15 },
  { _id: 3, value1: 20, value2: 10 }
]);
```

查询示例

```
db.collection.find({
  $expr: {
    $cmp: ["$value1", "$value2"]
  }
})
```

输出

```
[
  { "_id" : 1, "value1" : 10, "value2" : 20 },
  { "_id" : 3, "value1" : 20, "value2" : 10 }
]
```

在此示例中，$cmp运算符比较每个文档的value1和value2字段。结果是：

```
- `$cmp: ["$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).
```

代码示例

要查看使用该$cmp命令的代码示例，请选择要使用的语言的选项卡：

Node.js

以下是在带有mongodb驱动程序的 Node.js 应用程序中使用$cmp运算符的示例：

```
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

以下是在带pymongo驱动程序的 Python 应用程序中使用$cmp运算符的示例：

```
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 和 Python 示例的输出将与 MongoDB 命令行管理程序示例相同：

```
[
  { "_id" : 1, "value1" : 10, "value2" : 20 },
  { "_id" : 2, "value1" : 15, "value2" : 15 },
  { "_id" : 3, "value1" : 20, "value2" : 10 }
]
```

$collStats

4.0 版的新增内容

Amazon DocumentDB 中的`$collStats`聚合阶段提供有关指定集合的统计信息，类似于 MongoDB 外壳中的`db.collection.stats()`命令。此阶段可用于检索有关集合的信息，例如文档数量、集合的总大小以及各种性能指标。

参数

- `latencyStats`: （可选）指定收集延迟统计信息的选项的文档。亚马逊文档数据库不支持此参数。
- `recordStats`: （可选）指定收集记录统计信息的选项的文档。亚马逊文档数据库不支持此参数。
- `queryExecStats`: （可选）一个文档，它指定了用于收集查询执行统计信息的选项。亚马逊文档数据库不支持此参数。

示例（MongoDB 外壳）

以下示例演示如何使用`$collStats`聚合阶段检索有关`db`数据库`test`中名为的集合的统计信息。

创建示例文档

```
db.test.insertMany([
  { "name": "John", "age": 30 },
  { "name": "Jane", "age": 25 },
  { "name": "Bob", "age": 35 }
]);
```

查询示例

```
db.test.aggregate([
  { $collStats: {} }
]);
```

输出

```
{
  "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)
}
```

代码示例

要查看使用该$collStats命令的代码示例，请选择要使用的语言的选项卡：

Node.js

以下是如何使用官方 MongoDB Node.js 驱动程序在 Node.js 应用程序中使用$collStats聚合阶段的示例：

```
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

以下是如何使用 PyMongo 驱动程序在 Python 应用程序中使用$collStats聚合阶段的示例：

```
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 DocumentDB 中的$concat聚合运算符连接（或组合）文档中的多个字符串，生成一个可以返回给应用程序的字符串。这减少了在应用程序中完成的工作，因为字符串操作是在数据库级别执行的。

参数

- expression1: 要连接的第一个字符串。
- expression2: 要连接的第二个字符串。
- ...：要连接的其他字符串（可选）。

示例（MongoDB 外壳）

在此示例中，我们将用户的名字和姓氏连接起来生成每个人的全名。

创建示例文档

```
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"}
])
```

查询示例

```
db.people.aggregate([
  { $project: { full_name: { $concat: [ "$first_name", " ", "$last_name"] } } }
])
```

输出

```
{ "_id" : 1, "full_name" : "Jane Doe" }
{ "_id" : 2, "full_name" : "John Doe" }
{ "_id" : 3, "full_name" : "Steve Smith" }
```

代码示例

要查看使用该$concat命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$concatArrays聚合运算符用于将两个或多个数组连接成一个数组。当您需要将多个数据数组组合成一个数组以进行进一步处理或分析时，这可能很有用。

参数

- array1: 第一个要连接的数组。
- array2: 要连接的第二个数组。
- [array3, ...]: （可选）要连接的其他数组。

示例（MongoDB 外壳）

以下示例演示如何使用$concatArrays运算符将两个数组合并为一个数组。

创建示例文档

```
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"]
  }
]);
```

查询示例

```
db.collection.aggregate([
  {
    $project: {
      _id: 0,
      name: 1,
      all_activities: { $concatArrays: ["$hobbies", "$skills"] }
    }
  }
]);
```

输出

```
[
  {
    "name": "John Doe",
    "all_activities": [
      "reading",
      "swimming",
      "programming",
      "design"
    ]
  },
  {
    "name": "Jane Smith",
    "all_activities": [
      "hiking",
      "cooking",
      "marketing",
      "analysis"
    ]
  }
]
```

代码示例

要查看使用该$concatArrays命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$cond运算符用于计算条件表达式并返回两个可能的结果表达式之一。

参数

- if: 要计算的布尔表达式。
- then：如果表达式为真则返回的if表达式。
- else：如果表达式为假则返回的if表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$cond运算符根据一个人的年龄返回值。

创建示例文档

```
db.people.insertMany([
  { _id: 1, name: "John Doe", age: 35 },
  { _id: 2, name: "Jane Doe", age: 25 },
  { _id: 3, name: "Bob Smith", age: 65 }
]);
```

查询示例

```
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"
            }
          }
        }
      }
    }
  }
])
```

输出

```
[
  { "_id" : 1, "name" : "John Doe", "ageGroup" : "middle-aged" },
  { "_id" : 2, "name" : "Jane Doe", "ageGroup" : "young" },
  { "_id" : 3, "name" : "Bob Smith", "ageGroup" : "elderly" }
]
```

代码示例

要查看使用该$cond命令的代码示例，请选择要使用的语言的选项卡：

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

4.0 版的新增内容

Amazon DocumentDB 中的$convert运算符用于将值从一种数据类型转换为另一种数据类型。当您需要对不同类型的数据执行操作时，此运算符很有用，例如将字符串转换为数字或将日期转换为时间戳。

参数

- to：要将值转换为的目标数据类型。支持的值有 "string"、"double"、"long"、"int"、"date" 和 "boolean"。
- from：该值的当前数据类型。如果未指定，Amazon DocumentDB 将尝试自动检测数据类型。
- onError:（可选）转换失败时要返回的值。可以是特定值或以下特殊值之一："null""zerofill"、或"error"。
- onNull:（可选）如果输入值为，则返回的值null。可以是特定值或以下特殊值之一："null""zerofill"、或"error"。

示例（MongoDB 外壳）

以下示例演示如何使用$convert运算符将字符串值转换为日期。

创建示例文档

```
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" }
]);
```

查询示例

```
db.users.aggregate([
  {
    $project: {
      _id: 1,
      name: 1,
      joinedOn: {
        $convert: {
          input: "$joinedOn",
          to: "date",
          onError: "null",
          onNull: "null"
        }
      }
    }
  }
])
```

输出

```
[
  { "_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 }
]
```

代码示例

要查看使用该$convert命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$count

Amazon DocumentDB 中的$count聚合阶段用于计算进入该阶段的文档数量。它通常用作聚合管道的最后一个阶段，用于返回与前几个阶段匹配的文档总数。

参数

- field：要计数的字段。此参数是可选的，如果未提供，则该阶段将计算输入文档的总数。

示例（MongoDB 外壳）

以下示例演示如何使用$count舞台获取集合中的文档总数。

创建示例文档

```
db.users.insertMany([
  { name: "John", age: 30 },
  { name: "Jane", age: 25 },
  { name: "Bob", age: 35 },
  { name: "Alice", age: 28 }
]);
```

查询示例

```
db.users.aggregate([
```

```
  { $count: "total" }
]);
```

输出

```
{ "total" : 4 }
```

该示例汇总了users集合，并使用$count舞台来计算文档的总数。

代码示例

要查看使用该$count命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$currentOp

$currentOp聚合阶段返回有关数据库中当前正在运行的操作的信息。此阶段对于监控聚合管道中的活动查询和操作非常有用。

参数

- allUsers（可选）：如果设置为true，则返回所有用户的操作。默认值为 false。
- idleConnections（可选）：设置为时true，包括空闲连接。默认值为 false。
- idleCursors（可选）：如果设置为true，则包含有关空闲光标的信息。默认值为 false。
- idleSessions（可选）：如果设置为true，则包含有关空闲会话的信息。默认值为 true。
- localOps（可选）：设置为时true，包括本地操作。默认值为 false。

示例（MongoDB 外壳）

以下示例演示如何使用$currentOp聚合阶段检索有关活动读取操作的信息。

查询示例

```
db.aggregate([
  { $currentOp: { allUsers: true, idleConnections: false } },
  { $match: { op: "query" } }
])
```

输出

```
[
  {
    "opid": "12345",
    "active": true,
    "op": "query",
    "ns": "test.users",
    "secs_running": 2
  }
]
```

代码示例

要查看使用该$currentOp命令的代码示例，请选择要使用的语言的选项卡：

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

5.0 版的新增内容

Amazon DocumentDB 中的$dateAdd聚合运算符允许您在日期和时间值中添加持续时间。

参数

- date：用于添加持续时间的日期和时间值。
- duration：要添加到该date值中的持续时间。可以将其指定为带有years、、、、monthsweeksdayshoursminutes、和键的对象seconds。
- timezone:（可选）执行日期添加时要使用的时区。如果未指定，则使用 Amazon DocumentDB 集群的默认时区。

示例（MongoDB 外壳）

以下示例演示如何使用$dateAdd运算符为日期添加 2 天零 12 小时。

创建示例文档

```
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") }
```

```
]);
```

查询示例

```
db.events.aggregate([
  {
    $project: {
      _id: 1,
      eventDate: 1,
      eventDatePlustwodaysandtwelvehours: {
        $dateAdd: {
          startDate: {
            $dateAdd: {
              startDate: "$eventDate",
              unit: "day",
              amount: 2
            }
          },
          unit: "hour",
          amount: 12
        }
      }
    }
  }
])
```

输出

```
[
  {
    "_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)"
```

```
  }
]
```

代码示例

要查看使用该$dateAdd命令的代码示例，请选择要使用的语言的选项卡：

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

5.0 版的新增内容

弹性集群不支持。

$dateDiff聚合运算符以指定单位计算两个日期之间的差。它返回开始日期和结束日期之间跨越单位边界的数量。

参数

- startDate：开始日期表达式。
- endDate：结束日期表达式。
- unit: 差异的时间单位。支持的单位是year、quarter、month、week、day、hour、minute、second、和millisecond。

示例（MongoDB 外壳）

以下示例演示如何使用$dateDiff运算符来计算从下单到交货之间的天数。

创建示例文档

```
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")
  }
]);
```

查询示例

```
db.shipments.aggregate([
  {
    $project: {
```

```
      orderId: 1,
      orderDate: 1,
      deliveryDate: 1,
      daysToDeliver: {
        $dateDiff: {
          startDate: "$orderDate",
          endDate: "$deliveryDate",
          unit: "day"
        }
      }
    }
  }
]);
```

输出

```
[
  {
    _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
  }
]
```

代码示例

要查看使用该$dateDiff命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$date FromString

Amazon DocumentDB 中的$dateFromString聚合运算符允许您将日期时间字符串转换为日期对象。当您的数据存储为字符串但需要对数据执行基于日期的操作时，这很有用。

参数

- dateString：表示日期和时间的字符串。
- format:（可选）指定格式的字符串dateString。如果未提供，Amazon DocumentDB 将尝试按该格式解析字符串。 ISO-8601
- timezone:（可选）指定时区的字符串。如果未提供，Amazon DocumentDB 将使用服务器的时区。
- onError:（可选）指定转换失败时要采取的操作。可能的值是'error'（默认值，会引发错误）、'null'（返回null）或'replace'（用onErrorMessage选项中指定的替换字符串替换值）。
- onErrorMessage:（可选）如果设置onError为'replace'，则此选项指定替换字符串。

示例（MongoDB 外壳）

以下示例演示$dateFromString如何使用在 Amazon DocumentDB 中将日期字符串转换为日期对象。

创建示例文档

```
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"}
]);
```

查询示例

```
db.missionLog.aggregate([
  {
    $project: {
      event: '$event',
      logDate: {
        $dateFromString: {
          dateString: '$logDate'
        }
      }
    }
  }
]);
```

输出

```
[
  {
    "_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")
  }
]
```

代码示例

要查看使用该$dateFromString命令的代码示例，请选择要使用的语言的选项卡：

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

5.0 版的新增内容

Amazon DocumentDB 中的$dateSubtract聚合运算符允许您从日期值中减去指定的持续时间。

参数

- date：解析为日期或时间戳的日期表达式。
- subtrahend：一个持续时间表达式，它指定要从表达date式中减去的时间长度。
- unit：一个字符串，它指定subtrahend表达式的时间单位。支持的单位为 “年”、“季度”、“月”、“周”、“天”、“小时”、“分钟”、“秒” 和 “毫秒”。

示例（MongoDB 外壳）

以下示例演示如何使用$dateSubtract运算符从当前日期开始计算一年前的日期。

创建示例文档

```
db.events.insertOne({
  eventName: "Player joined",
  eventTime: ISODate("2023-04-01T12:00:00Z")
});
```

查询示例

```
db.events.aggregate([
  {
    $project: {
      eventName: 1,
      oneYearAgo: {
        $dateSubtract: {
          startDate: "$eventTime",
          amount: 1,
          unit: "year"
        }
      }
    }
  }
])
```

输出

```
{
  "_id" : ObjectId("64567890abcdef012345678"),
```

```
  "eventName" : "Player joined",
  "oneYearAgo" : ISODate("2022-04-01T12:00:00Z")
}
```

代码示例

要查看使用该$dateSubtract命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$date ToString

Amazon DocumentDB 中的$dateToString聚合运算符用于将日期或时间戳值转换为字符串表示形式。当您需要以特定方式格式化日期和时间以进行显示或进一步处理时，这很有用。

参数

- date：要转换为字符串的日期或时间戳值。
- format：一个字符串，它指定应以何种格式表示日期。格式字符串可以包含各种格式说明符，例如%Y四位数年份、%m两位数月份、%d两位数月份的日期等。
- timezone: （可选）用于转换的时区。如果未指定，则使用托管 Amazon DocumentDB 集群的服务器的时区。

- onNull:（可选）如果date参数为，则返回的值null。

示例（MongoDB 外壳）

以下示例演示了如何使用运$dateToString算符来格式化missionLog集合的logDate字段。

创建示例文档

```
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") }
]);
```

查询示例

```
db.missionLog.aggregate([
  {
    $project: {
      event: "$event",
      logDateFormatted: {
        $dateToString: {
          format: "%Y-%m-%d %H:%M:%S",
          date: "$logDate"
        }
      }
    }
  }
])
```

输出

```
[
  {
    "_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"
  }
]
```

代码示例

要查看使用该$dateToString命令的代码示例，请选择要使用的语言的选项卡：

Node.js

以下是在 Node.js 应用程序中使用$dateToString运算符的示例：

```
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

以下是在 Python 应用程序中使用$dateToString运算符的示例：

```
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

8.0 版的新增内容

弹性集群不支持。

Amazon DocumentDB 中的$dateTrunc聚合运算符将日期截断为指定单位。

参数

- date：解析为日期或时间戳的日期表达式。
- unit：一个字符串，它为子集表达式指定时间单位。支持的单位是year、quarter、month、week、day、hour、minute、second、和millisecond。

示例（MongoDB 外壳）

以下示例演示如何使用$dateTrunc运算符将日期截断为小时。

创建示例文档

```
db.events.insertMany([
  {
    eventName: "Event 1",
    eventTime: ISODate("2025-04-01T12:15:00Z")
  },
  {
    eventName: "Event 2",
    eventTime: ISODate("2025-08-15T14:33:22Z")
  },
]);
```

查询示例

```
db.events.aggregate([
  {
    $project: {
```

```
      eventName: 1,
      eventTime: 1,
      truncatedToHour: {
        $dateTrunc: {
          date: "$eventTime",
          unit: "hour"
        }
      }
    }
  }
]);
```

输出

```
[
  {
    _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')
  }
]
```

代码示例

要查看使用该$dateTrunc命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$day OfMonth

Amazon DocumentDB 中的$dayOfMonth聚合运算符检索给定日期的当月中的某一天（从 1 到 31）。此运算符可用于对文档中的日期字段进行分组、筛选或提取当月中的某天。

参数

- `date expression`：日期表达式可以是文档中的日期字段、日期对象或日期字符串。

示例（MongoDB 外壳）

此示例演示如何使用$dayOfMonth运算符从文档的日期字段中提取月份中的某一天。

创建示例文档

```
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") }
]);
```

查询示例

```
db.events.aggregate([
  { $project: {
    eventDay: { $dayOfMonth: "$eventDate" }
  }}
])
```

输出

```
[
  { "_id" : 1, "eventDay" : 15 },
  { "_id" : 2, "eventDay" : 28 },
  { "_id" : 3, "eventDay" : 1 },
  { "_id" : 4, "eventDay" : 30 }
]
```

代码示例

要查看使用该$dayOfMonth命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$day OfWeek

Amazon DocumentDB 中的$dayOfWeek运算符从给定的日期字段中提取一周中的某一天。它以 1（星期日）和 7（星期六）之间的数字返回一周中的某一天，这与 MongoDB 中的行为相同。

参数

- date field: 要从中提取星期几的日期字段。

示例（MongoDB 外壳）

此示例演示如何使用$dayOfWeek运算符从weather集合中的date字段中提取星期几。

创建示例文档

```
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")
  }
]);
```

查询示例

```
db.weather.aggregate([
  {
    $project: {
      dayOfWeek: { $dayOfWeek: "$date" }
    }
  }
]).pretty();
```

输出

```
{ "_id" : ObjectId("64272c6663f4f8ce422c2d91"), "dayOfWeek" : 7 }
{ "_id" : ObjectId("64272c6663f4f8ce422c2d92"), "dayOfWeek" : 1 }
{ "_id" : ObjectId("64272c6663f4f8ce422c2d93"), "dayOfWeek" : 2 }
```

代码示例

要查看使用该$dayOfWeek命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$day OfYear

Amazon DocumentDB 中的$dayOfYear运算符以 1 到 366 之间的数字返回日期的一年中的某一天（非闰年为 365）。

参数

- `expression`：要从中提取一年中的某天的日期字段或表达式。

示例（MongoDB 外壳）

此示例演示如何使用`$dayOfYear`运算符从 Amazon DocumentDB 集合中的日期字段中提取一年中的某一天。

创建示例文档

```
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")
  }
])
```

查询示例

```
db.weather.aggregate([
  {
    $project: {
      dayOfYear: { $dayOfYear: "$date" }
    }
  }
]).pretty()
```

输出

```
{ "_id" : ObjectId("642b86fc7d8e07af279bbe63"), "dayOfYear" : 74 }
{ "_id" : ObjectId("642b86fc7d8e07af279bbe64"), "dayOfYear" : 365 }
```

代码示例

要查看使用该`$dayOfYear`命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$divide

Amazon DocumentDB 聚合管道中的$divide运算符用于将一个数字除以另一个数字。它是对文档中的数值字段执行数学运算的有用运算符。

参数

- numerator：股息或要分割的数字。
- denominator: 除数或要除以的数字。

示例（MongoDB 外壳）

此示例演示如何使用$divide运算符根据员工的年薪和每年的工作时数计算员工的小时工资。

创建示例文档

```
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 }
]);
```

查询示例

```
db.employees.aggregate([
  {
    $project: {
      name: 1,
      hourlyRate: { $divide: ["$salary", "$hoursPerYear"] }
    }
  }
])
```

输出

```
[
  { "_id" : 1, "name" : "John Doe", "hourlyRate" : 38.46153846153846 },
  { "_id" : 2, "name" : "Jane Smith", "hourlyRate" : 43.26923076923077 },
  { "_id" : 3, "name" : "Bob Johnson", "hourlyRate" : 36.05769230769231 }
]
```

代码示例

要查看使用该$divide命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$eq

$eq聚合运算符比较两个值，true如果它们相等，则返回，否则返回false。

参数

- expression1：要比较的第一个值。
- expression2: 第二个要比较的值。

示例（MongoDB 外壳）

以下示例演示如何使用$eq运算符来检查产品数量是否与目标值匹配。

创建示例文档

```
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 }
]);
```

查询示例

```
db.inventory.aggregate([
  {
    $project: {
      item: 1,
      qty: 1,
      target: 1,
      meetsTarget: { $eq: ["$qty", "$target"] }
    }
  }
]);
```

输出

```
[
  { _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 }
]
```

代码示例

要查看使用$eq聚合运算符的代码示例，请选择要使用的语言的选项卡：

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

4.0 版的新增内容

Amazon DocumentDB 中的$exp运算符允许您将常数 e 提高到给定的数字。

参数

- expression：要计算的表达式。这可以是任何有效的聚合表达式，包括字段引用、算术运算和其他聚合阶段。

示例（MongoDB 外壳）

以下示例演示如何使用$exp运算符来查找quantity字段大于该price字段的所有文档。

创建示例文档

```
db.items.insertMany([
  { item: "canvas", quantity: 4 },
  { item: "journal", quantity: 2 }
]);
```

查询示例

```
db.items.aggregate([
  { $project: {
    "quantityRaised": {$exp: "$quantity"}}
  }
]);
```

输出

```
[
  {
    _id: ObjectId('6920b785311cf98b79d2950d'),
    quantityRaised: 54.598150033144236
  },
  {
    _id: ObjectId('6920b785311cf98b79d2950e'),
    quantityRaised: 7.38905609893065
  }
]
```

代码示例

要查看使用该$exp命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$filter

Amazon DocumentDB 中的$filter运算符用于将筛选表达式应用于数组的每个元素，并返回一个仅包含符合指定条件的元素的数组。当您需要对文档中的数组字段执行复杂的筛选操作时，此运算符很有用。

参数

- input: 要筛选的数组字段。
- as：用于cond表达式中数组中每个input元素的变量名。
- cond: 布尔表达式，用于确定是否应将给定元素包含在输出数组中。

示例（MongoDB 外壳）

以下示例演示如何使用$filter运算符来预测每个订单的客户，并创建一个新的数组字段 paidItems，其中仅包含价格大于 15 的 items 数组中的商品。从本质上讲，它会筛选每个订单的商品，仅包括价格超过15的产品。

创建示例文档

```
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 }
  ]}
]);
```

查询示例

```
db.orders.aggregate([
  {
    $project: {
      customer: 1,
      paidItems: {
        $filter: {
          input: "$items",
          as: "item",
          cond: { $gt: ["$$item.price", 15] }
        }
      }
    }
  }
]).pretty();
```

输出

```
[
  {
    _id: 1,
    customer: 'abc123',
    paidItems: [ { name: 'Product B', price: 20, qty: 1 } ]
  },
  { _id: 2, customer: 'def456', paidItems: [] },
  { _id: 3, customer: 'ghi789', paidItems: [] }
]
```

代码示例

要查看使用该$filter命令的代码示例，请选择要使用的语言的选项卡：

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

5.0 版的新增内容。

弹性集群不支持。

Amazon DocumentDB 中的$first操作员返回一组分组文档中的第一个文档。它通常在聚合管道中用来检索与特定条件匹配的第一个文档。

参数

- expression：作为每个组中第一个值返回的表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$first运算符检索聚合期间每个类别遇到的第一个项目值。

注意：根据管道中文档的当前顺序$first返回第一个文档。为了确保特定的顺序（例如按日期、价格等），应在$sort舞台之前使用$group舞台。

创建示例文档

```
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" }
]);
```

查询示例

```
db.products.aggregate([
  { $group: { _id: "$category", firstItem: { $first: "$item" } } }
]);
```

输出

```
[
  { "_id" : "food", "firstItem" : "abc" },
  { "_id" : "toy", "firstItem" : "xyz" }
]
```

代码示例

要查看使用该$first命令的代码示例，请选择要使用的语言的选项卡：

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('products');
```

```
    const result = await collection.aggregate([
      { $group: { _id: "$category", firstItem: { $first: "$item" } } }
    ]).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['products']

        result = list(collection.aggregate([
            { '$group': { '_id': '$category', 'firstItem': { '$first': '$item' } } }
        ]))

        pprint(result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()
```

```
example()
```

$floor

4.0 版的新增内容。

Amazon DocumentDB 中的$floor运算符返回小于或等于指定数字的最大整数。此运算符对于向下舍入数值很有用。

参数

- expression: 要向下舍入的数值表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$floor运算符将十进制值向下舍入到最接近的整数。

创建示例文档

```
db.numbers.insertOne({ value: 3.14 });
```

查询示例

```
db.numbers.aggregate([
  { $project: { _id: 0, floored: { $floor: "$value" } } }
]);
```

输出

```
{ "floored" : 3 }
```

代码示例

要查看使用该$floor命令的代码示例，请选择要使用的语言的选项卡：

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聚合阶段按接近指定点的顺序返回文档。它计算与该点的距离，并将该距离包括在输出文档中。

参数

- near：计算距离的起点，指定为 geoJSON 或传统坐标。
- distanceField：用于存储计算距离的字段名称。
- spherical: 表示是否使用球面几何图形的布尔值（GeoJSON 点是必需的）。
- maxDistance: 可选。距离中心点的最大距离。
- minDistance: 可选。距离中心点的最小距离。
- query: 可选。要应用的其他筛选条件。
- limit: 可选。要返回的最大文档数。
- key: 可选。存在多个地理空间索引时用于地理空间查询的字段。

示例（MongoDB 外壳）

以下示例演示如何使用$geoNear舞台查找离给定位置最近的门店。

创建示例文档

```
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] } }
]);
```

查询示例

```
db.stores.aggregate([
  {
    $geoNear: {
      near: { type: "Point", coordinates: [-122.4, 37.8] },
      distanceField: "distance",
      spherical: true
    }
  }
]);
```

输出

```
[
  { _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 }
]
```

代码示例

要查看使用$geoNear聚合阶段的代码示例，请选择要使用的语言对应的选项卡：

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()
```

$group

Amazon DocumentDB 中的$group聚合阶段允许您按指定的表达式对文档进行分组，并对分组的数据执行各种累积操作。这对于诸如根据分组数据计算总计、平均值或其他统计数据之类的任务非常有用。

参数

- _id：指定对输入文档进行分组时应采用的表达式。这可以是字段名、计算表达式或两者的组合。
- accumulator expressions:（可选）应应用于分组数据的一个或多个累加器表达式。这些表达式使用上面提到的累加器运算符。

示例（MongoDB 外壳）

以下示例按城市对客户进行分组，并计算每个城市的总订单金额。

创建示例文档

```
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 }] }
]);
```

查询示例

```
db.customers.aggregate([
  {
    $group: {
      _id: "$city",
      totalOrders: { $sum: { $sum: "$orders.amount" } }
    }
  }
```

```
]);
```

输出

```
[
  { _id: 'Chicago', totalOrders: 150 },
  { _id: 'Los Angeles', totalOrders: 450 },
  { _id: 'New York', totalOrders: 500 }
]
```

代码示例

要查看使用该$group命令的代码示例，请选择要使用的语言的选项卡：

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聚合运算符比较两个值，true如果第一个值大于第二个值，则返回，否则返回false。

参数

- expression1：要比较的第一个值。
- expression2: 第二个要比较的值。

示例（MongoDB 外壳）

以下示例演示如何使用$gt运算符来识别超过价格阈值的产品。

创建示例文档

```
db.products.insertMany([
  { _id: 1, name: "Laptop", price: 1200 },
  { _id: 2, name: "Mouse", price: 25 },
  { _id: 3, name: "Monitor", price: 400 }
]);
```

查询示例

```
db.products.aggregate([
  {
    $project: {
      name: 1,
      price: 1,
      expensive: { $gt: ["$price", 100] }
    }
  }
]);
```

输出

```
[
  { _id: 1, name: 'Laptop', price: 1200, expensive: true },
  { _id: 2, name: 'Mouse', price: 25, expensive: false },
  { _id: 3, name: 'Monitor', price: 400, expensive: true }
]
```

代码示例

要查看使用$gt聚合运算符的代码示例，请选择要使用的语言的选项卡：

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聚合运算符比较两个值，true如果第一个值大于或等于第二个值，则返回，否则返回false。

参数

- expression1：要比较的第一个值。
- expression2: 第二个要比较的值。

示例（MongoDB 外壳）

以下示例演示如何使用$gte运算符来检查学生是否通过了考试。

创建示例文档

```
db.students.insertMany([
  { _id: 1, name: "Alice", score: 85 },
  { _id: 2, name: "Bob", score: 60 },
  { _id: 3, name: "Charlie", score: 72 }
]);
```

查询示例

```
db.students.aggregate([
  {
    $project: {
      name: 1,
      score: 1,
      passed: { $gte: ["$score", 70] }
    }
  }
]);
```

输出

```
[
  { _id: 1, name: 'Alice', score: 85, passed: true },
  { _id: 2, name: 'Bob', score: 60, passed: false },
  { _id: 3, name: 'Charlie', score: 72, passed: true }
]
```

代码示例

要查看使用$gte聚合运算符的代码示例，请选择要使用的语言的选项卡：

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

运$hour算符从日期或时间戳字段中提取小时部分。

参数

- dateExpression：运算符的应用日期。这必须解析为有效的 BSON 日期（例如，像 $createdAt 这样的字段或日期文字）。

也可以按以下格式将参数指定为文档：

{日期:<dateExpression>，时区：<timezoneExpression>}

这允许应用时区感知日期操作。

```
- `<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.
```

示例（MongoDB 外壳）

以下示例演示如何使用$hour运算符从日期字段中提取小时分量并相应地对数据进行分组。

创建示例文档

```
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") }
]);
```

查询示例

```
db.events.aggregate([
  {
    $project: {
      hour: { $hour: "$timestamp" }
    }
  },
  {
    $group: {
      _id: "$hour",
      count: { $sum: 1 }
    }
  },
  {
    $sort: { _id: 1 }
  }
]);
```

输出

```
[
  { "_id": 8, "count": 1 },
  { "_id": 10, "count": 1 },
  { "_id": 12, "count": 1 },
  { "_id": 16, "count": 1 },
  { "_id": 23, "count": 1 }
]
```

此查询按timestamp字段的小时部分对事件进行分组，并计算每小时的事件数。

代码示例

要查看使用该$hour命令的代码示例，请选择要使用的语言的选项卡：

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

如果输入表达式的计算结果为空或未定义，则该$ifNull运算符用于返回指定值。在您想要提供默认值或处理 null/undefined 案例的情况下，此运算符可能很有用。

参数

- expression：要计算的表达式。
- replacement：如果<expression>计算结果为 null 或未定义，则返回的值。

示例（MongoDB 外壳）

以下示例演示了当name字段为空或未定义时如何使用$ifNull运算符来提供默认值。

创建示例文档

```
db.users.insertMany([
  { _id: 1, name: "John" },
  { _id: 2, name: null },
  { _id: 3 }
]);
```

查询示例

```
db.users.aggregate([
  {
    $project: {
      _id: 1,
      name: { $ifNull: ["$name", "No Name"] }
    }
  }
]);
```

输出

```
[
  { "_id": 1, "name": "John" },
  { "_id": 2, "name": "No Name" },
  { "_id": 3, "name": "No Name" }
]
```

代码示例

要查看使用该$ifNull命令的代码示例，请选择要使用的语言的选项卡：

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聚合运算符检查数组中是否存在指定值。true如果在数组中找到该值，则返回，false否则返回。

参数

- value：要搜索的值。
- array: 要在其中搜索的数组。

示例（MongoDB 外壳）

以下示例演示如何使用$in操作员来检查每位员工的技能组合中是否存在特定技能。

创建示例文档

```
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"] }
]);
```

查询示例

```
db.employees.aggregate([
  {
    $project: {
      name: 1,
      hasPython: { $in: ["Python", "$skills"] }
    }
  }
]);
```

输出

```
[
  { _id: 1, name: 'Sarah', hasPython: true },
  { _id: 2, name: 'Mike', hasPython: false },
  { _id: 3, name: 'Emma', hasPython: true }
]
```

代码示例

要查看使用$in聚合运算符的代码示例，请选择要使用的语言的选项卡：

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()
```

$index OfArray

Amazon DocumentDB 中的$indexOfArray运算符用于查找数组中指定元素首次出现的索引。此运算符返回数组中第一个与指定值匹配的元素的从零开始的索引位置。如果未找到该值，则返回 -1。

参数

- `array`: 要搜索的数组。
- `value`：要在数组中搜索的值。
- `start`: (可选) 数组中开始搜索的位置。默认值是 0。

示例（MongoDB 外壳）

以下示例演示如何使用 $index OfArray 运算符查找每个文档在 “fruits” 数组中首次出现的元素 “mango” 的索引。

创建示例文档

```
db.collection.insertMany([
  { _id: 1, fruits: ["apple", "banana", "cherry", "durian"] },
  { _id: 2, fruits: ["mango", "orange", "pineapple"] },
  { _id: 3, fruits: ["kiwi", "lemon", "mango"] }
]);
```

查询示例

```
db.collection.aggregate([
  {
    $project: {
      _id: 1,
      fruitIndex: { $indexOfArray: ["$fruits", "mango"] }
    }
  }
]);
```

输出

```
{ "_id" : 1, "fruitIndex" : 1 }
{ "_id" : 2, "fruitIndex" : 0 }
{ "_id" : 3, "fruitIndex" : 2 }
```

代码示例

要查看使用该`$indexOfArray`命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$index OfBytes

Amazon DocumentDB 中的 $index OfBytes 运算符用于根据字符的字节位置查找字符串中子字符串的起始索引。这在处理可能包含多字节字符（例如非拉丁语脚本中的字符）的文本数据时非常有用。

参数

- `string`: 要搜索的输入字符串。
- `substring`: 要在输入字符串中搜索的子字符串。
- `[<start>]`:（可选）搜索的起始位置（从零开始）。如果未指定，则从字符串的开头开始搜索。

示例（MongoDB 外壳）

以下示例演示如何使用在一组`$indexOfBytes`表示办公桌位置的字符串中查找第一个连字符的索引。

创建示例文档

```
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" }
]);
```

查询示例

```
db.people.aggregate([
  { $project: { stateLocation: { $indexOfBytes: ["$Desk", "-"] } } }
]);
```

输出

```
{ "_id" : 1, "stateLocation" : 11 }
{ "_id" : 2, "stateLocation" : 6 }
{ "_id" : 3, "stateLocation" : 7 }
{ "_id" : 4, "stateLocation" : 8 }
```

代码示例

要查看使用该$indexOfBytes命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$indexOfCP运算符用于以码点 (CP) 形式查找字符串表达式中指定子字符串首次出现的索引。这在解析和提取字符串字段中的内容时非常有用。

参数

- string expression: 要搜索的字符串。
- substring: 要搜索的子字符串。
- [<start>]: （可选）开始搜索的位置（从零开始的索引）。默认值为 0。

示例（MongoDB 外壳）

在此示例中，我们使用$indexOfCP运算符在每个文档的 Desk 字段中查找首次出现的连字符的索引。

创建示例文档

```
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"}
])
```

查询示例

```
db.people.aggregate([
  { $project: { stateLocation: { $indexOfCP: [ "$Desk", "-"] } } }
])
```

输出

```
{ "_id" : 1, "stateLocation" : 10 }
{ "_id" : 2, "stateLocation" : 6 }
{ "_id" : 3, "stateLocation" : 7 }
{ "_id" : 4, "stateLocation" : 8 }
```

代码示例

要查看使用该$indexOfCP命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$indexStats聚合阶段可以深入了解集合中索引的使用情况。此运算符允许您监控索引的访问模式，这可以帮助您就索引管理和优化做出明智的决策。

参数

无

示例（MongoDB 外壳）

以下示例演示如何使用$indexStats运算符来分析 Amazon DocumentDB 集合中的索引使用情况。

创建示例文档

```
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 }
]);
```

查询示例

```
db.grocery.aggregate([
  { $indexStats: {} }
]);
```

输出

```
[
  {
    "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")
    }
  }
]
```

在此示例中，$indexStats运算符显示自上次重置或服务器重启以来，product_1索引已被访问 5 次，索引已被访问 10 次。_id_

代码示例

要查看使用该$indexStats命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$isArray运算符用于检查文档中的字段是否为数组。此运算符在聚合管道和条件表达式中可用于处理数组类型字段。

参数

- field：用于检查它是否为数组的字段路径。

示例（MongoDB 外壳）

此示例演示如何使用$isArray运算符来识别 “清单” 字段为数组的文档。

创建示例文档

```
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}]}
]);
```

查询示例

```
db.videos.aggregate([
  {
    $match: {
      $isArray: "$inventory"
    }
  },
  {
    $project: {
      _id: 1,
      name: 1,
      "inventory.location": 1,
      "inventory.count": 1
    }
  }
]).pretty();
```

输出

```
{
  "_id": 4,
  "name": "Bravemind",
  "inventory": [
    {
      "location": "Mason City",
      "count": 250
    },
    {
      "location": "Des Moines",
      "count": 1000
    },
    {
      "location": "Ames",
      "count": 500
    }
  ]
}
```

代码示例

要查看使用该$isArray命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$iso DayOfWeek

Amazon DocumentDB 中的$isoDayOfWeek运算符以整数值形式返回某一日期的 ISO 一周中的某一天。ISO 周日期系统定义每周从星期一开始，到星期日结束，第 1 周是包含一年中第一个星期四的那一周。

参数

- expression：要为其返回 ISO 一周中的某天的日期表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$isoDayOfWeek运算符检索一组事件文档的 ISO 一周中的某一天。

创建示例文档

```
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") }
]);
```

查询示例

```
db.events.aggregate([
  { $project: {
    _id: 1,
    eventDate: 1,
    isoDayOfWeek: { $isoDayOfWeek: "$eventDate" }
  }}
]);
```

输出

```
[
  { "_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 }
]
```

代码示例

要查看使用该`$isoDayOfWeek`命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$isoWeek运算符返回某个日期的 ISO 周数。ISO 周日期系统是一种对一年中的几周进行编号的方式，其中新年的第一周是包含该年第一个星期四的那一周。这与公历不同，在公历中，新年的第一周是包含1月1日的那一周。

参数

无

示例（MongoDB 外壳）

以下示例演示如何使用$isoWeek运算符检索给定日期的 ISO 周数。

创建示例文档

```
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") }
])
```

查询示例

```
db.dates.aggregate([
  {
    $project: {
      _id: 1,
      isoWeek: { $isoWeek: "$date" }
    }
  }
])
```

输出

```
[
```

```
  { "_id": 1, "isoWeek": 52 },
  { "_id": 2, "isoWeek": 52 },
  { "_id": 3, "isoWeek": 1 }
]
```

代码示例

要查看使用该$isoWeek命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$iso WeekYear

Amazon DocumentDB 中的$isoWeekYear运算符返回给定日期的 ISO 8601 周年号。ISO 周年号与公历年的不同之处在于，周年可能与日历年不同，尤其是在年初和年底。

参数

- expression：要返回 ISO 8601 周年号的日期表达式。

示例（MongoDB 外壳）

此示例演示如何使用$isoWeekYear运算符检索事件集合中每个文档的日期字段的 ISO 8601 周年。

创建示例文档

```
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") }
]);
```

查询示例

```
db.events.aggregate([
```

```
  { $project: {
    name: 1,
    isoWeekYear: { $isoWeekYear: "$date" }
  }}
]);
```

输出

```
[
  { "_id" : 1, "name" : "Event 1", "isoWeekYear" : 2023 },
  { "_id" : 2, "name" : "Event 2", "isoWeekYear" : 2023 },
  { "_id" : 3, "name" : "Event 3", "isoWeekYear" : 2023 }
]
```

代码示例

要查看使用该$isoWeekYear命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$last运算符用于返回数组中符合查询条件的最后一个元素。它对于检索数组中满足特定条件的最新元素或最后一个元素特别有用。

参数

- expression: 匹配数组元素的表达式。

示例（MongoDB 外壳）

以下示例演示了如何将$last运算符与$filter结合使用，从满足特定条件的数组中检索最后一个元素（例如，主题是 “科学”）。

创建示例文档

```
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 }
    ]
  }
]);
```

查询示例

```
db.collection.aggregate([
    { $match: { name: "John" } },
    {
      $project: {
        name: 1,
        lastScienceScore: {
          $last: {
            $filter: {
              input: "$scores",
              as: "score",
              cond: { $eq: ["$$score.subject", "science"] }
            }
          }
```

```
        }
      }
    }
  ]);
```

输出

```
[
  {
    _id: 1,
    name: 'John',
    lastScienceScore: { subject: 'science', score: 90 }
  }
]
```

代码示例

要查看使用该$last命令的代码示例，请选择要使用的语言的选项卡：

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: { name: "John" } },
    {
      $project: {
        name: 1,
        lastScienceScore: {
          $last: {
            $filter: {
              input: "$scores",
              as: "score",
              cond: { $eq: ["$$score.subject", "science"] }
            }
```

```
          }
        }
      }
    }
  ]).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.collection

    pipeline = [
        { "$match": { "name": "John" } },
        {
            "$project": {
                "name": 1,
                "lastScienceScore": {
                    "$last": {
                        "$filter": {
                            "input": "$scores",
                            "as": "score",
                            "cond": { "$eq": ["$$score.subject", "science"] }
                        }
                    }
                }
            }
        }
    ]
```

```
    result = list(collection.aggregate(pipeline))

    print(result)

    client.close()

example()
```

$let

Amazon DocumentDB 中的$let运算符用于将变量绑定到值并在表达式中使用这些变量。它允许您定义局部变量，这些变量可以在聚合管道的同一阶段的后续表达式中使用。

参数

- vars：定义要在表达式中使用的变量的对象。
- in：使用 vars 参数中定义的变量的表达式。

示例（MongoDB 外壳）

此示例演示如何使用运$let算符来计算矩形的面积。

创建示例文档

```
db.shapes.insertMany([
  { name: "Rectangle 1", length: 5, width: 3 },
  { name: "Rectangle 2", length: 7, width: 4 },
  { name: "Rectangle 3", length: 6, width: 2 }
]);
```

查询示例

```
db.shapes.aggregate([
  {
    $project: {
      name: 1,
      area: {
        $let: {
          vars: {
            length: "$length",
```

```
            width: "$width"
          },
          in: {
            $multiply: ["$$length", "$$width"]
          }
        }
      }
    }
  }
])
```

输出

```
[
  {
    "_id": ObjectId("6161e5b1a3eba3c7f2960d03"),
    "name": "Rectangle 1",
    "area": 15
  },
  {
    "_id": ObjectId("6161e5b1a3eba3c7f2960d04"),
    "name": "Rectangle 2",
    "area": 28
  },
  {
    "_id": ObjectId("6161e5b1a3eba3c7f2960d05"),
    "name": "Rectangle 3",
    "area": 12
  }
]
```

代码示例

要查看使用该$let命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$limit运算符用于限制查询返回的文档数量。它与 M $limit ongoDB 运算符类似，但是在将其与亚马逊 DocumentDB 一起使用时需要考虑一些特定的注意事项。

在 Amazon DocumentDB 中，$limit运算符对于分页很有用，您想检索所有匹配文档的子集。它允许您控制每次响应中返回的文档数量，从而提高性能并减少通过网络传输的数据量。

参数

- limit：要返回的最大文档数。这必须是一个非负整数值。

示例（MongoDB 外壳）

以下示例演示如何使用$limit运算符返回最多一个与给定过滤器匹配的文档。

创建示例文档

```
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" }
]);
```

查询示例

```
db.test.createIndex({ comments: "text" });

db.test.find({
  $and: [
    { star_rating: 5 },
    { $text: { $search: "couch" } }
  ]
}).limit(1);
```

输出

```
[ { _id: 2, star_rating: 5, comments: 'comfortable couch' } ]
```

代码示例

要查看使用该$limit命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$literal

Amazon DocumentDB 中的$literal运算符用于表示聚合管道阶段内的文字值。它允许您包含特定值，例如数字、字符串或布尔值，而无需将其解释为字段引用或表达式。

当您需要将文字值作为更复杂的聚合管道的一部分时，例如在构建动态查询筛选器或执行计算时，此运算符特别有用。

参数

无

示例（MongoDB 外壳）

以下示例演示了如何使用$literal运算符在聚合管道中包含文字值。该$literal运算符用于将值 18 作为文字值包含在 $gt 表达式中。这允许聚合管道将年龄字段与字面值 18 进行比较，以确定该人是否为成年人。

创建示例文档

```
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" }
]);
```

查询示例

```
db.collection.aggregate([
```

```
  {
    $project: {
      name: 1,
      age: 1,
      city: 1,
      isAdult: { $gt: ["$age", { $literal: 18 }] }
    }
  }
]);
```

输出

```
[
  {
    "_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
  }
]
```

代码示例

要查看使用该$literal命令的代码示例，请选择要使用的语言的选项卡：

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

4.0 版的新增内容。

Amazon DocumentDB 中的$ln运算符计算指定数字的自然对数（以 e 为底）。它将数字的对数返回到以 e 为底的对数。

参数

- expression：要计算自然对数的数字。

示例（MongoDB 外壳）

以下示例演示如何使用运$log算符来计算数字的自然对数。

创建示例文档

```
db.numbers.insertMany([
  { _id: 1, value: 10 },
  { _id: 2, value: 100 },
  { _id: 3, value: 1000 }
]);
```

查询示例

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    naturalLog: { $ln: "$value" }
  }}
]);
```

输出

```
[
  { "_id" : 1, "naturalLog" : 2.302585092994046 },
  { "_id" : 2, "naturalLog" : 4.605170185988092 },
  { "_id" : 3, "naturalLog" : 6.907755278982137 }
]
```

代码示例

要查看使用该$ln命令的代码示例，请选择要使用的语言的选项卡：

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

4.0 版的新增内容。

Amazon DocumentDB 中的$log运算符计算数字的自然对数。它返回指定数字的以 e 为底的对数。

参数

- `expression`：要计算自然对数的数字。
- `base`：计算对数的基值。

示例（MongoDB 外壳）

以下示例演示如何使用运$log算符来计算数字的自然对数。

创建示例文档

```
db.numbers.insertMany([
  { _id: 1, value: 10 },
  { _id: 2, value: 100 },
  { _id: 3, value: 1000 }
]);
```

查询示例

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    naturalLog: { $log: ["$value", 10] }
  }}
]);
```

输出

```
[
  { "_id" : 1, "naturalLog" : 1 },
  { "_id" : 2, "naturalLog" : 2 },
  { "_id" : 3, "naturalLog" : 2.9999999999999996 }
]
```

代码示例

要查看使用该$log命令的代码示例，请选择要使用的语言的选项卡：

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

4.0 版的新增内容。

Amazon DocumentDB 中的$log10运算符用于计算数字以 10 为底的对数。它对于对聚合管道中的数值字段执行对数计算很有用。

参数

- expression：要计算以 10 为底的对数的数值表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$log10运算符计算数值字段以 10 为底的对数。

创建示例文档

```
db.numbers.insertMany([
  { _id: 1, value: 1 },
  { _id: 2, value: 10 },
  { _id: 3, value: 100 },
  { _id: 4, value: 1000 }
]);
```

查询示例

```
db.numbers.aggregate([
  {
    $project: {
      _id: 1,
      log10Value: { $log10: "$value" }
    }
  }
]);
```

输出

```
[
  { "_id": 1, "log10Value": 0 },
  { "_id": 2, "log10Value": 1 },
  { "_id": 3, "log10Value": 2 },
  { "_id": 4, "log10Value": 3 }
]
```

代码示例

要查看使用该$log10命令的代码示例，请选择要使用的语言的选项卡：

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聚合运算符比较两个值，true如果第一个值小于第二个值，则返回，否则返回false。

参数

- expression1：要比较的第一个值。
- expression2: 第二个要比较的值。

示例（MongoDB 外壳）

以下示例演示如何使用$lt运算符识别低库存商品。

创建示例文档

```
db.warehouse.insertMany([
  { _id: 1, item: "Bolts", stock: 5 },
  { _id: 2, item: "Nuts", stock: 25 },
  { _id: 3, item: "Screws", stock: 8 }
]);
```

查询示例

```
db.warehouse.aggregate([
  {
    $project: {
      item: 1,
      stock: 1,
      lowStock: { $lt: ["$stock", 10] }
    }
  }
]);
```

输出

```
[
  { _id: 1, item: 'Bolts', stock: 5, lowStock: true },
  { _id: 2, item: 'Nuts', stock: 25, lowStock: false },
  { _id: 3, item: 'Screws', stock: 8, lowStock: true }
]
```

代码示例

要查看使用$lt聚合运算符的代码示例，请选择要使用的语言的选项卡：

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`聚合运算符比较两个值，`true`如果第一个值小于或等于第二个值，则返回，否则返回`false`。

参数

- `expression1`：要比较的第一个值。
- `expression2`: 第二个要比较的值。

示例（MongoDB 外壳）

以下示例演示如何使用$lte运算符来识别经济实惠的项目。

创建示例文档

```
db.menu.insertMany([
  { _id: 1, dish: "Salad", price: 8 },
  { _id: 2, dish: "Pasta", price: 12 },
  { _id: 3, dish: "Soup", price: 6 }
]);
```

查询示例

```
db.menu.aggregate([
  {
    $project: {
      dish: 1,
      price: 1,
      affordable: { $lte: ["$price", 10] }
    }
  }
]);
```

输出

```
[
  { _id: 1, dish: 'Salad', price: 8, affordable: true },
  { _id: 2, dish: 'Pasta', price: 12, affordable: false },
  { _id: 3, dish: 'Soup', price: 6, affordable: true }
]
```

代码示例

要查看使用$lte聚合运算符的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$lookup聚合阶段允许您在两个集合之间执行左外连接。此操作允许您根据匹配的字段值合并来自多个集合的数据。当您需要将来自相关集合的数据合并到查询结果中时，它特别有用。

参数

- from：用于执行连接的集合的名称。
- localField：输入文档中要与之匹配的字段foreignField。
- foreignField: from 集合中文档中要与之匹配的字段localField。
- as：要添加到包含from集合中匹配文档的输出文档的新字段的名称。

示例（MongoDB 外壳）

以下示例演示了将集合中的数据联接到orders集合中的customers简单$lookup操作。

创建示例文档

```
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 }
]);
```

查询示例

```
db.customers.aggregate([
```

```
  {
    $lookup: {
      from: "orders",
      localField: "_id",
      foreignField: "customer_id",
      as: "orders"
    }
  }
]);
```

输出

```
[
  {
    _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 } ]
  }
]
```

代码示例

要查看使用该$lookup命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$ltrim

4.0 版的新增内容。

弹性集群不支持。

Amazon DocumentDB 中的$ltrim运算符用于从字符串中删除前导字符。默认情况下，它会删除前导空格字符，但您也可以通过传递 chars 参数来指定一组要删除的字符。

参数

- input：要从中删除前导空格字符的输入字符串。
- chars: (可选) 删除特定字符。

示例（MongoDB 外壳）

以下示例演示了从字符串开头删除指定字符 (” *”) 的用法。$ltrim

创建示例文档

```
db.collection.insertMany([
  { name: " *John Doe", age: 30 },
  { name: "Jane Doe*", age: 25 },
  { name: "  Bob Smith  ", age: 35 }
]);
```

查询示例

```
db.collection.aggregate([
  {
```

```
    $project: {
      _id: 0,
      name: {
        $ltrim: { input: "$name", chars: " *" }
      },
      age: 1
    }
  }
]);
```

输出

```
[
  { "name": "John Doe", "age": 30 },
  { "name": "Jane Doe ", "age": 25 },
  { "name": "Bob Smith  ", "age": 35 }
]
```

代码示例

要查看使用该$ltrim命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$map运算符允许您将指定的表达式应用于数组中的每个元素，并返回包含已转换元素的新数组。此运算符对于操作和转换数组中的数据特别有用，它可以将数组处理推到数据库级别，从而帮助简化应用程序代码并提高查询性能。

参数

- input: 要转换的数组。
- as: （可选）在 in 表达式中用于表示当前正在处理的元素的变量的名称。
- in: 要应用于输入数组中每个元素的表达式。

示例（MongoDB 外壳）

以下示例演示如何使用 $map 运算符来转换数字数组，将每个值加倍。

创建示例文档

```
db.collection.insertMany([
  { _id: 1, numbers: [1, 2, 3, 4, 5] },
  { _id: 2, numbers: [10, 20, 30, 40, 50] }
```

```
])
```

查询示例

```
db.collection.aggregate([
  {
    $project: {
      doubledNumbers: { $map: { input: "$numbers", as: "num", in: { $multiply: ["$
$num", 2] } } }
    }
  }
])
```

输出

```
[
  { _id: 1, doubledNumbers: [2, 4, 6, 8, 10] },
  { _id: 2, doubledNumbers: [20, 40, 60, 80, 100] }
]
```

代码示例

要查看使用该$map命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$match管道阶段用于筛选输入文档，仅筛选符合指定查询条件的文档。它是聚合操作中最常用的流水线阶段之一。该$match阶段在任何其他管道阶段之前应用，使您可以有效地减少后续阶段需要处理的文档数量。

参数

- query: 一份表达行动选择标准的文件。查询文档使用与find()方法相同的语法。

示例（MongoDB 外壳）

以下示例演示如何使用$match舞台根据特定字段值筛选文档。

创建示例文档

```
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" }
]);
```

查询示例

```
db.collection.aggregate([
  { $match: { age: { $gt: 30 } } },
  { $project: { _id: 1, name: 1, city: 1 } }
]);
```

输出

```
[
  { "_id": 3, "name": "Bob", "city": "Chicago" },
  { "_id": 4, "name": "Alice", "city": "Miami" }
]
```

该$match阶段会筛选文档，使其仅包括age字段大于 30 的文档。

代码示例

要查看使用该$match命令的代码示例，请选择要使用的语言的选项卡：

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聚合阶段用于返回管道阶段中所有文档中指定字段的最大值。此运算符对于查找一组文档中的最大值很有用。

参数

- expression：用于计算最大值的表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$max运算符在一组学生文档中查找最高分数。该$group阶段将所有文档组合在一起，$max运算符用于计算所有文档中该score字段的最大值。

创建示例文档

```
db.students.insertMany([
  { name: "John", score: 85 },
  { name: "Jane", score: 92 },
  { name: "Bob", score: 78 },
  { name: "Alice", score: 90 }
])
```

查询示例

```
db.students.aggregate([
  { $group: { _id: null, maxScore: { $max: "$score" } } },
  { $project: { _id: 0, maxScore: 1 } }
])
```

输出

```
[ { maxScore: 92 } ]
```

代码示例

要查看使用该$max命令的代码示例，请选择要使用的语言的选项卡：

Node.js

```
const { MongoClient } = require('mongodb');

async function findMaxScore() {
  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 students = db.collection('students');

  const result = await students.aggregate([
    { $group: { _id: null, maxScore: { $max: "$score" } } }
  ]).toArray();

  console.log(result);
  await client.close();
}

findMaxScore();
```

Python

```
from pymongo import MongoClient

def find_max_score():
    client = MongoClient('mongodb://<username>:<password>@<cluster-endpoint>:27017/?
tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0&readPreference=secondaryPreferred&retryWrites=false')
    db = client['test']
    students = db.students

    result = list(students.aggregate([
        { "$group": { "_id": None, "maxScore": { "$max": "$score" } } }
    ]))

    print(result)
    client.close()

find_max_score()
```

$meta

$meta聚合运算符访问与聚合管道中的文档关联的元数据。它通常用于检索文本搜索分数和按相关性对结果进行排序。

参数

- textScore：检索文本搜索分数，该分数表示文档与搜索查询的相关性。

示例（MongoDB 外壳）

以下示例演示了在聚合管道中使用$meta运算符来检索文本搜索分数并按文本搜索分数进行排序。

创建示例文档

```
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." }
]);
```

查询示例

```
db.articles.aggregate([
  { $match: { $text: { $search: "Python" } } },
  { $addFields: { score: { $meta: "textScore" } } },
  { $sort: { score: -1 } }
]);
```

输出

```
[
  {
    _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
  }
]
```

代码示例

要查看使用$meta聚合运算符的代码示例，请选择要使用的语言的选项卡：

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

在 8.0 中引入

Amazon DocumentDB 中的$merge聚合阶段用于将前一个管道阶段的结果合并到目标集合中。这对于根据输入文档中的数据在目标集合中更新或插入文档非常有用。

该$merge阶段允许您根据输入文档和目标集合之间的匹配条件执行各种操作，例如：

```
- Insert new documents
- Update existing documents
- Delete documents
- Fail the operation if there are any conflicts
```

参数

- into:（必填）要将输入文档合并到的目标集合的名称。
- on:（必填）用作输入文档和目标集合之间匹配条件的字段。
- whenMatched:（可选）当输入文档与目标集合中的现有文档相匹配时要执行的操作。支持的值为："merge"、"replace"、"keepExisting" 和 "fail"。
- whenNotMatched:（可选）当输入文档与目标集合中的任何文档都不匹配时要执行的操作。支持的值为："insert"和"fail"。

示例（MongoDB 外壳）

以下示例演示如何使用$merge舞台使用来自输入管道的新数据更新users集合。

创建示例文档

```
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" }
]);
```

查询示例

```
db.inputData.aggregate([
  {
    $merge: {
      into: "users",
      on: "_id",
      whenMatched: "merge",
      whenNotMatched: "insert"
    }
  }
])
```

输出

运行$merge管道后，该users集合将包含以下文档：

```
[
  { _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" }
]
```

代码示例

要查看使用该$merge命令的代码示例，请选择要使用的语言的选项卡：

Node.js

以下是在 Node.js 应用程序中使用 $merge 运算符的示例：

```
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

以下是在 Python 应用程序中使用 $merge 运算符的示例：

```
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 DocumentDB 中的$mergeObjects运算符用于将多个文档或对象合并为一个文档。当您需要合并两个或多个文档或对象的内容时，此运算符特别有用，这可能会将一个对象的值与另一个对象的值覆盖。

参数

- expression1: 第一个要合并的对象。
- expression2: （可选）要合并的第二个对象。
- expression3: （可选）要合并的其他对象。

示例（MongoDB 外壳）

以下示例演示如何使用$mergeObjects运算符组合两个对象。

创建示例文档

```
db.collection.insertMany([
  { "_id": 1, "name": "John", "address": { "city": "New York", "state": "NY" } },
  { "_id": 2, "name": "Jane", "address": { "city": "Los Angeles", "state": "CA" } }
]);
```

查询示例

```
db.collection.aggregate([
  {
    $project: {
      "combinedAddress": {
        $mergeObjects: ["$address", { "country": "USA" }]
      }
```

```
    }
  }
])
```

输出

```
[
  {
    "_id": 1,
    "combinedAddress": {
      "city": "New York",
      "state": "NY",
      "country": "USA"
    }
  },
  {
    "_id": 2,
    "combinedAddress": {
      "city": "Los Angeles",
      "state": "CA",
      "country": "USA"
    }
  }
]
```

代码示例

要查看使用该$mergeObjects命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$millisecond运算符用于提取日期值的毫秒部分。

参数

无

示例（MongoDB 外壳）

此示例演示如何使用$millisecond运算符提取日期值的毫秒部分。

创建示例文档

```
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")
  }
])
```

查询示例

```
db.events.aggregate([
  {
    $project: {
      name: 1,
      milliseconds: { $millisecond: "$timestamp" }
    }
  }
])
```

输出

```
[
  {
    "_id": ObjectId("644332a42054ed1b0d15f0c1"),
    "name": "Event 1",
    "milliseconds": 123
```

```
  },
  {
    "_id": ObjectId("644332a42054ed1b0d15f0c2"),
    "name": "Event 2",
    "milliseconds": 456
  },
  {
    "_id": ObjectId("644332a42054ed1b0d15f0c3"),
    "name": "Event 3",
    "milliseconds": 789
  }
]
```

代码示例

要查看使用该$millisecond命令的代码示例，请选择要使用的语言的选项卡：

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

运$min算符返回值数组中的最小值。它可以在聚合阶段用于在多个文档中查找指定字段的最小值。

参数

- expression：要计算的表达式。这可以是字段路径、变量或任何解析为值的表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$min运算符在多个文档中查找该age字段的最小值。

创建示例文档

```
db.users.insertMany([
  { name: "John", age: 35 },
  { name: "Jane", age: 28 },
  { name: "Bob", age: 42 },
```

```
  { name: "Alice", age: 31 }
]);
```

查询示例

```
db.users.aggregate([
  { $group: { _id: null, minAge: { $min: "$age" } } },
  { $project: { _id: 0, minAge: 1 } }
])
```

输出

```
[ { minAge: 28 } ]
```

代码示例

要查看使用该$min命令的代码示例，请选择要使用的语言的选项卡：

Node.js

```
const { MongoClient } = require('mongodb');

async function findMinAge() {
  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([
    { $group: {
        _id: null,
        minAge: { $min: "$age" }
      }}
  ]).toArray();

  console.log(result);
  client.close();
}

findMinAge();
```

Python

```
from pymongo import MongoClient

def find_min_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.aggregate([
        { "$group": {
            "_id": None,
            "minAge": { "$min": "$age" }
        }}
    ]))

    print(result)
    client.close()

find_min_age()
```

$minute

Amazon DocumentDB 中的$minute聚合管道阶段从日期或时间戳字段中提取分钟值。

当您需要在聚合管道中执行基于日期和时间的计算或分组时，此运算符很有用。

参数

- expression：要从中提取分钟值的日期或时间戳字段。

示例（MongoDB 外壳）

以下示例演示如何使用$minute运算符按从时间戳字段中提取的分钟值对文档进行分组，并计算每组中的文档数量。

创建示例文档

```
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") }
]);
```

查询示例

```
db.events.aggregate([
  {
    $group: {
      _id: {
        minute: { $minute: "$timestamp" }
      },
      count: { $count: {} }
    }
  },
  { $sort: { "_id.minute": 1 } }
]);
```

输出

```
[
  { "_id": { "minute": 30 }, "count": 2 },
  { "_id": { "minute": 31 }, "count": 2 },
  { "_id": { "minute": 32 }, "count": 1 }
]
```

代码示例

要查看使用该$minute命令的代码示例，请选择要使用的语言的选项卡：

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

运算$mod符是一种对数字执行模运算的算术运算符。它返回一个数字除以另一个数的余数。此运算符通常用于确定一个数字是奇数还是偶数，或者将项目分配到有限数量的组中。

参数

- expression1: 股息表达式。
- expression2: 除数表达式。

示例（MongoDB 外壳）

此示例演示如何使用$mod运算符来确定在 100 个包裹中配送时剩余小部件的数量。

创建示例文档

```
db.widgets.insertMany([
  { "_id" : 1, "widget" : "A", "count" : 80372 },
```

```
  { "_id" : 2, "widget" : "B", "count" : 409282 },
  { "_id" : 3, "widget" : "C", "count" : 60739 }
])
```

查询示例

```
db.widgets.aggregate([
  { $addFields: { leftOver: { $mod: [ "$count", 100 ] } } }
])
```

输出

```
[
  { "_id" : 1, "widget" : "A", "count" : 80372, "leftOver" : 72 },
  { "_id" : 2, "widget" : "B", "count" : 409282, "leftOver" : 82 },
  { "_id" : 3, "widget" : "C", "count" : 60739, "leftOver" : 39 }
]
```

输出显示每个文档的剩余部分count除以 100，这表示以 100 的包裹配送时剩余的小部件数量。

代码示例

要查看使用该$mod命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$month运算符以 1 到 12 之间的数字返回日期的月份。此运算符对于从日期字段中提取月份组成部分以及执行基于日期的聚合和分析非常有用。

参数

- `date_expression`：这是包含要从中提取月份的日期或时间戳的表达式或字段。

示例（MongoDB 外壳）

以下示例演示如何使用$month运算符从日期字段中提取月份并按月对数据进行分组。

创建示例文档

```
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") }
]);
```

查询示例

```
db.sales.aggregate([
  { $group: {
      _id: { month: { $month: "$date" } },
      totalSales: { $sum: "$price" }
    }},
  { $sort: { "_id.month": 1 } }
]);
```

输出

```
[
  { _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 }
]
```

代码示例

要查看使用该$month命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$multiply运算符用于将两个或多个字段或表达式的值相乘。此运算符对于对文档中的数值字段执行算术运算特别有用。它可以用于聚合管道的各个阶段，例如$project和$addFields，以创建新字段或修改现有字段。

参数

- expression1：第一个要相乘的数值表达式。
- expression2：第二个要相乘的数值表达式。
- [expression3, ...]:（可选）要相乘的其他数值表达式。

示例（MongoDB 外壳）

以下示例演示了$multiply使用信用卡bonus_rate进行行程的客户使用bonus_miles乘以base_miles和进行计算。

示例文档

```
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 }
]);
```

查询示例

```
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"]
      }
    }
  }
```

```
]);
```

输出

```
[
  { _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 }
]
```

代码示例

要查看使用该$multiply命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$natural运算符用于按自然顺序对文档进行排序，也就是文档被插入集合的顺序。这与默认的排序行为形成鲜明对比，默认排序行为是根据指定字段的值对文档进行排序。

参数

无

示例（MongoDB 外壳）

以下示例演示如何使用$natural运算符按自然顺序对集合中的文档进行排序。

创建示例文档

```
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 }
]);
```

查询示例

```
db.people.find({}, { "_id": 1, "name": 1 }).sort({ "$natural": 1 });
```

输出

```
[
  { "_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" }
]
```

该查询按自然顺序（即插入顺序）对集合中的文档进行排序。

代码示例

要查看使用该$natural命令的代码示例，请选择要使用的语言的选项卡：

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聚合运算符比较两个值，true如果两个值不相等，则返回，否则返回false。

参数

- expression1：要比较的第一个值。

- expression2: 第二个要比较的值。

示例（MongoDB 外壳）

以下示例演示如何使用$ne运算符来识别状态发生变化的订单。

创建示例文档

```
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" }
]);
```

查询示例

```
db.orders.aggregate([
  {
    $project: {
      orderId: 1,
      status: 1,
      expectedStatus: 1,
      needsAttention: { $ne: ["$status", "$expectedStatus"] }
    }
  }
]);
```

输出

```
[
  { _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 }
]
```

代码示例

要查看使用$ne聚合运算符的代码示例，请选择要使用的语言的选项卡：

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聚合运算符对表达式执行逻辑 NOT 运算。true如果表达式的计算结果为false，如果表达式的计算结果为，false则返回。true

参数

- expression: 要否定的表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$not运算符反转布尔值。

创建示例文档

```
db.users.insertMany([
  { _id: 1, name: "Alice", active: true },
  { _id: 2, name: "Bob", active: false },
  { _id: 3, name: "Charlie", active: true }
]);
```

查询示例

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      active: 1,
```

```
      inactive: { $not: ["$active"] }
    }
  }
]);
```

输出

```
[
  { _id: 1, name: 'Alice', active: true, inactive: false },
  { _id: 2, name: 'Bob', active: false, inactive: true },
  { _id: 3, name: 'Charlie', active: true, inactive: false }
]
```

代码示例

要查看使用$not聚合运算符的代码示例，请选择要使用的语言的选项卡：

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()
```

$object ToArray

Amazon DocumentDB 中的$objectToArray聚合运算符将对象（或文档）转换为数组。运算符的输入是一个文档，输出由输入文档中每个字段值对的数组元素组成。当您需要将文档的各个字段作为数组处理时，例如要查找具有特定字段最大值或最小值的文档时，此运算符很有用。

参数

- expression: 要转换为数组的文档表达式。

示例（MongoDB 外壳）

以下示例演示了如何使用$objectToArray操作员来查找视频租赁连锁店的最大库存量的文档。

创建示例文档

```
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
    }
  }
]);
```

查询示例

```
db.videos.aggregate([
  {
    $project: {
```

```
      name: 1,
      videos: {
        $objectToArray: "$inventory"
      }
    }
  },
  {
    $unwind: "$videos"
  },
  {
    $group: {
      _id: "$name",
      maxInventory: {
        $max: "$videos.v"
      }
    }
  }
]);
```

输出

```
[
  {
    "_id": "Bravemind",
    "maxInventory": 1000
  },
  {
    "_id": "Live Soft",
    "maxInventory": 1000
  },
  {
    "_id": "Romancing the Rock",
    "maxInventory": 500
  },
  {
    "_id": "Top Pilot",
    "maxInventory": 1000
  }
]
```

代码示例

要查看使用该$objectToArray命令的代码示例，请选择要使用的语言的选项卡：

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聚合运算符计算多个表达式，true如果至少有一个表达式的计算结果为，则返回。truefalse仅当所有表达式均为时，它才会返回false。

参数

- expressions：要计算的表达式数组。

示例（MongoDB 外壳）

以下示例演示如何使用$or运算符来检查产品是否符合多个条件中的任何一个。

创建示例文档

```
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 }
]);
```

查询示例

```
db.items.aggregate([
  {
    $project: {
      name: 1,
      price: 1,
      onSale: 1,
      goodDeal: {
        $or: [
          { $lt: ["$price", 50] },
          { $eq: ["$onSale", true] }
        ]
      }
    }
  }
]);
```

输出

```
[
  { _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 }
]
```

代码示例

要查看使用$or聚合运算符的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$out运算符用于将聚合管道的结果写入指定的集合。

$out应该是最后一步。

参数

- output_collection：要写入聚合结果的输出集合的名称。

注意：如果集合已经存在，则会将其替换为聚合阶段的结果。

示例（MongoDB 外壳）

以下示例演示如何在 Amazon DocumentDB 中使用$out运算符将聚合管道的结果写入新集合。

创建示例文档

```
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 }
]);
```

查询示例

```
db.products.aggregate([
  { $group: { _id: "$category", totalPrice: { $sum: "$price" } } },
  { $out: "product_categories" }
])
```

输出

无（结果写入输出集合）。

聚合管道按类别对产品进行分组，并计算每个类别的商品总价。$out操作员将结果写入名为“product_categories” 的新集合。

要查看输出集合中的结果，请执行以下操作：

```
db.product_categories.find()
[
{ "_id" : "Books", "totalPrice" : 125 },
{ "_id" : "Electronics", "totalPrice" : 300 }
]
```

代码示例

要查看使用该$out命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$pow

在 8.0 中引入

Amazon DocumentDB 中的$pow运算符允许您将数字提高为次方。这对于在聚合管道中执行指数计算非常有用。

参数

- <number>（必填）：要提高为次的数字。
- <exponent>（必填）：应将数字提高到的次方。

示例（MongoDB 外壳）

以下示例演示如何使用$pow运算符计算数字的平方。

创建示例文档

```
db.numbers.insertMany([
  { "_id": 1, "value": 2 },
  { "_id": 2, "value": 3 },
  { "_id": 3, "value": 4 }
]);
```

查询示例

```
db.numbers.aggregate([
  { $addFields: { "square": { $pow: ["$value", 2] } } }
])
```

输出

```
[
  { "_id": 1, "value": 2, "square": 4 },
  { "_id": 2, "value": 3, "square": 9 },
  { "_id": 3, "value": 4, "square": 16 }
]
```

代码示例

要查看使用该$pow命令的代码示例，请选择要使用的语言的选项卡：

Node.js

以下是在 Node.js 应用程序中使用 $pow 运算符的示例：

```
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

以下是在 Python 应用程序中使用 $pow 运算符的示例：

```
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

$push聚合运算符为每个组返回一个包含指定表达式中所有值的数组。它通常在$group舞台中用于将值累积到数组中。

参数

- expression：要为组中的每个文档计算的表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$push运算符收集每个类别的所有产品名称。

创建示例文档

```
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 }
]);
```

查询示例

```
db.sales.aggregate([
  {
    $group: {
      _id: "$category",
      products: { $push: "$product" }
    }
  }
]);
```

输出

```
[
  { _id: 'Furniture', products: [ 'Desk', 'Chair' ] },
```

```
  { _id: 'Electronics', products: [ 'Laptop', 'Mouse', 'Keyboard' ] }
]
```

代码示例

要查看使用$push聚合运算符的代码示例，请选择要使用的语言的选项卡：

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('sales');

  const result = await collection.aggregate([
    {
      $group: {
        _id: "$category",
        products: { $push: "$product" }
      }
    }
  ]).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['sales']
```

```
    result = list(collection.aggregate([
        {
            '$group': {
                '_id': '$category',
                'products': { '$push': '$product' }
            }
        }
    ]))

    print(result)
    client.close()

example()
```

$project

Amazon DocumentDB 中的$project运算符允许您有选择地在输出文档中包含或排除字段，将值传递到下一个管道阶段，以及从输入文档值中计算新字段。

参数

- field: 要在输出文档中包含或排除的字段，它可以是字段路径（例如，“a.b.c”）。
- 1或true：在输出中包含该字段。
- 0或false：从输出中排除该字段。

示例（MongoDB 外壳）

以下示例演示了$project运算符在学生集合中的用法

创建示例文档

```
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" }
  ]);
```

此查询仅在输出中包含name和math字段。除非明确排除，否则默认情况下会包含该_id字段。

```
db.students.aggregate([
  { $project: { "name": 1, "math": 1 } }
])
```

输出

```
{ _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 }
```

此查询从输出中排除grade和_id字段，显示所有其他字段（name、math、science）。

```
db.students.aggregate([
  { $project: { "grade": 0, "_id": 0 } }
])
```

输出

```
{ 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 }
```

代码示例

要查看使用该$project命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$rand

8.0 版的新增内容

Amazon DocumentDB 中的$rand运算符用于生成 0 到 1 之间的随机数。

参数

无

示例（MongoDB 外壳）

以下示例演示如何使用$rand运算符从temp集合中随机选择两个文档。

创建示例文档

```
db.items.insertMany([
  { "name": "pencil", "quantity": 110 },
  { "name": "pen", "quantity": 159 }
])
```

查询示例

```
db.items.aggregate([
  {
    $project: {
      randomValue: { $rand: {} }
    }
  }
])
```

输出

```
[
  {
    _id: ObjectId('6924a5edd66dcae121d29517'),
    randomValue: 0.8615243955294392
  },
  {
    _id: ObjectId('6924a5edd66dcae121d29518'),
    randomValue: 0.22815483022099903
  }
]
```

代码示例

要查看使用该$rand命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$range聚合运算符用于创建指定范围内连续数字的数组。该运算符对于生成数字序列特别有用，例如比赛中援助站的英里标记，如以下示例所示。

参数

- start：范围的起始值。
- end：范围的结束值。
- step:（可选）生成范围时要使用的步长值。如果未提供，则默认步长值为 1。

示例（MongoDB 外壳）

在此示例中，我们将使用$range运算符生成自行车比赛中供水站的英里标记。

创建示例文档

```
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 }
]);
```

查询示例

```
db.races.aggregate([
  {
    $project: {
      race: 1,
      "waterStations": { $range: [20, "$distance", 20] }
    }
  }
]);
```

输出

```
[
  {
    _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 ] }
]
```

代码示例

要查看使用该$range命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$redact

Amazon DocumentDB 中的$redact聚合阶段用于根据指定字段的值有选择地在输出文档中包含或排除内容。这在需要根据访问级别或用户权限控制敏感数据可见性的场景中特别有用。

参数

- $cond：文档中每个字段的计算结果为$$KEEP$$PRUNE、或$$DESCEND的表达式。
- $$KEEP：保留输出文档中的当前字段。
- $$PRUNE：从输出文档中移除当前字段。
- $$DESCEND: 以递归方式将$redact舞台应用于当前字段，即对象或数组。

示例（MongoDB 外壳）

在此示例中，我们将使用$redact舞台根据订单的状态筛选订单，仅显示具有特定状态值的订单。

创建示例文档

```
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" }
])
```

查询示例

```
db.orders.aggregate([
  {
    $redact: {
      $cond: {
```

```
        if: { $in: ["$status", ["shipped", "processing"]] },
        then: "$$KEEP",
        else: "$$PRUNE"
      }
    }
  }
])
```

输出

```
[
  { _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' }
]
```

在此示例中，$redact舞台检查每个文档中status字段的值。如果status是 “已发货” 或 “处理中”，则保留文档 ($$KEEP)。否则，文档将被修剪 () $$PRUNE。

代码示例

要查看使用该$redact命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$reduce聚合运算符用于将包含两个参数的函数累积应用于数组的元素，以将数组缩减为单个值。对于在聚合管道中对数组数据执行复杂的计算或转换，此运算符特别有用。

参数

- input: 要缩减的数组。
- initialValue：要在还原操作中使用的初始值。
- in：要对input数组的每个元素进行评估的表达式。此表达式应返回一个值，该值将在下一次缩减迭代中使用。

示例（MongoDB 外壳）

以下示例演示如何使用$reduce运算符计算数组中所有元素的总和。

创建示例文档

```
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] }
])
```

查询示例

```
db.orders.aggregate([
  {
    $project: {
      total: {
        $reduce: {
          input: "$items",
          initialValue: 0,
          in: { $add: ["$$value", "$$this"] }
        }
      }
    }
  }
])
```

输出

```
[
  { "_id": 1, "total": 15 },
  { "_id": 2, "total": 60 },
```

```
  { "_id": 3, "total": 80 },
  { "_id": 4, "total": 300 }
]
```

运$reduce算符遍历items数组，将每个元素加到 0 中initialValue。结果是数组中所有元素的总和。

代码示例

要查看使用该$reduce命令的代码示例，请选择要使用的语言的选项卡：

Node.js

以下是在 Node.js 应用程序中使用$reduce运算符的示例：

```
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

以下是在 python 应用程序中使用$reduce运算符的示例：

```
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

5.0 版的新增内容。

弹性集群不支持。

Amazon DocumentDB 中的$regexFind运算符用于对文档中的字符串字段执行正则表达式匹配。它允许您搜索和提取与给定正则表达式模式匹配的特定子字符串。

参数

- input：要搜索的字符串字段或表达式。
- regex：要匹配的正则表达式模式。
- options:（可选）为正则表达式指定可选参数的对象，例如区分大小写和多行匹配。支持的选项有i（不区分大小写）和m（多行）。

示例（MongoDB 外壳）

以下示例演示如何使用$regexFind运算符搜索name字段与特定正则表达式模式匹配的文档。

创建示例文档

```
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" }
]);
```

查询示例

```
db.users.aggregate([
  {
    $project: {
      names: {
        $regexFind: { input: '$name', regex: 'j', options: 'i' }
      }
    }
  },
  { $match: {names: {$ne: null}}}
])
```

此查询将返回该name字段包含字母 “j”（不区分大小写）的所有文档。

输出

```
[
  { _id: 1, names: { match: 'J', idx: 0, captures: [] } }
]
```

注意：如果您的查询使用的是 Amazon DocumentDB 规划器版本 1，则必须使用提示才能使用索引。如果没有提示，查询可能会执行集合扫描。要查看您的计划程序版本并了解有关使用提示的更多信息，请参阅 [Amazon DocumentDB 查询规划器文档] () https://docs.aws.amazon.com/documentdb/latest/devguide/query-planner.html。

代码示例

要查看使用该`$regexFind`命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$正则表达式 FindAll

在 5.0 中引入

Amazon DocumentDB 中的$regexFindAll运算符用于对文档中的字符串字段执行正则表达式匹配。它允许您搜索和提取与给定正则表达式模式匹配的特定子字符串，返回正则表达式的所有匹配项。

参数

- input：要搜索的字符串字段或表达式。
- regex：要匹配的正则表达式模式。
- options:（可选）为正则表达式指定可选参数的对象，例如区分大小写和多行匹配。支持的选项有i（不区分大小写）和m（多行）。

示例（MongoDB 外壳）

以下示例演示如何使用$regexFindAll运算符从email字段中提取所有字母序列。

创建示例文档

```
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" }

]);
```

查询示例

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      emailMatches: {
        $regexFindAll: { input: '$email', regex: '[a-z]+', options: 'i' }
      }
    }
  }
])
```

输出

```
[
  {
    _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: [] }
    ]
  }
]
```

注意：如果您的查询使用的是 Amazon DocumentDB 规划器版本 1，则必须使用提示才能使用索引。如果没有提示，查询可能会执行集合扫描。要查看您的计划程序版本并了解有关使用提示的更多信息，请参阅 [Amazon DocumentDB 查询规划器文档] () https://docs.aws.amazon.com/documentdb/latest/devguide/query-planner.html。

代码示例

要查看使用该`$regexFindAll`命令的代码示例，请选择要使用的语言的选项卡：

Node.js

以下是在 Node.js 应用程序中使用`$regexFind`运算符的示例：

```
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

以下是在 Python 应用程序中使用$regexFind运算符的示例：

```
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

5.0 版的新增内容。弹性集群不支持。

Amazon DocumentDB 中的$regexMatch运算符用于对字符串字段执行正则表达式匹配。它返回一个布尔值（true或false），表示输入字符串是否与指定的模式匹配。

参数

- input: 要根据正则表达式进行测试的字符串。
- regex：要匹配的正则表达式模式。
- options: （可选）用于修改正则表达式行为的标志，例如不区分大小写的匹配 (i) 或多行匹配 ()。m

示例（MongoDB 外壳）

以下示例演示如何使用$regexMatch运算符来检查名称是否以字母 “M” 开头。运算符false为每个文档返回true或。

创建示例文档

```
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" },

]);
```

查询示例

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      startsWithM: {
        $regexMatch: {
          input: "$name",
          regex: "^M",
          options: "i"
        }
      }
    }
  }
]);
```

输出

```
{ _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 }
```

代码示例

要查看使用该$regexMatch命令的代码示例，请选择要使用的语言的选项卡：

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

在 5.0 中引入

Amazon DocumentDB 中的$replaceAll运算符用于将字段中所有出现的指定字符串模式替换为新字符串。此运算符可用于数据标准化、文本清理和字符串操作等任务。

参数

- input：包含要替换的字符串的字段或表达式。

- `find`：要搜索和替换的字符串模式。
- `replacement`: 用于替换匹配匹配项的字符串。

示例（MongoDB 外壳）

以下示例演示了如何在聚合管道中使用`$replaceAll`运算符将所有出现的字符串 “Chocolatier” 替换为 “Chocolatier Co.” 在 “产品” 系列的 “品牌名称” 字段中。

创建示例文档

```
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
  }
]);
```

查询示例

```
db.products.aggregate([
```

```
  {
    $addFields: {
      "brandName": {
        $replaceAll: {
          input: "$brandName",
          find: "Chocolatier",
          replacement: "Chocolate Co."
        }
      }
    }
  }
])
```

输出

```
[
  {
    _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
  }
]
```

代码示例

要查看使用该$replaceAll命令的代码示例，请选择要使用的语言的选项卡：

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

在 5.0 中引入

Amazon DocumentDB 中的运算符是聚合管道中使用的字符串表达式运算符，用于将字符串中第一次出现的指定子字符串替换为替换字符串。`$replaceOne`此运算符区分大小写，仅替换找到的第一个匹配项。

参数

- `input`：要执行查找的字符串（字段）。
- `find`: 要在输入中搜索的字符串。
- `replacement`: 用于替换输入（字段）中首次出现的查找结果的字符串。

示例（MongoDB 外壳）

以下示例演示了如何在聚合管道中使用`$replaceOne`运算符来替换产品名称中的子字符串。

创建示例文档

```
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 }
]);
```

聚合示例

```
db.products.aggregate([
  {
    $addFields: {
      standardizedName: {
        $replaceOne: {
          input: "$name",
          find: "Pack",
          replacement: "Package"
        }
      }
    }
  }
]);
```

输出

输出显示，每个产品名称中只有第一次出现的 “Pack” 被替换为 “Package”。

```
[
  {
    _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"
  }
```

代码示例

要查看使用该$replaceOne命令的代码示例，请选择要使用的语言的选项卡：

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

该$replaceRoot运算符用于将根文档替换为指定的嵌入式文档。当您想要将嵌套文档提升到顶层或重构数据输出时，这很有用。

参数

- newRoot：将替换现有根文档的新根文档。

示例（MongoDB 外壳）

此示例说明如何从买家订单中提取配送地址信息，这对于生成发货标签或地址列表非常有用。

创建示例文档

```
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 } ])
```

查询示例

```
db.orders.aggregate([
  {
    $replaceRoot: {
      newRoot: "$shippingAddress"
    }
  }
])
```

输出

```
{
    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'
  }
```

代码示例

要查看使用该$replaceRoot命令的代码示例，请选择要使用的语言的选项卡：

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

8.0 版的新增内容

弹性集群不支持。

Amazon DocumentDB 中的$replaceWith聚合阶段用于将输入文档替换为新文档。输入文档上的所有现有字段（包括 _id 字段）都将替换为新文档。 $replaceWith通常用于扁平化文档或将嵌入式文档提升到顶层。

参数

- <replacement>（必填）：将取代现有文档的新文档。

示例（MongoDB 外壳）

以下示例演示如何使用$replaceWith运算符替换 Amazon DocumentDB 集合中的现有文档。

创建示例文档

```
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 ]
  }
]);
```

查询示例

```
db.restaurants.aggregate([
  { $replaceWith: {
      name: "$name",
      cuisine: "$cuisine",
      rating: { $avg: "$ratings" }
    }
  }
]);
```

输出

```
[
  {
    name: 'Biryani Adda',
    cuisine: 'Indian',
    rating: 3.4
  },
  {
    name: 'The Burger Spot',
    cuisine: 'American',
    rating: 2.7777777777777777
  }
]
```

代码示例

要查看使用该$replaceWith命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$reverseArray运算符用于按指定顺序反转数组中的元素。当您需要按相反方向对数组的元素进行重新排序时，此运算符很有用。

参数

- expression: 要反转的数组表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$reverseArray运算符来反转数组中元素的顺序。

创建示例文档

```
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]}
]);
```

查询示例

```
db.miles.aggregate([
  {
    $project: {
      _id: 1,
```

```
      member_since: 1,
      credit_card: 1,
      reversed_flight_miles: { $reverseArray: "$flight_miles" }
    }
  }
]);
```

输出

```
{ "_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 ] }
```

在此示例中，$reverseArray运算符用于反转flight_miles数组的顺序。输出中的结果reversed_flight_miles字段按相反的顺序显示数组中的元素。

代码示例

要查看使用该$reverseArray命令的代码示例，请选择要使用的语言的选项卡：

Node.js

以下是在 Node.js 应用程序中使用$reverseArray运算符的示例：

```
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

以下是在 Python 应用程序中使用$reverseArray运算符的示例：

```
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()
```

$rtrim

4.0 版的新增内容。

弹性集群不支持。

Amazon DocumentDB 中的$rtrim运算符用于从字符串中删除尾随字符。默认情况下，它会删除尾随的空格字符，但您也可以通过传递 chars 参数来指定一组要删除的字符。

参数

- input: 要从中删除尾随空格字符的输入字符串。
- chars: （可选）删除特定字符。

示例（MongoDB 外壳）

以下示例演示如何使用$rtrim运算符从字符串中删除尾随的指定字符 (” *”)。

创建示例文档

```
db.collection.insert([
  { "name": "John Doe*  ", "age": 30 },
  { "name": "Jane Smith  ", "age": 25 },
  { "name": "Bob Johnson", "age": 35 }
]);
```

查询示例

```
db.collection.aggregate([
  {
    $project: {
      _id: 0,
      name: { $rtrim: { input: "$name", chars: " *" } },
      age: 1
    }
  }
]);
```

输出

```
[
  { age: 30, name: 'John Doe' },
  { age: 25, name: 'Jane Smith' },
  { age: 35, name: 'Bob Johnson' }
]
```

代码示例

要查看使用该$rtrim命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$sample聚合阶段用于从集合中随机选择指定数量的文档。这对于诸如数据分析、测试和生成样本以供进一步处理之类的任务非常有用。

参数

- size：要随机选择的文档数量。

示例（MongoDB 外壳）

以下示例演示如何使用$sample舞台从temp集合中随机选择两个文档。

创建示例文档

```
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") }
]);
```

查询示例

```
db.temp.aggregate([
   { $sample: { size: 2 } }
])
```

输出

```
{ "_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") }
```

结果显示，10份文档中有2份是随机抽样的。现在，您可以使用这些文档来确定平均值或进行 min/max 计算。

代码示例

要查看使用该$sample命令的代码示例，请选择要使用的语言的选项卡：

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

5.0 版的新增内容。

Amazon DocumentDB 中的$search运算符用于提供文本搜索功能。

参数

无

示例（MongoDB 外壳）

以下示例演示如何使用$search运算符执行文本搜索查询。

创建示例文档

```
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." }
]);
```

查询示例

```
db.textcollection.find(
  { $text: { $search: "John" } }
);
```

输出

```
[
  {
    _id: 1,
    name: 'John Doe',
    description: 'This is a sample document about John Doe.'
  }
]
```

代码示例

要查看使用该$search命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$second运算符从日期或时间戳中提取秒部分。它用于从日期或时间戳字段中检索秒值。

参数

- expression：要从中提取秒值的日期或时间戳字段。此表达式可以是字段路径，也可以是任何解析为日期或时间戳的有效表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$second运算符从日期字段中提取秒分量。

创建示例文档

```
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") }
]);
```

查询示例

```
db.users.aggregate([{ $project: { name: 1, dobSeconds: { $second: "$dob" } } }])
```

输出

```
[
```

```
  { "_id" : ObjectId("6089a9c306a829d1f8b456a1"), "name" : "John", "dobSeconds" : 45 },
  { "_id" : ObjectId("6089a9c306a829d1f8b456a2"), "name" : "Jane", "dobSeconds" : 59 },
  { "_id" : ObjectId("6089a9c306a829d1f8b456a3"), "name" : "Bob", "dobSeconds" : 0 }
]
```

代码示例

要查看使用该$second命令的代码示例，请选择要使用的语言的选项卡：

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

8.0 版的新增内容

弹性集群不支持。

Amazon DocumentDB 中的$set聚合阶段允许您在聚合管道期间添加新字段或更新文档中的现有字段值。

参数

- expression：要计算的表达式。这可以是任何有效的聚合表达式，包括字段引用和算术运算。

示例（MongoDB 外壳）

以下示例演示了如何使用$set聚合阶段通过将字段乘以quantity字段来计算总数。price

创建示例文档

```
db.inventory.insertMany([
  { item: "pencil", quantity: 100, price: 0.24},
  { item: "pen", quantity: 204, price: 1.78 }
]);
```

聚合示例

```
db.inventory.aggregate([
  {
    $set: {
      total: { $multiply: ["$quantity", "$price"] }
    }
  }
])
```

输出

```
[
  {
    _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
  }
]
```

代码示例

要查看使用该$set命令的代码示例，请选择要使用的语言的选项卡：

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

4.0 版的新增内容。

Amazon DocumentDB 中的$setDifference运算符用于比较两个集合，并返回第一个集合中但不在第二个集合中的元素。此运算符对于查找两个集合之间的唯一元素很有用。

参数

- `firstSet`：要比较的第一组。
- `secondSet`：要比较的第二组。

示例（MongoDB 外壳）

以下示例演示如何使用$setDifference运算符来查找两个集合之间的唯一元素。

创建示例文档

```
db.collection.insertMany([
  { _id: 1, fruits: ["apple", "banana", "cherry", "date"] },
  { _id: 2, fruits: ["banana", "cherry", "date", "elderberry"] }
]);
```

查询示例

```
db.collection.aggregate([
  {
    $project: {
      uniqueFruits: { $setDifference: ["$fruits", ["banana", "cherry", "date"]] }
    }
  }
]);
```

输出

```
[
  { "_id": 1, "uniqueFruits": ["apple"] },
  { "_id": 2, "uniqueFruits": ["elderberry"] }
]
```

该查询执行以下步骤：

1. 它使用$project舞台uniqueFruits为每个文档创建一个新字段。

2. $setDifference运算符将fruits数组与数组进行比较，["banana", "cherry", "date"]并返回fruits数组中唯一的元素。

代码示例

要查看使用该$setDifference命令的代码示例，请选择要使用的语言的选项卡：

Node.js

以下是如何在 Node.js 应用程序中使用$setDifference运算符的示例：

```
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

以下是如何在 Python 应用程序中使用$setDifference运算符的示例：

```
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 DocumentDB 中的$setEquals运算符用于确定两组是否相等。它比较两个数组，并返回它们true是否包含相同的不同元素，无论它们的顺序如何。

参数

- expression1: 第一个要比较的数组。
- expression2: 第二个要比较的数组。

示例（MongoDB 外壳）

以下示例演示如何使用运$setEquals算符比较两组值。

创建示例文档

```
db.collection.insertMany([
  { _id: 1, fruits: ["apple", "banana", "cherry"] },
```

```
  { _id: 2, fruits: ["banana", "apple", "cherry"] },
  { _id: 3, fruits: ["apple", "banana", "orange"] }
])
```

查询示例

```
db.collection.find({
  $expr: {
    $setEquals: ["$fruits", ["apple", "banana", "cherry"]]
  }
})
```

输出

```
{ "_id" : 1, "fruits" : [ "apple", "banana", "cherry" ] }
{ "_id" : 2, "fruits" : [ "banana", "apple", "cherry" ] }
```

该查询使用$setEquals运算符将每个文档的fruits字段与数组进行比较["apple", "banana", "cherry"]。返回该fruits字段等于比较数组的文档。

代码示例

要查看使用该$setEquals命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$setIntersection运算符用于返回两个或多个数组之间的公共元素。此运算符在处理数据集时特别有用，它允许您找到多个数据集的交集。

参数

- array1: 第一个相交的数组。
- array2: 第二个要相交的数组。
- arrayN: （可选）要相交的其他数组。

示例（MongoDB 外壳）

以下示例演示如何使用$setIntersection运算符来查找两个数组之间的公共元素。

创建示例文档

```
db.collection.insertMany([
  { _id: 1, colors: ["red", "blue", "green"] },
  { _id: 2, colors: ["blue", "yellow", "orange"] },
  { _id: 3, colors: ["red", "green", "purple"] }
])
```

查询示例

```
db.collection.aggregate([
  { $project: {
      _id: 1,
      commonColors: { $setIntersection: ["$colors", ["red", "blue", "green"]] }
    }
  }
])
```

输出

```
[
  { "_id": 1, "commonColors": ["red", "blue", "green"] },
  { "_id": 2, "commonColors": ["blue"] },
  { "_id": 3, "commonColors": ["red", "green"] }
]
```

代码示例

要查看使用该$setIntersection命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$set IsSubset

Amazon DocumentDB 中的$setIsSubset运算符用于确定一组值是否为另一组值的子集。它对于对数组字段执行基于集合的比较和操作很有用。

参数

- field: 要应用$setIsSubset运算符的字段。
- set：用于比较该字段的集合。

示例（MongoDB 外壳）

以下示例演示如何使用$setIsSubset运算符来检查tags字段是否为指定集合的子集。

创建示例文档

```
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"] }
]);
```

查询示例

```
db.products.find({
  $expr: { $setIsSubset: [["tag1", "tag2"], "$tags"] }
})
```

注意： $setIsSubset 是一个聚合运算符，不能直接用于 find () 查询。在此示例中，$expr与一起使用find()来弥合查询运算符和聚合表达式之间的差距。

输出

```
[
  { "_id" : 1, "name" : "Product A", "tags" : [ "tag1", "tag2", "tag3" ] },
  { "_id" : 2, "name" : "Product B", "tags" : [ "tag1", "tag2" ] }
]
```

该查询返回该tags字段是集合子集的子集的文档["tag1", "tag2"]。

代码示例

要查看使用该$setIsSubset命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$setUnion聚合运算符用于组合两组或多组值，并返回一个包含输入集中所有唯一元素的集合。当您需要对文档中的数组字段执行基于集合的操作时，此运算符很有用。

参数

- expression1: 解析为数组的表达式。
- expression2: 解析为数组的表达式。
- expressionN：解析为数组的其他表达式（可选）。

示例（MongoDB 外壳）

以下示例演示如何使用$setUnion运算符将集合中两个数组字段中的唯一元素组合在一起。

创建示例文档

```
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"] }
]);
```

查询示例

```
db.users.aggregate([
  {
    $project: {
      name: 1,
      allInterests: { $setUnion: ["$hobbies", "$skills"] }
    }
  }
]);
```

输出

```
[
  { "_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" ] }
]
```

在此示例中，$setUnion运算符用于组合每个用户文档的hobbies和skills数组字段中的唯一元素。生成的allInterests字段包含每个用户的所有独特爱好和技能的组合。

代码示例

要查看使用该$setUnion命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$skip

在 Amazon DocumentDB 中，$skip运算符用于偏移查询结果的起点，允许您检索匹配文档的特定子集。这在分页场景中特别有用，在这种场景中，您要检索结果的后续页面。

参数

- skip：返回剩余文档之前要跳过的文档数。

示例（MongoDB 外壳）

以下示例演示如何使用$skip运算符从集合中检索第二页结果（文档 11-20）。

创建示例文档

```
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" }
]);
```

查询示例

```
db.collection.find({}, { "name": 1 })
```

```
            .skip(10)
            .limit(10);
```

输出

```
[
  { "_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" }
]
```

代码示例

要查看使用该$skip命令的代码示例，请选择要使用的语言的选项卡：

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聚合运算符使您可以通过从数组的开头或结尾遍历数组来返回数组的子集。这用于显示数组字段中有限数量的项目，例如顶部或底部 N 个项目。

参数

- array: 要切片的数组字段。
- n：一个整数，它指定要返回的元素数。正值从数组的开头开始，而负值从数组的末尾开始。

示例（MongoDB 外壳）

以下示例演示如何使用$slice返回每位厨师最喜欢的前两个糖果。

创建示例文档

```
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" ] }
]);
```

查询示例

```
db.sweets.aggregate([
  { $project: { _id: 0, name: 1, topTwoFavorites: { $slice: [ "$favorites", 2 ] } } }
]);
```

输出

```
[
  { name: 'Alvin', topTwoFavorites: [ 'chocolate', 'cake' ] },
  { name: 'Tom', topTwoFavorites: [ 'donuts', 'pudding' ] },
  { name: 'Jessica', topTwoFavorites: [ 'fudge', 'smores' ] },
  { name: 'Rachel', topTwoFavorites: [ 'ice cream' ] }
]
```

在此示例中，$slice运算符用于从每个文档的favorites数组中提取前两个元素。

代码示例

要查看使用该$slice命令的代码示例，请选择要使用的语言的选项卡：

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

运$size算符用于返回数组字段中的项目数。这可以用来确定存储在文档中的数组中的元素数量。

参数

- field: 要返回其数组大小的字段路径。

示例（MongoDB 外壳）

此示例说明如何使用$size运算符返回每个用户关注的团队数量。

创建示例文档

```
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"] }
]);
```

查询示例

```
db.profiles.aggregate([
  {
    $project: {
      _id: 0,
      name: 1,
      "numberOfTeams": { $size: "$teams" }
    }
  }
])
```

输出

```
[
  { name: 'John Doe', numberOfTeams: 3 },
  { name: 'Jane Smith', numberOfTeams: 2 },
  { name: 'Bob Johnson', numberOfTeams: 3 }
]
```

代码示例

要查看使用该$size命令的代码示例，请选择要使用的语言的选项卡：

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聚合阶段根据指定的字段值对管道中的文档进行排序。根据提供的排序标准，文档按升序或降序排列。

参数

- field：要作为排序依据的字段名称。
- order：1用于升序或-1降序。

示例（MongoDB 外壳）

以下示例演示了使用$sort舞台按价格降序对产品进行排序。

创建示例文档

```
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 }
]);
```

查询示例

```
db.products.aggregate([
  { $sort: { price: -1 } }
]);
```

输出

```
[
  { _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 }
]
```

代码示例

要查看使用$sort聚合阶段的代码示例，请选择要使用的语言对应的选项卡：

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()
```

$split

Amazon DocumentDB 中的$split聚合运算符用于根据指定的分隔符将字符串拆分为子字符串数组。这对于解析复杂的字符串字段和提取单个组件进行进一步处理非常有用。

参数

- string: 要拆分的字符串。
- delimiter：用于拆分输入字符串的字符或字符串。

示例（MongoDB 外壳）

在这个例子中，我们使用$split将 “Desk” 字段的组件分成一个数组，这样可以更轻松地处理数据。

创建示例文档

```
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" }
]);
```

查询示例

```
db.people.aggregate([
  { $project: { parts: { $split: ["$Desk", "-"] } } }
]);
```

输出

```
{ "_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" ] }
```

的输出$split创建了一个数组，可以在应用程序中使用该数组来显示员工的信息。

代码示例

要查看使用该$split命令的代码示例，请选择要使用的语言的选项卡：

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

4.0 版的新增内容。

Amazon DocumentDB 中的$sqrt运算符用于计算数字的平方根。

参数

- expression: 参数可以是任何有效的表达式，只要它解析为非负数即可。

示例（MongoDB 外壳）

以下示例演示如何使用运$sqrt算符来计算数字的平方根。

创建示例文档

```
db.numbers.insertMany([
  { "_id": 1, "number": 16 },
  { "_id": 2, "number": 36 },
  { "_id": 3, "number": 64 }
]);
```

查询示例

```
db.numbers.aggregate([
  { $project: {
    "_id": 1,
    "square_root": { $sqrt: "$number" }
  }}
]);
```

输出

```
[
  { _id: 1, square_root: 4 },
  { _id: 2, square_root: 6 },
  { _id: 3, square_root: 8 }
]
```

代码示例

要查看使用该$sqrt命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$str LenBytes

Amazon DocumentDB 中的$strLenBytes运算符用于确定字符串的长度（以字节为单位）。当您需要了解字符串字段的存储大小时，这很有用，尤其是在处理每个字符可能使用超过一个字节的 Unicode 字符时。

参数

- expression：要计算长度的字符串表达式。

示例（MongoDB 外壳）

此示例演示如何使用$strLenBytes运算符计算字符串字段的长度（以字节为单位）。

创建示例文档

```
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" }
]);
```

查询示例

```
db.people.aggregate([
  {
    $project: {
      "Desk": 1,
      "length": { $strLenBytes: "$Desk" }
    }
  }
])
```

输出

```
{ "_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 }
```

请注意，“Düseldorf-BVV-021” 字符串的长度为 19 字节，这与码点数 (18) 不同，因为 Unicode 字符 “U” 占用 2 个字节。

代码示例

要查看使用该`$strLenBytes`命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$strLenCP运算符用于确定以代码点为单位的字符串表达式的长度（Unicode 字符）。当你需要知道字符串中的字符数而不是字节数时，这很有用。

参数

- expression：要返回长度的字符串表达式（以代码点为单位）。

示例（MongoDB 外壳）

以下示例演示如何使用$strLenCP运算符来确定包含 Unicode 字符的字符串的长度。

创建示例文档

```
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" }
])
```

查询示例

```
db.people.aggregate([
  {
    $project: {
      "Desk": 1,
      "length": { $strLenCP: "$Desk" }
    }
  }
])
```

输出

```
{ "_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 }
```

请注意包含 Unicode 字符 (U) 的 “杜塞尔多夫-BVV-021” 字符串的长度测量值的差异。$strLenCP运算符正确计算 Unicode 字符数，而$strLenBytes运算符则计算字节数。

代码示例

要查看使用该$strLenCP命令的代码示例，请选择要使用的语言的选项卡：

Node.js

以下是在带有 MongoDB 驱动 Node.js 程序的应用程序中使用$strLenCP运算符的示例：

```
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

以下是在带 PyMongo 驱动程序的 Python 应用程序中使用$strLenCP运算符的示例：

```
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 DocumentDB 中的$strcasecmp运算符在两个字符串之间执行不区分大小写的比较。它返回一个整数值，表示两个输入字符串的字典比较，忽略大小写差异。

参数

- string1: 要比较的第一个字符串。
- string2: 要比较的第二个字符串。

示例（MongoDB 外壳）

此示例演示如何使用$strcasecmp运算符比较people集合中的办公桌位置字符串，同时忽略大小写差异。

创建示例文档

```
db.people.insertMany([
  { "_id": 1, "Desk": "mke233-wi" },
  { "_id": 2, "Desk": "MKE233-WI" },
```

```
  { "_id": 3, "Desk": "mke233-wi" }
]);
```

查询示例

```
db.people.aggregate([
  {
    $project: {
      item: 1,
      compare: { $strcasecmp: ["$Desk", "mke233-wi"] }
    }
  }
]);
```

输出

```
{ "_id" : 1, "compare" : 0 }
{ "_id" : 2, "compare" : 0 }
{ "_id" : 3, "compare" : 0 }
```

输出显示，"Desk"字段和字符串之间的比较会"mke233-wi"返回0所有三个文档，这表明忽略大小写时，字符串相等。

代码示例

要查看使用该$strcasecmp命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$substr运算符用于从给定字符串中提取子字符串。当您需要根据字符范围而不是字节范围定义子字符串时，它特别有用。在处理 Unicode 字符串时，这一点尤其重要，因为在这些字符串中，用于表示字符的字节数可能会有所不同。

参数

- `string`：要从中提取子字符串的输入字符串。
- `start`：要提取的子字符串的起始位置（从零开始）。可以是非负整数表达式。
- `length`：提取的子字符串中的字符数。可以是非负整数表达式。

示例（MongoDB 外壳）

在此示例中，我们将演示如何使用从员工办公桌位置提取州缩写。$substr

创建示例文档

```
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" }
]);
```

查询示例

```
db.people.aggregate([
  {
    $project: {
      "state": { $substr: ["$Desk", 12, 3] }
    }
  }
])
```

输出

```
{ "_id": 1, "state": "NRW" },
{ "_id": 2, "state": "HBB" },
{ "_id": 3, "state": "SHH" },
```

```
{ "_id": 4, "state": "BBB" }
```

代码示例

要查看使用该$substr命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$substrBytes运算符用于根据指定的字节范围从字符串中提取子字符串。当您需要从字符串中提取子字符串并且表示字符串中每个字符所需的字节数很重要时，此运算符很有用。

与之不同$substrCP，它对Unicode码点的数量进行$substrBytes运算，对表示字符串中字符所需的字节数进行运算。这在处理包含非 ASCII 字符的字符串时特别有用，因为这些字符可能需要多个字节才能表示。

注意： 自 3.4 版起$substr已被弃用。 $substr现在是的别名$substrBytes。

参数

- string：要从中提取子字符串的输入字符串。
- startByte：要提取的子字符串的起始字节位置（从零开始）。负值可用于指定从字符串末尾开始的位置。
- length：要提取的子字符串中的字节数。

示例（MongoDB 外壳）

在此示例中，我们将使用$substrBytes从包含非 ASCII 字符的字符串中提取子字符串。

创建示例文档

```
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" }
]);
```

查询示例

```
db.people.aggregate([
  {
    $project: {
      "state": { $substrBytes: [ "$Desk", 12, 3] }
    }
  }
])
```

输出

```
{ "_id": 1, "state": "NRW" },
{ "_id": 2, "state": "HBB" },
{ "_id": 3, "state": "SHH" },
{ "_id": 4, "state": "BBB" }
```

在此示例中，我们使用从字段$substrBytes的第 12 个字节开始提取一个 3 字节的Desk子字符串。这允许我们提取 2 个字符的状态缩写，即使该字符串可能包含非 ASCII 字符。

代码示例

要查看使用该$substrBytes命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$substrCP运算符用于从字符串中提取子字符串，其中子字符串指定为 UTF-8 代码点范围 (CP)。此运算符在处理 Unicode 字符串时特别有用，因为它允许您提取子字符串，而不必担心字符的底层字节表示形式。

与对字节位置进行$substrBytes操作的运算符不同，该$substrCP运算符使用码点位置。这使得处理包含非 ASCII 字符的字符串变得更加容易，因为代码点的数量可能与字节数或字符数不匹配。

参数

- string：要从中提取子字符串的输入字符串。
- start：从中提取子字符串的起始码点位置（从零开始）。
- length：要提取的代码点数。

示例（MongoDB 外壳）

在此示例中，我们将使用$substrCP运算符从包含员工办公桌位置的字符串中提取州缩写。

创建示例文档

```
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" }
]);
```

查询示例

```
db.people.aggregate([
  {
    $project: {
      "state": { $substrCP: ["$Desk", 25, 2] }
    }
  }
]);
```

输出

```
{ "_id" : 1, "state" : "MN" }
{ "_id" : 2, "state" : "LA" }
{ "_id" : 3, "state" : "CA" }
```

在此示例中，我们知道州缩写从Desk字段的第 25 个代码点开始，长度为 2 个代码点。通过使用$substrCP运算符，我们可以提取状态缩写，而不必担心字符串的底层字节表示形式。

代码示例

要查看使用该$substrCP命令的代码示例，请选择要使用的语言的选项卡：

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()
```

在 Node.js 和 Python 示例中，我们都使用$substrCP运算符从Desk字段中提取状态缩写，类似于 MongoDB 命令行管理程序示例。

$subtract

Amazon DocumentDB 中的$subtract运算符用于减去值。它可以用来减去日期、数字或两者的组合。此运算符可用于计算两个日期之间的差值或从数字中减去一个值。

参数

- expression1：要减去的第一个值。
- expression2：要从<expression1>中减去的第二个值。

示例（MongoDB 外壳）

以下示例演示如何使用$subtract运算符来计算两个日期之间的差。

创建示例文档

```
db.dates.insert([
  {
  "_id": 1,
  "startDate": ISODate("2023-01-01T00:00:00Z"),
  "endDate": ISODate("2023-01-05T12:00:00Z")
  }
]);
```

查询示例

```
db.dates.aggregate([
  {
    $project: {
      _id: 1,
```

```
      durationDays: {
        $divide: [
          { $subtract: ["$endDate", "$startDate"] },
          1000 * 60 * 60 * 24  // milliseconds in a day
        ]
      }
    }
  }
]);
```

输出

```
[ { _id: 1, durationDays: 4.5 } ]
```

在此示例中，$subtract运算符用于计算$endDate和之间的差值（$startDate以天为单位）。

代码示例

要查看使用该$subtract命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$sum运算符返回一组中每个文档的指定表达式的总和。它是一个组累加器运算符，通常用于聚合管道的 $group 阶段来执行求和计算。

参数

- expression：要求和的数值表达式。这可以是字段路径、表达式或常量。

示例（MongoDB 外壳）

以下示例演示如何使用运$sum算符来计算每种产品的总销售额。

创建示例文档

```
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 }
]);
```

查询示例

```
db.sales.aggregate([
```

```
  { $group: {
      _id: "$product",
      totalSales: { $sum: { $multiply: [ "$price", "$quantity" ] } }
    }}
]);
```

输出

```
[
  { "_id": "abc", "totalSales": 50 },
  { "_id": "xyz", "totalSales": 120 }
]
```

代码示例

要查看使用该$sum命令的代码示例，请选择要使用的语言的选项卡：

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('sales');

    const result = await collection.aggregate([
      { $group: {
          _id: "$product",
          totalSales: { $sum: { $multiply: [ "$price", "$quantity" ] } }
        }}
    ]).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.sales

        result = list(collection.aggregate([
            { '$group': {
                '_id': '$product',
                'totalSales': { '$sum': { '$multiply': [ '$price', '$quantity' ] } }
            }}
        ]))

        pprint(result)

    except Exception as e:
        print(f"An error occurred: {e}")

    finally:
        if client:
            client.close()

example()
```

$switch

4.0 版的新增内容。

弹性集群不支持。

$switch运算符是 Amazon DocumentDB 中的条件表达式运算符，它允许您评估案例表达式列表并返回计算结果为 true 的第一个案例的值，如果没有大小写表达式为真，则返回默认值。

参数

- branches：一个文档数组，每个文档都有一个包含要计算的布尔表达式的 case 字段，以及一个包含在 case 表达式为真时要返回的值的 then 字段。
- default:（可选）如果没有一个 case 表达式为真时要返回的值。

示例（MongoDB 外壳）

以下示例演示如何使用$switch运算符根据订单总额确定订单的运费。

创建示例文档

```
db.orders.insertMany([
  { _id: 1, total: 50 },
  { _id: 2, total: 150 },
  { _id: 3, total: 250 }
]);
```

查询示例

```
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
        }
      }
    }
  }
])
```

输出

```
[
  {
    "_id": 1,
    "total": 50,
    "shippingCost": 5
  },
  {
    "_id": 2,
    "total": 150,
    "shippingCost": 10
  },
  {
    "_id": 3,
    "total": 250,
    "shippingCost": 15
  }
]
```

代码示例

要查看使用该$switch命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$toBool

Amazon DocumentDB 中的$toBool运算符将表达式转换为布尔值。

参数

- expression：要转换为布尔值的表达式。

注意：任何字符串都将转换为true。

示例（MongoDB 外壳）

以下示例演示如何使用$toBool运算符对来自不同数据类型的设备状态值进行标准化。

创建示例文档

```
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 }
]);
```

查询示例

```
db.deviceStates.aggregate([
  {
    $project: {
      _id: 1,
      deviceId: 1,
      isActive: { $toBool: "$status" }
    }
  }
]);
```

输出

```
[
  { "_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 }
]
```

代码示例

要查看使用该$toBool命令的代码示例，请选择要使用的语言的选项卡：

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

4.0 版的新增内容

Amazon DocumentDB 中的$toDate聚合运算符用于将日期或日期和时间字符串转换为 BSON 日期类型。这是运算符的反向$dateToString运算。

参数

- dateString：要转换为 BSON 日期类型的日期或日期和时间的字符串表示形式。
- format:（可选）指定格式的字符串dateString。如果未提供，则操作员将尝试dateString以各种标准日期和时间格式解析。
- timezone:（可选）一个字符串，表示用于转换的时区。如果未提供，则使用当地时区。

示例（MongoDB 外壳）

以下示例演示如何使用$toDate运算符将日期字符串转换为 BSON 日期类型。

创建示例文档

```
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" }
]);
```

查询示例

```
db.events.aggregate([
  {
    $project: {
      eventName: 1,
      eventTimeDate: { $toDate: "$eventTime" }
    }
  }
]);
```

输出

```
[
  {
    "_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")
  }
]
```

代码示例

要查看使用该$toDate命令的代码示例，请选择要使用的语言的选项卡：

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

4.0 版的新增内容

Amazon DocumentDB 中的$toDecimal运算符用于将值转换为 Decimal128 数据类型。当您需要执行精确的十进制算术或处理无法使用 Double 数据类型精确表示的大十进制值时，这很有用。

参数

- expression：要转换为 Decimal128 数据类型的表达式。

示例（MongoDB 外壳）

此示例演示如何使用$toDecimal运算符将字符串值转换为 Decimal128 数据类型。

创建示例文档

```
db.numbers.insertOne({ _id: 1, value: "3.14" });
db.numbers.insertOne({ _id: 2, value: "2.71" });
```

查询示例

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    decimalValue: { $toDecimal: "$value" }
  }}
])
```

输出

```
[
  { "_id" : 1, "decimalValue" : Decimal128("3.14") },
  { "_id" : 2, "decimalValue" : Decimal128("2.71") }
]
```

代码示例

要查看使用该$toDecimal命令的代码示例，请选择要使用的语言的选项卡：

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

4.0 版的新增内容

Amazon DocumentDB 中的$toDouble运算符用于将值转换为双精度 64 位浮点数。当您需要对最初不是数字格式的值执行算术运算时，这可能很有用。

参数

<expression>：要转换为双精度值的表达式。这可以是任何解析为数字、字符串或布尔值的有效表达式。

示例（MongoDB 外壳）

此示例演示如何使用$toDouble运算符将字符串值转换为数值以进行数学计算。

创建示例文档

```
db.numbers.insertMany([
  { _id: 1, value: "10.5" },
  { _id: 2, value: "20.25" },
  { _id: 3, value: "7" }
])
```

查询示例

```
db.numbers.aggregate([
  {
    $project: {
      _id: 1,
      value: 1,
      double_value: { $toDouble: "$value" },
      double_plus_five: { $add: [{ $toDouble: "$value" }, 5] }
    }
  }
])
```

输出

```
[
  { "_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 }
]
```

代码示例

要查看使用该$toDouble命令的代码示例，请选择要使用的语言的选项卡：

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

4.0 版的新增内容

Amazon DocumentDB 中的$toInt运算符用于将输入值转换为整数数据类型。当您需要确保将字段或表达式表示为整数时，此运算符很有用，这对于某些操作或数据处理任务可能很重要。

参数

- expression：要转换为整数的表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$toInt运算符将字符串值转换为整数。

创建示例文档

```
db.numbers.insertMany([
  { "name": "one", "value": "1" },
  { "name": "hundred", "value": "100" }
]);
```

查询示例

```
db.numbers.aggregate([
  { $project: {
    "_id": 0,
    "name": 1,
    "intValue": { $toInt: "$value" }
  }}
]);
```

输出

```
{ "name": "one", "intValue": 1 }
{ "name": "hundred", "intValue": 100 }
```

代码示例

要查看使用该$toInt命令的代码示例，请选择要使用的语言的选项卡：

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

4.0 版的新增内容

Amazon DocumentDB 中的$toLong运算符用于将值转换为 64 位整数（长）数据类型。当您需要对可能存储为字符串或其他数据类型的数值执行算术运算或比较时，这可能很有用。

参数

- `expression`：要转换为 64 位整数的表达式。

示例（MongoDB 外壳）

此示例演示如何使用$toLong运算符将字符串值转换为 64 位整数。

创建示例文档

```
db.numbers.insertMany([
  { _id: 1, value: "42" },
  { _id: 3, value: "92233720368547758O7" }
]);
```

查询示例

```
db.numbers.aggregate([
```

```
  {
    $project: {
      _id: 1,
      longValue: { $toLong: "$value" }
    }
  }
])
```

输出

```
[
  { "_id" : 1, "longValue" : 42 },
  { "_id" : 3, "longValue" : 9223372036854775807 }
]
```

代码示例

要查看使用该$toLong命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$toLower运算符用于将字符串转换为小写。

参数

- expression: 要转换为小写的字符串表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$toLower运算符将Desk字段转换为小写。

创建示例文档

```
db.locations.insertMany([
  { "_id": 1, "Desk": "Düsseldorf-BVV-021" },
  { "_id": 2, "Desk": "Munich-HGG-32a" }
]);
```

查询示例

```
db.locations.aggregate([
  { $project: { item: { $toLower: "$Desk" } } }
]);
```

输出

```
{ "_id" : 1, "item" : "düsseldorf-bvv-021" }
{ "_id" : 2, "item" : "munich-hgg-32a" }
```

代码示例

要查看使用该$toLower命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$to ObjectId

4.0 版的新增内容

Amazon DocumentDB 中的`$toObjectId`运算符用于将的字符串表示形式转换为实际 ObjectId 的数据类型。 ObjectId 这在处理以字符串表示形式存储的数据时非常有用 ObjectIds，因为它允许您执行需要该 ObjectId 数据类型的操作。

参数

- `expression`: 表示有效值的字符串表达式 ObjectId。

示例（MongoDB 外壳）

以下示例演示如何使用`$toObjectId`运算符将的字符串表示形式转换为 ObjectId 数据类型。 ObjectId

创建示例文档

```
db.employees.insertMany([
```

```
  { _id: 1, empId:"64e5f8886218c620cf0e8f8a", name: "Carol Smith", employeeId:
 "c720a" },
  { _id: 2, empId:"64e5f94e6218c620cf0e8f8c", name: "Bill Taylor", employeeId:
 "c721a" }
]);
```

查询示例

```
db.employees.aggregate([
  { $project: {
    "empIdAsObjectId": {$toObjectId: "$empId"}}
  }
]);
```

输出

```
[
  { _id: 1, empIdAsObjectId: ObjectId('64e5f8886218c620cf0e8f8a') },
  { _id: 2, empIdAsObjectId: ObjectId('64e5f94e6218c620cf0e8f8c') }
]
```

代码示例

要查看使用该$toObjectId命令的代码示例，请选择要使用的语言的选项卡：

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

4.0 版的新增内容

Amazon DocumentDB 中的$toString运算符用于将任何类型的值（空值除外）转换为字符串表示形式。当您需要对最初不是字符串格式的值执行字符串操作时，这可能很有用。

参数

- `expression`: 要转换为字符串的表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$toString运算符将数值转换为字符串。

创建示例文档

```
db.numbers.insertMany([
  { "_id": 1, "value": 42 },
  { "_id": 2, "value": 3.14 }
]);
```

查询示例

```
db.numbers.aggregate([
  { $project: {
    _id: 1,
    valueAsString: { $toString: "$value" }
  }}
]);
```

输出

```
{ "_id": 1, "valueAsString": "42" }
{ "_id": 2, "valueAsString": "3.14" }
```

代码示例

要查看使用该$toString命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$toUpper运算符用于将字符串转换为大写。

参数

- expression: 要转换为大写的字符串表达式。

示例（MongoDB 外壳）

以下示例演示如何使用运$toUpper算符将Desk字段转换为大写。

创建示例文档

```
db.locations.insertMany([
  { "_id": 1, "Desk": "düsseldorf-bvv-021" },
  { "_id": 2, "Desk": "munich-hgg-32a" }
]);
```

查询示例

```
db.locations.aggregate([
  { $project: { item: { $toUpper: "$Desk" } } }
]);
```

输出

```
{ "_id" : 1, "item" : "DüSSELDORF-BVV-021" }
{ "_id" : 2, "item" : "MUNICH-HGG-32A" }
```

代码示例

要查看使用该$toUpper命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$trim

4.0 版的新增内容

Amazon DocumentDB 中的$trim运算符用于从字符串中删除前导 and/or 尾随的空格字符。

参数

- input: 要修剪的字符串表达式。
- chars: （可选）指定要从输入的开头和结尾修剪的字符，默认为空格。

示例（MongoDB 外壳）

以下示例演示如何使用$trim运算符从字符串中删除前导和尾随空格。

创建示例文档

```
db.people.insertMany([
  { "name": "   John Doe   " },
  { "name": "   Bob Johnson   " }
])
```

查询示例

```
db.people.aggregate([
  { $project: {
    "name": { $trim: {input: "$name"}}
  }}
])
```

输出

```
[
  { "name": "John Doe" },
  { "name": "Bob Johnson" }
]
```

代码示例

要查看使用该$trim命令的代码示例，请选择要使用的语言的选项卡：

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聚合运算符返回指定字段的 BSON 数据类型。这对于在聚合操作期间识别字段值的数据类型非常有用。

参数

- `expression`：要返回其类型的字段或表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$type运算符来识别每种产品的价格字段的数据类型。

创建示例文档

```
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 }
]);
```

查询示例

```
db.inventory.aggregate([
  {
    $project: {
      item: 1,
      price: 1,
      priceType: { $type: "$price" }
    }
  }
]);
```

输出

```
[
  { _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' }
]
```

代码示例

要查看使用$type聚合运算符的代码示例，请选择要使用的语言的选项卡：

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

8.0 版的新增内容

弹性集群不支持。

Amazon DocumentDB 中的$unset聚合阶段允许您从文档中删除字段。

参数

- expression：字段名称或多个字段名称的列表。

示例（MongoDB 外壳）

以下示例演示了如何使用$unset聚合阶段来移除字price段。

创建示例文档

```
db.inventory.insertMany([
  { item: "pencil", quantity: 100, price: 0.24},
  { item: "pen", quantity: 204, price: 1.78 }
]);
```

聚合示例

```
db.inventory.aggregate([
  {
    $unset: "price"
  }
])
```

输出

```
[
  {
    _id: ObjectId('69248951d66dcae121d2950d'),
    item: 'pencil',
    quantity: 100
```

```
  },
  {
    _id: ObjectId('69248951d66dcae121d2950e'),
    item: 'pen',
    quantity: 204
  }
]
```

代码示例

要查看使用该$unset命令的代码示例，请选择要使用的语言的选项卡：

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

该$unwind运算符用于从输入文档中解构一个数组字段，为每个元素输出一个文档。当您要对数组的各个元素执行操作（例如筛选、排序或转换数据）时，这可能很有用。

参数

- path: 要展开的数组字段的路径。
- includeArrayIndex: （可选）指定用于保存数组元素索引的新字段的名称。
- preserveNullAndEmptyArrays: （可选）确定当数组字段为空还是空数组时，操作是保留原始文档。

示例（MongoDB 外壳）

以下示例演示如何使用$unwind运算符解构数组字段并对各个元素执行进一步的操作。

创建示例文档

```
db.people.insertMany([
  { _id: 1, name: "jon", hobbies: ["painting", "dancing", "singing"] },
  { _id: 2, name: "jane", hobbies: ["reading", "swimming"] },
  { _id: 3, name: "jack", hobbies: [] }
])
```

查询示例

```
db.people.aggregate([
  { $unwind: "$hobbies" }
])
```

输出

```
[
  { _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' }
]
```

代码示例

要查看使用该$unwind命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$向量搜索

8.0 版的新增内容

弹性集群不支持。

Amazon DocumentDB 中的$vectorSearch运算符允许您执行矢量搜索，这是机器学习中使用的一种方法，通过使用距离或相似度指标比较它们的矢量表示来查找相似的数据点。该功能将 JSON-based 文档数据库的灵活性和丰富的查询功能与矢量搜索的强大功能相结合，使您能够构建机器学习和生成式 AI 用例，例如语义搜索、产品推荐等。

参数

- <exact>（可选）：指定是运行精确最近邻 (ENN) 还是近似最近邻 (ANN) 搜索的标志。值可以是以下值之一：
- false-运行 ANN 搜索
- true-运行 ENN 搜索

如果省略或设置为 false，numCandidates则为必填项。

```
- `<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.
```

示例（MongoDB 外壳）

以下示例演示如何使用$vectorSearch运算符根据向量表示法查找相似的产品描述。

创建示例文档

```
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]
  }
]);
```

创建矢量搜索索引

```
db.runCommand(
    {
        createIndexes: "products",
        indexes: [{
            key: {
                "description_vector": "vector"
```

```
            },
            vectorOptions: {
                type: "hnsw",
                dimensions: 3,
                similarity: "cosine",
                m: 16,
                efConstruction: 64
            },
            name: "description_index"
        }]
    }
);
```

查询示例

```
db.products.aggregate([
  { $vectorSearch: {
      index: "description_index",
      limit: 2,
      numCandidates: 10,
      path: "description_vector",
      queryVector: [0.1, 0.2, 0.3]
    }
  }
]);
```

输出

```
[
  {
    "_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 ]
  }
]
```

代码示例

要查看使用该$vectorSearch命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$week运算符根据 ISO 8601 标准返回日期的周数 (0-53)。周数是根据年份和星期几计算的，星期一是一周的第一天。

参数

无

示例（MongoDB 外壳）

以下示例演示如何使用$week运算符检索给定日期的星期数。

创建示例文档

```
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") }
]);
```

查询示例

```
db.events.aggregate([
```

```
  { $project: {
    _id: 1,
    week: { $week: "$date" }
  }}
]);
```

输出

```
[
  { "_id": 1, "week": 1 },
  { "_id": 2, "week": 2 },
  { "_id": 3, "week": 53 }
]
```

代码示例

要查看使用该$week命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$year运算符从日期或时间戳中提取年份部分。

参数

- expression：要从中提取年份部分的日期或时间戳表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$year运算符从日期字段中提取年份部分。

创建示例文档

```
db.events.insertMany([
  { "_id": 1, "date": ISODate("2023-04-15T00:00:00Z") },
  { "_id": 3, "date": ISODate("2021-12-31T00:00:00Z") }
```

```
]);
```

查询示例

```
db.events.aggregate([
  { $project: { year: { $year: "$date" } } }
]);
```

输出

```
[
  { "_id": 1, "year": 2023 },
  { "_id": 3, "year": 2021 }
]
```

代码示例

要查看使用该$year命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$zip运算符允许您将多个数组组合成单个元组数组（键值对）。当您需要通过合并来自不同源或文档中数组的数据来创建新文档或对象时，这很有用。

参数

- inputs：解析为数组的表达式数组。这些数组将被组合成一个元组数组。
- useLongestLength:（可选）如果true，则输出数组将具有最长的输入数组的长度，用null值填充较短的数组。如果false，则输出数组的长度将等于最短的输入数组。
- defaults:（可选）如果相应的输入数组短于最长的输入数组且useLongestLength为，则用于元组的默认值数组。true

示例（MongoDB 外壳）

以下示例演示如何使用$zip运算符将两个数组组合成单个元组数组。

创建示例文档

```
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"]
  }
])
```

查询示例

```
db.grades.aggregate([
  {
    $project: {
      "name": 1,
      "scoredClasses": {
        $zip: {
          inputs: ["$scores", "$classes"],
          useLongestLength: true,
          defaults: [null, null]
        }
      }
    }
  }
])
```

输出

```
[
  {
    "_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"]
    ]
  }
]
```

代码示例

要查看使用该$zip命令的代码示例，请选择要使用的语言的选项卡：

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()
```

地理空间

本节提供有关 Amazon DocumentDB 支持的地理空间运算符的详细信息。

主题

- $geometry
- $geoIntersects
- $geoWithin
- $maxDistance
- $minDistance
- $near
- $nearSphere

$geometry

Amazon DocumentDB 中的`$geometry`运算符用于在地理空间查询中指定 GeoJSON 几何对象。此运算符可与其他地理空间查询运算符（如`$geoWithin`和）结合使用`$geoIntersects`，对您的数据执行空间查询。

在亚马逊 DocumentDB 中，`$geometry`运算符支持以下 GeoJSON 几何类型：

- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon
- GeometryCollection

参数

- `type`: geoJSON 几何对象的类型，例如`PointPolygon`、等
- `coordinates`：表示几何形状的坐标数组。坐标数组的结构取决于几何类型。

示例（MongoDB 外壳）

以下示例演示如何使用`$geometry`运算符在 Amazon DocumentDB 中执行`$geoIntersects`查询。

创建示例文档

```
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]
      ]]
    }
  }
]);
```

查询示例

```
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]
        ]]
      }
    }
```

```
  }
})
```

输出

```
[
  {
    "_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]
        ]
      ]
    }
  }
]
```

代码示例

要查看使用该$geometry命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$geoIntersects运算符用于查找其地理空间数据与指定 GeoJSON 对象相交的文档。对于需要根据文档与给定地理形状（例如多边形或多面）的空间关系来识别文档的应用程序，此运算符非常有用。

参数

- $geometry: 一个 GeoJSON 对象，它表示要检查交叉点的形状。支持的 GeoJSON 对象类型有Point、LineStringPolygon、和。MultiPolygon

示例（MongoDB 外壳）

以下示例演示如何使用$geoIntersects运算符在 Amazon DocumentDB 中查找一组给定坐标的状态名称。

创建示例文档

```
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]
      ]]
    }
  }
]);
```

查询示例

```
var location = [-73.965355, 40.782865];

db.states.find({
  "loc": {
    "$geoIntersects": {
      "$geometry": {
        "type": "Point",
        "coordinates": location
      }
    }
  }
}, {
  "name": 1
});
```

输出

```
{ "_id" : ObjectId("536b0a143004b15885c91a2c"), "name" : "New York" }
```

代码示例

要查看使用该$geoIntersects命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$geoWithin运算符用于查找位置数据（表示为 GeoJSON 对象）完全包含在指定形状（例如多边形或多面）内的文档。这对于查询位于特定地理区域内的对象非常有用。

参数

- $geometry: 一个 GeoJSON 对象，它表示要查询的形状。

示例（MongoDB 外壳）

以下示例演示如何使用$geoWithin运算符查找位于纽约州内的所有机场。

创建示例文档

```
// 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]
        }
    }
]);
```

查询示例

```
var state = db.states.findOne({"name": "New York"});

db.airports.find({
    "loc": {
        "$geoWithin": {
            "$geometry": state.loc
        }
    }
}, {
    "name": 1,
    "type": 1,
    "code": 1,
    "_id": 0
});
```

输出

```
[
  {
    "name": "John F. Kennedy International Airport",
    "type": "airport",
    "code": "JFK"
```

```
  },
  {
    "name": "LaGuardia Airport",
    "type": "airport",
    "code": "LGA"
  }
]
```

代码示例

要查看使用该$geoWithin命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$maxDistance运算符用于指定距离 GeoJSON 点的最大距离（以米为单位），文档必须在该距离之内才能包含在查询结果中。此运算符与运$nearSphere算符配合使用以执行地理空间查询。

参数

- $maxDistance：距离参考点的最大距离（以米为单位），文档必须在该距离之内才能包含在查询结果中。

示例（MongoDB 外壳）

以下示例演示如何使用 Amazon DocumentDB 中的$maxDistance运算符来查找距离波士顿 100 千米范围内的所有州首府。

创建示例文档

```
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] } }
]);
```

查询示例

```
db.capitals.find(
  {
    location: {
      $nearSphere: {
        $geometry: { type: "Point", coordinates: [-71.0589, 42.3601] },
        $maxDistance: 100000
      }
    }
  },
  { state: 1, city: 1, _id: 0 }
);
```

输出

```
[
  { "state": "Rhode Island", "city": "Providence" },
  { "state": "New Hampshire", "city": "Concord" }
]
```

代码示例

要查看使用该$maxDistance命令的代码示例，请选择要使用的语言的选项卡：

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

$minDistance是与$nearSphere或$geoNear结合使用的查找运算符，用于筛选距离中心点至少达到指定最小距离的文档。该运算符在 Amazon DocumentDB 中受支持，其功能与 MongoDB 中的对应运算符类似。

参数

- $minDistance：从中心点到结果中包含文档的最小距离（以米为单位）。

示例（MongoDB 外壳）

在此示例中，我们将找到华盛顿州西雅图特定地点 2 公里半径范围内的所有餐厅。

创建示例文档

```
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]
    }
  }
]);
```

查询示例

```
db.usarestaurants.find({
  "location": {
    "$nearSphere": {
      "$geometry": {
        "type": "Point",
        "coordinates": [-122.3516, 47.6156]
      },
      "$minDistance": 1,
      "$maxDistance": 2000
    }
  }
}, {
  "name": 1
});
```

输出

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "name" : "Noodle House" }
{ "_id" : ObjectId("611f3da985009a81ad38e74c"), "name" : "Pike Place Grill" }
```

代码示例

要查看使用该$minDistance命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$near运算符用于查找地理位置靠近指定点的文档。它返回按距离排序的文档，最接近的文档排在最前面。此运算符需要 2dsphere 地理空间索引，可用于位置数据的邻近查询。

参数

- $geometry: 一个 GeoJSON 点对象，用于定义近距离查询的中心点。
- $maxDistance: （可选）文档与查询相匹配的最大距离（以米为单位）。
- $minDistance: （可选）文档与查询相匹配的最小距离（以米为单位）。

索引要求

- 2dsphere index：对 GeoJSON 点数据进行地理空间查询所必需的。

示例（MongoDB 外壳）

以下示例演示如何使用$near运算符查找距离华盛顿州西雅图特定地点最近的餐厅。

创建示例文档

```
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] }
  }
]);
```

创建 2dsphere 索引

```
db.usarestaurants.createIndex({ "location": "2dsphere" });
```

使用 geoJSON 点查询示例

```
db.usarestaurants.find({
  location: {
    $near: {
      $geometry: {
        type: "Point",
        coordinates: [-122.3516, 47.6156]
      },
      $maxDistance: 100,
      $minDistance: 10
    }
  }
```

```
});
```

输出

```
{
  "_id" : ObjectId("69031ec9ea1c2922a1ce5f4a"),
  "name" : "Noodle House",
  "city" : "Seattle",
  "state" : "Washington",
  "rating" : 4.8,
  "location" : {
    "type" : "Point",
    "coordinates" : [ -122.3517, 47.6159 ]
  }
}
```

代码示例

要查看使用该$near命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$nearSphere运算符用于查找距离地理空间点指定距离内的文档。此运算符对于地理空间查询特别有用，例如查找给定位置一定半径范围内的所有餐厅。

参数

- $geometry: 表示参考点的 GeoJSON 对象。必须是带有type和coordinates字段的Point对象。
- $minDistance:（可选）文档必须位于的参考点的最小距离（以米为单位）。
- $maxDistance:（可选）从文档必须位于的参考点的最大距离（以米为单位）。

示例（MongoDB 外壳）

在此示例中，我们将找到距离华盛顿州西雅图特定地点 2 千米（2000 米）范围内的所有餐厅。

创建示例文档

```
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] }
  }
]);
```

查询示例

```
db.usarestaurants.find({
  location: {
    $nearSphere: {
      $geometry: {
        type: "Point",
```

```
        coordinates: [-122.3516, 47.6156]
      },
      $minDistance: 1,
      $maxDistance: 2000
    }
  }
}, {
  name: 1
});
```

输出

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "name" : "Noodle House" }
{ "_id" : ObjectId("611f3da985009a81ad38e74c"), "name" : "Pike Place Grill" }
```

代码示例

要查看使用该$nearSphere命令的代码示例，请选择要使用的语言的选项卡：

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()
```

查询和投影运算符

本节提供有关 Amazon DocumentDB 支持的查询和投影运算符的详细信息。

主题

- $
- $all
- $and
- $bits AllClear
- $bits AllSet
- $bits AnyClear
- $bits AnySet
- $comment
- $elemMatch
- $eq
- $exists
- $expr
- $gt
- $gte
- $in
- $jsonSchema
- $lt
- $mod
- $lte
- $meta
- $ne
- $nin
- $nor
- $not
- $or
- $regex
- $slice
- $size

- $text
- $type

$

$投影运算符将数组字段的内容限制为仅返回与查询条件匹配的第一个元素。它用于投影单个匹配的数组元素。

参数

- `field.$`: 带有位置运算符的数组字段，用于投影第一个匹配的元素。

示例（MongoDB 外壳）

以下示例演示了使用$投影运算符仅返回匹配的数组元素。

创建示例文档

```
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] }
]);
```

查询示例

```
db.students.find(
  { grades: { $gte: 90 } },
  { name: 1, "grades.$": 1 }
);
```

输出

```
{ "_id" : 1, "name" : "Alice", "grades" : [ 92 ] }
{ "_id" : 2, "name" : "Bob", "grades" : [ 92 ] }
{ "_id" : 3, "name" : "Charlie", "grades" : [ 95 ] }
```

在此示例中，仅返回每个学生大于或等于 90 的一年级成绩。

代码示例

要查看使用$投影运算符的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$all运算符用于匹配其中字段值为数组且包含所有指定元素的文档，无论数组中元素的顺序如何。

参数

- field：要检查的字段的名称。
- [value1, value2, ...]：数组中要匹配的值列表。

在$all表达式$elemMatch中使用

有关在$all表达式中使用$elemMatch运算符的限制，请参阅# $all ###### $elemMatch。

字段名称中的美元 ($)

有关在嵌套对象中查询$前缀字段的限制$all，请参阅字段名称中的美元符号（$）和句点（.）。

示例（MongoDB 外壳）

以下示例演示如何使用$all运算符来检索文档，其中 “颜色” 字段是一个同时包含 “红色” 和 “蓝色” 的数组。

创建示例文档

```
db.example.insertMany([
  { "Item": "Pen", "Colors": ["Red", "Blue", "Green"] },
  { "Item": "Notebook", "Colors": ["Blue", "White"] },
  { "Item": "Poster Paint", "Colors": ["Red", "Yellow", "White"] }
])
```

查询示例

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

输出

```
{
  "_id" : ObjectId("6137d6c5b3a1d35e0b6ee6ad"),
  "Item" : "Pen",
  "Colors" : [
          "Red",
          "Blue",
          "Green"
  ]
}
```

代码示例

要查看使用该$all命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$and运算符用于组合多个表达式并将其作为单个条件进行评估。true如果所有提供的表达式的计算结果均为，则返回true，false否则返回。此运算符对于将多个条件应用于查询很有用。

参数

- expression1：计算结果为布尔值的必填表达式。
- expression2：计算结果为布尔值的必填表达式。
- ...：计算结果为布尔值的其他必需表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$and运算符查找 “用户” 集合中的所有文档，其中 “年龄” 字段大于 18 且 “状态” 字段为 “活动”。

创建示例文档

```
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" }
```

```
]);
```

查询示例

```
db.users.find({
  $and: [
    { age: { $gt: 18 } },
    { status: "active" }
  ]
});
```

输出

```
[
  { "_id" : ObjectId("614e3c4b63f5892e7c4e2345"), "name" : "John", "age" : 25,
 "status" : "active" },
  { "_id" : ObjectId("614e3c4b63f5892e7c4e2347"), "name" : "Alice", "age" : 22,
 "status" : "active" }
]
```

代码示例

要查看使用该$and命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$bits AllClear

Amazon DocumentDB 中的$bitsAllClear运算符用于匹配已清除字段中所有指定位（设置为 0）的文档。此运算符可用于对存储的数据执行按位运算。

参数

- `field`：用于检查指定位是否被清除的字段。
- `value`：指定应检查哪些位的数字位掩码，或要检查的位位置列表。数字位掩码可以是二进制 (0b...)、十进制、十六进制 (0x...)、八进制 (0o...) 或二进制 () 形式。BinData在位位置列表中，最低有效位的位置为 0。

示例（MongoDB 外壳）

以下示例演示了$bitsAllClear运算符在 Amazon DocumentDB 中的用法。

创建示例文档

```
db.collection.insertMany([
  { _id: 1, bits: 0b1010 },
  { _id: 2, bits: 0b1100 },
  { _id: 3, bits: 0b0101 }
]);
```

查询示例

```
db.collection.find({
  bits: { $bitsAllClear: 0b0011 }
})
```

输出

```
{ "_id" : 2, "bits" : 12 }
```

该查询会检查bits字段中是否清除了位掩码指定的所有位0b0011（两个最低有效位）。带有 _id 2 的文档满足此条件，因为其bits字段已清除这些位。

代码示例

要查看使用该$bitsAllClear命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$bits AllSet

Amazon DocumentDB 中的$bitsAllSet运算符用于查询字段中特定位集全部设置为 1 的文档。此运算符允许您对字段值执行按位运算，并且在需要检查数值中各个位的状态时非常有用。

参数

- field：要对其执行按位运算的数值字段的名称。
- value：指定应检查哪些位的数字位掩码，或要检查的位位置列表。数字位掩码可以是二进制 (0b...)、十进制、十六进制 (0x...)、八进制 (0o...) 或二进制 () 形式。BinData在位位置列表中，最低有效位的位置为 0。

示例（MongoDB 外壳）

以下示例演示如何使用$bitsAllSet运算符来查找该flags字段设置了由位掩码指定的所有位的文档。

创建示例文档

```
db.collection.insert([
  { _id: 1, flags: 0b1010 },
  { _id: 2, flags: 0b1100 },
  { _id: 3, flags: 0b1110 }
])
```

查询示例

```
db.collection.find({ flags: { $bitsAllSet: 0b1100 } })
```

输出

```
{ "_id": 2, "flags": 12 },
{ "_id": 3, "flags": 14 }
```

在此示例中，查询会检查flags字段中是否包含由位掩码0b1100（表示十进制值 12）指定的所有位的文档。带有 _id 2 和 3 的文档符合此标准，因为它们的flags字段值设置了所有必需的位（第 3 和第 4 个最低有效位）。

代码示例

要查看使用该$bitsAllSet命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$bits AnyClear

Amazon DocumentDB 中的$bitsAnyClear运算符用于匹配清除字段中任何指定位（设置为 0）的文档。这对于对存储在文档中的字段值执行按位运算很有用。

参数

- field：要检查的字段。
- value：指定应检查哪些位的数字位掩码，或要检查的位位置列表。数字位掩码可以是二进制 (0b...)、十进制、十六进制 (0x...)、八进制 (0o...) 或二进制 () 形式。BinData在位位置列表中，最低有效位的位置为 0。

示例（MongoDB 外壳）

以下示例演示如何使用$bitsAnyClear运算符来检查items集合status字段中是否有任何位被清除。

创建示例文档

```
db.items.insertMany([
  { "_id": 1, "status": 7 },
  { "_id": 2, "status": 15 },
  { "_id": 3, "status": 31 }
]);
```

查询示例

```
db.items.find({ "status": { $bitsAnyClear: 8 } })
```

输出

```
{ "_id" : 1, "status" : 7 }
```

在此示例中，该查询会检查该status字段在位掩码（二进制）中是否有清除位 8 (01000) 的文档。status值为7（二进制111）的文档与查询相匹配，因为它在提供的位掩码中至少有一个空位。匹配的清除位是第 4 个最低有效位。

代码示例

要查看使用该$bitsAnyClear命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$bits AnySet

Amazon DocumentDB 中的$bitsAnySet运算符用于查询字段中指定位中至少有一个位设置为 1 的文档。此运算符允许您对存储在字段中的值执行按位运算，从而可以高效查询和分析具有按位特征的数据。

参数

- `field`：要应用按位运算的字段名称。
- `value`：指定应检查哪些位的数字位掩码，或要检查的位位置列表。数字位掩码可以是二进制 (0b...)、十进制、十六进制 (0x...)、八进制 (0o...) 或二进制 () 形式。BinData在位位置列表中，最低有效位的位置为 0。

示例（MongoDB 外壳）

以下示例演示如何使用$bitsAnySet运算符查找flags字段中至少设置了一个位的文档。

创建示例文档

```
db.collection.insertMany([
  { _id: 1, flags: 0b1010 },
  { _id: 2, flags: 0b1100 },
  { _id: 3, flags: 0b0011 },
  { _id: 4, flags: 0b0100 }
]);
```

查询示例

```
db.collection.find({
  flags: { $bitsAnySet: 0b1010 }
})
```

输出

```
{ "_id" : 1, "flags" : 10 }
{ "_id" : 2, "flags" : 12 }
{ "_id" : 3, "flags" : 3 }
```

该查询返回的文档中，该flags字段中至少设置了0b1010位掩码中指定的一个位。

代码示例

要查看使用该$bitsAnySet命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$comment运算符用于为查询添加注释。这对于提供有关查询的其他上下文或信息很有用，这可能有助于调试或记录目的。所附注释将作为诸如 db.currenTop () 之类的操作输出的一部分出现。

参数

- `string`：查询所附的评论。

示例（MongoDB 外壳）

以下示例演示了如何在 Amazon DocumentDB 中使用$comment运算符。

创建示例文档

```
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" }
]);
```

查询示例

```
db.users.find({ age: { $gt: 25 } }, { _id: 0, name: 1, age: 1 }).comment("Retrieve
 users older than 25");
```

输出

```
{ "name" : "John Doe", "age" : 30 }
{ "name" : "Bob Johnson", "age" : 35 }
```

代码示例

要查看使用该$comment命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$elemMatch运算符用于查询数组字段，并返回数组中至少有一个元素符合指定条件的文档。当您拥有包含嵌套数组或嵌入式文档的复杂数据结构时，此运算符特别有用。

Planner 版本 2.0 添加了对索引的支持$elemMatch。

参数

- field：要查询的数组字段。
- query：与数组元素匹配的标准。

在**$all**表达式**$elemMatch**中使用

有关在$all表达式中使用$elemMatch运算符的限制，请参阅# $all ###### $elemMatch。

示例（MongoDB 外壳）

以下示例演示如何使用$elemMatch运算符来查找parts数组中至少有一个符合指定条件的元素的文档。

创建示例文档

```
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 }] }
]);
```

查询示例

```
db.col.find({
  parts: { "$elemMatch": { part: "xyz", qty: { $lt: 11 } } }
})
```

输出

```
{ "_id" : 1, "parts" : [ { "part" : "xyz", "qty" : 10 }, { "part" : "abc", "qty" :
 20 } ] }
{ "_id" : 2, "parts" : [ { "part" : "xyz", "qty" : 5 }, { "part" : "abc", "qty" :
 10 } ] }
```

代码示例

要查看使用该$elemMatch命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$eq运算符用于匹配字段值等于指定值的文档。该运算符通常用于检索符合指定条件的文档find()的方法。

参数

- field：要检查相等条件的字段。
- value：要与字段进行比较的值。

示例（MongoDB 外壳）

以下示例演示如何使用$eq运算符来查找该name字段等于的所有文档"Thai Curry Palace"。

创建示例文档

```
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"] }
]);
```

查询示例

```
db.restaurants.find({ name: { $eq: "Thai Curry Palace" } });
```

输出

```
{ "_id" : ObjectId("68ee586f916df9d39f3d9414"), "name" : "Thai Curry Palace",
 "cuisine" : "Thai", "features" : [ "Private Dining" ] }
```

代码示例

要查看使用该$eq命令的代码示例，请选择要使用的语言的选项卡：

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

该$exists运算符用于检查文档中是否存在字段。 $exists在 Amazon DocumentDB 中处理稀疏索引时特别有用，因为索引字段可能不会出现在所有文档中。

要使用您在查询中创建的稀疏索引，必须在涵盖索引的字段中使用 $exists 子句。如果省略$exists，Amazon DocumentDB 将不会使用稀疏索引进行查询。

参数

- field：要检查是否存在的字段名称。
- value：一个布尔值（true或false），用于指定匹配文档中该字段应存在 (truefalse) 还是不存在 ()。

示例（MongoDB 外壳）

以下示例演示了如何在food集合中的special_diets字段上使用带有稀疏索引的$exists运算符。

创建示例文档

```
db.food.insertMany([
  { _id: 1, name: "Apple", special_diets: ["vegetarian", "gluten-free"] },
  { _id: 2, name: "Broccoli" },
  { _id: 3, name: "Chicken", special_diets: ["dairy-free"] }
]);
```

在 “special_diets” 字段上创建稀疏索引

```
db.food.createIndex({ "special_diets": 1 }, { sparse: true, name:
 "special_diets_sparse_asc" });
```

查询示例

```
db.food.find({ "special_diets": { $exists: true } });
```

输出

```
[
  { "_id" : 1, "name" : "Apple", "special_diets" : [ "vegetarian", "gluten-free" ] },
  { "_id" : 3, "name" : "Chicken", "special_diets" : [ "dairy-free" ] }
]
```

代码示例

要查看使用该$exists命令的代码示例，请选择要使用的语言的选项卡：

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

4.0 版的新增内容。

弹性集群不支持。

Amazon DocumentDB 中的$expr运算符允许您在查询语言中使用聚合表达式。它使您能够对文档中的字段执行复杂的比较和计算，类似于使用聚合管道阶段的方式。

参数

- expression：返回布尔值的表达式，允许您对文档字段执行比较和计算。

示例（MongoDB 外壳）

以下示例演示如何使用$expr运算符来查找manufacturingCost字段大于该price字段的所有文档。

创建示例文档

```
db.inventory.insertMany([
  { item: "abc", manufacturingCost: 500, price: 100 },
  { item: "def", manufacturingCost: 300, price: 450 },
  { item: "ghi", manufacturingCost: 400, price: 120 }
]);
```

查询示例

```
db.inventory.find({
  $expr: {
    $gt: ["$manufacturingCost", "$price"]
  }
})
```

输出

```
{ "_id" : ObjectId("60b9d4d68d2cac581bc5a89a"), "item" : "abc", "manufacturingCost" :
 500, "price" : 100 },
{ "_id" : ObjectId("60b9d4d68d2cac581bc5a89c"), "item" : "ghi", "manufacturingCost" :
 400, "price" : 120 }
```

代码示例

要查看使用该$expr命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$gt运算符用于选择指定字段值大于指定值的文档。此运算符可用于根据数值比较筛选和查询数据。

参数

- field：要比较的字段。
- value：要比较的值。

示例（MongoDB 外壳）

以下示例演示如何使用$gt运算符查找age字段大于 30 的所有文档。

创建示例文档

```
db.users.insertMany([
  { name: "John", age: 25 },
  { name: "Jane", age: 32 },
  { name: "Bob", age: 45 },
  { name: "Alice", age: 28 }
]);
```

查询示例

```
db.users.find({ age: { $gt: 30 } });
```

输出

```
{ "_id" : ObjectId("6249e5c22a5d39884a0a0001"), "name" : "Jane", "age" : 32 },
{ "_id" : ObjectId("6249e5c22a5d39884a0a0002"), "name" : "Bob", "age" : 45 }
```

代码示例

要查看使用该$gt命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$gte运算符用于匹配大于或等于指定值的值。此运算符可用于根据数值比较筛选和查询数据。

参数

- `field`：要根据提供的值进行检查的字段。
- `value`：要与字段进行比较的值。

示例（MongoDB 外壳）

以下示例演示如何在 Amazon DocumentDB 中使用$gte运算符来查找 “年龄” 字段大于或等于 25 的所有文档。

创建示例文档

```
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 }
]);
```

查询示例

```
db.users.find({ age: { $gte: 25 } }, { _id: 0, name: 1, age: 1 });
```

输出

```
{ "name" : "Jane", "age" : 25 }
{ "name" : "Bob", "age" : 30 }
{ "name" : "Alice", "age" : 35 }
```

代码示例

要查看使用该$gte命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的运算符是一种逻辑查询运算符，它允许您查找字段值等于数组中指定的任何值的文档。$in

参数

- field：要对照提供的数组进行检查的字段。
- [value1, value2, ...]：要与指定字段匹配的值数组。

字段名称中的美元 (**$**)

有关在嵌套对象中查询$前缀字段的限制$in，请参阅字段名称中的美元符号（$）和句点（.）。

示例（MongoDB 外壳）

以下示例演示如何使用$in运算符来查找该color字段是所提供数组中值之一的文档。

创建示例文档

```
db.colors.insertMany([
  { "_id": 1, "color": "red" },
  { "_id": 2, "color": "green" },
  { "_id": 3, "color": "blue" },
  { "_id": 4, "color": "yellow" },
  { "_id": 5, "color": "purple" }
])
```

查询示例

```
db.colors.find({ "color": { "$in": ["red", "blue", "purple"] } })
```

输出

```
{ "_id": 1, "color": "red" },
{ "_id": 3, "color": "blue" },
{ "_id": 5, "color": "purple" }
```

代码示例

要查看使用该$in命令的代码示例，请选择要使用的语言的选项卡：

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

4.0 版的新增内容。

弹性集群不支持。

Amazon DocumentDB 中的$jsonSchema运算符用于根据指定的 JSON 架构筛选文档。此运算符允许您查询与特定 JSON 架构匹配的文档，确保检索到的文档符合特定的结构和数据类型要求。

在创建集合时使用$jsonSchema评估查询运算符，可以验证插入到集合中的文档的架构。有关更多信息，请参阅使用 JSON 架构验证。

参数

- required（数组）：指定文档中的必填字段。
- properties（object）：定义文档中每个字段的数据类型和其他约束。

示例（MongoDB 外壳）

以下示例演示如何使用$jsonSchema运算符筛选employees集合，以仅检索具有name、employeeId和age字段且employeeId字段类型为的文档string。

创建示例文档

```
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" }
]);
```

查询示例

```
db.employees.aggregate([
  { $match: {
    $jsonSchema: {
      required: ["name", "employeeId", "age"],
      properties: { "employeeId": { "bsonType": "string" } }
    }
  }}
]);
```

输出

```
{ "_id" : ObjectId("6908e8b61f77fc26b2ecd26f"), "name" : { "firstName" : "Emily",
 "lastName" : "Brown" }, "employeeId" : "2", "age" : 25 }
```

代码示例

要查看使用该$jsonSchema命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$lt运算符用于匹配小于指定值的值。此运算符可用于根据数值比较筛选和查询数据。

参数

- field：要检查的字段名称。
- value：要比较的值。

示例（MongoDB 外壳）

以下示例演示如何使用$lt运算符检索Inventory.OnHand值小于 50 的文档。

创建示例文档

```
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"] }
]);
```

查询示例

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

输出

```
{ "_id" : 1, "Inventory" : { "OnHand" : 25, "Sold" : 60 }, "Colors" : [ "Red",
 "Blue" ] }
{ "_id" : 3, "Inventory" : { "OnHand" : 10, "Sold" : 85 }, "Colors" : [ "Red",
 "Green" ] }
```

代码示例

要查看使用该$lt命令的代码示例，请选择要使用的语言的选项卡：

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查询运算符选择字段值除以除数具有指定余数的文档。这对于根据模算术条件筛选文档很有用。

参数

- `divisor`: 要除以的数字。

- `remainder`：预期的剩余值。

示例（MongoDB 外壳）

以下示例演示如何使用$mod运算符查找数量为奇数的所有订单。

创建示例文档

```
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 }
]);
```

查询示例

```
db.orders.find({ quantity: { $mod: [2, 1] } });
```

输出

```
{ "_id" : 1, "item" : "Widget", "quantity" : 15 }
{ "_id" : 3, "item" : "Tool", "quantity" : 7 }
{ "_id" : 5, "item" : "Part", "quantity" : 9 }
```

此查询返回的文档中，数量除以 2 的余数为 1，实际上选择了所有奇数。

代码示例

要查看使用$mod查询运算符的代码示例，请选择要使用的语言对应的选项卡：

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 DocumentDB 中的`$lte`运算符用于匹配指定字段值小于或等于指定值的文档。此运算符对于根据数值比较筛选和查询数据非常有用。

参数

- `field`：要比较的字段。
- `value`：要比较的值。

示例（MongoDB 外壳）

以下示例演示如何使用`$lte`运算符来检索`quantity`字段小于或等于 10 的文档。

创建示例文档

```
db.inventory.insertMany([
  { item: "canvas", qty: 100 },
  { item: "paint", qty: 50 },
  { item: "brush", qty: 10 },
  { item: "paper", qty: 5 }
]);
```

查询示例

```
db.inventory.find({ qty: { $lte: 10 } });
```

输出

```
{ "_id" : ObjectId("..."), "item" : "brush", "qty" : 10 },
{ "_id" : ObjectId("..."), "item" : "paper", "qty" : 5 }
```

代码示例

要查看使用该$lte命令的代码示例，请选择要使用的语言的选项卡：

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

该$meta运算符用于访问与当前查询执行相关的元数据。此运算符主要用于文本搜索操作，其中元数据可以提供有关匹配文档相关性的信息。

参数

- textScore：检索文档的文本搜索分数。该分数表示文档与文本搜索查询的相关性。

示例（MongoDB 外壳）

以下示例演示如何使用$meta运算符检索与文本搜索查询相匹配的文档的文本搜索分数。

创建示例文档

```
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." }
```

```
]);
```

创建文本索引

```
db.documents.createIndex({ content: "text" });
```

查询示例

```
db.documents.find(
  { $text: { $search: "coffee" } },
  { _id: 0, title: 1, content: 1, score: { $meta: "textScore" } }
).sort({ score: { $meta: "textScore" } });
```

输出

```
[
  {
    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
  }
]
```

代码示例

要查看使用该$meta命令的代码示例，请选择要使用的语言的选项卡：

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

该$ne运算符用于匹配字段值不等于指定值的文档。它是一个比较运算符，可以在查询谓词中用于筛选文档。

Planner 版本 2.0 增加了对索引的支持$ne。

参数

- `field`：要检查的字段。
- `value`：要检查的值。

示例（MongoDB 外壳）

在此示例中，我们将查找`users`集合中该`status`字段不等于的所有文档`"active"`。

创建示例文档

```
db.users.insertMany([
  { name: "John", status: "active" },
  { name: "Jane", status: "inactive" },
  { name: "Bob", status: "suspended" },
  { name: "Alice", status: "active" }
]);
```

查询示例

```
db.users.find({ status: { $ne: "active" } });
```

输出

```
[
  {
    _id: ObjectId('...'),
    name: 'Jane',
    status: 'inactive'
  },
  {
    _id: ObjectId('...'),
    name: 'Bob',
    status: 'suspended'
  }
]
```

代码示例

要查看使用该`$ne`命令的代码示例，请选择要使用的语言的选项卡：

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

该$nin运算符用于匹配不在指定数组中的值。它是$in运算符的倒数，它匹配指定数组中的值。

Planner 版本 2.0 添加了对索引的支持$nin。

参数

- field：要检查的字段。
- array：要检查的值数组。

字段名称中的美元 ($)

有关在嵌套对象中查询$前缀字段的限制$nin，请参阅字段名称中的美元符号（$）和句点（.）。

示例（MongoDB 外壳）

以下示例演示如何使用$nin运算符来查找category字段不等于 “虚构” 或 “神秘” 的文档。

创建示例文档

```
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" }
])
```

查询示例

```
db.books.find({
  category: {
    $nin: ["Fiction", "Mystery"]
  }
})
```

输出

```
[
  {
    _id: ObjectId('...'),
    title: 'The Martian',
    author: 'Andy Weir',
    category: 'Science Fiction'
  },
```

```
  {
    _id: ObjectId('...'),
    title: 'The Alchemist',
    author: 'Paulo Coelho',
    category: 'Philosophy'
  }
]
```

代码示例

要查看使用该$nin命令的代码示例，请选择要使用的语言的选项卡：

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

该$nor运算符用于匹配所有指定查询条件均不为真的文档。它类似于逻辑 “NOR” 运算，如果所有操作数都不为真，则结果为真。

参数

- expression1：要计算的第一个表达式。
- expression2：要计算的第二个表达式。
- expressionN：其他要评估的表达式。

示例（MongoDB 外壳）

以下示例通过检索字段不小于 20 且qty字段不等于 “XL” 的文档来演示$nor运算符的用法。size

创建示例文档

```
db.items.insertMany([
  { qty: 10, size: "M" },
  { qty: 15, size: "XL" },
  { qty: 25, size: "L" },
  { qty: 30, size: "XL" }
])
```

查询示例

```
db.items.find({
  $nor: [
    { qty: { $lt: 20 } },
    { size: "XL" }
  ]
```

```
})
```

输出

```
[
  { "_id" : ObjectId("..."), "qty" : 25, "size" : "L" }
]
```

代码示例

要查看使用该$nor命令的代码示例，请选择要使用的语言的选项卡：

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

该$not运算符用于否定给定表达式的结果。它允许您选择与指定条件不匹配的文档。

Planner 版本 2.0 增加了对$not {eq}和的索引支持$not {in}。

参数

- expression: 要否定的表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$not运算符来查找status字段不等于 “active” 的文档。

创建示例文档

```
db.users.insertMany([
  { name: "John", status: "active" },
  { name: "Jane", status: "inactive" },
  { name: "Bob", status: "pending" },
  { name: "Alice", status: "active" }
```

```
]);
```

查询示例

```
db.users.find({ status: { $not: { $eq: "active" } } });
```

输出

```
[
  {
    _id: ObjectId('...'),
    name: 'Jane',
    status: 'inactive'
  },
  {
    _id: ObjectId('...'),
    name: 'Bob',
    status: 'pending'
  }
]
```

代码示例

要查看使用该$not命令的代码示例，请选择要使用的语言的选项卡：

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

该$or运算符用于对包含两个或更多表达式的数组执行逻辑 OR 运算。它返回至少与其中一个表达式匹配的文档。当您需要查询满足多个条件中任意一个的文档时，此运算符很有用。

参数

- expression1：要计算的第一个表达式。
- expression2：要计算的第二个表达式。
- ...：要评估的其他表达式（可选）。

示例（MongoDB 外壳）

以下示例演示了如何使用$or运算符来查找文档，其中模型为 “H make eavy H1TruckForYou” 时为 “”，或者模型为 “Bolid 1SportForYou” 时为 “”。

创建示例文档

```
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 }
]);
```

查询示例

```
db.cars.find({
  $or: [
    { make: "TruckForYou", model: "Heavy H1" },
    { make: "SportForYou", model: "Bolid 1" }
  ]
});
```

输出

```
[
  {
    _id: ObjectId('...'),
    make: 'TruckForYou',
    model: 'Heavy H1',
    year: 2020
  },
  {
    _id: ObjectId('...'),
    make: 'SportForYou',
    model: 'Bolid 1',
    year: 2021
  }
]
```

代码示例

要查看使用该$or命令的代码示例，请选择要使用的语言的选项卡：

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

该$regex运算符允许您对字符串字段执行正则表达式匹配。它是基于复杂模式搜索和筛选文档的强大工具。

参数

- `regular expression`：要与字段匹配的正则表达式模式。
- `$options`:（可选）提供修改搜索行为的选项，例如区分大小写、全局匹配等。

示例（MongoDB 外壳）

以下示例演示如何使用$regex运算符来搜索 “名称” 字段与特定模式匹配的文档。

创建示例文档

```
db.users.insertMany([
  { name: "John Doe" },
  { name: "Jane Smith" },
  { name: "Alice Johnson" },
  { name: "Bob Williams" },
  { name: "Charlie Davis" }
]);
```

查询示例

```
db.users.find({ name: { $regex: /^A/ } })
```

输出

```
[
  { "_id" : ObjectId("..."), "name" : "Alice Johnson" }
]
```

此查询将返回 “名称” 字段以字母 “A” 开头的所有文档。

代码示例

要查看使用该$regex命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$slice

$slice投影运算符限制查询结果中返回的数组元素的数量。它允许您从数组字段的开头或结尾检索特定数量的元素，而无需加载整个数组。

参数

- field：要投影的数组字段。
- count：要返回的元素数量。正值从一开始返回元素，从结尾返回负值。

示例（MongoDB 外壳）

以下示例演示如何使用$slice投影运算符仅返回数组字段中的前两项。

创建示例文档

```
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"] }
]);
```

查询示例

```
db.inventory.find(
  {},
  { item: 1, tags: { $slice: 2 } }
)
```

输出

```
{ "_id" : 1, "item" : "notebook", "tags" : [ "office", "school" ] }
{ "_id" : 2, "item" : "pen", "tags" : [ "office", "writing" ] }
{ "_id" : 3, "item" : "folder", "tags" : [ "office", "supplies" ] }
```

代码示例

要查看使用$slice投影运算符的代码示例，请选择要使用的语言的选项卡：

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查询运算符匹配数组字段具有精确指定数量的元素的文档。这对于根据数组长度筛选文档很有用。

参数

- field：要检查的数组字段。
- count：数组必须包含的元素的确切数量。

示例（MongoDB 外壳）

以下示例演示如何使用$size运算符查找恰好具有三个标签的所有产品。

创建示例文档

```
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"] }
]);
```

查询示例

```
db.products.find({ tags: { $size: 3 } });
```

输出

```
{ "_id" : 1, "name" : "Laptop", "tags" : [ "electronics", "computers", "portable" ] }
{ "_id" : 3, "name" : "Desk", "tags" : [ "furniture", "office", "workspace" ] }
```

代码示例

要查看使用$size查询运算符的代码示例，请选择要使用的语言对应的选项卡：

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

该$text运算符用于对文档集合中的文本索引字段执行全文搜索。此运算符允许您搜索包含特定单词或短语的文档，并且可以与其他查询运算符结合使用以根据其他条件筛选结果。

参数

- $search：要搜索的文本字符串。

示例（MongoDB 外壳）

以下示例演示如何使用$text运算符搜索包含 “兴趣” 一词的文档，并根据 “star_rating” 字段筛选结果。

创建示例文档

```
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" }
]);
```

创建文本索引

```
db.test.createIndex({ comments: "text" });
```

查询示例

```
db.test.find({$and: [{star_rating: 5}, {$text: {$search: "interest"}}]})
```

输出

```
{ "_id" : 5, "star_rating" : 5, "comments" : "interesting couch" }
{ "_id" : 6, "star_rating" : 5, "comments" : "interested in couch for sale, year
 2022" }
```

上面的命令返回的文档的文本索引字段包含任何形式的 “兴趣”，“star_rating” 等于 5。

代码示例

要查看使用该$text命令的代码示例，请选择要使用的语言的选项卡：

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

运$type算符用于检查文档中字段的数据类型。它可以在需要特定于类型的操作或验证时使用。运$type算符返回已计算表达式的 BSON 类型。返回的类型是一个字符串，它对应于字段或表达式的类型。

Planner 版本 2.0 增加了对索引的支持$type。

参数

- expression：要计算的表达式。

示例（MongoDB 外壳）

创建示例文档

```
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 }
]);
```

查询示例

```
db.documents.find({
  $or: [
    { age: { $type: "number" } },
    { email: { $type: "string" } },
    { name: { $type: "string" } }
  ]
})
```

输出

```
[
  { "_id": 1, "name": "John", "age": 30, "email": "john@example.com" },
  { "_id": 2, "name": "Jane", "age": "25", "email": 123456 }
]
```

此查询将返回age字段类型为 “数字”、email字段类型为 “字符串”、name字段类型为 “字符串” 的文档。

代码示例

要查看使用该$type命令的代码示例，请选择要使用的语言的选项卡：

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()
```

更新运算符

本节提供有关亚马逊 DocumentDB 支持的更新操作员的详细信息。

主题

- $
- $[]
- $ [<标识符>]
- $add ToSet
- $bit
- $currentDate
- $each
- $inc
- $max
- $min
- $mul
- $pop
- $position
- $pull
- $pullAll
- $push
- $rename
- $set

- $set OnInsert
- $slice
- $sort
- $unset

$

$位置运算符更新第一个匹配查询条件的数组元素。它充当匹配数组元素位置的占位符。

参数

- `field.$`: 带有位置运算符的数组字段，用于更新第一个匹配的元素。

示例（MongoDB 外壳）

以下示例演示了使用$位置运算符来更新特定的数组元素。

创建示例文档

```
db.inventory.insertMany([
  { _id: 1, item: "Widget", quantities: [10, 20, 30] },
  { _id: 2, item: "Gadget", quantities: [5, 15, 25] }
]);
```

查询示例

```
db.inventory.updateOne(
  { _id: 1, quantities: 20 },
  { $set: { "quantities.$": 22 } }
);
```

输出

```
{
  "_id" : 1,
  "item" : "Widget",
  "quantities" : [ 10, 22, 30 ]
}
```

代码示例

要查看使用$位置运算符的代码示例，请选择要使用的语言的选项卡：

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()
```

$[]

al $[] l 位置运算符更新数组中的所有元素。当你需要修改数组字段中的每个元素时，就会使用它。

参数

- `field.$[]`: 带有 all 位置运算符的数组字段，用于更新所有元素。

示例（MongoDB 外壳）

以下示例演示如何使用$[]运算符更新所有数组元素。

创建示例文档

```
db.products.insertOne({
  _id: 1,
  name: "Laptop",
  prices: [1000, 1100, 1200]
});
```

查询示例

```
db.products.updateOne(
  { _id: 1 },
  { $inc: { "prices.$[]": 50 } }
);
```

输出

```
{
  "_id" : 1,
  "name" : "Laptop",
  "prices" : [ 1050, 1150, 1250 ]
```

```
}
```

代码示例

要查看使用$[]运算符的代码示例，请选择要使用的语言对应的选项卡：

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()
```

$ [<标识符>]

$[<identifier>]过滤后的位置运算符会更新所有符合指定过滤条件的数组元素。它与有选择地更新数组元素的arrayFilters选项一起使用。

参数

- field.$[identifier]: 带有筛选位置运算符的数组字段。
- arrayFilters：一组筛选条件，用于确定要更新的元素。

示例（MongoDB 外壳）

以下示例演示如何使用$[<identifier>]运算符根据条件更新特定的数组元素。

创建示例文档

```
db.students.insertOne({
  _id: 1,
  name: "Alice",
  grades: [
    { subject: "Math", score: 85 },
    { subject: "Science", score: 92 },
    { subject: "History", score: 78 }
  ]
});
```

查询示例

```
db.students.updateOne(
  { _id: 1 },
```

```
  { $inc: { "grades.$[elem].score": 5 } },
  { arrayFilters: [{ "elem.score": { $gte: 80 } }] }
);
```

输出

```
{
  "_id" : 1,
  "name" : "Alice",
  "grades" : [
    { "subject" : "Math", "score" : 90 },
    { "subject" : "Science", "score" : 97 },
    { "subject" : "History", "score" : 78 }
  ]
}
```

代码示例

要查看使用$[<identifier>]运算符的代码示例，请选择要使用的语言的选项卡：

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()
```

$add ToSet

仅当数组中尚不存在值时，Amazon DocumentDB 中的$addToSet运算符才用于向数组添加值。这对于确保数组包含唯一元素很有用。

参数

- `field`：要更新的字段。
- `value`：要添加到数组字段的值。这可以是单个值或表达式。

示例（MongoDB 外壳）

以下示例演示如何使用$addToSet运算符向数组添加唯一元素。

创建示例文档

```
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"] }
])
```

查询示例

```
db.products.update(
  { "item": "apple" },
  { $addToSet: { "tags": "green" } }
)
```

输出

```
{ "_id": 1, "item": "apple", "tags": ["fruit", "red", "round", "green"] }
```

在此示例中，$addToSet操作员将 “绿色” 标签添加到文档的 “标签” 数组中，其中 “项目” 字段为 “apple”。由于 “green” 不在数组中，因此添加了它。

代码示例

要查看使用该$addToSet命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的运$bit算符允许您对给定字段的位执行按位运算。这对于设置、清除或检查数字中各个位的状态之类的任务很有用。

参数

- `field`：要对其执行按位运算的字段。
- `and`：用于对字段执行按位与运算的整数值。
- `or`：用于对字段执行按位或运算的整数值。
- `xor`：用于对字段执行按位异或运算的整数值。

示例（MongoDB 外壳）

以下示例演示如何使用$bit运算符对数值字段执行按位运算。

创建示例文档

```
db.numbers.insert([
  { "_id": 1, "number": 5 },
  { "_id": 2, "number": 12 }
])
```

查询示例

```
db.numbers.update(
  { "_id": 1 },
  { "$bit": { "number": { "and": 3 } } }
)
```

输出

```
{
  "_id": 1,
  "number": 1
}
```

在此示例中，$bit运算符用于对文档的 “数字” 字段执行按位与运算，其值为 1。_id结果是 “数字” 字段的值设置为 1，这是在原始值 5 和值 3 之间按位运算的结果。

代码示例

要查看使用该$bit命令的代码示例，请选择要使用的语言的选项卡：

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

运$currentDate算符用于将字段的值设置为当前日期和时间。此运算符对于在插入或更新文档时使用当前时间戳自动更新字段很有用。

参数

- field：要使用当前日期和时间更新的字段。
- type:（可选）指定用于当前日期的 BSON 类型。可以是 date 或 timestamp。

示例（MongoDB 外壳）

以下示例演示如何使用$currentDate运算符将lastModified字段设置为插入新文档时的当前日期和时间。

创建示例文档

```
db.users.insert({
  name: "John Doe",
  email: "john.doe@example.com"
})
```

查询示例

```
db.users.updateOne(
  { name: "John Doe" },
  { $currentDate: { lastModified: true } }
)
```

查看更新的文档

```
db.users.findOne({ name: "John Doe" })
```

输出

```
{
  _id: ObjectId('...'),
  name: 'John Doe',
  email: 'john.doe@example.com',
  lastModified: ISODate('2025-10-25T22:50:29.963Z')
}
```

代码示例

要查看使用该$currentDate命令的代码示例，请选择要使用的语言的选项卡：

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

该$each运算符与其他更新运算符（例如$push和）结合使用$addToSet，向数组字段添加多个值。它允许在单个操作中向数组添加多个元素，而不必执行多个更新操作。

参数

- value：要添加到数组字段的值数组。

示例（MongoDB 外壳）

以下示例演示如何使用带有$each运算符的$push运算符将多个元素添加到数组字段中。

创建示例文档

```
db.fruits.insertOne({
  _id: 1,
  fruits: ["apple", "banana"]
})
```

查询示例

```
db.fruits.updateOne(
  { _id: 1 },
  { $push: { fruits: { $each: ["cherry", "durian", "elderberry"] } } }
)
```

查看更新的文档

```
db.fruits.findOne({ _id: 1 })
```

输出

```
{
  _id: 1,
  fruits: [ 'apple', 'banana', 'cherry', 'durian', 'elderberry' ]
}
```

代码示例

要查看使用该$each命令的代码示例，请选择要使用的语言的选项卡：

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

该$inc运算符用于将字段的值增加指定量。它用于更新数值字段，例如计数器或评级，而无需检索当前值、计算新值然后更新该字段。

参数

- field：要递增的字段的名称。
- amount：该字段的增量值。这可以是正值或负值。

示例（MongoDB 外壳）

以下示例演示如何使用$inc运算符来增加文档的age字段。

创建示例文档

```
db.users.insertOne({_id: 123, name: "John Doe", age: 30})
```

查询示例

```
db.users.updateOne({_id: 123}, {$inc: {age: 1}})
```

查看更新的文档

```
db.users.findOne({_id: 123})
```

输出

```
{ "_id" : 123, "name" : "John Doe", "age" : 31 }
```

代码示例

要查看使用该$inc命令的代码示例，请选择要使用的语言的选项卡：

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

只有当指定值大于当前字段值时，$max更新运算符才会更新字段的值。此运算符对于在更新期间保持最大值很有用。

参数

- field：要更新的字段。
- value：要与当前字段值进行比较的值。

示例（MongoDB 外壳）

以下示例演示如何使用$max运算符更新玩家记录的最高分数。

创建示例文档

```
db.scores.insertMany([
  { _id: 1, player: "Alice", highScore: 85 },
  { _id: 2, player: "Bob", highScore: 92 },
  { _id: 3, player: "Charlie", highScore: 78 }
])
```

更新示例

```
db.scores.updateOne(
  { _id: 1 },
  { $max: { highScore: 95 } }
)
```

结果

Alice 的highScore字段更新为 95，因为 95 大于当前值 85。

```
{ "_id": 1, "player": "Alice", "highScore": 95 }
```

代码示例

要查看使用该$max命令的代码示例，请选择要使用的语言的选项卡：

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

只有当指定值小于当前字段值时，$min更新运算符才会更新字段的值。此运算符对于在更新期间保持最小值很有用。

参数

- field：要更新的字段。
- value：要与当前字段值进行比较的值。

示例（MongoDB 外壳）

以下示例演示如何使用$min操作员更新气象站记录的最低温度。

创建示例文档

```
db.weather.insertMany([
  { _id: 1, station: "Station A", lowestTemp: 15 },
  { _id: 2, station: "Station B", lowestTemp: 20 },
  { _id: 3, station: "Station C", lowestTemp: 18 }
])
```

更新示例

```
db.weather.updateOne(
  { _id: 1 },
  { $min: { lowestTemp: 12 } }
)
```

结果

站点 A 的lowestTemp字段更新为 12，因为 12 小于当前值 15。

```
{ "_id": 1, "station": "Station A", "lowestTemp": 12 }
```

代码示例

要查看使用该$min命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$mul运算符用于将字段的值乘以指定的数字。这对于自动一致地更新多个文档非常有用，例如根据信用卡状态更新飞行里程。

参数

- field：要相乘的字段。
- multiplier：字段值乘以的数字。

示例（MongoDB 外壳）

此示例演示如何使用$mul运算符将该credit_card字段所在的所有文档的flight_miles值加倍true。

创建示例文档

```
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] }
]);
```

查询示例

```
db.miles.update(
  { "credit_card": { "$eq": true } },
  { "$mul": { "flight_miles.$[]": NumberInt(2) } },
  { "multi": true }
);
```

输出

```
{ "_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 ] }
```

对于持有信用卡的乘客，他们的飞行里程翻了一番。

$[]位置数组运算符用于将$mul运算应用于flight_miles数组中的每个元素。

代码示例

要查看使用该$mul命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$pop运算符用于从数组字段中删除第一个或最后一个元素。当您需要维护固定大小的数组或在文档中实现类似队列的数据结构时，它特别有用。

参数

- `field`: 要从中移除元素的数组字段的名称。
- `value`：一个整数值，用于确定要移除的元素的位置。值为1移除最后一个元素，而值为-1移除第一个元素。

示例（MongoDB 外壳）

此示例演示如何使用$pop运算符从数组字段中删除第一个和最后一个元素。

创建示例文档

```
db.users.insertMany([
  { "_id": 1, "name": "John Doe", "hobbies": ["reading", "swimming", "hiking"] },
  { "_id": 2, "name": "Jane Smith", "hobbies": ["cooking", "gardening", "painting"] }
])
```

查询示例

```
// 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 } })
```

输出

```
{ "_id" : 1, "name" : "John Doe", "hobbies" : [ "swimming", "hiking" ] }
{ "_id" : 2, "name" : "Jane Smith", "hobbies" : [ "cooking", "gardening" ] }
```

代码示例

要查看使用该$pop命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$position修饰符指定$push运算符在数组中插入元素的位置。如果不使用$position修饰$push符，则运算符将元素插入数组的末尾。

参数

- field：要更新的数组字段。
- num: 数组中应插入元素的位置，基于从零开始的索引。

注意：要使用$position修饰符，它必须与修$each饰符一起出现。

示例（MongoDB 外壳）

以下示例演示如何使用$position操作员在项目管理系统的特定位置插入任务。

创建示例文档

```
db.projects.insertOne({ "_id": 1, "name": "Website Redesign", "tasks": ["Design
 mockups"] })
```

查询示例 1-在开头添加紧急任务

```
db.projects.updateOne(
   { _id: 1 },
   {
     $push: {
        tasks: {
           $each: ["Security audit", "Performance review"],
           $position: 0
        }
     }
   }
)
```

输出 1

```
{ "_id": 1, "name": "Website Redesign", "tasks": ["Security audit", "Performance
 review", "Design mockups"] }
```

查询示例 2-在特定位置添加任务

```
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
        }
     }
   }
)
```

输出 2

```
{ "_id": 2, "name": "Mobile App", "tasks": ["Setup project", "Create wireframes", "Code
 review", "Testing phase", "Deploy to store"] }
```

代码示例

要查看使用该$position命令的代码示例，请选择要使用的语言的选项卡：

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

该$pull运算符用于从数组中删除一个或多个与指定条件匹配的值的所有实例。当您需要从文档中的数组字段中删除特定元素时，此运算符很有用。

参数

- `field`：要从中删除值的数组字段的名称。
- `value`：决定要从数组中删除哪些元素的值或条件。

示例（MongoDB 外壳）

以下示例演示如何使用$pull运算符从数组字段中移除元素。

创建示例文档

```
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"]
  }
])
```

查询示例

```
db.restaurants.updateMany(
  { cuisine: "Italian" },
  { $pull: { features: "Takeout" } }
)
```

输出

```
{
  "acknowledged" : true,
  "matchedCount" : 1,
  "modifiedCount" : 1
}
```

上面的查询从所有cuisine字段为 “意大利语” 的文档中删除 “Takeout” 功能。

代码示例

要查看使用该$pull命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$pullAll运算符用于从数组字段中删除指定值的所有实例。当你需要在一次操作中从数组中移除多个元素时，这特别有用。

参数

- field：要从中移除元素的数组字段的名称。
- value：要从数组字段中移除的值数组。

示例（MongoDB 外壳）

以下示例演示如何使用$pullAll运算符从数组字段中移除多个元素。

创建示例文档

```
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"]
  }
])
```

查询示例

```
db.restaurants.update(
  { "name": "Taj Mahal" },
  { $pullAll: { "features": ["Private Dining", "Live Music"] } }
)
```

输出

```
{
  "name": "Taj Mahal",
  "cuisine": "Indian",
  "features": []
}
```

代码示例

要查看使用该$pullAll命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$push运算符用于向文档的数组字段中添加项目。当您需要在不覆盖整个数组的情况下向现有数组追加新数据时，此运算符特别有用。

参数

- `field`: 应向其添加新元素的数组字段的名称。
- `value`：要添加到数组中的值。
- `position`:（可选）一个修饰符，用于指定应在数组中添加新元素的位置。支持的修饰符包括`$`（添加到数组末尾）和`$[]`（添加到数组末尾，忽略任何数组过滤器）。

示例（MongoDB 外壳）

以下示例演示如何使用$push运算符向文档中的数组字段添加新元素。

创建示例文档

```
db.users.insert([
  { _id: 1, name: "John Doe", hobbies: ["reading", "swimming"] },
  { _id: 2, name: "Jane Smith", hobbies: ["gardening", "cooking"] }
```

```
])
```

查询示例

```
db.users.updateOne(
  { _id: 1 },
  { $push: { hobbies: "hiking" } }
)
```

输出

```
{
  "acknowledged" : true,
  "matchedCount" : 1,
  "modifiedCount" : 1
}
```

运行更新后，带有的文档的hobbies数组_id: 1将更新为["reading", "swimming", "hiking"]。

代码示例

要查看使用该$push命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$rename运算符用于重命名文档中的字段。当您需要更新文档结构或将其与新的数据模型对齐时，此运算符可能特别有用。

参数

- field：要重命名的字段。
- newName：该字段的新名称。

示例（MongoDB 外壳）

以下示例演示如何使用$rename运算符在"Date.DoW"字段设置为的文档"Date.DayOfWeek"中将该"DocName"字段重命名为"Document 1"。

创建示例文档

```
db.example.insertOne({
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DoW": "Saturday"
    },
    "Words": 2482
})
```

查询示例

```
db.example.update(
    { "DocName": "Document 1" },
    { $rename: { "Date.DoW": "Date.DayOfWeek" } }
)
```

输出

```
{
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DayOfWeek": "Saturday"
    },
    "Words": 2482
}
```

代码示例

要查看使用该$rename命令的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$set运算符用于更新文档中指定字段的值。此运算符允许您在文档中添加新字段或修改现有字段。它是 MongoDB Java 驱动程序中的基本更新操作符，与亚马逊 DocumentDB 兼容。

参数

- field：要更新的字段。
- value：该字段的新值。

示例（MongoDB 外壳）

以下示例演示如何使用$set运算符更新文档中的Item字段。

创建示例文档

```
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
    }
  }
])
```

查询示例

```
db.example.update(
  { "Item": "Pen" },
```

```
  { $set: { "Item": "Gel Pen" } }
)
```

输出

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

代码示例

要查看使用该$set命令的代码示例，请选择要使用的语言的选项卡：

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()
```

$set OnInsert

如果要插入文档，Amazon DocumentDB 中的`$setOnInsert`运算符用于设置字段的值，但是如果要更新文档，则该运算符无效。

参数

- `field`：要设置的字段。
- `value`：要分配给该字段的值。

示例（MongoDB 外壳）

以下示例演示了`$setOnInsert`运算符在 Amazon DocumentDB 中的用法。如果文档尚不存在，则它会创建一个新文档，但是如果正在更新该文档，则不起作用。

创建示例文档

```
db.users.insertOne({
```

```
  _id: 1,
  name: "John Doe",
  age: 30
})
```

查询示例 1-更新现有文档

```
db.users.update(
  { _id: 1 },
  {
    $set: { age: 31 },
    $setOnInsert: { createdAt: new Date() }
  },
  { upsert: true }
)
```

输出 1

```
{
  _id: 1,
  name: "John Doe",
  age: 31
}
```

输出显示文档已更新，但由于文档已经存在，因此未添加该createdAt字段。该$setOnInsert运算符仅在插入新文档时适用。

查询示例 2-插入新文档（upsert）

```
db.users.update(
  { _id: 2 },
  {
    $set: { name: "Jane Smith", age: 25 },
    $setOnInsert: { createdAt: new Date() }
  },
  { upsert: true }
)
```

输出 2

```
{
```

```
  _id: 2,
  name: "Jane Smith",
  age: 25,
  createdAt: ISODate("2025-10-31T09:57:52.459Z")
}
}
```

代码示例

要查看使用该$setOnInsert命令的代码示例，请选择要使用的语言的选项卡：

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

u $slice pdate 运算符通过限制数组的大小来修改数组。与$push运算符一起使用时，它会限制数组中元素的数量，只保留指定数量的最新或最旧的元素。

参数

- field：要修改的数组字段。
- count：要保留的最大元素数。正值保留前 N 个元素，负值保留最后 N 个元素。

示例（MongoDB 外壳）

以下示例演示如何使用$slice更新运算符$push来维护固定大小的近期分数组。

创建示例文档

```
db.students.insertOne({
  _id: 1,
```

```
  name: "Alice",
  scores: [85, 90, 78]
});
```

查询示例

```
db.students.updateOne(
  { _id: 1 },
  {
    $push: {
      scores: {
        $each: [92, 88],
        $slice: -3
      }
    }
  }
)
```

输出

```
{
  "_id" : 1,
  "name" : "Alice",
  "scores" : [ 78, 92, 88 ]
}
```

在此示例中，$slice: -3修饰符在将新值推送到数组后仅保留最后三个元素。

代码示例

要查看使用$slice更新运算符的代码示例，请选择要使用的语言的选项卡：

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

与运算符一起使用时，u $sort pdate 修饰$push符会对数组元素进行排序。它根据指定的字段值或元素本身按升序或降序排列数组元素。

参数

- field：要修改的数组字段。
- order：1用于升序或-1降序。

示例（MongoDB 外壳）

以下示例演示了使用$sort修饰符wit $push h来添加新的测验分数并按降序对它们进行排序。

创建示例文档

```
db.students.insertOne({
  _id: 1,
  name: "Bob",
  quizzes: [
    { score: 85, date: "2024-01-15" },
    { score: 92, date: "2024-02-10" }
  ]
});
```

查询示例

```
db.students.updateOne(
  { _id: 1 },
  {
    $push: {
```

```
      quizzes: {
        $each: [{ score: 78, date: "2024-03-05" }],
        $sort: { score: -1 }
      }
    }
  }
)
```

输出

```
{
  "_id" : 1,
  "name" : "Bob",
  "quizzes" : [
    { "score" : 92, "date" : "2024-02-10" },
    { "score" : 85, "date" : "2024-01-15" },
    { "score" : 78, "date" : "2024-03-05" }
  ]
}
```

代码示例

要查看使用$sort更新修饰符的代码示例，请选择要使用的语言的选项卡：

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 DocumentDB 中的$unset运算符用于从文档中删除指定字段。使用删除字段时$unset，该字段将从文档中删除，并相应地减小文档的大小。当你想从文档中删除不必要的数据时，这可能很有用。

参数

- field：要从文档中移除的字段。这可以是单个字段，也可以是嵌套字段的虚线路径。

示例（MongoDB 外壳）

以下示例演示如何使用$unset运算符从example集合中的文档中移除该Words字段。

创建示例文档

```
db.example.insert({
    "DocName": "Document 1",
    "Date": {
        "Month": 4,
        "Day": 18,
        "Year": 1987,
        "DoW": "Saturday"
    },
    "Words": 2482
})
```

查询示例

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

输出

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

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

在此示例中，$unset运算符用于从文档中移除DocName等于 “文档 1” 的Words字段。生成的文档不再包含该Words字段。

代码示例

要查看使用该$unset命令的代码示例，请选择要使用的语言的选项卡：

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 生成式人工智能

Amazon DocumentDB 的功能是让机器学习 (ML) 和生成式人工智能 (AI) 模型能够实时处理 Amazon DocumentDB 中存储的数据。客户无需再花时间管理单独的基础设施、编写代码连接其他服务以及从主数据库中复制数据。

有关人工智能以及 AWS 如何支持您的 AI 需求的更多信息，请参阅这篇 “What-is” 文章。

主题

- No-code 使用 Amazon A SageMaker I Canvas 进行机器学习
- Amazon DocumentDB 向量搜索

No-code 使用 Amazon A SageMaker I Canvas 进行机器学习

Amazon SageMaker AI Canvas 使您无需编写任何代码即可构建自己的 AI/ML 模型。您可以为回归和预测等常见用例构建 ML 模型，并且可以从 Amazon Bedrock 访问并评估基础模型 (FM)。您还可以从 Amazon A SageMaker I 访问公共 FM JumpStart 进行内容生成、文本提取和文本摘要，以支持生成式 AI 解决方案。

如何使用 SageMaker AI Canvas 构建无代码机器学习模型

亚马逊 DocumentDB 现在与亚马逊 A SageMaker I Canvas 集成，可使用存储在亚马逊 DocumentDB 中的数据实现无代码机器学习 (ML)。现在，您无需编写任何一行代码，就可以使用 Amazon DocumentDB 中存储的数据，为回归和预测需求构建 ML 模型，并使用基础模型进行内容摘要和生成。

SageMaker AI Canvas 提供了一个可视化界面，允许 Amazon DocumentDB 客户无需任何 AI/ML 专业知识或编写一行代码即可生成预测。客户现在可以从中启动 SageMaker AI Canvas 工作空间 AWS 管理控制台，导入和加入 Amazon DocumentDB 数据，用于数据准备和模型训练。Amazon DocumentDB 中的数据现在可以在 SageMaker AI Canvas 中用来构建和增强模型，以预测客户流失、检测欺诈、预测维护故障、预测业务指标和生成内容。借助 SageMaker AI Canvas 与 Quick 的原生集成，客户现在可以跨团队发布和共享 ML-driven 见解。默认情况下， SageMaker AI Canvas 中的数据摄取管道在 Amazon DocumentDB 辅助实例上运行，从而确保应用程序 SageMaker 和 AI Canvas 摄取工作负载的性能不受阻碍。

亚马逊 DocumentDB 客户可以通过导航到新的亚马逊 Docu No-Code mentDB 机器学习控制台页面并连接到新的或可用的 A SageMaker I Canvas 工作空间来开始使用 AI Canvas。 SageMaker

配置 A SageMaker I 域和用户配置文件

您可以从以 “仅限 VPC” 模式运行的 A SageMaker I 域连接到 Amazon DocumentDB 集群。通过在您的 VPC 中启动 SageMaker AI 域，您可以控制来自 SageMaker AI Studio 和 Canvas 环境的数据流。这使您可以限制互联网访问，使用标准 AWS 网络和安全功能监控和检查流量，并通过 VPC 终端节点连接到其他 AWS 资源。请参阅《亚马逊 SageMaker AI 开发者指南》中的 Amazon A SageMaker I Canvas 入门和在无法访问互联网的 VPC 中配置亚马逊 SageMaker AI Canvas，创建用于连接到亚马逊 DocumentDB 集群的 A SageMaker I 域。

为亚马逊 DocumentDB 和 SageMaker AI Canvas 配置 IAM 访问权限

已将 `AmazonDocDBConsoleFullAccess` 附加到其关联角色和身份的 Amazon DocumentDB 用户可以访问 AWS 管理控制台。在上述角色或身份中添加以下操作，即可通过 Amazon A SageMaker I Canvas 访问无代码机器学习。

```
"sagemaker:CreatePresignedDomainUrl",
"sagemaker:DescribeDomain",
"sagemaker:ListDomains",
"sagemaker:ListUserProfiles"
```

为 SageMaker AI Canvas 创建数据库用户和角色

您可以使用 Amazon DocumentDB 中的基于角色的访问控制 (RBAC)，限制用户可以对数据库执行的操作的访问权限。RBAC 的原理是向用户授予一个或多个角色。这些角色决定了用户可以对数据库资源执行的操作。

作为 Canvas 用户，请使用用户名和密码凭证连接到 Amazon DocumentDB 数据库。您可以使用 Amazon DocumentDB RBAC 功能 user/role 为对特定数据库具有读取权限的 Canvas 用户创建数据库。

例如，使用 `createUser` 操作：

```
db.createUser({
user: "canvas_user",
pwd: "<insert-password>",
roles: [{role: "read", db: "sample-database-1"}]
})
```

这将创建一个拥有 `sample-database-1` 数据库读取权限的 `canvas_user`。Canvas 分析师可以使用此凭证访问 Amazon DocumentDB 集群中的数据。请参阅 使用访问控制 Role-Based 访问数据库 了解更多信息。

可用区

无代码集成适用于同时支持亚马逊 DocumentDB 和 Amazon AI Canvas 的区域。 SageMaker 区域包括：

- us-east-1（弗吉尼亚州北部）
- us-east-2（俄亥俄州）
- us-west-2 (俄勒冈)
- ap-northeast-1 (东京)
- ap-northeast-2 (首尔)
- ap-south-1（孟买）
- ap-southeast-1 (新加坡)
- ap-southeast-2 (悉尼)
- eu-central-1 (法兰克福)
- eu-west-1 (爱尔兰)

请参阅《亚马逊 SageMaker AI 开发者指南》中的 Amazon SageMaker AI Canv as，了解最新的地区可用性。

Amazon DocumentDB 向量搜索

向量搜索是机器学习使用的一种方法，它通过使用距离或相似度指标比较向量表示来查找与给定数据点相似的数据点。两个向量在向量空间中越接近，就可以认为底层项目就越相似。这种方法有助于捕捉数据的语义意义。这种方法在推荐系统、自然语言处理和图像识别等各种应用中都很有用。

Amazon DocumentDB 的矢量搜索将 JSON-based 文档数据库的灵活性和丰富的查询功能与矢量搜索的强大功能相结合。如果要使用现有的 Amazon DocumentDB 数据或灵活的文档数据结构来构建机器学习和生成式人工智能用例，例如语义搜索体验、产品推荐、个性化、聊天机器人、欺诈检测和异常检测，Amazon DocumentDB 向量搜索就是您的理想之选。Amazon DocumentDB 5.0 基于实例的集群可以使用向量搜索。

主题

- 插入向量
- 创建向量索引
- 获取索引定义
- 查询向量
- 特征和限制
- 最佳实践

插入向量

要在 Amazon DocumentDB 数据库插入向量，可以使用现有的插入方法：

示例

以下示例在测试数据库中创建了一个包含五个文档的集合。每个文档都含有两个字段：产品名称及其相应的向量嵌入。

```
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]}
]);
```

创建向量索引

Amazon DocumentDB 同时支持分层可导航小世界 (HNSW) 索引和平面压缩式倒置文件 (IVFFLat) 索引方法。IVFFlat 索引以列表形式分割向量，然后搜索这些列表中最接近查询向量的选定子集。另一方面，HNSW 索引则将向量数据组织成多层次的图。尽管 HNSW 的构建时间慢于 IVFFlat，但它提供了更好的查询性能和查全率。与 IVFFlat 不同的是，HNSW 不涉及训练步骤，因此无需加载任何初始数据即可生成索引。对于大多数用例，建议使用 HNSW 索引类型进行向量搜索。

如果不创建向量索引，Amazon DocumentDB 会进行精确的近邻搜索，从而确保完美的查全率。但是，速度在生产场景中至关重要。我们建议使用向量索引，这样可以牺牲一些查全率来提高速度。值得注意的是，添加向量索引可能会产生不同的查询结果。

模板

您可以使用以下 `createIndex` 或 `runCommand` 模板，在向量字段上构建向量索引：

Using createIndex

在某些驱动程序中，比如 mongosh 和 Java，使用 `createIndex` 中的参数 `vectorOptions` 可能会导致错误。在这种情况下，建议使用 `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

在某些驱动程序中，比如 mongosh 和 Java，使用 `createIndex` 中的参数 `vectorOptions` 可能会导致错误。在这种情况下，建议使用 `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"
      }]
  }
);
```

参数	要求	数据类型	说明	值
name	optional	字符串	指定索引的名称。	字母数字
type	optional		指定索引类型。	支持：HNSW 或 IVFFLAT 默认：HNSW（引擎修补版本 3.0.4574 及更高版本）
dimensions	必需	integer	指定向量数据中的维度数。	最多 2,000 个维度。
similarity	必需	字符串	指定相似度计算使用的距离指标。	• **euclidean** • **cosine** • **dotProduct**
lists	IVFFlat 必需	integer	指定 IVFFlat 索引用于对向量数据进行分组的集群数。documents/1000 对于最多 100 万个文档和`sqrt(# of documents)` 超过 100 万个文档，推荐设置为 #。	最小值：1 最大值：请参阅下文 特征和限制 中各实例类型列表的表格。
m	optional	integer	指定 HNSW 索引的最大连接数	默认值：16 范围：[2, 100]
efConstruction	optional	integer	指定用于构建 HNSW 索引图的	默认值：64 范围：[4, 1000]

参数	要求	数据类型	说明	值
			动态候选列表的大小。 efConstruction 必须大于或等于 (2 * m)。	

请务必适当设置子参数的值，例如 ivffLat m 和 efConstruction HNSW 的子参数 accuracy/recall，因为这会影响搜索的构建时间和性能。lists列表值越大，查询速度越快，因为这可以减少每个列表中的向量数量，从而产生较小的区域。但是，较小的区域可能会产生更多的查全错误，从而降低准确度。对于 HNSW 来说，增加 efConstruction 和 m 的值可提高准确度，但也会增加索引构建的时间和大小。请参阅以下示例：

示例

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
    }
  }
)
```

获取索引定义

您可以使用以下 getIndexes 命令查看索引的详细信息，包括向量索引：

示例

```
db.collection.getIndexes()
```

示例输出

```
[
 {
  "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"
 }
]
```

查询向量

Amazon DocumentDB 支持两个向量搜索运算符来查询向量：

经典矢量搜索运算符

使用以下模板查询向量：

```
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]
      }
    }
  }
]);
```

参数	要求	Type	说明	值
vectorSearch	必需	operator	在 $search 命令中用于查询向量。	
vector	必需	array	表示可用于查找相似向量的查询向量。	
path	必需	字符串	定义向量字段的名称。	
k	必需	integer	指定搜索返回的最大结果数量。	

参数	要求	Type	说明	值
similarity	必需	字符串	指定相似度计算使用的距离指标。	• **euclidean** • **cosine** • **dotProduct**
probes	optional	integer	需要向量搜索检查的集群数量。值越大，查全率越好，但会牺牲速度。可以将其设置为列表数量，以进行精确的近邻搜索（此时计划程序不会使用索引）。开始微调的推荐设置为 `sqrt(# of lists)`。	默认：1
efSearch	optional	integer	指定 HNSW 索引在搜索期间使用的动态候选列表的大小。`efSearch` 的值越大，查全率越好，但会牺牲速度。	原定设置值：40 范围：[1, 1000]

务必微调 `efSearch`（HNSW）或 `probes`（IVFFlat）的值，才能实现所需的性能和准确度。请参阅以下示例操作：

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
      }
    }
  }
]);
```

示例输出

此操作的输出将类似于以下内容：

```
{ "_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 运算符（适用于亚马逊 DocumentDB 8.0 及更高版本）

使用以下模板查询向量：

```
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],
  }
}])
```

特征和限制

版本兼容性

- 亚马逊 DocumentDB 的矢量搜索仅适用于基于实例的亚马逊 DocumentDB 5.0 以上集群。

向量

- Amazon DocumentDB 最多可以为 2,000 个维度的向量编制索引。但是，如果不使用索引，可以存储多达 16,000 个维度。

索引

- 对于创建 IVFFLat 索引，列表参数的推荐设置是最多 100 万 documents/1000 个文档和超过 100 万个文档`sqrt(# of documents)`的数量。由于工作内存的限制，Amazon DocumentDB 支持的某个最大列表参数值具体取决于维度的数量。下表为 500、1000 和 2,000 个维度的向量提供了列表参数最大值，以供您参考：

实例类型	含有 500 个维度的列表	含有 1000 个维度的列表	含有 2000 个维度的列表
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

实例类型	含有 500 个维度的列表	含有 1000 个维度的列表	含有 2000 个维度的列表
r5.4xl	7,842	7,599	7,138
r5.8xl	11,220	10,974	10,498
r5.12xl	13,774	13,526	13,044
r5.16xl	15,943	15,694	15,208
r5.24xl	19,585	19,335	18,845

- 向量索引不支持 `compound`、`sparse` 或 `partial` 等其他索引选项。
- 在亚马逊 DocumentDB 5.0 中，HNSW 索引不支持并行索引构建。

向量查询

- 对于向量搜索查询，务必微调 `probes` 或 `efSearch` 等参数才能获得最佳结果。`probes` 或 `efSearch` 参数的值越大，查全率越高，速度就越低。开始微调探针参数的推荐设置为 `sqrt(# of lists)`。

最佳实践

了解在 Amazon DocumentDB 中使用向量搜索的最佳实践。随着新的最佳实践的确定，此节将不断更新。

- 使用平面压缩式倒置文件 (IVFFLat) 创建索引涉及根据相似性对数据点进行集群和组织。因此，为了使索引更加有效，建议在创建索引之前至少加载一些数据。
- 对于向量搜索查询，务必微调 `probes` 或 `efSearch` 等参数才能获得最佳结果。`probes` 或 `efSearch` 参数的值越大，查全率越高，速度就越低。开始微调 `probes` 参数的推荐设置为 `sqrt(lists)`。

资源

- 向量搜索最新消息博客文章
- 语义搜索代码示例

- Amazon DocumentDB 向量搜索代码示例

迁移到 Amazon DocumentDB

Amazon DocumentDB 是一项完全托管的与 MongoDB API 兼容的数据库服务。您可以使用本部分中详述的过程，将数据从本地或 Amazon Elastic Compute Cloud (Amazon EC2) 上运行的 MongoDB 数据库迁移到 Amazon DocumentDB。

主题

- 使用迁移到亚马逊 DocumentDB AWS Database Migration Service (DMS)：快速入门指南
- Amazon DocumentDB 迁移运行手册
- 从 Couchbase 服务器迁移

使用迁移到亚马逊 DocumentDB AWS Database Migration Service (DMS)：快速入门指南

主题

- 准备 DMS 源
- 设置 DMS
- 启用 DocumentDB 压缩
- 创建复制任务
- 监控进度
- 附加信息

准备 DMS 源

要启用 DocumentDB 变更流或启用 MongoDB Oplog 以支持 DMS 变更数据捕获（CDC），请参阅 启用变更流。

- DMS 源必须保留所有正在进行的变更，直至所有包含的集合的 DMS 完全加载完成。
- DocumentDB 变更流是基于时间的。确保您的 `change_stream_log_retention_duration` 设置足够大，足以覆盖完成完全加载所需要的时间。
- MongoDB Oplog 为固定大小。确保其大小可容纳完全加载期间的所有操作。

设置 DMS

创建 DMS 实例、源端点和目标端点，并测试每个端点。

启用 DocumentDB 压缩

通过向 DocumentDB 集群附加自定义参数组并将 default_collection_compression 参数更新为 enabled 来启用压缩。请参阅管理集合级文档压缩了解更多信息。

创建复制任务

1. 在 DMS 控制台的导航窗格中，选择迁移或复制，然后选择任务。
2. 选择创建任务。
3. 在创建任务页面的任务配置部分中：
   - 输入唯一且有意义的任务标识符（例如，“mongodb-docdb-replication”）。
   - 在源数据库端点下拉菜单中选择您之前创建的源端点。
   - 在目标数据库端点下拉菜单中选择您之前创建的目标端点。
   - 对于任务类型，选择迁移和复制。
4. 在设置部分中：
   - 对于任务日志，请选中 “打开日 CloudWatch志” 复选框。
   - 对于编辑模式（位于部分顶部），选择 JSON 编辑器并设置以下属性：
     - 将 `ParallelApplyThreads` 设置为 5（位于 `TargetMetadata` 下方）。这将在 CDC 中每秒启用大约 1000 次 insert/update /删除操作。
     - 将 `MaxFullLoadSubTasks` 设置为 16（位于 `FullLoadSettings` 下方）。根据您的实例大小酌情增加此值。
     - 对于大型集合（超过 100GB），请启用自动分区（在“表映射”下和 `parallel-load` 属性下）：
       - “type”：“partitions-auto”
       - “number-of-partitions”：16

监控进度

使用 AWS DMS 控制台或创建自定义仪表板（仪表板工具）来跟踪迁移情况。重点关注以下指标：

- FullLoadThroughputBandwidthTarget— 测量 DMS 在迁移的满负荷阶段向目标数据库传输数据时使用的网络带宽 (in KB/second)。
- CDCLatencyTarget— 测量从源数据库发生更改到将该更改应用于目标数据库之间的时间延迟（以秒为单位）。
- CDCThroughputRowsTarget— 测量 DMS 在迁移的持续复制阶段每秒向目标数据库应用的行数。

附加信息

有关 Amazon DocumentDB 和的更多信息 AWS DMS，请参阅：更多信息请参阅。

- Amazon DocumentDB 迁移运行手册
- 从 MongoDB 迁移到 Amazon DocumentDB

Amazon DocumentDB 迁移运行手册

本运行手册提供了使用 (DMS) 将 MongoDB 数据库迁移到 Amazon DocumentDB 的全面指南。 AWS Database Migration Service 旨在从初始发现到迁移后验证的整个端到端迁移过程中，为数据库管理员、云工程师和开发人员提供支持。

鉴于 MongoDB 与 Amazon DocumentDB 在实现和支持功能方面的差异，本运行手册强调了结构化和系统化的方法。本运行手册概述了重要的迁移前评测，重点介绍了兼容性注意事项，并详细说明了在最大限度减少中断的情况下确保迁移成功所需执行的关键任务。

本运行手册包含以下主题：

- 兼容性：了解 Amazon DocumentDB 中支持的 MongoDB 功能和数据类型，识别潜在的不兼容性。
- 工作负载发现— 分析现有 MongoDB 工作负载， read/write 包括模式、数据量和性能基准。
- 索引迁移：分析用于提取和转换 MongoDB 索引的策略，以在 Amazon DocumentDB 中实现最佳性能。
- 用户迁移：详细介绍将数据库用户、角色和访问控制迁移到 Amazon DocumentDB 的方法。
- 数据迁移— 介绍使用各种数据迁移方法 AWS DMS，包括满载和变更数据捕获 (CDC)。
- 监控：详细介绍使用 DMS 或原生工具进行迁移时的各种监控方法。
- 验证：提供进行迁移后数据完整性检查、功能验证和性能比较的过程。

通过遵循本运行手册中的指导，团队可以确保平稳、安全且高效地过渡到 Amazon DocumentDB，同时保留应用程序功能并最大限度地降低风险。

兼容性

主题

- 核心功能兼容性
- Amazon DocumentDB 兼容性评测工具

从 MongoDB 迁移到 Amazon DocumentDB 时，全面的初始评测和功能兼容性检查是成功迁移的关键。此过程从全面清点您的 MongoDB 功能开始，包括聚合管道运算符、查询模式、索引和数据模型。

由于亚马逊 DocumentDB 与 MongoDB 3.6、4.0、5.0 和 8.0 API 兼容，因此使用新功能的应用程序可能需要重构。 MongoDB-specific 需要评估的关键领域包括分片机制（Amazon DocumentDB 使用不同的方法）、事务实现、变更流功能以及索引类型（尤其是稀疏索引和部分索引）。

性能特征也存在差异，Amazon DocumentDB 针对企业工作负载进行了优化，性能可预测。测试应包括对两个系统运行具有代表性的工作负载，以识别可能需要优化的查询模式。

在评测阶段，监控执行计划以发现潜在的性能差距非常重要。这有助于制定清晰的迁移路线图，识别必要的应用程序变更以及建立切合实际的时间表，以便实现平稳过渡。

核心功能兼容性

全面功能支持

- CRUD 操作：全面支持所有基本的创建、读取、更新和删除操作（包括批量和查询运算符），从而提供无缝的应用程序兼容性。
- 丰富的索引功能：利用对单字段索引、复合索引、TTL 索引、部分索引、稀疏索引和 2dsphere 索引的全面支持，优化查询性能，以及利用文本索引（版本 5）进行基于文本的查找。
- Enterprise-grade 复制 — 受益于强大的自动故障转移机制，该机制具有只读副本，可实现卓越的高可用性，而不会产生操作开销。
- 高级备份解决方案 — 借助具有 Point-in-Time 恢复 (PITR) 功能的自动备份系统和按需手动快照来保护数据，让您高枕无忧。

增强 AWS-集成功能

- 简化的聚合：利用最常用的聚合阶段（`$match`、`$group`、`$sort`、`$project` 等），已针对企业工作负载优化性能。
- 事务支持：实现多文档和多集合事务，非常适合大多数业务应用程序需求。
- Real-time 数据跟踪 — 通过简单的命令启用变更流，并通过简单的参数组设置来延长更改流的保留期，以实现实时数据变更监控。
- Location-based 服务 — 实施支持`$geoNear`运算符和 2dsphere 索引的地理空间应用程序。
- 文本搜索功能：利用内置的文本搜索功能满足内容发现需求。

现代架构优势

- Cloud-native 设计 — 享受 AWS经过优化的架构，该架构可取代传统功能，例如更高效 MapReduce 的聚合管道操作。
- 增强的安全性-受益于 AWS Identity and Access Management (IAM)、 SCRAM-SHA-1、 SCRAM-SHA-256、 X.509 证书身份验证和基于密码的身份验证。
- 可预测的性能：体验专为企业工作负载而优化的稳定性能。

要全面了解 Amazon DocumentDB 的功能，请参阅 Amazon DocumentDB 中支持的 MongoDB API、操作和数据类型 和 功能差异：Amazon DocumentDB 和 MongoDB，以最大限度地发挥数据库的潜力。

Amazon DocumentDB 并不支持 MongoDB 提供的所有索引。我们提供免费的索引工具来检查兼容性。我们建议运行索引工具来评测不兼容性，并据此规划解决方法。

Amazon DocumentDB 兼容性评测工具

MongoDB 与 Amazon DocumentDB 的兼容性工具是一个可用的开源工具，它通过分析 MongoD B 日志或应用程序源代码来帮助评估 MongoDB 工作 GitHub 负载与亚马逊 DocumentDB 的兼容性。

主要特征

- 识别工作负载中的 MongoDB API 使用模式
- 在迁移之前标记潜在的兼容性问题
- 生成包含建议的详细兼容性报告
- 可用作在本地运行的独立实用程序

评测方法

Log-based 评估

- 优点：
  - 捕获实际运行时行为和查询模式
  - 识别现实世界的使用频率和性能特征
  - 检测可能在源代码中不可见的动态查询
  - 无需访问应用程序源代码
- 缺点：
  - 需要访问 MongoDB 日志并启用分析
  - 仅捕获在日志记录期间发生的操作
  - 可能会遗漏不常使用的功能或季节性工作负载

源代码分析

- 优点：
  - 全面覆盖代码库中所有潜在的 MongoDB 操作
  - 可以识别罕见执行的代码路径中的问题
  - 检测可能受 Amazon DocumentDB 差异影响的客户端逻辑
  - 无需运行应用程序即可执行评测
- 缺点：
  - 可能会标记存在但从未在生产环境中执行的代码
  - 需要访问完整的应用程序源代码
  - 对动态构造的查询进行分析的能力有限

为了获得最佳结果，我们建议尽可能使用这两种评测方法，以便在迁移之前全面了解兼容性挑战。

工作负载发现

从 MongoDB 迁移到 Amazon DocumentDB 需要对现有数据库工作负载有透彻的了解。工作负载发现是分析数据库使用模式、数据结构、查询性能和操作依赖关系的过程，旨在确保以最少的中断实现无缝

过渡。本节概述了工作负载发现所涉及的关键步骤，以促进从 MongoDB 到 Amazon DocumentDB 的有效迁移。

主题

- 评测现有的 MongoDB 部署
- 识别数据模型差异
- 查询和性能分析
- 安全与访问控制审查
- 操作和监控注意事项

评测现有的 MongoDB 部署

在迁移之前，评估当前的 MongoDB 环境至关重要，包括：

- 集群架构：确定节点数量、副本集和分片配置。从 MongoDB 迁移到 Amazon DocumentDB 时，了解您的 MongoDB 分片配置非常重要，因为 Amazon DocumentDB 不支持用户控制的分片。为分片 MongoDB 环境设计的应用程序需要进行架构变更，因为 Amazon DocumentDB 在其基于存储的架构中使用了不同的扩展方法。迁移到 Amazon DocumentDB 时，您需要调整数据分配策略，并且可能需要整合分片集合。
- 存储和数据量：测量集群的总数据大小和索引大小。配合使用 Oplog 审查工具，以了解写入模式和数据增长速度。有关调整集群大小的更多信息，请参阅 实例大小调整。
- 工作负载模式：分析读取和写入吞吐量、查询执行频率和索引效率。
- 操作依赖关系：记录依赖于 MongoDB 的所有应用程序、服务和集成。

识别数据模型差异

尽管 Amazon DocumentDB 是 MongoDB-compatible，但支持的功能还是有差异的，例如：

- 事务：Amazon DocumentDB 支持 ACID 事务，但存在一些限制。
- 架构设计：确保文档结构、嵌入式文档和参考符合 Amazon DocumentDB 的最佳实践。

查询和性能分析

了解查询行为有助于优化迁移和迁移后的性能。需要分析的关键领域包括：

- 慢查询：使用 MongoDB 的分析工具识别执行时间较长的查询。

- 查询模式：对常见的查询类型进行分类，包括 CRUD 操作和聚合。
- 索引使用情况：评测索引是否得到有效利用，或者是否需要在 Amazon DocumentDB 中进行优化。要评测索引使用情况并优化 Amazon DocumentDB 中的性能，请在关键查询中将 `$indexStats` 聚合管道阶段与 `explain()` 方法结合使用。首先运行 `db.collection.aggregate([{$indexStats{}}])` 以识别正在使用的索引。您可以通过使用 `explainPlan` 执行最频繁的查询来执行更详细的分析。
- 并发和工作负载分配：评估读取和写入比率、连接池和性能瓶颈。

安全与访问控制审查

身份验证和授权

- MongoDB RBAC 到亚马逊 DocumentDB IAM 和 RBAC — 将 MongoDB 基于角色的访问控制用户和角色映射到 (IAM) 策略 AWS Identity and Access Management 和亚马逊 DocumentDB SCRAM 身份验证用户。
- 用户迁移策略：规划将数据库用户、自定义角色和权限迁移到 Amazon DocumentDB 支持的身份验证机制。
- 权限差异：识别 MongoDB 权限中没有直接的 Amazon DocumentDB 等效权限（例如，集群管理角色）。
- 应用程序身份验证：更新 Amazon DocumentDB 密码策略的连接字符串和凭证管理。您可以使用 Secrets Manager 来存储您的凭证和轮换密码。
- 服务帐户管理-在中建立管理服务帐户凭据的流程 AWS Secrets Manager。
- 最低权限实施：审查并优化访问控制，以在新环境中实施最低权限原则。

加密

确保静态加密和传输中加密符合合规要求。

网络配置

规划虚拟私有云（VPC）设置和安全组规则。

操作和监控注意事项

为了保持系统可靠性，工作负载发现还应包括：

- 备份和恢复策略：评估现有的备份方法和 Amazon DocumentDB 的备份功能。

- AWS Backup 集成 — 利用 AWS Backup 跨 AWS 服务（包括 Amazon DocumentDB）进行集中备份管理。
- CloudWatch 指标 — 将 MongoDB 监控指标映射到亚马逊 DocumentDB 的 CPU、内存、连接和存储的指标。 CloudWatch
- 性能详情：实施 Amazon DocumentDB 性能详情，以通过详细的查询分析，可视化数据库负载并分析性能问题。
- Profiler — 将 Amazon DocumentDB 分析器配置为捕获运行缓慢的操作（与 MongoDB 的分析器类似，但使用亚马逊设置）。 DocumentDB-specific
  - 通过参数组启用并设置适当的阈值。
  - 分析探查器数据以识别优化机会
- CloudWatch 事件 — 为 Amazon DocumentDB 集群事件设置事件驱动型监控。
  - 为备份事件、维护时段和失效转移配置通知。
  - 与 Amazon SNS 集成，以提供警报和自动响应 AWS Lambda 。
- 审计日志记录：规划审计日志记录配置，以跟踪用户活动和安全相关事件。
- 增强监控-以 1 秒为间隔启用对精细 OS-level 指标的增强监控。

索引迁移

从 MongoDB 迁移到 Amazon DocumentDB 不仅需要传输数据，还需要传输索引，以保持查询性能和优化数据库操作。本节概述了在确保兼容性和效率的同时，将索引从 MongoDB 迁移到 Amazon DocumentDB 的详细分步流程。

使用 Amazon DocumentDB 索引工具

克隆索引工具

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

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

从 MongoDB 导出索引（如果从 MongoDB 迁移）

```
python3 migrationtools/documentdb_index_tool.py --dump-indexes --dir
 mongodb_index_export --uri
```

```
'mongodb://localhost:27017'
```

从 Amazon DocumentDB 导出索引（如果从 Amazon DocumentDB 迁移）

```
python3 migrationtools/documentdb_index_tool.py --dump-indexes --dir docdb_index_export
 --uri
'mongodb://user:password@mydocdb.cluster-cdtjj00yfi95.eu-west-
2.docdb.amazonaws.com:27017/?tls=true&tlsCAFile=rds-combined-ca-
bundle.pem&replicaSet=rs0&retryWrites=false'
```

导入索引

```
python3 migrationtools/documentdb_index_tool.py --restore-indexes --skip-incompatible
 --dir
mongodb_index_export --uri 'mongodb://user:password@mydocdb.cluster-cdtjj00yfi95.eu-
west-
2.docdb.amazonaws.com:27017/?tls=true&tlsCAFile=rds-combined-ca-
bundle.pem&replicaSet=rs0&retryWrites=false'
```

验证索引

```
python3 migrationtools/documentdb_index_tool.py --show-issues --dir
 mongodb_index_export
```

用户迁移

将用户从 MongoDB 迁移到 Amazon DocumentDB 对于维护访问控制、身份验证和数据库安全至关重要。本节概述了使用 Amazon DocumentDB 导出用户工具成功迁移 MongoDB 用户，同时保留其角色和权限的详细步骤。

使用 Amazon DocumentDB 导出用户工具

将用户和角色从 MongoDB 或 Amazon DocumentDB `Export Users tool`导出 JavaScript 到文件中，然后可以使用这些文件在另一个集群中重新创建它们。

先决条件

```
# 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
```

步骤 1：导出用户和角色

```
# Export users and roles to JavaScript files
python3 docdbExportUsers.py \
--users-file mongodb-users.js \
--roles-file mongodb-roles.js \
--uri "mongodb://admin:password@source-host:27017/"
```

步骤 2：编辑用户文件

```
// Example of how to update the users.js file
// Find each user creation statement and add the password
db.getSiblingDB("admin").createUser({
user: "appuser",
// Add password here
pwd: "newpassword",
roles: [
{ role: "readWrite", db: "mydb" }
]
})
```

步骤 3：将自定义角色恢复到 Amazon DocumentDB

```
# Import roles first
mongo --ssl \
--host target-host:27017 \
--sslCAFile rds-combined-ca-bundle.pem \
--username admin \
--password password \
mongodb-roles.js
```

步骤 4：将用户恢复到 Amazon DocumentDB

```
# Import users after roles are created
mongo --ssl \
```

```
--host target-host:27017 \
--sslCAFile rds-combined-ca-bundle.pem \
--username admin \
--password password \
mongodb-users.js
```

重要提示

- 出于安全考虑，不会导出密码，必须手动将密码添加到 users.js 文件中。
- 必须先导入角色再导入用户，以确保角色分配正确。
- 该工具生成可以直接使用 mongo shell 执行的 JavaScript 文件。
- 迁移期间会保留自定义角色及其权限。
- 此方法允许在导入之前查看和修改用户权限。

此方法提供了一种安全且灵活的途径将用户和角色从 MongoDB 迁移到 Amazon DocumentDB，同时允许在迁移过程中重置密码。

数据迁移

主题

- 在线迁移
- 离线迁移
- 先决条件
- 准备 Amazon DocumentDB 集群
- 执行数据转储（mongodump）
- 将转储文件传输到恢复环境
- 将数据恢复到 Amazon DocumentDB（mongorestore）

在线迁移

本节提供了从 MongoDB 到 Amazon DocumentDB 的在线迁移的详细步骤，以最大限度地减少停机时间并实现 AWS DMS 持续复制。首先，您需要将 Amazon DocumentDB 集群设置为目标，并确保将 MongoDB 实例正确配置为源，通常需要副本集模式才能捕获变更数据。接下来，您将创建 DMS 复制实例，并使用必要的连接详细信息定义源端点和目标端点。验证端点后，配置并启动迁移任务，该任务可能包括完全数据加载和/或持续复制。

配置目标（Amazon DocumentDB）

> **Note**
>
> 如果您已预置要迁移到的 Amazon DocumentDB 集群，则可以跳过此步骤。

创建自定义参数组

请参阅中的 AWS 管理控制台 或 AWS CLI 程序创建 Amazon DocumentDB 集群参数组。

创建 Amazon DocumentDB 集群

> **Note**
>
> 虽然本指南中还有其他用于创建 Amazon DocumentDB 集群的过程，但本节中的步骤专门适用于将大量数据迁移到新集群的任务。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在 Amazon DocumentDB 管理控制台上，集群下，选择创建。
4. 在创建 Amazon DocumentDB 集群页面的集群类型部分，选择Instance-based 集群（这是默认选项）。
5. 在集群配置部分中：
   - 在集群标识符中，输入唯一名称，例如 **`mydocdbcluster`**。请注意，无论如何输入，控制台都会将所有集群的名称更改为小写。
   - 对于引擎版本，选择 5.0.0。
6. 在集群存储配置部分中，按原样保留 Amazon DocumentDB 标准设置（此为默认选项）。

7. 在实例配置部分：

   - 对于数据库实例类，选择内存优化类（包括 r 类）（这是默认值）。
   - 对于实例类，根据工作负载选择实例类。例如：
     - db.r6g.large：适用于较小的工作负载
     - db.r6g.4xlarge：适用于较大的工作负载

     作为最佳实践，我们建议选择尽可能大的实例以获得最佳的完全加载吞吐量，并在迁移完成后缩减规模。
   - 对于实例数，选择 1 个实例。选择一个实例有助于成本最小化。我们建议您在完成完全加载迁移后，扩展到三个实例以实现高可用性。
8. 在身份验证部分中，输入主要用户的用户名，然后选择自行管理。输入密码，然后确认密码。
9. 在网络设置部分中，选择 VPC 和子网组，然后配置 VPC 安全组。通过更新入站规则，确保您的 Amazon DocumentDB 安全组允许来自 DMS 实例安全组的入站连接。
10. 在该Encryption-at-rest部分中，启用加密（推荐），然后选择或输入 KMS 密钥。
11. 在备份部分中，设置备份保留期（1 到 35 天）。
12. 检查您的配置，然后选择创建集群。

    部署时间通常在 10 到 15 分钟之间，

配置源

MongoDB 和 Amazon DocumentDB 都可以充当迁移源，具体取决于您的场景：

- MongoDB 作为源 — 从本地或自行管理的 MongoDB 迁移到 Amazon DocumentDB 或其他数据库服务时很常见。 AWS 需要在副本集模式下运行并具有足够大的 oplog（确保其大小可以容纳完全加载期间的所有操作），以支持迁移期间的变更数据捕获。
- Amazon DocumentDB 作为源：通常用于跨区域复制、版本升级或迁移到 MongoDB Atlas 等其他数据库服务。需要 启用变更流，在集群参数组中设置 `change_stream_log_retention_duration` 参数以捕获迁移期间的持续变更。确保您的 `change_stream_log_retention_duration` 设置足够大，足以覆盖完成完全加载所需要的时间。

在开始迁移之前，请将源配置为允许 AWS DMS 访问。

创建具有适当权限的 MongoDB 用户：

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

配置网络和身份验证。

在为 MongoDB 到 DMS 的迁移配置网络连接时：

EC2-hosted MongoDB 源

- 修改 EC2 安全组以允许来自 DMS 复制实例安全组的入站流量。
- 为 TCP 端口 27017（或您的自定义 MongoDB 端口）添加规则。
- 使用 DMS 复制实例的安全组 ID 作为源，以实现精确访问控制。
- 确保 EC2 实例的子网可路由至 DMS 复制实例的子网。

On-premises MongoDB 源

- 配置您的防火墙以允许来自 DMS 复制实例的公有 IP 地址的入站连接。
- 如果使用 Direct Connect 或 VPN，请确保您的网络和包含 DMS 实例的 VPC 之间路由正确。
- 使用 telnet 或 nc 命令测试从 DMS 子网到 MongoDB 服务器的连接。

MongoDB Atlas 源

- 将 DMS 复制实例 IP 地址添加到 MongoDB Atlas IP 允许列表。
- 如果 Atlas 正在运行，则在 AWS VPC 和 MongoDB Atlas VPC 之间配置 VPC 对等互连。 AWS
- 如果在其他云提供商上运行，请设置 AWS PrivateLink 私有连接（企业级）。
- 创建具有适当 read/write 权限的专用用户。
- 使用 MongoDB Atlas 连接字符串，将“SSL 模式”设置为“verify-full”。
- 确保迁移期间有足够的 oplog 大小。

Amazon DocumentDB 源

配置您的源 Amazon DocumentDB 安全组，以允许来自 DMS 复制实例安全组的入站流量。

创建 DMS 复制实例

我们建议使用 DMS Buddy 来预置您的 DMS 基础设施，因为这样可以创建具有最佳 DMS 设置和实例大小的最佳迁移基础设施。

如果您更喜欢手动配置，请执行以下步骤：

1. 打开 AWS DMS 控制台，选择创建复制实例。
2. 输入复制实例的详细信息：
   - 实例名称：选择唯一名称。
   - 实例类：根据工作负载进行选择。示例：dms.r5.large（小型工作负载），dms.r5.4xlarge（大型工作负载）。
   - 引擎版本：3.5.4
   - 分配的存储空间：默认值为 100GB（如果需要可增加）。这取决于文档大小 updates/second 和满载持续时间。
   - Multi-AZ 部署：如果需要，启用以获得高可用性。
   - 选择与 Amazon DocumentDB 相同的 VPC。
   - 确保安全组允许来自源和 Amazon DocumentDB 的入站流量。
3. 选择创建复制实例，然后等待状态变为可用。

创建 DMS 端点

创建源端点

对于 MongoDB 源

1. 在 DMS 控制台的导航窗格中，选择迁移或复制，然后选择端点。
2. 选择创建端点。
3. 在创建端点页面上，选择源端点。
4. 在端点配置部分中：
   - 输入唯一且有意义的端点标识符（例如，“mongodb-source”）。
   - 选择 MongoDB 作为源引擎。
   - 对于端点数据库的访问权限，请选择手动提供访问信息。
   - 在服务器名称中，输入您的*MongoDB server DNS name/IP address*。

- 对于端口，输入 27017（默认 MongoDB 端口）。
- 对于身份验证模式，请为您的应用程序选择适当的模式 (password/SSL)（默认为密钥管理器）。
- 如果身份验证模式为密码，请提供：
  - 用户名和密码：输入 MongoDB 凭证。
  - 数据库名称：您的源数据库名称。
  - 身份验证机制： SCRAM-SHA-1 （默认）或适当的机制

5. 对于元数据模式，保留文档的默认设置。
6. 其他连接属性：

- authSource=admin（如果身份验证数据库不同）
- replicaSet=<your-replica-set-name>（对于 CDC 而言必需）

对于 Amazon DocumentDB 源

1. 在 DMS 控制台的导航窗格中，选择迁移或复制，然后选择端点。
2. 选择创建端点。
3. 在创建端点页面上，选择源端点。
4. 在端点配置部分中：

- 输入唯一且有意义的端点标识符（例如，“docdb-source”）。
- 选择 Amazon DocumentDB 作为源引擎。
- 对于端点数据库的访问权限，请选择手动提供访问信息。
- 在服务器名称中，输入您的*source Amazon DocumentDB cluster endpoint*。
- 对于端口，输入 27017（默认的 Amazon DocumentDB 端口）。
- 对于 SSL 模式，选择 verify-full（推荐用于 Amazon DocumentDB）。
- 对于 CA 证书，选择 Amazon RDS 根 CA 证书。
- 对于身份验证模式，请为您的应用程序选择适当的模式 (password/SSL)（默认为密钥管理器）。
- 如果身份验证模式为密码，请提供：
  - 用户名和密码：输入 Amazon DocumentDB 凭证。
  - 数据库名称：您的源数据库名称。
  - 身份验证机制： SCRAM-SHA-1 （默认）或适当的机制

5. 对于元数据模式，保留文档的默认设置。

创建目标端点（Amazon DocumentDB）

1. 在 DMS 控制台的导航窗格中，选择迁移或复制，然后选择端点。
2. 选择创建端点。
3. 在创建端点页面中，选择目标端点。
4. 在端点配置部分中：
   - 输入唯一且有意义的端点标识符（例如，“docdb-target”）。
   - 选择 Amazon DocumentDB 作为目标引擎。
   - 对于访问端点数据库，选择要用于对数据库的访问进行身份验证的方法：
     - 如果您选择 AWS Secrets Manager，请在密钥字段中选择其中存储有 Amazon DocumentDB 凭证的密钥。
     - 如果您选择手动提供访问信息：
       - 在服务器名称中，输入您的*target Amazon DocumentDB cluster endpoint*。
       - 对于端口，输入 27017（默认的 Amazon DocumentDB 端口）。
       - 对于 SSL 模式，选择 verify-full（推荐用于 Amazon DocumentDB）。
       - 对于 CA 证书，下载并指定用于 SSL 验证的 CA 证书捆绑包。
       - 对于身份验证模式，请为您的应用程序选择适当的模式 (password/SSL)（默认为密钥管理器）。
       - 如果身份验证模式为密码，请提供：
         - 用户名和密码：输入 Amazon DocumentDB 凭证。
         - 数据库名称：您的源数据库名称。
         - 身份验证机制： SCRAM-SHA-1 （默认）或适当的机制
5. 对于元数据模式，保留文档的默认设置。

创建复制任务

1. 在 DMS 控制台的导航窗格中，选择迁移或复制，然后选择任务。
2. 选择创建任务。
3. 在创建任务页面的任务配置部分中：

- 输入唯一且有意义的任务标识符（例如，“mongodb-docdb-replication”）。
- 在源数据库端点下拉菜单中选择您之前创建的源端点。
- 在目标数据库端点下拉菜单中选择您之前创建的目标端点。
- 对于任务类型，选择迁移和复制。

4. 在设置部分中：

- 对于目标表准备模式，保留默认设置。
- 对于在完全加载完成后停止任务，保留默认设置。
- 对于 LOB 列设置，按原样保留受限 LOB 模式设置。
- 对于数据验证，保留默认设置关闭。
- 对于任务日志，请选中 “打开日 CloudWatch志” 复选框。
- 对Batch-optimized 于 “应用”，保留默认设置为未选中（关闭）。

5. 回到任务设置部分的顶部，在编辑模式中选择 JSON 编辑器并设置以下属性：

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

6. 在表映射部分中，添加新选择规则：

- 对于架构名称，添加要迁移的源数据库。使用 % 指定多个数据库。
- 对于架构表名称，添加要迁移的源集合。使用 % 指定多个集合。
- 对于操作，保留默认设置包含

7. 对于大型集合（超过 100GB），添加表设置规则：

- 对于架构名称，添加要迁移的源数据库。使用 % 指定多个数据库。
- 对于架构表名称，添加要迁移的源集合。使用 % 指定多个集合。
- 对于分区数，输入 16（应小于 `MaxFullLoadSubTask`）。

8. 在迁移前评测部分中，确保将其关闭。

离线迁移

本节概述了使用原生 MongoDB 工具 `mongodump` 和 `mongorestore` 执行从自行管理的 MongoDB 实例到 Amazon DocumentDB 的离线迁移流程。

先决条件

源 MongoDB 要求

- 使用适当的权限访问源 MongoDB 实例。
- 如果需要，请安装 `mongodump`（在 MongoDB 安装期间进行安装）。
- 确保有足够的磁盘空间来存放转储文件。

目标 Amazon DocumentDB 要求

- 确保您已预置 Amazon DocumentDB 集群。
- 确保在与 Amazon DocumentDB 相同的 VPC 中存在 EC2 实例，以方便迁移。
- 您的源环境与 Amazon DocumentDB 之间必须有可用的网络连接。
- `mongorestore` 必须安装在迁移 EC2 实例上。
- 必须配置适当的 IAM 权限才能访问 Amazon DocumentDB，

一般要求

- AWS CLI 必须配置（如果使用 AWS 服务进行中间存储）
- 必须有足够的带宽可用于数据传输。
- 应批准停机时间（如果要进行实时迁移，请考虑其他方法）

准备 Amazon DocumentDB 集群

在 AWS中创建 Amazon DocumentDB 集群：

- 根据您的工作负载选择合适的实例大小。
- 配置 VPC、子网和安全组。
- 通过参数组启用必要的参数。

执行数据转储（mongodump）

选择以下选项之一来创建转储文件：

- 选项 1：基本

```
mongodump --
uri="mongodb://<source_user>:<source_password>@<source_host>:<source_port>/
<database>" --
out=/path/to/dump
```

- 选项 2：更优的控制和性能

```
mongodump \
--uri="mongodb://<source_user>:<source_password>@<sourcehost>:<source_port>" \
--out=/path/to/dump \
--gzip \# Compress output
--numParallelCollections=4 \# Parallel collections dump
--ssl \# If using SSL
--authenticationDatabase=admin \ # If auth is required
--readPreference=secondaryPreferred # If replica set
```

- 选项 3：大型数据库

```
mongodump \
--host=<source_host> \
--port=<source_port> \
--username=<source_user> \
--password=<source_password> \
--db=<specific_db> \# Only dump specific DB
--collection=<specific_collection> \ # Only dump specific collection
--query='{ "date": { "$gt": "2020-01-01" } }' \ # Filter documents
--archive=/path/to/archive.gz \# Single archive output
--gzip \
--ssl
```

将转储文件传输到恢复环境

根据您的转储大小选择适当的方法：

- 小：直接复制到您的迁移机器（您之前创建的 EC2 实例）：

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

- 中：使用 Amazon S3 作为中间存储：

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

- 大型 — 对于非常大的数据库，可以考虑 AWS DataSync 物理传输。

将数据恢复到 Amazon DocumentDB（mongorestore）

在开始恢复过程之前，请在 Amazon DocumentDB 中创建索引。您可以使用 Amazon DocumentDB 索引工具导出和导入索引。

选择以下选项之一来恢复数据：

- 选项 1：基本恢复

```
mongorestore --uri="mongodb://<docdb_user>:<docdb_password>@<docdb_endpoint>:27017"
/path/to/dump
```

- 选项 2：更优的控制和性能

```
mongorestore \
--uri="mongodb://<docdb_user>:<docdb_password>@<docdb_endpoint>:27017" \
--ssl \
--sslCAFile=/path/to/rds-combined-ca-bundle.pem \ # DocumentDB CA cert
--gzip \# If dumped with gzip
--numParallelCollections=4 \# Parallel restoration
--numInsertionWorkersPerCollection=4 \# Parallel documents insertion
--noIndexRestore \# skip indexes as they are pre-created
/path/to/dump
```

- 选项 3：大型数据库或特定控件

```
mongorestore \
--host=<docdb_endpoint> \
--port=27017 \
--username=<docdb_user> \
--password=<docdb_password> \
--ssl \
--sslCAFile=/path/to/rds-combined-ca-bundle.pem \
```

```
--archive=/path/to/archive.gz \# If using archive format
--gzip \
--nsInclude="db1.*" \# Only restore specific namespaces
--nsExclude="db1.sensitive_data" \ # Exclude specific collections if needed
--noIndexRestore \# skip indexes as they are pre-created
--writeConcern="{w: 'majority'}" # Ensure write durability
```

监控

本节提供了详细的监控流程，用于跟踪正在进行的迁移的进度、性能和运行状况：

从 MongoDB 迁移到 Amazon DocumentDB

或者

从 Amazon DocumentDB 迁移到 Amazon DocumentDB

无论采用何种迁移方法（AWS DMS、或其他工具） mongodump/mongorestore，监控步骤都适用。

AWS DMS 迁移监控（如果适用）

监控以下关键 CloudWatch 指标：

完全加载阶段指标

- FullLoadThroughputBandwidthTarget— 满负荷期间的网络带宽 (KB/second)
- FullLoadThroughputRowsTarget— 每秒 rows/documents 加载的次数
- FullLoadThroughputTablesTarget— 每分钟 tables/collections 完成的次数
- FullLoadProgressPercent— 满载完成的百分比
- TablesLoaded— tables/collections 成功加载的数量
- TablesLoading— tables/collections 当前加载的数量
- TablesQueued— tables/collections 等待加载的次数
- TablesErrored— 加载 tables/collections 失败的数量

CDC 阶段指标

- CDCLatencyTarget— 源更改和目标应用程序之间的时间延迟（秒）
- CDCLatencySource— 更改源与 DMS 读取源之间的时间延迟（秒）

- CDCThroughputRowsTarget— 正在进行的复制期间应用的每秒行数
- CDCThroughputBandwidthTarget— CDC 期间的网络带宽 (KB/second)
- CDCIncomingChanges— 从源收到的变更事件数
- CDCChangesMemoryTarget— 用于在目标端存储更改的内存 (MB)

资源指标

- CPUUtilization：复制实例的 CPU 使用率
- FreeableMemory— 复制实例上的可用内存
- FreeStorageSpace— 复制实例上的可用存储空间
- NetworkTransmitThroughput— 复制实例的网络吞吐量
- NetworkReceiveThroughput— 复制实例的网络吞吐量

错误指标

- ErrorsCount— 迁移期间的错误总数
- TableErrorsCount— 特定于表格的错误数量
- RecordsErrorsCount— 特定于记录的错误数量

为诸如迁移性能下降之类的关键指标创建 CloudWatch 警报`CPUUtilization`，`CDCLatencyTarget`并在迁移性能下降时接收通知。

DMS 日志（日CloudWatch 志）

1. 前往 Amazon CloudWatch 日志控制台。
2. 在您的日志组中进行查找和选择。看起来类似于“dms-tasks –”。
3. 查找可能包含错误信息的日志流：
    - 名称中带有“error”的流
    - 带有任务 ID 或端点名称的流
    - 迁移期间的最新日志流
4. 在这些流中，搜索关键词，例如：
    - “error”

- “exception”
- “failed”
- “warning”

DMS 任务状态（使用 AWS CLI)

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

预期的状态流：

正在创建 → 就绪 → 正在运行 → 正在停止 → 已停止（或失败）

使用 **docdb-Dashboarder** 进行监控

该docdb-dashboarder工具通过自动生成包含基本性能指标的 CloudWatch 控制面板，为 Amazon DocumentDB 集群提供全面监控。这些控制面板显示关键的集群级别指标（副本延迟、操作计数器）、实例级别指标（CPU、内存、连接）以及存储指标（卷使用率、备份存储）。对于迁移场景，该工具提供专用控制面板，可通过 CDC 复制延迟和操作率等指标来跟踪迁移进度。仪表板可以同时监控多个集群，并包括对 NVMe-backed 实例的支持。通过可视化这些指标，团队可以主动识别性能瓶颈，优化资源分配，并确保其 Amazon DocumentDB 部署顺利运行。该工具无需手动创建控制面板，同时在所有环境中提供一致的监控。有关设置说明和高级配置选项，请参阅 Amazon DocumentDB 仪表板工具存储库。 GitHub

验证

主题

- 验证核对清单
- 架构和索引验证
- 数据采样和字段级别验证
- 使用 DataDiffer 工具进行验证

本节提供详细的验证流程，以确保以下迁移后的数据一致性、完整性和应用程序兼容性：

从 MongoDB 迁移到 Amazon DocumentDB

或者

从 Amazon DocumentDB 迁移到 Amazon DocumentDB

无论采用何种迁移方法（AWS DMS、或其他工具） mongodump/mongorestore，验证步骤都适用。

验证核对清单

验证每个集合中的文档数量在源与目标之间是否匹配：

MongoDB 源

```
mongo --host <source_host> --port <port> --username <user> -- password <password> --
eval
"db.<collection>.count()"
```

Amazon DocumentDB 目标

```
mongo --host <target_host> --port <port> --username <user> -- password <password> --
eval
"db.<collection>.count()"
```

架构和索引验证

请确保：

- 所有集合都存在于目标中。
- 索引已正确复制。
- 架构定义（如果强制执行）相同。

检查集合（源与目标）

```
mongo --host <source_host> --eval "show collections"
mongo --host <target_host> --ssl --eval "show collections"
```

检查索引（源与目标）

```
mongo --host <source_host> --eval" db.<collection>.getIndexes()"
mongo --host <target_host> --ssl –eval" db.<collection>.getIndexes()"
```

比较集合列表，确保没有缺失或多余的集合。

通过检查索引名称、键定义、唯一约束和 TTL 索引（如果有）来验证索引。

检查架构验证规则（如果在 MongoDB 中使用架构验证）

```
mongo --host <source_host> --eval" db.getCollectionInfos({name: '<collection>'})
[0].options.validator"
   mongo --host <target_host> --ssl -eval" db.getCollectionInfos({name:
 '<collection>'})[0].options.validator"
```

数据采样和字段级别验证

您可以对文档进行随机采样，并比较源与目标之间的字段。

手动采样

获取五个随机文档（源）：

```
mongo --host <source_host> --eval "db.<collection>.aggregate([{ \$sample: { size:
 5 } }])"
```

获取相同的文档 ID（目标）：

```
mongo --host <target_host> --ssl -eval "db.<collection>.find({ _id: { \$in:
 [<list_of_ids>] } })"
```

自动采样

```
import pymongo
# Connect to source and target
source_client = pymongo.MongoClient("<source_uri>")
target_client = pymongo.MongoClient("<target_uri>", ssl=True)
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']}")
```

使用 DataDiffer 工具进行验证

该DataDiffer 工具提供了一种在源数据库和目标数据库之间比较数据的可靠方法。

先决条件

在安装该 DataDiffer 工具之前，必须满足以下先决条件：

- Python 3.7+
- PyMongo 图书馆
- 到源 MongoDB 集群和目标 Amazon DocumentDB 集群的网络连接

设置和安装

克隆存储库并导航到该 DataDiffer 目录

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

安装所需的依赖项

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

运行数据验证

创建包含连接详细信息的配置文件（例如 config.json）

```
{
"source": {
"uri": "mongodb://username:password@source-mongodb-
host:27017/?replicaSet=rs0",
"db": "your_database",
"collection": "your_collection"
},
"target": {
"uri": "mongodb://username:password@target-docdb-
cluster.region.docdb.amazonaws.com:27017/?tls=true&tlsCAFile=global-
bundle.pem&replicaSet=rs0",
"db": "your_database",
"collection": "your_collection"
},
"options": {
```

```
"batch_size": 1000,
"threads": 4,
"sample_size": 0,
"verbose": true
}
}
```

运行该 DataDiffer 工具

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

对于大型集合，使用采样来验证数据子集

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

要验证多个集合，请创建单独的配置文件或使用批处理模式

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

解析结果

工具将输出：

- 源和目标中的文档总数量
- 匹配文档的数量
- 缺失文档的数量
- 存在差异的文档的数量
- 详细的差异报告（如果有）

最佳实践

以下是使用该 DataDiffer 工具时的最佳做法：

- 分阶段运行：首先验证文档数量，然后对关键文档进行采样，最后根据需要进行全面比较。
- 检查架构差异：与 MongoDB 相比，Amazon DocumentDB 有一些限制。该工具将突出显示不兼容的数据类型或结构。
- 在静默期间进行验证：在写入操作最少时运行验证以确保一致性。
- 监控资源使用情况：比较过程可能占用大量资源。相应地调整批处理大小和线程数。

- 验证索引：数据验证后，确保已在目标 Amazon DocumentDB 集群上创建所有必需的索引。
- 文档验证结果：记录每个集合的验证结果，将其作为迁移文档的一部分。

从 Couchbase 服务器迁移

主题

- 简介
- 与亚马逊 DocumentDB 的比较
- Discovery
- 规划
- 迁移
- 验证

简介

本指南介绍了从 Couchbase Server 迁移到 Amazon DocumentDB 时需要考虑的要点。它解释了迁移的发现、规划、执行和验证阶段的注意事项。它还说明了如何执行离线和在线迁移。

与亚马逊 DocumentDB 的比较

	Couchbase 服务器	Amazon DocumentDB
数据组织	在 7.0 及更高版本中，数据按存储桶、范围和集合进行组织。在早期版本中，数据被组织到存储桶中。	数据被组织成数据库和集合。
兼容性	每项服务都有单独的 API（例如数据、索引、搜索等）。二级查找使用 SQL++（以前称为 N1QL）；这是一种基于 ANSI-standard SQL 的查询语言，因此许多开发人员都很熟悉。	亚马逊 DocumentDB 与 MongoDB API 兼容。
架构	存储空间已连接到每个集群实例。您不能独立于存储来扩展计算。	Amazon DocumentDB 专为云而设计，可避免传统数据库架构的

	Couchbase 服务器	Amazon DocumentDB
		限制。计算层和存储层在 Amazon DocumentDB 中是分开的，计算层可以独立于存储进行扩展。
按需添加读取容量	可以通过添加实例来扩展集群。由于存储连接到运行服务的实例，因此横向扩展所需的时间取决于需要移动到新实例或重新平衡的数据量。	您可以通过在集群中创建多达 15 个 Amazon DocumentDB 副本来实现对您的 Amazon DocumentDB 集群的读取扩展。对存储层没有影响。
从节点故障中快速恢复	集群具有自动故障转移功能，但是使集群恢复到最大容量所需的时间取决于需要移动到新实例的数据量。	无论集群中的数据量如何，Amazon DocumentDB 通常可以在 30 秒内对主集群进行故障转移，并在 8-10 分钟内将集群恢复到最大容量。
随着数据的增长扩展存储	对于自行管理的集群，存储空间和 iOS 不会自动扩展。	亚马逊 DocumentDB 存储空间和 iOS 会自动扩展。
在不影响性能的情况下备份数据	备份由备份服务执行，默认情况下不启用。由于存储和计算不是分开的，因此可能会对性能产生影响。	默认情况下，Amazon DocumentDB 备份处于启用状态，无法关闭。备份由存储层处理，因此它们对计算层的影响为零。Amazon DocumentDB 支持从集群快照还原和还原到某个时间点。
数据持久性	一个集群中最多可以有 3 个副本数据副本，总共有 4 个副本。运行数据服务的每个实例都将有活动数据副本和 1、2 或 3 个副本副本。	无论有多少计算实例，Amazon DocumentDB 都会维护 6 个数据副本，写入法定人数为 4，并保持不变。存储层保存 4 个数据副本后，客户端会收到确认信息。

	Couchbase 服务器	Amazon DocumentDB
一致性	支持 K/V 操作的即时一致性。Couchbase SDK 会将 K/V 请求路由到包含数据活动副本的特定实例，因此在确认更新后，可以保证客户端读取该更新。将更新复制到其他服务（索引、搜索、分析、事件）最终是一致的。	Amazon DocumentDB 副本最终是一致的。如果需要立即读取一致性，则客户端可以从主实例读取。
复制	Cross-Data Center Replication (XDCR) 在许多:许多拓扑中提供经过筛选的、passive/active主动-主动的数据复制。	Amazon DocumentDB 全球集群在 1：多（最多 10 个）拓扑中提供主动-被动复制。

Discovery

迁移到 Amazon DocumentDB 需要对现有数据库工作负载有透彻的了解。工作负载发现是分析您的 Couchbase 集群配置和操作特征（数据集、索引和工作负载）的过程，以帮助确保在最小中断的情况下实现无缝过渡。

集群配置

Couchbase 使用以服务为中心的架构，其中每项功能都对应一项服务。对您的 Couchbase 集群执行以下命令以确定正在使用哪些服务（请参阅获取节点信息）：

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

示例输出：

```
[
  "backup",
  "cbas",
  "eventing",
  "fts",
  "index",
```

```
  "kv",
  "n1ql"
]
```

Couchbase 服务包括以下内容：

数据服务 (kv)

数据服务提供对内存和磁盘上数据的 read/write 访问。

亚马逊 DocumentDB 支持通过 MongoDB API 对 JSON 数据进行 K/V 操作。

查询服务 (n1ql)

查询服务支持通过 SQL++ 查询 JSON 数据。

亚马逊 DocumentDB 支持通过 MongoDB API 查询 JSON 数据。

索引服务（索引）

索引服务创建和维护数据索引，从而实现更快的查询。

亚马逊 DocumentDB 支持默认主索引和通过 MongoDB API 在 JSON 数据上创建二级索引。

搜索服务 (fts)

搜索服务支持创建用于全文搜索的索引。

Amazon DocumentDB 的原生全文搜索功能允许您通过 MongoDB API 使用特殊用途的文本索引对大型文本数据集执行文本搜索。对于高级搜索用例，Amazon DocumentDB Zero-etl 与亚马逊 OpenSearch 服务的集成提供了对亚马逊文档数据库数据的高级搜索功能，例如模糊搜索、跨馆藏搜索和多语言搜索。

分析服务 (cba)

分析服务支持近乎实时地分析 JSON 数据。

亚马逊 DocumentDB 支持通过 MongoDB API 对 JSON 数据进行临时查询。您还可以使用在亚马逊 EMR 上运行的 Apache Spark 对亚马逊文档数据库中的 JSON 数据进行复杂查询。

活动服务（活动）

事件服务执行用户定义的业务逻辑以响应数据更改。

Amazon DocumentDB 通过在您的 Amazon DocumentDB 集群中每次数据发生变化时调用 AWS Lambda 函数来自动执行事件驱动的工作负载。

备份服务（备份）

备份服务计划完整和增量数据备份，并合并以前的数据备份。

Amazon DocumentDB 会持续将您的数据备份到 Amazon S3，保留期为 1-35 天，这样您就可以快速恢复到备份保留期内的任何时间。作为持续备份过程的一部分，Amazon DocumentDB 还会自动拍摄数据快照。您还可以使用管理 Amazon DocumentDB 的备份和恢复。 AWS Backup。

操作特征

使用适用于 Couchbase 的发现工具获取有关您的数据集、索引和工作负载的以下信息。这些信息将帮助您调整亚马逊文档数据库集群的规模。

数据集

该工具检索以下存储桶、范围和集合信息：

1. 存储桶名称
2. 存储桶类型
3. 作用域名称
4. 集合名称
5. 总大小（字节）
6. 商品总数
7. 项目大小（字节）

索引

该工具检索以下索引统计数据以及所有存储桶的所有索引定义。请注意，不包括主索引，因为 Amazon DocumentDB 会自动为每个集合创建主索引。

1. 存储桶名称
2. 作用域名称
3. 集合名称
4. 索引名

5. 索引大小（字节）

工作负载

该工具检索 K/V 和 N1QL 查询指标。 K/V 指标值在存储桶级别收集，SQL++ 指标在集群级别收集。

工具命令行选项如下所示：

```
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>
```

以下是一个示例命令：

```
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 指标值将基于过去一周每 10 分钟采样一次（请参阅 HTTP 方法和 URI）。SQL++ 指标值将基于过去 60 秒内每 1 秒的样本（参见常规标签）。该命令的输出将位于以下文件中：

collection-stats.csv — 存储桶、范围和集合信息

```
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 — 索引名称和大小

```
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 — 获取、设置和删除所有存储桶的指标

```
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++ 为集群选择、删除和插入指标

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

<bucket-name>ind@@ exes-.txt — 存储桶中所有索引的索引定义。请注意，不包括主索引，因为 Amazon DocumentDB 会自动为每个集合创建主索引。

```
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`)
...
```

规划

在规划阶段，您将确定 Amazon DocumentDB 集群要求以及 Couchbase 存储桶、范围和馆藏与亚马逊文档数据库和馆藏的映射。

亚马逊 DocumentDB 集群要求

使用在发现阶段收集的数据来调整您的 Amazon DocumentDB 集群的大小。有关调整您的 Amazon DocumentDB 集群规模的更多信息，请参阅实例大小调整。

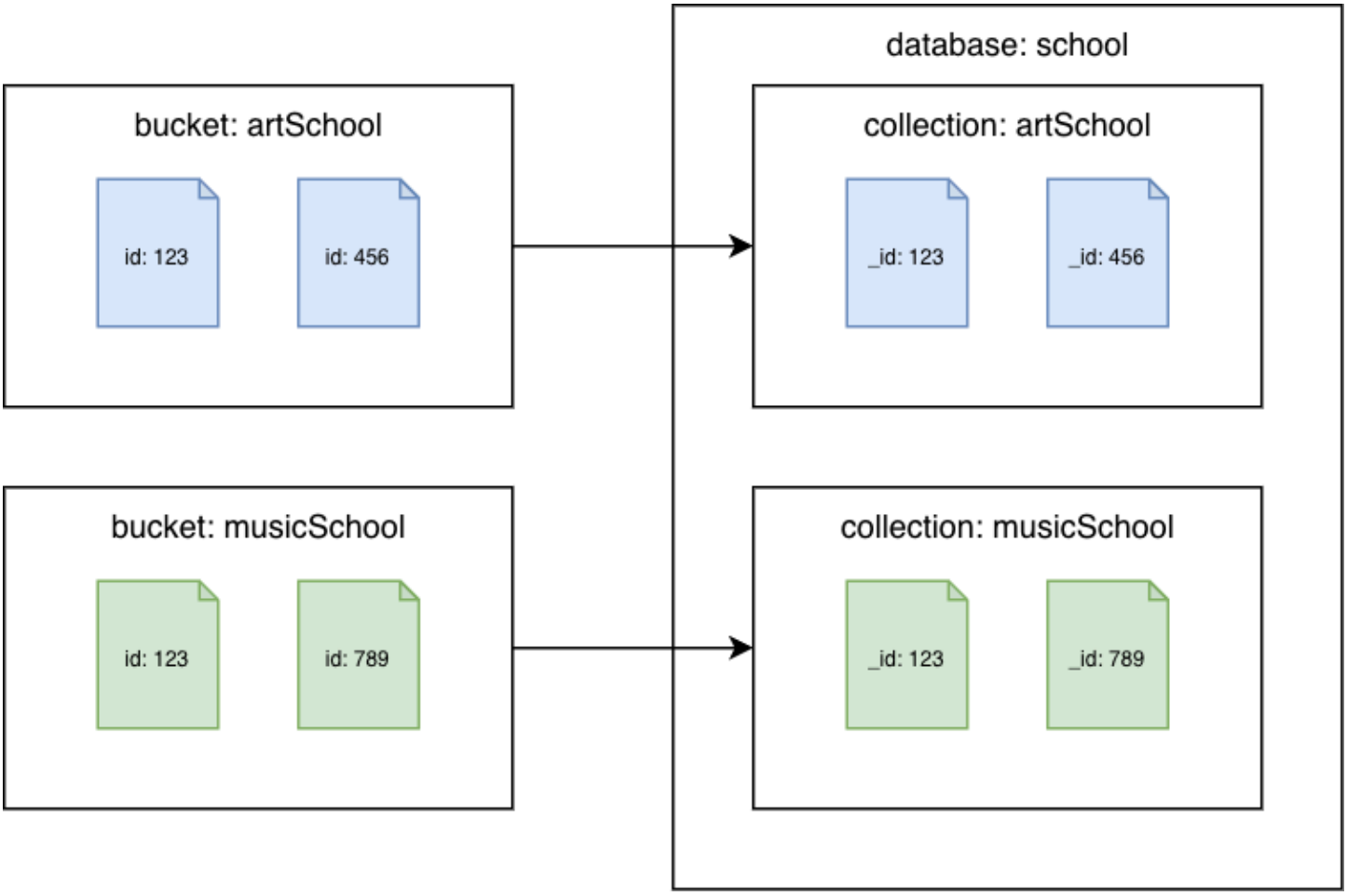
将存储桶、作用域和集合映射到数据库和集合

确定将存在于您的 Amazon DocumentDB 集群中的数据库和馆藏。根据您的 Couchbase 集群中数据的组织方式，考虑以下选项。这些不是唯一的选择，但它们提供了可供您考虑的起点。

Couchbase Server 6.x 或更早版本

Couchbase 存储到亚马逊 DocumentDB 馆藏中

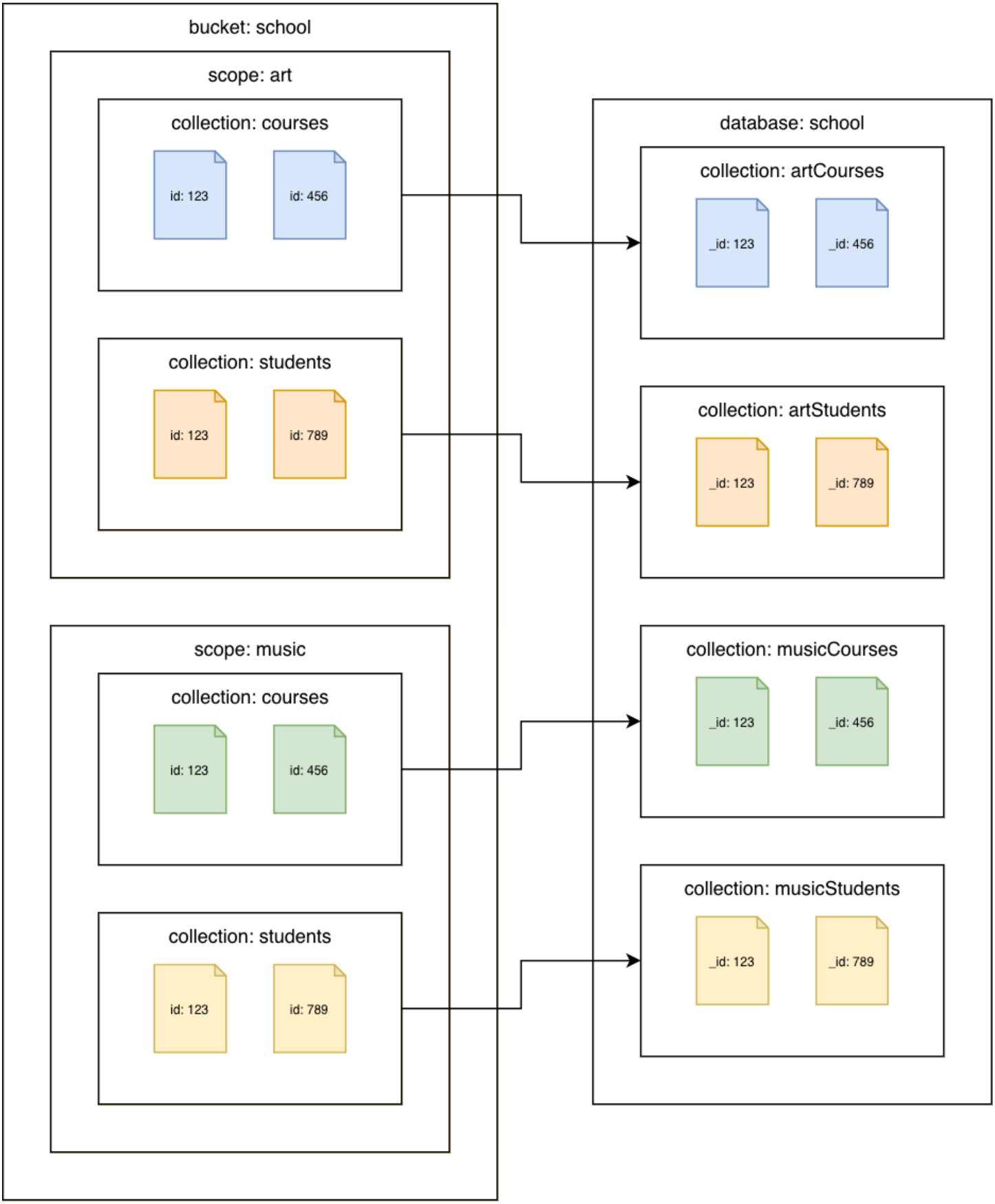
将每个存储桶迁移到不同的亚马逊文档数据库集合。在这种情况下，Couchbase 文档`id`值将用作亚马逊 `_id` DocumentDB 的值。



Couchbase Server 7.0 或更高版本

Couchbase 馆藏到亚马逊 DocumentDB 馆藏

将每个馆藏迁移到不同的亚马逊 DocumentDB 馆藏。在这种情况下，Couchbase 文档`id`值将用作亚马逊 `_id` DocumentDB 的值。


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

迁移

索引迁移

迁移到 Amazon DocumentDB 不仅需要传输数据，还需要传输索引，以保持查询性能和优化数据库操作。本节概述了在确保兼容性和效率的同时将索引迁移到 Amazon DocumentDB 的详细分步过程。

使用 Amazon Q 将 SQL++ `CREATE INDEX` 语句转换为亚马逊 DocumentDB 命令`createIndex()`。

1. 上传索引——由 Couchbase 的发现工具创建的.txt <bucket name>文件。
2. 输入以下提示：

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

Amazon Q 将生成等效的亚马逊 DocumentDB 命令`createIndex()`。请注意，您可能需要根据将 Couchbase 存储桶、范围和集合映射到 Amazon DocumentDB 馆藏的方式来更新馆藏名称。

例如：

indexes-beer-sample.txt

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

Amazon Q 输出示例（摘录）：

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

db.beerSample.createIndex(
  { "code": 1 },
  {
    "name": "code",
    "background": true,
    "partialFilterExpression": { "type": "brewery" }
```

```
  }
)
```

有关 Amazon Q 无法转换的任何索引，请参阅管理 Amazon DocumentDB 索引、索引和索引属性了解更多信息。

重构代码以使用 MongoDB API

客户使用 Couchbase 软件开发工具包连接到 Couchbase 服务器。亚马逊 DocumentDB 客户端使用 MongoDB 驱动程序连接到亚马逊 DocumentDB。MongoDB 驱动程序也支持 Couchbase 软件开发工具包支持的所有语言。有关适用于您的语言的驱动程序的更多信息，请参阅 MongoDB 驱动程序。

由于 Couchbase Server 和 Amazon DocumentDB 的 API 不同，因此您需要重构代码才能使用相应的 MongoDB API。你可以使用 Amazon Q 将 K/V API 调用和 SQL++ 查询转换为等效的 MongoDB API：

1. 上传源代码文件。
2. 输入以下提示：

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

使用 Hello Couchbase Python 代码示例，Amazon Q 生成以下内容（摘录）：

```
from datetime import timedelta
from pymongo import MongoClient

# Connection parameters
database_name = "travel-sample"

# Connect to Amazon DocumentDB cluster
client = MongoClient('<Amazon DocumentDB connection string>')

# 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")
```

有关使用 Python、、PHP、Go、Java、C#/NET Node.js、R 和 Ruby 连接亚马逊文档数据库的示例，请参阅以编程方式连接到亚马逊文档数据库。

选择迁移方法

将数据迁移到 Amazon DocumentDB 时，有两种选择：

1. 离线迁移
2. 在线迁移

离线迁移

在以下情况下，可以考虑离线迁移：

- 停机时间是可以接受的：离线迁移包括停止对源数据库的写入操作，导出数据，然后将其导入到 Amazon DocumentDB。此过程会导致您的应用程序停机。如果您的应用程序或工作负载可以容忍这段不可用期，那么离线迁移是一个可行的选择。
- 迁移较小的数据集或进行概念验证：对于较小的数据集，导出和导入过程所需的时间相对较短，这使得离线迁移成为一种快速而简单的方法。它也非常适合停机时间不那么重要的开发、测试和概念验证环境。
- 简单性是重中之重：使用 cbexport 和 mongoimport 的离线方法通常是迁移数据的最直接方法。它避免了在线迁移方法中涉及的变更数据捕获 (CDC) 的复杂性。
- 无需复制正在进行的更改：如果源数据库在迁移期间没有主动接收更改，或者如果在迁移过程中捕获这些更改并将其应用于目标数据库并不重要，则可以使用离线方法。

主题

- Couchbase Server 6.x 或更早版本
- Couchbase Server 7.0 或更高版本

Couchbase Server 6.x 或更早版本

Couchbase 存储桶到亚马逊 DocumentDB 馆藏

使用 cbexport json 导出数据，为存储桶中的所有数据创建 JSON 转储。对于该`--format`选项，你可以使用`lines`或`list`。

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

使用 mongoimport 将数据导入亚马逊文档数据库集合，并使用相应的选项来导入行或列表：

台词：

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

清单：

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

Couchbase Server 7.0 或更高版本

要执行离线迁移，请使用 cbexport 和 mongoimport 工具：

带有默认范围和默认集合的 Couchbase 存储桶

使用 cbexport json 导出数据，创建存储桶中所有集合的 JSON 转储。对于该`--format`选项，你可以使用`lines`或`list`。

```
cbexport json \
  --cluster <source cluster endpoint> \
  --bucket <bucket name> \
  --format <lines | list> \
```

```
  --username <username> \
  --password <password> \
  --output export.json \
  --include-key _id
```

使用 mongoimport 将数据导入亚马逊文档数据库集合，并使用相应的选项来导入行或列表：

台词：

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

清单：

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

Couchbase 馆藏到亚马逊 DocumentDB 馆藏

使用 cbexport json 导出数据，为每个集合创建一个 JSON 转储。使用该`--include-data`选项导出每个集合。对于该`--format`选项，你可以使用`lines`或`list`。使用`--scope-field`和`--collection-field`选项将作用域和集合的名称存储在每个 JSON 文档的指定字段中。

```
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"
```

由于 cbexport 将_scope和_collection字段添加到每个导出的文档中，因此您可以通过搜索和替换或任何您喜欢的方法将它们从导出文件中的每个文档中删除。sed

使用 mongoimport 将每个馆藏的数据导入到 Amazon Documpor t 集合中，并使用相应的选项来导入行或列表：

台词：

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

清单：

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

在线迁移

当您需要最大限度地减少停机时间并且需要近乎实时地将正在进行的更改复制到 Amazon DocumentDB 时，可以考虑在线迁移。

请参阅如何执行从 Couchbase 到亚马逊 DocumentDB 的实时迁移，了解如何实时迁移到亚马逊 DocumentDB。本文档将指导您部署解决方案以及将存储桶实时迁移到 Amazon DocumentDB 集群。

主题

- Couchbase Server 6.x 或更早版本
- Couchbase Server 7.0 或更高版本

Couchbase Server 6.x 或更早版本

Couchbase 存储桶到亚马逊 DocumentDB 馆藏

Couchbase 的迁移实用程序已预先配置为将 Couchbase 存储桶在线迁移到亚马逊 DocumentDB 集合。查看接收器连接器配置，`document.id.strategy`参数配置为使用消息键值作为`_id`字段值（参见 Sin k 连接器 ID 策略属性）：

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

Couchbase Server 7.0 或更高版本

带有默认范围和默认集合的 Couchbase 存储桶

Couchbase 的迁移实用程序已预先配置为将 Couchbase 存储桶在线迁移到亚马逊 DocumentDB 集合。查看接收器连接器配置，`document.id.strategy`参数配置为使用消息键值作为`_id`字段值（参见 Sin k 连接器 ID 策略属性）：

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

Couchbase 馆藏到亚马逊 DocumentDB 馆藏

将源连接器配置为将每个作用域中的每个 Couchbase 集合流式传输到单独的主题（请参阅源配置选项）。例如：

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

将接收器连接器配置为从每个主题流式传输到单独的 Amazon DocumentDB 集合（请参阅接收器连接器配置属性）。例如：

```
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>'
```

验证

本节提供了详细的验证流程，用于在迁移到 Amazon DocumentDB 后验证数据的一致性和完整性。无论采用何种迁移方法，验证步骤都适用。

主题

- 验证目标中是否存在所有集合
- 验证源群集和目标群集之间的文档数量
- 比较源群集和目标群集之间的文档

验证目标中是否存在所有集合

Couchbase 来源

选项 1：查询工作台

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

选项 2：cbq 工具

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

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

Amazon DocumentDB 目标

mongosh（参见 Connect 连接到你的亚马逊 DocumentDB 集群）：

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

验证源群集和目标群集之间的文档数量

Couchbase 来源

Couchbase Server 6.x 或更早版本

选项 1：查询工作台

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

选项 2：cbq

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

Couchbase Server 7.0 或更高版本

选项 1：查询工作台

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

选项 2：cbq

```
cbq \
```

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

Amazon DocumentDB 目标

mongosh（参见 Connect 连接到你的亚马逊 DocumentDB 集群）：

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

比较源群集和目标群集之间的文档

Couchbase 来源

Couchbase Server 6.x 或更早版本

选项 1：查询工作台

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

选项 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 或更高版本

选项 1：查询工作台

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

选项 2：cbq

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

Amazon DocumentDB 目标

mongosh（参见 Connect 连接到你的亚马逊 DocumentDB 集群）：

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

升级 Amazon DocumentDB

主题

- Amazon DocumentDB 引擎版本支持日期
- Amazon DocumentDB 主版本就地升级
- 亚马逊 DocumentDB 次要版本升级
- 使用升级您的亚马逊文档数据库集群 AWS Database Migration Service
- 使用长期支持（LTS）版本

Amazon DocumentDB 引擎版本支持日期

您可以参照下列日期规划您的测试和升级周期。

亚马逊 DocumentDB 主要版本的发布日历

Amazon DocumentDB 目前支持以下主要版本。要升级您的引擎版本，请参阅升级主引擎版本或使用升级 AWS DMS。

引擎版本	发行日期	标准支持结束日期	扩展支持开始日期（第 1 年定价）	扩展支持开始日期（第 3 年定价）	扩展支持结束日期
版本 3.6	2019 年 1 月 9 日	2026 年 3 月 30 日	2026 年 3 月 31 日	2028 年 3 月 31 日	2029 年 3 月 30 日
版本 4.0	2020 年 11 月 9 日	N/A	N/A	N/A	N/A
版本 5.0	2023 年 3 月 1 日	N/A	N/A	N/A	N/A
版本 8.0	2025 年 11 月 14 日	N/A	N/A	N/A	N/A

在标准支持下，Amazon DocumentDB 支持多个引擎版本。在标准支持终止日期之后，您可以继续运行版本，但需支付扩展支持费用。有关更多信息，请参阅 Amazon DocumentDB 扩展支持 和 Amazon DocumentDB 定价。

亚马逊 DocumentDB 次要版本的发布日历

Amazon DocumentDB 目前支持以下次要版本。发布时间表将根据其他功能或修复程序而有所不同。次要版本可能比相应主要版本更早结束标准支持。要升级您的引擎版本，请参阅升级次要引擎版本。

> **Note**
>
> 从主版本 5.0 开始提供次要版本。

引擎版本	发行日期
版本 5.0.0 (LTS)	2026 年 2 月 16 日
版本 5.0.1	2026 年 6 月 8 日

亚马逊 DocumentDB 将某些版本指定为 Long-Term 支持 (LTS) 版本。有关更多信息，请参阅 使用长期支持（LTS）版本。

Amazon DocumentDB 主版本就地升级

您可以对 Amazon DocumentDB 集群执行就地主版本升级 (MVU)，同时保持相同的终端节点、存储空间和标签。您的应用程序无需修改即可继续运行。此功能在所有提供 Amazon DocumentDB 的地区均可免费使用。

> **Important**
>
> 在升级期间，您的集群将不可用，并且会经历多次重启。开始升级后，请勿连接到集群、读取集群或写入集群。停机时间因集合、索引、数据库和实例的数量而异。我们建议在维护时段或低利用率时段执行升级。
>
> 一旦升级，就无法降级到以前的版本。如果需要，您可以将升级前的快照还原到新集群。

主题

- 支持的升级途径
- 先决条件
- 最佳实践
- 从 Amazon DocumentDB 5.0 升级到 8.0 后会发生什么变化
- Post-upgrade 从 3.6 或 4.0 升级的集群的注意事项
- 执行升级
- 问题排查

支持的升级途径

源版本	目标版本
亚马逊 DocumentDB 3.6	Amazon DocumentDB 5.0
Amazon DocumentDB 4.0	Amazon DocumentDB 5.0
Amazon DocumentDB 5.0	亚马逊 DocumentDB 8.0

> **Note**
>
> In-place 全局群集或弹性集群不支持 MVU。要升级全局群集，请移除辅助群集，将主群集转换为区域群集，执行 MVU，然后通过添加使用相同名称的辅助集群来重新创建全局群集以保留您的终端节点。当升级的主服务器将数据复制到新的辅助服务器时，您将产生 I/O 费用。有关详细步骤，请参阅从 Amazon DocumentDB 全局集群中删除某集群。

先决条件

- 实例类型 — 亚马逊 DocumentDB 4.0+ 不支持 db.r4 实例。在升级之前，请将任何`db.r4.*`或`db.r5.*`实例修改为实例或更新的实例。请参阅修改 Amazon DocumentDB 实例和不同区域支持的实例类。
- 操作系统补丁-升级之前，对所有实例应用所有待处理的操作系统维护操作。请参阅Amazon DocumentDB 操作系统更新。

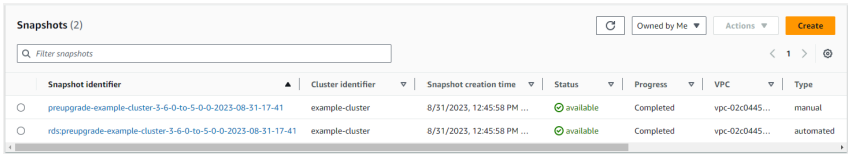
> **Note**
>
> 待处理的集群级引擎补丁可能会隐藏实例操作系统补丁。如果需要，请先应用引擎补丁。请参阅对集群的引擎版本执行补丁更新。

- 突发性能实例（t-family 实例）的索引限制 — 如果您在突发性能实例上有超过 3,000 个索引，请在升级之前将主实例扩展到至少 db.r5.large。升级完成后，您可以缩小规模。

实例	MVU 的最大索引
db.t4g.medium	3K
db.t3.medium	10K

- 参数组-我们建议在升级之前为目标版本准备好自定义集群参数组。如果未指定参数组，则将使用目标版本的默认参数组（例如，`default.docdb5.0`或`default.docdb8.0`）。
- 手动快照-在升级之前创建手动快照。升级过程会创建名为的自动快照`preupgrade-<name>-<version>-<timestamp>`，但我们强烈建议您拥有自己的备份。请参阅创建手动集群快照。

> **Note**
>
> 主版本就地升级已完成后，升级过程创建的自动快照不会自动删除。只要这个快照在保留期范围内，它就不会发生任何费用。一旦您已经确认自己的集群升级成功，您就可以选择删除此快照。



最佳实践

升级之前 — 使用克隆进行测试

1. 克隆 Amazon DocumentDB 集群卷用于创建集群的克隆。除非修改克隆上的数据，否则不会产生存储成本。

2. 将克隆的实例计数与目标集群进行匹配，以获得真实的时间估计。
3. 在克隆上执行 MVU 并全面测试功能差异。
4. 运行describe-db-clusters并查找，检查是否已安排升级PendingModifiedValues.EngineVersion。如果您修改了集群并选择在下一个维护时段内应用该集群，则该计划将不会显示在控制台中，但您可以在以下位置查看 AWS CLI：

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

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

5. 如果测试成功，请继续在生产集群上进行升级。

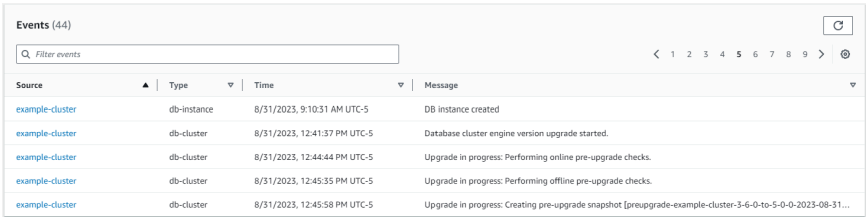
升级期间

您可以通过订阅集群维护事件来监控主版本就地升级进度。升级完成后，您将收到 “数据库集群主版本已升级” 事件。升级期间发生的此事件和其他事件显示在 Amazon DocumentDB 控制台集群详情页面的 “事件和标签” 部分。然后，集群状态upgrading将从变为available。

升级期间会生成以下事件：

1. 数据库集群引擎主版本升级已启动。集群保持联机状态。
2. 正在进行升级准备：正在启动在线升级预检查。
3. 正在进行升级准备：已完成在线升级预检查。
4. 在主实例完成该 patch/upgrade 过程的同时使数据库集群脱机。
5. 升级准备进行中：开始离线升级预检。
6. 升级准备正在进行中：已完成离线升级预检查。
7. <version-to><timestamp>升级正在进行中：正在创建升级前快照 [升级前-<cluster-name>-<version-from>--]。
8. 升级进行中：正在克隆卷。
9. 升级进行中：正在升级写入器。
10. 升级进行中：正在升级阅读器。
11. 数据库集群引擎主要版本已升级。

事件在控制台的 “事件” 页面下方可见：



从中 AWS CLI，您可以运行`aws docdb describe-events`以监控升级进度。要自动接收通知，请使用订阅事件并将其发送`aws docdb create-event-subscription`到 Amazon SNS 以获取电子邮件、推送消息或其他传送方式。有关更多信息，请参阅 订阅 Amazon DocumentDB 事件。

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

该命令返回的输出类似于下方内容：

```
{
    "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"
        }
    ]
}
```

升级后

> **Warning**
>
> 就地升级主版本后，您的 Amazon DocumentDB 集群会立即重新填充数据库引擎用于优化查询执行计划的索引元数据。此过程完成后，查询性能将恢复到预期水平。它通常会在几分钟内完成，但最多可能需要两个小时，具体取决于集群上的索引数量。

在此期间，请勿重启、故障转移或扩展 up/down 您的写入器实例，因为这可能会中断索引元数据的重新计算。我们建议您等到观察到预期的查询性能后再进行此类更改。
通过以下集群事件跟踪进度：

- Post-upgrade 集群状态：索引元数据刷新过程已启动
- Post-upgrade 集群状态：索引元数据刷新过程在 X 秒内完成

如果索引元数据刷新过程未在三小时内完成，或者该过程完成后您仍然遇到性能问题，请联系 AWS 支持人员。

1. 手动拍摄升级后的集群快照，以备需要恢复到升级后的状态。就地主版本升级完成后，自动快照过程将立即恢复。
2. 标记从 3.6 升级的集群。添加标签以区分从 3.6 升级的集群。（参见the section called “Post-upgrade 从 3.6 或 4.0 升级的集群的注意事项”）。
3. 更新您的驱动程序。要使用新功能（例如 8.0 中的排序规则、视图或 Zstd 压缩），请升级到相应的 MongoDB API 版本。有关更多信息，请参阅 Amazon DocumentDB 8.0 的新增功能。
4. 彻底测试。根据升级后的集群验证您的应用程序。

从 Amazon DocumentDB 5.0 升级到 8.0 后会发生什么变化

在将主要版本从 Amazon DocumentDB 5.0 升级到 8.0 之后，将启用或更改以下功能：

- 排序规则。亚马逊 DocumentDB 8.0 支持排序规则。升级后，默认情况下，新集合及其索引以及现有集合上的新索引将启用排序规则。
- 文本索引。新的文本索引是使用文本索引 V2 创建的，该版本使用更新的文本搜索解析器来改进 MongoDB 兼容性。现有文本索引不受影响。
- 查询计划器版本。如果您没有自定义参数组，则会为 Amazon DocumentDB 8.0 创建一个新的默认参数组，并自动选择 Planner 版本 3。在查询计划器版本 3 中，还可以使用视图。
- 压缩。亚马逊 DocumentDB 8.0 支持使用 Zstd 算法进行基于字典的文档压缩。升级后，将在默认启用 Zstd 压缩的情况下创建新的集合。5.0 版本的现有集合保留其压缩设置。要在现有集合上使用 Zstd 压缩，可以修改其压缩设置。有关更多信息，请参阅Dictionary-based 压缩。
- 索引重建。如果您要从亚马逊 DocumentDB 5.0 升级到亚马逊 DocumentDB 8.0，则无需重建索引。

> **Important**
>
> 亚马逊 DocumentDB 8.0 需要 TLS 1.2 或更高版本。不再支持 TLS 1.0 和 TLS 1.1。

> **Note**
>
> 有关功能差异的完整列表，请参阅Amazon DocumentDB 与 MongoDB 兼容性。

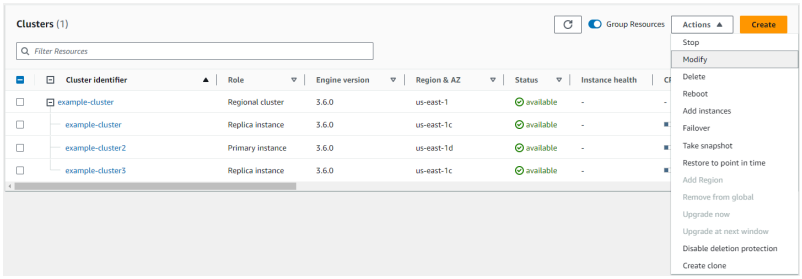
Post-upgrade 从 3.6 或 4.0 升级的集群的注意事项

- 索引重建。MVU 会保留原始索引。Amazon DocumentDB 5.0 改进了索引维护和垃圾回收，特别是对于低基数索引。从 3.6 或 4.0 升级后，我们建议重新构建索引以确保最佳查询性能（可选，涉及其他内容 I/O）。请参阅使用 `re Index` 维护索引。
- 子文档数值比较（仅限 3.6）。从 3.6 升级的集群继承了 3.6 行为，即子文档中的数字类型不跨类型进行比较。例如，`{a: {b: 1}}`在 3.6 中`{a: {b: NumberLong(1)}}`不等于，但在 4.0 及更高版本中它们相等。此行为会影响从 3.6 升级的所有集群。

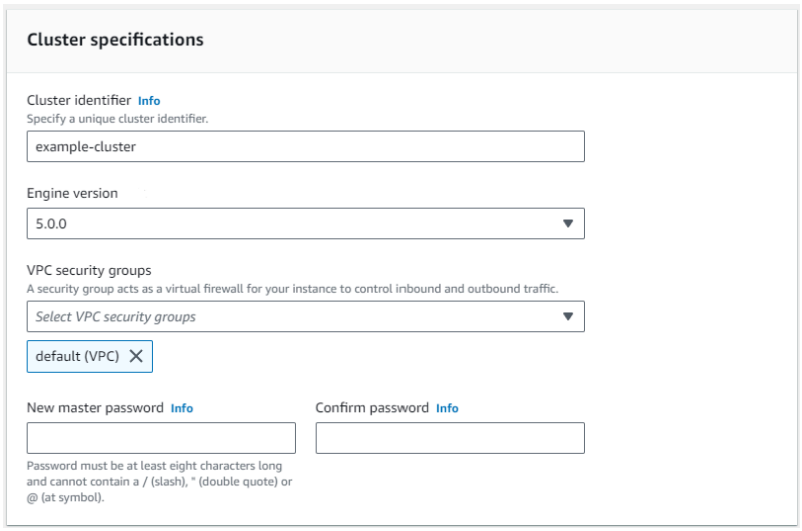
执行升级

Using the AWS 管理控制台

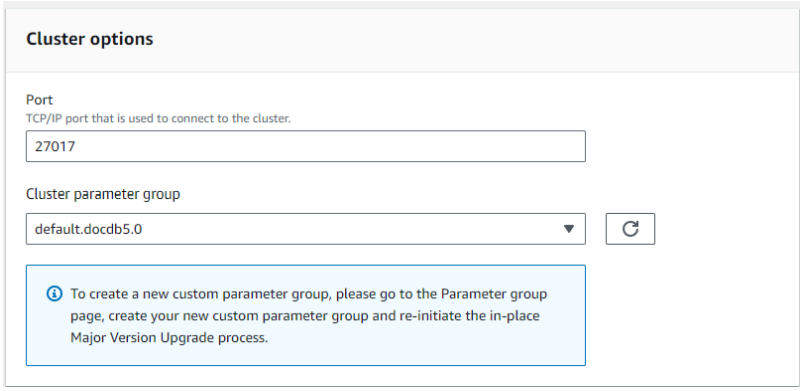
1. 登录 AWS 管理控制台并 打开 Amazon DocumentDB 控制台。
2. 在集群表中，选择源集群，选择操作，然后选择修改。



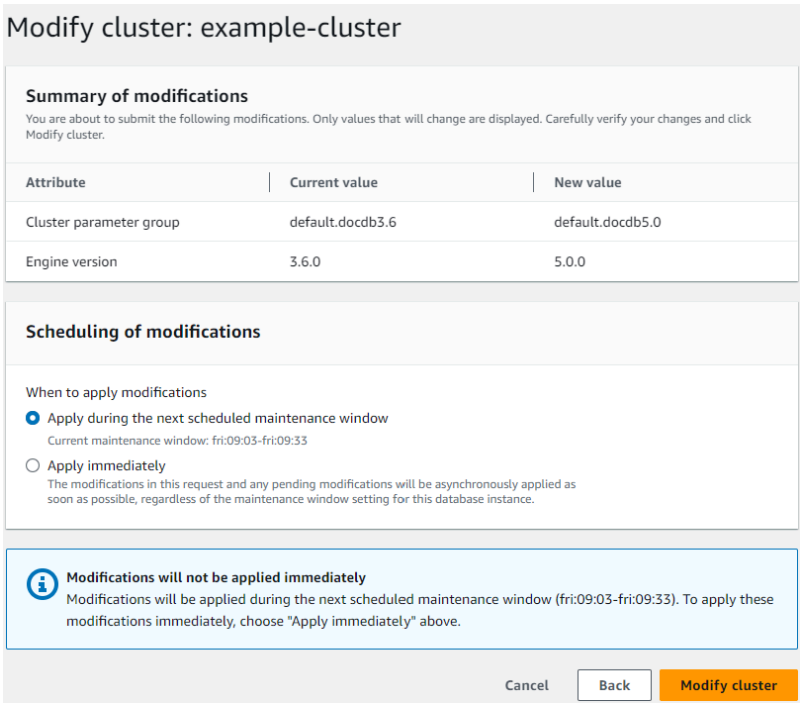
3. 在集群规格中，从引擎版本下拉列表中选择目标版本（例如 5.0.0 或 8.0.0）。



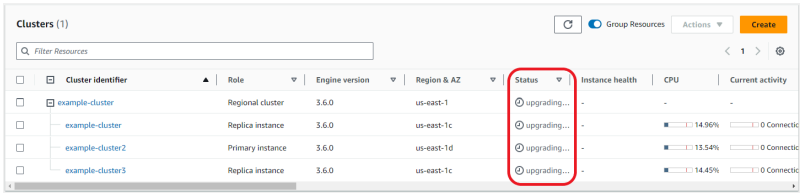
4. 在集群选项中，为目标引擎版本选择集群参数组。您可以使用默认参数组（例如，default.docdb5.0 或 default.docdb 8.0）或您创建的自定义参数组。



5. 选择继续，选择您的计划首选项（立即应用或下一个维护时段），然后选择修改集群。



6. 当集群变为升级时，监控集群表中的集群状态：



Using the AWS CLI

[modify-db-cluster](#)与`--allow-major-version-upgrade`旗帜一起使用：

```
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
```

*placeholder*用集群的信息替换每个信息。

问题排查

- 升级失败并回滚。如果升级失败，它将自动尝试回滚。成功回滚会生成以下事件：“数据库集群处于无法升级的状态。” 您的集群恢复到升级前的状态，您可以继续使用它。重新尝试之前，请联系 AWS 支持人员进行故障排除。
- Post-upgrade 性能。索引元数据刷新运行时，可能会出现暂时的性能下降和 CPU 使用率过高。如果降级持续超过 3 小时，请联系 AWS 支持人员。

如需其他帮助，请联系 Supp AWS ort。

亚马逊 DocumentDB 次要版本升级

> **Note**
>
> 本主题介绍同一主要版本中的次要版本升级（例如，从 5.0.0 升级到 5.0.1）。有关升级到其他主要版本的信息，请参阅Amazon DocumentDB 主版本就地升级。

您可以通过修改集群来升级 Amazon DocumentDB 集群的次要引擎版本（例如，从 5.0.0 升级到 5.0.1）。次要版本升级可以立即应用，也可以在下一个维护时段内应用。

在执行次要版本升级之前

在执行次要版本升级之前，以下操作可能会有所帮助：

- 在流量较低的时段执行升级，以减少任何短暂停机时间的影响。
- 使用`describe-db-engine-versions`命令验证目标次要版本是否是当前版本的有效升级目标。
- 对于全局群集，必须先升级辅助群集，然后再升级主群集。有关更多信息，请参阅 全球集群修补。

> **Important**
>
> - 升级是单向操作 ——升级后不能降级。
> - 在升级过程中，您的集群将经历短暂的停机时间。

执行次要版本升级

Using the AWS 管理控制台

要执行次要版本升级，请使用 AWS 管理控制台：

1. 登录 AWS 管理控制台并 打开 Amazon DocumentDB 控制台。
2. 在集群表中，选择要升级的集群，选择操作，然后选择修改。
3. 在修改集群对话框的集群规格部分，从引擎版本下拉菜单中选择目标次要版本。
4. 向下滚动并选择 “继续”。
5. 在 “修改计划” 部分中，选择立即应用更改或在下一个维护时段内应用更改。

   然后选择 Modify cluster (修改集群)。

Using the AWS CLI

将 `modify-db-cluster` 命令与 `--engine-version` 参数一起使用：

```
aws docdb modify-db-cluster \
    --db-cluster-identifier mydocdbcluster \
    --engine-version 5.0.1 \
    --apply-immediately \
    --region us-east-1
```

在上面的示例中，将每个*user input placeholder*替换为集群的信息。

> **Note**
>
> 要在下一个维护时段而不是立即应用升级，请使用`--no-apply-immediately`代替`--apply-immediately`。

验证次要版本升级

升级完成后，使用以下命令验证引擎版本 AWS CLI：

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

此操作的输出将类似于以下内容：

```
"5.0.1"
```

使用升级您的亚马逊文档数据库集群 AWS Database Migration Service

> **Important**
>
> Amazon DocumentDB 的支持生命周期与 MongoDB 不同，MongoDB 的生命周期终止使用计划并不适用于 Amazon DocumentDB。

您可以使用将您的 Amazon DocumentDB 集群升级到更高的版本，同时最大限度地减少停机时间。AWS DMS AWS DMS 是一项完全托管的服务，可以轻松地从较旧的 Amazon DocumentDB 版本、关系数据库和非关系数据库迁移到您的目标 Amazon DocumentDB 集群。

主题

- 步骤 1：启用变更流
- 步骤 2：修改变更流保留期限
- 步骤 3：迁移您的索引
- 步骤 4：创建一个 AWS DMS 复制实例
- 步骤 5：创建一个 AWS DMS 源端点
- 步骤 6：创建一个 AWS DMS 目标端点
- 步骤 7：创建并运行迁移任务
- 步骤 8：将应用程序端点更改为目标 Amazon DocumentDB 集群

步骤 1：启用变更流

要最大限度地缩短停机时间迁移， AWS DMS 需要访问集群的变更流。Amazon DocumentDB 变更流提供在集群的集合和数据库中发生的按时间顺序排列的更新事件序列。通过从变更流读取，可以 AWS DMS 执行变更数据捕获 (CDC)，并将增量更新应用于目标 Amazon DocumentDB 集群。

要为特定数据库上的所有集合启用变更流，请使用 mongo Shell 对您的 Amazon DocumentDB 集群进行身份验证并执行以下命令：

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

步骤 2：修改变更流保留期限

接下来，根据您希望在变更流中保留变更事件的时长来修改变更流保留期。例如，如果您预计使用 AWS DMS 的 Amazon DocumentDB 集群迁移需要 12 个小时，则应将更改流保留时间设置为大于 12 小时的值。Amazon DocumentDB 集群的默认保留期为三小时。您可以使用 AWS 管理控制台 或将 Amazon DocumentDB 集群的更改流日志保留期限修改为一小时到七天之间。 AWS CLI有关更多详细信息，请参阅修改变更流日志保留期。

步骤 3：迁移您的索引

在您的目标 Amazon DocumentDB 集群上创建与源 Amazon DocumentDB 集群相同的索引。尽管它 AWS DMS 可以处理数据的迁移，但它不迁移索引。要迁移索引，请使用 Amazon DocumentDB 索引工具从源 Amazon DocumentDB 集群中导出索引。您可以通过创建 Amazon DocumentDB 工具 GitHub 存储库的克隆版并按照中的说明进行操作来获取该工具。README.md您可以从亚马逊 EC2 实例或与您的亚马逊文档数据库集群运行在同一 Amazon VPC 中的 AWS Cloud9 环境中运行该工具。

在以下示例中，将每个 *user input placeholder* 替换为您自己的信息。

以下代码从您的源 Amazon DocumentDB 集群中转储索引：

```
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/
```

成功导出索引后，即可在目标 Amazon DocumentDB 集群中还原这些索引。要还原您在上一步中所导出的索引，请使用 Amazon DocumentDB 索引工具。以下命令将从指定目录还原目标 Amazon DocumentDB 集群中的索引。

```
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
```

要确认您已正确还原索引，请使用 mongo Shell 连接到您的目标 Amazon DocumentDB 集群，并列出给定集合的索引。参见以下代码：

```
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()
```

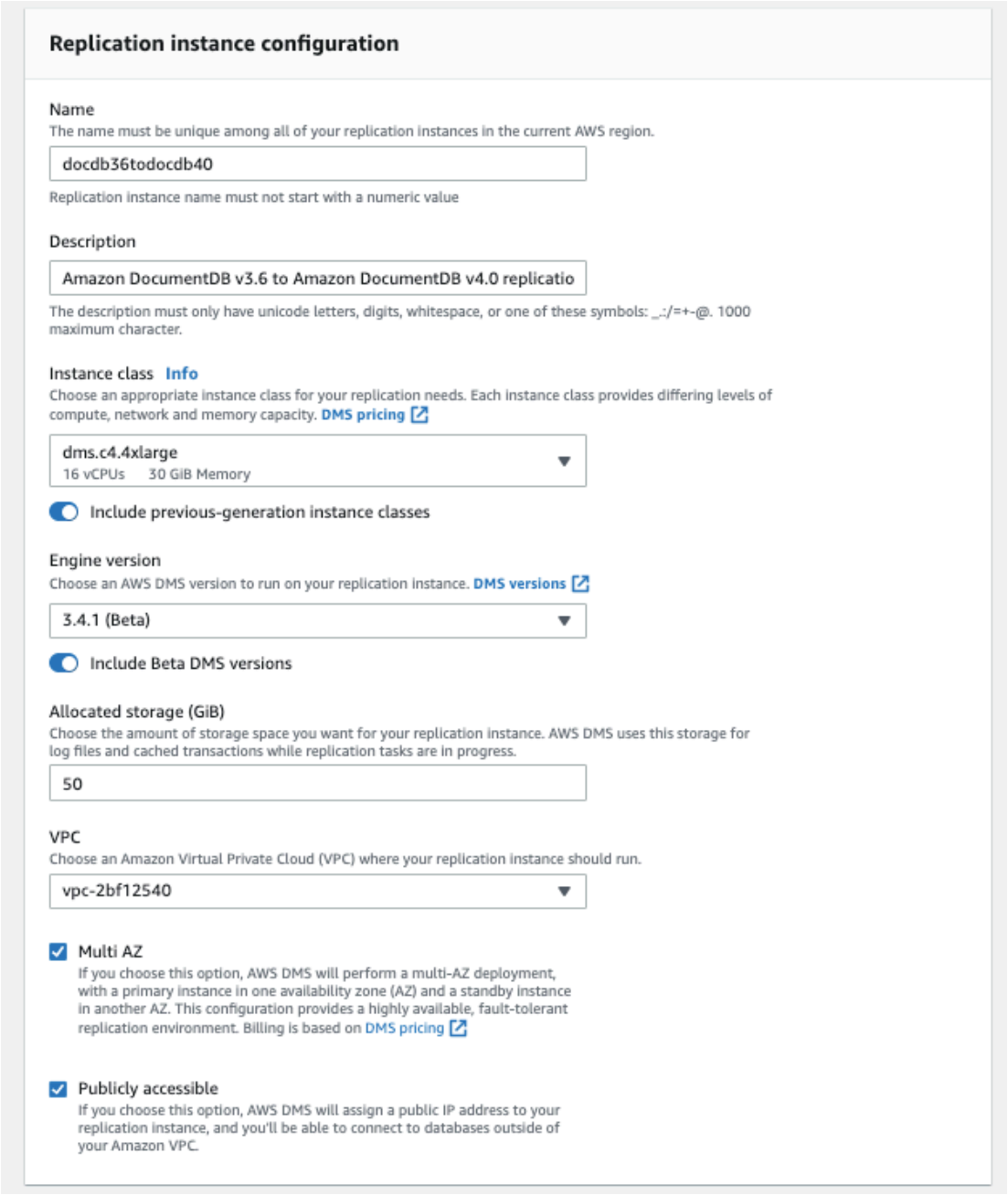
步骤 4：创建一个 AWS DMS 复制实例

AWS DMS 复制实例连接您的源 Amazon DocumentDB 集群并从中读取数据，然后将其写入您的目标 Amazon DocumentDB 集群。 AWS DMS 复制实例可以执行批量加载和 CDC 操作。大部分这种处理发生在内存中。但是，大型事务可能需要部分缓冲到磁盘上。缓存事务和日志文件也会写入磁盘。迁移数据后，复制实例还将流式传输所有更改事件，以确保源和目标保持同步。

要创建 AWS DMS 复制实例，请执行以下操作：

1. 打开控制 AWS DMS 台。
2. 在导航窗格中，选择复制实例。
3. 选择 Create replication instance (创建复制实例)，并输入以下信息：
   - 对于 Name（名称），输入您选定的名称。例如 `docdb36todocdb40`。
   - 对于 Description (描述)，输入事件的描述。对于 listitem，Amazon DocumentDB 3.6 至 Amazon DocumentDB 4.0 的复制实例。
   - 对于实例类，请根据需要选择大小。
   - 对于引擎版本，请选择`3.4.1.`

- 对于 Amazon VPC，请选择存放源集群和目标 Amazon DocumentDB 集群的 Amazon VPC。
- 对于已分配存储（GiB），请使用默认值 50 GiB。如果工作负载的写入吞吐量较高，请增加此值以匹配您的工作负载。
- 对于 Multi-AZ，如果您需要高可用性和故障转移支持，请选择 “是”。
- 对于 Publicly accessible (公开访问)，请启用此选项。



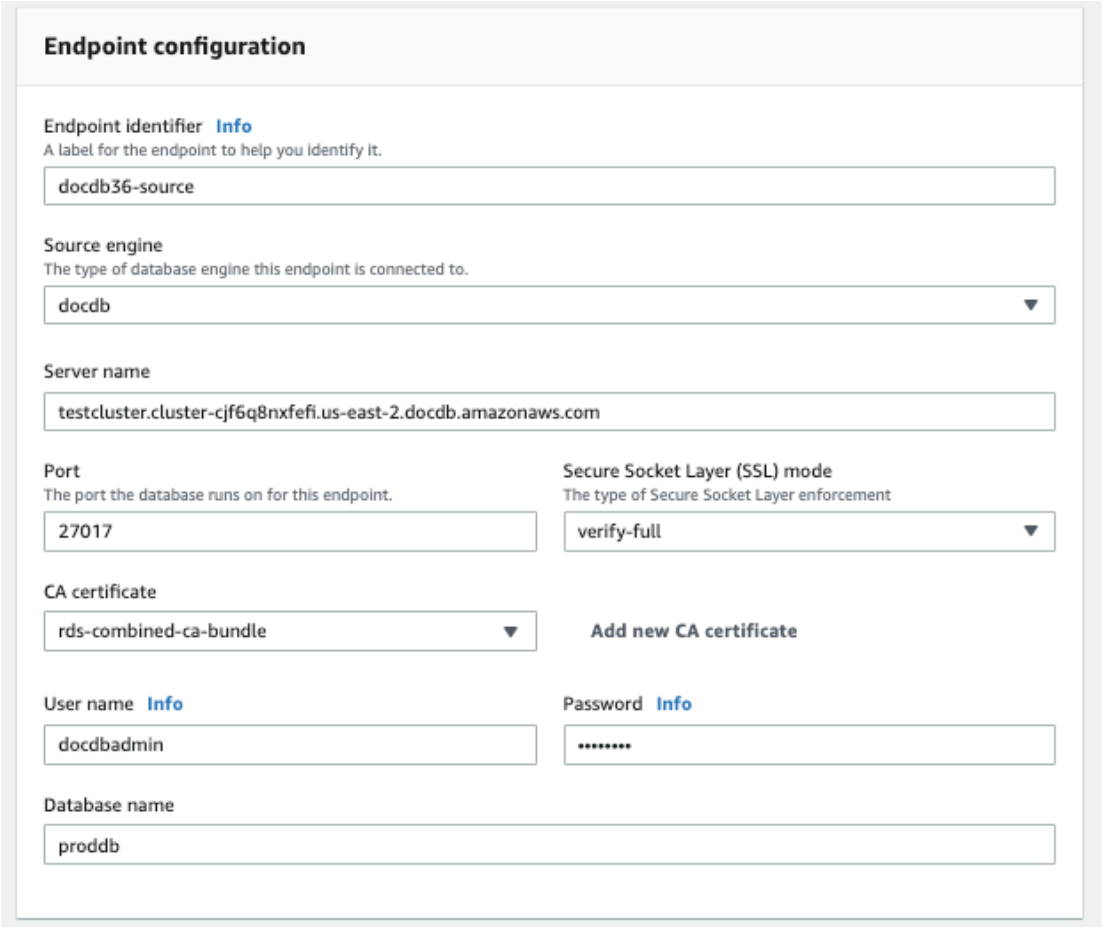
4. 选择创建复制实例。

步骤 5：创建一个 AWS DMS 源端点

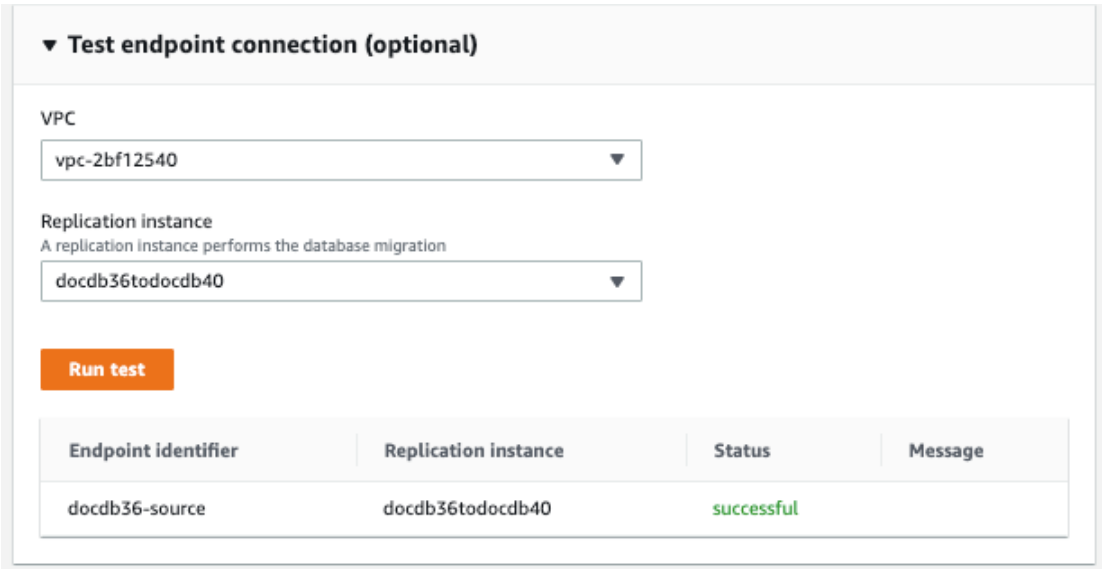
源终端点用于源 Amazon DocumentDB 集群。

创建源端点

1. 打开控制 AWS DMS 台。
2. 在导航窗格中，选择端点。
3. 选择 `Create endpoint` 并输入以下信息：
   - 对于 Endpoint type (端点类型)，请选择 Source (源)。
   - 对于 Endpoint identifier (端点标识符)，请输入容易记住的名称，例如 `docdb-source`。
   - 对于源引擎，请选择 `docdb`。
   - 对于 Server name (服务器名称)，请输入您的 Amazon DocumentDB 集群的 DNS 名称。
   - 对于 Port (端口)，请输入您的 Amazon DocumentDB 集群的端口号。
   - 对于 SSL mode (SSL 模式)，请选择 `verify-full`。
   - 对于 CA 证书，请选择添加新 CA 证书。下载新 CA 证书、以创建 TLS 连接捆绑。然后，对于 Certificate identifier (证书标识符)，输入 `rds-combined-ca-bundle` 。对于 Import certificate file (导入证书文件)，选择 Choose file (选择文件)，然后导航到之前下载的 `.pem` 文件。选择并打开此文件。选择导入证书，然后`rds-combined-ca-bundle`从选择证书下拉列表中选择
   - 对于用户名，请输入您的源 Amazon DocumentDB 集群的主用户名。
   - 对于密码，请输入您的源 Amazon DocumentDB 集群的主密码。
   - 对于数据库名称，请输入要升级的数据库名称。



4. 测试您的连接以验证其已成功设置。



5. 选择创建端点。

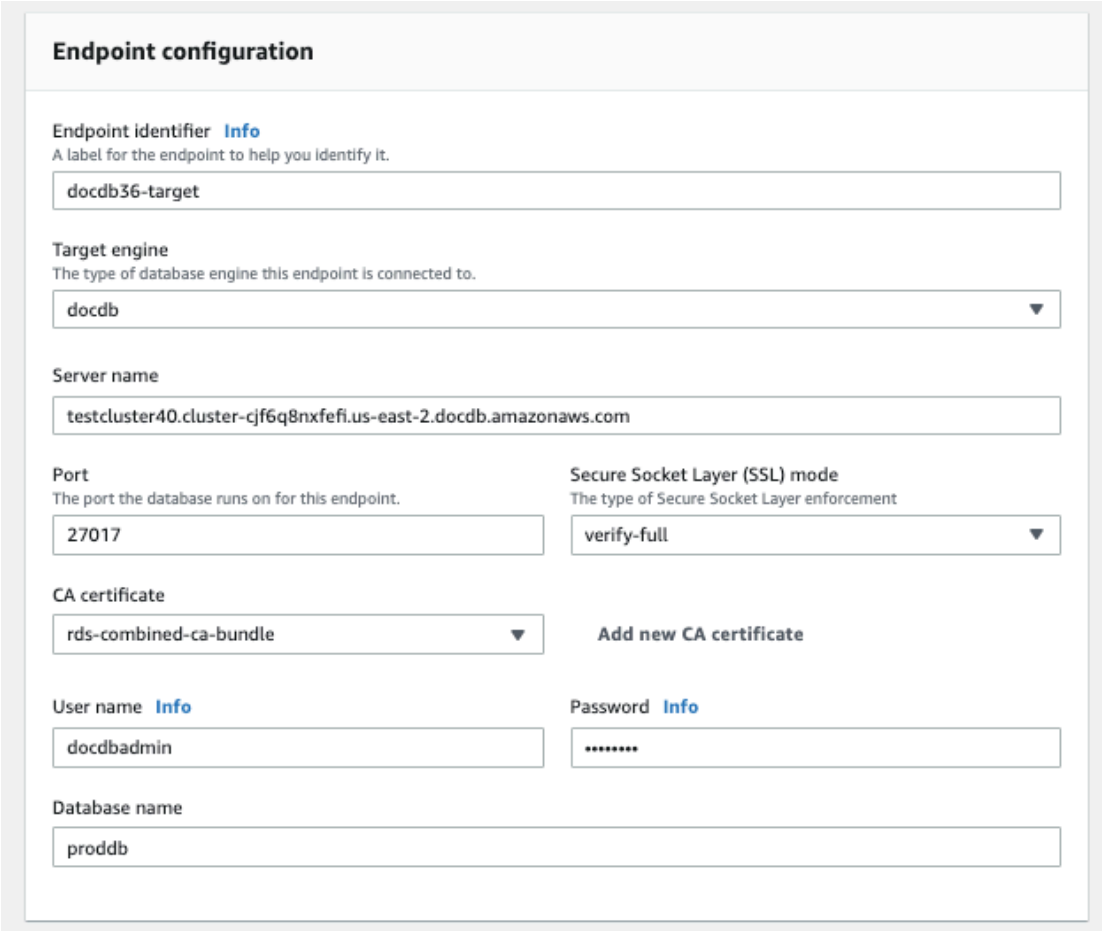
> **Note**
>
> AWS DMS 一次只能迁移一个数据库。

步骤 6：创建一个 AWS DMS 目标端点

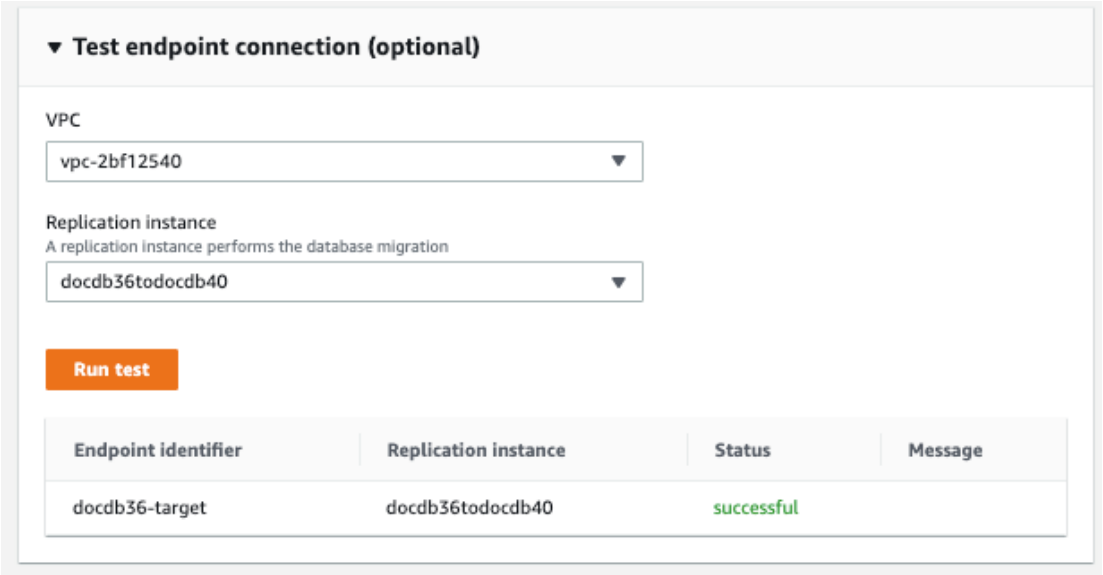
目标端点用于您的目标 Amazon DocumentDB 集群。

创建目标端点：

1. 打开 AWS DMS 控制台。
2. 在导航窗格中，选择端点。
3. 选择 Create endpoint (创建端点)，然后输入以下信息：
   - 对于 Endpoint type (端点类型)，请选择 Target (目标)。
   - 对于 Endpoint identifier (端点标识符)，请输入容易记住的名称，例如 `docdb-target`。
   - 对于源引擎，请选择 `docdb`。
   - 对于 Server name (服务器名称)，请输入您的 Amazon DocumentDB 集群的 DNS 名称。
   - 对于 Port (端口)，请输入您的 Amazon DocumentDB 集群的端口号。
   - 对于 SSL mode (SSL 模式)，请选择 `verify-full`。
   - 对于 CA 证书，请从选择证书下拉列表中选择现有`rds-combined-ca-bundle` 证书。
   - 对于用户名，请输入您的目标 Amazon DocumentDB 集群的主用户名。
   - 对于密码，请输入您的目标 Amazon DocumentDB 集群的主密码。
   - 对于数据库名称，请输入与源端点设置相同的数据库名称。



4. 测试您的连接以验证其已成功设置。



5. 选择创建端点。

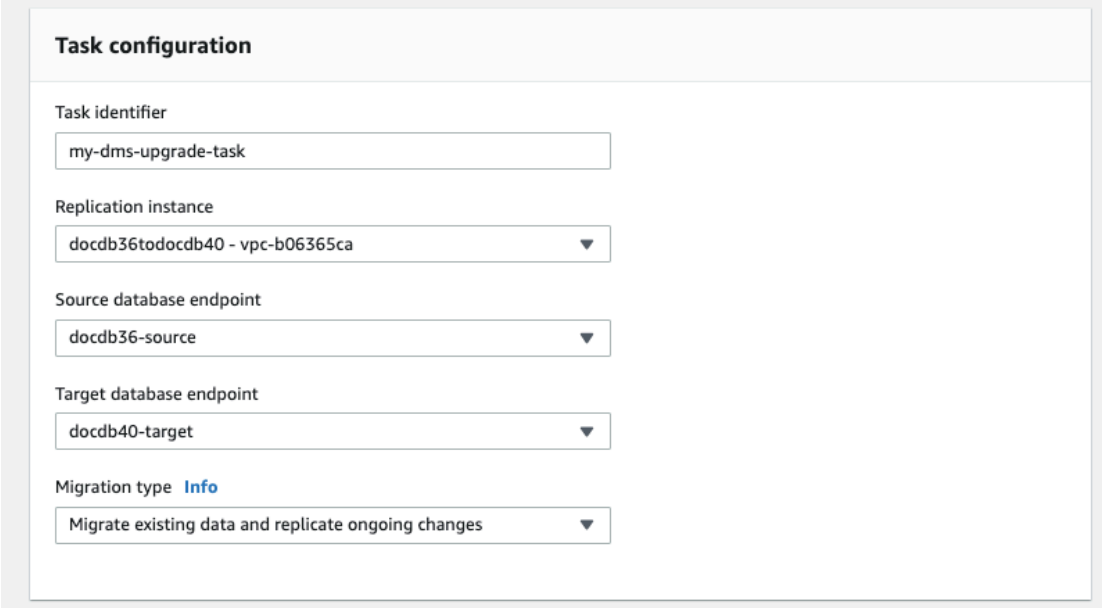
步骤 7：创建并运行迁移任务

AWS DMS 任务会将复制实例与您的源实例和目标实例绑定。创建迁移任务时，需要指定源终结点、目标终结点、复制实例和任何所需的迁移设置。可以用三种不同的迁移类型创建 AWS DMS 任务：迁移现有数据、迁移现有数据、复制正在进行的更改或仅复制数据更改。由于本演练的目的是在最短停机时间内升级 Amazon DocumentDB 集群，因此这些步骤利用该选项来迁移现有数据和复制正在进行的更改。使用此选项，可以在迁移现有数据时 AWS DMS 捕获更改。 AWS DMS 即使加载了批量数据，仍会继续捕获和应用更改。最终，源数据库和目标数据库将保持同步，从而实现停机时间最少的迁移。

以下是创建迁移任务以实现最短停机时间迁移的步骤：

1. 打开控制 AWS DMS 台。
2. 在导航窗格中，选择数据库迁移任务。
3. 选择创建数据库迁移任务，然后在任务配置部分中输入以下信息：

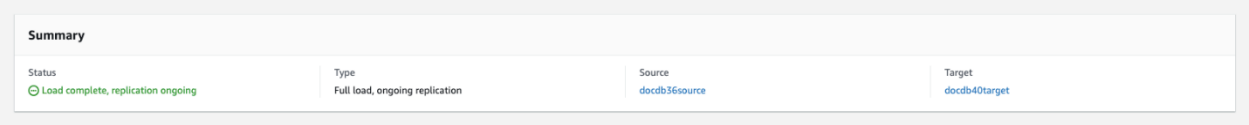
   - 对于端点标识符，请输入容易记住的名称，例如 `my-dms-upgrade-task`。
   - 对于描述性 Amazon 资源名称（ARN），输入一个用户友好名称以覆盖默认 DMS ARN。
   - 对于 Replication instance (复制实例)，请选择您在 步骤 4：创建一个 AWS DMS 复制实例 中创建的复制实例。
   - 对于源数据库端点，选择您在 步骤 5：创建一个 AWS DMS 源端点 中创建的源端点。
   - 对于目标端点，选择您在 步骤 6：创建一个 AWS DMS 目标端点 中创建的目标端点。
   - 对于迁移类型，选择迁移和复制。



4. 在任务设置部分中输入以下信息：

   - 对于目标表准备模式部分，选择不执行任何操作。这将确保在步骤 3 中创建的索引不会被删除。
   - 在 “任务日志” 子部分中，选择 “开启日 CloudWatch 志”。
   - 对于迁移任务启动配置，选择创建时自动启动。创建迁移任务后，将自动启动迁移任务。
   - 选择创建数据库迁移任务。

AWS DMS 现在开始将数据从您的源 Amazon DocumentDB 集群迁移到您的目标 Amazon DocumentDB 集群。任务状态从 Starting (正在启动) 更改为 Running (正在运行)。您可以通过在 AWS DMS 控制台中选择 “任务” 来监控进度。几次之后 minutes/hours （取决于迁移的大小），状态应从变为加载完成，复制正在进行中。这意味着 AWS DMS 已完成从您的源 Amazon DocumentDB 集群到目标 Amazon DocumentDB 集群的满负荷迁移，现在正在复制更改事件。



最终，您的源和目标将保持同步。您可以通过对集合运行 `count()` 操作来验证所有更改事件是否已迁移，从而验证其是否处于同步状态。

步骤 8：将应用程序端点更改为目标 Amazon DocumentDB 集群

完成完全加载并且 CDC 流程持续复制后，您即可将应用程序的数据库连接端点从源 Amazon DocumentDB 集群更改为目标 Amazon DocumentDB 集群。

使用长期支持（LTS）版本

亚马逊 DocumentDB 将某些版本指定为 Long-Term 支持 (LTS) 版本。与使用非 LTS 版本的集群相比，使用 LTS 的数据库集群在相同版本上停留的时间更长，升级周期也更短。只有在出现关键稳定性和安全补丁时才会修补 LTS 版本，而不引入新功能。可能会发布其他可选补丁，例如用于调试关键故障的指标。这些将与所需的补丁一起累积使用。

当前的 LTS 版本

Amazon DocumentDB 的当前 LTS 版本是：

- 次要版本 5.0.0

要确定您的集群所在的版本，请参阅the section called “如何检查我现有的引擎版本 5.0.0 集群是否在 LTS 上”。

有关 LTS 版本的支持时间表和发布周期的详细信息，请参阅 Amazon DocumentDB 引擎版本支持日期。亚马逊 DocumentDB 将通过标准沟通渠道宣布下一代 LTS 版本。

> **Note**
>
> 尚未为亚马逊 DocumentDB 8.0 指定一个 LTS 版本。如果您需要 LTS，请使用引擎版本 5.0.0 和补丁版本 3.0.17983 或更高版本。

谁应该使用 LTS

亚马逊 DocumentDB LTS 版本专为升级窗口有限的集群而设计。LTS 非常适合数据库可用性直接影响业务运营的生产环境。

在以下情况下，可以考虑发布 LTS 版本：

- 您的应用程序有严格的正常运行时间要求，这限制了升级机会
- 您拥有应用程序所需的所有必需的数据库引擎功能和增强功能

在以下情况下，可以考虑使用标准版本：

- 您的应用程序可以处理最少的中断
- 您希望在最新功能和增强功能发布后立即获得它们

> **Important**
>
> - 升级是单向操作-升级后无法降级
> - 在升级过程中，您的集群将经历短暂的停机时间

如何创建新的 LTS 集群

要创建 LTS 集群，请执行以下步骤：

1. 在管理控制台中，创建一个新的引擎版本 5.0.0 集群
2. 创建集群后，使用以下命令验证您使用的是引擎版本 5.0.0： AWS CLI

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

此操作的输出将类似于以下内容：

```
"5.0.0"
```

如何从引擎版本 3.6.0 或 4.0.0 升级到 5.0.0 的 LTS 集群

要升级到 LTS 集群，请执行以下步骤：

1. 在管理控制台中，在升级亚马逊 DocumentDB 引擎版本后，将引擎版本 3.6.0 或 4.0.0 集群的主版本升级 (MVU) 到 5.0.0
2. 升级后使用以下命令验证您使用的是引擎版本 5.0.0： AWS CLI

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

如何检查我现有的引擎版本 5.0.0 集群是否在 LTS 上

要检查您的引擎版本 5.0.0 集群是否运行 LTS 版本，请执行以下操作：

1. 使用以下命令验证您使用的是引擎版本 5.0.0： AWS CLI

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

2. 如果需要，请在对集群的引擎版本执行补丁更新之后升级现有引擎版本 5.0.0 集群

Amazon DocumentDB 扩展支持

借助 Amazon DocumentDB 扩展支持，您可以在标准支持终止日期后继续运行引擎版本，但需要额外付费。您的集群将在标准支持终止日期后注册到扩展支持中。如果您未在引擎版本标准支持终止日期前完成升级，则您的集群将按扩展支持费率计费。

主题

- 扩展支持概述
- 扩展支持费用
- 责任

Amazon DocumentDB 扩展支持提供以下更新和技术支持：

- 实例或集群的严重和高 CVE 的安全更新
- 针对关键问题的错误修复和补丁
- 能够在标准 Amazon DocumentDB 服务水平协议范围内建立支持案例并获得故障排除帮助

这项付费服务让您有更多时间升级到支持的引擎版本。例如，Amazon DocumentDB 3.6 版本的标准支持终止日期为 2026 年 3 月 30 日，但是您尚未准备好在此日期之前升级到 5.0 或 8.0 版。在这种情况下，亚马逊文档数据库将于 2026 年 3 月 31 日将您的 3.6 版集群注册到扩展支持中，以便您可以继续使用亚马逊 DocumentDB 版本 3.6。从 2026 年 7 月 1 日起，对亚马逊 DocumentDB 3.6 版集群收取扩展支持高级 AWS 费用。

在引擎版本标准支持终止日期后，Amazon DocumentDB 扩展支持可提供长达 3 年的服务。此时间之后，如果您尚未将引擎版本升级到支持的版本，那么 Amazon DocumentDB 将升级您的引擎版本。建议您尽快升级到支持的引擎版本。

Amazon DocumentDB 版本 3.6 标准支持终止日期和扩展支持日期：

- 版本 3.6 发布日期：2019 年 1 月 9 日
- 版本 3.6 标准支持终止日期：2026 年 3 月 30 日
- 3.6 版 Extended Support 第一年起定价 — 2026 年 7 月 1 日
- 版本 3.6 扩展支持第 3 年定价起始日期：2028 年 3 月 31 日
- 版本 3.6 扩展支持终止日期：2029 年 3 月 30 日

扩展支持概述

如果您的集群尚未升级至最近发布的引擎版本，Amazon DocumentDB 会在扩展支持终止日期前将您的集群升级至该版本。升级期将在该引擎版本标准支持终止日期后开始。

如果集群的引擎版本已达到标准支持终止日期，则可以使用该引擎版本创建新集群。Amazon DocumentDB 会将这些新集群注册到 Amazon DocumentDB 扩展支持中，并向您收取此项服务的费用。如果您在版本 3.6 的标准支持终止日期前升级到仍处于 Amazon DocumentDB 标准支持期的引擎版本，则无需支付扩展支持附加费。

您可以随时终止 Amazon DocumentDB 扩展支持的注册。要终止注册，请将每个已注册的集群升级到仍处于 Amazon DocumentDB 标准支持期的更高引擎版本。Amazon DocumentDB 扩展支持注册的终止将在您完成升级到仍处于 Amazon DocumentDB 标准支持期的更高引擎版本的当天生效。

有关 Amazon DocumentDB 标准支持终止日期和 Amazon DocumentDB 扩展支持终止日期的更多信息，请参阅 Amazon DocumentDB 引擎版本支持日期。

扩展支持费用

从 2026 年 7 月 1 日起，所有在 Extended Support 上运行的集群都将产生费用。有关 Amazon DocumentDB 版本 3.6 标准支持终止日期，请参阅 Amazon DocumentDB 引擎版本支持日期。

当您执行以下操作之一时，Amazon DocumentDB 扩展支持的额外费用将停止：

- 升级到标准支持涵盖范围内的引擎版本
- 删除在标准支持终止日期后运行版本 3.6 的集群

例如，亚马逊 DocumentDB 3.6 版本于 2026 年 3 月 31 日进入扩展支持，但要到 2026 年 7 月 1 日才开始收费。如果您在 2026 年 7 月 31 日将亚马逊 DocumentDB 3.6 版集群升级到亚马逊 DocumentDB 版本 5.0，则只需为亚马逊 DocumentDB 3.6 版本的 30 天延长支持付费。有关更多信息，请参阅 Amazon DocumentDB 定价。

责任

以下是 Amazon DocumentDB 扩展支持方面 Amazon DocumentDB 与您的责任划分。

Amazon DocumentDB 的责任

在 Amazon DocumentDB 版本 3.6 标准支持终止日期之后，Amazon DocumentDB 将为注册扩展支持的版本 3.6 集群应用补丁、错误修复和升级。

你的责任

您负责在 Amazon DocumentDB 版本 3.6 扩展支持终止日期之前，将引擎升级到更高的引擎版本。如果您不升级集群，则在 Amazon DocumentDB 版本 3.6 扩展支持终止日期之后，Amazon DocumentDB 会尝试将您的集群升级到 Amazon DocumentDB 标准支持范围内所支持的最新引擎版本。

Amazon DocumentDB 中的安全性

云安全 AWS 是重中之重。作为 AWS 客户，您可以从专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构中受益。

安全是双方共同承担 AWS 的责任。该文档帮助您了解如何在使用 Amazon DocumentDB 时应用责任共担模式。责任共担模式将其描述为云的 安全性和云中 的安全性：

- 云安全 — AWS 负责保护在 AWS 云中运行 AWS 服务的基础架构。 AWS 还为您提供可以安全使用的服务。 Third-party 作为AWS 合规计划的一部分，审计师定期测试和验证我们安全的有效性。要了解适用于 Amazon DocumentDB（与 MongoDB 兼容）的合规性计划，请参阅AWS 合规性计划范围内的服务。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您组织的要求以及适用的法律法规。

Amazon DocumentDB 已获得联邦风险与授权管理计划 (FedRAMP) 的授权。它拥有（美国）地区的 FedRAMP 高级授权， AWS GovCloud 对美国地区有 FedRAMP 中级授权。 AWS East/West 有关合规工作的详细信息 AWS 以及合规工作，请参阅合规性计划范围内的AWS 服务。

> **Note**
>
> 本章既适用于基于实例的集群，也适用于弹性集群。有关更多信息，请参阅以下主题。

您还将学习如何使用其他 AWS 服务来帮助您监控和保护您的 Amazon DocumentDB 资源。以下主题说明如何配置 Amazon DocumentDB 以实现您的安全性和合规性目标。

主题

- 使用 Amazon DocumentDB 进行密码管理以及 AWS Secrets Manager
- Amazon DocumentDB 中的数据保护
- 适用于 Amazon DocumentDB 的 Identity and Access Management
- Cross-service 混乱的副手预防
- 使用 IAM 身份进行身份验证
- 管理 Amazon DocumentDB 用户
- 使用访问控制 Role-Based 访问数据库

- Amazon DocumentDB 中的日志记录和监控
- 更新您的 Amazon DocumentDB TLS 证书
- Amazon DocumentDB 中的合规性验证
- Amazon DocumentDB 中的故障恢复能力
- Amazon DocumentDB 中的基础设施安全性
- 亚马逊 DocumentDB API 和接口 VPC 终端节点 (AWS PrivateLink)
- Amazon DocumentDB 的安全最佳实践
- 审核 Amazon DocumentDB 事件
- Amazon VPC 和 Amazon DocumentDB

使用 Amazon DocumentDB 进行密码管理以及 AWS Secrets Manager

Amazon DocumentDB 与 Secrets Manager 集成，可以管理集群的主用户密码。

主题

- Secrets Manager 与 Amazon DocumentDB 集成的限制
- 使用管理主用户密码的概述 AWS Secrets Manager
- 强制执行 Amazon DocumentDB 对主用户密码的管理 AWS Secrets Manager
- 使用 Secrets Manager 管理集群的主用户密码

Secrets Manager 与 Amazon DocumentDB 集成的限制

以下功能不支持使用 Secrets Manager 管理主用户密码：

- 属于 Amazon DocumentDB 全局数据库的集群。
- Amazon DocumentDB 跨区域只读副本

使用管理主用户密码的概述 AWS Secrets Manager

使用 AWS Secrets Manager，您可以将代码中的硬编码凭证（包括数据库密码）替换为对 Secrets Manager 的 API 调用，以编程方式检索密钥。有关 Secrets Manager 的更多信息，请参阅《AWS

Secrets Manager 用户指南》https://docs.aws.amazon.com/secretsmanager/latest/userguide/intro.html。

当你在 Secrets Manager 中存储数据库密钥时，你的 AWS 账户会产生费用。有关定价的信息，请参阅 AWS Secrets Manager 定价。

在执行以下操作之一时，您可以指定 Amazon DocumentDB 在 Secrets Manager 中管理 Amazon DocumentDB 集群的主用户密码：

- 创建集群
- 修改集群

当您指定 Amazon DocumentDB 在 Secrets Manager 中管理主用户密码时，Amazon DocumentDB 会生成密码并将其存储在 Secrets Manager 中。您可以直接与密钥交互以检索主用户的凭证。您还可以指定客户托管密钥来加密密钥，或者使用 Secrets Manager 提供的 KMS 密钥。

Amazon DocumentDB 管理密钥的设置，默认情况下每七天轮换一次密钥。您可以修改某些设置，例如轮换计划。如果您删除在 Secrets Manager 中管理密钥的集群，则该密钥及其关联的元数据也会被删除。

要使用密钥中的凭证连接到集群，您可以从 Secrets Manager 检索密钥。有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的 “使用密钥中的凭据从 JDBC 获取AWS Secrets Manager 密钥 AWS Secrets Manager和 Connect 到 SQL 数据库”。

强制执行 Amazon DocumentDB 对主用户密码的管理 AWS Secrets Manager

您可以使用 IAM 条件密钥强制 Amazon DocumentDB 在 AWS Secrets Manager中管理主用户密码。除非主用户密码由 Amazon DocumentDB 在 Secrets Manager 中进行管理，否则以下策略不允许用户创建或恢复实例或集群。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
```

```
            "Action": [
                "rds:CreateDBCluster"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "rds:ManageMasterUserPassword": false
                }
            }
        }
    ]
}
```

使用 Secrets Manager 管理集群的主用户密码

执行以下操作时，可以配置 Amazon DocumentDB 在 Secrets Manager 中管理主用户密码：

- 创建 Amazon DocumentDB 集群
- 修改 Amazon DocumentDB 集群

您可以使用 Amazon DocumentDB 控制台或 AWS CLI 来执行这些操作。

Amazon DocumentDB 中的数据保护

责任 AWS 共担模式分适用于亚马逊 DocumentDB 中的数据保护。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础结构上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题解答AWS条款。 有关欧洲数据保护的信息，请参阅通用数据保护条例（GDPR）中心。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 用于 SSL/TLS 与 AWS 资源通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。有关使用 CloudTrail 跟踪捕获 AWS 活动的信息，请参阅《AWS CloudTrail 用户指南》中的使用跟 CloudTrail 踪。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。

- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-3 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-3 版》https://aws.amazon.com/compliance/fips/。

强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括您使用控制台、API 或软件开发工具包 AWS 服务 使用 Amazon DocumentDB 或其他软件开发工具包 AWS CLI的情况。 AWS 在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供 URL，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

主题

- Client-side 字段级加密
- Amazon DocumentDB 静态数据加密
- 加密传输中数据
- 密钥管理

Client-side 字段级加密

Amazon DocumentDB 客户端字段级加密 (FLE) 允许您在客户端应用程序中的敏感数据传输到 Amazon DocumentDB 集群之前对其进行加密。敏感数据在集群中存储和处理时保持加密状态，检索时在客户端应用程序中解密。

主题

- 开始使用
- 在客户端文件中查询

开始使用

Amazon DocumentDB 中客户端 FLE 的初始配置分为四个步骤，包括创建加密密钥、将角色与应用程序关联、配置应用程序以及使用加密选项定义 CRUD 操作。

主题

- 步骤 1：创建加密密钥

- 步骤 2：将角色与应用程序关联
- 步骤 3：配置应用程序
- 步骤 4：定义 CRUD 操作
- 示例：客户端字段级加密配置文件

步骤 1：创建加密密钥

使用 AWS Key Management Service，创建用于加密和解密敏感数据字段的对称密钥，并为其提供必要的 IAM 使用权限。 AWS KMS 存储用于加密数据密钥 (DK) 的客户密钥 (CK)。将客户密钥存储在 KMS 中以增强您的安全状况。数据密钥是存储在 Amazon DocumentDB 集合中的辅助密钥，在将文档存储到 Amazon DocumentDB 之前，需要使用数据密钥对敏感字段进行加密。客户密钥对数据密钥进行加密，而数据密钥反过来会加密和解密您的数据。如果使用的是全局集群，则可以创建一个多区域密钥，供不同区域的不同服务角色使用。

有关（包括如何创建密钥）的更多信息，请参阅密AWS 钥管理服务开发人员指南。 AWS Key Management Service

步骤 2：将角色与应用程序关联

创建具有适当 AWS KMS 权限的 IAM 策略。此策略允许附加到其上的 IAM 身份获取加密和解密资源字段中指定的 KMS 密钥。您的应用程序使用此 IAM 角色进行身份验证 AWS KMS。

策略应如下所示：

```
{ "Effect": "Allow",
"Action": ["kms:Decrypt", "kms:Encrypt"],
"Resource": "Customer Key ARN"
}
```

步骤 3：配置应用程序

现在，您在中定义了客户密钥 AWS KMS 并创建了一个 IAM 角色，并为其提供了访问客户密钥的正确 IAM 权限。导入所需的软件包。

```
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. 指定 “aws” 作为 KMS 提供商类型，然后输入在上一步中检索到的账户证书。

```
provider = "aws"
kms_providers = {
    provider: {
        "accessKeyId": current_credentials.access_key,
        "secretAccessKey": current_credentials.secret_key
    }
}
```

2. 指定用于加密数据密钥的客户密钥：

```
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. 配置 MongoClient 对象：

```
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. 生成您的数据密钥：

```
data_key_id = client_encryption.create_data_key(provider,
    customer_key,
    key_alt_name = [key_alt_name])
```

5. 检索现有的数据密钥：

```
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})
```

步骤 4：定义 CRUD 操作

使用加密选项定义 CRUD 操作。

1. 将集合定义为 write/read /delete 单个文档：

```
coll = client.gameinfo.users
```

2. 显式加密：加密字段并插入：

> **Note**
>
> 必须只提供“key_id”或“key_alt_name”中的一个。

```
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"]
})
```

示例：客户端字段级加密配置文件

在以下示例中，将每个 *placeholder* 替换为您自己的信息。

```
# 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()
```

在客户端文件中查询

Amazon DocumentDB 仅支持使用客户端 FLE 加密值进行相等查询。

例如，要查询加密玩家分数等于 500 的文档，客户端使用显式加密方法对查询值进行加密：

```
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 静态数据加密

> **Note**
>
> AWS KMS 正在将 “客户主密钥 (CMK)” 一词替换为 “AWS KMS key和 KMS 密钥”。这一概念并未改变。为了防止重大更改，AWS KMS 保留了该术语的一些变体。

您可以通过在创建集群时指定存储加密选项来在 Amazon DocumentDB 集群中加密静态数据。存储加密在整个集群范围内启用，应用于所有实例，包括主实例和任何副本。它还应用于集群的存储卷、数据、索引、日志、自动备份和快照。

Amazon DocumentDB 使用 256 位高级加密标准 (AES-256)，使用存储在 () 中的加密密钥来加密您的数据。 AWS Key Management Service AWS KMS使用静态加密启用的 Amazon DocumentDB 集群时，您无需修改应用程序逻辑或客户端连接。Amazon DocumentDB 以透明方式处理数据的加密和解密，这对性能产生的影响最小。

Amazon DocumentDB 集成 AWS KMS 并使用一种称为信封加密的方法来保护您的数据。当使用 Amazon DocumentDB 集群进行加密时 AWS KMS，Amazon DocumentDB AWS KMS 会要求使用您的 KMS 密钥生成用于加密存储卷的密文数据密钥。加密文字数据密钥使用您定义的 KMS 密钥进行加密，并与加密的数据和存储元数据一起存储。当 Amazon DocumentDB 需要访问您的加密数据时，它会请求 AWS KMS 使用您的 KMS 密钥解密密文数据密钥，并将明文数据密钥缓存在内存中，以高效地加密和解密存储卷中的数据。

Amazon DocumentDB 中的存储加密功能适用于所有支持的实例大小以及所有可用 Amazon DocumentDB AWS 区域 的地方。

为 Amazon DocumentDB 集群启用静态加密

当使用或 () 配置集群时，您可以在 Amazon DocumentDB 集群上启用 AWS 管理控制台 或禁用静态加密。 AWS Command Line Interface AWS CLI默认情况下，您使用控制台创建的集群启用了静态加密功能。默认情况下，您使用创建的 AWS CLI 集群会禁用静态加密。因此，您必须使用 `--storage-encrypted` 参数显式启用静态加密。无论哪种情况，在创建集群后，都无法更改静态加密选项。

Amazon DocumentDB 用于检索和管理加密密钥，并定义控制如何使用 AWS KMS 这些密钥的策略。如果您未指定 AWS KMS 密钥标识符，Amazon DocumentDB 将使用默认的 AWS 托管服务 KMS 密钥。Amazon DocumentDB 会为你的每个 KMS 密钥创建一个单独的 KMS 密钥 AWS 区域 。 AWS 账户有关更多信息，请参阅 AWS Key Management Service 概念。

要开始创建自己的 KMS 密钥，请参阅AWS Key Management Service 开发人员指南中的入门。

> **Important**
>
> 您必须使用对称加密 KMS 密钥加密您的集群，因为 Amazon DocumentDB 仅支持对称加密 KMS 密钥。请勿使用非对称 KMS 密钥尝试对 Amazon DocumentDB 集群中的数据进行加密。有关更多信息，请参阅AWS Key Management Service 开发人员指南中的非对称 KMS 密钥 AWS KMS。

如果 Amazon DocumentDB 无法再访问集群的 KMS 密钥（例如，当拥有 AWS 账户 该密钥的被暂停、密钥被禁用、密钥计划删除，或者 Amazon DocumentDB 所依赖的密钥策略或授权被移除时），则集群将首先转换为状态。`inaccessible-encryption-credentials-recoverable`当集群处于此状态时，Amazon DocumentDB 会停止集群的实例，并且您无法读取或写入集群，但是如果在 7 天内恢复对 KMS 密钥的访问权限，集群仍然可以恢复。如果在 7 天内未恢复访问权限，则集群将转换为终端`inaccessible-encryption-credentials`状态。从终端状态来看，集群不再可用，数据库的当前状态也无法恢复 — 您只能从备份中恢复或使用原始 KMS 密钥执行时间点还原。对于 Amazon DocumentDB，备份始终启用至少 1 天。

> **Note**
>
> 作为全局群集一部分的群集的行为有所不同。当 Amazon DocumentDB 检测到无法再访问 KMS 密钥时，全局集群中的所有集群都将直接转换为终端`inaccessible-encryption-credentials`状态，跳过可恢复状态。这是因为作为全局群集一部分的群集只有在它是全局群集中唯一的集群时才能停止和启动。要恢复，必须从快照恢复或执行时间点恢复。要删除原始群集，必须先从全局群集中移除每个群集，然后再将其删除。

> **Important**
>
> 在已创建加密集群的 KMS 密钥后，您无法更改它。请确保先确定您的加密密钥要求，然后再创建加密的集群。

Using the AWS 管理控制台

您可在创建集群时，指定静态加密选项。默认情况下，当您使用 AWS 管理控制台创建集群时，静态加密处于启用状态。集群创建之后无法修改该选项。

在创建集群时指定静态加密选项

1. 如入门部分所述创建 Amazon DocumentDB 集群。但在步骤 6 中，不要选择创建集群。
2. 在 Authentication (身份验证) 部分下，选择 Show advanced settings (显示高级设置)。
3. 向下滚动到 Encryption-at-rest部分。
4. 选择要进行静态加密的选项。无论您选择哪个选项，都无法在创建集群后更改它。
   - 要对此集群中的静态数据进行加密，请选择启用加密。
   - 如果您不想对此集群中的静态数据进行加密，请选择Disable encryption (禁用加密)。
5. 选择您想要的主密钥。Amazon DocumentDB 使用 AWS Key Management Service (AWS KMS) 来检索和管理加密密钥，并定义控制如何使用这些密钥的策略。如果您未指定 AWS KMS 密钥标识符，Amazon DocumentDB 将使用默认的 AWS 托管服务 KMS 密钥。有关更多信息，请参阅 AWS Key Management Service 概念。

   > **Note**
   >
   > 创建加密的集群后，您无法更改该集群的 KMS 密钥。请确保先确定您的加密密钥要求，然后再创建加密的集群。

6. 根据需要完成其他部分，然后创建您的集群。

Using the AWS CLI

要使用加密 Amazon DocumentDB 集群 AWS CLI，请运行`create-db-cluster`命令并指定选项。`--storage-encrypted`默认情况下，使用创建的 Amazon DocumentDB 集群 AWS CLI 不启用存储加密。

以下示例创建启用了存储加密的 Amazon DocumentDB 集群。

在以下示例中，将每个*`user input placeholder`*示例替换为集群的信息。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb create-db-cluster \
  --db-cluster-identifier mydocdbcluster \
  --port 27017 \
```

```
  --engine docdb \
  --master-username SampleUser1 \
  --master-user-password primaryPassword \
  --storage-encrypted
```

对于 Windows：

```
aws docdb create-db-cluster ^
  --db-cluster-identifier SampleUser1 ^
  --port 27017 ^
  --engine docdb ^
  --master-username SampleUser1 ^
  --master-user-password primaryPassword ^
  --storage-encrypted
```

在创建加密的 Amazon DocumentDB 集群时，您可以指定 AWS KMS 密钥标识符，如以下示例所示。

Example

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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**
>
> 创建加密的集群后，您无法更改该集群的 KMS 密钥。请确保先确定您的加密密钥要求，然后再创建加密的集群。

解析处于无法访问加密状态的 Amazon DocumentDB 集群

当 Amazon DocumentDB 无法访问集群加密时使用的 KMS 密钥时，Amazon DocumentDB 集群将变为无法访问的加密状态。有两种这样的状态，恢复路径取决于集群所处的状态。

##############

当集群处于`inaccessible-encryption-credentials-recoverable`状态时，您可以`available`通过恢复 Amazon DocumentDB 对 KMS 密钥的访问权限，然后启动集群来将集群恢复到。要解决此状态，请执行以下操作：

1. AWS 账户 确认拥有 KMS 密钥的处于活动状态。如果账户被暂停，请重新激活。
2. 确认已启用 KMS 密钥。有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的启用和禁用密钥。
3. 检查是否计划删除 KMS 密钥。如果是，请取消计划中的密钥删除。有关更多信息，请参阅《AWS Key Management Service 开发者指南》中的计划和取消密钥删除。
4. 确认 KMS 密钥策略和亚马逊 DocumentDB 所依赖的任何授权仍然允许亚马逊 DocumentDB 使用该密钥。
5. 恢复对 KMS 密钥的访问后，使用 AWS 管理控制台 或运行`start-db-cluster` AWS CLI 命令启动集群。

   Example

   对于 Linux、macOS 或 Unix：

   ```
   aws docdb start-db-cluster \
     --db-cluster-identifier example-cluster
   ```

   对于 Windows：

   ```
   aws docdb start-db-cluster ^
     --db-cluster-identifier example-cluster
   ```

> **Important**
>
> 如果在 7 天内未恢复对 KMS 密钥的访问权限，则集群将转换为无法启动的终端`inaccessible-encryption-credentials`状态。

###########

`inaccessible-encryption-credentials`状态为终端。集群无法启动，数据库的运行状态也无法恢复。要恢复数据，请从快照还原或对新集群执行时间点还原。您仍必须有权访问原始 KMS 密钥才能执行恢复。如果 KMS 密钥已删除，则无法恢复数据。

有关更多信息，请参阅从集群快照还原和还原到某个时间点。

> **Note**
>
> 作为全局群集一部分的集群会直接转换到失去对 KMS 密钥的访问权限时的`inaccessible-encryption-credentials`状态，因为作为全局群集一部分的集群只有在全局集群中唯一的集群时才能停止和启动。要删除处于此状态的群集，请先从全局群集中移除每个群集，然后分别删除这些群集。

如果由于启用了删除保护而无法删除处于`inaccessible-encryption-credentials`状态的集群，AWS CLI 请在重试删除之前使用关闭删除保护。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster \
  --db-cluster-identifier example-cluster \
  --no-deletion-protection
```

对于 Windows：

```
aws docdb modify-db-cluster ^
  --db-cluster-identifier example-cluster ^
  --no-deletion-protection
```

然后，您可以使用`delete-db-cluster`命令删除集群。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb delete-db-cluster \
  --db-cluster-identifier example-cluster \
  --skip-final-snapshot
```

对于 Windows：

```
aws docdb delete-db-cluster ^
  --db-cluster-identifier example-cluster ^
  --skip-final-snapshot
```

如果运行上述命令后集群仍未删除，请联系 Su AWS pp ort。

Amazon DocumentDB 加密集群的限制

Amazon DocumentDB 加密的集群存在以下限制。

- 您只能在创建 Amazon DocumentDB 集群时而不能在创建它之后启用或禁用静态加密。但是，您可以通过创建未加密群集的快照，然后将未加密的快照还原为新集群，同时指定静态加密选项。

  有关更多信息，请参阅以下主题：

  - 创建手动集群快照
  - 从集群快照还原
  - 复制 Amazon DocumentDB 集群快照
- 不能通过对已启用存储加密的 Amazon DocumentDB 集群进行修改来禁用加密。
- Amazon DocumentDB 集群中的所有实例、自动备份、快照和索引都使用相同的 KMS 密钥进行加密。

加密传输中数据

您可以使用传输层安全性协议（TLS）加密应用程序与 Amazon DocumentDB 集群之间的连接。默认情况下，为新创建的 Amazon DocumentDB 集群启用传输中加密。可以选择在创建集群时或稍后禁用它。启用传输中加密后，需要使用 TLS 进行安全连接才能连接到集群。有关使用 TLS 连接到 Amazon DocumentDB 的更多信息，请参阅 以编程方式连接到 Amazon DocumentDB。

管理 Amazon DocumentDB 集群 TLS 设置

Amazon DocumentDB 集群的传输中加密通过集群参数组中的 TLS 参数进行管理。您可以使用 AWS 管理控制台 或 AWS Command Line Interface ()AWS CLI管理您的 Amazon DocumentDB 集群 TLS 设置。有关如何验证和修改当前 TLS 设置的信息，请参阅以下部分。

Using the AWS 管理控制台

请遵照以下步骤，使用控制台执行 TLS 加密的管理任务，例如识别参数组、验证 TLS 值以及进行必要的修改。

> **Note**
>
> 除非您在创建集群时以其他方式指定，否则将使用默认集群参数组创建集群。`default`集群参数组中的参数无法修改（例如，`tls` enabled/disabled）。因此，如果您的集群使用的是 `default` 集群参数组，则需要修改集群以使用非默认集群参数组。首先，您可能需要创建自定义集群参数组。有关更多信息，请参阅 创建 Amazon DocumentDB 集群参数组。

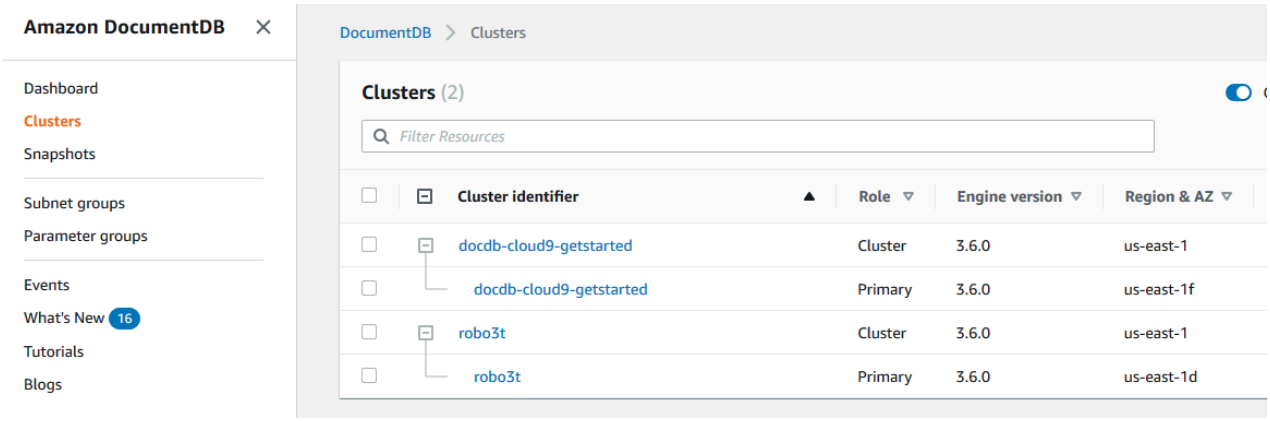
1. 确定集群使用的集群参数组

   a. 打开亚马逊文档数据库控制台，网址为。https://console.aws.amazon.com/docdb

   b. 在导航窗格中，选择集群。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

   c. 请注意，在集群导航框中，集群标识符列既显示集群又显示实例。实例列在集群下方。请参阅以下屏幕截图以供参考。



d. 选择您感兴趣的集群。

e. 选择配置选项卡，向下滚动到集群详细信息底部，然后找到集群参数组。请注意集群参数组的名称。

如果集群参数组的名称是 `default`（例如，`default.docdb3.6`），则您必须创建一个自定义集群参数组，并将其设置为集群的参数组，然后才能继续。有关更多信息，请参阅下列内容：

1. 创建 Amazon DocumentDB 集群参数组：如果没有可以使用的自定义集群参数组，请创建一个。
2. 修改 Amazon DocumentDB 集群：修改您的集群以使用自定义集群参数组。

2. 确定 **`tls`** 集群参数的当前值。

a. 打开亚马逊文档数据库控制台，网址为。https://console.aws.amazon.com/docdb

b. 在导航窗格中，选择参数组。

c. 在集群参数组列表中，选择您感兴趣的集群参数组的名称。

d. 找到群集参数部分。在集群参数列表中，找到 `tls` 集群参数行。此时，以下四列很重要：

- Cluster parameter name (集群参数名称)：集群参数的名称。对于管理 TLS，需要关注的是 `tls` 集群参数。
- Values (值)：每个集群参数的当前值。
- Allowed values (允许的值)：可应用到集群参数的值的列表。
- Apply type (应用类型)：static (静态)或 dynamic (动态)。对静态集群参数的更改只能在重启实例时应用。对动态集群参数的更改可以立即应用，也可以在重启实例时应用。

3. 修改 **`tls`** 集群参数的值。

如果 tls 的值不是所需的，请为此集群参数组修改其值。要更改 tls 集群参数的值，请按照以下步骤从前一部分继续执行。

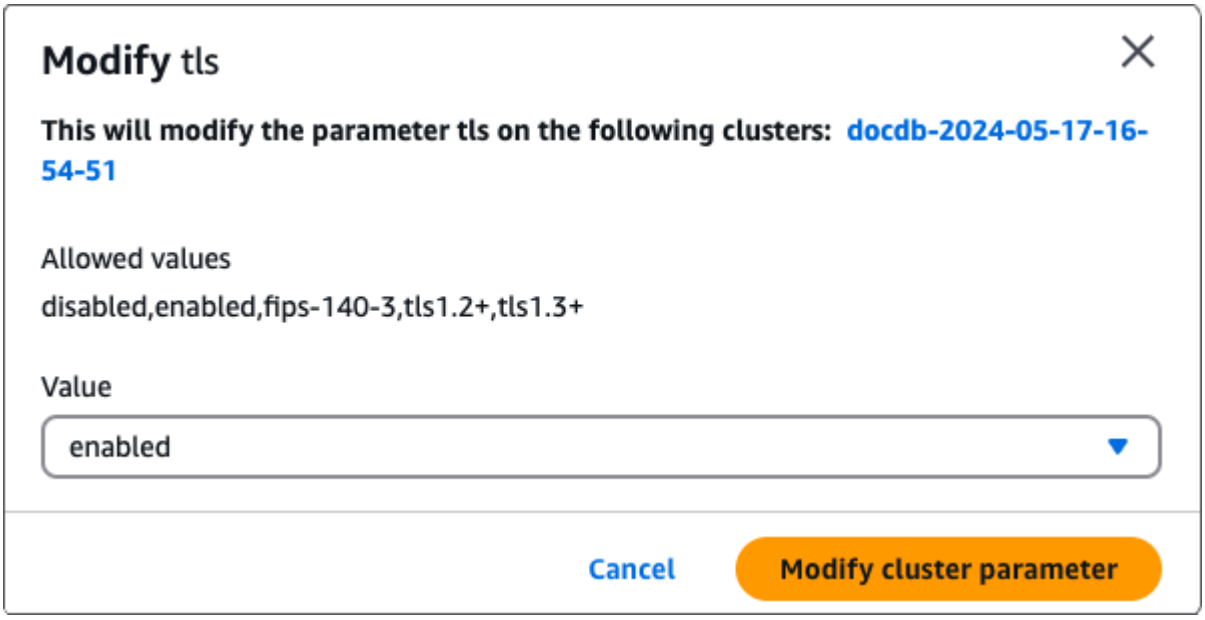
a. 选择集群参数名称 (tls) 左侧的按钮。

b. 选择编辑。

c. 要更改 tls 的值，请在 Modify（修改）**tls** 对话框中，从下拉列表中为此集群参数选择所需的值。

   有效值为：

   - 已禁用：禁用 TLS
   - 启用-启用 TLS。对于亚马逊 DocumentDB 5.0 及更早版本，这将启用 TLS 1.0 到 1.3，但我们建议使用 TLS 1.2 或更高版本。从次要版本 5.0.1、亚马逊 DocumentDB 8.0 及更高版本开始，亚马逊 DocumentDB 5.0 及更高版本仅支持 TLS 1.2 及更高版本。
   - fips-140-3：使用 FIPS 启用 TLS。集群仅接受符合联邦信息处理标准 (FIPS) 出版物 140-3 要求的安全连接。在以下区域，这是唯一受支持的 Amazon DocumentDB 5.0（引擎版本 3.0.3727）集群启用：ca-central-1、us-west-2、us-east-1、us-east-2、us-gov-east-1、us-gov-west-1。
   - tls1.2+：启用 TLS 版本 1.2 及更高版本。从 Amazon DocumentDB 4.0（引擎版本 2.0.10980）和 Amazon DocumentDB（引擎版本 3.0.11051）开始才支持此功能。
   - tls1.3+：启用 TLS 版本 1.3 及更高版本。从 Amazon DocumentDB 4.0（引擎版本 2.0.10980）和 Amazon DocumentDB（引擎版本 3.0.11051）开始才支持此功能。



d. 选择 Modify cluster parameter (修改集群参数)。更改在重启时应用到每个集群实例。

4. 重启 Amazon DocumentDB 实例

   重启集群的每个实例，以便将更改应用到集群中的所有实例。

   a. 打开亚马逊文档数据库控制台，网址为。https://console.aws.amazon.com/docdb

   b. 在导航窗格中，选择 Instances (实例)。

   c. 要指定待重启的实例，请在实例列表中找到实例，并选择该实例名称左侧的按钮。

   d. 选择 Actions（操作），然后选择 Reboot（重启）。通过选择重启来确认您要重启。

Using the AWS CLI

请遵照以下步骤，使用 AWS CLI执行 TLS 加密的管理任务，例如识别参数组、验证 TLS 值以及进行必要的修改。

> **Note**
>
> 除非您在创建集群时以其他方式指定，否则将使用默认集群参数组创建集群。`default`集群参数组中的参数无法修改（例如，`tls` enabled/disabled）。因此，如果您的集群使用的是 `default` 集群参数组，则需要修改集群以使用非默认集群参数组。您可能需要先创建自定义集群参数组。有关更多信息，请参阅 创建 Amazon DocumentDB 集群参数组。

1. 确定集群使用的集群参数组

   运行具有以下选项的 describe-db-clusters 命令：

   - --db-cluster-identifier
   - --query

   在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
aws docdb describe-db-clusters \
  --db-cluster-identifier mydocdbcluster \
  --query 'DBClusters[*].[DBClusterIdentifier,DBClusterParameterGroup]'
```

   此操作的输出将类似如下（JSON 格式）：

```
[
    [
        "mydocdbcluster",
        "myparametergroup"
    ]
]
```

   如果集群参数组的名称是 default（例如，default.docdb3.6），则您必须有一个自定义集群参数组，并将其设置为集群的参数组，然后才能继续。有关更多信息，请参阅以下主题：

   1. 创建 Amazon DocumentDB 集群参数组：如果没有可以使用的自定义集群参数组，请创建一个。
   2. 修改 Amazon DocumentDB 集群：修改您的集群以使用自定义集群参数组。

2. 确定 **tls** 集群参数的当前值。

   要获取有关此集群参数组的更多信息，请运行具有以下选项的 describe-db-cluster-parameters 命令：

   - --db-cluster-parameter-group-name
   - --query

     将输出限制为仅包含感兴趣的字段：ParameterName、ParameterValue、AllowedValues 和 ApplyType。

在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
aws docdb describe-db-cluster-parameters \
  --db-cluster-parameter-group-name myparametergroup \
  --query 'Parameters[*].[ParameterName,ParameterValue,AllowedValues,ApplyType]'
```

此操作的输出将类似如下（JSON 格式）：

```
[
    [
        "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. 修改 **tls** 集群参数的值。

如果 tls 的值不是所需的，则为此集群参数组修改其值。要更改 tls 集群参数的值，请运行具有以下选项的 modify-db-cluster-parameter-group 命令：

- --db-cluster-parameter-group-name：必需。要修改的集群参数组的名称。不能是 default.* 集群参数组。
- --parameters：必需。要修改的集群参数组的参数列表。
  - ParameterName：必需。要修改的集群参数的名称。

- ParameterValue：必需。此集群参数的新值。必须是集群参数的 AllowedValues 之一。
  - enabled— 集群接受使用 TLS 的安全连接。对于亚马逊 DocumentDB 5.0 及更早版本，这将启用 TLS 1.0 到 1.3，但我们建议使用 TLS 1.2 或更高版本。从次要版本 5.0.1、亚马逊 DocumentDB 8.0 及更高版本开始，亚马逊 DocumentDB 5.0 及更高版本仅支持 TLS 1.2 及更高版本。
  - disabled：此集群不接受使用 TLS 的安全连接。
  - fips-140-3：集群仅接受符合联邦信息处理标准 (FIPS) 出版物 140-3 要求的安全连接。仅在以下区域支持 Amazon DocumentDB 5.0（引擎版本 3.0.3727）集群启用：ca-central-1、us-west-2、us-east-1、us-east-2、us-gov-east-1、us-gov-west-1。
  - tls1.2+：集群接受使用 TLS 版本 1.2 及更高版本的安全连接。从 Amazon DocumentDB 4.0（引擎版本 2.0.10980）和 Amazon DocumentDB 5.0（引擎版本 3.0.11051）开始才支持此功能。
  - tls1.3+：集群接受使用 TLS 版本 1.3 及更高版本的安全连接。从 Amazon DocumentDB 4.0（引擎版本 2.0.10980）和 Amazon DocumentDB 5.0（引擎版本 3.0.11051）开始才支持此功能。
- ApplyMethod：何时应用此修改。对于静态集群参数（如 tle），此值必须为 pending-reboot。
  - pending-reboot：仅在重启后才将更改应用到实例。您必须分别重启每个集群实例，以便此更改在所有集群实例上生效。

在以下示例中，将每个*user input placeholder*示例替换为集群的信息。

以下代码禁用 tls，在重启时将更改应用到每个实例。

```
aws docdb modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=tls,ParameterValue=disabled,ApplyMethod=pending-
reboot"
```

以下代码启用tls，在每个实例重启时将更改应用于每个实例。

```
aws docdb modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
```

```
  --parameters "ParameterName=tls,ParameterValue=enabled,ApplyMethod=pending-
reboot"
```

以下代码启用带 fips-140-3 的 TLS，在重启时将更改应用到每个实例。

```
aws docdb modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup2 \
  --parameters "ParameterName=tls,ParameterValue=fips-140-3,ApplyMethod=pending-
reboot"
```

此操作的输出将类似如下（JSON 格式）：

```
{
    "DBClusterParameterGroupName": "myparametergroup"
}
```

4. 重启 Amazon DocumentDB 实例

重启集群的每个实例，以便将更改应用到集群中的所有实例。要重启 Amazon DocumentDB 实例，请运行具有以下选项的 reboot-db-instance 命令：

- --db-instance-identifier

以下代码将重启实例 mydocdbinstance。

在以下示例中，将每个*user input placeholder*示例替换为集群的信息。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb reboot-db-instance \
  --db-instance-identifier mydocdbinstance
```

对于 Windows：

```
aws docdb reboot-db-instance ^
  --db-instance-identifier mydocdbinstance
```

此操作的输出将类似如下（JSON 格式）：

```
{
    "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"
    }
```

```
}
```

实例重启需要几分钟时间。只有在实例状态为 available (可用) 时，才能使用实例。您可以使用控制台或 AWS CLI监控实例状态。有关更多信息，请参阅 监控 Amazon DocumentDB 实例的状态。

密钥管理

Amazon DocumentDB 使用 AWS Key Management Service (AWS KMS) 来检索和管理加密密钥。AWS KMS 将安全、高度可用的硬件和软件相结合，提供可扩展到云端的密钥管理系统。使用 AWS KMS，您可以创建加密密钥并定义控制如何使用这些密钥的策略。 AWS KMS 支持 AWS CloudTrail，因此您可以审核密钥使用情况，以验证密钥的使用是否正确。

您的 AWS KMS 密钥可以与亚马逊文档数据库和支持的 AWS 服务结合使用，例如亚马逊简单存储服务 (Amazon S3)、亚马逊关系数据库服务 (Amazon RDS)、亚马逊弹性区块存储 (Amazon EBS) Elastic Block Store 和 Amazon Redshift。有关支持的服务列表 AWS KMS，请参阅《AWS Key Management Service 开发人员指南》 AWS KMS中的AWS 服务使用方式。有关的信息 AWS KMS，请参阅什么是 AWS Key Management Service？

适用于 Amazon DocumentDB 的 Identity and Access Management

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以通过身份验证（登录）和获得授权（具有权限）来使用 Amazon DocumentDB 资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

- 受众
- 使用身份进行身份验证
- 使用策略管理访问
- Amazon DocumentDB 如何配合 IAM 工作
- Identity-based 亚马逊 DocumentDB 的策略示例
- Amazon DocumentDB 身份和访问问题排查
- 管理对 Amazon DocumentDB 资源的访问权限
- 将基于身份的策略（IAM 策略）用于 Amazon DocumentDB

- AWS 亚马逊 DocumentDB 的托管政策
- Amazon DocumentDB API 权限：操作、资源和条件参考

受众

您的使用方式 AWS Identity and Access Management (IAM) 因您的角色而异：

- 服务用户：如果您无法访问功能，请从管理员处请求权限（请参阅Amazon DocumentDB 身份和访问问题排查）
- 服务管理员：确定用户访问权限并提交权限请求（请参阅Amazon DocumentDB 如何配合 IAM 工作）
- IAM 管理员：编写用于管理访问权限的策略（请参阅Identity-based 亚马逊 DocumentDB 的策略示例）

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份进行身份验证 AWS 账户根用户，或者通过担任 IAM 角色进行身份验证。

您可以使用来自身份源的证书 AWS IAM Identity Center（例如（IAM Identity Center）、单点登录身份验证或 Google/Facebook 证书，以联合身份登录。有关登录的更多信息，请参阅《AWS 登录 用户指南》中的如何登录您的 AWS 账户。

对于编程访问，AWS 提供 SDK 和 CLI 来对请求进行加密签名。有关更多信息，请参阅《IAM 用户指南》中的适用于 API 请求的AWS 签名版本 4。

AWS 账户 根用户

创建时 AWS 账户，首先会有一个名为 AWS 账户 root 用户的登录身份，该身份可以完全访问所有资源 AWS 服务 和资源。我们强烈建议不要使用根用户进行日常任务。有关需要根用户凭证的任务，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户使用与身份提供商的联合身份验证才能 AWS 服务 使用临时证书进行访问。

联合身份是指来自您的企业目录、Web 身份提供商的用户 Directory Service ，或者 AWS 服务 使用来自身份源的凭据进行访问的用户。联合身份代入可提供临时凭证的角色。

要集中管理访问权限，建议使用。 AWS IAM Identity Center有关更多信息，请参阅《AWS IAM Identity Center 用户指南》中的什么是 IAM Identity Center？。

IAM 用户和群组

IAM 用户是对某个人员或应用程序具有特定权限的一个身份。建议使用临时凭证，而非具有长期凭证的 IAM 用户。有关更多信息，请参阅 IAM 用户指南中的要求人类用户使用身份提供商的联合身份验证才能 AWS 使用临时证书进行访问。

IAM 组指定一组 IAM 用户，便于更轻松地对大量用户进行权限管理。有关更多信息，请参阅《IAM 用户指南》中的 IAM 用户使用案例。

IAM 角色

IAM 角色是具有特定权限的身份，可提供临时凭证。您可以通过从用户切换到 IAM 角色（控制台）或调用 AWS CLI 或 AWS API 操作来代入角色。有关更多信息，请参阅《IAM 用户指南》中的担任角色的方法。

IAM 角色对于联合用户访问、临时 IAM 用户权限、跨账户访问、跨服务访问以及在 Amazon EC2 上运行的应用程序非常有用。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略定义了与身份或资源关联时的权限。 AWS 在委托人提出请求时评估这些政策。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概述。

管理员使用策略，通过定义哪个主体可以在什么条件下对哪些资源执行哪些操作来指定谁有权访问什么。

默认情况下，用户和角色没有权限。IAM 管理员创建 IAM 策略并将其添加到角色中，然后用户可以担任这些角色。IAM 策略定义权限，与执行操作所用的方法无关。

Identity-based 政策

Identity-based 策略是您附加到身份（用户、组或角色）的 JSON 权限策略文档。这些策略控制身份可以执行什么操作、对哪些资源执行以及在什么条件下执行。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

Identity-based 策略可以是内联策略（直接嵌入到单个身份中）或托管策略（附加到多个身份的独立策略）。要了解如何在托管策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管策略与内联策略之间进行选择。

Resource-based 政策

Resource-based 策略是您附加到资源的 JSON 策略文档。示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。您必须在基于资源的策略中指定主体。

Resource-based 策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

其他策略类型

AWS 支持其他策略类型，这些策略类型可以设置更常见的策略类型授予的最大权限：

- 权限边界 – 设置基于身份的策略可以授予 IAM 实体的最大权限。有关更多信息，请参阅《 IAM 用户指南》中的 IAM 实体的权限边界。
- 服务控制策略（ SCP ）– 指定 AWS Organizations中组织或组织单元的最大权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。
- 资源控制策略（ RCP ）– 设置对账户中资源的最大可用权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的资源控制策略（ RCP ）。
- 会话策略 – 在为角色或联合用户创建临时会话时，作为参数传递的高级策略。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

Amazon DocumentDB 如何配合 IAM 工作

在使用 IAM 管理对 Amazon DocumentDB 的访问权限之前，您应该了解哪些 IAM 功能可用于 Amazon DocumentDB。

您可以配合 Amazon DocumentDB 使用的 IAM 功能

IAM 功能	Instance-based 集群	弹性集群
Identity-based 政策	支持	是

IAM 功能	Instance-based 集群	弹性集群
Resource-based 政策	否	否
策略操作	支持	是
策略资源	支持	是
策略条件键（特定于服务）	支持	是
ACL	否	否
ABAC（策略中的标签）	部分	是
临时凭证	支持	是
主体权限	支持	是
服务角色	支持	是
Service-linked 角色	否	是

要全面了解 Amazon DocumentDB 和其他 AWS 服务如何与大多数 IAM 功能配合使用，请参阅 IAM 用户指南中与 IAM 配合使用的AWS 服务。

Identity-based 亚马逊 DocumentDB 的政策

支持基于身份的策略：是

Identity-based 策略是您可以附加到身份（例如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

Identity-based 亚马逊 DocumentDB 的策略示例

要查看 Amazon DocumentDB 基于身份的策略示例，请参阅 Identity-based 亚马逊 DocumentDB 的策略示例。

Resource-based 亚马逊 DocumentDB 中的政策

支持基于资源的策略：否

Resource-based 策略是您附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户访问，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

Amazon DocumentDB 的策略操作

支持策略操作：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。在策略中包含操作以授予执行关联操作的权限。

> **Note**
>
> 对于某些管理功能，Amazon DocumentDB 使用与 Amazon Relational Database Service (Amazon RDS) 共享的操作技术。
> 要查看 RDS 操作的列表，请参阅《服务授权参考》中 Amazon Relational Database Service 定义的操作。
> 要查看 Amazon DocumentDB 弹性集群的策略操作，请参阅《服务授权参考》中的 Amazon DocumentDB 弹性集群定义的操作。

Amazon DocumentDB 中的策略操作在操作前面使用以下前缀：

```
aws
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
```

```
      "aws:action1",
      "aws:action2"
         ]
```

要查看 Amazon DocumentDB 基于身份的策略示例，请参阅 Identity-based 亚马逊 DocumentDB 的策略示例。

Amazon DocumentDB 的策略资源

支持策略资源：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

`Resource` JSON 策略元素指定要向其应用操作的一个或多个对象。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于不支持资源级权限的操作，请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

> **Note**
>
> 对于某些管理功能，Amazon DocumentDB 使用与 Amazon Relational Database Service (Amazon RDS) 共享的操作技术。
> 要查看 RDS 资源类型及其 ARN 的列表，请参阅《服务授权参考》中 Amazon Relational Database Service 定义的资源。要了解您可以用哪些操作指定每种资源的 ARN，请参阅 Amazon Relational Database Service 定义的操作。
> 要查看 Amazon DocumentDB 弹性集群的资源类型，请参阅《服务授权参考》中的 Amazon DocumentDB 弹性集群定义的资源类型。

要查看 Amazon DocumentDB 基于身份的策略示例，请参阅 Identity-based 亚马逊 DocumentDB 的策略示例。

Amazon DocumentDB 的策略条件密钥

支持特定于服务的策略条件键：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

`Condition` 元素根据定义的条件指定语句何时执行。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

> **Note**
>
> 对于某些管理功能，Amazon DocumentDB 使用与 Amazon Relational Database Service (Amazon RDS) 共享的操作技术。
> 要查看 RDS 条件密钥的列表，请参阅服务授权参考中的 Amazon Relational Database Service 的条件密钥。要了解您可以用哪些操作和资源使用条件密钥，请参阅 Amazon Relational Database Service 定义的操作。
> 要查看 Amazon DocumentDB 弹性集群的条件密钥，请参阅《服务授权参考》中 Amazon DocumentDB 弹性集群的条件密钥。

要查看 Amazon DocumentDB 基于身份的策略示例，请参阅 Identity-based 亚马逊 DocumentDB 的策略示例。

Amazon DocumentDB 中的 ACL

支持 ACL：否

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，但它们不使用 JSON 策略文档格式。

配合 Amazon DocumentDB 的 ABAC

> **Note**
>
> 基于实例的集群仅部分支持 ABAC，但弹性集群则支持 ABAC。

Attribute-based 访问控制 (ABAC) 是一种授权策略，它根据称为标签的属性来定义权限。您可以将标签附加到 IAM 实体和 AWS 资源，然后设计 ABAC 策略以允许在委托人的标签与资源上的标签匹配时进行操作。

要基于标签控制访问，您需要使用 aws:ResourceTag/*key-name*aws:RequestTag/*key-name* 或 aws:TagKeys 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息，请参阅《IAM 用户指南》中的使用 ABAC 授权定义权限。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

配合 Amazon DocumentDB 使用临时凭证

支持临时凭证：是

临时证书提供对 AWS 资源的短期访问权限，并且是在您使用联合身份或切换角色时自动创建的。AWS 建议您动态生成临时证书，而不是使用长期访问密钥。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的临时安全凭证和使用 IAM 的。AWS 服务

Cross-service Amazon DocumentDB 的主要权限

支持转发访问会话（FAS）：是

转发访问会话 (FAS) 使用调用主体的权限 AWS 服务，再加上 AWS 服务 向下游服务发出请求的请求。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

Amazon DocumentDB 的服务角色

支持服务角色：是

服务角色是由一项服务担任、代表您执行操作的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

> **Warning**
>
> 更改服务角色的权限可能破坏 Amazon DocumentDB 功能。仅 Amazon DocumentDB 提供如此行事的指导时才编辑服务角色。

Service-linked 亚马逊 DocumentDB 的角色

> **Note**
>
> Service-linked 基于实例的集群不支持角色，但弹性集群支持角色。

服务相关角色是一种与服务相关联的 AWS 服务服务角色。该服务可以代替您执行操作。 Service-linked 角色出现在您的， AWS 账户 并且归服务所有。IAM 管理员可以查看但不能编辑服务关联角色的权限。

有关创建或管理服务相关角色的详细信息，请参阅能够与 IAM 搭配使用的AWS 服务。在表中查找Service-linked 角色列Yes中包含的服务。选择是链接以查看该服务的服务相关角色文档。

Identity-based 亚马逊 DocumentDB 的策略示例

默认情况下，用户和角色没有创建或修改 Amazon DocumentDB 资源的权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略（控制台）。

有关 Amazon DocumentDB 定义的操作和资源类型的详细信息，包括每种资源类型的 ARN 的格式，请参阅《服务授权参考》中的 Amazon Relational Database Service 的操作、资源和条件密钥。

> **Note**
>
> 对于某些管理功能，Amazon DocumentDB 使用与 Amazon Relational Database Service (Amazon RDS) 共享的操作技术。
> 有关 Amazon DocumentDB 弹性集群的策略操作，请参阅《服务授权参考》中的 Amazon DocumentDB 弹性集群的操作、资源和条件键。

主题

- 策略最佳实践
- 使用 Amazon DocumentDB 控制台
- 允许用户查看他们自己的权限

策略最佳实践

Identity-based 策略决定了是否有人可以在您的账户中创建、访问或删除亚马逊文档数据库资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的AWS 托管策略或工作职能的AWS 托管策略。
- 应用最低权限：在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限：您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定的方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性：IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言（JSON）和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM Access Analyzer 验证策略。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的使用 MFA 保护 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

使用 Amazon DocumentDB 控制台

要访问 Amazon DocumentDB（与 MongoDB 兼容）控制台，您必须具有一组最低的权限。这些权限必须允许您列出和查看有关您的 Amazon DocumentDB 资源的详细信息。 AWS 账户如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与其尝试执行的 API 操作相匹配的操作。

为确保用户和角色仍然可以使用亚马逊文档数据库控制台，还需要将亚马逊*ConsoleAccess*文档数据库*ReadOnly* AWS 或托管策略附加到实体。有关更多信息，请参阅 IAM 用户指南中的为用户添加权限。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
    "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 身份和访问问题排查

使用以下信息帮助您诊断和修复在使用 Amazon DocumentDB 和 IAM 时可能遇到的常见问题。

主题

- 我无权在 Amazon DocumentDB 中执行操作
- 我无权执行 iam：PassRole
- 我想允许我以外的人进入 AWS 账户 访问我的亚马逊 DocumentDB 资源

我无权在 Amazon DocumentDB 中执行操作

如果您收到错误提示，指明您无权执行某个操作，则必须更新策略以允许执行该操作。

当 `mateojackson` IAM 用户尝试使用控制台查看有关虚构 *`my-example-widget`* 资源的详细信息，但不拥有虚构 `aws:`*`GetWidget`* 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 aws:GetWidget on resource: my-example-widget
```

在此情况下，必须更新 `mateojackson` 用户的策略，以允许使用 `aws:`*`GetWidget`* 操作访问 *`my-example-widget`* 资源。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam：PassRole

如果您收到一个错误：您无权执行 `iam:PassRole` 操作，则必须更新您的策略以允许您将角色传递给 Amazon DocumentDB。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或与服务相关的角色。为此，您必须具有将角色传递到服务的权限。

当名为 `marymajor` 的 IAM 用户尝试使用控制台在 Amazon DocumentDB 中执行操作时，以下示例错误出现。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 `iam:PassRole` 操作。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我想允许我以外的人进入 AWS 账户 访问我的亚马逊 DocumentDB 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 Amazon DocumentDB 是否支持这些功能，请参阅 Amazon DocumentDB 如何配合 IAM 工作。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

管理对 Amazon DocumentDB 资源的访问权限

每个 AWS 资源都归人所有 AWS 账户，创建或访问资源的权限受权限策略的约束。账户管理员可以向 IAM 身份（即用户、群组和角色）附加权限策略，某些服务（例如 AWS Lambda）还支持向资源附加权限策略。

> **Note**
>
> 账户管理员 (或管理员用户) 是具有管理员权限的用户。有关更多信息，请参阅《IAM 用户指南》中的 IAM 最佳实践。

主题

- Amazon DocumentDB 资源和操作

- 了解资源所有权
- 管理对 资源的访问
- 指定策略元素：操作、效果、资源和主体
- 在策略中指定条件

Amazon DocumentDB 资源和操作

在 Amazon DocumentDB 中，主要资源是集群。Amazon DocumentDB 支持可配合主要资源一起使用的其他资源，如实例、参数组和事件订阅。这些资源称作子资源。

这些资源和子资源具有与其关联的唯一 Amazon 资源名称（ARN，如下表所示。

资源类型	ARN 格式
Cluster	`arn:aws:rds:` *region*`:`*account-id* `:cluster:` *db-cluster-name*
集群参数组	`arn:aws:rds:` *region*`:`*account-id* `:cluster-pg:` *cluster-parameter-group-name*
集群快照	`arn:aws:rds:` *region*`:`*account-id* `:cluster-snapshot:` *cluster-snapshot-name*
实例	`arn:aws:rds:` *region*`:`*account-id* `:db:`*db-instance-name*
安全组	`arn:aws:rds:` *region*`:`*account-id* `:secgrp:`*security-group-name*
子网组	`arn:aws:rds:` *region*`:`*account-id* `:subgrp:`*subnet-group-name*

Amazon DocumentDB 提供一组操作来处理 Amazon DocumentDB 资源。有关可用操作的列表，请参阅操作。

了解资源所有权

资源所有者 AWS 账户 是创建资源的人。也就是说，资源所有者是 AWS 账户 对创建资源的请求进行身份验证的委托人实体（根账户、IAM 用户或 IAM 角色）。以下示例说明了它的工作原理：

- 如果您使用您的 AWS 账户 根账户证书创建亚马逊文档数据库资源（例如实例），则您 AWS 账户 就是亚马逊文档数据库资源的所有者。
- 如果您在中创建 IAM 用户 AWS 账户 并向该用户授予创建 Amazon DocumentDB 资源的权限，则该用户可以创建 Amazon DocumentDB 资源。但是，用户所属的您 AWS 账户拥有亚马逊 DocumentDB 资源。
- 如果您在中创建 AWS 账户 具有创建 Amazon DocumentDB 资源的权限的 IAM 角色，则任何能够担任该角色的人都可以创建 Amazon DocumentDB 资源。该角色所属的您 AWS 账户拥有亚马逊 DocumentDB 资源。

管理对 资源的访问

权限策略规定谁可以访问哪些内容。下一节介绍创建权限策略时的可用选项。

> **Note**
>
> 本节讨论在 Amazon DocumentDB 背景下使用 IAM。这里不提供有关 IAM 服务的详细信息。有关完整的 IAM 文档，请参阅《IAM 用户指南》中的什么是 IAM？。有关 IAM 策略语法和说明的信息，请参阅《IAM 用户指南》中的 AWS IAM 策略参考。

附加到 IAM 身份的策略称作基于身份的策略 (IAM policy)。附加到资源的策略称作基于资源的 策略。Amazon DocumentDB 只支持基于身份的策略（IAM 策略）。

主题

- Identity-based 策略（IAM 策略）
- Resource-based 政策

Identity-based 策略（IAM 策略）

您可以向 IAM 身份附加策略。例如，您可以执行以下操作：

- 将权限策略附加到您账户中的用户或组：账户管理员可以使用与特定用户关联的权限策略授予该用户创建 Amazon DocumentDB 资源（如实例）的权限。
- 向角色附加权限策略（授予跨账户权限）：您可以向 IAM 角色附加基于身份的权限策略，以授予跨账户的权限。例如，管理员可以创建一个角色来向其他人 AWS 账户 或 AWS 服务授予跨账户权限，如下所示：

1. 账户 A 管理员可以创建一个 IAM 角色，然后向该角色附加授予其访问账户 A 中资源的权限策略。
2. 账户 A 管理员可以把信任策略附加至用来标识账户 B 的角色，账户 B 由此可以作为主体代入该角色。
3. 然后，账户 B 管理员可以将代入该角色的权限委托给账户 B 中的任何用户。这样，账户 B 中的用户就可以创建或访问账户 A 中的资源。如果您想向 AWS 服务授予担任该角色的权限，则信任策略中的委托人也可以是 AWS 服务委托人。

有关使用 IAM 委托权限的更多信息，请参阅《IAM 用户指南》中的访问权限管理。

以下是允许 ID 为 123456789012 的用户为您的 AWS 账户创建实例的示例策略。新实例必须使用以 default 开头的选项组和参数组，并且它必须使用 default 子网组。

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"
            ]
        }
    ]
}
```

有关配合 Amazon DocumentDB 使用基于身份的策略的更多信息，请参阅 将基于身份的策略（IAM 策略）用于 Amazon DocumentDB。有关用户、组、角色和权限的更多信息，请参阅《IAM 用户指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/id.html中的身份（用户、组和角色）。

Resource-based 政策

其他服务如 Amazon Simple Storage Service (Amazon S3) 支持基于资源的权限策略。例如，您可以将策略附加到 Amazon S3 存储桶以管理对该存储桶的访问权限。Amazon DocumentDB 不支持基于资源的策略。

指定策略元素：操作、效果、资源和主体

对于每种 Amazon DocumentDB 资源（请参阅 Amazon DocumentDB 资源和操作），该服务都定义一组 API 操作。有关更多信息，请参阅操作。要对这些 API 操作授予权限，Amazon DocumentDB 定义了一组您可以在策略中指定的操作。执行一个 API 操作可能需要多个操作的权限。

以下是基本的策略元素：

- 资源：在策略中，您可以使用 Amazon 资源名称（ARN）标识策略应用到的资源。
- 操作：您可以使用操作关键字标识要允许或拒绝的资源操作。例如，`rds:DescribeDBInstances` 权限允许用户执行 `DescribeDBInstances` 操作。
- 效果：您可以指定当用户请求特定操作（可以是允许或拒绝）时的效果。如果没有显式授予（允许）对资源的访问权限，则隐式拒绝访问。您也可显式拒绝对资源的访问，这样可确保用户无法访问该资源，即使有其他策略授予了访问权限的情况下也是如此。
- 主体：在基于身份的策略（IAM 策略）中，附加了策略的用户是隐式主体。对于基于资源的策略，您可以指定要接收权限的用户、账户、服务或其他实体（仅适用于基于资源的策略）。Amazon DocumentDB 不支持基于资源的策略。

有关 IAM 策略语法和描述的更多信息，请参阅《IAM 用户指南》中的 AWS IAM 策略参考。

有关显示所有 Amazon DocumentDB API 操作及它们适用的资源的表，请参阅 Amazon DocumentDB API 权限：操作、资源和条件参考。

在策略中指定条件

当您授予权限时，可使用 IAM 策略语言来指定规定策略何时生效的条件。例如，您可能希望策略仅在特定日期后应用。有关使用策略语言指定条件的更多信息，请参阅《IAM 用户指南》中的条件。

要表示条件，您可以使用预定义的条件键。Amazon DocumentDB 没有可以在 IAM policy略中使用的服务专属上下文密钥。有关可用于所有服务的全局条件上下文密钥列表，请参阅 IAM 用户指南中的可提供条件密钥。

将基于身份的策略（IAM 策略）用于 Amazon DocumentDB

> **Important**
>
> 对于某些管理功能，Amazon DocumentDB 使用与 Amazon RDS 共享的操作技术。亚马逊 DocumentDB 控制台和 API 调用记录为对亚马逊 RDS API 的调用。 AWS CLI
> 我们建议您首先回顾以下介绍性主题，这些主题讲解了您可用于管理 Amazon DocumentDB 资源访问的基本概念和选项。有关更多信息，请参阅 管理对 Amazon DocumentDB 资源的访问权限。

本主题提供了基于身份的策略的示例，在这些策略中，账户管理员可以向 IAM 身份（即：用户、组和角色）附加权限策略。

以下是 IAM policy略示例。

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"
            ]
        }
    ]
}
```

该策略包含一个为 IAM 用户指定以下权限的语句：

- 该策略允许 IAM 用户使用 CreatedBin Stance 操作创建实例（这也适用于创建数据库实例操作和）。 AWS CLI AWS 管理控制台
- Resource 元素指定用户可以执行操作的资源。使用 Amazon 资源名称 (ARN) 指定资源。此 ARN 包括资源所属服务的名称 (rds)、 AWS 区域 （ *表示本示例中的任何区域）、用户账号（在本示例中123456789012为用户 ID）和资源类型。

  该示例中的 Resource 元素为用户指定有关资源的以下策略限制：

  - 新实例的实例标识符必须以 test 开头（例如，testCustomerData1、test-region2-data）。
  - 新实例的集群参数组必须以 default 开头。
  - 新实例的子网组必须是 default 子网组。

该策略不指定 Principal 元素，因为在基于身份的策略中，您未指定获取权限的委托人。附加了策略的用户是隐式委托人。向 IAM 角色附加权限策略后，该角色的信任策略中标识的主体将获取权限。

有关显示所有 Amazon DocumentDB API 操作及其适用资源的表，请参阅 Amazon DocumentDB API 权限：操作、资源和条件参考。

使用 Amazon DocumentDB 控制台所要求的权限

对于要使用 Amazon DocumentDB 控制台的用户，该用户必须拥有一组最小权限。这些权限允许用户描述他们的 Amazon DocumentDB 资源 AWS 账户 并提供其他相关信息，包括 Amazon EC2 安全和网络信息。

如果创建比必需的最低权限更为严格的 IAM 策略，对于附加了该 IAM 策略的用户， 控制台将无法按预期正常运行。要确保这些用户仍可使用 Amazon DocumentDB 控制台，也可向用户附加 AmazonDocDBConsoleFullAccess 托管策略，如 AWS 亚马逊 DocumentDB 的托管政策 中所述。

对于仅调用 AWS CLI 或 Amazon DocumentDB API 的用户，您无需为其设置最低控制台权限。

客户管理型策略示例

本节的用户策略示例介绍如何授予各 Amazon DocumentDB 操作的权限。当您使用 Amazon DocumentDB API 操作、 AWS 软件开发工具包或。 AWS CLI当您使用控制台时，您需要授予特定于控制台的其他权限，使用 Amazon DocumentDB 控制台所要求的权限 中对此进行了讨论。

对于某些管理功能，Amazon DocumentDB 使用与 Amazon Relational Database Service (Amazon RDS) 和 Amazon Neptune 共享的操作技术。

> **Note**
>
> 所有示例都使用美国东部（弗吉尼亚州北部）区域 (us-east-1) 并且包含虚构的账户 ID。

示例

- 示例 1：允许用户对任何 Amazon DocumentDB 资源执行任何描述操作
- 示例 2：防止用户删除实例
- 示例 3：除非启用存储加密，否则阻止用户创建集群

示例 1：允许用户对任何 Amazon DocumentDB 资源执行任何描述操作

以下权限策略对用户授予权限以运行以 `Describe` 开头的所有操作。这些操作显示有关 Amazon DocumentDB 资源（如实例）的信息。 `Resource` 组件中的通配符 (*) 表示对该账户拥有的所有 Amazon DocumentDB 资源允许这些操作。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowRDSDescribe",
         "Effect":"Allow",
         "Action":"rds:Describe*",
         "Resource":"*"
      }
   ]
}
```

示例 2：防止用户删除实例

以下权限策略授予权限以防止用户删除特定实例。例如，您可能想禁止任何非管理员用户删除您的生产实例。

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"
      }
   ]
}
```

示例 3：除非启用存储加密，否则阻止用户创建集群

除非启用存储加密，否则以下权限策略拒绝用户创建 Amazon DocumentDB 集群的权限。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "PreventUnencryptedDocumentDB",
         "Effect": "Deny",
         "Action": "RDS:CreateDBCluster",
         "Condition": {
         "Bool": {
         "rds:StorageEncrypted": "false"
      },
         "StringEquals": {
         "rds:DatabaseEngine": "docdb"
         }
      },
      "Resource": "*"
      }
   ]
}
```

AWS 亚马逊 DocumentDB 的托管政策

要向用户、群组和角色添加权限，使用 AWS 托管策略比自己编写策略要容易得多。创建仅为团队提供所需权限的 IAM 客户管理型策略需要时间和专业知识。要快速入门，您可以使用我们的 AWS 托管策略。这些政策涵盖常见用例，可在您的 AWS 账户中使用。有关 AWS 托管策略的更多信息，请参阅《Identity and A ccess Managem AWS ent 用户指南》中的AWS 托管策略。

AWS 服务维护和更新 AWS 托管策略。您无法更改 AWS 托管策略中的权限。服务偶尔会向 AWS 托管策略添加其他权限以支持新功能。此类更新会影响附加策略的所有身份（用户、组和角色）。当推出新功能或有新操作可用时，服务最有可能更新 AWS 托管策略。服务不会从 AWS 托管策略中移除权限，因此策略更新不会破坏您的现有权限。

此外，还 AWS 支持跨多个服务的工作职能的托管策略。例如，`ViewOnlyAccess` AWS 托管策略提供对许多 AWS 服务和资源的只读访问权限。当服务启动一项新功能时， AWS 会为新操作和资源添加只读权限。有关工作职能策略的列表和说明，请参阅 AWS IAM 用户指南中的用于工作职能的AWS 托管策略。

以下 AWS 托管策略仅适用于 Amazon DocumentDB，您可以将其附加到账户中的用户：

- AmazonDocDBFullAccess— 授予根账户对所有 Amazon DocumentDB 资源的完全访问权限 AWS 。
- AmazonDocDBReadOnlyAccess— 授予根账户对所有 Amazon DocumentDB 资源的只读访问权限 AWS 。
- AmazonDocDBConsoleFullAccess：授予使用 AWS 管理控制台管理 Amazon DocumentDB 和 Amazon DocumentDB 弹性集群资源的完全访问权限。
- AmazonDocDBElasticReadOnlyAccess— 授予根账户对所有 Amazon DocumentDB 弹性集群资源的只读访问权限 AWS 。
- AmazonDocDBElasticFullAccess— 授予根账户对所有 Amazon DocumentDB 弹性集群资源的完全访问权限 AWS 。

AmazonDocDBFullAccess

此策略授予了允许主体完全访问 Amazon DocumentDB 所有 Amazon DocumentDB 操作的管理权限。此策略中的权限如下分组：

- Amazon DocumentDB 权限允许所有Amazon DocumentDB 操作。

- 需要此策略中的一些 Amazon EC2 权限来验证 API 请求中的已传递资源。这旨在确保 Amazon DocumentDB 能够配合集群成功使用资源。此策略中的其他 Amazon EC2 权限允许 Amazon DocumentDB 创建必要的 AWS 资源，使您能够连接到您的集群。
- 在 API 调用期间，Amazon DocumentDB 权限用于验证请求中的已传递资源。Amazon DocumentDB 需要这些资源才能配合 Amazon DocumentDB 集群一起使用传递的密钥。
- Amazon DocumentDB 需要这些 CloudWatch 日志才能确保日志传输目标可达，并且这些日志对于代理日志的使用有效。

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

此策略授予了允许用户查看 Amazon DocumentDB 中信息的只读权限。附加有这种策略的主体不能进行任何更新或删除现有资源，也不能创建新的 Amazon DocumentDB 资源。例如，拥有这些权限的主体可以查看与其账户关联的集群列表和配置，但不能更改任何集群的配置或设置。此策略中的权限如下分组：

- Amazon DocumentDB 权限允许您列出 Amazon DocumentDB 资源，描述它们并获取有关它们的信息。
- Amazon EC2 权限用于描述与某集群关联的 Amazon VPC、子网、安全组和 ENI。
- Amazon DocumentDB 权限用于描述与该集群关联的密钥。

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

授予使用以下方式管理 Amazon DocumentDB 资源的完全访问权限： AWS 管理控制台

- 允许所有 Amazon DocumentDB 和 Amazon DocumentDB 集群操作的 Amazon DocumentDB 权限。
- 需要此策略中的一些 Amazon EC2 权限来验证 API 请求中的已传递资源。这是为了确保 Amazon DocumentDB 能够成功使用资源来准备和维护集群。该策略中的其余亚马逊 EC2 权限允许亚马逊文档数据库创建所需的 AWS 资源，使您能够连接到 vpcendPoint 等集群。
- AWS KMS 在 API 调用期间，权限 AWS KMS 用于验证请求中传递的资源。Amazon DocumentDB 需要它们才能配合 Amazon DocumentDB 弹性集群使用已传递的密钥加密和解密静态数据。
- Amazon DocumentDB 需要这些 CloudWatch 日志才能确保日志传输目标可达，并且这些日志对于审计和分析日志的使用有效。
- 需要 Secrets Manager 权限来验证给定机密并使用它为 Amazon DocumentDB 弹性集群设置管理员用户。
- Amazon DocumentDB 集群管理操作需要 Amazon RDS 权限。对于某些管理功能，Amazon DocumentDB 使用与 Amazon RDS 共享的操作技术。
- SNS 允许主体访问 Amazon Simple Notification Service (Amazon SNS) 订阅和主题及发布 Amazon DocumentDB 消息。
- 创建为发布指标和日志所需的服务关联角色需要 IAM 权限。

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

此策略授予了允许用户查看 Amazon DocumentDB 中弹性集群信息的只读权限。附加有这种策略的主体不能进行任何更新或删除现有资源，也不能创建新的 Amazon DocumentDB 资源。例如，拥有这些权限的主体可以查看与其账户关联的集群列表和配置，但不能更改任何集群的配置或设置。此策略中的权限如下分组：

- Amazon DocumentDB 弹性集群权限允许您列出 Amazon DocumentDB 弹性集群资源，描述它们并获取有关它们的信息。
- CloudWatch 权限用于验证服务指标。

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

此策略授予了允许主体完全访问针对 Amazon DocumentDB 弹性集群的所有 Amazon DocumentDB 操作的管理权限。

此策略使用条件内的 AWS 标签 (https://docs.aws.amazon.com/tag-editor/latest/userguide/tagging.html) 来限制对资源的访问权限。如果您将要使用机密，则必须将它用标签密钥 `DocDBElasticFullAccess` 和标签值标记。如果您将要使用客户托管的密钥，则必须将它用标签密钥 `DocDBElasticFullAccess` 和标签值标记。

此策略中的权限如下分组：

- Amazon DocumentDB 弹性集群权限允许所有 Amazon DocumentDB 操作。
- 需要此策略中的一些 Amazon EC2 权限来验证 API 请求中的已传递资源。这是为了确保 Amazon DocumentDB 能够成功使用资源来准备和维护集群。此策略中的其他 Amazon EC2 权限允许 Amazon DocumentDB 创建所需的 AWS 资源，使您能够像 VPC 终端节点一样连接到您的集群。
- AWS KMS Amazon DocumentDB 需要权限才能使用传递的密钥对亚马逊文档数据库弹性集群中的静态数据进行加密和解密。

  > **Note**
  >
  > 客户托管的密钥必须有一个带密钥 `DocDBElasticFullAccess` 和标签值的标签。

- SecretsManager 需要权限才能验证给定的密钥并使用它为 Amazon DocumentDB 弹性集群设置管理员用户。

  > **Note**
  >
  > 用过的机密必须有一个带密钥 `DocDBElasticFullAccess` 和标签值的标签。

- 创建为发布指标和日志所需的服务关联角色需要 IAM 权限。

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

你无法附着AmazonDocDBElasticServiceRolePolicy在你的 AWS Identity and Access Management 实体上。这种策略附加到允许Amazon DocumentDB 代表您执行操作的服务关联角色。有关更多信息，请参阅 Service-linked 弹性集群中的角色。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": [
                        "AWS/DocDB-Elastic"
                    ]
                }
            }
        }
    ]
}
```

亚马逊 DocumentDB 更新为 AWS 托管策略

更改	描述	日期
AmazonDocDBElasticFullAccess, AmazonDocDBConsoleFullAccess：更改	更新了策略，添加了待处理的维护操作。	2/11/2025
AmazonDocDBElasticFullAccess, AmazonDocDBConsoleFullAccess：更改	更新了策略以添加 start/stop 群集和复制群集快照操作。	2/21/2024
AmazonDocDBElasticReadOnlyAccess, AmazonDocDBElasticFullAccess：更改	策略已更新以增加 `cloudwatch:GetMetricData` 操作。	6/21/2023
AmazonDocDBElasticReadOnlyAccess：新策略	Amazon DocumentDB 弹性集群的新托管策略。	6/8/2023
AmazonDocDBElasticFullAccess：新策略	Amazon DocumentDB 弹性集群的新托管策略。	6/5/2023
AmazonDocDB-ElasticServiceRolePolicy：新策略	亚马逊 DocumentDB 为亚马逊 DocumentDB 弹性集群创建了一个新的 AWS ServiceRoleForDocDB-Elastic 服务关联角色。	11/30/2022
AmazonDocDBConsoleFullAccess：更改	策略已更新，以增加 Amazon DocumentDB 全局权限和弹性集群权限。	11/30/2022
AmazonDocDBConsoleFullAccess、AmazonDocDBFullAccess、AmazonDocDBReadOnlyAccess：新策略	服务启动。	1/19/2017

Amazon DocumentDB API 权限：操作、资源和条件参考

在您设置 将基于身份的策略（IAM 策略）用于 Amazon DocumentDB 并编写您可以附加到 IAM 身份的权限策略（基于身份的策略）时，可以使用以下章节作为参考。

下文列出了每个 Amazon DocumentDB API 操作。列表中包括您可以授予执行操作权限的相应操作、可以授予权限的 AWS 资源以及可以包含的用于精细访问控制的条件密钥。您需要在策略的 `Action` 字段中指定操作、在策略的 `Resource` 字段中指定资源值、在策略的 `Condition` 字段中指定条件。有关条件的更多信息，请参阅“在策略中指定条件”。

您可以在 Amazon DocumentDB 政策中使用 AWS全局条件键来表达条件。有关 AWS范围密钥的完整列表，请参阅 IAM 用户指南中的可用密钥。

您可以使用 IAM policy simulator 测试 IAM 策略 它会自动提供每项操作所需的资源和参数列表，包括 Amazon DocumentDB AWS 操作。IAM policy simulator 确定您指定的每个操作所要求的权限。有关 IAM policy simulator 的信息，请参阅 IAM 用户指南中的用 IAM 策略模拟器测试 IAM 策略。

> **Note**
>
> 要指定操作，请在 API 操作名称之前使用 `rds:` 前缀（例如，`rds:CreateDBInstance`）。

下面列出了 Amazon RDS API 操作及其相关操作、资源和条件密钥。

主题

- 支持资源级权限的 Amazon DocumentDB 操作
- 不支持资源级权限的 Amazon DocumentDB 操作

支持资源级权限的 Amazon DocumentDB 操作

Resource-level 权限允许用户指定允许用户对其执行操作的资源。Amazon DocumentDB 部分支持资源级权限。这意味着对于某些 Amazon DocumentDB 操作，您可以基于须满足的条件或允许用户使用的具体资源，控制何时允许用户使用这些操作。例如，您可以向用户授予仅修改特定实例的权限。

下面列出了 Amazon DocumentDB API 操作及其相关操作、资源和条件密钥。

> **Note**
>
> 对于某些管理功能，Amazon DocumentDB 使用与 Amazon RDS 共享的操作技术。有关 Amazon DocumentDB 操作和权限的更多信息，请参阅《服务授权参考》中的 Amazon RDS 的操作、资源和条件键。

Amazon DocumentDB API 运作和操作	资源	条件键
AddTagsToResource rds:AddTagsToResource	实例 arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
	子网组 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
ApplyPendingMaintenanceAction rds:ApplyPendingMaintenanceAction	实例 arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
CopyDBClusterSnapshot rds:CopyDBClusterSnapshot	集群快照 arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
CreateDBCluster rds:CreateDBCluster	Cluster	rds:cluster-tag

Amazon DocumentDB API 运作和操作	资源	条件键
	arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-name*	
	集群参数组 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
	子网组 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
CreateDBClusterParameterGroup rds:CreateDBClusterParameterGroup	集群参数组 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
	集群快照 arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag

Amazon DocumentDB API 运作和操作	资源	条件键
CreateDBInstance rds:CreateDBInstance	实例 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
CreateDBSubnetGroup rds:CreateDBSubnetGroup	子网组 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
DeleteDBInstance rds:DeleteDBInstance	实例 arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
DeleteDBSubnetGroup rds:DeleteDBSubnetGroup	子网组 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
DescribeDBClusterParameterGroups rds:DescribeDBClusterParameterGroups	集群参数组 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag

Amazon DocumentDB API 运作和操作	资源	条件键
DescribeDBClusterParameters rds:DescribeDBClusterParameters	集群参数组 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
DescribeDBClusters rds:DescribeDBClusters	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
DescribeDBClusterSnapshotAttributes rds:DescribeDBClusterSnapshotAttributes	集群快照 arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
DescribeDBSubnetGroups rds:DescribeDBSubnetGroups	子网组 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
DescribePendingMaintenanceActions rds:DescribePendingMaintenanceActions	实例 arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:DatabaseClass rds:db-tag

Amazon DocumentDB API 运作和操作	资源	条件键
FailoverDBCluster rds:FailoverDBCluster	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
ListTagsForResource rds:ListTagsForResource	实例 arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
	子网组 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
	集群参数组 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag
ModifyDBClusterParameterGroup rds:ModifyDBClusterParameterGroup	集群参数组 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag

Amazon DocumentDB API 运作和操作	资源	条件键
ModifyDBClusterSnapshotAttribute rds:ModifyDBClusterSnapshotAttribute	集群快照 arn:aws:rds: *region*:*account-id* :cluster-snapshot: *cluster-snapshot-name*	rds:cluster-snapshot-tag
ModifyDBInstance rds:ModifyDBInstance	实例 arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:DatabaseClass rds:db-tag
RebootDBInstance rds:RebootDBInstance	实例 arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
RemoveTagsFromResource rds:RemoveTagsFromResource	实例 arn:aws:rds: *region*:*account-id* :db:*db-instance-name*	rds:db-tag
	子网组 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag
ResetDBClusterParameterGroup rds:ResetDBClusterParameterGroup	集群参数组 arn:aws:rds: *region*:*account-id* :cluster-pg: *cluster-parameter-group-name*	rds:cluster-pg-tag

Amazon DocumentDB API 运作和操作	资源	条件键
RestoreDBClusterFromSnapshot rds:RestoreDBClusterFromSnapshot	Cluster arn:aws:rds: *region*:*account-id* :cluster: *db-cluster-instance-name*	rds:cluster-tag
	集群快照 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
	子网组 arn:aws:rds: *region*:*account-id* :subgrp:*subnet-group-name*	rds:subgrp-tag

不支持资源级权限的 Amazon DocumentDB 操作

您可以使用一个 IAM policy 略中的所有 Amazon DocumentDB 操作授予或拒绝用户使用该操作的权限。但是，并非所有 Amazon DocumentDB 操作都支持资源级权限，这使您能够指定可对其执行操作的资源。以下 Amazon DocumentDB API 操作当前不支持资源级权限。因此，要在IAM policy略中使用这些操作，您必须通过对您声明中的 Resource 元素使用 * 通配符，授予用户对该操作使用所有资源的权限。

- rds:DescribeDBClusterSnapshots
- rds:DescribeDBInstances

Cross-service 混乱的副手预防

混淆代理问题是一个安全性问题，即不具有某操作执行权限的实体可能会迫使具有更高权限的实体执行该操作。在中 AWS，跨服务模仿可能会导致混乱的副手问题。 Cross-service 当一个服务（调用服务）调用另一个服务（被调用的服务）时，可能会发生模仿行为。可以操纵调用服务，使用其权限以在其他情况下该服务不应有访问权限的方式对另一个客户的资源进行操作。为防止这种情况， AWS 提供可帮助您保护所有服务的数据的工具，而这些服务中的服务主体有权限访问账户中的资源。

我们建议在资源策略中使用aws:SourceArn和aws:SourceAccount全局条件上下文密钥来限制 Amazon DocumentDB 向该资源提供的其他服务的权限。如果您只希望将一个资源与跨服务访问相关联，请使用。aws:SourceArn如果您想允许该账户中的任何资源与跨服务使用操作相关联，请使用。aws:SourceAccount

防范混淆代理问题最有效的方法是使用 aws:SourceArn 全局条件上下文键和资源的完整 ARN。如果不知道资源的完整 ARN，或者正在指定多个资源，请针对 ARN 未知部分使用带有通配符字符（*）的 aws:SourceArn 全局上下文条件键。例如 arn:aws:*servicename*:*:*123456789012*:*。

如果 aws:SourceArn 值不包含账户 ID，例如 Amazon S3 存储桶 ARN，您必须使用两个全局条件上下文键来限制权限。

的值aws:SourceArn必须是聚类。

以下示例显示了如何在 Amazon DocumentDB 中使用aws:SourceArn和aws:SourceAccount全局条件上下文键来防止出现混淆的副手问题。

```
{
  "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"
      }
    }
  }
}
```

使用 IAM 身份进行身份验证

Amazon DocumentDB 用户和应用程序可以使用 IAM 用户和角色进行身份验证，以便进入 Amazon DocumentDB 集群。Amazon DocumentDB IAM 身份验证是一种无需密码的身份验证方法。此外，使用 IAM 时，客户端应用程序不会将密码密钥发送到 Amazon DocumentDB 集群。 roles/users相反，AWS STS 使用临时安全令牌对客户端连接进行身份验证。 Non-administrative 现在，用户和应用程序在连接到不同的 Amazon DocumentDB 集群和其他服务时可以使用相同的 IAM 身份 ARN。 AWS

您也可以选择同时使用基于密码的身份验证和 IAM 身份验证，对访问 Amazon DocumentDB 集群的用户和应用程序进行身份验证。只有 Amazon DocumentDB 基于实例的集群版本 5.0 才可以使用 IAM 身份验证。Amazon DocumentDB 主用户不支持用 IAM 身份 ARN 进行 IAM 身份验证。

> **Note**
>
> 主用户只能使用现有基于密码的身份验证进行身份验证。

主题

- 开始使用 IAM 用户和角色进行身份验证
- 配置 AWS 使用以下方法计算要向 Amazon DocumentDB 进行身份验证的类型 AWS IAM
- 监控 IAM 身份验证请求
- 使用 IAM 身份验证
- 支持 IAM 的驱动程序
- IAM 身份验证的常见问题解答

开始使用 IAM 用户和角色进行身份验证

具有 IAM 身份的 Amazon DocumentDB 用户和角色在 `$external` 数据库中进行创建和管理。

创建用户

以主用户身份进行连接，然后创建 IAM 用户和角色：

```
use $external;
db.createUser(
    {
        user: "arn:aws:iam::123456789123:user/iamuser",
        mechanisms: ["MONGODB-AWS"],
        roles: [ { role: "readWrite", db: "readWriteDB" } ]
    }
);
```

或者，使用 IAM 角色添加 Amazon DocumentDB 用户：

```
use $external;
db.createUser(
    {
        user: "arn:aws:iam::123456789123:role/iamrole",
        mechanisms: ["MONGODB-AWS"],
        roles: [ { role: "readWrite", db: "readWriteDB" } ]
    }
);
```

修改 IAM 用户或角色

修改现有 IAM 用户：

```
use $external;
db.updateUser(
  "arn:aws:iam::123456789123:user/iamuser",
  {
    roles: [ { role: "read", db: "readDB" } ]
  }
);
```

修改现有 IAM 角色：

```
use $external;
db.updateUser(
  "arn:aws:iam::123456789123:role/iamrole",
  {
    roles: [ { role: "read", db: "readDB" } ]
  }
```

```
);
```

授予或撤消 IAM 用户的角色：

```
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" } ]
);
```

授予或撤消 IAM 角色的角色：

```
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" } ]
);
```

删除 IAM 用户或角色

删除现有 IAM 用户：

```
use $external;
db.dropUser("arn:aws:iam::123456789123:user/iamuser");
```

删除现有 IAM 角色：

```
use $external;
```

```
db.dropUser("arn:aws:iam::123456789123:role/iamrole");
```

配置连接 URI 以使用 AWS IAM 进行身份验证

要使用 AWS IAM 进行身份验证，请使用以下 URI 参数：authSourceas $external 和 authMechanism as MONGODB-AWS。如果使用 IAM 用户，则将用户名和密码字段分别替换为访问密钥和私有密钥。如果代入 IAM 角色，则将该角色附加到您所在的环境（例如， AWS Lambda 函数、Amazon EC2 实例）。利用 MONGODB-AWS 机制进行身份验证时，无需专门传递任何凭证。如果使用支持 MONGODB-AWS 身份验证机制的 MongoDB 驱动程序，这些驱动程序还可以从计算实例（例如 Amazon EC2、Lambda 函数等）检索 IAM 角色凭证。以下示例使用 mongo Shell 进行身份验证，方法是使用 MONGODB-AWS 机制，通过手动传递（IAM 用户的）访问密钥和私有密钥来演示对 Amazon DocumentDB 进行身份验证。

以下示例使用 Python 代码进行身份验证，方法是使用 MONGODB-AWS 机制演示对 Amazon DocumentDB 进行身份验证，无需显式传递任何凭证（使用附加到此环境的 IAM 角色）。

```
##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')
```

以下示例使用 mongo Shell 进行身份验证，方法是使用 MONGODB-AWS 机制，通过手动传递（IAM 用户的）访问密钥和私有密钥来演示对 Amazon DocumentDB 进行身份验证。

```
$ mongo 'mongodb://<access_key>:<secret_key>@<cluster_endpoint>:<db_port>/test?
authSource=%24external&authMechanism=MONGODB-AWS'
```

以下示例使用 mongo Shell 进行身份验证，方法是使用 MONGODB-AWS 机制演示对 Amazon DocumentDB 进行身份验证，无需显式传递任何凭证（使用附加到此环境的 IAM 角色）。

```
$ mongo 'mongodb://<cluster_endpoint>:<db_port>/test?authSource=
%24external&authMechanism=MONGODB-AWS'
```

配置 AWS 使用以下方法计算要向 Amazon DocumentDB 进行身份验证的类型 AWS IAM

使用 Amazon EC2/AWS Lambda/AWS Fargate

Amazon EC2 使用以下环境变量。如果有附加到 EC2 实例的 IAM 角色，或与 Lambda 函数或 Amazon ECS 任务关联的执行 IAM 角色，则这些变量会自动填充，驱动程序可以从环境中获取这些值：

```
AWS_ACCESS_KEY_ID
AWS_SECRET_ACCESS_KEY
AWS_SESSION_TOKEN
```

有关环境变量的更多信息，请参阅《AWS Lambda 开发人员指南》中的使用 Lambda 环境变量。

使用 Amazon EKS

向 Amazon Elastic Kubernetes Service (Amazon EKS) 容器组分配角色会自动设置以下两个环境变量：

```
AWS_WEB_IDENTITY_TOKEN_FILE - path of web identity token file
AWS_ROLE_ARN - Name of IAM role to connect with
```

借助这些变量，使用 AWS SDK 调用手动代入您的代码中的角色`AssumeRoleWithWebIdentity`：

- 省略 `ProviderID` 参数。
- 在 `AWS_WEB_IDENTITY_TOKEN_FILE` 环境变量所述的文件中查找 `WebIdentityToken` 参数的值。

有关更多信息，请参阅《Amazon EKS 用户指南》中的什么是 Amazon EKS。

监控 IAM 身份验证请求

使用 Amazon DocumentDB 审核

前往 Amazon 中的审核日志文件夹 CloudWatch，使用不同的搜索模式获取 IAM 身份验证的日志。例如，用 `{ $.param.mechanism = "MONGODB-AWS" }` 作为“搜索所有日志流” 的搜索模式。

有关审核中所支持事件的更多信息，请参阅 审核 Amazon DocumentDB 事件。

使用亚马逊 CloudWatch 指标

`StsGetCallerIdentityCalls`：该指标显示了 Amazon DocumentDB 实例对区域化 AWS Security Token Service ()AWS STS终端节点的`GetCallerIdentity`调用次数。有关数据库实例需要进行 STS `GetCallerIdentity` 调用的原因，请参阅`MONGODB-AWS`身份验证规范。

使用 IAM 身份验证

如果您不想在自己的数据库中管理用户名和密码，可以使用 IAM 身份验证。只有 Amazon DocumentDB 基于实例的集群版本 5.0 才可以使用 IAM 身份验证。

IAM 身份验证依赖于 STS 服务。在使用 IAM 身份验证进行连接并收到 STS 节流异常时，我们建议您评估是否可以降低连接速率。

有关 IAM 配额的信息，请参阅 IAM 用户指南中的 IAM 和 AWS STS 配额。

支持 IAM 的驱动程序

支持 Amazon DocumentDB 5.0 和 `MONGODB-AWS` 身份验证机制的驱动程序应与 Amazon DocumentDB 中的 IAM 身份验证实现配合使用。

> **Important**
>
> 版本早于 6.13.1 的 Node.js 驱动程序存在已知限制，Amazon DocumentDB 的 IAM 身份验证目前不支持这些限制。 Node.js 使用驱动程序（例如 mongosh）的 Node.js 驱动程序和工具必须升级为使用 Node.js 驱动程序版本 6.13.1 或更高版本。

IAM 身份验证的常见问题解答

是否有我可以参考的示例？

请参阅以下页面了解示例用例和配置：

- 人类用户如何使用 IAM 用户和 IAM 角色对 Amazon DocumentDB 进行身份验证
- Password-less 使用 IAM 角色对亚马逊文档数据库进行身份验证

我在@@ 使用我的 Python 驱动程序时遇到了一个错误：“pymongo.errors。 ConfigurationError: MONGODB-AWS 身份验证需要 pymongo- auth-aws”。我该如何解决这个问题？

安装带有 IAM 身份验证的 Python 驱动程序时，请务必使用以下语句：

```
pip install 'pymongo[aws]'
```

这将安装 IAM 身份验证正常运行所需的其他 AWS 依赖项。

当我的 IAM 角色临时凭证到期时，连接会中断吗？

不会，临时 IAM 凭证仅用于建立连接和身份验证。然后，所有进一步的身份验证都是在 Amazon DocumentDB 集群中完成的。即使有 IAM 证书 rotate/expire，连接也不会中断或失效。

管理 Amazon DocumentDB 用户

在 Amazon DocumentDB 中，用户与密码一起使用才能通过集群的身份验证。每个集群都有在创建集群时建立的主登录凭证。

> **Note**
>
> 所有在 2020 年 3 月 26 日之前创建的新用户都已被授予 `dbAdminAnyDatabase`、`readWriteAnyDatabase` 和 `clusterAdmin` 角色。建议您重新评估所有用户并根据需要修改角色，以便为集群中所有用户强制执行最低权限。
> 有关更多信息，请参阅 使用访问控制 Role-Based 访问数据库。

主用户和#######

新创建的 Amazon DocumentDB 集群有两个用户：主用户和 `serviceadmin` 用户。

主用户是单个特权用户，可以执行管理任务并创建具有角色的其他用户。首次连接到 Amazon DocumentDB 集群时，必须使用主登录凭证进行身份验证。主用户在创建 Amazon DocumentDB 集群时收到该集群的这些管理权限，并被授予 `root` 角色。

在创建集群时会隐式创建 `serviceadmin` 用户。每个 Amazon DocumentDB 集群都有一个能够管理您的集群的`serviceadmin`用户。 AWS 您无法登录、删掉、重命名、更改密码，或者修改 `serviceadmin` 的权限。任何此类尝试都会导致错误。

> **Note**
>
> 无法删除 Amazon DocumentDB 集群的主用户和 `serviceadmin` 用户，也无法撤销主用户的 `root` 角色。
> 如果您忘记了主用户密码，则可以使用 AWS 管理控制台 或将其重置 AWS CLI。

创建其他用户

作为主用户（或具有 `createUser` 角色的任何用户）连接后，您可以创建新用户，如下所示。

```
db.createUser(
    {
        user: "sample-user-1",
        pwd: "password123",
        roles:
            [{"db":"admin", "role":"dbAdminAnyDatabase" }]
    }
)
```

要查看用户的详细信息，可以使用 show users 命令，如下所示。您还可以使用 dropUser 命令删除用户。有关更多信息，请参阅 通用命令。

```
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"
     }
    ]
}
```

在上面的示例中，新用户 sample-user-1 受 admin 数据库的限制。对于新用户来说，情况总是如此。Amazon DocumentDB 没有 authenticationDatabase 的概念，因此所有身份验证都是在 admin 数据库中进行的。

在创建用户时，如果在指定角色时省略 db 字段，则 Amazon DocumentDB 会隐式地将该角色归属于发出连接的数据库。例如，如果您的连接是针对数据库 sample-database 发出的，并且您运行以下命令，则用户 sample-user-2 将在 admin 数据库中创建并具有对数据库 sample-database 的 readWrite 权限。

```
db.createUser(
    {
        user: "sample-user-2",
        pwd: "password123",
        roles:
            ["readWrite"]
    }
)
```

创建具有跨所有数据库范围的角色的用户（例如，readInAnyDatabase）需要在创建用户时处于 admin 数据库上下文，或者在创建用户时明确指定角色的数据库。

要切换数据库上下文，可以使用以下命令。

```
use admin
```

要了解有关基于角色的访问控制以及对集群用户强制执行最低权限的更多信息，请参阅使用访问控制 Role-Based 访问数据库。

自动轮换 Amazon DocumentDB 的密码

使用 AWS Secrets Manager，您可以将代码中的硬编码凭据（包括密码）替换为对 Secrets Manager 的 API 调用，以编程方式检索密钥。这有助于确保检查您的代码的人不会泄露密钥，因为其中根本不

包含密钥。此外，您还可以配置 Secrets Manager 以根据指定的计划自动轮换密钥。这使您能够将长期密钥替换为短期密钥，这有助于显著减少泄露风险。

使用 Secrets Manager，你可以使用 Secrets Manager 提供的 AWS Lambda 功能自动轮换你的 Amazon DocumentDB 密码（即机密）。

有关 AWS Secrets Manager Amazon DocumentDB 原生集成的更多信息，请参阅以下内容：

- 博客：如何在 AWS Secrets Manager 中轮换 Amazon DocumentDB 和 Amazon Redshift 凭证
- 什么是 S AWS ecrets Manager？
- 轮换 AWS Secrets Manager 密钥
- Secrets Manager 中的 Amazon DocumentDB 凭证

使用访问控制 Role-Based 访问数据库

您可以使用 Amazon DocumentDB（与 MongoDB 兼容）中的基于角色的访问控制（RBAC）限制用户可以对数据库执行的操作的访问权限。RBAC 的原理是向用户授予一个或多个角色。这些角色决定了用户可以对数据库资源执行的操作。Amazon DocumentDB 目前既支持限定在数据库级别的内置角色（例如`read`、`readWrite`、`readAnyDatabase`、`clusterAdmin`），也支持限定为特定操作的用户定义角色和精细资源（例如基于您的要求的集合）。

RBAC 的常见使用案例包括通过创建对集群中数据库具有只读访问权限的用户来强制执行最低权限，以及允许单个用户访问集群中给定数据库的多租户应用程序设计。

> **Note**
>
> 所有在 2020 年 3 月 26 日之前创建的新用户都已被授予 `dbAdminAnyDatabase`、`readWriteAnyDatabase` 和 `clusterAdmin` 角色。建议您重新评估所有现有用户并根据需要修改角色，以便为您的集群强制执行最低权限。

主题

- RBAC 概念
- RBAC 内置角色入门
- RBAC 用户定义角色入门
- 以用户身份连接到 Amazon DocumentDB

- 通用命令
- 功能差异
- 限制
- 使用访问控制 Role-Based 访问数据库

RBAC 概念

以下是与基于角色的访问控制相关的重要术语和概念。有关 Amazon DocumentDB 用户的更多信息，请参阅 管理 Amazon DocumentDB 用户。

- 用户 可进行身份验证以访问数据库并执行操作的单个实体。
- 密码 用于对用户进行身份验证的密钥。
- 角色 授权用户对一个或多个数据库执行操作。
- 管理数据库 用户在其中存储并已得到授权的数据库。
- 数据库 (**db**) 集群中包含用于存储文档的集合的命名空间。

以下命令创建名为 `sample-user` 的用户。

```
db.createUser({user: "sample-user", pwd: "abc123", roles: [{role: "read", db: "sample-database"}]})
```

在本示例中：

- `user: "sample-user"` 表示用户名。
- `pwd: "abc123"` 表示用户密码。
- `role: "read", "db: "sample-database"` 表示用户 `sample-user` 将在 `sample-database` 中具有读取权限。



以下示例显示您为用户 `sample-user` 指定 `db.getUser(sample-user)` 后的输出。在此示例中，用户 `sample-user` 驻留在 `admin` 数据库中，但具有 `sample-database` 数据库的读取角色。



在创建用户时，如果在指定角色时省略 db 字段，则 Amazon DocumentDB 会隐式地将该角色归属于发出连接的数据库。例如，如果您的连接是针对数据库 sample-database 发出的，并且您运行以下命令，则用户 sample-user 将在 admin 数据库中创建并具有对数据库 sample-database 的 readWrite 权限。

```
db.createUser({user: "sample-user", pwd: "abc123", roles: ["readWrite"]})
```

此操作的输出将类似于下文。

```
{
   "user":"sample-user",
   "roles":[
      {
         "db":"sample-database",
         "role":"readWrite"
      }
   ]
}
```

创建具有跨所有数据库范围的角色的用户（例如，readAnyDatabase）需要在创建用户时处于 admin 数据库的上下文，或者在创建用户时明确指定角色的数据库。要针对 admin 数据库发出命令，可以使用命令 use admin。有关更多信息，请参阅 通用命令。

RBAC 内置角色入门

为帮助您掌握基于角色的访问控制，本部分将向您介绍通过为三个用户创建具有不同职能的角色来强制执行最低权限的示例方案。

- user1 是一位新上任的经理，需要能够查看和访问集群中的所有数据库。

- user2 是一位新员工，只需访问同一集群中的一个数据库 sample-database-1。
- user3 是现有员工，需要查看和访问同一集群中其以前无权访问的不同数据库 sample-database-2。

在这之后，user1 和 user2 都离开了公司，所以必须撤销他们的访问权限。

要创建用户和授予角色，进入集群用于进行身份验证的用户必须具有可以为 createUser 和 grantRole 执行操作的关联角色。例如，角色 admin 和 userAdminAnyDatabase 都可以授予执行上述功能的权限。有关每个角色的操作，请参阅使用访问控制 Role-Based 访问数据库。

> **Note**
>
> 在 Amazon DocumentDB 中，所有用户和角色操作（例如，create、get、drop、grant、revoke 等等）都在 admin 数据库中隐式执行，无论您是否对 admin 数据库发出命令。

首先，要了解集群中的当前用户和角色，可以运行 show users 命令，如以下示例所示。您将看到两个用户，即集群的 serviceadmin 和主用户。这两个用户始终存在，无法将它们删除。有关更多信息，请参阅 管理 Amazon DocumentDB 用户。

```
show users
```

对于 user1，使用以下命令创建具有对整个集群中所有数据库的读写访问权限的角色。

```
db.createUser({user: "user1", pwd: "abc123", roles: [{role: "readWriteAnyDatabase", db:
 "admin"}]})
```

此操作的输出将类似于下文。

```
{
   "user":"user1",
   "roles":[
      {
         "role":"readWriteAnyDatabase",
         "db":"admin"
      }
   ]
```

```
}
```

对于 user2，使用以下命令创建对数据库 sample-database-1 具有只读访问权限的角色。

```
db.createUser({user: "user2", pwd: "abc123", roles: [{role: "read", db: "sample-
database-1"}]})
```

此操作的输出将类似于下文。

```
{
   "user":"user2",
   "roles":[
      {
         "role":"read",
         "db":"sample-database-1"
      }
   ]
}
```

要模拟 user3 是现有用户的方案，请先创建用户 user3，然后将新角色分配给 user3。

```
db.createUser({user: "user3", pwd: "abc123", roles: [{role: "readWrite", db: "sample-
database-1"}]})
```

此操作的输出将类似于下文。

```
{
   "user":"user3",
   "roles":[
      {
         "role":"readWrite",
         "db":"sample-database-1"
      }
   ]
}
```

现在已创建了用户 user3，请为 user3 分配可对 sample-database-2 进行 read 的角色。

```
db.grantRolesToUser("user3", [{role: "read", db: "sample-database-2"}])
```

最后，user1 和 user2 都离开了公司，需要撤销他们对集群的访问权限。您可以通过删除用户来执行此操作，如下所示。

```
db.dropUser("user1")
db.dropUser("user2")
```

要确保所有用户都具有适当角色，可使用以下命令列出所有用户。

```
show users
```

此操作的输出将类似于下文。

```
{
   "_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 用户定义角色入门

为了帮助您开始使用用户定义的角色，本节将带您了解通过为具有不同工作功能的三个用户创建角色来强制执行最低权限的示例场景。

在本例中，以下条件适用：

- user1 是一位新上任的经理，需要能够查看和访问集群中的所有数据库。
- user2 是一位新员工，只需要对同一集群中的一个数据库 sample-database-1 执行“查找”操作。
- user3 是现有员工，需要查看和访问不同数据库 sample-database-2 中的特定集合 col2，而他们以前在同一集群中无法访问该集合。
- 对于 user1，使用以下命令创建具有对整个集群中所有数据库的读写访问权限的角色。

```
db.createUser(
 {
     user: "user1", pwd: "abc123",
     roles: [{role: "readWriteAnyDatabase", db: "admin"}]
 }
 )
```

此操作的输出将类似于下文。

```
{
   "user":"user1",
   "roles":[
      {
         "role":"readWriteAnyDatabase",
         "db":"admin"
      }
   ]
}
```

对于 user2，使用以下命令创建一个对数据库 sample-database-1 中所有集合具有“查找”权限的角色。请注意，此角色将确保任何关联用户只能运行查找查询。

```
db.createRole(
{
    role: "findRole",
    privileges: [
    {
        resource: {db: "sample-database-1", collection: ""}, actions: ["find"]
    }],
    roles: []
}
)
```

此操作的输出将类似于下文。

```
{
   "role":"findRole",
   "privileges":[
      {
         "resource":{
            "db":"sample-database-1",
            "collection":""
         },
         "actions":[
            "find"
         ]
      }
   ],
   "roles":[

   ]
}
```

接下来，创建用户（user2）并将最近创建的角色 findRole 附加到该用户。

```
db.createUser(
{
    user: "user2",
    pwd: "abc123",
    roles: []
})
```

```
 db.grantRolesToUser("user2",["findRole"])
```

为了模拟 user3 是现有用户的场景，首先创建用户 user3，然后创建一个名为 collectionRole 的新角色，我们将在下一步中分配到 user3。

现在，您可以分配新角色到 user3。此新角色将允许user3插入、更新、删除和查找sample-database-2中一个特定集合 col2 的访问权限。

```
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: []
}
)
```

此操作的输出将类似于下文。

```
{
   "role":"collectionRole",
   "privileges":[
      {
         "resource":{
            "db":"sample-database-2",
            "collection":"col2"
         },
         "actions":[
            "find",
            "update",
            "insert",
```

```
            "remove"
         ]
      }
   ],
   "roles":[

   ]
}
```

现在已创建了用户 user3，你可以位角色 collectionFind 分配 user3。

```
db.grantRolesToUser("user3",["collectionRole"])
```

最后，user1 和 user2 都离开了公司，需要撤销他们对集群的访问权限。您可以通过删除用户来执行此操作，如下所示。

```
db.dropUser("user1")
db.dropUser("user2")
```

要确保所有用户都具有适当角色，可使用以下命令列出所有用户。

```
show users
```

此操作的输出将类似于下文。

```
{
   "_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"
      }
   ]
}
```

以用户身份连接到 Amazon DocumentDB

在连接到 Amazon DocumentDB 集群时，需要在特定数据库的上下文中进行连接。默认情况下，如果未在连接字符串中指定数据库，则会在 `test` 数据库上下文中自动连接到集群。所有集合级别命令（如 `insert` 和 `find`）都是针对 `test` 数据库中的集合发出的。

要查看您所在上下文或者换句话说，对其发出命令的数据库，请在 mongo Shell 中使用 db 命令，如下所示。

查询：

```
db
```

输出：

```
test
```

尽管默认连接可能位于 `test` 数据库的上下文中，但这并不一定意味着与连接关联的用户有权对 `test` 数据库执行操作。在上述示例方案中，如果您用于进行身份验证的用户 `user3` 是具有对 `sample-database-1` 数据库的 `readWrite` 权限的角色，则连接的默认上下文是 `test` 数据库。但是，如果您尝试将文档插入 `test` 数据库中的集合中，您将收到 Authorization failure (授权失败) 错误消息。这是因为该用户未获得对该数据库执行该命令的授权，如下所示。

查询：

```
db
```

输出：

```
test
```

查询：

```
db.col.insert({x:1})
```

输出：

```
WriteCommandError({ "ok" : 0, "code" : 13, "errmsg" : "Authorization failure" })
```

如果您将连接上下文更改为 sample-database-1 数据库，则可以写入到集合中，因为该用户具有执行此操作的授权。

查询：

```
use sample-database-1
```

输出：

```
switched to db sample-database-1
```

查询：

```
db.col.insert({x:1})
```

输出：

```
WriteResult({ "nInserted" : 1})
```

使用特定用户对集群进行身份验证时，还可以在连接字符串中指定数据库。如果这样做，则用户在通过 admin 数据库的身份验证后，不再需要执行 use 命令。

以下连接字符串根据 admin 数据库对用户进行身份验证，但连接上下文则是 sample-database-1 数据库。

```
mongo "mongodb://user3:abc123@sample-cluster.node.us-east-1.docdb.amazonaws.com:27017/
sample-database-2"
```

通用命令

本节提供在 Amazon DocumentDB 中使用基于角色的访问控制的常用命令示例。您必须位于 admin 数据库的上下文中，才能创建和修改用户和角色。您可以使用 use admin 命令切换到 admin 数据库。

> **Note**
>
> 对用户和角色的修改将隐式发生在 admin 数据库中。创建具有跨所有数据库范围的角色的用户（例如，readAnyDatabase）需要在创建用户时处于 admin 数据库上下文（即 use admin），或者在创建用户时明确指定角色的数据库（如本节的示例 2 所示）。

示例 1：创建对数据库 foo 具有 read 角色的用户。

```
db.createUser({user: "readInFooBar", pwd: "abc123", roles: [{role: "read", db:
 "foo"}]})
```

此操作的输出将类似于下文。

```
{
   "user":"readInFooBar",
   "roles":[
      {
         "role":"read",
         "db":"foo"
      }
   ]
}
```

示例 2：创建具有所有数据库的读取访问权限的用户。

```
db.createUser({user: "readAllDBs", pwd: "abc123", roles: [{role: "readAnyDatabase", db:
 "admin"}]})
```

此操作的输出将类似于下文。

```
{
   "user":"readAllDBs",
   "roles":[
      {
         "role":"readAnyDatabase",
         "db":"admin"
      }
   ]
}
```

示例 3：向新数据库的现有用户授予 read 角色。

```
db.grantRolesToUser("readInFooBar", [{role: "read", db: "bar"}])
```

示例 4：更新用户的角色。

```
db.updateUser("readInFooBar", {roles: [{role: "read", db: "foo"}, {role: "read", db:
 "baz"}]})
```

示例 5：撤销用户对数据库的访问权限。

```
db.revokeRolesFromUser("readInFooBar", [{role: "read", db: "baz"}])
```

示例 6：描述内置角色。

```
db.getRole("read", {showPrivileges:true})
```

此操作的输出将类似于下文。

```
{
   "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"
         ]
      }
   }
```

示例 7：从集群中删除用户。

```
db.dropUser("readInFooBar")
```

此操作的输出将类似于下文。

```
true
```

示例 8：创建对特定集合具有读写权限的角色

```
db.createRole(
{
    role: "collectionRole",
    privileges: [
    {
        resource: {db: "sample-database-2", collection: "col2"}, actions: ["find",
 "update", "insert", "remove"]
    }],
    roles: []
}
)
```

此操作的输出将类似于下文。

```
{
   "role":"collectionRole",
   "privileges":[
      {
         "resource":{
            "db":"sample-database-2",
            "collection":"col2"
         },
         "actions":[
            "find",
            "update",
            "insert",
            "remove"
         ]
      }
   ],
   "roles":[

   ]
}
```

示例 9：创建用户并分配用户定义角色

```
db.createUser(
{
    user: "user3",
    pwd: "abc123",
```

```
    roles: []
})

db.grantRolesToUser("user3",["collectionRole"])
```

示例 10：向用户定义角色授予其他权限

```
db.grantPrivilegesToRole(
  "collectionRole",
  [
    {
      resource: { db: "sample-database-1", collection: "col1" },
      actions: ["find", "update", "insert", "remove"]
    }
  ]
)
```

示例 11：删除用户定义角色的权限

```
db.revokePrivilegesFromRole(
  "collectionRole",
  [
    {
      resource: { db: "sample-database-1", collection: "col2" },
      actions: ["find", "update", "insert", "remove"]
    }
  ]
)
```

示例 12：更新现有用户定义角色

```
db.updateRole(
  "collectionRole",
  {
    privileges: [
    {
        resource: {db: "sample-database-3", collection: "sample-collection-3"},
 actions: ["find", "update", "insert", "remove"]
    }],
    roles: []
  }
)
```

功能差异

在 Amazon DocumentDB 中，用户和角色定义存储在 admin 数据库中，而且根据 admin 数据库对用户进行身份验证。此功能与 MongoDB 社区版不同，但与 MongoDB Atlas 一致。

Amazon DocumentDB 还支持变更流，该功能提供按时间顺序排列的更改事件，这些事件在您的集集群合中发生。listChangeStreams 操作应用于集群级别（即跨所有数据库），modifyChangeStreams 操作可以应用于数据库级别和集群级别。

限制

下表包含 Amazon DocumentDB 中 Role-Based 访问控制的限制。

说明	限制
每个集群的用户数	1000
与用户关联的角色数	1000
用户定义角色数	100
与权限关联的资源数	100

使用访问控制 Role-Based 访问数据库

借助基于角色的访问控制，您可以创建一个用户并向其授予一个或多个角色，以确定该用户可以在数据库或集群中执行哪些操作。

以下是 Amazon DocumentDB 中目前支持的内置角色的列表。

> **Note**
>
> 在 Amazon DocumentDB 4.0 和 5.0 中，ListCollection 和 ListDatabase 命令可以选择使用 authorizedCollections 和 authorizedDatabases 参数列出用户有权访问分别需要 listCollections 和 listDatabase 角色的集合和数据库。此外，用户现在可以在不需要 KillCursor 角色的情况下终止自己的游标。

Database user

角色名称	说明	操作
read	授予用户对指定数据库的读取权限。	changeStreams collStats dbStats find killCursors listIndexes listCollections
readWrite	授予用户对指定数据库的读取和写入访问权限。	使用 read 权限的所有操作。 createCollection dropCollection createIndex dropIndex insert killCursors listIndexes listCollections remove

角色名称	说明	操作
		update

Cluster user

角色名称	说明	操作
读 AnyDatabase	授予用户对集群中所有数据库的读取权限。	使用 read 权限的所有操作。 listChangeStreams listDatabases
readWriteAnyDatabase	授予用户对集群中所有数据库的读写权限。	使用 readWrite 权限的所有操作。 listChangeStreams listDatabases
userAdminAnyDatabase	授予用户为所有用户分配或修改对指定数据库的角色或权限的能力。	changeCustomData changePassword createUser dropRole dropUser grantRole listDatabases revokeRole

角色名称	说明	操作
		viewRole viewUser
dbAdminAnyDatabase	授予用户对所有指定数据库执行数据库管理角色的能力。	使用 dbAdmin 权限的所有操作。 dropCollection listDatabases listChangeStreams modifyChangeStreams

Superuser

角色名称	说明	操作
root	授予用户对以下所有角色的合并资源和操作的访问权限：readWriteAnyDatabase 、dbAdminAnyDatabase 、userAdminAnyDatabase 、clusterAdmin 、restore 和 backup。	使用 readWriteAnyDatabase 、dbAdminAnyDatabase 、userAdminAnyDatabase 、clusterAdmin 、restore 和 backup 的所有操作。

Database administrator

角色名称	说明	操作
dbAdmin	授予用户对指定数据库执行管理任务的能力。	bypassDocumentValidation collMod collStats createCollection createIndex dropCollection dropDatabase dropIndex dbStats find killCursors listIndexes listCollections modifyChangeStreams
dbOwner	通过合并角色 dbAdmin 和 readWrite 授予用户对指定数据库执行任何管理任务的能力。	使用 dbAdmin 和 readWrite 的所有操作。

Cluster administrator

角色名称	说明	操作
clusterAdmin	通过合并 clusterManager 、clusterMonitor 和 hostManager 角色授予用户最大的集群管理访问权限。	使用 clusterManager 、clusterMonitor 和 hostManager 的所有操作。 listChangeStreams dropDatabase modifyChangeStreams
clusterManager	授予用户对指定集群执行管理和监控操作的能力。	listChangeStreams listSessions modifyChangeStreams replSetGetConfig
clusterMonitor	授予用户对监控工具可具有只读访问权限的能力。	collStats dbStats find getParameter hostInfo indexStats

角色名称	说明	操作
		killCursors listChangeStreams listCollections listDatabases listIndexes listSessions replSetGetConfig serverStatus top
hostManager	授予用户监视和管理服务器的能力。	auditConfigure killCursors killAnyCursor killAnySession killop

Backup administrator

角色名称	说明	操作
backup	授予用户备份数据所需的访问权限。	getParameter insert

角色名称	说明	操作
		find listChangeStreams listCollections listDatabases listIndexes update

角色名称	说明	操作
restore	授予用户还原数据所需的访问权限。	bypassDocumentValidation changeCustomData changePassword collMod createCollection createIndex createUser dropCollection dropRole dropUser getParameter grantRole find insert listCollections modifyChangeStreams revokeRole

角色名称	说明	操作

		`remove`
		`viewRole`
		`viewUser`
		`update`

Amazon DocumentDB 中的日志记录和监控

亚马逊 DocumentDB（兼容 MongoDB）提供了各种亚马逊指标，您可以监控这些 CloudWatch 指标，以确定您的亚马逊 DocumentDB 集群和实例的运行状况和性能。您可以使用各种工具查看亚马逊 DocumentDB 指标，包括亚马逊 DocumentDB 控制台、 AWS CLI、 CloudWatch 亚马逊控制台和 API。 CloudWatch 有关监控的更多信息，请参阅监控 Amazon DocumentDB。

除了 Amazon CloudWatch 指标外，您还可以使用分析器记录在集群上执行的操作的执行时间和详细信息。对于监控集群上速度最慢的操作以帮助您提高单个查询的性能和整体集群性能，分析器非常有用。启用后，操作将记录到 Amazon CloudWatch Logs 中，您可以使用 CloudWatch Insight 来分析、监控和存档您的 Amazon DocumentDB 分析数据。有关更多信息，请参阅 分析 Amazon DocumentDB 操作。

Amazon DocumentDB 还与一项服务集成，该服务提供用户 AWS CloudTrail、角色或 AWS 服务在 Amazon DocumentDB 中执行的操作的记录（兼容 MongoDB）。 CloudTrail 将亚马逊文档数据库的所有 AWS CLI API 调用捕获为事件，包括来自亚马逊 DocumentDB 的调用和对亚马逊 D AWS 管理控制台 ocumentDB 软件开发工具包的代码调用。有关更多信息，请参阅 使用记录亚马逊 DocumentDB API 调用 AWS CloudTrail。

使用 Amazon DocumentDB，您可以审核在集群中执行的事件。记录的事件的示例包括成功和失败的身份验证尝试、删除数据库中的集合或创建索引。默认情况下，在 Amazon DocumentDB 上禁用审核，并要求您选择使用该功能。有关更多信息，请参阅 审核 Amazon DocumentDB 事件。

更新您的 Amazon DocumentDB TLS 证书

主题

- 更新您的应用程序和 Amazon DocumentDB 集群
- 自动服务器证书轮换

- 问题排查
- 常见问题

Amazon DocumentDB 集群的证书颁发机构（CA）证书已于 2024 年 8 月更新。如果您使用 Amazon DocumentDB 集群时启用了传输层安全性协议（TLS）（默认设置），而且您没有轮换您的客户端应用程序和服务器证书，则需要执行以下步骤以避免您的应用程序与 Amazon DocumentDB 集群之间发生连接性问题。

- 步骤 1：下载新的 CA 证书并更新您的应用程序
- 步骤 2：更新服务器证书

作为 Amazon DocumentDB 的标准维护和安全最佳实践的一部分，CA 和服务器证书已更新。客户端应用程序必须将新的 CA 证书添加到其信任存储中，且现有 Amazon DocumentDB 实例必须更新为在此到期日期之前使用新的 CA 证书。

更新您的应用程序和 Amazon DocumentDB 集群

按照此部分中的步骤更新应用程序的 CA 证书捆绑包（步骤 1），以及您集群的服务器证书（步骤 2）。在将变更应用于生产环境之前，我们强烈建议您在开发环境或登台环境中测试这些步骤。

> **Note**
>
> 您必须完成每个 AWS 区域 拥有亚马逊 DocumentDB 集群的步骤 1 和步骤 2。

步骤 1：下载新的 CA 证书并更新您的应用程序

下载新的 CA 证书并更新您的应用程序，以便使用新的 CA 证书创建与 Amazon DocumentDB 的 TLS 连接。从 https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem 下载新的 CA 证书捆绑包。此操作将下载名为 `global-bundle.pem` 的文件。

> **Note**
>
> 如果您要访问包含旧 CA 证书 (`rds-ca-2019-root.pem`) 和新 CA 证书（`rds-ca-rsa2048-g1`、`rds-ca-rsa4096-g1`、`rds-ca-ecc384-g1`）的密钥库，请验证密钥库是否选择了 `global-bundle`。

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

接下来，更新应用程序以使用新的证书捆绑包。新 CA 捆绑包同时包含旧 CA 证书 (rds-ca-2019) 和新 CA 证书 (rds-ca-rsa2048-g1、rds-ca-rsa4096-g1、rds-ca-ecc384-g1)。新的 CA 捆绑包中同时包含这两个 CA 证书，您可以通过两个步骤来更新应用程序和集群。

对于 Java 应用程序，必须使用新的 CA 证书创建新的信任存储。有关说明，请参阅 启用了 TLS 的情况下的连接 主题中的 Java 选项卡。

要验证您的应用程序使用的是否是最新的 CA 证书捆绑包，请参阅我如何确定我使用的是最新的 CA 捆绑包？。如果您已在应用程序中使用最新的 CA 证书捆绑包，则可跳至步骤 2。

有关将 CA 捆绑与您的应用程序结合使用的示例，请参阅 加密传输中数据 和 启用了 TLS 的情况下的连接。

> **Note**
>
> 目前，MongoDB Go Driver 1.2.1 只接受 `sslcertificateauthorityfile` 中的一个 CA 服务器证书。有关启启用了 TLS 的情况下的连接用 TLS 时使用 Go 连接到亚马逊文档数据库的信息，请参阅。

步骤 2：更新服务器证书

在更新应用程序以使用新的 CA 捆绑包后，下一步是通过修改 Amazon DocumentDB 集群中每个实例来更新服务器证书。要修改实例以使用新的服务器证书，请参阅以下说明。

Amazon DocumentDB 提供以下 CA 来签署数据库实例的数据库服务器证书。

- rds-ca-ecc384-g1：使用具有 ECC 384 私有密钥算法和 SHA384 签名算法的证书颁发机构。此 CA 支持服务器证书自动轮换。这目前仅在 Amazon DocumentDB 4.0 和 5.0 上受支持。
- rds-ca-rsa2048-g1 — 在大多数地区使用具有 RS A 2048 私钥算法和 SHA256 签名算法的证书颁发机构。 AWS 此 CA 支持服务器证书自动轮换。
- rds-ca-rsa4096-g1：使用具有 RSA 4096 私有密钥算法和 SHA384 签名算法的证书颁发机构。此 CA 支持服务器证书自动轮换。

> **Note**
>
> 如果您使用的是 AWS CLI，则可以使用 d es cribe-certifices 查看上一节中列出的证书颁发机构的有效性。

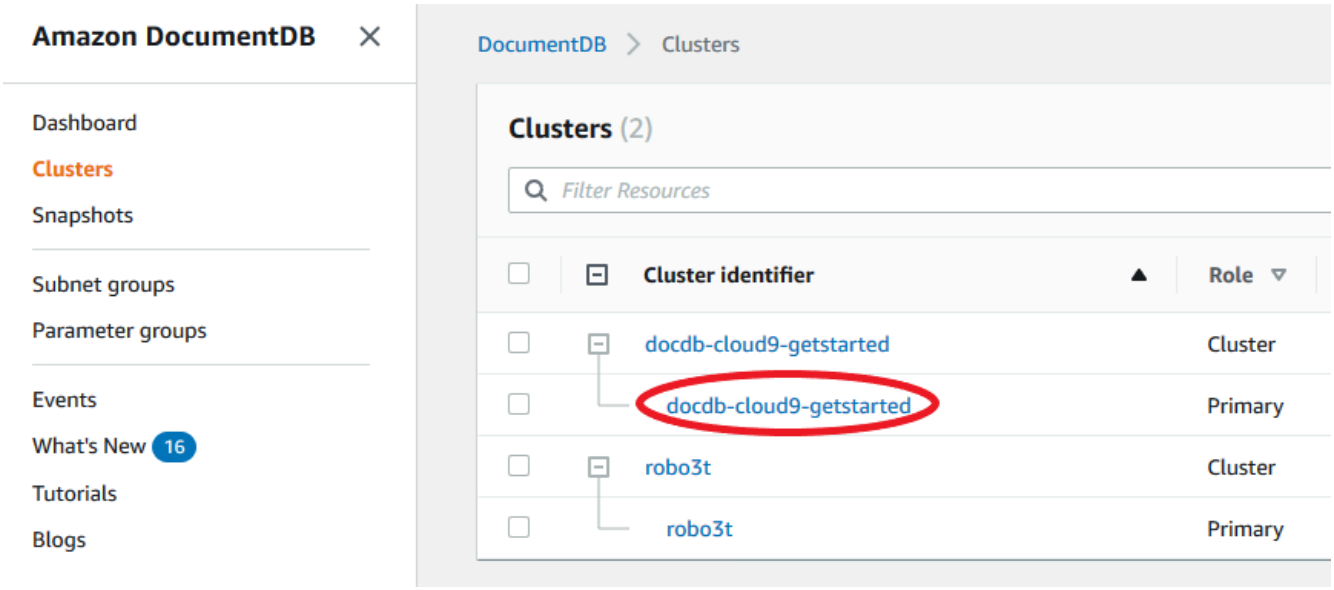
这些 CA 证书包含在区域和全球证书捆绑包中。当您将 rds-ca-rsa2048-g1、rds-ca-rsa4096-g1 或 rds-ca-ecc384-g1 CA 用于数据库时，Amazon DocumentDB 会管理数据库上的数据库服务器证书。Amazon DocumentDB 会在数据库服务器证书过期前自动轮换。

> **Note**
>
> 如果集群运行在以下引擎修订版本上，Amazon DocumentDB 不需要重启即可进行证书轮换：
>
> - Amazon DocumentDB 3.6：1.0.208662 或更高版本
> - Amazon DocumentDB 4.0：2.0.10179 或更高版本
> - Amazon DocumentDB 5.0：3.0.4780 或更高版本
>
> 您可以通过运行以下命令来确定当前的 Amazon DocumentDB 引擎修补版本：`db.runCommand({getEngineVersion: 1})`。
> 在更新服务器证书之前，请确保您已完成 步骤 1。

Using the AWS 管理控制台

完成以下步骤，以使用 AWS 管理控制台为现有 Amazon DocumentDB 实例标识和轮换旧服务器证书。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在屏幕右上角的区域列表中，选择您的集群 AWS 区域 所在的。
3. 在控制台左侧的导航窗格中，选择集群。
4. 您可能需要确定哪些实例仍在旧服务器证书上 (`rds-ca-2019`)。您可以在 Clusters（集群）表最右侧的 Certificate authority（证书颁发机构）列中执行此操作。
5. 在 Clusters（集群）表中，您将在最左边看到 Cluster identifier（集群标识符）列。您的实例列在集群下，类似于以下屏幕截图。



6. 选中您感兴趣的实例左侧的框。
7. 选择 Actions（操作），然后选择 Modify（修改）。
8. 在 Certificate authority (证书颁发机构) 下，为此实例选择新的服务器证书（`rds-ca-rsa2048-g1`）。
9. 在下一页上可以看到所做更改的摘要。请注意，在修改实例之前，会有一个额外的警报提醒您确保应用程序使用的是最新的证书 CA 捆绑包，以免造成连接中断。
10. 可以选择在下一个维护时段内应用修改，也可以立即应用。如果您打算立即修改服务器证书，请使用 Apply Immediately (立即应用) 选项。
11. 选择修改实例以完成更新。

Using the AWS CLI

完成以下步骤，以使用 AWS CLI为现有 Amazon DocumentDB 实例标识和轮换旧服务器证书。

1. 要立即修改实例，请对集群中的每个实例执行下面的命令。

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier>
 --ca-certificate-identifier rds-ca-rsa2048-g1 --apply-immediately
```

2. 要修改集群中的实例，以便在集群的下一个维护时段中使用新的 CA 证书，请对集群中的每个实例执行以下命令。

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier>
 --ca-certificate-identifier rds-ca-rsa2048-g1 --no-apply-immediately
```

自动服务器证书轮换

Amazon DocumentDB 支持服务器证书自动轮换。服务器证书是颁发给每个集群实例的叶证书。与根 CA 证书不同，服务器证书有效期较短（12 个月），Amazon DocumentDB 无需您执行任何操作即可自动处理证书轮换。Amazon DocumentDB 使用相同的根 CA 进行自动轮换，因此您无需下载新的 CA 服务包。

> **Important**
>
> 在连接到 Amazon DocumentDB 集群时，我们建议您信任根 CA 服务包，而不是直接信任每个服务器证书。这将防止服务器证书轮换后出现连接错误。请参阅启用了 TLS 的情况下的连接。

Amazon DocumentDB 尝试在您首选的维护时段中，在服务器证书的半生命周期轮换您的服务器证书。新的服务器证书的有效期为 12 个月。

使用 describe-db-engine-versions 命令并检查 `SupportsCertificateRotationWithoutRestart` 标志，以确定引擎版本是否支持无需重启即可轮换证书。

> **Note**
>
> 如果集群运行在以下引擎修订版本上，Amazon DocumentDB 支持无需重启即可进行服务器证书轮换：
>
> - Amazon DocumentDB 3.6：1.0.208662 或更高版本
> - Amazon DocumentDB 4.0：2.0.10179 或更高版本
> - Amazon DocumentDB 5.0：3.0.4780 或更高版本
>
> 您可以通过运行以下命令来确定当前的 Amazon DocumentDB 引擎补丁版本：`db.runCommand({getEngineVersion: 1})`。
> 如果您使用的是较旧的引擎补丁版本，Amazon DocumentDB 将轮换服务器证书，并在您首选的维护时段安排数据库重启事件。

问题排查

如果您在证书轮换过程中遇到集群连接问题，我们建议您执行以下操作：

- 验证您的客户端是否使用最新的证书捆绑包。请参阅我如何确定我使用的是最新的 CA 捆绑包？。
- 验证您的实例是否使用最新的证书。请参阅我如何知道我的哪个 Amazon DocumentDB 实例正在使用 old/new 服务器证书？。
- 验证您的应用程序是否在使用最新证书 CA。某些驱动程序（如 Java 和 Go）需要额外的代码，才能将多个证书从证书捆绑包导入到信任存储中。有关使用 TLS 连接到 Amazon DocumentDB 的更多信息，请参阅 以编程方式连接到 Amazon DocumentDB。
- 请联系客户支持人员。如果您有任何疑问或问题，请联系 支持。

常见问题

以下是有关 TLS 证书的一些常见问题的答案。

如果我有疑问或问题，应该怎么办？

如果您有任何疑问或问题，请联系 支持。

如何知道我是否在使用 TLS 连接我的 Amazon DocumentDB 集群？

您可以通过检查集群的集群参数组的 `tls` 参数来确定您的集群是否使用 TLS。如果将 `tls` 参数设置为 `enabled`，则表示您正在使用 TLS 证书连接到您的集群。有关更多信息，请参阅 管理 Amazon DocumentDB 集群参数组。

为什么要更新 CA 和服务器证书？

作为 Amazon DocumentDB 的标准维护和安全最佳实践的一部分，CA 和服务器证书正在更新。

如果我在到期日之前没有采取任何行动会怎样？

如果您使用已过期的 CA 证书通过 TLS 连接到您的 Amazon DocumentDB 集群，那么通过 TLS 连接的应用程序将无法再与 Amazon DocumentDB 集群通信。

Amazon DocumentDB 将不会在到期之前自动轮换您的数据库证书。您必须在到期日之前或之后更新应用程序和集群以使用新的 CA 证书。

我如何知道我的哪个 Amazon DocumentDB 实例正在使用 old/new 服务器证书？

要识别仍在使用旧服务器证书的 Amazon DocumentDB 实例，您可以使用亚马逊 D AWS 管理控制台 ocumentDB 或。 AWS CLI

使用 AWS 管理控制台

要识别集群中正在使用旧证书的实例

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在屏幕右上角的区域列表中，选择您的实例 AWS 区域 所在的。
3. 在控制台左侧的导航窗格中，选择集群。
4. Certificate authority (证书颁发机构) 列（靠近表格最右端）会显示哪些实例仍在使用旧的服务器证书（`rds-ca-2019`），哪些实例正在使用新的服务器证书（`rds-ca-rsa2048-g1`）。

使用 AWS CLI

要识别集群中正在使用旧服务器证书的实例，请使用带以下的 `describe-db-clusters` 命令。

```
aws docdb describe-db-instances \
    --filters Name=engine,Values=docdb \
    --query 'DBInstances[*].
{CertificateVersion:CACertificateIdentifier,InstanceID:DBInstanceIdentifier}'
```

如何修改 Amazon DocumentDB 集群中的单个实例以更新服务器证书？

我们建议您在给定集群中同时更新所有实例的服务器证书。要修改您的集群中的实例，可以使用控制台或 AWS CLI。

> **Note**
>
> 在更新服务器证书之前，请确保您已完成步骤 1。

使用 AWS 管理控制台

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在屏幕右上角的区域列表中，选择您的集群 AWS 区域 所在的。
3. 在控制台左侧的导航窗格中，选择集群。
4. Certificate authority (证书颁发机构) 列（靠近表格最右端）会显示哪些实例仍在使用旧的服务器证书 (`rds-ca-2019`)。
5. 在 Clusters（集群）表的 Cluster identifier（集群标识符）下，选择要修改的实例。
6. 选择 Actions（操作），然后选择 Modify（修改）。
7. 在 Certificate authority (证书颁发机构) 下，为此实例选择新的服务器证书（`rds-ca-rsa2048-g1`）。
8. 在下一页上可以看到所做更改的摘要。请注意，在修改实例之前，会有一个额外的警报提醒您确保应用程序使用的是最新的证书 CA 捆绑包，以免造成连接中断。
9. 可以选择在下一个维护时段内应用修改，也可以立即应用。
10. 选择修改实例以完成更新。

使用 AWS CLI

完成以下步骤，以使用 AWS CLI为现有 Amazon DocumentDB 实例标识和轮换旧服务器证书。

1. 要立即修改实例，请对集群中的每个实例执行下面的命令。

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier> --
ca-certificate-identifier rds-ca-rsa2048-g1 --apply-immediately
```

2. 要修改集群中的实例，以便在集群的下一个维护时段中使用新的 CA 证书，请对集群中的每个实例执行以下命令。

```
aws docdb modify-db-instance --db-instance-identifier <yourInstanceIdentifier> --
ca-certificate-identifier rds-ca-rsa2048-g1 --no-apply-immediately
```

如果我向现有集群中添加新的实例，会怎么样？

创建的所有新实例都将使用旧的服务器证书，并且需要使用旧的 CA 证书的 TLS 连接。2024 年 1 月 25 日之后创建的任何新 Amazon DocumentDB 实例将默认使用新证书 rds-ca-rsa2048-g1。

如果我的集群发生实例替换或故障转移，会怎么样？

如果集群中有实例替换，则创建的新实例将继续使用该实例以前使用的服务器证书。我们建议您同时更新所有实例的服务器证书。如果集群中发生失效转移，则使用新主实例上的服务器证书。

如果我没有使用 TLS 连接到我的集群，我还需要更新每个实例吗？

我们强烈建议启用 TLS。如果未启用 TLS，我们仍然建议您在将来计划使用 TLS 连接集群时轮换 Amazon DocumentDB 实例上的证书。如果您从未计划使用 TLS 连接到 Amazon DocumentDB 集群，则不需要执行操作。

如果我目前没有使用 TLS 连接到集群，但计划将来这样做，该怎么办？

如果您的集群是在 2024 年 1 月之前创建的，请按照上一部分中的步骤 1 和步骤 2 进行操作，以确保您的应用程序使用的是更新后的 CA 捆绑包，且每个 Amazon DocumentDB 实例使用的都是最新的服务器证书。如果您的集群是在 2025 年 1 月 25 日之后创建的，那么您的集群已经具有最新的服务器证书 (rds-ca-rsa2048-g1)。要验证您的应用程序使用的是否是最新的 CA 捆绑包，请参阅如果我没有使用 TLS 连接到我的集群，我还需要更新每个实例吗？

截止日期能否延至 2024 年 8 月以后？

如果您的应用程序通过 TLS 进行连接，则无法延长截止日期。

我如何确定我使用的是最新的 CA 捆绑包？

要验证是否在使用最新的捆绑包，您可以使用以下命令。要运行此命令，必须安装 java，并且 java 工具必须位于 Shell 的 PATH 变量中。有关更多信息，请参阅使用 Java。

macOS 和 Amazon Linux

```
keytool -printcert -v -file global-bundle.pem
```

Windows

```
keytool -printcert -v -file global-bundle.p7b
```

为什么我在 CA 捆绑包的名称中看到 “RDS”？

对于某些管理功能，例如证书管理，Amazon DocumentDB 使用与 Amazon Relational Database Service (Amazon RDS) 共享的操作技术。

新的证书何时到期？

新服务器证书将（通常）按以下方式过期：

- rds-ca-rsa2048-g1：2061 过期
- rds-ca-rsa4096-g1：2121 过期
- rds-ca-ecc384-g1：2121 过期

如果我在证书到期之前未采取行动，会看到什么样的错误？

错误消息因驱动程序而异。通常，您会看到含有“证书已过期”字符串的证书验证错误。

如果我应用了新的服务器证书，我可以恢复为使用旧的服务器证书吗？

如果您需要将实例恢复为使用旧的服务器证书，建议您对集群中的所有实例都执行此操作。您可以使用 AWS 管理控制台 或恢复集群中每个实例的服务器证书 AWS CLI。

使用 AWS 管理控制台

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在屏幕右上角的区域列表中，选择您的集群 AWS 区域 所在的。
3. 在控制台左侧的导航窗格中，选择集群。
4. 在 Clusters（集群）表的 Cluster identifier（集群标识符）下，选择要修改的实例。选择 Actions (操作)，然后选择 Modify (修改)。
5. 在 Certificate authority (证书颁发机构) 下，您可以选择旧的服务器证书 (`rds-ca-2019`)。
6. 选择 Continue (继续) 以查看修改摘要。
7. 在显示的页面中，您可以选择安排在下一个维护时段中应用修改，或立即应用修改。进行选择，然后选择 Modify instance (修改实例)。

> **Note**
>
> 如果您选择立即应用修改，则该操作也将同时应用等待修改队列中的所有更改。如果任何待处理修改需要停机，选择此选项可导致意外停机。

使用 AWS CLI

```
aws docdb modify-db-instance --db-instance-identifier <db_instance_name> ca-
certificate-identifier rds-ca-2019 <--apply-immediately | --no-apply-immediately>
```

如果您选择 `--no-apply-immediately`，则将在集群的下一个维护时段内应用所做的更改。

如果我从快照还原或执行时间点还原，它会有新的服务器证书吗？

如果在 2024 年 8 月之后还原快照或执行时间点还原，则创建的新集群将使用新 CA 证书。

如果我在从 Mac OS 直接连接到我的 Amazon DocumentDB 集群时遇到问题，该怎么办？

Mac OS 已更新对可信证书的要求。现在，可信证书的有效期必须不超过 397 天（请参阅 https://support.apple.com/en-us/HT211025）。

> **Note**
>
> 在较新版本的 Mac OS 中会出现这种限制。

Amazon DocumentDB 实例证书的有效期超过四年，超过 Mac 操作系统的最大有效期。要从运行 Mac OS 的计算机直接连接到 Amazon DocumentDB 集群，您必须在创建 TLS 连接时允许使用无效证书。在这种情况下，无效证书是指其有效期超过 397 天。在连接到 Amazon DocumentDB 集群时，您应在允许使用无效证书之前了解风险。

要使用从 Mac OS 连接到 Amazon DocumentDB 集群 AWS CLI，请使用参数。`tlsAllowInvalidCertificates`

```
mongo --tls --host <hostname> --username <username> --password <password> --port 27017
 --tlsAllowInvalidCertificates
```

Amazon DocumentDB 中的合规性验证

作为多个 AWS 合规性计划的一部分，第三方审计员将评估 Amazon DocumentDB 的安全性和合规性，包括以下内容：

- 系统和组织控制（SOC）1、2 和 3。有关更多信息，请参阅 SOC。

- 联邦风险与授权管理项目 (FedRAMP)。有关更多信息，请参阅合规性计划范围内的AWS 服务。
- 支付卡行业数据安全标准（PCI DSS）。有关更多信息，请参阅 PCI DSS。
- ISO 9001、27001、27017 和 27018。有关更多信息，请参阅 ISO 认证。
- 《健康保险流通与责任法案》商业伙伴协议（HIPAA BAA）。有关更多信息，请参阅 HIPAA 合规性。

AWS 在 “按合规计划划分的范围内的 AWS 服务” 中提供了经常更新的特定合规计划范围内的AWS 服务列表。

Third-party 审计报告可供您使用下载 AWS Artifact。有关更多信息，请参阅在 Artifac t 中 AWS 下载报告。

有关 AWS 合规计划的更多信息，请参阅AWS 合规计划。

您在使用 Amazon DocumentDB 时的合规性责任由您数据的敏感性、您组织的合规性目标以及适用的法律法规决定。如果您在使用 Amazon DocumentDB 时必须符合 HIPAA 或 PCI 等标准， AWS 请提供资源来帮助：

- AWS 合规资源 — 可能适用于您所在行业和所在地区的工作手册和指南的集合。
- 安全与合规性快速入门指南 — 部署指南，讨论架构注意事项，并提供在上部署以安全性和合规性为重点的基准环境的步骤。 AWS
- AWS Config — 一项评估您的资源配置在多大程度上符合内部实践、行业准则和法规的服务。
- AWS Sec@@ urity Hub — 全面了解您的安全状态 AWS ，可帮助您检查自己是否符合安全行业标准和最佳实践。
- HIPAA 安全与合规架构白皮书 — 一份描述公司如何使用来创建应用程序的白皮书。 AWS HIPAA-compliant

Amazon DocumentDB 中的故障恢复能力

AWS 全球基础设施是围绕 AWS 区域 可用区构建的。 AWS 区域 提供多个物理分隔和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络连接。利用可用区，您可以设计和操作在可用区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础架构相比，可用区具有更高的可用性、容错性和可扩展性。

只有在至少两个可用区中包含至少两个子网的 Amazon VPC 中，才能创建 Amazon DocumentDB 集群。通过跨至少两个可用区分配您的集群实例，Amazon DocumentDB 可帮助确保集群中有可用的实

例，即使出现不太可能发生的可用区故障。Amazon DocumentDB 集群的集群卷始终跨三个可用区提供持久性存储，数据丢失的可能性更小。

有关 AWS 区域 和可用区的更多信息，请参阅AWS 全球基础设施。

除了 AWS 全球基础设施外，Amazon DocumentDB 还提供多项功能来帮助支持您的数据弹性和备份需求。

Fault-tolerant 和自我修复存储

存储卷的每 10 GB 部分都以六种方式跨三个可用区复制。Amazon DocumentDB 使用容错存储，这种容错存储在不影响数据库写入可用性情况下透明地处理多达两个数据副本的丢失，及在不影响读取可用性情况下透明地处理多达三个数据副本的丢失。Amazon DocumentDB 存储还具有自我修复性；数据块和磁盘会不断扫描错误并自动更换。

手动备份和还原

Amazon DocumentDB 提供了创建集群的完整备份以实现长期保留和恢复的功能。有关更多信息，请参阅 在 Amazon DocumentDB 中进行备份和还原。

Point-in-time 恢复

Point-in-time 恢复有助于保护您的 Amazon DocumentDB 集群免受意外写入或删除操作的影响。使用时间点恢复，您不必担心创建、维护或计划按需备份。有关更多信息，请参阅 还原到某个时间点。

Amazon DocumentDB 中的基础设施安全性

作为一项托管服务，Amazon DocumentDB 受到 AWS 全球网络安全的保护。有关 AWS 安全服务以及如何 AWS 保护基础设施的信息，请参阅AWS 云安全。要使用基础设施安全的最佳实践来设计您的 AWS 环境，请参阅 S AWS ecurity Pillar Well-Architected Fram ework 中的基础设施保护。

您可以使用 AWS 已发布的 API 调用通过网络访问亚马逊文档数据库。客户端必须支持以下内容：

- 传输层安全性协议（TLS）。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密 (PFS) 的密码套件，例如 DHE（短暂的）或 ECDHE（椭圆曲线短暂的 Diffie-Hellman）。 Diffie-Hellman大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

您可以从任何网络位置调用这些 API 操作。您可以使用 Amazon DocumentDB 策略来控制来自特定 Amazon Virtual Private Cloud (Amazon VPC) 终端节点或特定 VPC 的访问。实际上，这可以将对给定 Amazon DocumentDB 资源的网络访问与网络中的 AWS 特定 VPC 隔离开来。

> **Note**
>
> Amazon DocumentDB 不支持基于资源的访问策略。

亚马逊 DocumentDB API 和接口 VPC 终端节点 (AWS PrivateLink)

> **Note**
>
> 亚马逊 DocumentDB 弹性集群不支持 VP AWS PrivateLink C 终端节点。

您可以通过创建接口 VPC 端点 在 VPC 和 Amazon DocumentDB API 端点之间建立私有连接。接口端点由提供支持 AWS PrivateLink。

虽然基于 Amazon DocumentDB 实例的集群不需要接口 VPC 终端节点连接，但您无需互联网网关、AWS PrivateLink NAT 设备、VPN 连接或连接即可私密访问 Amazon DocumentDB API 操作。 Direct Connect VPC 中的 Amazon DocumentDB 实例不需要公有 IP 地址即可与 Amazon DocumentDB API 端点进行通信，进而启动、修改或终止数据库实例和数据库集群。您的 Amazon DocumentDB 实例也不需要公有 IP 地址即可使用任何可用的 Amazon DocumentDB API 操作。您的 VPC 和 Amazon DocumentDB 之间的流量不会脱离 Amazon 网络。

每个接口端点均由子网中的一个或多个弹性网络接口表示。有关更多信息，请参阅《Amazon EC2 用户指南》中的弹性网络接口。

有关 VPC 终端节点的更多信息，请参阅 Amazon Virtual Private Cloud (AWS PrivateLink) 用户指南中的AWS 服务 使用接口 VPC 终端节点访问和。有关亚马逊 DocumentDB 操作的更多信息，请参阅亚马逊 Docum entDB API 参考。

主题

- VPC 端点注意事项
- 区域可用性
- 为 Amazon DocumentDB API 创建接口 VPC 端点
- 为 Amazon DocumentDB API 创建 VPC 端点策略

VPC 端点注意事项

在为 Amazon DocumentDB API 端点设置接口 VPC 端点之前，请务必查看《Amazon Virtual Private Cloud (AWS PrivateLink) 用户指南》中的接口端点先决条件。

可以从 VPC 使用 AWS PrivateLink获取所有与管理 Amazon DocumentDB 资源相关的所有 Amazon DocumentDB API 操作。

Amazon DocumentDB API 端点支持 VPC 端点策略。默认情况下，允许通过端点对 Amazon DocumentDB 操作进行完全访问。有关更多信息，请参阅《Amazon Virtual Private Cloud (AWS PrivateLink) 用户指南》中的使用端点策略控制对 VPC 端点的访问。

区域可用性

亚马逊 DocumentDB API 目前支持以下方面的 VPC 终端节点： AWS 区域

- 美国东部（俄亥俄州）
- 美国东部（弗吉尼亚州北部）
- 美国西部（俄勒冈州）
- 非洲（开普敦）
- 亚太地区（香港）
- Asia Pacific (Mumbai)
- 亚太地区（海得拉巴）
- 亚太地区（大阪）
- 亚太地区（首尔）
- 亚太地区（新加坡）
- 亚太地区（悉尼）
- 亚太地区（东京）
- 加拿大（中部）
- 中国（北京）
- 中国（宁夏）
- 欧洲地区（法兰克福）
- 欧洲地区（爱尔兰）
- 欧洲地区（伦敦）

- Europe (Paris)
- 欧洲（西班牙）
- 欧洲地区（米兰）
- 中东（阿联酋）：
- 南美洲（圣保罗）
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)

为 Amazon DocumentDB API 创建接口 VPC 端点

您可以使用亚马逊 VPC 控制台或 AWS Command Line Interface ()AWS CLI为亚马逊 DocumentDB API 创建 VPC 终端节点。有关更多信息，请参阅 Amazon Vir tual Private Cloud (AWS PrivateLink) 用户指南中的 AWS 服务 使用接口 VPC 终端节点访问和。

使用服务名称 `com.amazonaws.region.rds` 为 Amazon DocumentDB API 创建 VPC 端点。

中国除外 AWS 区域 ，如果您为终端节点启用私有 DNS，则可以使用其默认 DNS 名称 AWS 区域，使用 VPC 终端节点向亚马逊 DocumentDB 发出 API 请求，例如 rds.us-east-1.amazonaws.com。对于中国（北京）和中国（宁夏 AWS 区域），您可以分别使用 rds-api.cn-north-1.amazonaws.com .cn 和 rds-api.cn-northwest-1.amazonaws.com .cn 向 VPC 终端节点发出 API 请求。

有关更多信息，请参阅 Amazon Vir tual Private Cloud (AWS PrivateLink) 用户指南中的 AWS 服务 使用接口 VPC 终端节点访问和。

为 Amazon DocumentDB API 创建 VPC 端点策略

您可以为 VPC 端点附加端点策略，以控制对 Amazon DocumentDB API 的访问。该策略指定以下信息：

- 可执行操作的主体。
- 可执行的操作。
- 可对其执行操作的资源。

有关更多信息，请参阅《Amazon Virtual Private Cloud (AWS PrivateLink) 用户指南》中的使用端点策略控制对 VPC 端点的访问。

示例：Amazon DocumentDB API 操作的 VPC 端点策略

下面是用于 Amazon DocumentDB API 的端点策略示例。当附加到端点时，此策略会向所有委托人授予对列出的针对所有资源的 Amazon DocumentDB API 操作的访问权限。

```
{
"Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "docdb:CreateDBInstance",
            "docdb:ModifyDBInstance",
            "docdb:CreateDBSnapshot"
         ],
         "Resource":"*"
      }
   ]
}
```

示例：拒绝来自指定 AWS 账户的所有访问的 VPC 终端节点策略

以下 VPC 终端节点策略拒绝 AWS 账户 123456789012 使用该终端节点访问所有资源。此策略允许来自其他账户的所有操作。

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
   ]
}
```

Amazon DocumentDB 的安全最佳实践

为了获得最佳安全实践，您必须使用 AWS Identity and Access Management (IAM) 账户来控制对亚马逊 DocumentDB API 操作的访问权限，尤其是创建、修改或删除亚马逊 DocumentDB 资源的操作。此类资源包括集群、安全组和参数组。此外，您还必须使用 IAM 来控制执行常见管理任务的操作，例如备份和还原集群。创建 IAM 角色时，请采用最小权限原则。

- 使用基于角色的访问控制强制执行最低权限。
- 为每个管理 Amazon DocumentDB 资源的人员分配个人 IAM 账户。请勿使用 AWS 账户 根用户来管理 Amazon DocumentDB 资源。为每个人（包括您自己）创建一个 IAM 用户。
- 授予每位用户履行其职责所需的最小权限集。
- 使用 IAM 组有效地管理适用于多个用户的权限。有关 IAM 的更多信息，请参阅 IAM 用户指南。有关 IAM 最佳实践的信息，请参阅 IAM 最佳实践。
- 定期轮换 IAM 凭证。
- 将 S AWS ecrets Manager 配置为自动轮换 Amazon DocumentDB 的密钥。有关更多信息，请参阅 S AWS ecrets Manager 用户指南中的轮换您的 Sec rets Manager 密钥和轮换亚马逊 DocumentDB 的AWS 密钥。
- 使用传输层安全性协议（TLS）和静态加密来加密您的数据。

审核 Amazon DocumentDB 事件

使用 Amazon DocumentDB（与 MongoDB 兼容），您可以审核集群中执行的事件。记录的事件的示例包括成功和失败的身份验证尝试、删除数据库中的集合或创建索引。默认情况下，在 Amazon DocumentDB 上禁用审计，并要求您选择使用该功能。

启用审计后，Amazon DocumentDB 会将数据定义语言 (DDL)、数据操纵语言 (DML)、身份验证、授权和用户管理事件记录到亚马逊日志。 CloudWatch 启用审计后，Amazon DocumentDB 会将集群的审计记录（JSON 文档）导出到亚马逊 CloudWatch 日志。您可以使用亚马逊 CloudWatch 日志来分析、监控和存档您的 Amazon DocumentDB 审计事件。

尽管 Amazon DocumentDB 不收取额外费用来启用审计，但您需要为日志的 CloudWatch 使用支付标准费率。有关 CloudWatch 日志定价的信息，请参阅 Amazon CloudWatch 定价。

Amazon DocumentDB 审计功能与监控的服务资源使用情况明显不同。 AWS CloudTrail CloudTrail 记录使用 AWS Command Line Interface (AWS CLI) 或 AWS 管理控制台 对集群、实例、参数组和快照等资源执行的操作。默认情况下，带的资源审计 CloudTrail 处于开启状态，无法禁用。Amazon

DocumentDB 审计功能是一种可选功能。它记录在集群中对对象（例如数据库、集合、索引和用户）采取的操作。

主题

- 支持的事件
- 启用审核
- 禁用审核
- 访问审核事件
- 筛选 DML 审核事件

支持的事件

Amazon DocumentDB 审核支持以下事件类别：

- 数据定义语言 (DDL)：包括数据库管理操作、连接、用户管理和授权。
- 数据操作语言读取事件（DML 读取）：包括 `find()` 和各种聚合运算符、算术运算符、布尔运算符和其他读取查询运算符。
- 数据操纵语言写入事件（DML 写入）：包括 `insert()`, `update()`, `delete()`, 和 `bulkWrite()` 运算符

事件类型如下所示。

事件类型	类别	说明
`authCheck`	Authorization	结果代码 0：成功
		结果代码 13：未经授权的执行操作的尝试。
`authenticate`	Connection	对新连接进行的成功或失败的身份验证尝试。
`auditConfigure`	DDL	审核筛选配置。

事件类型	类别	说明
createDatabase	DDL	创建新数据库。
createCollection	DDL	在数据库中创建新集合。
createIndex	DDL	在集合中创建新索引。
dropCollection	DDL	删除数据库中的集合。
dropDatabase	DDL	删除数据库。
dropIndex	DDL	删除集合中的索引。
modifyChangeStreams	DDL	已创建变更流。
renameCollection	DDL	删除数据库中的集合。
createRole	角色管理	创建角色。
dropAllRolesFromDatabase	角色管理	删除数据库中的所有角色。
dropRole	角色管理	删除角色。
grantPrivilegesToRole	角色管理	授予角色权限。
grantRolesToRole	角色管理	向用户定义角色授予角色。
revokePrivilegesFromRole	角色管理	撤消角色权限。
revokeRolesFromRole	角色管理	撤消用户定义角色限制。
updateRole	角色管理	更新角色。

事件类型	类别	说明
createUser	用户管理	创建新用户。
dropAllUsersFromDatabase	用户管理	删除数据库中的所有用户。
dropUser	用户管理	删除现有用户。
grantRolesToUser	用户管理	授予用户角色。
revokeRolesFromUser	用户管理	撤消用户角色。
updateUser	UserManagement	更新现有用户。
insert	DML 写入	将一个或多个文档插入到集合中。
delete	DML 写入	从集合中删除一个或多个文档。
update	DML 写入	修改集合中的一个或多个现有文档。
bulkWrite	DML 写入	通过控制执行顺序执行多个写入操作。
setAuditConfig	DML 写入	为 DML 审核设置新的筛选器。
count	DML 读取	返回与集合或视图的 find () 查询相匹配的文档数量。
countDocuments	DML 读取	返回与集合或视图查询相匹配的文档数量。

事件类型	类别	说明
find	DML 读取	选择集合或视图中的文档，然后将光标返回到所选文档。
getAuditConfig	DML 读取	检索 DML 审核的当前筛选器。
findAndModify	DML 读取 和 DML 写入	修改并返回单个文档。
findOneAndDelete	DML 读取 和 DML 写入	根据筛选和排序标准删除单个文档，返回已删除的文档。
findOneAndReplace	DML 读取 和 DML 写入	根据指定的筛选器替换单个文档。
findOneAndUpdate	DML 读取 和 DML 写入	根据筛选和排序标准更新单个文档。
aggregate	DML 读取 和 DML 写入	支持聚合管道中的 API。
distinct	DML 读取	在单个集合或视图中查找指定字段的不同值，并在数组中返回结果。

> **Note**
>
> DML 事件文档参数字段中的值的大小限制为 1KB。如果该值超过 1KB，Amazon DocumentDB 将截断该值。

> **Note**
>
> 目前不审核 TTL 删除事件。

启用审核

在集群上启用审核是一个两步过程。确保两个步骤都已完成，否则审核日志将不会发送到 CloudWatch 日志。

步骤 1：启用 audit_logs 集群参数

要启用审核，您需要修改参数组中的 `audit_logs` 参数。`audit_logs` 是要记录的事件列表，以逗号分隔。事件必须以小写形式指定，列表元素之间不应有空格。

您可以为参数组设置以下值：

值	说明	
`ddl`	设置此设置将启用对 DDL 事件的审计，例如 createDatabase、DropDatabase、CreateCollection、DropCollection、createIndex、authCheck、身份验证、创建用户、DropUser、授权、撤销、更新用户和删除 RolesToUser RolesFromUser AllUsersFromDatabase	
`dml_read`	设置此项将启用对 DML 读取事件的审计，例如 find、sort	

值	说明	
	count、distinct、group、projecta、unwind、geoNear、geoIntersects、geoWithin 和其他 MongoDB 读取查询运算符。	
dml_write	设置此项将启用对 DML 写入事件的审核，例如 insert()、update()、delete() 和 bulkWrite()	
all	设置此项将启用对数据库事件的审核，例如读取查询、写入查询、数据库操作和管理员操作。	
none	设置此项将禁用审核	

值	说明	
enabled（传统）	这是一个等同于“ddl”的传统参数设置。设置此设置将启用对 DDL 事件的审计，例如 createDatabase、DropDatabase、CreateCollection、DropCollection、createIndex、authCheck、身份验证、创建用户、DropUser、授权、撤销、更新用户和丢弃。RolesToUser RolesFromUser AllUsersFromDatabase我们不建议使用此设置，因为它是传统设置。	
disabled （传统）	这是一个等同于“无”的传统参数设置。我们不建议使用此设置，因为它是传统设置。	

> **Note**
>
> audit_logs 集群参数的默认值为 none（传统“disabled”）。

您也可以组合使用上述值。

值	说明	
`ddl, dml_read`	设置此项将启用对 DDL 事件和 DML 读取事件的审核。	
`ddl, dml_write`	设置此项将启用对 DDL 事件和 DML 写入的审核	
`dml_read, dml_write`	设置此项将启用对所有 DML 事件的审核	

> **Note**
>
> 您无法修改默认参数组。

有关更多信息，请参阅下列内容：

- 创建 Amazon DocumentDB 集群参数组

  创建自定义参数组后，通过将 `audit_logs` 参数值更改为 `all` 来修改它。
- 修改 Amazon DocumentDB 集群参数组

步骤 2：启用 Amazon CloudWatch 日志导出

当`audit_logs`集群参数的值为`enabled`、`ddldml_readdml_write`、或时，您还必须启用 Amazon DocumentDB 以将日志导出到亚马逊。 CloudWatch如果您省略了这两个步骤中的任何一个，则审核日志将不会发送到。 CloudWatch

在创建集群、执行时间点恢复或还原快照时，您可以按照以下步骤启用 CloudWatch 日志。

Using the AWS 管理控制台

要启用 Amazon DocumentDB CloudWatch 使用控制台将日志导出到，请参阅以下主题：

- 创建集群时：在 使用创建集群和主实例 AWS 管理控制台中，请参阅创建集群：其他配置（步骤 5，日志导出）
- 修改现有集群时：修改 Amazon DocumentDB 集群
- 执行集群快照还原时：从集群快照还原
- 执行时间点还原时：还原到某个时间点

Using the AWS CLI

在创建新集群时启用审计日志

以下代码创建集群sample-cluster并启用 CloudWatch 审计日志。

Example

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

在修改现有集群时启用审计日志

以下代码修改集群sample-cluster并启用 CloudWatch 审计日志。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster \
   --db-cluster-identifier sample-cluster \
   --cloudwatch-logs-export-configuration '{"EnableLogTypes":["audit"]}'
```

对于 Windows：

```
aws docdb modify-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --cloudwatch-logs-export-configuration '{"EnableLogTypes":["audit"]}'
```

这些操作的输出将类似于下文（JSON 格式）。

```
{
    "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"
    }
}
```

禁用审核

您可以通过禁用 CloudWatch 日志导出和禁用`audit_logs`参数来禁用审计。

禁用 CloudWatch 日志导出

您可以使用 AWS 管理控制台 或禁用导出审核日志 AWS CLI。

Using the AWS 管理控制台

以下过程使用禁用 Amazon DocumentDB AWS 管理控制台 将日志导出到。 CloudWatch

禁用审计日志

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。然后，选择要禁用导出日志的集群名称左侧的按钮。
3. 选择 Actions (操作)，然后选择 Modify (修改)。
4. 向下滚动到 Log exports (日志导出) 部分并选择 Disabled (已禁用)。
5. 选择继续。
6. 检查更改，然后选择何时将该更改应用到集群。
   - Apply during the next scheduled maintenance window (在下一个计划的维护时段内应用)
   - Apply immediately (立即应用)

7. 选择修改集群。

Using the AWS CLI

以下代码修改集群sample-cluster并禁用 CloudWatch 审计日志。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster \
   --db-cluster-identifier sample-cluster \
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["audit"]}'
```

对于 Windows：

```
aws docdb modify-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["audit"]}'
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "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"
            }
        ]
    }
}
```

禁用 audit_logs 参数

要禁用集群的 audit_logs 参数，您可以修改集群，使其使用 audit_logs 参数值为 disabled 的参数组。或者，您可以修改集群的参数组中的 audit_logs 参数值，使其为 disabled。

有关更多信息，请参阅以下主题：

- 修改 Amazon DocumentDB 集群
- 修改 Amazon DocumentDB 集群参数组

访问审核事件

使用以下步骤访问您在 Amazon 上的审计事件 CloudWatch。

1. 打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 确保您与 Amazon DocumentDB 集群位于同一区域。
3. 在导航窗格中，选择日志。
4. 要查找集群的审核日志，请从列表中找到并选择 **/aws/docdb/*yourClusterName*/audit**。

此时，每个实例名称的下方将显示该实例的审计事件。

筛选 DML 审核事件

开始使用 DML 审核筛选

可以在 DML 审计事件写入 Amazon CloudWatch 之前对其进行过滤。要利用此功能，必须启用审核日志和 DML 日志记录。Amazon DocumentDB 支持 `atype`、`command`、`user`、`namespace` 和 `auditAuthorizationSuccess` 筛选。

> **Note**
>
> 不筛选 DDL 事件。

可以在 `db.adminCommand( { command } )` 操作中使用 `setAuditConfig`、`filter` 和 `auditAuthorizationSuccess` 参数指定审核筛选器，随时启用审核筛选：

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter:
         {
            //filter conditions
         },
      auditAuthorizationSuccess: true | false
   }
)
```

还可以通过运行以下命令检索审核筛选器设置：

```
db.admin.runCommand( { getAuditConfig: 1})
```

安全要求

在设置或列出 DML 审计过滤器admindb时，只有 users/roles 具有特权操作的数据库auditConfigure才能执行上述命令。您可以使用 [`clusterAdmin`、`hostManager`、`root`] 中的内置角色之一，也可以创建具有 `auditConfigure` 权限的自定义角色。以下是使用具有 `auditConfigure` 权限的现有角色的示例，以及使用自定义角色的示例。

具有内置角色的用户：

```
use admin
db.createUser(
  {
    user: "myClusterAdmin",
    pwd: "password123",
    roles: [ { role: "clusterAdmin", db: "admin" } ]
  }
)
```

具有自定义角色的用户：

```
use admin
db.createRole(
   {
     role: "myRole",
     privileges: [
       { resource: { cluster: true }, actions: [ "auditConfigure" ] }
     ],
     roles: []
   }
)
db.createUser(
  {
    user: "myUser",
    pwd: "myPassword",
    roles: [ { role: "myRole", db: "admin" } ]
  }
)
```

筛选使用案例

示例：按命令筛选事件

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
        "$and": [
         {
            "param.command":
               {
```

```
                  $in: [ "find","count", "insert", "delete", "update",
 "findandmodify" ]
               }
         }
         ]
      },
      auditAuthorizationSuccess: true
   }
)
```

示例：按用户名筛选事件

在此示例中，只有用户“myUser”将进行日志记录：

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
      "$and": [
         {
            "param.user":
               {
                  $in: [ "myUser" ]
               }
         }
         ]},
      auditAuthorizationSuccess: true})
```

示例：按 **atype** 筛选

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {atype: "authCheck"},
      auditAuthorizationSuccess: true
   })
```

> **Note**
>
> 所有 DML 日志均有 atype 类型的 authCheck。只有 DDL 具有不同的 atype。如果您authCheck在中输入了以外的值filter，它将不会生成 DML 登录 CloudWatch。

示例：使用由运算符连接的多个筛选器进行筛选

```
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**
>
> 顶层仅支持 $and、$or 和 $nor。不支持任何其他运算符，它们会导致出错。

示例：通过 **auditAuthorizationSuccess** 按事件筛选

在此筛选器中，所有成功通过授权的命令都不会进行记录：

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {},
      auditAuthorizationSuccess: false
   }
)
```

示例：使用 **$in** 和 **$nin** 条件进行筛选

同时在 $in 和 $nin 中使用时，该命令不会进行日志记录，因为两个条件之间有一个隐含的“和”。在此示例中，正则表达式将阻止 find 命令，因此不会记录任何内容：

```
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})
```

示例：按 **namespace** 筛选

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {
      "$and": [
         {
            "param.ns":
               {
                  $in: [ "test.foo" ]
               }
         }
```

```
        ]},
      auditAuthorizationSuccess: true})
```

示例：重置为默认筛选器

重置为默认值意味着每个 DML 审核事件都将记录。要将筛选重置为默认值，请运行以下命令：

```
db.admin.runCommand(
   {
      setAuditConfig: 1,
      filter: {},
      auditAuthorizationSuccess: true
   }
)
```

Amazon VPC 和 Amazon DocumentDB

亚马逊虚拟私有云（亚马逊 VPC）使您可以将 AWS 资源（例如亚马逊 DocumentDB 实例）启动到虚拟私有云 (VPC) 中。

使用 VPC 时，您的虚拟联网环境完全由您控制。您可以选择自己的 IP 地址范围、创建子网以及配置路由和访问控制列表。在 VPC 中运行集群不会产生额外费用。

账户具有原定设置 VPC。除非您另行指定，否则所有新集群都将在默认 VPC 中创建。

主题

- VPC 中的 DocumentDB 集群
- 访问 VPC 中的 Amazon DocumentDB 集群
- 创建用于文档数据库集群的 IPv4-only VPC
- 创建双堆栈 VPC 以用于 DocumentDB 集群

接下来，您可以查找与 Amazon DocumentDB 集群相关的 VPC 功能的讨论。有关 Amazon VPC 的更多信息，请参阅 Amazon VPC 用户指南。

VPC 中的 DocumentDB 集群

您的 Amazon DocumentDB 集群位于虚拟私有云（VPC）中。VPC 是一种在逻辑上与中的其他虚拟网络隔离的虚拟网络 AWS Cloud。Amazon VPC 使您可以在 VPC 中启动 AWS 资源，例如亚马逊文档

数据库集群或亚马逊 EC2 实例。VPC 可以是您的账户附带的默认 VPC，也可以是您创建的 VPC。所有 VPC 都与您的 AWS 账户关联。

默认 VPC 具有可用来隔离 VPC 内资源的三个子网。默认 VPC 还具有一个互联网网关，可用来从 VPC 外部访问 VPC 内部的资源。

有关涉及 VPC 内外的 Amazon DocumentDB 集群的场景，请参阅 访问 VPC 中的 Amazon DocumentDB 集群。

主题

- 使用 VPC 中的集群
- 使用子网组
- Amazon DocumentDB IP 寻址
- 在 VPC 中创建集群

使用 VPC 中的集群

下面是有关使用 VPC 中的集群的一些提示：

- 您的 VPC 必须至少有两个子网。这些子网必须位于您要部署集群 AWS 区域 的两个不同可用区中。子网是 VPC 的 IP 地址范围段，您可以指定子网，利用子网并根据安全和操作需要对集群进行分组。
- 如果要让 VPC 中的集群实现公开访问，请确保开启 VPC 属性 `DNS hostnames` 和 `DNS resolution`。
- 您的 VPC 必须具有您创建的子网组。您可通过指定创建的子网来创建子网组。Amazon DocumentDB 选择子网及该子网组中的 IP 地址，以与集群中的主实例关联。主实例使用包含该子网的可用区。
- 您的 VPC 必须具有允许访问集群的 VPC 安全组。

  有关更多信息，请参阅 访问 VPC 中的 Amazon DocumentDB 集群。
- 每个子网必须包含足够大的 CIDR 数据块，以便在维护活动（包括失效转移和扩展计算）期间有可供 Amazon DocumentDB 使用的备用 IP 地址。例如，一个范围，比如 10.0.0。 0/24 和 10.0.1。 0/24 通常足够大。
- VPC 可以具有 `instance tenancy` 属性值 `default` 或 `dedicated`。所有默认 VPC 的 `instance tenancy` 属性设置为“default”，则默认的 VPC 可以支持任何实例类。

如果您选择将集群放在 `instance tenancy` 属性设置为 `dedicated` 的专用 VPC 中，则集群的实例类必须是已批准的 Amazon EC2 专用实例类型之一。例如，`r5.large` EC2 专用实例对应于 `db.r5.large` 实例类。有关 VPC 中实例租赁的信息，请参阅《Amazon Elastic Compute Cloud 用户指南》中的 Amazon EC2 实例。

有关可位于专用实例中的实例类型的更多信息，请参阅 Amazon EC2 定价页上的 Amazon EC2 专用实例。

> **Note**
>
> 当您为集群将 `instance tenancy` 属性设置为 `dedicated` 时，不能保证集群将在专属主机上运行。

使用子网组

子网 是您指定的用来根据安全和操作需要对资源进行分组的 VPC 的 IP 地址范围段。子网组是您在 VPC 中创建并随后指定给集群的子网（通常为私有子网）的集合。通过使用子网组，您可以在使用 AWS CLI 或 Amazon DocumentDB API 创建集群时指定特定的 VPC。如果您使用控制台，则可以选择要使用的 VPC 和子网组。

每个子网组应包含给定 AWS 区域中至少两个可用区的子网。在 VPC 中创建集群时，为其选择子网组。对于子网组，Amazon DocumentDB 选择子网及该子网中的 IP 地址，以与集群中的主实例关联。数据库使用包含该子网的可用区。DocumentDB 始终从具有可用 IP 地址空间的子网中分配 IP 地址。

子网组中的子网可以是公有子网或私有子网，这取决于您为网络访问控制列表（网络 ACL）和路由表设置的配置。对于希望可公开访问的集群，其子网组中的所有子网必须均为公有子网。如果与可公开访问的集群关联的子网从公有更改为私有，则可能会影响集群的可用性。

要创建支持双堆栈模式的子网组，请确保添加到该子网组的每个子网都有一个与之关联的互联网协议版本 6（IPv6）CIDR 数据块。有关更多信息，请参阅《Amazon Virtual Private Cloud 用户指南》中的 Amazon DocumentDB IP 寻址 和 VPC 的 IPv6 支持。

当 Amazon DocumentDB 在 VPC 中创建集群时，使用子网组中的 IP 地址将网络接口分配给集群。但是，我们强烈建议您使用域名系统（DNS）名称来连接到集群。之所以建议这样做，是因为底层 IP 地址在失效转移期间会发生变化。

> **Note**
>
> 对于在 VPC 中运行的每个集群，请确保子网组的每个子网中预留至少一个地址，以供 Amazon DocumentDB 用来执行恢复操作。

共享子网

您可以在共享 VPC 中创建集群。

使用共享 VPC 时需要记住的一些注意事项：

- 您可以将集群从共享 VPC 子网移动到非共享 VPC 子网，反之亦然。
- 共享 VPC 中的参与者必须在 VPC 中创建安全组，才能允许他们创建集群。
- 共享 VPC 中的拥有者和参与者可以使用 DocumentDB 查询访问数据库。但是，只有资源的创建者才能对该资源进行任何 API 调用。

Amazon DocumentDB IP 寻址

IP 地址使 VPC 中的资源能够相互通信以及与 Internet 上的资源进行通信。Amazon DocumentDB 同时支持 IPv4 和 IPv6 寻址协议。默认情况下，Amazon DocumentDB 和 Amazon VPC 使用 IPv4 寻址协议。您无法关闭这种行为。创建 VPC 时，请确保指定 IPv4 CIDR 块 (一系列私有 IPv4 地址)。您可以选择将 IPv6 CIDR 数据块分配给您的 VPC 和子网，并将来自该数据块的 IPv6 地址分配给子网中的集群。

> **Note**
>
> Dual-stack 只有亚马逊 DocumentDB 版本 4.0 和 5.0 支持模式（IPv6 寻址）。

对 IPv6 协议的支持扩展了支持的 IP 地址数量。通过使用 IPv6 协议，您可以确保有足够的可用地址来应对 Internet 的未来发展。新的和现有的 DocumentDB 资源可以在 VPC 内使用 IPv4 和 IPv6 地址。在应用程序不同部分使用的两个协议之间配置、保护和转换网络流量可能会产生运营开销。您可以对 Amazon DocumentDB 资源的 IPv6 协议进行标准化，以简化网络配置。

主题

- IPv4 地址
- IPv6 地址

- Dual-stack 模式

IPv4 地址

创建 VPC 时，您必须以 CIDR 块（如 `10.0.0.0/16`）的形式为 VPC 指定一个 IPv4 地址范围。子网组定义此 CIDR 数据块中可供集群使用的 IP 地址范围。这些 IP 地址可以是私有地址，也可以是公有地址。

私有 IPv4 地址是指无法通过 Internet 访问的 IP 地址。您可以使用私有 IPv4 地址在同一 VPC 中的集群和其他资源（如 Amazon EC2 实例）之间进行通信。每个集群都有一个用于在 VPC 中通信的私有 IP 地址。

公有 IP 地址是指可通过 Internet 访问的 IPv4 地址。您的 DocumentDB 集群不允许使用公有 IP 寻址。任何公有 IP 地址都应通过互联网网关和公有子网中的 EC2 进行解析。

要了解如何创建您可用于常见 Amazon DocumentDB 场景的 VPC（只有私有 IPv4 地址），请参阅 创建用于文档数据库集群的 IPv4-only VPC。

IPv6 地址

您可以选择向 VPC 和子网关联 IPv6 CIDR 块，然后将此块中的 IPv6 地址分配给 VPC 中的资源。每个 IPv6 地址都是全局唯一的。

我们将自动从 Amazon 的 IPv6 地址池中为您的 VPC 分配 IPv6 CIDR 块。您不能自行选择范围。

连接到 IPv6 地址时，请确保满足以下条件：

- 客户端配置为允许通过 IPv6 进行客户端到数据库的通信。
- 集群使用的 DocumentDB 安全组已正确配置，允许通过 IPv6 进行客户端到数据库的通信。
- 客户端操作系统堆栈允许 IPv6 地址上有流量，操作系统驱动程序和库已配置为选择正确的默认集群端点（IPv4 或 IPv6）。

有关 IPv6 的更多信息，请参阅《Amazon Virtual Private Cloud 用户指南》中的 IP 寻址。

Dual-stack 模式

当集群可以通过 IPv4 和 IPv6 寻址协议进行通信时，将在双堆栈模式下运行。因此，资源可以通过 IPv4 和/或 IPv6 与集群进行通信。DocumentDB 禁用互联网网关访问私有双堆栈模式集群的 IPv6 端点。DocumentDB 这样做是为了确保您的 IPv6 端点是私有端点，并且只能从您的 VPC 内部对其进行访问。

主题

- Dual-stack 模式和子网组
- 使用双堆栈模式集群
- 修改 IPv4-only 集群以使用双堆栈模式
- Dual-stack 模式区域和版本可用性
- 双堆栈网络集群的限制

Dual-stack 模式和子网组

要使用双堆栈模式，请确保与集群关联的子网组中的每个子网都具有与之关联的 IPv6 CIDR 数据块。您可以创建新的子网组或修改现有的子网组来满足此要求。当集群处于双堆栈模式后，客户端可以与其正常连接。确保准确配置客户端安全防火墙和 DocumentDB 集群安全组，以允许通过 IPv6 的流量。要进行连接，客户端需要使用集群的端点。客户端应用程序可以指定连接到数据库时首选哪种协议。在双堆栈模式下，集群会检测客户端的首选网络协议（IPv4 或 IPv6），并使用该协议进行连接。

如果某子网组因子网删除或 CIDR 断开关联而停止支持双堆栈模式，则与该子网组关联的集群存在网络状态不兼容的风险。此外，创建新的双堆栈模式集群时，您不能使用子网组。

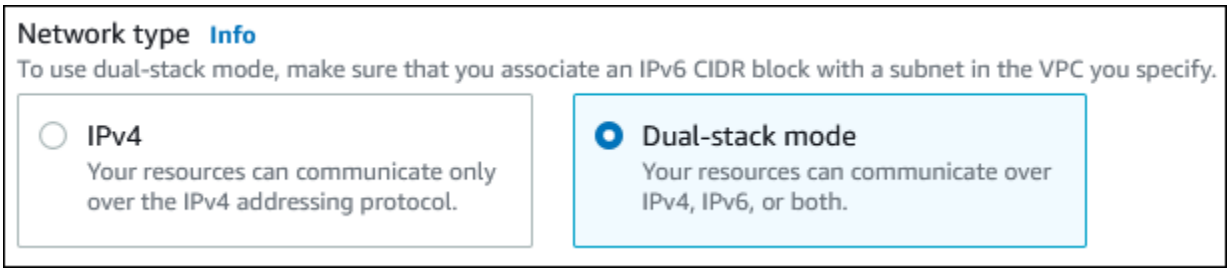
要使用确定子网组是否支持双栈模式 AWS 管理控制台，请在子网组的详细信息页面上查看网络类型。要使用确定子网组是否支持双栈模式 AWS CLI，请运行`describe-db-subnet-groups`命令并在输出`SupportedNetworkTypes`中查看。

只读副本被视为独立的集群，并且可以具有与主集群不同的网络类型。如果您更改只读副本的主集群的网络类型，则只读副本不会受到影响。当您恢复集群时，可以将其恢复为支持的任何网络类型。

使用双堆栈模式集群

创建或修改集群时，您可以指定双堆栈模式，以允许您的资源通过 IPv4 和/或 IPv6 与集群进行通信。

使用创建或修改集群时，可以在网络类型部分指定双堆栈模式。 AWS 管理控制台 下图显示了控制台中的网络类型部分：



使用创建或修改集群时，请将--network-type选项设置DUAL为使用双堆栈模式。 AWS CLI 当您使用 DocumentDB API 创建或修改集群时，请将 NetworkType 参数设置为 DUAL 以使用双堆栈模式。如果指定的 DocumentDB 引擎版本或子网组不支持双堆栈模式，则返回 NetworkTypeNotSupported 错误。

有关创建集群的更多信息，请参阅 创建 Amazon DocumentDB 集群。有关修改集群的更多信息，请参阅 修改 Amazon DocumentDB 集群。

要使用控制台确定集群是否处于双堆栈模式，请查看集群的连接和安全选项卡上的网络类型。

修改 IPv4-only 集群以使用双堆栈模式

您可以修改 IPv4-only 集群以使用双堆栈模式。为此，请更改集群的网络类型。

建议您在维护时段内更改 Amazon DocumentDB 集群的网络类型。您可以使用 modify-db-cluster 命令手动设置网络类型。

在修改集群以使用双堆栈模式之前，请确保其子网组支持双堆栈模式。如果与集群关联的子网组不支持双堆栈模式，请在修改集群时指定支持该模式的其他子网组。修改集群的子网组可能会导致停机。

如果在将集群更改为使用双堆栈模式之前修改集群的子网组，请确保该子网组在更改前后对集群有效。

我们建议您在仅将 NetworkType 参数设置为 DUAL 的情况下运行 ModifyDBCluster 调用，以将网络更改为双堆栈模式。在同一 API 调用中将其他参数与 NetworkType 一起添加可能会导致停机。要修改多个参数，请确保网络类型修改已成功完成，然后使用其他参数发送另一个 ModifyDBCluster 请求。

如果更改后无法连接到集群，请确保已准确配置客户端和数据库安全防火墙和路由表，以允许流量流向选定网络（IPv4 或 IPv6）上的数据库。您可能还需要修改操作系统参数、库或驱动程序才能使用 IPv6 地址进行连接。

修改 IPv4-only 群集以使用双堆栈模式

1. 修改子网组以支持双堆栈模式，或者创建支持双堆栈模式的子网组：

   a. 将 IPv6 CIDR 块与 VPC 关联。

      有关说明，请参阅《Amazon VPC 用户指南》中的将 CIDR 数据块添加到 VPC 或从中删除。

   b. 将 IPv6 CIDR 数据块附加到子网组中的所有子网。

      有关说明，请参阅《Amazon Virtual Private Cloud 用户指南》中的将 IPv6 CIDR 数据块添加到子网或从中删除。

c. 确认子网组支持双堆栈模式。

如果您使用的是 AWS 管理控制台，请选择子网组，并确保 “支持的网络类型” 值为 Dual。

如果您使用的是 AWS CLI，请运行describe-db-subnet-groups命令，并确保集群的SupportedNetworkType值为Dual。

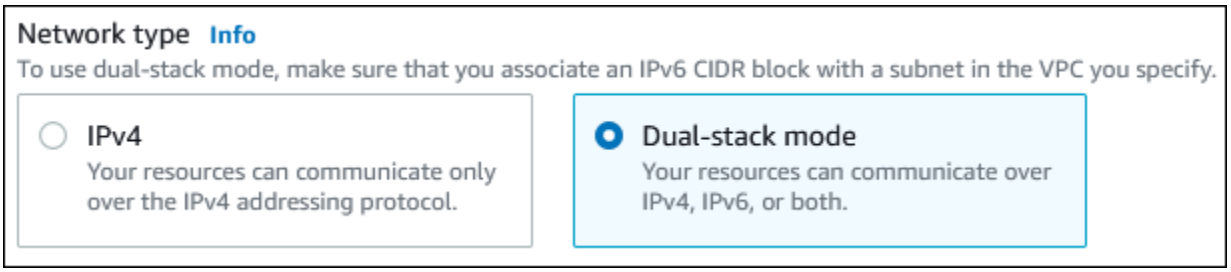
2. 修改与集群关联的安全组以允许与数据库建立 IPv6 连接，或者新建允许 IPv6 连接的安全组。

有关说明，请参阅《Amazon Virtual Private Cloud 用户指南》中的安全组规则。

3. 修改集群以支持双堆栈模式。为此，请将网络类型设置为Dual-stack 模式。

如果您使用控制台，请确保以下设置正确：

- 网络类型-Dual-stack 模式



- 子网组：在上一步中配置的子网组
- 安全组：在上一步中配置的安全组

如果您使用的是 AWS CLI，请确保以下设置正确：

- `--network-type` — `dual`
- `--db-subnet-group-name`：在上一步中配置的子网组
- `--vpc-security-group-ids`：在上一步中配置的 VPC 安全组

例如：

```
aws docdb modify-db-cluster --db-cluster-identifier <cluster-name> --network-type
 "DUAL"
```

4. 确认集群支持双堆栈模式。

如果您使用的是控制台，请为集群选择连接和安全选项卡，并确保网络类型值为Dual-stack 模式。

如果您使用的是 AWS CLI，请运行`describe-db-cluster`命令，并确保集群的`NetworkType`值为`dual`。

在集群端点上运行 dig 命令，以标识与其关联的 IPv6 地址：

```
dig <db-cluster-endpoint> AAAA
```

使用集群端点（而不是 IPv6 地址）连接到集群。

Dual-stack 模式区域和版本可用性

功能可用性和支持因 AWS 区域而异。

区域支持

Amazon DocumentDB 的双堆栈模式目前适用于所有商业版。 AWS 区域

版本支持

Dual-stack 亚马逊 DocumentDB 版本 4.0、5.0 和 8.0 支持模式。如果您无法在其中任何一个版本上访问双堆栈模式，请确保您的集群上运行的是最新的引擎补丁版本。

双堆栈网络集群的限制

以下限制适用于双堆栈网络集群：

- 集群不能仅使用 IPv6 协议。它们可以仅使用 IPv4，也可以同时使用 IPv4 和 IPv6 协议（双栈模式）。
- Amazon DocumentDB 不支持本机 IPv6 子网。
- 使用双堆栈模式的集群必须是私有的。它们不可公开访问。

在 VPC 中创建集群

以下过程帮助您在 VPC 中创建集群。要使用默认 VPC，可以从步骤 2 开始，并使用已经为您创建的 VPC 和子网组。您还可以根据需要创建额外的 VPC。

> **Note**
>
> 如果要让 VPC 中的集群实现公开访问，则必须通过启用 VPC 属性 `DNS hostnames` 和 `DNS resolution` 更新 VPC 的 DNS 信息。有关更新 VPC 实例的 DNS 的信息，请参阅《Amazon Virtual Private Cloud 用户指南》中的查看和更新 VPC 的 DNS 属性。

要在 VPC 中创建集群，请执行以下步骤：

- 第 1 步：创建 VPC
- 第 2 步：创建一个子网组
- 步骤 3：创建 VPC 安全组
- 步骤 4：在 VPC 中创建集群

第 1 步：创建 VPC

创建一个 VPC，该 VPC 具有的子网位于至少两个可用区内。您在创建子网组时将使用这些子网。如果您拥有默认 VPC，则系统会在该 AWS 区域中的每个可用区中自动为您创建子网。

有关更多信息，请参阅 创建用于文档数据库集群的 IPv4-only VPC，或者请参阅《Amazon Virtual Private Cloud 用户指南》中的创建 VPC。

第 2 步：创建一个子网组

子网组是您在 VPC 中创建并随后指定给集群的子网（通常为私有子网）的集合。子网组允许您在使用 AWS CLI 或 DocumentDB API 创建集群时指定特定 VPC。如果您使用 AWS 管理控制台，则只需选择要使用的 VPC 和子网即可。每个子网组必须至少包含 AWS 区域中至少两个可用区的一个子网。作为最佳实践，每个子网组应至少包含 AWS 区域中每个可用区的一个子网。

对于希望可公开访问的集群，子网组中的子网必须具有互联网网关。有关子网的互联网网关的更多信息，请参阅《Amazon Virtual Private Cloud 用户指南》中的使用互联网网关为 VPC 启用互联网访问。

> **Note**
>
> 本地区域的子网组只能有一个子网。

在 VPC 中创建集群时，您可以选择子网组。Amazon DocumentDB 选择要与集群关联的子网及该子网中的 IP 地址。如果不存在子网组，Amazon DocumentDB 会在您创建集群时创建默认子网

组。DocumentDB 使用该 IP 地址创建弹性网络接口并将其关联到您的集群。集群使用包含该子网的可用区。

在此步骤中，您创建一个子网组，然后添加为 VPC 创建的子网。

创建子网组

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择子网组。
3. 选择创建。
4. 对于名称，键入您的子网组的名称。
5. 对于描述，键入您的子网组的描述。
6. 在添加子网部分中，对于 VPC，选择默认 VPC 或您创建的 VPC。然后，从可用区中选择包含子网的可用区，并从子网中选择子网。
7. 选择创建。

   您的新子网组显示在 DocumentDB 控制台的子网组列表中。您可以选择该子网组，在窗口底部的详细信息窗格中查看详细信息，其中包括与该组关联的所有子网。

步骤 3：创建 VPC 安全组

在创建集群之前，请先创建要与之关联的 VPC 安全组。如果您不创建 VPC 安全组，则可以在创建集群时使用默认安全组。有关如何为您的集群创建安全组的说明，请参阅创建用于文档数据库集群的 IPv4-only VPC或参阅 Amazon Virtual Private Cloud 用户指南中的使用安全组控制 AWS 资源流量。

步骤 4：在 VPC 中创建集群

在此步骤中，创建一个集群，并使用您在之前的步骤中创建的 VPC 名称、子网组和 VPC 安全组。

> **Note**
>
> 如果要让 VPC 中的集群实现公开访问，则必须启用 VPC 属性 `DNS hostnames` 和 `DNS resolution`。有关更多信息，请参阅《Amazon Virtual Private Cloud 用户指南》中的查看和更新 VPC 的 DNS 属性。

有关如何创建集群的详细信息，请参阅 创建 Amazon DocumentDB 集群。

在连接部分中出现提示时，输入 VPC 名称、子网组和 VPC 安全组。

> **Note**
>
> DocumentDB 集群当前不支持更新 VPC。

访问 VPC 中的 Amazon DocumentDB 集群

Amazon DocumentDB 支持以下场景来访问 VPC 中的集群：

主题

- VPC 中的集群由同一 VPC 中的 Amazon EC2 实例访问
- VPC 中的集群由另一 VPC 中的 Amazon EC2 实例访问

VPC 中的集群由同一 VPC 中的 Amazon EC2 实例访问

VPC 中的集群的常见用途是与在相同 VPC 的 Amazon EC2 实例中运行的应用程序服务器共享数据。

要管理同一 VPC 中 EC2 实例与集群之间的访问，最简单方法如下所示：

- 为集群创建所属 VPC 安全组。此安全组可用于限制对集群的访问权限。例如，您可以为该安全组创建自定义规则。该规则可能允许使用您创建集群时分配给集群的端口以及您用来访问集群的 IP 地址（用于开发或其他目的）进行 TCP 访问。
- 创建您的 EC2 实例（Web 服务器和客户端）将位于其中的 VPC 安全组。如果需要，此安全组可允许使用 VPC 的路由表从 Internet 访问 EC2 实例。例如，您可设置此安全组的规则以允许通过端口 22 对 EC2 实例进行 TCP 访问。
- 在集群的安全组中创建自定义规则，允许从为 EC2 实例创建的安全组连接。这些规则将允许安全组的任何成员访问集群。

在单独的可用区中还有一个额外的公有和私有子网。DocumentDB 子网组需要位于至少两个可用区中的一个子网。额外的子网使将来很容易切换到多可用区集群部署。

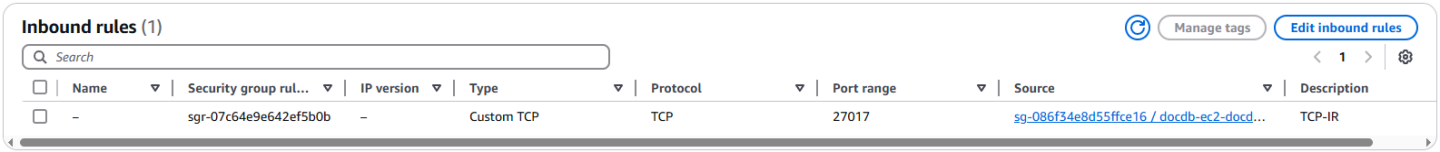
有关如何为此场景创建包含公有子网和私有子网的 VPC 的说明，请参阅 创建用于文档数据库集群的 IPv4-only VPC。

> **Tip**
>
> 创建集群时，您可以在 Amazon EC2 实例与 DocumentDB 集群之间自动设置网络连接。有关更多信息，请参阅 自动连接 Amazon EC2。

要在 VPC 安全组中创建允许来自其他安全组的连接的规则，请执行以下操作：

1. 登录 AWS 管理控制台 并打开 Amazon VPC 控制台，网址为https://console.aws.amazon.com/vpc。
2. 在导航窗格中，找到并选择安全组。
3. 选择或创建要允许另一个安全组的成员访问的安全组。这是要用于您的集群的安全组。选择 Inbound rules（入站规则）选项卡，然后选择 Edit inbound rules（编辑入站规则）。
4. 在 Edit inbound rules（编辑入站规则）页面上，选择 Add rule（添加规则）。
5. 对于类型，选择与创建集群时使用的端口相对应的条目，例如自定义 TCP。
6. 在源字段中，开始键入安全组的 ID，其中列出了匹配的安全组。选择您希望其成员可以访问此安全组保护的资源的安全组。在前面的方案中，这是您用于 EC2 实例的安全组。
7. 如果需要，可通过在源字段中创建类型为所有 TCP 的规则以及安全组，对 TCP 协议重复这些步骤。如果您打算使用 UDP 协议，请在 Source（源）框中创建 Type（类型）为 All UDP（所有 UDP）的规则以及安全组。
8. 选择保存规则。

以下屏幕显示了包含其来源的安全组的入站规则。



有关从 EC2 实例连接到集群的更多信息，请参阅 自动连接 Amazon EC2。

VPC 中的集群由另一 VPC 中的 Amazon EC2 实例访问

当您的集群与您用来访问其的 EC2 实例位于不同的 VPC 中时，可以使用 VPC 对等连接来访问该集群。

VPC 对等连接是两个 VPC 之间的网络连接，通过此连接，您可以使用私有 IP 地址在这两个 VPC 之间路由流量。这两个 VPC 中的资源可以彼此通信，就像它们在同一网络中一样。您可以在自己的 VPC

之间、与其他账户中的 VPC 或其他 AWS 账户中的 VPC 之间创建 VPC 对等连接。 AWS 区域要了解有关 VPC 对等的更多信息，请参阅 Amazon Virtual Private Cloud 用户指南 中的 VPC 对等。

创建用于文档数据库集群的 IPv4-only VPC

一种常见的场景包括虚拟私有云（VPC）中基于 Amazon VPC 服务的集群。例如，此 VPC 可以与在同一 VPC 中运行的服务或应用程序共享数据。在本主题中，针对此场景创建 VPC。

主题

- 步骤 1：创建包含私有子网和公有子网的 VPC
- 步骤 2：为公开应用程序创建 VPC 安全组
- 步骤 3：为私有集群创建 VPC 安全组
- 步骤 4：创建子网组
- 删除 VPC

您的集群只需对您的应用程序可用，而无需对公共互联网可用。因此，请创建包含公有子网和私有子网的 VPC。应用程序托管在公有子网中，因此能够访问公共互联网。集群托管在私有子网中。应用程序可以连接到集群，因为它托管在同一 VPC 内。但是，集群不可用于公共互联网，从而提高了安全性。

本主题中的此过程在单独的可用区中配置额外的公有子网和私有子网。此过程未使用这些子网。DocumentDB 子网组需要位于至少两个可用区中的一个子网。借助额外的子网，可以更轻松地配置多个 DocumentDB 实例。

本主题介绍为 Amazon DocumentDB 集群配置 VPC。有关 Amazon VPC 的更多信息，请参阅 Amazon VPC 用户指南。

> **Tip**
>
> 创建集群时，您可以在 Amazon EC2 实例与 DocumentDB 集群之间自动设置网络连接。网络配置类似于此场景中描述的配置。有关更多信息，请参阅 自动连接 Amazon EC2。

步骤 1：创建包含私有子网和公有子网的 VPC

使用以下步骤创建包含公有和私有子网的 VPC。

创建 VPC 和子网

1. 打开位于 https://console.aws.amazon.com/vpc 的 Amazon VPC 控制台。

2. 在的右上角 AWS 管理控制台，选择要在其中创建 VPC 的区域。此示例使用 美国西部（俄勒冈）区域。
3. 在左上角，选择 VPC Dashboard（VPC 控制面板）。要开始创建 VPC，请选择 Create VPC（创建 VPC）。
4. 对于 VPC Settings（VPC 设置）下的 Resources to create（要创建的资源），选择 VPC and more（VPC 及更多）。
5. 对于 VPC settings（VPC 设置），请设置以下值：
   - 名称标签自动生成：**example**
   - IPv4 CIDR 数据块：**10.0.0.0/16**
   - IPv6 CIDR 数据块：无 IPv6 CIDR 数据块
   - 租赁：默认值
   - 可用区（AZ）的数量：2
   - 自定义可用区：保留默认值
   - 公有子网的数量：2
   - 私有子网的数量：2
   - 自定义子网 CIDR 数据块：保留默认值
   - NAT 网关（$）：无
   - VPC 端点：无
   - DNS 选项：保留默认值
6. 选择创建 VPC。

步骤 2：为公开应用程序创建 VPC 安全组

接下来创建安全组以便进行公共访问。要连接到 VPC 中的公有 EC2 实例，请将入站规则添加到 VPC 安全组。这些规则允许流量从互联网进行连接。

创建 VPC 安全组

1. 打开位于 https://console.aws.amazon.com/vpc 的 Amazon VPC 控制台。
2. 依次选择 VPC 控制面板)、安全组和创建安全组。
3. 在创建安全组页面上，设置以下值：
   - 安全组名称：**example-securitygroup**

- 描述：**Application security group**
- VPC：选择您之前创建的 VPC，例如：vpc-example。

4. 将入站规则添加到安全组。

   a. 确定要用来通过 Secure Shell（SSH）连接到 VPC 中的 EC2 实例的 IP 地址。要确定您的公有 IP 地址，可以在不同的浏览器窗口或选项卡中使用该服务https://checkip.amazonaws.com。IP 地址的一个示例为 `203.0.113.25/32`。

   在许多情况下，您可能通过互联网服务提供商（ISP）进行连接，或者在不使用静态 IP 地址的情况下从防火墙之后进行连接。如果是这样，请找出客户端计算机使用的 IP 地址范围。

   > **Warning**
   >
   > 如果您使用 `0.0.0.0/0` 进行 SSH 访问，则所有 IP 地址可能能够使用 SSH 访问您的公有实例。在测试环境下短时间内，此方法尚可接受，但它对于生产环境并不安全。在生产环境中，将仅向特定 IP 地址或地址范围授权使用 SSH 访问您的实例。

   b. 在入站规则部分中，选择添加规则。

   c. 为新入站规则设置以下值，以允许 SSH 访问 Amazon EC2 实例。完成此操作后，您就可以连接到 EC2 实例来安装应用程序和其他实用程序。您还可以连接到 EC2 实例，以便上载应用程序的内容。

      - 类型：**SSH**
      - 来源：您在步骤 a 中创建的 IP 地址或范围，例如：**203.0.113.25/32**。

   d. 选择添加规则。

   e. 为新入站规则设置以下值以允许针对应用程序的 HTTP 访问：

      - 类型：**HTTP**
      - 来源：**0.0.0.0/0**

5. 请选择 Create security group（创建安全组）以创建安全组。

   请记下安全组 ID，因为稍后在另一个过程中需要使用该 ID。

步骤 3：为私有集群创建 VPC 安全组

要保持您的集群为私有，请创建第二个安全组进行私有访问。要连接到 VPC 中的私有集群，请将入站规则添加到 VPC 安全组，以仅支持来自应用程序的流量。

创建 VPC 安全组

1. 打开位于 https://console.aws.amazon.com/vpc 的 Amazon VPC 控制台。
2. 依次选择 VPC 控制面板)、安全组和创建安全组。
3. 在创建安全组页面上，设置以下值：
   - 安全组名称：**example-securitygroup**
   - 描述：**Instance security group**
   - VPC：选择您之前创建的 VPC，例如：vpc-example
4. 将入站规则添加到安全组。
   a. 在入站规则部分中，选择添加规则。
   b. 为新入站规则设置以下值，以允许 Amazon EC2 实例中端口 27017 上的 DocumentDB 流量。完成此操作后，您就可以从应用程序连接到集群。这样，您就可以从应用程序将数据存储和检索到数据库。
      - 类型：**Custom TCP**
      - 来源：您先前在本主题中创建的应用程序安全组的标识符，例如：sg-9edd5cfb。
   c. 选择添加规则。
   d. 为新入站规则设置以下值以允许针对应用程序的 HTTP 访问：
      - 类型：**HTTP**
      - 来源：**0.0.0.0/0**
5. 请选择 Create security group（创建安全组）以创建安全组。

步骤 4：创建子网组

子网组是在 VPC 中创建的一组子网，随后可为集群指定这些子网。通过子网组，您可能能够在创建集群时指定特定的 VPC。

创建子网组

1. 在 VPC 中识别数据库的私有子网。
   a. 打开位于 https://console.aws.amazon.com/vpc 的 Amazon VPC 控制台。
   b. 选择 VPC Dashboard（VPC 控制面板），然后选择 Subnets（子网）。

c. 记下您在步骤 1 中创建的名为如下的子网的子网 ID：example-subnet-private1-us-west-2a 和 example-subnet-private2-us-west-2b。创建子网组时需要子网 ID。

2. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb

   确保您连接到 Amazon DocumentDB 控制台，而不是 Amazon VPC 控制台。

3. 在导航窗格中，选择子网组。
4. 选择创建。
5. 在创建子网组页面的子网组详细信息部分中设置以下值：

   - 名称：**example-db-subnet-group**
   - 描述：**Instance security group**

6. 在添加子网部分中，选择以下值：

   - VPC：选择您之前创建的 VPC，例如：vpc-example
   - 可用区：选择在步骤 1 中创建的两个可用区。示例：us-west-2a 和 us-west-2b
   - 子网：选择您在步骤 1 中创建的私有子网。

7. 选择创建。

您的新子网组显示在 DocumentDB 控制台的子网组列表中。您可以选择子网组以在详细信息窗格中查看详细信息。这些详细信息包括与该组关联的所有子网。

> **Note**
>
> 如果您已创建此 VPC 以将其与 DocumentDB 集群关联，请按照创建 Amazon DocumentDB 集群中的说明创建集群。

删除 VPC

如果不再需要 VPC 及其中使用的其他资源，可以将其删除。

> **Note**
>
> 如果您在本主题中创建的 VPC 中添加了资源，则可能需要先删除这些资源，然后才能删除 VPC。例如，这些资源可能包括 Amazon EC2 实例或 DocumentDB 集群。有关更多信息，请参阅 Amazon VPC 用户指南中的您的 VPC 的安全性。

删除 VPC 和相关资源

1. 删除子网组：
   a. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
   b. 在导航窗格中，选择子网组。
   c. 选择要删除的子网组，例如 example-db-subnet-group。
   d. 选择 Delete (删除)，然后在确认窗口中选择 Delete (删除)。
2. 记下 VPC ID：
   a. 打开位于 https://console.aws.amazon.com/vpc 的 Amazon VPC 控制台。
   b. 选择 VPC 控制面板，然后选择您的 VPC。
   c. 在列表中，标识您创建的 VPC，例如 vpc-example。
   d. 记下所创建 VPC 的 VPC ID。在后面的步骤中，您需要此 VPC ID。
3. 删除安全组：
   a. 打开位于 https://console.aws.amazon.com/vpc 的 Amazon VPC 控制台。
   b. 选择 VPC 控制面板，然后选择安全组。
   c. 选择 Amazon DocumentDB 集群的安全组，例如 example-securitygroup。
   d. 对于操作，请选择删除安全组，然后在确认对话框中选择删除。
   e. 回到安全组页面上，选择 Amazon EC2 实例的安全组，例如 example-securitygroup。
   f. 对于操作，请选择删除安全组，然后在确认对话框中选择删除。
4. 删除 VPC：
   a. 打开位于 https://console.aws.amazon.com/vpc 的 Amazon VPC 控制台。
   b. 选择 VPC 控制面板，然后选择您的 VPC。
   c. 选择要删除的 VPC，例如 vpc-example。

d. 对于操作，请选择“删除 VPC”。

确认页面显示与 VPC 关联的其他资源，这些资源也将被删除，包括与其关联的子网。

e. 在确认对话框中，输入 **delete**，然后选择删除。

创建双堆栈 VPC 以用于 DocumentDB 集群

一种常见的场景包括虚拟私有云（VPC）中基于 Amazon VPC 服务的集群。此 VPC 与在同一 VPC 中运行的公有 Amazon EC2 实例共享数据。在本主题中，针对此场景创建 VPC。

主题

- 步骤 1：创建包含私有子网和公有子网的 VPC
- 步骤 2：为公有 Amazon EC2 实例创建 VPC 安全组
- 步骤 3：为私有集群创建 VPC 安全组
- 步骤 4：创建子网组
- 步骤 5：在双堆栈模式下创建 Amazon EC2 实例
- 步骤 6：在双堆栈模式下创建集群
- 步骤 7：连接 Amazon EC2 实例和数据库集群
- 删除 VPC

在此过程中，您将为此场景创建 VPC，该VPC适用于在双堆栈模式下运行的数据库。 Dual-stack 模式允许通过 IPv6 寻址协议进行连接。有关 IP 地址的更多信息，请参阅 Amazon DocumentDB IP 寻址。

Dual-stack 大多数地区都支持网络集群。有关更多信息，请参阅 Dual-stack 模式区域和版本可用性。要查看双堆栈模式的限制，请参阅 双堆栈网络集群的限制。

本主题和 IPv4-only 主题在同一 VPC 中创建公有子网和私有子网。有关在一个 VPC 中创建 Amazon DocumentDB 集群并在另一个 VPC 中创建 Amazon EC2 实例的信息，请参阅 访问 VPC 中的 Amazon DocumentDB 集群。

DocumentDB 集群只需对 Amazon EC2 实例可用，而无需对公共互联网可用。因此，请创建包含公有子网和私有子网的 VPC。EC2 实例托管在公有子网中，因此能够访问公共互联网。集群托管在私有子网中。EC2 实例可以连接到集群，因为它托管在同一 VPC 内。但是，集群不可用于公共互联网，从而提高了安全性。

本主题中的此过程在单独的可用区中配置额外的公有子网和私有子网。此过程未使用这些子网。DocumentDB 子网组需要位于至少两个可用区中的一个子网。借助额外的子网，可以轻松地配置多个 DocumentDB 实例。

要创建使用双堆栈模式的集群，请为网络类型设置指定 Dual-stack mod e。您也可以使用相同的设置修改集群。有关创建集群的更多信息，请参阅 创建 Amazon DocumentDB 集群。有关修改数据库集群的更多信息，请参阅 修改 Amazon DocumentDB 集群。

本主题介绍为 Amazon DocumentDB 集群配置 VPC。有关 Amazon VPC 的更多信息，请参阅 Amazon VPC 用户指南。

步骤 1：创建包含私有子网和公有子网的 VPC

使用以下步骤创建包含公有和私有子网的 VPC。

创建 VPC 和子网

1. 打开位于 https://console.aws.amazon.com/vpc 的 Amazon VPC 控制台。
2. 在的右上角 AWS 管理控制台，选择要在其中创建 VPC 的区域。此示例使用 美国西部（俄勒冈）区域。
3. 在左上角，选择 VPC Dashboard（VPC 控制面板）。要开始创建 VPC，请选择 Create VPC（创建 VPC）。
4. 对于 VPC Settings（VPC 设置）下的 Resources to create（要创建的资源），选择 VPC and more（VPC 及更多）。
5. 对于 VPC settings（VPC 设置），请设置以下值：

   - 名称标签自动生成：**example-dual-stack**
   - IPv4 CIDR 数据块：**10.0.0.0/16**
   - IPv6 网段—— Amazon-provided IPv6 网段封锁
   - 租赁：默认值
   - 可用区（AZ）的数量：2
   - 自定义可用区：保留默认值
   - 公有子网的数量：2
   - 私有子网的数量：2
   - 自定义子网 CIDR 数据块：保留默认值
   - NAT 网关（$）：无

- 仅出口互联网网关：否
- VPC 端点：无
- DNS 选项：保留默认值

6. 选择创建 VPC。

步骤 2：为公有 Amazon EC2 实例创建 VPC 安全组

接下来创建安全组以便公共访问。要连接到 VPC 中的公有 EC2 实例，请将入站规则添加到 VPC 安全组，以允许流量从互联网连接。

创建 VPC 安全组

1. 打开位于 https://console.aws.amazon.com/vpc 的 Amazon VPC 控制台。
2. 依次选择 VPC 控制面板)、安全组和创建安全组。
3. 在创建安全组页面上，设置以下值：
   - 安全组名称：**example-dual-stack-securitygroup**
   - 描述：**Dual-stack security group**
   - VPC：选择您之前创建的 VPC，例如：vpc-example-dual-stack。
4. 将入站规则添加到安全组。
   a. 确定要用来通过 Secure Shell（SSH）连接到 VPC 中的 EC2 实例的 IP 地址。要确定您的公有 IP 地址，可以在不同的浏览器窗口或选项卡中使用该服务https://checkip.amazonaws.com。

      互联网协议版本 4 (IPv4) 地址范围的一个示例是`203.0.113.25/32`。互联网协议版本 6（IPv6）地址范围的示例为 `2001:db8:1234:1a00::/64`。

      在许多情况下，您可能通过互联网服务提供商（ISP）进行连接，或者在不使用静态 IP 地址的情况下从防火墙之后进行连接。如果是这样，请找出客户端计算机使用的 IP 地址范围。

      > **Warning**
      >
      > 如果对 IPv4 使用 `0.0.0.0/0` 或对 IPv6 使用 `::0`，则使所有 IP 地址可能能够使用 SSH 访问您的公有实例。在测试环境下短时间内，此方法尚可接受，但它对于生产环境并不安全。在生产环境中，请仅授权特定 IP 地址或地址范围访问您的实例。

b. 在入站规则部分中，选择添加规则。

c. 为新入站规则设置以下值，以允许 SSH 访问 Amazon EC2 实例。完成此操作后，您就可以连接到 EC2 实例来安装应用程序或其他实用程序。指定一个 IP 地址，以便您可以访问 EC2 实例：

- 类型：**SSH**
- 来源：您在步骤 a 中创建的 IP 地址或范围。IPv4 地址范围的一个示例为 **203.0.113.25/3**2。IPv6 地址范围的一个示例为 **2001:DB8::/32**。

d. 选择添加规则。

5. 请选择 Create security group（创建安全组）以创建安全组。

请记下安全组 ID，因为稍后在另一个过程中需要使用该 ID。

步骤 3：为私有集群创建 VPC 安全组

要保持您的集群为私有，请创建第二个安全组进行私有访问。要连接到 VPC 中的私有集群，请将入站规则添加到 VPC 安全组。它们仅允许来自 Amazon EC2 实例的流量。

创建 VPC 安全组

1. 打开位于 https://console.aws.amazon.com/vpc 的 Amazon VPC 控制台。
2. 依次选择 VPC 控制面板)、安全组和创建安全组。
3. 在创建安全组页面上，设置以下值：

- 安全组名称：**example-dual-stack-cluster-securitygroup**
- 描述：**Dual-stack cluster security group**
- VPC：选择您之前创建的 VPC，例如：vpc-example-dual-stack

4. 将入站规则添加到安全组。

a. 在入站规则部分中，选择添加规则。

b. 为新入站规则设置以下值，以允许 Amazon EC2 实例中端口 27017 上的 DocumentDB 流量。完成此操作后，您就可以从 EC2 实例连接到集群。这样您就可以将数据从 EC2 实例发送到数据库。

- 类型：**Custom TCP**
- 来源：您先前在本主题中创建的 EC2 安全组的标识符，例如：sg-9edd5cfb。

5. 请选择 Create security group（创建安全组）以创建安全组。

步骤 4：创建子网组

子网组是在 VPC 中创建的一组子网，随后可为集群指定这些子网。通过使用子网组，您可以在创建集群时指定特定的 VPC。要创建与 DUAL 兼容的子网组，所有子网都必须与 DUAL 兼容。要与 DUAL 兼容，子网必须具有与之关联的 IPv6 CIDR。

创建子网组

1. 在 VPC 中识别数据库的私有子网。

   a. 打开位于 https://console.aws.amazon.com/vpc 的 Amazon VPC 控制台。

   b. 选择 VPC Dashboard（VPC 控制面板），然后选择 Subnets（子网）。

   c. 记下您在步骤 1 中创建的名为如下的子网的子网 ID：example-dual-stack-subnet-private1-us-west-2a 和 example-dual-stack-subnet-private2-us-west-2b。创建子网组时需要子网 ID。

2. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb

   确保您连接到 Amazon DocumentDB 控制台，而不是 Amazon VPC 控制台。

3. 在导航窗格中，选择子网组。

4. 选择创建。

5. 在创建子网组页面的子网组详细信息部分中设置以下值：

   - 名称：**example-dual-stack-cluster-subnet-group**
   - 描述：**Dual-stack cluster subnet group**

6. 在添加子网部分中，选择以下值：

   - VPC：选择您之前创建的 VPC，例如：vpc-example-dual-stack
   - 可用区：选择在步骤 1 中创建的两个可用区。示例：us-west-2a 和 us-west-2b
   - 子网：选择您在步骤 1 中创建的私有子网。

7. 选择创建。

您的新子网组显示在 DocumentDB 控制台的子网组列表中。您可以选择子网组以在详细信息窗格中查看详细信息。这些详细信息包括与该组关联的所有子网。

步骤 5：在双堆栈模式下创建 Amazon EC2 实例

要创建 Amazon EC2 实例，请按照《Amazon Elastic Compute Cloud 用户指南》中的使用启动实例向导启动 EC2 实例中的说明操作。

在 Configure Instance Details（配置实例详细信息）页面上，设置以下值并将其他值保留为其原定设置值：

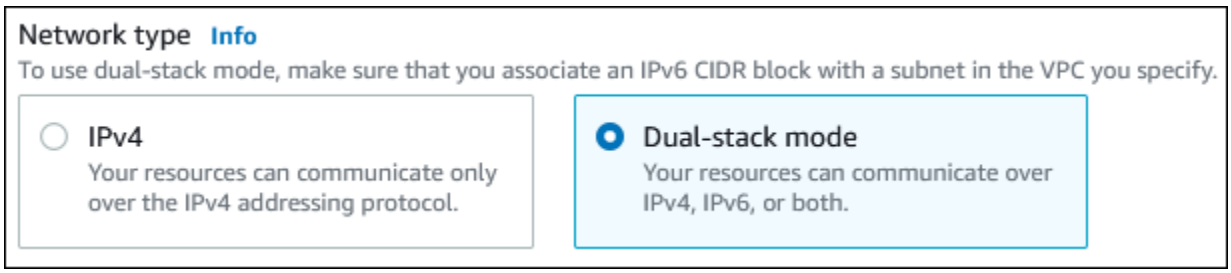
- 网络：选择同时具有公有子网和私有子网的现有 VPC，如在 步骤 1：创建包含私有子网和公有子网的 VPC 中创建的 vpc-example-dual-stack-vpc（vpc-identifier）。
- 子网：选择一个现有的公有子网，如在 步骤 2：为公有 Amazon EC2 实例创建 VPC 安全组 中创建的 subnet-identifier | example-dual-stack-subnet-public1-us-east-2a | us-east-2a。
- Auto-assign 公有 IP-选择启用。
- Auto-assign IPv6 IP-选择启用。
- 防火墙（安全组）：选择选择现有安全组。
- 常见安全组：选择现有安全组，例如在 步骤 2：为公有 Amazon EC2 实例创建 VPC 安全组 中创建的 example-dual-stack-securitygroup。确保您选择的安全组包括 Secure Shell (SSH) 和 HTTP 访问的入站规则。

步骤 6：在双堆栈模式下创建集群

在此步骤中，您将创建在双堆栈模式下运行的数据库集群。！！！注意：在对控制台进行 IPv6 更新后，需要编辑此部分！！！

在双堆栈模式下创建集群

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在控制台的右上角，选择要 AWS 区域 在哪里创建 DocumentDB 集群。此示例使用美国东部（俄亥俄州）区域。
3. 在导航窗格中，选择集群。
4. 在集群列表页面上，选择创建。
5. 在创建 Amazon DocumentDB 集群页面上，确保选择了Instance-based 集群选项。
6. 在 “连接” 部分的网络类型下，选择Dual-stack 模式。



7. 在页面底部，打开显示高级设置。
8. 在网络设置部分中，选择以下值：

   - 虚拟私有云（VPC）：选择同时具有公有子网和私有子网的现有 VPC，如在 步骤 1：创建包含私有子网和公有子网的 VPC 中创建的 vpc-example-dual-stack（vpc-identifier）。

     VPC 的子网必须位于不同的可用区中。
   - 子网组：选择 VPC 的子网组，如在 步骤 4：创建子网组 中创建的 example-dual-stack-cluster-subnet-group。
   - 公有访问权限：选择否。
   - VPC 安全组（防火墙）：选择选择现有。
   - 现有 VPC 安全组：选择为私有访问配置的现有 VPC 安全组，如在 步骤 3：为私有集群创建 VPC 安全组 中创建的 example-dual-stack-cluster-securitygroup。

     通过选择与其他每个安全组关联的 X 来删除该安全组，如默认安全组。
   - 可用区：选择您在步骤 1 中创建的可用区。示例：us-west-2a。

     为避免跨可用区的流量，请确保集群和 EC2 实例位于同一个可用区内。
9. 对于其余部分，请指定集群设置。有关每项设置的信息，请参阅创建 Amazon DocumentDB 集群。

步骤 7：连接 Amazon EC2 实例和数据库集群

在双堆栈模式下创建 Amazon EC2 实例和 DocumentDB 集群后，您可以使用 IPv6 协议连接到每个实例。要使用 IPv6 协议连接到 EC2 实例，请按照《Amazon Elastic Compute Cloud 用户指南》中的连接 EC2 实例中的说明操作。

要从 EC2 实例连接到您的 DocumentDB 集群，请按照“手动连接 Amazon EC2”主题中的 步骤 5：安装 MongoDB Shell 中的说明操作（并继续执行此过程中的后续步骤 6 和步骤 7）。

删除 VPC

如果不再需要 VPC 及其中使用的其他资源，可以将其删除。

> **Note**
>
> 如果您在本主题中创建的 VPC 中添加了资源，则可能需要先删除这些资源，然后才能删除 VPC。例如，这些资源可能包括 Amazon EC2 实例或 DocumentDB 集群。有关更多信息，请参阅 Amazon VPC 用户指南中的您的 VPC 的安全性。

删除 VPC 和相关资源

1. 删除子网组：
   a. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
   b. 在导航窗格中，选择子网组。
   c. 选择要删除的子网组，例如 example-dual-stack-cluster-subnet-group。
   d. 选择 Delete (删除)，然后在确认窗口中选择 Delete (删除)。
2. 记下 VPC ID：
   a. 打开位于 https://console.aws.amazon.com/vpc 的 Amazon VPC 控制台。
   b. 选择 VPC 控制面板，然后选择您的 VPC。
   c. 在列表中，标识您创建的 VPC，例如 vpc-example-dual-stack。
   d. 记下所创建 VPC 的 VPC ID。在后面的步骤中，您需要此 VPC ID。
3. 删除安全组：
   a. 打开位于 https://console.aws.amazon.com/vpc 的 Amazon VPC 控制台。
   b. 选择 VPC 控制面板，然后选择安全组。
   c. 选择 Amazon DocumentDB 集群的安全组，例如 example-dual-stack-securitygroup。
   d. 对于操作，请选择删除安全组，然后在确认对话框中选择删除。
   e. 回到安全组页面上，选择 Amazon EC2 实例的安全组，例如 example-securitygroup。
   f. 对于操作，请选择删除安全组，然后在确认对话框中选择删除。
4. 删除 NAT 网关：

a. 打开位于 https://console.aws.amazon.com/vpc 的 Amazon VPC 控制台。
b. 选择 VPC 控制面板，然后选择安全组。
c. 选择您创建的 VPC 的 NAT 网关。使用 VPC ID 标识正确的 NAT 网关。
d. 对于 Actions（操作），请选择 Delete NAT gateway（删除 NAT 网关）。
e. 在确认对话框中，输入 **delete**，然后选择删除。

5. 删除 VPC：

a. 打开位于 https://console.aws.amazon.com/vpc 的 Amazon VPC 控制台。
b. 选择 VPC 控制面板，然后选择您的 VPC。
c. 选择要删除的 VPC，例如 vpc-example-dual-stack。
d. 对于操作，请选择“删除 VPC”。

   确认页面显示与 VPC 关联的其他资源，这些资源也将被删除，包括与其关联的子网。

e. 在确认对话框中，输入 **delete**，然后选择删除。

6. 释放弹性 IP 地址：

a. 通过 https://console.aws.amazon.com/ec2 打开 EC2 控制台。
b. 选择 EC2 控制面板，然后选择弹性 IP。
c. 选择要释放的弹性 IP 地址。
d. 对于 Actions（操作），请选择 Release Elastic IP addresses（释放弹性 IP 地址）。
e. 在确认对话框中，选择释放。

在 Amazon DocumentDB 中进行备份和还原

Amazon DocumentDB（与 MongoDB 兼容）可将数据持续备份到 Amazon Simple Storage Service (Amazon S3) 中 1-35 天，这样您就可以快速恢复到备份留存期内的任何时间点。作为持续备份过程的一部分，Amazon DocumentDB 还会自动拍摄数据快照。

> **Note**
>
> 这些是服务托管的 Amazon S3 存储桶，您无法访问备份文件。如果要控制自己的备份，请按照转储、恢复、导入和导出数据中的说明进行操作。

还可以通过为集群的数据创建手动快照来超出备份保留期保留备份数据。备份过程不会影响您的集群的性能。

本节讨论在 Amazon DocumentDB 中的备份功能的使用案例，并为您展示如何管理 Amazon DocumentDB 集群的备份。

主题

- 备份和还原：概念
- 了解 备份存储使用量
- 转储、还原、导入和导出数据
- 集群快照注意事项
- 比较自动快照和手动快照
- 创建手动集群快照
- 复制 Amazon DocumentDB 集群快照
- Amazon DocumentDB 集群快照共享
- 从集群快照还原
- 还原到某个时间点
- 删除集群快照

备份和还原：概念

名词	说明	API（动词）

备份保留期	介于 1 到 35 天之间的时间段，在此时间段内，您可以执行时间点还原。	`create-db-cluster`
		`modify-db-cluster`
		`restore-db-cluster-to-point-in-time`

Amazon DocumentDB 存储卷	高可用性和高持久性的存储卷，可以跨三个可用区以六种方式复制数据。无论 Amazon DocumentDB 集群中的实例数量是多少，该集群都具有高持久性。	`create-db-cluster` `delete-db-cluster`

备份时段	拍摄自动快照的一天中的时间段。	`create-db-cluster`
		`describe-db-cluster`
		`modify-db-cluster`

自动快照	每日快照是集群的完整备份，由 Amazon DocumentD	`restore-db-cluster-from-snapshot` `describe-db-cluster-snapshot-attributes`

名词	说明	API（动词）
	B 中的连续备份过程自动创建。	`describe-db-cluster-snapshots`

手动快照	您手动创建的快照，用于超出备份期间保留集群的完整备份。	`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`

了解 备份存储使用量

Amazon DocumentDB 备份存储包含备份留存期内的连续备份和留存期外的手动快照。要控制备份存储使用量，可以缩短备份保留间隔和/或删除不再需要的旧的手动快照。有关 Amazon DocumentDB 备份的一般信息，请参阅在 Amazon DocumentDB 中进行备份和还原。有关 Amazon DocumentDB 备份存储的定价信息，请参阅 Amazon DocumentDB 定价。

要控制成本，您可以监控持续备份和存在时间超出保留期的手动快照所消耗的存储量。然后，您可以缩短备份保留间隔，并在不再需要手动快照时删除它们。

您可以使用亚马逊 CloudWatch 指标`TotalBackupStorageBilled`、`SnapshotStorageUsed`和`BackupRetentionPeriodStorageUsed`来查看和监控您的 Amazon DocumentDB 备份使用的存储量，如下所示：

- `BackupRetentionPeriodStorageUsed` 表示目前用于存储连续备份的备份存储量。此度量值取决于集群卷的大小和您在保留期内所做的更改的数量。但是，出于计费目的，此指标不会超过保留期内的集群卷的累计大小。例如，如果您的集群大小为 100 GiB 且保留期为两天，则 `BackRetentionPeriodStorageUsed` 的最大值为 200 GiB (100 GiB + 100 GiB)。

- SnapshotStorageUsed 表示用于存储超出备份保留期的手动快照的备份存储量。在保留期内拍摄的手动快照不计入备份存储。同样，自动快照不会计入备份存储。每个快照的大小是您拍摄快照时的集群卷的大小。SnapshotStorageUsed 值取决于您保留的快照数和每个快照的大小。例如，假设您有一个保留期外的快照，并且在拍摄快照时集群卷大小为 100 GiB。SnapshotStorageUsed 的数量将为 100 GiB。

- TotalBackupStorageBilled 表示 BackupRetentionPeriodStorageUsed 与 SnapshotStorageUsed 之和再减去免费备份存储量（等于一天的集群卷的大小）。例如，如果集群大小为 100 GiB，您有一天的留存期，并且您有一个留存期外的快照，则 TotalBackupStorageBilled 为 100 GiB (100 GiB + 100 GiB - 100 GiB)。

- 这些指标是为每个 Amazon DocumentDB 集群单独计算的。

您可以通过控制台监控您的 Amazon DocumentDB 集群并使用 CloudWatch 指标生成报告。CloudWatch 有关如何使用 CloudWatch 指标的更多信息，请参阅监控 Amazon DocumentDB。

转储、还原、导入和导出数据

您可以使用 mongodump、mongorestore、mongoexport 和 mongoimport 实用工具将数据移入和移出 Amazon DocumentDB 集群。此部分讨论了所有这些工具和配置的用途，以帮助您实现更好的性能。

主题

- mongodump
- mongorestor
- 蒙戈出口
- mongoimpor
- 教程

mongodump

mongodump 实用工具创建 MongoDB 数据库的二进制 (BSON) 备份。由于以二进制格式存储数据可实现较高的大小效率，因此，在希望将数据还原到 Amazon DocumentDB 集群中时，mongodump 工具是转储源 MongoDB 部署中的数据的首选方法。

根据用于执行命令的实例或计算机上的可用资源，您可以使用 `--numParallelCollections` 选项增加从默认 1 转储的并行集合的数量，从而加快 mongodump 的速度。一个好的经验法则是，在 Amazon DocumentDB 集群的主实例上，为每个 vCPU 启动一个工作线程。

> **Note**
>
> 我们对 Amazon DocumentDB 推荐高达且包括版本 100.6.1 的 MongoDB 数据库工具。您可以在此处下载 MongoDB 数据库工具。

示例用法

下面是 Amazon DocumentDB 集群 `sample-cluster` 中的 mongodump 实用工具的使用示例。

```
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
```

mongorestor

利用 `mongorestore` 实用工具，您可以还原使用 mongodump 实用工具创建的数据库的二进制 (BSON) 备份。您可以通过使用 `--numInsertionWorkersPerCollection` 选项（默认值为 1）增加还原期间每个集合的工作线程数来提高还原性能。一个好的经验法则是，在 Amazon DocumentDB 集群的主实例上，为每个 vCPU 启动一个工作线程。

示例用法

下面是 Amazon DocumentDB 集群 `sample-cluster` 中的 mongorestore 实用工具的使用示例。

```
mongorestore --ssl \
    --host="sample-cluster.node.us-east-1.docdb.amazonaws.com:27017" \
    --username=sample-user \
    --password=abc0123 \
```

```
    --sslCAFile global-bundle.pem <fileToBeRestored>
```

####

mongoexport 工具将 Amazon DocumentDB 中的数据导出为 JSON、CSV 或 TSV 文件格式。mongoexport 工具是导出要求可供人类或机器读取的数据的首选方法。

> **Note**
>
> mongoexport 无法直接支持并行导出。但是，可以通过同时为不同的集合执行多个 mongoexport 作业来提高性能。

示例用法

下面是 Amazon DocumentDB 集群 sample-cluster 中的 mongoexport 工具的使用示例。

```
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
```

mongoimpor

mongoimport 工具将 JSON、CSV 或 TSV 文件的内容导入 Amazon DocumentDB 集群中。可以使用 --numInsertionWorkers 参数实现并行化和加快导入速度（默认值为 1）。

示例用法

下面是 Amazon DocumentDB 集群 sample-cluster 中的 mongoimport 工具的使用示例。

```
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
```

教程

以下教程介绍如何使用 mongodump、mongorestore、mongoexport 和 mongoimport 实用工具将数据移入和移出 Amazon DocumentDB 集群。

1. 先决条件：在开始之前，请确保您的 Amazon DocumentDB 集群已配置，并且您有权访问位于您的集群所在的同一 VPC 中的 Amazon EC2 实例。有关更多信息，请参阅 使用 Amazon EC2 进行连接。

   要能够使用 mongo 实用工具，您必须已在 EC2 实例中安装 mongodb-org-tools 包，如下所示。

   ```
   sudo yum install mongodb-org-tools-4.0.18
   ```

   由于 Amazon DocumentDB 默认使用传输层安全性协议（TLS）加密，因此您还必须下载 Amazon RDS 证书颁发机构 (CA) 文件以使用 mongo Shell 进行连接，如下所示。

   ```
   wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
   ```

2. 下载示例数据：在本教程中，您将下载一些包含餐厅相关信息的示例数据。

   ```
   wget https://raw.githubusercontent.com/ozlerhakan/mongodb-json-files/master/
   datasets/restaurant.json
   ```

3. 将示例数据导入 Amazon DocumentDB 中：由于数据采用的是逻辑 JSON 格式，因此，您将使用 mongoimport 实用工具将数据导入 Amazon DocumentDB 集群中。

   ```
   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. 使用 **mongodump** 转储数据：现在，Amazon DocumentDB 集群中已有数据，您可以使用 mongodump 实用工具对该数据进行二进制转储。

```
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. 删除 **restaurants** 集合：在还原 business 数据库中的 restaurants 集合之前，必须先删除该数据库中已有的集合，如下所示。

```
use business
```

```
db.restaurants.drop()
```

6. 使用 **mongorestore** 还原数据：在步骤 3 中对数据进行二进制转储后，您现在可以使用 mongorestore 实用工具将数据还原到 Amazon DocumentDB 集群。

```
mongorestore --ssl \
    --host="tutorialCluster.us-east-1.docdb.amazonaws.com:27017" \
    --numParallelCollections 4 \
    --username=<yourUsername> \
    --password=<yourPassword> \
    --sslCAFile global-bundle.pem restaurantDump.bson
```

7. 使用 **mongoexport** 导出数据：要完成本教程，请以 JSON 文件的格式从集群中导出数据，与您在步骤 1 中导入的文件没有什么不同。

```
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. 验证：您可以使用以下命令验证步骤 5 的输出是否会产生与步骤 1 相同的结果。

```
wc -l restaurant.json
```

此命令的输出：

```
2548 restaurant.json
```

```
wc -l restaurant2.json
```

此命令的输出：

```
2548 restaurant2.json
```

集群快照注意事项

Amazon DocumentDB 会在集群的备份窗口内创建集群的每日自动快照。Amazon DocumentDB 根据您指定的备份留存期保存集群的自动快照。如果需要，您可以将集群恢复到备份保留期中的任意时间点。在相同区域中对相同集群执行复制操作时，不会拍摄自动快照。

主题

- 备份存储
- 备份时段
- 备份保留期
- 复制集群快照加密

除了自动集群快照之外，您还可以手动创建集群快照。您可以同时复制自动快照和手动快照。有关更多信息，请参阅创建手动集群快照和复制 Amazon DocumentDB 集群快照。

> **Note**
>
> 您的集群必须处于可用 状态，才能拍摄自动快照。
> 您无法共享 Amazon DocumentDB 自动集群快照。解决方法是通过复制自动快照来创建手动快照，然后共享该副本。有关复制快照的更多信息，请参阅复制 Amazon DocumentDB 集群快照。有关从快照还原集群的更多信息，请参阅从集群快照还原。

备份存储

您的 Amazon Documen AWS 区域 tDB 备份存储空间由备份保留期所需的备份存储空间组成，其中包括该地区的自动和手动集群快照。默认备份保留期为 1 天。有关备份存储定价的更多信息，请参阅 Amazon DocumentDB 定价。

在删除集群时，所有自动快照都将被删除且无法恢复。但是，在删除集群时不会删除手动快照。如果您选择让 Amazon DocumentDB 在删除集群之前创建最终快照（手动快照），则可以使用该最终快照来恢复集群。

有关快照和存储的更多信息，请参阅了解 备份存储使用量。

备份时段

自动快照在每天的首选备份时段中进行。如果快照所需的时间超过了分配到备份时段的时间，那么即使备份时段已结束，备份过程也会继续直至完成。备份时段不能与集群的每周维护时段重叠。

如果创建集群时未指定首选备份时段，Amazon DocumentDB 将分配 30 分钟的默认备份时段。该时段是随机从与集群区域关联的 8 小时时间数据块中选择出来的。通过修改集群来更改您的首选备份时段。有关更多信息，请参阅 修改 Amazon DocumentDB 集群。

区域名称	Region	UTC 时间数据块
美国东部（俄亥俄州）	us-east-2	03:00-11:00
美国东部（弗吉尼亚州北部）	us-east-1	03:00-11:00
美国西部（俄勒冈州）	us-west-2	06:00-14:00
非洲（开普敦）	af-south-1	03:00–11:00
亚太地区（香港）	ap-east-1	06:00-14:00
亚太地区（海得拉巴）	ap-south-2	06:30–14:30
亚太地区（马来西亚）	ap-southeast-5	13:00-21:00
亚太地区（孟买）	ap-south-1	06:00-14:00
亚太地区（大阪）	ap-northeast-3	12:00-20:00

区域名称	Region	UTC 时间数据块
亚太地区（首尔）	ap-northeast-2	13:00-21:00
亚太地区（新加坡）	ap-southeast-1	14:00-22:00
亚太地区（悉尼）	ap-southeast-2	12:00-20:00
亚太地区（雅加达）	ap-southeast-3	08:00-16:00
亚太地区（墨尔本）	ap-southeast-4	11:00-19:00
亚太地区（泰国）	ap-southeast-7	15:00-23:00
亚太地区（东京）	ap-northeast-1	13:00-21:00
加拿大（中部）	ca-central-1	03:00-11:00
加拿大西部（卡尔加里）	ca-west-1	18:00-02:00
中国（北京）	cn-north-1	06:00-14:00
中国（宁夏）	cn-northwest-1	06:00-14:00
欧洲地区（法兰克福）	eu-central-1	21:00-05:00
欧洲（苏黎世）	eu-central-2	02:00-10:00
欧洲地区（爱尔兰）	eu-west-1	22:00-06:00
欧洲（伦敦）	eu-west-2	22:00-06:00
欧洲地区（米兰）	eu-south-1	02:00-10:00
欧洲地区（巴黎）	eu-west-3	23:59-07:29
欧洲（西班牙）	eu-south-2	02:00–10:00
欧洲地区（斯德哥尔摩）	eu-north-1	04:00 — 12:00
墨西哥（中部）	mx-central-1	03:00-11:00

区域名称	Region	UTC 时间数据块
中东（阿联酋）：	me-central-1	05:00–13:00
南美洲（圣保罗）	sa-east-1	00:00-08:00
以色列（特拉维夫）	il-central-1	04:00-12:00
AWS GovCloud (US-East)	us-gov-east-1	17:00-01:00
AWS GovCloud (US-West)	us-gov-west-1	06:00-14:00

备份保留期

备份留存期是自动备份在被自动删除之前保留的天数。Amazon DocumentDB 支持 1-35 天的备份留存期。

您可以在创建集群时设置备份保留期。如果您未明确设置备份保留期，则会为集群分配 1 天的默认备份保留期。创建集群后，您可以通过使用 AWS 管理控制台 或修改集群来修改备份保留期 AWS CLI。有关更多信息，请参阅 修改 Amazon DocumentDB 集群。

复制集群快照加密

集群和快照加密基于 KMS 加密密钥。 KMS 密钥 ID 是 Amazon 资源名称 (ARN)、 KMS 密钥标识符或 KMS 加密密钥的 KMS 密钥别名。

以下准则和限制适用：

- 创建快照时会从集群推断出加密。如果集群是加密的，则该集群的快照也使用相同的 KMS 密钥加密。如果集群未加密，则快照也不会加密。
- 如果您从 Amazon Web Services 账户复制加密的集群快照，则可以为 `KmsKeyId` 指定值来使用新的 KMS 加密密钥加密副本。如果您不为 `KmsKeyId` 指定值，则使用与源集群快照相同的 KMS 密钥来加密集群快照的副本。
- 如果您复制从其他 Amazon Web Services 账户共享的加密集群快照，则必须为 `KmsKeyId` 指定值。
- 要将加密的集群快照复制到另一个 Amazon Web Services 区域，请设置 `KmsKeyId` 为要用于加密目标区域中集群快照副本的 KMS 密钥 ID。KMS 加密密钥是特定于其创建时所在的 Amazon Web

Services 区域的，您无法将一个 Amazon Web Services 区域中的加密密钥用于另一个 Amazon Web Services 区域。

- 如果您复制未加密的集群快照并为 `KmsKeyId` 参数指定值，则会返回错误。

比较自动快照和手动快照

以下是 Amazon DocumentDB（与 MongoDB 兼容）自动快照和手动快照的主要功能。

Amazon DocumentDB 自动快照具有以下主要特征：

- 自动快照命名：自动快照名称遵循模式 `rds:<cluster-name>-yyyy-mm-dd-hh-mm`，其中 `yyyy-mm-dd-hh-mm` 表示创建快照的日期和时间。
- 按计划自动创建：当您创建或修改集群时，可将备份留存期设置为从 1 到 35 天的整数值。默认情况下，新集群的备份保留期为 1 天。备份保留期定义自动快照在被自动删除之前保留的天数。您不能对 Amazon DocumentDB 集群禁用自动备份。

  除了设置备份保留期，您还可以设置备份时段，即一天中创建自动快照的时间。
- 删除自动快照：当您删除自动快照的集群时，将会删除自动快照。您不能手动删除自动快照。
- 增量：在备份留存期内会记录数据库更新，因此存在增量更改记录。
- 从自动快照还原：您可以使用 AWS 管理控制台 或 AWS CLI从自动快照还原。使用从快照还原时 AWS CLI，必须在集群可用后单独添加实例。
- 共享：您无法共享 Amazon DocumentDB 自动集群快照。解决方法是通过复制自动快照来创建手动快照，然后共享该副本。有关复制快照的更多信息，请参阅复制 Amazon DocumentDB 集群快照。有关从快照还原集群的更多信息，请参阅从集群快照还原。
- 可从备份留存期内的任何点还原：由于数据库更新采用增量式记录，所以可将集群还原至备份留存期内的任何时间点。

  当您使用从自动快照或从时间点还原中恢复时 AWS CLI，必须在集群可用后单独添加实例。

Amazon DocumentDB 手动快照具有以下主要特征：

- 按需创建 — Amazon DocumentDB 手动快照是使用亚马逊 DocumentDB 管理控制台按需创建的，或者。 AWS CLI
- 删除手动快照：仅当您使用 Amazon DocumentDB 控制台或 AWS CLI显式删除手动快照时，才会将其删除。在删除手动快照的集群时，不会删除手动快照。
- 完整备份：拍摄手动快照时，会创建并存储您的集群的完整备份。

- 手动快照命名：您可以指定手动快照名称。Amazon DocumentDB 不会在名称中添加 `datetime` 印章，因此，如果您想在名称中包含该信息，则必须添加该信息。
- 从手动快照还原：您可以使用控制台或 AWS CLI从手动快照还原。使用从快照还原时 AWS CLI，必须在集群可用后单独添加实例。
- S@@ ervic e Quotas — 每个用户最多只能拥有 100 个手动快照 AWS 区域。
- 共享：您可以共享手动集群快照，这些快照可以由授权的 AWS 账户复制。您可以共享加密或未加密的手动快照。有关复制快照的更多信息，请参阅复制 Amazon DocumentDB 集群快照。
- 还原到制作手动快照的时间：在从手动快照还原时，将还原到制作手动快照的时间。

  使用从快照还原时 AWS CLI，必须在集群可用后单独添加实例。

创建手动集群快照

您可以使用 AWS 管理控制台 或创建手动快照 AWS CLI。创建快照所用时间因数据库大小而异。在创建快照时，您必须执行以下操作：

1. 确定要备份的集群。
2. 为快照指定名称。这样，您以后便可从中还原。

Using the AWS 管理控制台

要使用创建手动快照 AWS 管理控制台，您可以按照以下任一方法进行操作。

1. 方法 1：
   1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
   2. 在导航窗格中，选择快照。

      > **Tip**
      >
      > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

   3. 在 Snapshots (快照) 页面上，选择 Create (创建)。
   4. 在 Create cluster snapshot (创建集群快照) 页面上：

a. 集群标识符：从集群的下拉列表中，选择要为其创建快照的集群。

b. 快照标识符：输入快照的名称。

快照命名约束：

- 长度为 [1-255] 个字母、数字或连字符。
- 第一个字符必须是字母。
- 不能以连字符结尾或包含两个连续的连字符。
- 对于每个区域的每个 AWS 账户的所有集群（跨 Amazon RDS、Amazon Neptune 和 Amazon DocumentDB）必须是唯一的。

c. 选择创建。

2. 方法 2：

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在 Clusters (集群) 页面上，选择要拍摄快照的集群左侧的按钮。
4. 从 Actions (操作) 菜单中，选择 Take snapshot (拍摄快照)。
5. 在 Create cluster snapshot (创建集群快照) 页面上：

a. 快照标识符：输入快照的名称。

快照命名约束：

- 长度为 [1-63] 个字母、数字或连字符。
- 第一个字符必须是字母。
- 不能以连字符结尾或包含两个连续的连字符。
- 对于每个区域的每个 AWS 账户的所有集群（跨 Amazon RDS、Amazon Neptune 和 Amazon DocumentDB）必须是唯一的。

b. 选择创建。

Using the AWS CLI

要使用创建集群快照 AWS CLI，请使用带有以下参数的`create-db-cluster-snapshot`操作。

参数

- **`--db-cluster-identifier`**：必需。要拍摄快照的集群的名称。该集群必须存在且可用。
- **`--db-cluster-snapshot-identifier`**：必需。正在创建的手动快照的名称。

以下示例为名为 `sample-cluster` 的集群创建名为 `sample-cluster-snapshot` 的快照。

对于 Linux、macOS 或 Unix：

```
aws docdb create-db-cluster-snapshot \
    --db-cluster-identifier sample-cluster \
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

对于 Windows：

```
aws docdb create-db-cluster-snapshot ^
    --db-cluster-identifier sample-cluster ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

此操作的输出将类似于下文。

```
{
    "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"
    }
}
```

复制 Amazon DocumentDB 集群快照

在 Amazon DocumentDB 中，您可以将快照复制到同一个文件 AWS 区域 或另一个文件中。 AWS 区域您也可以将共享快照以相同 AWS 区域 或不同的方式复制到您的帐户 AWS 区域。有关共享快照的更多信息，请参阅Amazon DocumentDB 集群快照共享。

> **Note**
>
> Amazon DocumentDB 根据您保留的备份和快照数据量以及您保留的时间对您进行收费。有关与 Amazon DocumentDB 备份和快照关联的存储的更多信息，请参阅了解 备份存储使用量。有关 Amazon DocumentDB 存储的定价信息，请参阅 Amazon DocumentDB 定价。

主题

- 复制共享快照
- 跨复制快照 AWS 区域
- 限制
- 处理加密
- 参数组注意事项
- 复制集群快照

复制共享快照

您可以复制其他 AWS 账户与您共享的快照。如果您要复制已从其他 AWS 账户共享的加密快照，则必须有权访问用于 AWS KMS 加密快照的加密密钥。有关更多信息，请参阅 处理加密。

跨复制快照 AWS 区域

当您将快照复制到与源快照不同的快照时 AWS 区域，每个副本都是完整快照。 AWS 区域 完整快照副本包含恢复 Amazon DocumentDB 集群需要的所有数据和元数据。

限制

复制快照时，存在以下一些限制：

- 如果您在目标快照可用之前删除了源快照，则快照复制将失败。在删除源快照之前，请确保目标快照的状态为 AVAILABLE。
- 每个账户最多可以同时进行到同一目标区域的五个快照复制请求。
- 根据所 AWS 区域 涉及的内容和要复制的数据量，跨区域快照复制可能需要数小时才能完成。有时，某一给定的源 AWS 区域可能会发出大量跨区域快照复制请求。在这些情况下，Amazon DocumentDB 可能会将来自该来源的新跨区域复制请求放 AWS 区域 入队列，直到一些正在进行的副本完成。当复制请求在队列中时，不显示有关这些复制请求的进度信息。复制开始后即显示进度信息。

处理加密

您可以复制已使用 AWS KMS 加密密钥加密的快照。如果您复制加密的快照，则此快照的副本也必须加密。如果您在其中复制加密快照 AWS 区域，则可以使用与原始快照相同的 AWS KMS 加密密钥对副本进行加密，也可以指定不同的 AWS KMS 加密密钥。如果您跨区域复制加密快照，则不能对副本使用与源快照相同的 AWS KMS 加密密钥，因为 AWS KMS 密钥是 Region-specific。相反，您必须指定在目标 AWS 区域 n 中有效的 AWS KMS 密钥。

源快照在复制过程中保持加密状态。有关更多信息，请参阅 Amazon DocumentDB 中的数据保护。

> **Note**
>
> 对于 Amazon DocumentDB 集群快照，在复制快照时，您无法对未加密的集群快照进行加密。

参数组注意事项

跨区域复制快照时，复制不包括由原始 Amazon DocumentDB 集群使用的参数组。当您还原快照以创建新集群时，该集群将获得创建 AWS 区域 它的默认参数组。要为新的集群提供与源相同的参数组，您必须执行以下操作：

1. 在目标中 AWS 区域，使用与原始集群相同的设置创建一个 Amazon DocumentDB 集群参数组。如果新版本中已经存在一个 AWS 区域，则可以使用那个。
2. 在目标中恢复快照后 AWS 区域，修改新的 Amazon DocumentDB 集群并添加上一步中的新参数组或现有参数组。有关更多信息，请参阅 修改 Amazon DocumentDB 集群。

复制集群快照

您可以使用 AWS 管理控制台 或复制 Amazon DocumentDB 集群 AWS CLI，如下所示。

Using the AWS 管理控制台

要使用创建群集快照的副本 AWS 管理控制台，请完成以下步骤。此过程适用于在相同 AWS 区域区域或跨区域复制加密或未加密的集群快照。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择快照，然后选择要复制的快照左侧的按钮。

   > **Tip**
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在 Actions 菜单中，选择 Copy。
4. 在出现的 Make Copy of cluster snapshot（复制集群快照）页面中，完成 Settings（设置）部分。

   a. 目标区域：可选。要将集群快照复制到其他集群快照 AWS 区域，请 AWS 区域 为目标区域选择该快照。

   b. 新快照标识符：输入新快照的名称。

      目标快照命名约束：

      - 不能是现有快照的名称。
      - 长度为 [1-63] 个字母、数字或连字符。
      - 第一个字符必须是字母。
      - 不能以连字符结尾或包含两个连续的连字符。

- 每个区域的 Amazon RDS、Neptune 和 Amazon DocumentDB 中的所有集群都必须是唯一 AWS 账户的。

c. 复制标签：要将您在源快照上拥有的任何标签复制到快照副本，请选择复制标签。

5. 完成本Encryption-at-rest节。

a. 静态加密：如果您的快照未加密，则这些选项对您不适用，因为您无法从未加密的快照创建加密的副本。如果您的快照已加密，则可以更改静态加密期间 AWS KMS key 使用的快照。

有关对快照副本进行加密的更多信息，请参阅复制集群快照加密。

有关静态加密的更多信息，请参阅 Amazon DocumentDB 静态数据加密。

b. AWS KMS 密钥-从下拉列表中选择以下选项之一：

- （默认）aws/rds-账号和 AWS KMS 密钥 ID 列在此选项后面。
- <some-key-name>：如果您创建了密钥，它将列出并可供您选择。
- 输入一个密钥 ARN：在 ARN 框中，输入 AWS KMS 密钥的 Amazon 资源名称（ARN）。ARN 的格式为 `arn:aws:kms:<region>:<accountID>:key/<key-id>` 。

6. 要制作所选快照的副本，请选择 Copy snapshot (复制快照)。或者，您可以选择取消以便不创建快照的副本。

Using the AWS CLI

要使用 AWS CLI创建未加密集群快照的副本，请使用带以下参数的 `copy-db-cluster-snapshot` 操作。如果要将快照复制到另一个快照 AWS 区域，请运行将快照复制 AWS 区域 到的命令。

- **`--source-db-cluster-snapshot-identifier`**：必需。要制作副本的集群快照的标识符。集群快照必须存在并且处于可用 状态。如果您要将快照复制到另一个快照 AWS 区域 或者要复制共享集群快照，则此标识符必须采用源集群快照的 ARN 格式。此参数不区分大小写。
- **`--target-db-cluster-snapshot-identifier`**：必需。要从源集群快照创建的新集群快照标识符。此参数不区分大小写。

目标快照命名约束：

- 不能是现有快照的名称。
- 长度为 [1-63] 个字母、数字或连字符。

  - 第一个字符必须是字母。
  - 不能以连字符结尾或包含两个连续的连字符。
  - 每个区域的 Amazon RDS、Neptune 和 Amazon DocumentDB 中的所有集群都必须是唯一 AWS 账户的。
- **`--source-region`**— 如果您要将快照复制到另一个快照 AWS 区域，请指定要从中复制加密的集群快照。 AWS 区域

  如果您复制快照到另一 AWS 区域 ，且不指定 `--source-region`，则必须指定 `pre-signed-url` 选项代替。该`pre-signed-url`值必须是包含签名版本 4 签名请求的 URL，以便在复制集群快照的源 AWS 区域 中调用`CopyDBClusterSnapshot`操作。要了解更多信息`pre-signed-url`，请参阅 CopyDBClusterSnapshot。
- **`--kms-key-id`**：用于对集群快照副本进行加密的密钥的 KMS 密钥标识符。

  如果您要将加密的集群快照复制到另一个集群快照 AWS 区域，则此参数为必填项。您必须为目标指定 KMS 密钥 AWS 区域。

  如果您要在同一个中复制加密的集群快照 AWS 区域，则 AWS KMS 密钥参数是可选的。集群快照的副本使用与源集群快照相同的 AWS KMS 密钥进行加密。如果要指定用于 AWS KMS 加密副本的新加密密钥，则可以使用此参数进行加密。
- **`--copy-tags`**— 可选。要复制过来的标签和值。

要在正在进行复制时取消操作，您可以在集群快照处于正在复制状态时删除由 `--target-db-cluster-snapshot-identifier` 或 `TargetDBClusterSnapshotIdentifier` 标识的目标集群快照。

Example

示例 1：将未加密的快照复制到同一区域

以下 AWS CLI 示例创建了与源快照`sample-cluster-snapshot` AWS 区域 相同`sample-cluster-snapshot-copy`的 named 的副本。创建副本时，原始快照上的所有标签都将复制到快照副本。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此操作的输出将类似于下文。

```
{
    "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

示例 2：跨复制未加密的快照 AWS 区域

以下 AWS CLI 示例创建了具有 AR `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:sample-cluster-snapshot` N 的`sample-cluster-snapshot`副本。此副本命名为，`sample-cluster-snapshot-copy`并且 AWS 区域 在其中运行命令。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此操作的输出将类似于下文。

```
{
    "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

示例 3：跨复制加密快照 AWS 区域

以下 AWS CLI 示例创建了sample-cluster-snapshot从 us-west-2 区域到 us-east-1 区域的副本。此命令是在 us-east-1 区域中调用。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此操作的输出将类似于下文。

```
{
    "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

示例 4：跨复制未加密的共享快照 AWS 区域

以下 AWS CLI 示例，账户-123456789012，创建了账户sample-cluster-snapshot共享的未加密集群快照的副本，该快照999999999999从 us-east-1 区域到 us-west-2 区域。在 us-west-2 区域中调用此命令。有关共享快照的更多信息，请参阅共享快照。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此操作的输出将类似于下文。

```
{
 "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

示例 5：跨复制加密的共享快照 AWS 区域

以下 AWS CLI 示例，账户-123456789012 创建账户sample-cluster-snapshot共享的加密集群快照的副本，999999999999从 us-east-1 区域到 us-west-2 区域。目标快照使用客户管理的 KMS 密钥进行加密：arn:aws:kms:us-west-2:123456789012:key/6c1f3264-1797-472b-ba37-03011e682d28。在 us-west-2 区域中调用此命令。有关共享快照的更多信息，请参阅共享快照。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此操作的输出将类似于下文。

```
{
 "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**
>
> 有关对快照副本进行加密的更多信息，请参阅复制集群快照加密。
> 有关静态加密的更多信息，请参阅 Amazon DocumentDB 静态数据加密。

Amazon DocumentDB 集群快照共享

使用 Amazon DocumentDB，您可以按以下方式共享手动集群快照：

- 共享手动集群快照（无论是加密还是未加密）都允许授权 AWS 账户复制快照。
- 共享手动集群快照（无论是加密的还是未加密的）都使授权 AWS 账户能够直接从快照中恢复集群，而不必复制集群并从中恢复。

> **Note**
>
> 要共享自动集群快照，请通过复制自动快照来创建手动集群快照，然后共享该副本。此过程也适用于 AWS 备份生成的资源。

您可以与最多 20 个其他人共享手动快照 AWS 账户。您也可以将未加密的手动快照作为公有快照进行共享，这样所有 账户均可使用此快照。当以公有快照形式共享快照时，确保不要将您的私有信息包含在任何公有快照之中。

当与其他人共享手动快照 AWS 账户，并且使用 AWS CLI 或 Amazon DocumentDB API 从共享快照恢复集群时，必须将共享快照的亚马逊资源名称 (ARN) 指定为快照标识符。

共享加密的快照

以下限制适用于共享加密快照：

- 您无法公开共享加密的快照。
- 您无法共享已使用共享快照的账户的默认 AWS KMS 加密密钥加密的快照。

按照以下步骤共享加密的快照。

1. 与您希望能够访问快照的所有账户共享用于加密快照的 AWS Key Management Service (AWS KMS) 加密密钥。

您可以通过将其他 AWS 账户添加到密钥策略中来与其他账户共享 AWS KMS 加密 AWS KMS 密钥。有关更新密钥策略的详细信息，请参阅AWS Key Management Service 开发人员指南中的在 AWS KMS 中使用密钥策略。有关创建密钥策略的示例，请参阅本主题下文中的 创建 IAM 策略以启用加密快照的复制。

2. 如下所示，使用与其他账户共享加密快照。 AWS CLI

允许访问 AWS KMS 加密密钥

AWS 账户 要让其他人复制从您的账户共享的加密快照，则您与之共享快照的账户必须有权访问加密快照的 AWS KMS 密钥。要允许其他账户访问 AWS KMS 密钥，请在密钥策略中 AWS KMS 使用您作为委托人共享的账户的 ARN 更新该密钥的 AWS KMS 密钥策略。然后允许 `kms:CreateGrant` 操作。

在您授予账户访问您的 AWS KMS 加密密钥的权限后，要复制您的加密快照，该账户必须创建一个 AWS Identity and Access Management (IAM) 用户（如果还没有）。此外，该账户还必须向该 IAM 用户附加一个 IAM 策略，允许该用户使用您的 AWS KMS 密钥复制加密快照。由于 AWS KMS 安全限制，该账户必须是 IAM 用户，并且不能是根 AWS 账户 身份。

在以下密钥策略示例中，用户 123451234512 是加密密钥的所有者。 AWS KMS 用户 123456789012 是要与之共享密钥的账户。此更新的密钥政策允许账户访问 AWS KMS 密钥。它通过将用户 123456789012 的根 AWS 账户 身份的 ARN 作为策略的委托人并允许该操作来实现此目的。`kms:CreateGrant`

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}}
        }
    ]
}
```

创建 IAM 策略以启用加密快照的复制

当外部用户 AWS 账户 有权访问您的 AWS KMS 密钥时，该账户的所有者可以创建策略，允许为该账户创建的 IAM 用户复制使用该 AWS KMS 密钥加密的加密快照。

以下示例显示了可以附加到 AWS 账户 123456789012 的 IAM 用户的策略。该策略允许 IAM 用户从账户 123451234512 中复制已在 us-west-2 区域使用密钥 AWS KMS 加密的共享快照。`c989c1dd-a3f2-4a5d-8d96-e793d082ab26`

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
                }
            }
        }
    ]
}
```

有关更新密钥政策的详细信息，请参阅 AWS Key Management Service 开发人员指南中的在 AWS KMS中使用密钥政策。

共享快照

您可以使用或共享 Amazon DocumentDB 手动集群快照（或自动快照的副本）： AWS 管理控制台 AWS CLI

Using the AWS 管理控制台

要使用共享快照 AWS 管理控制台，请完成以下步骤：

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择快照。
3. 选择要共享的手动快照。
4. 在操作下拉菜单中，选择“共享”。
5. 为数据库快照可见性选择以选项之一。

   - 如果源未加密，请选择 P ubli c 以允许所有 AWS 账户从您的手动快照还原集群。或者选择 P ri vate，仅允许您指定的 AWS 账户从手动快照还原群集。

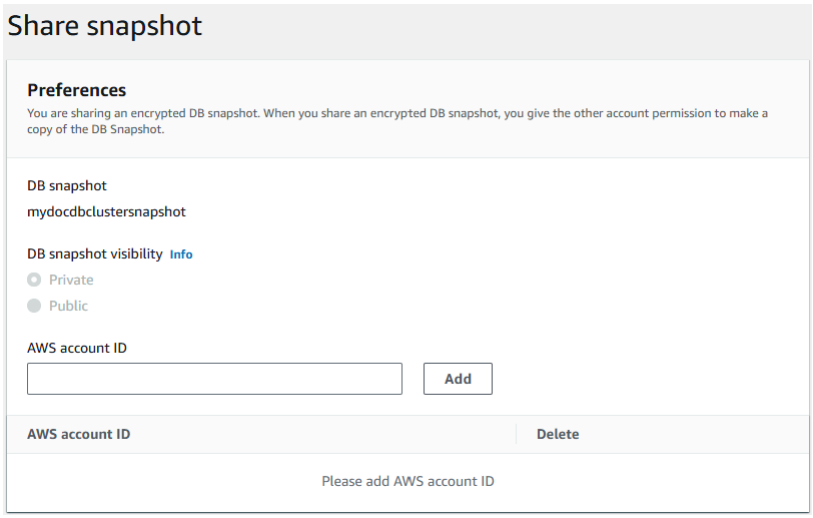
     > **Warning**
     >
     > 如果您将数据库快照可见性设置为 “公开”，则所有 AWS 账户都可以从您的手动快照还原集群并可以访问您的数据。请勿将包含私有信息的任何手动集群快照作为公有共享。

   - 如果源已加密，由于已加密的快照无法公开共享，DB snapshot visibility (数据库快照可见性) 将设为 Private (私密)。

     > **Note**
     >
     > 使用默认值加密的快照 AWS KMS key 无法共享。

6. 在AWS 账户 ID 中 AWS ，输入您想要允许从手动快照还原集群的账户的账户标识符，然后选择添加。重复此操作以添加其他 AWS 账户标识符，最多 20 个 AWS 账户。

   如果您在向允许的账户列表中添加 AWS 账户标识符时出错，则可以通过选择错误的 AWS 账户标识符右侧的 “删除” 将其从列表中删除。



7. 为所有要允许恢复手动快照的 AWS 账户添加标识符后，选择 “保存” 以保存所做的更改。

Using the AWS CLI

要使用共享快照，请使用 Amazon DocumentDB 操作`modify-db-snapshot-attribute`。AWS CLI使用`--values-to-add`参数为有权恢复手动快照 AWS 账户 的 ID 添加列表。

以下示例允许两个 AWS 账户 标识符 123451234512 和 123456789012 恢复名为的快照。`manual-snapshot1`它还会删除 `all` 属性值以将该快照标记为私有。

对于 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster-snapshot-attribute \
    --db-cluster-snapshot-identifier sample-cluster-snapshot \
    --attribute-name restore \
    --values-to-add '["123451234512","123456789012"]'
```

对于 Windows：

```
aws docdb modify-db-cluster-snapshot-attribute ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot ^
    --attribute-name restore ^
    --values-to-add '["123451234512","123456789012"]'
```

此操作的输出将类似于下文。

```
{
```

```
    "DBClusterSnapshotAttributesResult": {
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot",
        "DBClusterSnapshotAttributes": [
            {
                "AttributeName": "restore",
                "AttributeValues": [
                    "123451234512",
                    "123456789012"
                ]
            }
        ]
    }
}
```

要从列表中删除 AWS 账户 标识符，请使用`--values-to-remove`参数。以下示例阻止 AWS 账户 ID 123456789012 恢复快照。

对于 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster-snapshot-attribute \
    --db-cluster-snapshot-identifier sample-cluster-snapshot \
    --attribute-name restore \
    --values-to-remove '["123456789012"]'
```

对于 Windows：

```
aws docdb modify-db-cluster-snapshot-attribute ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot ^
    --attribute-name restore ^
    --values-to-remove '["123456789012"]'
```

此操作的输出将类似于下文。

```
{
    "DBClusterSnapshotAttributesResult": {
        "DBClusterSnapshotIdentifier": "sample-cluster-snapshot",
        "DBClusterSnapshotAttributes": [
            {
                "AttributeName": "restore",
                "AttributeValues": [
                    "123451234512"
```

```
                ]
            }
        ]
    }
}
```

从集群快照还原

Amazon DocumentDB（与 MongoDB 兼容）会为您的存储卷创建一个集群快照。可通过从集群快照还原来创建新集群。在还原集群时，您需提供用于还原的集群快照的名称以及还原创建的新集群的名称。您无法从快照还原到现有集群，因为还原时将新建一个新集群。

当从集群快照还原集群时：

- 此操作仅还原集群，而不还原集群的实例。您必须调用 `create-db-instance` 操作为还原的集群创建实例，并在 `--db-cluster-identifier` 中指定还原的集群的标识符。您只能在集群可用 后才能创建实例。
- 您无法将加密快照还原到未加密集群。但是，您可以通过指定密 AWS KMS 钥将未加密的快照还原到加密的集群。
- 要从加密快照恢复集群，您必须有权访问 AWS KMS 密钥。

> **Note**
>
> 您无法将 3.6 集群恢复为 4.0 集群，但可以从一个集群版本迁移到另一个集群版本。有关更多信息，请转至 迁移到 Amazon DocumentDB。

Using the AWS 管理控制台

以下过程说明如何使用 Amazon DocumentDB 管理控制台从集群快照中还原 Amazon DocumentDB 集群。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择快照，然后选择要用于还原集群的快照左侧的按钮。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标
> (≡)。

3. 在 Actions (操作) 菜单上，选择 Restore (还原)。
4. 在还原快照页面上，填写配置部分。

   a. 集群标识符：新集群的名称。您可以接受 Amazon DocumentDB 提供的名称或键入您喜欢的名称。提供的 Amazon DocumentDB 名称采用 `docdb-` 加上 UTC 时间戳的格式；例如，`docdb-yyyy-mm-dd-hh-mm-ss`。

   b. 实例类：新集群的实例类。您可以接受默认实例类或从下拉列表中选择实例类。

   c. 实例数：要使用此集群创建的实例的数量。您可以接受 3 个实例的默认值（1 个主副本 read/write 和 2 个只读副本），也可以从下拉列表中选择实例数量。
5. 对于集群存储配置，请选择一个存储选项。

> **Note**
>
> 亚马逊 DocumentDB I/O-Optimized 存储配置仅适用于亚马逊 DocumentDB 5.0 和 8.0 引擎版本。

6. 如果您对集群配置满意，请选择 Restore cluster (还原集群) 并等待集群还原。
7. 如果您更希望更改某些配置（如指定非默认 Amazon VPC 或安全组），请在页面底部左边选择显示高级设置，然后继续执行以下步骤。

   a. 完成 Network settings (网络设置) 部分。

      - 虚拟私有云（VPC）：接受当前 VPC，或者从下拉列表中选择一个 VPC。
      - 子网组：接受 `default` 子网组，或从下拉列表中选择一个子网组。
      - VPC 安全组：接受 `default (VPC)` 安全组，或从列表中选择一个安全组。

   b. 完成集群选项部分。

      - 数据库端口：接受默认端口 `27017`，或使用向上或向下箭头来设置要用于应用程序连接的端口。

   c. 完成加密部分。

- 静态加密：如果您的快照已加密，那么这些选项对您不可用。如果它未加密，您可以选择以下选项之一：
  - 要加密所有集群的数据，请选择启用静态加密。如果您选择此选项，则必须指定一个 KMS 密钥。
  - 要让集群的数据不加密，请选择禁用静态加密。如果您选择此选项，您便已完成加密部分。
- AWS KMS 密钥-从下拉列表中选择以下选项之一：
  - （默认）aws/rds-账号和 AWS KMS 密钥 ID 列在此选项后面。
  - Customer-managed key — 只有在 AWS Identity and Access Management (IAM) 控制台中创建了 IAM 加密密钥时，此选项才可用。您可以选择该密钥来加密集群。
  - 输入密钥 ARN — 在 ARN 框中，输入密钥的亚马逊资源名称 (ARN)。 AWS KMS ARN 的格式为 `arn:aws:kms:<region>:<accountID>:key/<key-id>`。

d. 完成 Log exports (日志导出) 部分。

- 选择要发布到的日志类型 CloudWatch-选择以下选项之一：
  - 已启用-允许您的集群将 DDL 日志导出到 Amazon CloudWatch 日志。
  - 已禁用-阻止您的集群将 DDL 日志导出到 Amazon CloudWatch 日志。Disabled (已禁用) 为默认值。
- IAM 角色：从列表中选择 RDS 服务相关角色。

e. 完成 Tags (标签) 部分。

- 添加标签：在密钥框中，输入集群标签的名称。在 Value (值) 框中，可以选择输入标签值。标签与 AWS Identity and Access Management (IAM) 策略一起使用，用于管理对 Amazon DocumentDB 资源的访问权限并控制可以对资源应用哪些操作。

f. 完成 Deletion protection (删除保护) 部分。

- 启用删除保护：防止集群被意外删除。启用该选项后，您将无法删除集群。

8. 选择 Restore cluster (还原集群)。

Using the AWS CLI

要使用从快照还原集群 AWS CLI，请使用带有以下参数的`restore-db-cluster-from-snapshot`操作。有关更多信息，请参阅 RestoreDBClusterFromSnapshot。

- **--db-cluster-identifier**：必需。操作创建的集群的名称。在执行此操作之前，不能存在此名称的集群。

  集群命名约束：

  - 长度为 [1-63] 个字母、数字或连字符。
  - 第一个字符必须是字母。
  - 不能以连字符结尾或包含两个连续的连字符。
  - 每个区域的 Amazon RDS、Neptune 和 Amazon DocumentDB 中的所有集群都必须是唯一 AWS 账户的。
- **--snapshot-identifier**：必需。用于自其还原的快照的名称。具有此名称的快照必须存在并且处于可用 状态。
- **--engine**：必需。必须是 docdb。
- **--storage-type standard | iopt1**：可选。默认值：standard。
- **--kms-key-id**— 可选。还原加密快照或从未加密快照还原时加密集群时使用的密 AWS KMS 钥标识符的 ARN。无论快照是否加密，提供 AWS KMS 密钥 ID 都会使用该 AWS KMS 密钥对还原的集群进行加密。

  --kms-key-id 的格式为 arn:aws:kms:<region>:<accountID>:key/<key-id>。如果不为 --kms-key-id 参数指定值，则会出现以下情况：

  - 如果中的--snapshot-identifier快照已加密，则使用用于加密快照的相同 AWS KMS 密钥对还原的集群进行加密。
  - 如果 --snapshot-identifier 中的快照未加密，还原的集群也不会加密。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此操作的输出将类似于下文。

```
{
    "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
    }
}
```

在集群状态为可用 后，请为集群创建至少一个实例。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此操作的输出将类似于下文。

```
{
    "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"
    }
}
```

还原到某个时间点

您可以使用 AWS 管理控制台 或 AWS Command Line Interface (AWS CLI) 将集群还原到集群备份保留期内的任何时间点。

> **Note**
>
> 您无法将 3.6 集群的时间点恢复到 4.0 集群，但可以从一个集群版本迁移到另一个集群版本。有关更多信息，请转至 迁移到 Amazon DocumentDB。

在将集群还原至某个时间点时，请牢记以下几点。

- 创建新集群时将使用与源集群相同的配置，区别在于创建新集群时将使用默认参数组。要将新集群的参数组设置为源集群的参数组，请在新集群可用 后对其进行修改。有关修改集群的更多信息，请参阅修改 Amazon DocumentDB 集群。

Using the AWS 管理控制台

可以使用 AWS 管理控制台完成以下操作，以将集群还原到该集群的备份保留期内的某个时间点。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。在集群列表中，选择要还原的集群左侧的按钮。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在 Actions (操作) 菜单上，选择 Restore to point in time (还原到时间点)。
4. 完成 Restore time (恢复时间) 部分，用于指定将还原到的日期和时间。
   a. 还原日期：选择或输入介于最早还原时间和最新还原时间之间的日期。
   b. 还原时间：选择或输入介于最早还原时间和最新还原时间之间的小时、分钟和秒。
5. 填写 Configuration (配置) 部分。
   a. 集群标识符：接受默认标识符，或者输入您喜欢的标识符。

      集群命名约束：

      - 长度为 [1-63] 个字母、数字或连字符。
      - 第一个字符必须是字母。

- 不能以连字符结尾或包含两个连续的连字符。
- 每个区域的 Amazon RDS、Neptune 和 Amazon DocumentDB 中的所有集群都必须是唯一 AWS 账户的。

b. 实例类：从下拉列表中，选择要用于集群实例的实例类。

c. 实例数：从下拉列表中，选择还原此集群时您希望创建的实例的数量。

6. 对于集群存储配置，请选择一个存储选项。

> **Note**
>
> 亚马逊 DocumentDB I/O-Optimized 存储配置仅适用于亚马逊 DocumentDB 5.0 和 8.0 引擎版本。

7. 可选。要配置网络设置、集群选项和启用日志导出，请选择 Show advanced settings (显示高级设置)，然后完成以下各部分。否则，请继续下一步。

- Network settings (网络设置)
  1. 虚拟私有云（VPC）：从下拉列表中，选择要用于此集群的 VPC。
  2. 子网组：从下拉列表中，为此集群选择子网组。
  3. VPC 安全组：从下拉列表中，为此集群选择 VPC 安全组。

- Cluster options (集群选项)
  1. 端口：接受默认端口 (27017)，或者使用向上或向下箭头来设置与此集群进行通信的端口。

- 日志导出
  1. 审核日志-选择此选项可启用将审核日志导出到 Amazon CloudWatch Logs。如果您选择此选项，则必须在集群的自定义参数组中启用 `audit_logs`。有关更多信息，请参阅 审核 Amazon DocumentDB 事件。
  2. Profiler 日志-选择此选项可启用将操作分析器日志导出到 Ama CloudWatch zon Logs。如果您选择此选项，还必须在集群的自定义参数组中修改以下参数：
     - `profiler`：设置为 `enabled`。
     - `profiler_threshold_ms`：设置为值 `[0-INT_MAX]`，以设置分析操作的阈值。

- `profiler_sampling_rate`：设置为值 `[0.0-1.0]`，以设置要分析的缓慢操作的百分比。

有关更多信息，请参阅 分析 Amazon DocumentDB 操作。

3. Profiler 日志 — 将分析器日志导出到 Amazon CloudWatch
4. IAM 角色：从下拉列表中，选择 RDS 服务相关角色。

- 标签

1. 添加标签：在密钥框中，输入集群标签的名称。在 Value (值) 框中，可以选择输入标签值。标签与 AWS Identity and Access Management (IAM) 策略结合使用，以管理对 Amazon DocumentDB 资源的访问并控制可将什么操作应用于资源。

- 删除保护

1. 启用删除保护：防止集群被意外删除。启用该选项后，您将无法删除集群。

8. 要还原集群，请选择 Create cluster (创建集群)。或者，您可以选择 Cancel (取消) 以取消操作。

Using the AWS CLI

要使用快照的备份保留期将集群还原到某个时间点，请使用带有以下参数的 `restore-db-cluster-to-point-in-time` 操作。

- **`--db-cluster-identifier`**：必填项。要创建的新集群的名称。在执行该操作之前，不能存在此集群。参数值必须满足以下约束。

  集群命名约束：

  - 长度为 [1-63] 个字母、数字或连字符。
  - 第一个字符必须是字母。
  - 不能以连字符结尾或包含两个连续的连字符。
  - 每个区域的 Amazon RDS、Neptune 和 Amazon DocumentDB 中的所有集群都必须是唯一 AWS 账户的。

- **`--restore-to-time`**：要将集群还原到的 UTC 日期和时间。例如 `2018-06-07T23:45:00Z`。

  时间约束：

  - 必须在集群的最新可还原时间之前。
  - 如果未提供 `--use-latest-restorable-time` 参数，则必须指定。

  - 如果 --use-latest-restorable-time 参数为 true，则无法指定。
  - 如果 --restore-type 参数值为 copy-on-write，则无法指定。
- **--source-db-cluster-identifier**：要从中还原的源集群的名称。该集群必须存在且可用。
- **--use-latest-restorable-time** 或 **--no-use-latest-restorable-time**：是否还原至最近的可还原备份时间。如果未提供 --restore-to-time 参数，则无法指定。
- **--storage-type standard | iopt1**：可选。默认值：standard。

该 AWS CLI 操作restore-db-cluster-to-point-in-time仅恢复集群，而不会恢复该集群的实例。您必须调用 create-db-instance 操作为还原的集群创建实例，并在 --db-cluster-identifier 中指定还原的集群的标识符。仅当完成 restore-db-cluster-to-point-in-time 操作并且还原的集群可用 后，您才能创建实例。

Example

以下示例从快照 sample-cluster-snapshot 创建到最近可还原时间的 sample-cluster-restored。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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

以下示例从快照 sample-cluster-snapshot 创建到 2018 年 12 月 11 日 03:15 (UTC) (在 sample-cluster 的备份保留期内) 的 sample-cluster-restored。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此操作的输出将类似于下文。

```
{
    "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
    }
}
```

删除集群快照

手动快照是一种完全备份，只有使用 AWS 管理控制台 或手动删除它时才会将其删除 AWS CLI。无法手动删除自动快照，因为自动快照仅在快照保留期已过或删除快照的集群时才会被删除。

Using the AWS 管理控制台

要使用删除手动集群快照 AWS 管理控制台，请完成以下步骤。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择快照。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在快照列表中，选择要删除的快照左侧的按钮。快照的类型必须为手动。

   1. 您可以通过检查快照类型列下是否列出 `manual` 或 `automatic`，来验证快照 Type（类型）是否为 manual（手动）。
4. 从 Actions 菜单中选择 Delete 。如果 Delete (删除) 选项不可用，您可能选择了自动快照。
5. 要删除快照，请在删除确认屏幕中选择 Delete (删除)。要保留快照，请选择 Cancel (取消)。

Using the AWS CLI

Amazon DocumentDB 手动集群快照是可以使用 AWS CLI手动删除的完整备份。您不能手动删除自动快照。

要使用删除手动集群快照 AWS CLI，请使用带有以下参数的`delete-db-cluster-snapshot`操作。

参数

- **`--db-cluster-snapshot-identifier`**：必需。要删除的手动快照的名称。

以下示例将删除集群快照 `sample-cluster-snapshot`。

对于 Linux、macOS 或 Unix：

```
aws docdb delete-db-cluster-snapshot \
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

对于 Windows：

```
aws docdb delete-db-cluster-snapshot ^
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

此操作的输出将列出您删除的集群快照的详细信息。

管理 Amazon DocumentDB 资源

以下各节介绍各种组件及其用于管理 Amazon DocumentDB（与 MongoDB 兼容）实现的相关任务。

主题

- Amazon DocumentDB 操作任务概述
- Amazon DocumentDB 全局集群概览
- 管理 Amazon DocumentDB 集群
- 管理 Amazon DocumentDB 实例
- 管理 Amazon DocumentDB 子网组
- Amazon DocumentDB 高可用性和复制
- 管理 Amazon DocumentDB 索引
- 管理集合级文档压缩
- 在 Amazon DocumentDB 8.0 中管理基于字典的压缩
- 管理 Amazon DocumentDB 事件
- 选择区域和可用区
- 管理 Amazon DocumentDB 集群参数组
- 了解 Amazon DocumentDB 端点
- 了解 Amazon DocumentDB Amazon 资源名称（ARN）
- 标记 Amazon DocumentDB 资源
- 维护 Amazon DocumentDB
- 了解服务相关角色
- 将变更流与 Amazon DocumentDB 结合使用
- 在 Amazon DocumentDB 8.0 中使用排序规则
- 在 Amazon DocumentDB 8.0 中使用视图
- 使用 AWS Lambda 有变更流

Amazon DocumentDB 操作任务概述

本部分讲述 Amazon DocumentDB 集群的操作任务以及如何使用 AWS CLI完成这些任务。

主题

- 向 Amazon DocumentDB 集群添加副本
- 描述集群和实例
- 创建集群快照
- 从快照还原
- 从集群中删除实例
- 删除集群

向 Amazon DocumentDB 集群添加副本

在您创建您的 Amazon DocumentDB 集群的主实例之后，您可以添加一个或多个副本。副本是只读实例，它有两个用途：

- 可扩展性：如果您有大量需要同时访问的客户端，则可以添加多个副本以进行读取扩展。
- 高可用性：如果主实例发生故障，Amazon DocumentDB 会自动故障转移到一个副本实例并将该副本实例指定为新的主实例。如果副本发生故障，则集群中的其他实例仍能够处理请求，直到发生故障的节点恢复为止。

每个 Amazon DocumentDB 集群可支持多达 15 个副本。

> **Note**
>
> 为实现最大容错能力，您应在不同的可用区中部署副本。这可以确保您的 Amazon DocumentDB 集群即使在整个可用区变得不可用时也能够继续正常运行。

以下 AWS CLI 示例说明如何添加新的副本。`--availability-zone` 参数将副本置于指定的可用区中。

```
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
```

描述集群和实例

以下 AWS CLI 示例列出了一个区域中的所有 Amazon DocumentDB 集群。对于某些管理功能（如集群和实例生命周期管理），Amazon DocumentDB 利用与 Amazon RDS 共享的操作技术。`filterName=engine,Values=docdb` 筛选器参数仅返回 Amazon DocumentDB 集群。

有关描述和修改集群的更多信息，请参阅Amazon DocumentDB 集群生命周期。

```
aws docdb describe-db-clusters --filter Name=engine,Values=docdb
```

此操作的输出将类似于下文。

```
{
    "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",
            ...
        }
    ]
}
```

以下 AWS CLI 示例列出了亚马逊文档数据库集群中的实例。有关描述和修改集群的更多信息，请参阅Amazon DocumentDB 实例生命周期。

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterMembers]'
```

输出如下所示。此输出中有两个实例。主实例是 sample-instance-1 ("IsClusterWriter": true)。此外，还有一个副本实例，即 sample-instance2 ("IsClusterWriter: false")。

```
[
    [
        [
            {
                "DBInstanceIdentifier": "sample-instance-1",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            },
            {
                "DBInstanceIdentifier": "sample-cluster-2",
                "IsClusterWriter": false,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            }
        ]
    ]
]
```

创建集群快照

集群快照是 Amazon DocumentDB 集群中的数据的完整备份。创建快照后，Amazon DocumentDB 将直接从集群卷读取数据。因此，即使您的集群当前没有任何实例在运行，您也可以创建快照。创建快照所用时间因集群卷大小而异。

Amazon DocumentDB 支持自动备份，自动备份在每天的首选备份时段：白天的一个 30 分钟时段进行。以下 AWS CLI 示例说明如何查看集群的备份窗口：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].PreferredBackupWindow'
```

此输出显示备份时段（UTC 时间）：

```
[
    "00:18-00:48"
]
```

您可以在创建 Amazon DocumentDB 集群时定义备份时段。您还可以更改备份时段，如以下示例所示：如果您不定义备份时段，Amazon DocumentDB 会自动给您的集群分配一个时段。

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --preferred-backup-window "02:00-02:30"
```

除了自动备份以外，您还可以随时手动创建集群快照。当您执行此操作时，您指定要备份的集群以及快照的唯一名称，以便稍后从此快照还原。

以下 AWS CLI 示例说明如何创建数据快照。

```
aws docdb create-db-cluster-snapshot \
    --db-cluster-identifier sample-cluster \
    --db-cluster-snapshot-identifier sample-cluster-snapshot
```

从快照还原

您可以将集群快照还原到新的 Amazon DocumentDB 存储位置。为此，您需要提供快照的名称和新集群的名称。您无法从快照还原到现有集群；但 Amazon DocumentDB 会创建一个新集群，然后使用快照数据填充到其中。

以下示例显示集群 `sample-cluster` 的所有快照。

```
aws docdb describe-db-cluster-snapshots \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusterSnapshots[*].[DBClusterSnapshotIdentifier,SnapshotType,Status]'
```

输出如下所示。手动快照就是您手动创建的快照，而自动快照是在集群备份时段内由 Amazon DocumentDB 创建的。

```
[
        "sample-cluster-snapshot",
        "manual",
        "available"
    ],
    [
        "rds:sample-cluster",
        "automated",
        "available"
    ]
]
```

以下示例演示如何从快照还原 Amazon DocumentDB 集群。

```
aws docdb restore-db-cluster-from-snapshot \
    --engine docdb \
    --db-cluster-identifier new-sample-cluster \
    --snapshot-identifier sample-cluster-snapshot
```

新的集群没有任何实例与之相关联；因此，如果您想要与集群进行交互，您必须给它添加实例。

```
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
```

您可以使用以下 AWS CLI 操作来监控集群和实例的创建进度。当集群和实例处于可用状态时，您可连接到新集群的终端节点并访问您的数据。

```
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]'
```

从集群中删除实例

Amazon DocumentDB 将所有数据都存储在集群卷中。即使您从集群中删除所有实例，数据仍会保留在该集群卷中。如果您需要再次访问数据，则可随时向该集群中添加实例，并在停止处恢复操作。

以下示例显示如何从 Amazon DocumentDB 集群中删除实例。

```
aws docdb delete-db-instance \
    --db-instance-identifier sample-instance
```

删除集群

在您删除 Amazon DocumentDB 集群之前，您必须先删除其所有实例。以下 AWS CLI 示例返回有关集群中实例的信息。如果此操作会返回任何实例标识符，您必须删除各个实例。有关更多信息，请参阅从集群中删除实例。

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterMembers[*].DBInstanceIdentifier'
```

当没有任何剩余的实例时，您就可以删除集群了。此时，您必须选择以下选项之一：

- 创建最终快照：将所有集群数据捕获到一个快照中，以便您以后可以使用这些数据重新创建一个新实例。下例说明具体做法：

  ```
  aws docdb delete-db-cluster \
  ```

```
    --db-cluster-identifier sample-cluster \
    --final-db-snapshot-identifier sample-cluster-snapshot
```

- 跳过最终快照：永久舍弃所有集群数据。此操作无法撤消。下例说明具体做法：

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --skip-final-snapshot
```

Amazon DocumentDB 全局集群概览

什么是全局集群？

全局集群由一个主区域和最多 10 个只读辅助区域组成。您可以直接向主区域中的主数据库集群发出写入操作，Amazon DocumentDB 将使用专用基础设施自动将数据复制到辅助区域。延迟通常不到一秒。

全局集群有何用处？

- 从区域范围的停机中恢复：如果发生区域范围的停机，您可以在几分钟内将其中一个辅助集群提升为主集群，典型的恢复时间目标 (RTO) 不到一分钟。恢复点目标（RPO）通常以秒为单位进行测量，但这取决于发生故障时整个网络的滞后时间。
- 全球读取本地延迟：如果您在世界各地设有办事处，则可以使用 全局数据库在主 区域将其主要信息来源保持最新。您其他区域的办事处可以访问各自区域中的信息，存在本地延迟。
- 可扩展辅助集群：您可以通过向辅助 区域添加更多只读实例来扩展辅助集群。辅助集群为只读模式，因此它最多可以支持 16 个只读副本实例，而不符合单个集群通常 15 个此类副本的限制。
- 从主到辅助集群快速复制：全局集群执行的复制对主数据库集群造成的性能影响不大。数据库实例的资源完全专用于承担应用程序读取和写入工作负载。
- 更改流 — 您可以从全局群集中的辅助群集读取更改流，从而减轻主群集的负载，并允许您扩展辅助区域的读取工作负载。

目前全局集群的局限性有哪些？

- Amazon DocumentDB v3.6 不支持全局集群。
- 除 db.t3、db.t4g 和 db.r4 之外的所有实例类型都支持全局集群。

- 全局集群在以下区域不可用：南美洲（圣保罗）、欧洲地区（米兰）、中国（北京）和中国（宁夏）。
- 当各区域使用不同引擎版本时，不支持切换和全局失效转移。当引擎版本不匹配时，支持手动失效转移。
- 只有主集群才能执行写入操作。执行写操作的客户端连接到主集群的集群端点。
- 您的集群最多可以拥有 10 个辅助区域和一个主区域。
- 辅助集群无法停止。如果主集群有与之关联的辅助集群，则无法将其停止。只有没有辅助集群的区域集群才能停止。
- 在某些情况下，附加到辅助集群的副本可能会重新启动。如果主区域的实例重新启动或发生故障转移，辅助区域中的副本也会重新启动。随后集群将不可用，直到所有副本与主数据库集群的写入器实例恢复同步。此行为是预期的。在更改主集群之前，请务必了解对全局集群的影响。

主题

- 快速入门指南：全局集群
- 管理 Amazon DocumentDB 全局集群
- 连接到 Amazon DocumentDB 全局集群
- 监控 Amazon DocumentDB 全局集群
- 灾难恢复和 Amazon DocumentDB 全局集群

快速入门指南：全局集群

主题

- 配置
- 创建 Amazon DocumentDB 全局集群
- 添加一个 AWS 区域 到亚马逊 DocumentDB 全球集群
- 对您的 Amazon DocumentDB 全局集群使用快照

配置

Amazon DocumentDB 全球集群至少跨越两个。 AWS 区域主区域支持具有一个主（写入器）实例和多达 15 个副本实例的集群，而辅助区域运行一个完全由多达 16 个副本实例组成的只读集群。一个全局群集最多可以有 10 个辅助区域。该表列出了全局集群中允许的集群、实例和副本的最大数量。

说明	主要 AWS 区域	中学 AWS 区域
集群	1	10（最大值）
写入器实例	1	0
Read-only 每个集群的实例（Amazon DocumentDB 副本）	15（最大值）	16（合计）
Read-only 实例（允许的最大实例，给定辅助区域的实际数量）	15：s	s = 辅助节点的总数 AWS 区域

集群具有以下具体要求：

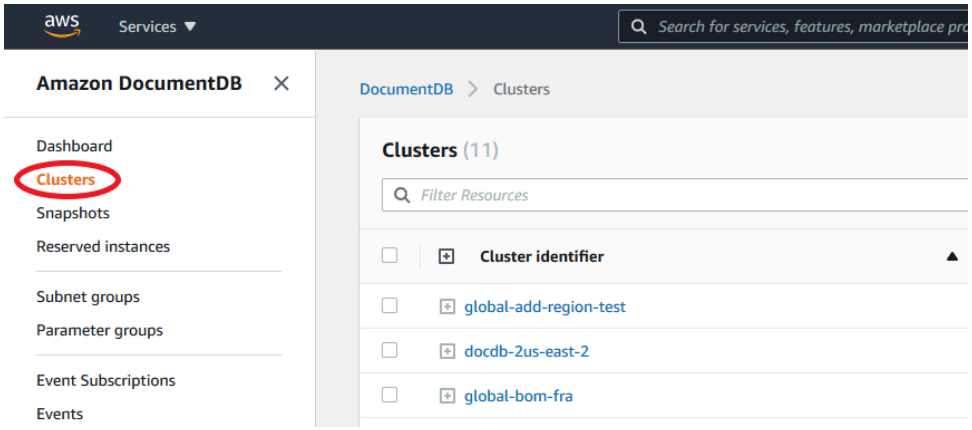
- 数据库实例类要求-您仅可以使用 db.r5 类和 db.r6g 实例类。
- AWS 区域 要求 — 主群集必须位于一个区域中，并且至少一个辅助群集必须位于同一账户的不同区域。您最多可以创建 10 个辅助（只读）集群，每个集群必须位于不同的区域。换句话说，没有两个集群可位于同一个区域中。
- 命名要求：为每个集群选择的名称在所有区域中必须唯一。即使不同的集群位于不同的区域中，也不能对其使用相同的名称。

创建 Amazon DocumentDB 全局集群

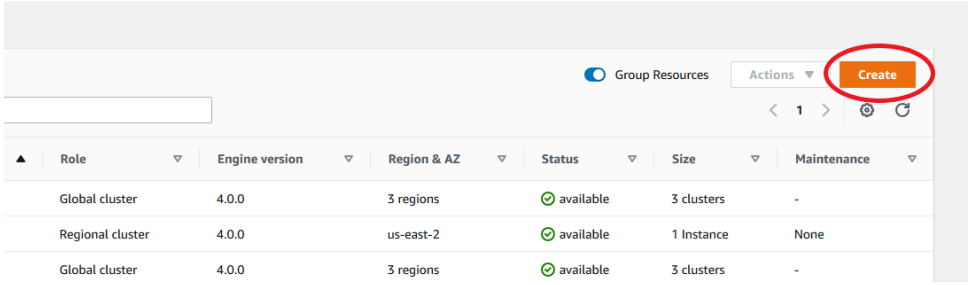
您是否准备好构建您的第一个全球集群？ 在本节中，我们将说明如何使用 AWS 管理控制台 或 AWS CLI 并按照以下说明创建包含新数据库集群和实例的全新全局集群。

使用 AWS 管理控制台

1. 在中 AWS 管理控制台，导航到亚马逊 Documen tDB。
2. 抵达 Amazon DocumentDB 控制台后，选择集群。

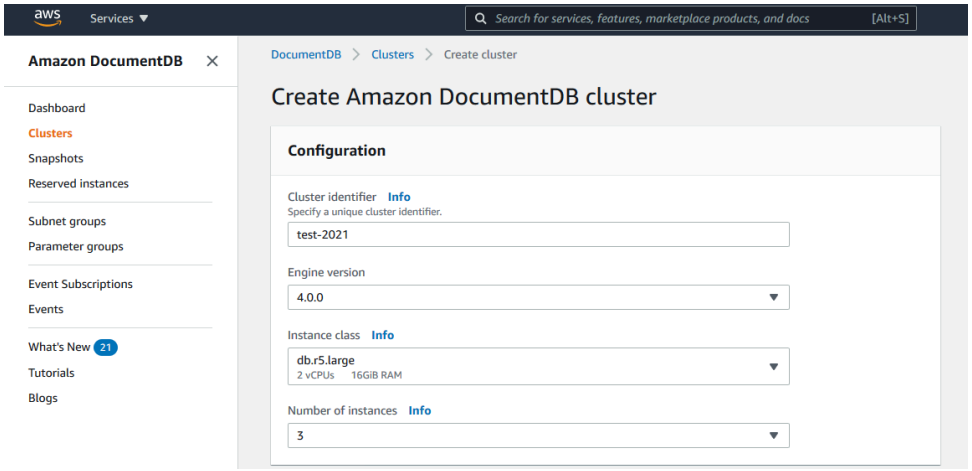


3. 选择创建。

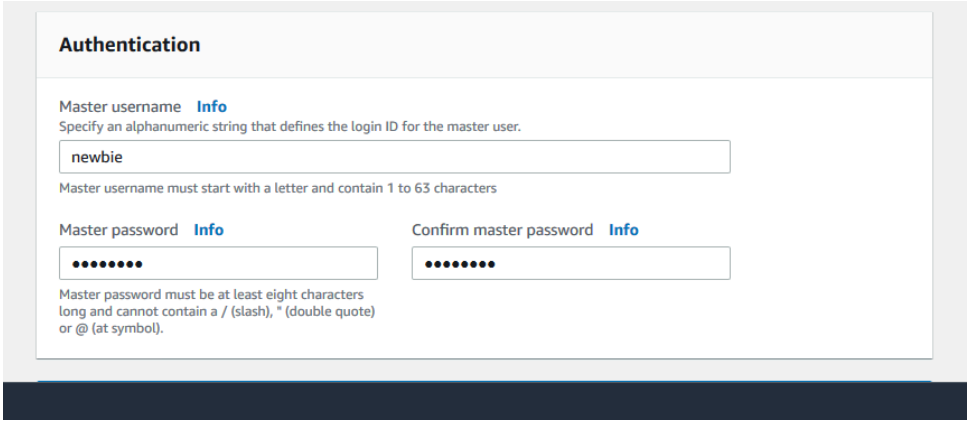


4. 相应地填写创建 Amazon DocumentDB 集群表单的配置部分：

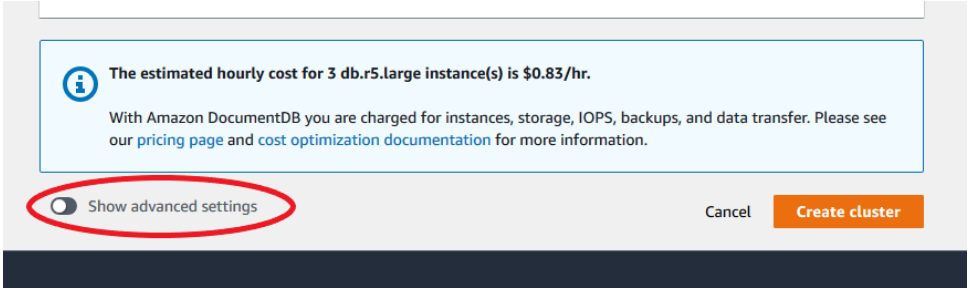
   - 实例标识符：您可以输入此实例的唯一标识符，或允许 Amazon DocumentDB 基于集群标识符提供实例标识符。
   - 引擎版本：选择 4.0.0
   - 实例类：选择 db.r5.large
   - 实例数目：选择 3。



5. 在身份验证部分，填写主用户名和主密码。

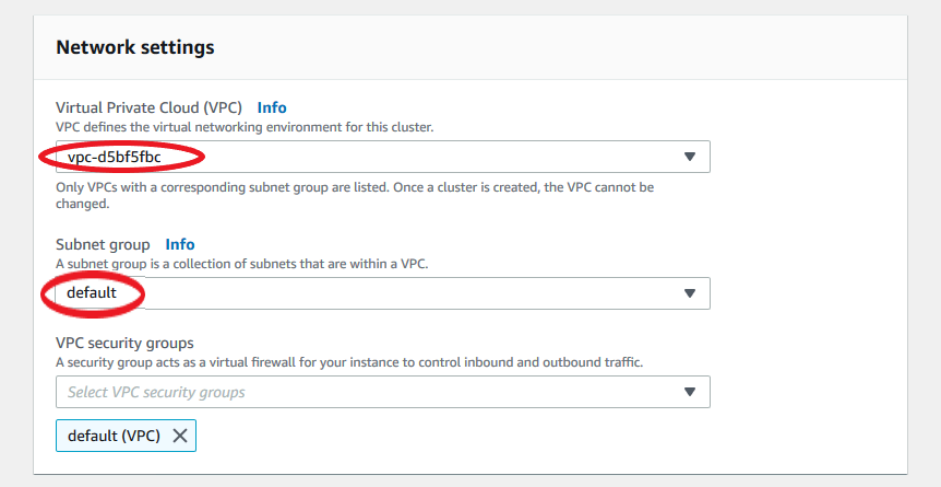


6. 选择显示高级设置。

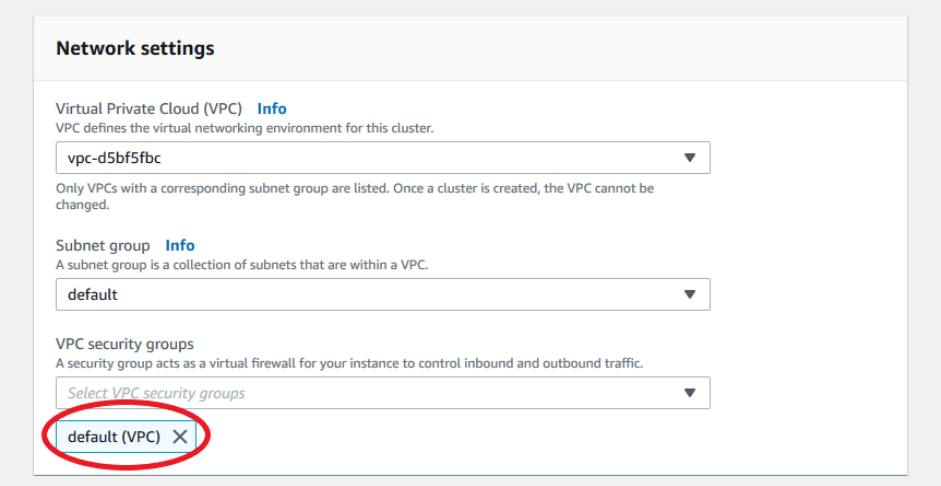


7. 在网络设置 部分中：

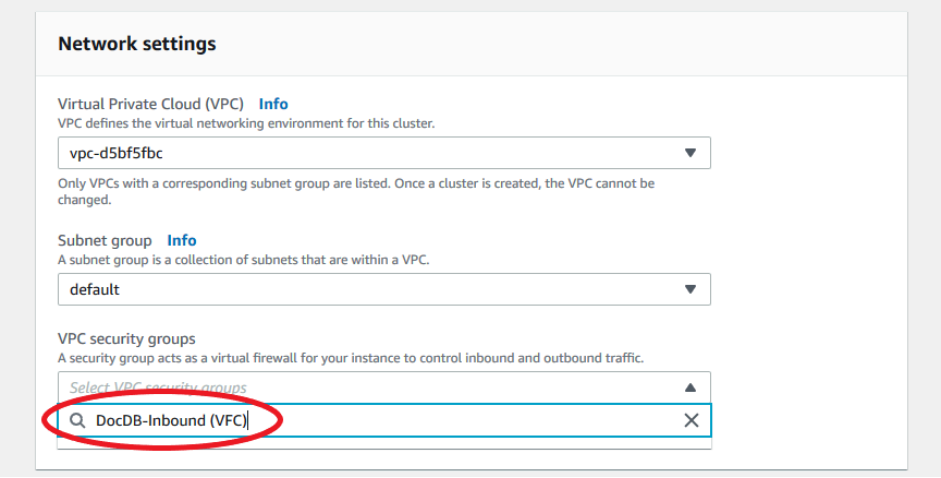
   - 保留虚拟私有云（VPC）和子网组的默认选项。



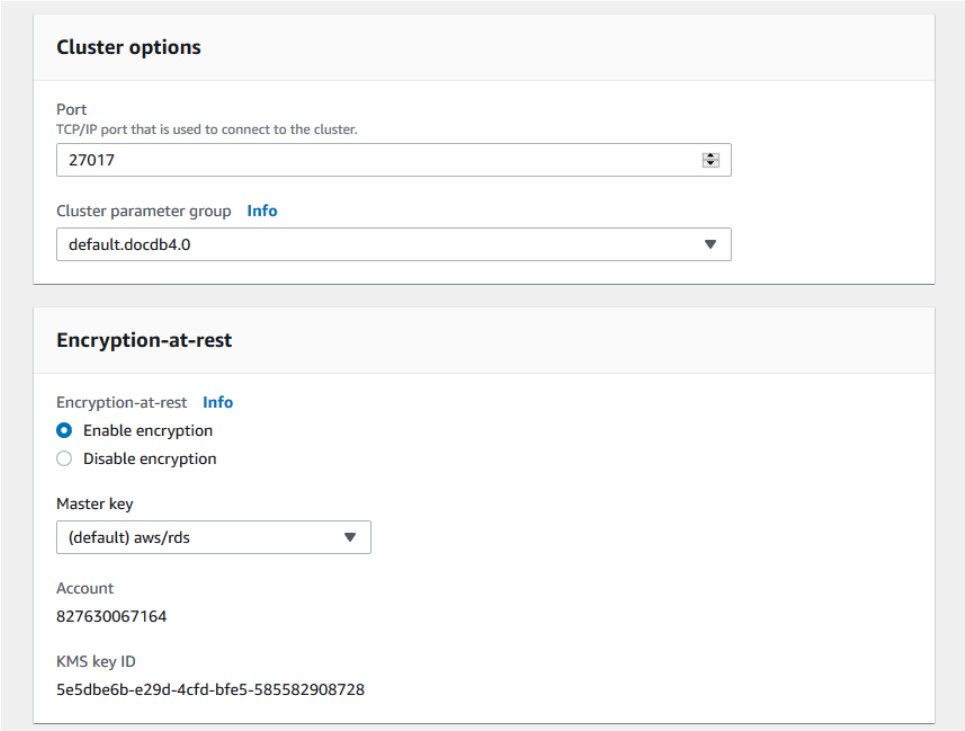
   - 对于 VPC 安全组，应已添加默认（VPC）。



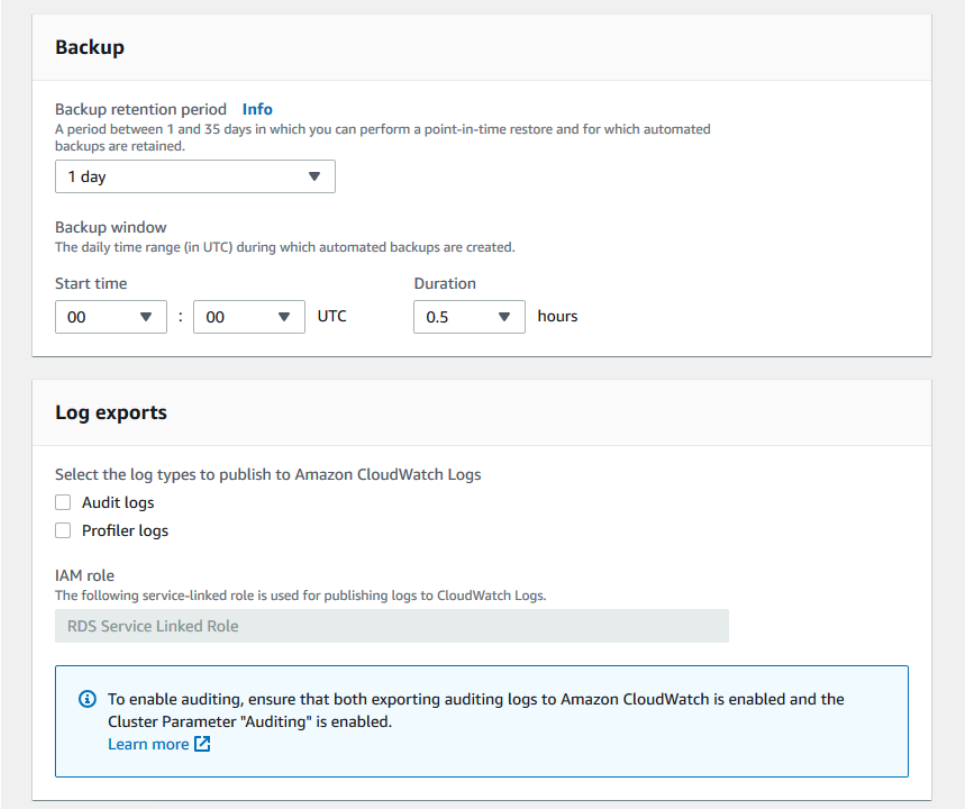
- 在 VPC 安全组字段中键DocDB入并选择 DocDB-Inbound (VFC)。



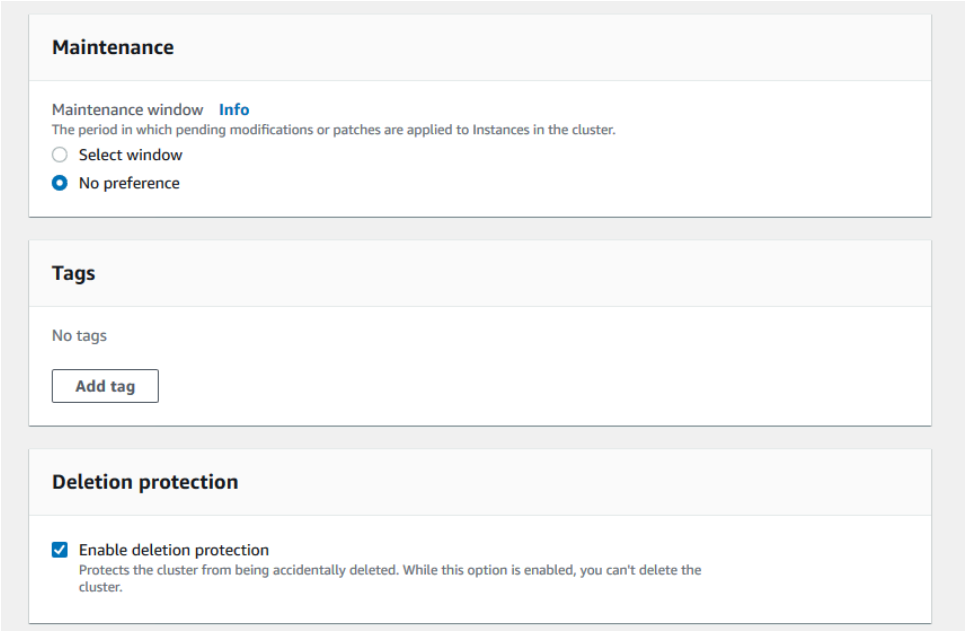
8. 对于群集选项和 Encryption-at-rest，保留默认选项。



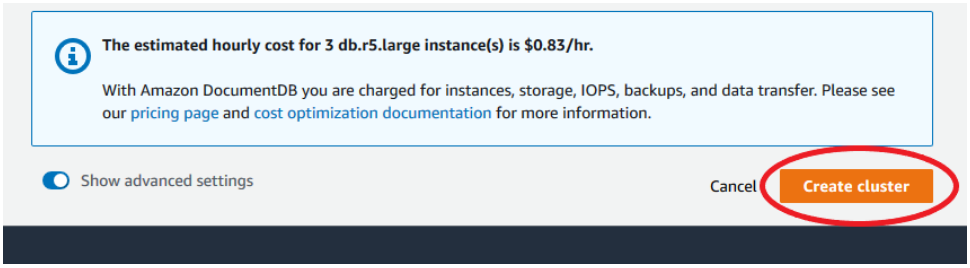
9. 对于备份和日志导出，保留默认选项。



10. 对于维护、标签和删除保护，保留默认选项。



11. 现在选择创建集群。



使用 AWS CLI

要创建 Amazon DocumentDB 区域集群，请调用 create-global-cluster。 AWS CLI以下 AWS CLI 命令创建名为的 Amazon DocumentDB 集群。`global-cluster-id`有关删除保护的更多信息，请参阅 删除 Amazon DocumentDB 集群。

此外，`--engine-version` 是一个默认成主引擎最新版本的可选参数。当前的默认引擎版本为`5.0.0`（注意：Amazon DocumentDB 8.0 可用，但必须明确指定为）。`8.0.0`发布主引擎新版本时，`--engine-version` 的默认引擎版本将更新，以反映最近的主引擎版本。因此，对于生产工作负载，尤其是那些依赖脚本、自动化或 CloudFormation 模板的工作负载，我们建议您明确指定预期的主要版本。`--engine-version`

如果未指定 `db-subnet-group-name` 或 `vpc-security-group-id`，则 Amazon DocumentDB 将使用给定区域的默认子网组和 Amazon VPC 安全组。

在以下示例中，将每个 *`user input placeholder`* 替换为您自己的信息。

对于 Linux、macOS 或 Unix：

```
aws docdb create-db-cluster \
      --global-cluster-identifier global-cluster-id \
      --source-db-cluster-identifier arn:aws:rds:us-east-1:111122223333:cluster-id
```

对于 Windows：

```
aws docdb create-db-cluster ^
      --global-cluster-identifier global-cluster-id ^
      --source-db-cluster-identifier arn:aws:rds:us-east-1:111122223333:cluster-id
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "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"
    }
}
```

创建集群需要几分钟时间。您可以使用 AWS 管理控制台 或 AWS CLI 来监控集群的状态。有关更多信息，请参阅 监控 Amazon DocumentDB 集群的状态。

> **Important**
>
> 当您使用创建 Amazon DocumentDB 区域集群时，不会创建任何实例。 AWS CLI 因此，您必须显式创建主实例和所需的任何副本实例。您可以使用控制台或创建 AWS CLI 实例。有关更多信息，请参阅亚马逊 DocumentDB API 参考中的向集群添加 Amazon DocumentDB 实例和 createdbCluster。

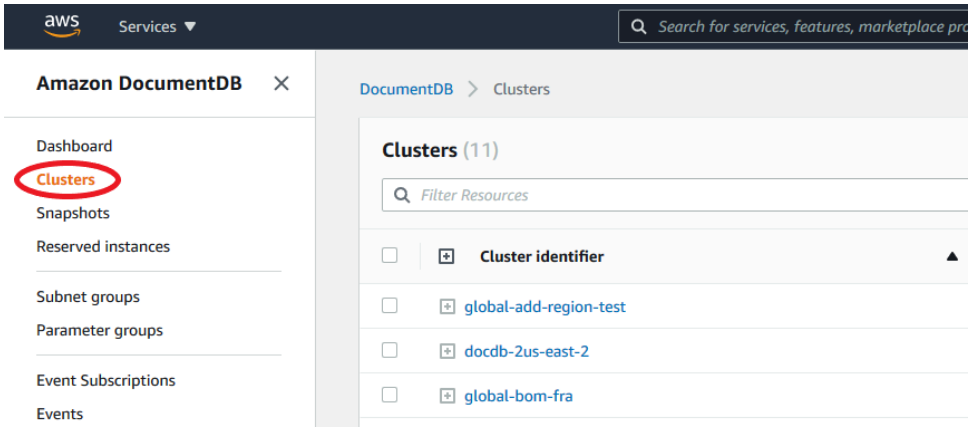
一旦您的区域集群可用，您就可以使用以下说明在另一个区域添加辅助集群：添加一个 AWS 区域 到亚马逊 DocumentDB 全球集群。添加区域后，您的区域集群将成为您的主集群，而您在自己选择的区域中有一个新的辅助集群。

添加一个 AWS 区域 到亚马逊 DocumentDB 全球集群

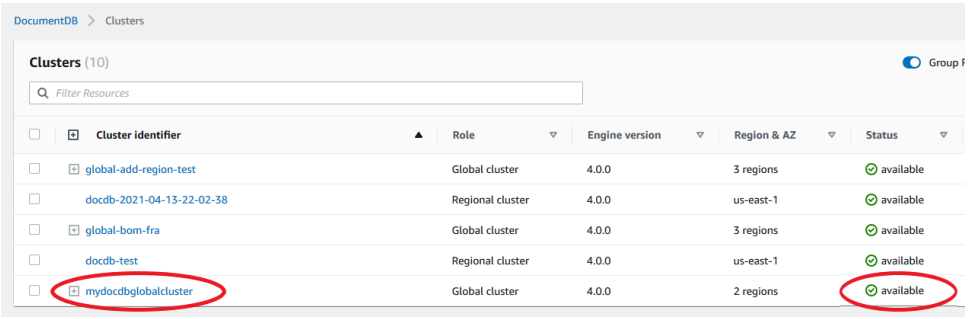
全局群集需要在与主群集不同的区域中至少有一个辅助群集，并且您最多可以添加 10 个辅助群集。注意：对于添加的每个辅助集群，您必须减少主集群的允许副本的数量。例如，如果您的全局群集有 10 个辅助区域，则您的主集群只能有 5 个（而不是 15 个）副本。有关更多信息，请参阅 Amazon DocumentDB 全局集群的配置要求。

使用 AWS 管理控制台

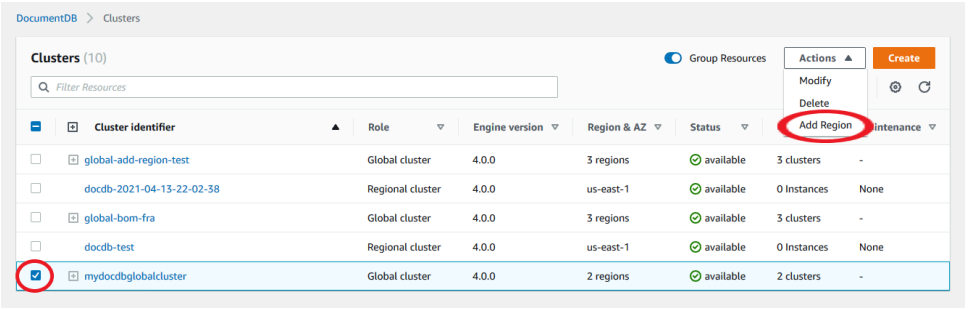
1. 登录 AWS 管理控制台 并打开亚马逊 DocumentDB 控制台。
2. 在导航窗格中，选择集群。



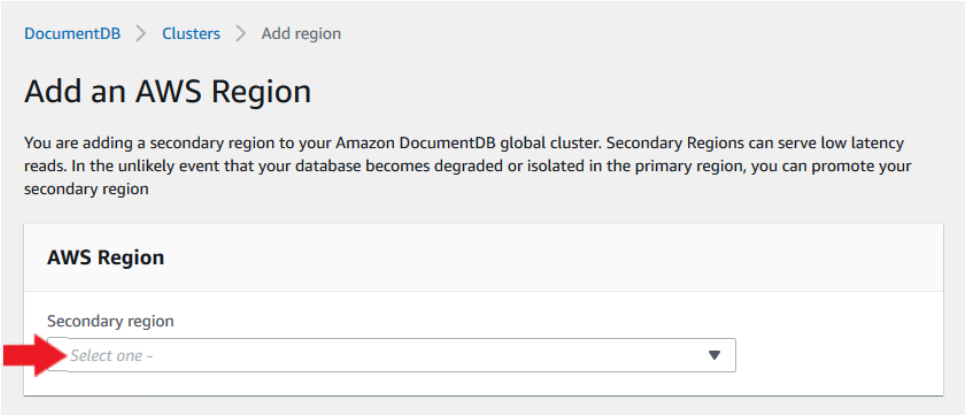
3. 选择您要要向其中添加辅助集群的集群。确保集群为 `Available`。



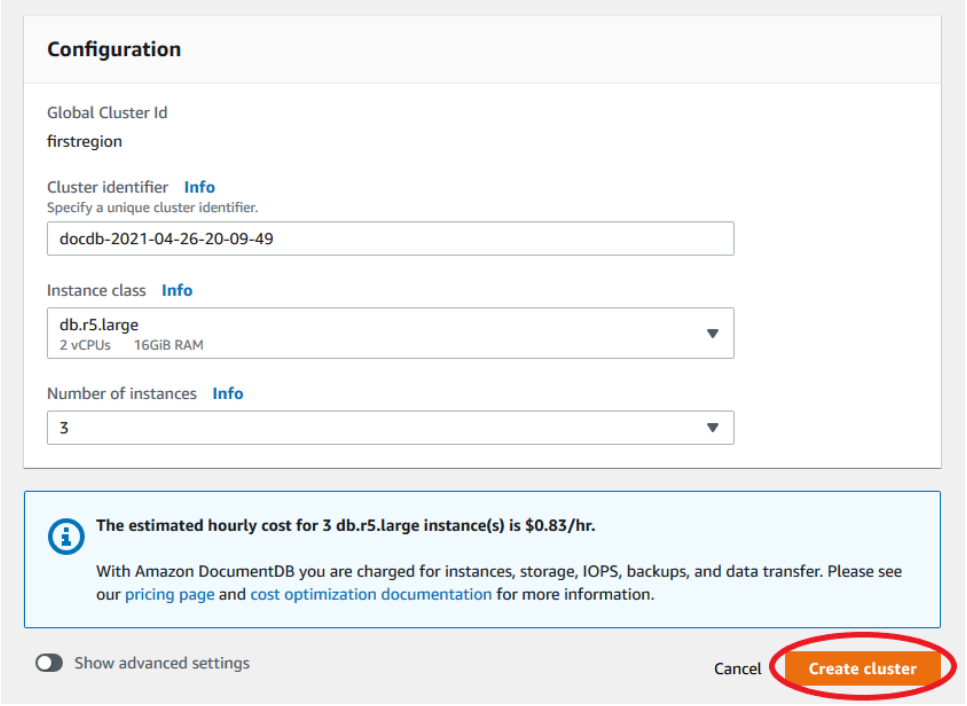
4. 选择操作的下拉菜单列表，然后选择添加区域。



5. 在添加 AWS 区域 页面上，选择辅助区域。您不能为同一个全局集群选择已有辅助集群的区域。此外，该区域也不能是主集群所在的同一个区域。如果这是您正在添加的第一个区域，则您还必须指定您选择的全局集群标识符。



6. 在新区域中填写辅助集群的其余字段，然后选择创建集群。完成添加区域后，您可以在 AWS 管理控制台中的集群列表中看到该区域。



使用 AWS CLI

- 使用带有 (`--global-cluster-identifier`) 全局集群名称的 `create-db-cluster` CLI 命令。对于其他参数，请执行以下操作：
  - 对于`--region`，请选择与您的主要区域不同的 AWS 区域 区域。
  - 为 `--engine` 和 `--engine-version` 参数选择特定的值。
  - 对于加密集群，请将您的主 AWS 区域 集群指定`--source-region`为用于加密的集群。

以下示例创建了一个新 Amazon DocumentDB 数据库集群，并将其作为只读辅助集群附加到全局集群。在最后一步中，实例将添加到新集群。

在以下示例中，将每个 *user input placeholder* 替换为您自己的信息。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

对您的 Amazon DocumentDB 全局集群使用快照

您可以恢复 Amazon DocumentDB 集群的快照以用作您全局集群的起点。要这样做，您必须恢复快照并创建新集群。这个集群将充当您全局群集的主集群。然后，您可以将另一个区域添加到恢复的集群，从而将其转变成全局集群。

管理 Amazon DocumentDB 全局集群

您可对构成全局集群的各个集群执行大多数的管理操作。当您在控制台中 集群页面上选择 对相关资源分组时，您可以看到主集群和辅助集群分组到关联的全局集群之下。

全局群集的 “配置” 选项卡显示集群的运行 AWS 区域 位置、版本和全局群集标识符。

主题

- 修改 Amazon DocumentDB 全局集群
- 修改 Amazon DocumentDB 全局集群的参数
- 从 Amazon DocumentDB 全局集群中删除某集群
- 从 Amazon DocumentDB 全局集群删除集群
- 在辅助区域中创建无管控 Amazon DocumentDB 集群

修改 Amazon DocumentDB 全局集群

中的集群页面 AWS 管理控制台 列出了您的所有全局集群，显示了每个集群的主集群和辅助集群。全局数据库有自己的配置设置。具体来说，它具有与其主集群和辅助集群关联的区域。

当您更改全局集群时，您有机会取消更改。

选择 Continue (继续) 时，即表示您确认更改。

修改 Amazon DocumentDB 全局集群的参数

您可以为全局集群中的每个集群独立配置集群参数组。大多数参数的工作方式与其他类型的 Amazon DocumentDB 集群相同。我们建议您在全局数据库中使所有集群之间的设置保持一致。在将辅助集群提升为主集群时，此操作有助于避免意外的行为变化。

例如，对于时区和字符集使用相同设置，可避免在不同集群作为主集群时出现不一致的行为。

从 Amazon DocumentDB 全局集群中删除某集群

存在您可能需要从自身全局群集中删除群集的几种情况。例如，如果主集群被降级或隔离，您可能希望从全局集群中删除 集群。然后，它将成为独立的预置集群，可用于创建新的全局集群。要了解更多信息，请参阅对 Amazon DocumentDB 全局集群执行手动失效转移。

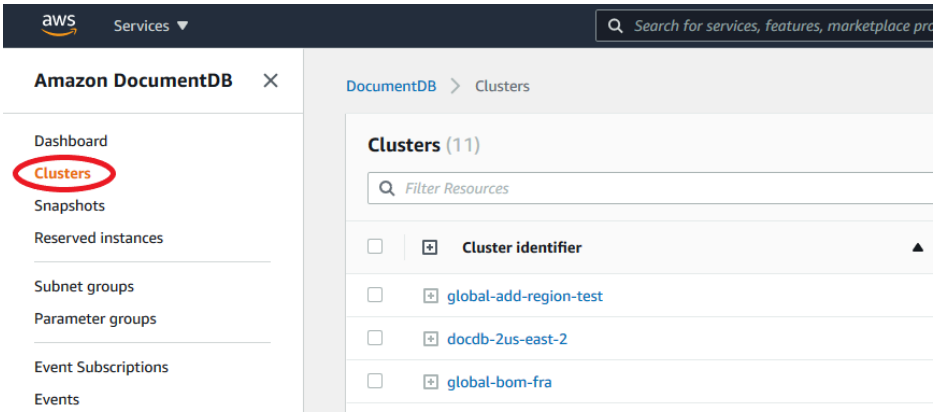
您也可能会想要删除集群，因为您想要删除不再需要的全局集群。在分离所有关联的集群以后，您才能删除全局集群，仅保留主集群。有关更多信息，请参阅 从 Amazon DocumentDB 全局集群删除集群。

> **Note**
>
> 当集群从全局集群中分离时，它将不再与主集群同步。它变成了具有全部 read/write 功能的独立预配置集群。此外，它在 Amazon DocumentDB 控制台中不再可见。只有您在控制台中选择集群位于其中的区域时，它才可见。

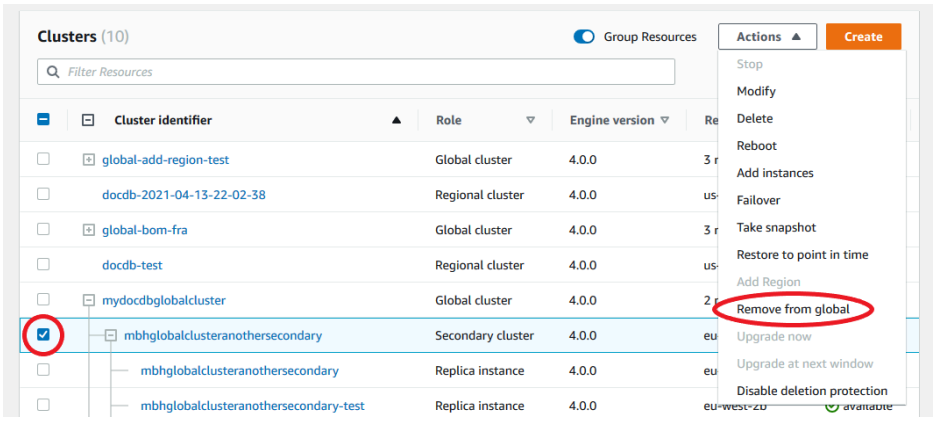
您可以使用 AWS 管理控制台、或 RDS API 从全局集群中 AWS CLI移除集群。

Using the AWS 管理控制台

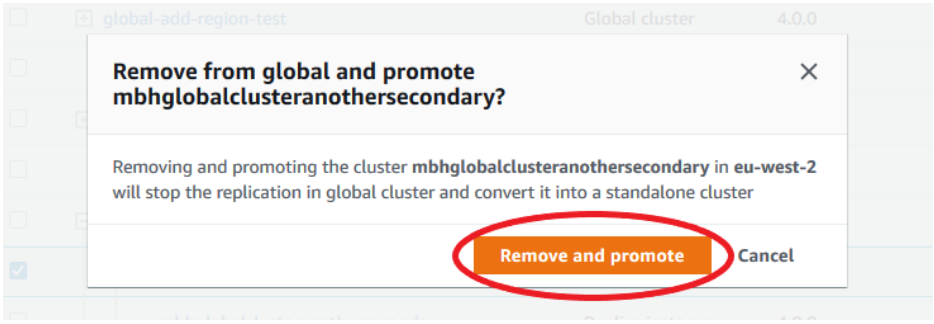
1. 登录 AWS 管理控制台 并导航到亚马逊 DocumentDB 控制台。
2. 请在左侧导航中，选择集群。



3. 展开全局群集，从而您可以看到所有的辅助群集。选择您想删除的辅助集群。选择操作，然后在下拉菜单中，选择从全局集群删除。



4. 将会出现一个提示，要求确认您要将辅助集群从全局集群中分离。选择 删除并提升以从全局集群中删除集群。



现在集群不再作为主集群中的辅助集群，也不再与主集群同步。它是一个 read/write 功能齐全的独立集群。

在移除或删除所有辅助集群后，您可以按同样方式移除主集群。在删除全部辅助集群之前，您无法将主集群从全局集群中分离或删除。全局数据库可能保留在集群列表中，其中有零个区域和可用区。如果不想再使用此全局集群，则可以删除。

Using the AWS CLI

要从全局集群删除 集群，请使用以下参数运行 `remove-from-global-cluster` CLI 命令：

- `--global-cluster-identifier`：全局集群的名称（标识符）。
- `--db-cluster-identifier`：要从全局集群删除的每集群的名称。

以下示例先从全局集群中删除辅助集群，然后删除主集群。

对于 Linux、macOS 或 Unix：

```
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
```

对全局集群中的每个辅助区域重复 `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN` 命令。

对于 Windows：

```
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
```

对全局集群中的每个辅助区域重复 `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN` 命令。

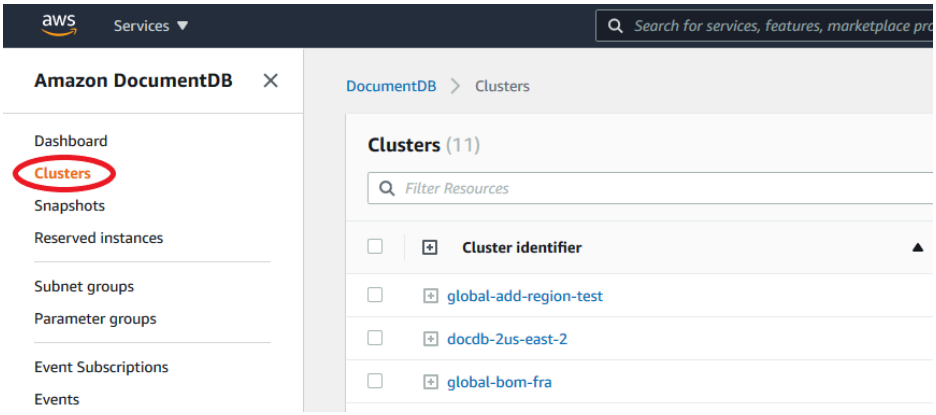
从 Amazon DocumentDB 全局集群删除集群

要删除全局集群，请执行以下操作：

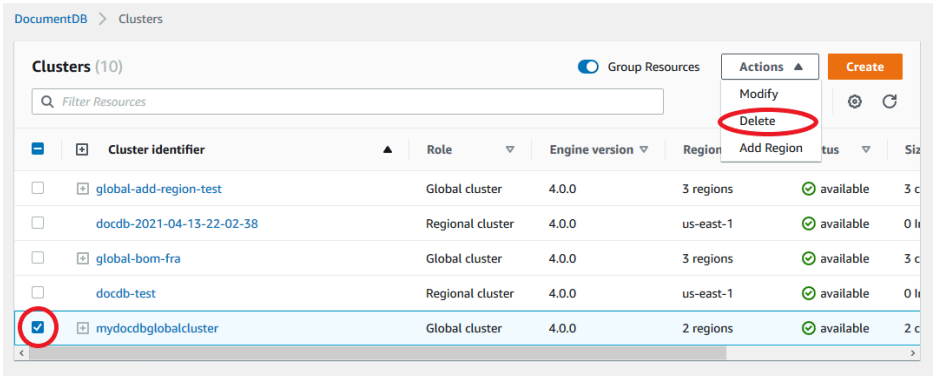
- 从全局集群中删除所有辅助集群。每个集群都会变成独立的集群。请参阅上一部分：从 Amazon DocumentDB 全局集群中删除某集群。
- 从每个独立集群中删除所有副本。
- 从全局集群删除主集群。这将成为独立集群。
- 从主集群中，首先删除所有副本，然后删除主实例。从新近独立的集群中删除主实例通常还会删除集群和全局集群。

Using the AWS 管理控制台

1. 登录 AWS 管理控制台 并导航到亚马逊 DocumentDB 控制台。
2. 选择集群并查找您想要删除的全局集群。



3. 选择您的全局群集，从操作菜单中选择删除。



确认所有集群从全局集群中移除。全局集群应显示 0 个区域和可用区且集群的大小为 0。如果全局集群包含任何集群，则无法将其删除。首先，您必须遵循上一步 从 Amazon DocumentDB 全局集群中删除某集群 中的说明。

Using the AWS CLI

要删除全局群集，请使用名称 AWS 区域 和全局群集标识符运行 `delete-global-cluster` CLI 命令，如以下示例所示。

对于 Linux、macOS 或 Unix：

```
aws docdb --region primary_region delete-global-cluster \
   --global-cluster-identifier global_cluster_id
```

对于 Windows：

```
aws docdb --region primary_region delete-global-cluster ^
   --global-cluster-identifier global_cluster_id
```

在辅助区域中创建无管控 Amazon DocumentDB 集群

尽管 Amazon DocumentDB 全局集群需要至少一个与主集群 AWS 区域 不同的辅助集群，但您可以为辅助群集使用无头配置。无管控辅助 Amazon DocumentDB 集群是没有实例的集群。此类型的配置可以降低全局集群的开支。在 Amazon DocumentDB 集群中，计算和存储是分离的。如果没有实例，您就无需为计算付费，而只需为存储付费。如果设置正确，无管控辅助存储卷将与主集群保持同步。

您可以像平常创建 Amazon DocumentDB 全局数集群一样添加辅助集群。但是，在主集群开始复制到辅助集群之后，您将从辅助集群中删除该只读实例。此辅助集群现在被视为“无管控”集群，因为其不再有实例。但是，存储卷与主 Amazon DocumentDB 集群保持同步。

> **Important**
>
> 我们只向能够容忍区域级失效超过 15 分钟的客户推荐无头集群。这是因为借助无头辅助群集从区域级失效中恢复将要求用户在失效转移后创建一个新实例。新实例可能耗时大约 10-15 分钟变得可用。

将无管控辅助集群添加到您的全局集群

1. 登录 AWS 管理控制台 并打开亚马逊 DocumentDB 控制台。
2. 请在左侧导航中，选择集群。
3. 选择需要辅助集群的全局集群。确保主数据库集群为 `Available`。
4. 对于 Actions (操作)，选择 Add region (添加区域)。
5. 在添加区域页面上，选择辅助区域。

   > **Note**
   >
   > 您不能为同一个全局集群选择已有辅助集群的区域。此外，该区域也不能是主集群所在的同一个区域。

6. 在新区域中填写辅助集群的其余字段。这些配置选项与任何集群实例的配置选项相同。
7. 添加区域。完成向自身全局集群添加该区域后，您将在 AWS 管理控制台中 `Clusters` 的列表中看到它。

8. 在继续操作之前，使用 AWS 管理控制台 或检查辅助群集及其读取器实例的状态 AWS CLI。如果您使用 AWS CLI，则这里是一个示例命令：

```
$ aws docdb describe-db-clusters --db-cluster-identifier secondary-cluster-id --
query '*[].[Status]' --output text
```

新添加的辅助集群的状态可能需要几分钟的时间才能从“正在创建”更改为“可用”。当集群处于可用状态时，您可以删除读取器实例。

9. 在辅助集群中选择读取器实例，然后选择删除。
10. 删除读取器实例后，辅助集群仍然是全局集群的组成部分。它应该没有与之关联的实例。

> **Note**
>
> 如果主区域发生计划外停机事件，您可以使用此无管控辅助 Amazon DocumentDB 集群手动恢复 Amazon DocumentDB 。

连接到 Amazon DocumentDB 全局集群

连接到全局集群的方式取决于您是需要写入集群还是从集群读取：

- 对于只读请求或查询，您连接到 AWS 区域中集群的读取器端点。
- 要运行数据操作语言 (DML) 或数据定义语言 (DDL) 语句，应连接到主集群的集群端点。此端点可能与您的应用程序位于 AWS 区域 不同的位置。

当您在控制台中查看全局集群时，您可以看到与其所有集群关联的所有通用端点。

连接到全局集群的方式取决于您是需要写入数据库还是从数据库读取。对于要从主区域提供的 DDL、DML 和读取操作，您应该连接到主集群。我们建议您使用集群端点以副本集模式连接到您的主集群，将写入偏好设置为 `secondaryPreferred=true`。这会将写入流量路由到主集群的写入器实例，并将读取流量路由到主集群的副本实例。

对于跨区域、只读流量，您应该连接到一个辅助集群。我们建议您使用集群端点以副本集模式连接到您的辅助集群。由于所有实例都是只读副本实例，因此您应指定除之外的读取首选项`primary`（例如`secondarysecondaryPreferred`、或`nearest`）。为了最大限度地减少延迟，请选择您所在区域或离您最近的区域中的任何读取器端点。

监控 Amazon DocumentDB 全局集群

Amazon DocumentDB（兼容 MongoDB）与集成， CloudWatch 因此您可以收集和分析集群的运行指标。您可以使用 CloudWatch控制台、Amazon DocumentDB 控制台、 AWS Command Line Interface (AWS CLI) 或 API 来监控这些指标。 CloudWatch

要监控全局集群，请使用以下 CloudWatch 指标。

指标	说明
`GlobalClusterReplicatedWriteIO`	按照 5 分钟报告从主 AWS 区域 卷中的群集卷复制到辅助 AWS 区域卷的集群卷的计费写入 I/O 操作的平均数量。复制到每个辅助区域的 `ReplicatedWriteIOs` 数量与主区域执行的区域内 `VolumeWriteIOPs` 数量相同。
`GlobalClusterDataTransferBytes`	从主群集传输到辅助群集的数据量 AWS 区域，AWS 区域 以字节为单位。
`GlobalClusterReplicationLag`	将更改事件从主群集复制到辅助群集时的延迟量（ AWS 区域 以毫秒为单位） AWS 区域

有关如何查看这些指标的更多信息，请参阅查看 CloudWatch 数据。

灾难恢复和 Amazon DocumentDB 全局集群

主题

- 对 Amazon DocumentDB 全局集群执行托管式失效转移
- 对 Amazon DocumentDB 全局集群执行手动失效转移
- 对 Amazon DocumentDB 全局集群执行切换
- 解除全局集群切换或失效转移的阻止

通过使用全局集群，您可以快速从区域故障等灾难中恢复。灾难恢复通常以 RTO 和 RPO 的值来衡量。

- 恢复时间目标 (RTO)：灾难后系统恢复工作状态所需的时间。换言之，RTO 用于衡量停机时间。对于全局集群，RTO 在几分钟内。

- 恢复点目标 (RPO)：可能丢失的数据量（按时间衡量）。对于全局集群，RPO 通常以秒为单位进行测量。
- 要从计划外停机中恢复，您可以执行跨区域故障转移到全局集群中的一个辅助区域。当全局集群有多个辅助区域时，请确保分离您想要提升为主区域的所有辅助区域。然后，您将其中一个辅助区域提升为新的主 AWS 区域。最后，在其他每个辅助区域中创建新的集群，并将这些集群附加到全局集群。

对 Amazon DocumentDB 全局集群执行托管式失效转移

这种方法用于在发生真实的区域性灾难或完全的服务级别中断时实现业务连续性。

在托管式失效转移期间，主集群会失效转移到您选择的辅助区域，同时维护 Amazon DocumentDB 全局集群的现有复制拓扑。所选的辅助集群将其一个只读节点提升为完全写入器状态。此步骤允许集群代入主集群的角色。在此集群代入其新角色期间，您的数据库在短时间内不可用。当该辅助集群成为新的主集群时，未从旧的主集群复制到选定辅助集群的数据可能会丢失。在与新主卷同步之前，旧主卷会尽力尝试拍一张快照，以便在快照中保存未被复制的数据。

> **Note**
>
> 仅当主集群和所有辅助集群具有相同的引擎版本时，您才能对 Amazon DocumentDB 全局集群执行托管式跨区域集群失效转移。如果您的引擎版本不兼容，则可以按照对 Amazon DocumentDB 全局集群执行手动失效转移中的步骤手动执行失效转移。
> 如果区域的引擎版本不匹配，则失效转移将被阻止。检查是否有任何待处理的升级并应用这些升级，以确保所有区域的引擎版本都匹配并且全局群集故障转移畅通无阻。有关更多信息，请参阅 解除全局集群切换或失效转移的阻止。

为最大限度地减少数据丢失，我们建议您在使用此特征之前执行以下操作：

- 使应用程序离线以防止写入内容被发送到 Amazon DocumentDB 全局集群的主集群。
- 查看所有 Amazon DocumentDB 辅助集群的滞后时间。选择复制滞后最小的辅助区域可以最大限度地减少当前出现故障的主区域的数据丢失。通过查看亚马逊中的`GlobalClusterReplicationLag`指标，查看全局集群中所有 Amazon DocumentDB 辅助集群的延迟时间。 CloudWatch这些指标显示复制到辅助集群滞后于复制到主集群的时间（以毫秒为单位）。

  有关亚马逊 DocumentDB CloudWatch 指标的更多信息，请参阅。Amazon DocumentDB 指标

在托管式失效转移期间，所选的辅助集群将提升为新角色，即主集群。但是，它不会继承主集群的各种配置选项。配置不匹配可能会导致性能问题、工作负载不兼容和其他异常行为。为避免出现此类问题，我们建议您解决以下方面的 Amazon DocumentDB 全局集群之间的差异问题：

- 为新的主数据库集群配置 Amazon DocumentDB 集群参数组（如有必要）- 您可以为 Amazon DocumentDB 全局集群中的每个集群单独配置 Amazon DocumentDB 集群参数组。因此，当您提升辅助集群以接管主数据库集群的角色时，辅助数据库集群中参数组的配置可能与主数据库集群的配置不同。如果是这样，请修改提升后的辅助集群的参数组，使其与主集群的设置一致。要了解如何操作，请参阅修改 Amazon DocumentDB 集群参数组。
- 配置监控工具和选项，例如 Amazon CloudWatch 事件和警报 — 根据全局集群的需要，为提升的集群配置相同的日志记录功能、警报等。与参数组一样，在故障转移过程中，这些特征的配置不会从主数据库集群继承。某些 CloudWatch 指标（例如复制延迟）仅适用于次要区域。因此，失效转移会更改查看这些指标和对指标设置警报的方式，并且可能要求更改任何预定义的控制面板。有关 Amazon DocumentDB 集群和监控的更多信息，请参阅 监控 Amazon DocumentDB。

通常，所选的辅助集群会在一分钟内代入主角色。一旦新的主区域的写入器节点可用，您就可以将应用程序连接到该节点并恢复工作负载。Amazon DocumentDB 提升新的主集群后，它会自动重建所有其他辅助区域集群。

由于 Amazon DocumentDB 全局集群使用异步复制，因此每个辅助区域的复制滞后可能会有所不同。Amazon DocumentDB 重建这些辅助区域，使其具有与新的主区域集群完全相同的时间点数据。完成重建任务的持续时间可能需要几分钟到几小时，具体取决于存储卷的大小和区域之间的距离。当辅助区域集群从新的主区域完成重建后，它们就可供进行读取访问了。一旦新的主写入器已提升并可用，新的主区域的集群就可以处理 Amazon DocumentDB 全局集群的读取和写入操作。

为了还原全局集群的原始拓扑，Amazon DocumentDB 会监控旧主区域的可用性。一旦该区域正常运行并再次可用，Amazon DocumentDB 就会自动将其作为辅助区域重新添加到全局集群中。在旧的主区域中创建新的存储卷之前，Amazon DocumentDB 会尝试在出现故障时拍摄旧存储卷的快照。它这样做是为了让您可以用它来恢复任何丢失的数据。如果此操作成功，Amazon DocumentDB 将名为 “rds: docdb-unplanned-global-failover-name-old-primary-” 的快照放在快照部分中。DB-cluster-timestamp AWS 管理控制台还可以在 `DescribeDBClusterSnapshots` API 操作返回的信息中看到列出的此快照。

> **Note**
>
> 旧存储卷的快照是系统快照，受旧的主集群上配置的备份保留期限制。要在保留期之外保留此快照，可以复制它以另存为手动快照。要了解有关复制快照的更多信息（包括定价），请参阅复制集群快照。

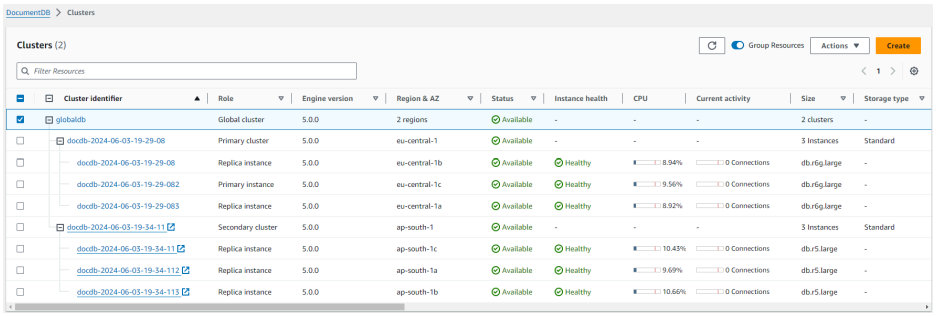
还原原始拓扑后，您可以通过在对业务和工作负载最有意义的时候执行切换操作，将全局集群失效自动恢复到原始主区域。为此，请按照对 Amazon DocumentDB 全局集群执行切换中的步骤进行操作。

您可以使用 AWS 管理控制台、或 Amazon DocumentDB API 对亚马逊文档数据库全局集群进行故障切换。 AWS CLI

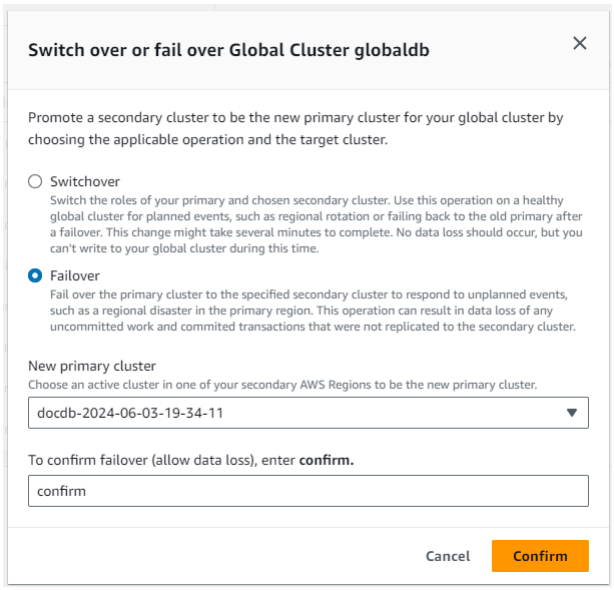
Using the AWS 管理控制台

对 Amazon DocumentDB 全局集群执行托管式失效转移

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。
3. 查找并选择要进行失效转移的 Amazon DocumentDB 全局集群。



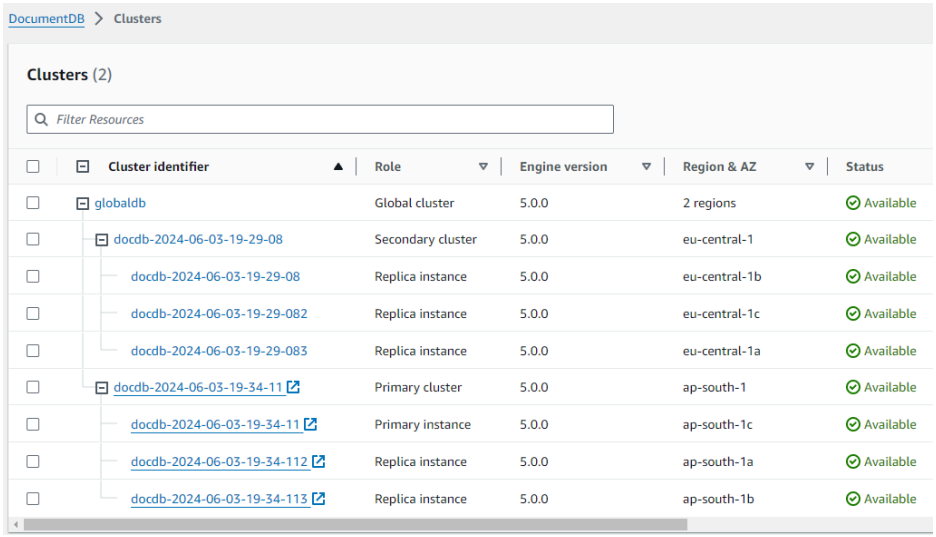
4. 从操作菜单中选择切换或失效转移。
5. 在弹出的对话框中，选择失效转移，然后从新主集群字段下拉列表中选择辅助集群。



6. 在最后一个字段中，键入“confirm”。然后，选择 Confirm（确认）。

   主群集的状态更改为 “Failing-over”。这种情况应该需要一分钟左右。在此期间，新主集群的状态显示“正在修改...”。新主集群升级后将显示“可用”，并且能够处理读取和写入事务。与新主集群重新同步时，包含旧主集群的辅助区域将显示“正在重新同步...”。与新主集群类似，只有在状态变为“可用”后，旧主集群才能处理事务。

7. 完成后，原始主集群将变为辅助集群。选定的辅助集群将变为主集群。



Using the AWS CLI

对 Amazon DocumentDB 全局集群执行托管式失效转移

运行 failover-global-cluster CLI 命令以对 Amazon DocumentDB 全局集群进行失效转移。使用命令，传递下列选项的值：

- `--region`
- `--global-cluster-identifier`
- `--target-db-cluster-identifier`
- `--allow-data-loss`

在以下示例中，将每个*user input placeholder*示例替换为集群的信息。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

对 Amazon DocumentDB 全局集群执行手动失效转移

如果一个集群中的整个集群 AWS 区域 不可用，则可以将全局群集中的另一个集群提升为具有 read/write 功能。

如果另一个 AWS 区域 中的集群成为主集群效果更好，您可以手动启用全局集群失效转移机制。例如，您可以提高其中一个辅助集群的容量，然后将其提升成为主集群。或者，它们之间的活动平衡 AWS 区域 可能会发生变化，因此将主集群切换到不同的集群 AWS 区域 可能会降低写入操作的延迟。

以下过程概述了如何在 Amazon DocumentDB 全局集群中提升其中一个辅助集群。

提升辅助集群

1. 停机时停止向主集群发出 DML 语句和其他写入操作。 AWS 区域
2. 从辅助群集中识别出 AWS 区域 要用作新的主群集的群集。如果您的全局群集中有两个（或更多）辅助 AWS 区域 群集，请选择延迟时间最少的辅助群集。
3. 从全局集群分离您所选的辅助集群。

   从全局群集中移除辅助群集会立即停止从主群集向该辅助群集的复制，并将其提升为具有全部 read/write 功能的独立预配置群集。与该停机区域中的主集群关联的任何其他辅助集群仍然可用，并且可以接受应用程序的调用。它们还会消耗资源。由于您要重新创建全局集群，为避免分裂大脑和其他问题，请先删除其他辅助集群，再在后续步骤中创建新的全局集群。

   有关分离的详细步骤，请参阅 从 Amazon DocumentDB 全局集群中删除某集群。
4. 在下一步中，当您开始向集群添加区域时，该集群将成为新的全局数据库的主集群。
5. AWS 区域 向集群中添加。执行此操作后，从主数据库集群到辅助数据库集群的复制过程将会开始。
6. 根据需要添加更多内容 AWS 区域 ，以重新创建支持您的应用程序所需的拓扑。确保在做出这些更改之前、更改期间和更改之后，将应用程序写入内容发送到正确的集群，以避免全局集群中集群之间的数据不一致（大脑分裂问题）。
7. 当停机问题已解决且您已准备好再次将原始 AWS 区域 指定为主集群时，请按相反顺序执行相同步骤：
8. 从全局集群移除其中一个辅助集群。这将使其能够为 read/write 流量提供服务。
9. 将所有写入流量重定向到原始 AWS 区域中的主集群。
10. 添加 AWS 区域 可以 AWS 区域 像以前一样设置一个或多个辅助群集。

Amazon Document AWS DB 全球集群可以使用软件开发工具包进行管理，这使您能够针对灾难恢复和业务连续性规划用例创建解决方案来自动执行全局集群故障转移流程。其中一个解决方案已通过 Apache 2.0 许可提供给我们的客户，并且可以从我们的工具库中访问此处。该解决方案利用 Amazon Route 53 进行终端节点管理，并提供可根据相应事件触发的 AWS Lambda 功能。

对 Amazon DocumentDB 全局集群执行切换

通过使用切换，您可以定期更改主集群的区域。此方法适用于受控场景，例如操作维护和其他计划内操作过程。

切换有三种常见使用案例：

- 适用于对特定行业施加的“区域轮换”要求。例如，金融服务法规可能要求第 0 层系统在几个月内切换到不同的区域，以确保定期执行灾难恢复过程。

- 适用于多区域“全天候”应用程序。例如，一家企业可能希望根据不同时区的工作时间在不同区域提供延迟更低的写入。
- 作为一种零数据丢失方法，可在失效转移后失效自动恢复到原始主区域。

> **Note**
>
> 切换功能专为用于正常运行的 Amazon DocumentDB 全局集群而设计。要从计划外停机中进行恢复，请按照对 Amazon DocumentDB 全局集群执行手动失效转移中的相应过程操作。
> 要执行切换，所有辅助区域都必须运行与主区域完全相同的引擎版本。如果区域的引擎版本不匹配，则切换将被阻止。检查是否有任何待处理的升级并应用这些升级，以确保所有区域的引擎版本都匹配并且全局集群切换畅通无阻。有关更多信息，请参阅 解除全局集群切换或失效转移的阻止。

在切换过程中，Amazon DocumentDB 会将您的主集群切换到您选择的辅助区域，同时维护全局集群的现有复制拓扑。在开始切换过程之前，Amazon DocumentDB 会等待所有辅助区域集群与主区域集群完全同步。然后，主区域中的数据库集群将变为只读状态，所选辅助集群将其一个只读节点提升为完全写入器状态。将此节点提升为写入器将允许该辅助集群代入主集群的角色。由于所有辅助集群在过程开始时都与主集群同步，因此新的主集群将继续执行 Amazon DocumentDB 全局集群的操作，而不会丢失任何数据。您的数据库在短时间内不可用，而主集群和所选的辅助集群将担任其新角色。

为了优化应用程序可用性，我们建议您在使用此特征之前执行以下操作：

- 在非高峰时间段，或在向主集群写入操作最少的其他时间执行此操作。
- 使应用程序离线以防止写入内容被发送到 Amazon DocumentDB 全局集群的主集群。
- 通过查看亚马逊中的`GlobalClusterReplicationLag`指标，查看全局集群中所有 Amazon DocumentDB 辅助集群的延迟时间。 CloudWatch该指标显示复制到辅助集群滞后于复制到主集群的时间（以毫秒为单位）。该值与 Amazon DocumentDB 完成切换所需的时间成正比。因此，滞后值越大，切换所需的时间就越长。

  有关亚马逊 DocumentDB CloudWatch 指标的更多信息，请参阅。Amazon DocumentDB 指标

在切换期间，所选的辅助数据库集群将提升为新角色，即主数据库集群。但是，它不会继承主数据库集群的各种配置选项。配置不匹配可能会导致性能问题、工作负载不兼容和其他异常行为。为避免出现此类问题，我们建议您解决以下方面的 Amazon DocumentDB 全局集群之间的差异问题：

- 为新的主数据库集群配置 Amazon DocumentDB 数据库集群参数组（如有必要）- 您可以为 Amazon DocumentDB 全局集群中的每个集群单独配置 Amazon DocumentDB 集群参数组。这意味着，当您提升辅助数据库集群以接管主数据库集群的角色时，辅助数据库集群中参数组的配置可能与主数据库集群的配置不同。如果是这样，请修改提升后的辅助数据库集群的参数组，使其与主集群的设置一致。要了解如何操作，请参阅管理 Amazon DocumentDB 集群参数组。
- 配置监控工具和选项，例如 Amazon Ev CloudWatch ents 和警报 — 根据全局集群的需要，为提升的集群配置相同的日志记录功能、警报等。与参数组一样，在切换过程中，这些特征的配置不会从主数据库集群继承。某些 CloudWatch 指标（例如复制延迟）仅适用于主要区域。因此，切换会更改查看这些指标和对指标设置警报的方式，并且可能要求更改任何预定义的控制面板。有关更多信息，请参阅 监控 Amazon DocumentDB。

> **Note**
>
> 通常，角色切换最多可能需要几分钟。

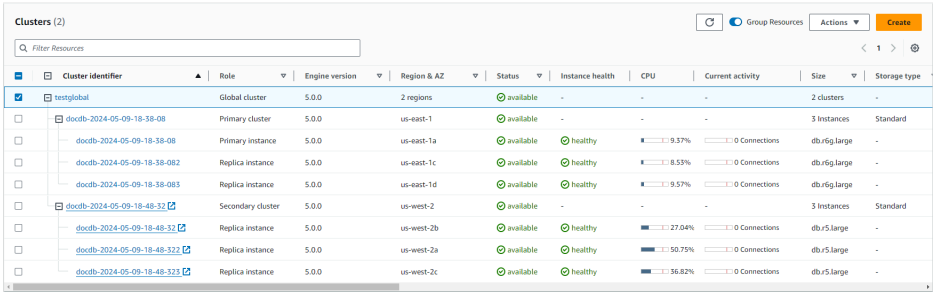
切换过程完成后，提升后的 Amazon DocumentDB 集群即可处理全局集群的写入操作。

您可以使用 AWS 管理控制台 或切换您的 Amazon DocumentDB 全局集群： AWS CLI

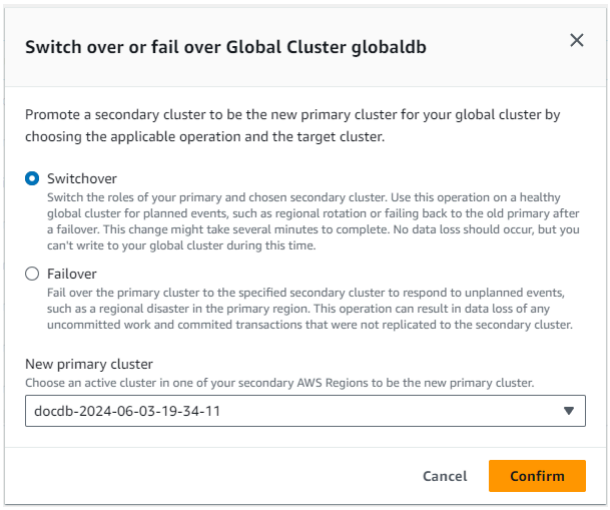
Using the AWS 管理控制台

对 Amazon DocumentDB 全局集群执行切换

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。
3. 查找并选择要进行切换的 Amazon DocumentDB 全局集群。



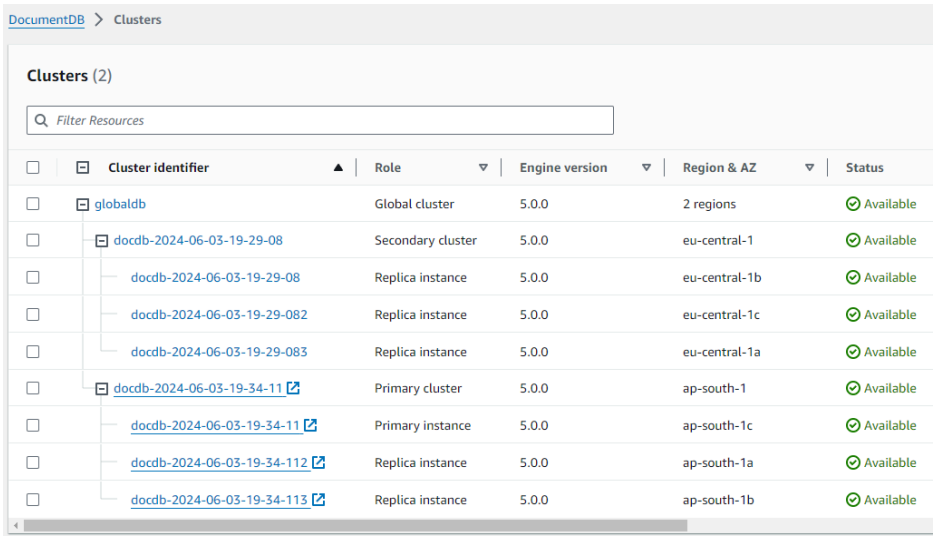
4. 从操作菜单中选择切换或失效转移。
5. 在弹出的对话框中，选择切换，然后从新主集群字段下拉列表中选择辅助集群。



6. 选择确认。

   主群集的状态更改为 “Switching-over”。这种情况应该需要三分钟左右。在此期间，所有区域集群的状态显示“正在修改...”。一旦区域同步且新主集群升级后，所有状态字段将显示“可用”，并且能够处理事务。

7. 完成后，原始主集群将变为辅助集群。选定的辅助集群将变为主集群。



Using the AWS CLI

对 Amazon DocumentDB 全局集群执行切换

运行 `switchover-global-cluster` CLI 命令以切换 Amazon DocumentDB 全局集群。使用命令，传递下列选项的值：

- `--region`

- `--global-cluster-identifier`
- `--target-db-cluster-identifier`

在以下示例中，将每个*user input placeholder*示例替换为集群的信息。

对于 Linux、macOS 或 Unix：

```
aws docdb switchover-global-cluster \
   --region region_of_primary \
   --global-cluster-identifier global_cluster_id \
   --target-db-cluster-identifier arn_of_secondary_to_promote
```

对于 Windows：

```
aws docdb switchover-global-cluster ^
   --region region_of_primary ^
   --global-cluster-identifier global_cluster_id ^
   --target-db-cluster-identifier arn_of_secondary_to_promote
```

解除全局集群切换或失效转移的阻止

当全局集群中的所有区域集群并非都使用相同的引擎版本时，会阻止全局集群切换和失效转移。如果版本不匹配，则在调用切换或失效转移时，您可能会在响应中收到以下错误：指定的目标数据库集群正在运行的引擎版本与源数据库集群的补丁级别不同。我们建议定期应用最新的引擎版本，以确保您运行的是最新的更新，从而使您的全局集群保持健康状态。

要解决此错误，请先应用所有待处理的维护措施项将所有次要区域更新为相同的引擎版本，然后将主区域更新为相同的引擎版本。要查看待处理维护操作项目并应用任何必要的变更来更正问题，请按照以下选项卡之一中的说明进行操作：

Using the AWS 管理控制台

要解除对全局集群切换或失效转移的阻止，您必须确定集群是否有任何待处理维护操作并应用这些操作。请按照以下步骤查看并应用维护操作：

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。

3. 在集群表中，在集群标识符列中找到您的全局集群。在您的全局集群下，记下给定全局集群的每个辅助集群和主集群，然后对每个集群执行以下步骤。
4. 对于每个辅助集群：
   a. 如果更新可用于您的集群，则会在维护列中以可用、必需或下一个时段指示。
   b. 要采取操作，请选择集群以显示其详细信息，然后选择维护和备份。将显示待处理维护项目。
   c. 在描述下，如果指示“有新的维护更新可用”，请将其选中，然后选择立即应用。
5. 对于您的主集群：
   a. 如果更新可用于您的集群，则会在维护列中以可用、必需或下一个时段指示。
   b. 要采取操作，请选择集群以显示其详细信息，然后选择维护和备份。将显示待处理维护项目。
   c. 在描述下，如果指示“有新的维护更新可用”，请将其选中，然后选择立即应用。

Using the AWS CLI

要解除对全局集群切换或失效转移的阻止，您必须确定集群是否有任何待处理维护操作并应用这些操作。请按照以下步骤，先在全局集群的辅助集群上查看并应用维护操作，然后再在全局集群的主集群上查看并应用维护操作：

1. 先在每个辅助区域的区域集群上运行以下命令，然后再在主区域的区域集群上运行以下命令。
2. 运行具有 `--resource-identifier` 选项的 `describe-pending-maintenance-actions` CLI 命令，以确定是否有任何维护操作可用于您的 Amazon DocumentDB 区域集群。

   在以下示例中，将每个*user input placeholder*示例替换为集群的信息。

   对于 Linux、macOS 或 Unix：

   ```
   aws docdb describe-pending-maintenance-action \
      --resource-identifier arn:aws:rds:us-
   east-1:001234567890:cluster:docdb-2025-03-27-19-21-15
   ```

   对于 Windows：

   ```
   aws docdb describe-pending-maintenance-action ^
   ```

```
   --resource-identifier arn:aws:rds:us-
east-1:001234567890:cluster:docdb-2025-03-27-19-21-15
```

结果类似如下：

```
{
    "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. 如果需要维护操作，请运行具有以下选项的 apply-pending-maintenance-action CLI 命令：

- --resource-identifier
- --apply-action
- --opt-in-type
- --region

在以下示例中，将每个*user input placeholder*示例替换为集群的信息。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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. 维护操作完成后，再次运行 describe-pending-maintenance-actions 命令以确保您的集群没有其他待处理的操作。

   您想要的结果是：

```
{
    "PendingMaintenanceActions": []
}
```

Using the Amazon DocumentDB API

要解除对全局集群切换或失效转移的阻止，您必须确定集群是否有任何待处理维护操作并应用这些操作。使用以下 API 查看并应用维护操作：

1. 先在每个辅助区域的区域集群上运行以下命令，然后再在主区域的区域集群上运行以下命令。
2. 调用 PendingMaintenanceAction API 以确定是否有任何维护操作可用于您的 Amazon DocumentDB 全局集群。
3. 通过调用 ApplyPendingMaintenanceAction API 来应用任何更改。

管理 Amazon DocumentDB 集群

要管理 Amazon DocumentDB 集群，您必须拥有带适当 Amazon DocumentDB 控制面板权限的 IAM policy。这些权限使您能够创建、修改和删除集群和实例。AmazonDocDBFullAccess 策略提供管理 Amazon DocumentDB 集群所需的所有权限。

以下主题说明如何在使用 Amazon DocumentDB 集群时执行各种任务，包括创建、删除、修改、连接和查看集群。

主题

- 了解集群
- Amazon DocumentDB 集群设置
- Amazon DocumentDB 集群存储配置
- 确定集群的状态
- Amazon DocumentDB 集群生命周期
- 扩展 Amazon DocumentDB 集群
- 克隆 Amazon DocumentDB 集群卷
- 了解 Amazon DocumentDB 集群容错能力

了解集群

Amazon DocumentDB 分离计算和存储，并将数据复制和备份负载分流到集群卷。集群卷提供持久、可靠且高度可用的存储层，可跨三个可用区以六种方式复制数据。副本可实现更高的数据可用性和读取扩展。每个集群可以扩展到最多 15 个副本。

名词	说明	API 操作（动词）

Cluster	包含一个或多个实例和一个管理这些实例的数据的集群存储卷。	`create-db-cluster`
		`delete-db-cluster`
		`describe-db-clusters`
		`modify-db-cluster`

实例	读取集群存储卷以及向其中写入数据将通过实例进行。在给定的集群中，有两种类型的实例：主实例和副本实例。一个集群始终有一个主实例，并且可以有 0–15 个副本实例。	`create-db-instance`
		`delete-db-instance`
		`describe-db-instances`
		`modify-db-instance`
		`describe-orderable-db-instance-options`

名词	说明	API 操作（动词）
		`reboot-db-instance`
集群卷	跨三个可用区的虚拟数据库存储卷，每个可用区均有两个集群数据副本。	N/A
主实例	支持读取和写入操作，并执行针对集群卷的所有数据修改。每个集群均有一个主实例。	N/A
副本实例	仅支持读取操作。除主实例之外，每个 Amazon DocumentDB 集群最多可拥有 15 个副本实例。多个副本共同分担读取工作负载。通过将副本置于单独的可用区中，您还可以提高数据库可用性。	N/A
集群端点	Amazon DocumentDB 集群的一个端点，用于连接到该集群的当前主实例。每个 Amazon DocumentDB 集群都具有一个集群端点和一个主实例。	N/A
读取器端点	Amazon DocumentDB 集群的一个端点，用于连接到该集群的可用副本之一。每个 Amazon DocumentDB 集群都具有一个读取器端点。如果有多个副本，则读取器端点会将每个连接请求定向到 Amazon DocumentDB 副本之一。	N/A

名词	说明	API 操作（动词）
实例端点	Amazon DocumentDB 集群中的实例的一个端点，用于连接到该特定实例。集群中的每个实例，不论其实例类型，都有各自唯一的实例端点。	N/A

Amazon DocumentDB 集群设置

创建或修改集群时，了解哪些参数不可改变，哪些参数在创建集群后可修改，这一点很重要。下表列出了特定于集群的所有设置或参数。如表中所示，有些是可修改的，有些不可修改。

> **Note**
>
> 这些设置不应与 Amazon DocumentDB 集群参数组及其参数混淆。有关集群参数组的更多信息，请参阅 管理 Amazon DocumentDB 集群参数组。

参数	可修改	备注
DBClusterIdentifier	是	命名约束： • 长度为 [1-63] 个字母、数字或连字符。 • 第一个字符必须是字母。 • 不能以连字符结尾或包含两个连续的连字符。 • 每个区域的 Amazon RDS、Amazon Neptune 和 Amazon DocumentDB 中的所有集群都必须是唯一 AWS 账户的。
Engine	否	必须是 docdb。
BackupRetentionPeriod	是	必须介于 [1-35] 天之间。
DBClusterParameterGroupName	是	命名约束：

参数	可修改	备注
		• 长度为 [1-255] 个字母数字字符。 • 第一个字符必须是字母。 • 不能以连字符结尾或包含两个连续的连字符。
DBSubnetGroupName	否	创建集群后，无法修改集群的子网。
EngineVersion	否	值可以是 `5.0.0`（默认）、`4.0.0` 或 `3.6.0`。
KmsKeyId	否	如果您选择加密集群，则无法更改用于加密集群的密 AWS KMS 钥。
MasterUsername	否	创建集群后，无法修改 `MasterUsername` 。 命名约束： • 长度为 [1-63] 个字母数字字符。 • 第一个字符必须是字母。 • 不能是数据库引擎的保留字。

MasterUserPassword	是	约束：
		• 长度为 [8–100] 个可打印 ASCII 字符。
		• 可以使用任何可打印 ASCII 字符，以下字符除外：
		  • **/**（正斜杠）
		  • **"**（双引号）
		  • **@**（@ 符号）

Port	是	端口号适用于集群中的所有实例。
PreferredBackupWindow	支持	
PreferredMaintenanceWindow	是	

参数	可修改	备注
StorageEncrypted	否	如果选择加密集群，则不能对其进行加密。
StorageType	是	数据库集群的存储类型：标准 (standard) 或 I/O-Optimized (iopt1)。 默认值：standard 可以使用 CreateDBCluster 和 ModifyDBCluster 配置此参数。 有关更多信息，请参阅 Amazon DocumentDB 集群存储配置。
Tags	是	
VpcSecurityGroupIds	否	创建集群后，您不能修改集群所在的 VPC。

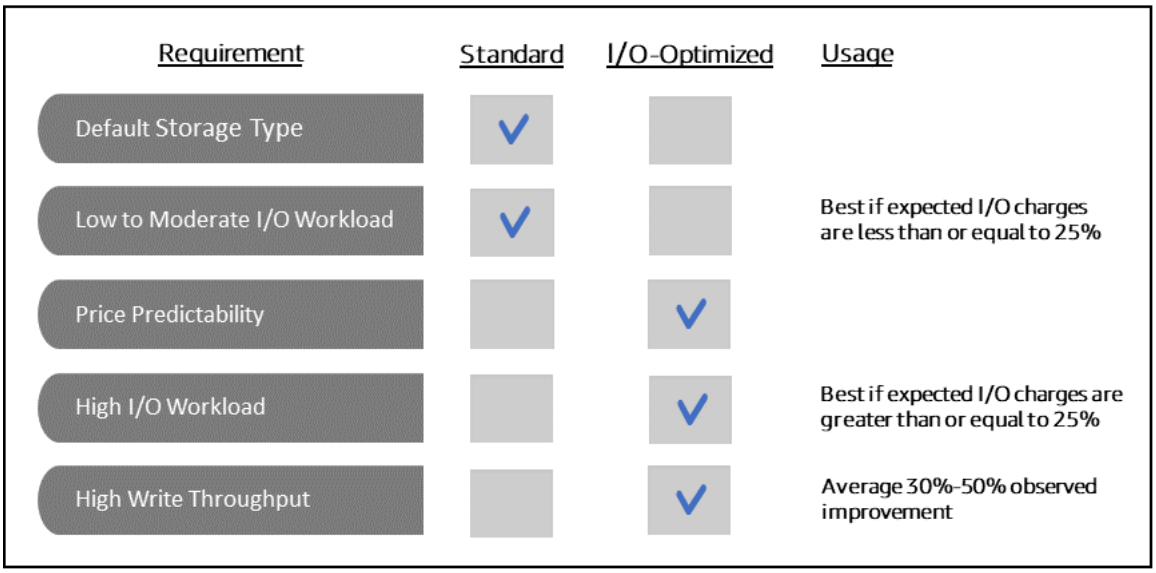
Amazon DocumentDB 集群存储配置

从 Amazon DocumentDB 5.0 开始，基于实例的集群支持以下两种存储配置类型：

- 亚马逊 DocumentDB 标准存储：专为低至中度 I/O 消耗的客户而设计。如果您预计 I/O 成本将低于亚马逊文档数据库集群总量的 25%，那么此选择可能非常适合您。使用 Amazon DocumentDB 标准存储配置，除了实例和存储费用外，您还 I/O 按请求付费计费。这意味着您的账单可能会根据周期和使用量有所不同。该配置经过量身定制，可适应您的应用不断变 I/O 化的需求。
- Amazon DocumentDB I/O-optimized 存储：专为优先考虑价格可预测性或拥有 I/O 密集型应用程序的客户而设计。该 I/O-optimized 配置为工作负载 I/O 密集的客户提供了更高的性能、更高的吞吐量和更短的延迟。如果您预计 I/O 成本将超过 Amazon DocumentDB 集群总成本的 25%，则此选项可提供更高的性价比。使用 Amazon DocumentDB I/O-optimized 存储配置，您无需根据 I/O 操作付费，从而确保每个账单周期的成本可预测。该配置可以稳定成本，同时改善性能。

您可以每 30 天将现有数据库集群切换到 Amazon DocumentDB I/O-optimized 存储空间一次。您可以在任何时间切换回 Amazon DocumentDB 标准存储。下一次修改存储配置的日期 I/O-optimized 可以使用describe-db-clusters命令进行跟踪，使用 AWS CLI 或通过集群配置页面 AWS 管理控制台 中的命令进行跟踪。

您可以创建包含 Amazon DocumentDB I/O-optimized 配置的新数据库集群，也可以通过单击几下AWS 管理控制台、在 AWS Command Line Interface (AWS CLI) 中更改单个参数或通过AWS 软件开发工具包转换现有数据库集群。在修改存储配置期间或修改之后，无需停机或重启实例。

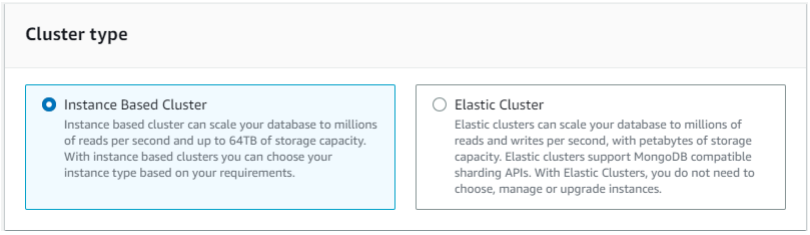


创建集 I/O-optimized 群

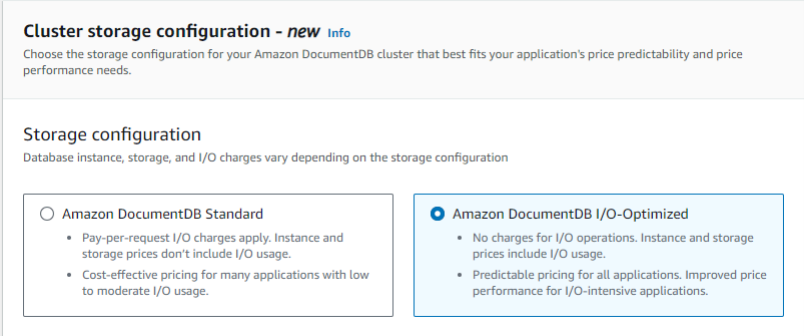
Using the AWS 管理控制台

要使用以下方法创建或修改 I/O-optimized 集群 AWS 管理控制台：

1. 在 Amazon DocumentDB 管理控制台的集群下，选择创建或选择集群，再选择操作，然后选择修改。
2. 如果要创建新的集群，请确保在集群类型部分中选择基于实例的集群（此为默认选项）。



3. 在 “配置” 部分的 “集群存储配置” 下，选择 Amazon Documen I/O-Optimized tDB。



4. 完成集群的创建或修改，并选择创建集群或修改集群。

   有关创建集群的完整流程，请参阅 使用创建集群和主实例 AWS 管理控制台。

   有关修改集群的完整流程，请参阅 修改 Amazon DocumentDB 集群。

Using the AWS CLI

要使用以下方法创建 I/O-optimized 集群，请执行以下 AWS CLI操作：

在以下示例中，用您自己的信息替换每个*user input placeholder*示例。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

确定存储配置的成本分析

借助 Amazon DocumentDB，您可以灵活地为自己拥有的每个数据库集群选择存储配置。为了在标准和之间正确分配集群 I/O-optimized，您可以按集群跟踪您的 Amazon DocumentDB 成本。为此，您可以向现有集群添加标签，在 AWS 账单与成本管理 控制面板中启用成本分配标签，并在 AWS Cost Explorer Service 中分析给定集群的成本。有关成本分析的信息，请参阅博客“使用成本分配标签”。

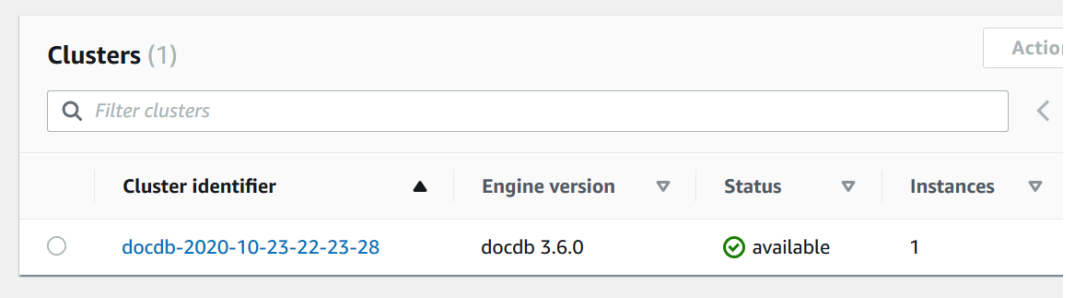
确定集群的状态

您可以使用 AWS 管理控制台 或来确定集群的状态 AWS CLI。

Using the AWS 管理控制台

使用以下步骤通过以下步骤查看您的 Amazon DocumentDB 集群的状态 AWS 管理控制台

1. 登录并打开亚马逊文档数据库控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。
3. 在 Cluster identifier (集群标识符) 列中，找到您感兴趣的集群名称。然后，要查找集群的状态，请跨该行阅读至 Status (状态) 列，如下所示。



Using the AWS CLI

使用 `describe-db-clusters` 操作以使用 AWS CLI查看 Amazon DocumentDB 集群的状态。

以下代码可查找集群 `sample-cluster` 的状态。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster  \
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

对于 Windows：

```
aws docdb describe-db-clusters ^
    --db-cluster-identifier sample-cluster  ^
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

此操作的输出将类似于下文（JSON 格式）。

```
[
    [
        "sample-cluster",
        "available"
    ]
]
```

Amazon DocumentDB 集群生命周期

Amazon DocumentDB 集群的生命周期包括创建、描述、修改和删除集群。本节提供有关如何完成这些过程的信息。

主题

- 创建 Amazon DocumentDB 集群
- 描述 Amazon DocumentDB 集群
- 修改 Amazon DocumentDB 集群
- 确定待处理维护
- 对集群的引擎版本执行补丁更新
- 停止和启动 Amazon DocumentDB 集群
- 删除 Amazon DocumentDB 集群

创建 Amazon DocumentDB 集群

Amazon DocumentDB 集群由一些实例和一个表示该集群的数据的集群卷组成。集群卷作为单个虚拟卷在三个可用区之间的 6 个方向进行复制。集群包含一个主实例，以及可选的最多 15 个副本实例。

以下各节介绍如何使用 AWS 管理控制台 或创建 Amazon DocumentDB 集群。 AWS CLI然后，您可为该集群添加更多副本实例。使用控制台创建 Amazon DocumentDB 集群时，会同时自动为您创建一个

主实例。如果您使用创建您的 AWS CLI Amazon DocumentDB 集群，则在集群的状态变为可用之后，您必须为该集群创建主实例。

先决条件

以下是创建 Amazon DocumentDB 集群的先决条件。

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明操作。

   在注册时，将接到电话或收到短信，要求使用电话键盘输入一个验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

VPC 先决条件

只能在 Amazon Virtual Private Cloud (Amazon VPC) 中创建 Amazon DocumentDB 集群。要在 Amazon DocumentDB 集群中使用您的 Amazon VPC，该 Amazon VPC 必须最少在两个可用区中均拥有至少一个子网。通过跨可用区分配您的集群实例，您可以确保集群中有可用的实例，避免出现可用区故障。

子网先决条件

创建 Amazon DocumentDB 集群时，您必须选择一个 VPC 和该 VPC 中的对应子网组来启动您的集群。子网确定可用区以及该可用区内要用于启动实例的 IP 范围。为进行此讨论，我们将互换使用术语子网 和可用区。子网组是一组指定的子网（或可用区）。子网组可让您指定要用于启动 Amazon DocumentDB 实例的可用区。例如，在包含三个实例的集群中，为了实现高可用性，建议在单独的可用区中配置各个实例。因此，如果单个可用区出现故障，它只会影响单个实例。

Amazon DocumentDB 实例目前可在多达三个可用区中预配置。即使子网组拥有三个以上的子网，您也只能使用其中的三个子网来创建 Amazon DocumentDB 集群。因此，在创建子网组时，建议仅选择要将实例部署到的三个子网。在美国东部（弗吉尼亚州北部），您的子网组可以有六个子网（或可用区）。但是，当已预配置一个 Amazon DocumentDB 集群后，Amazon DocumentDB 将选择这些可用区中将用于预配置实例的三个可用区。

例如，假设您在创建集群时，Amazon DocumentDB 选择可用区 {1A、1B 和 1C}。如果您尝试在可用区 {1D} 中创建实例，API 调用将失败。但是，如果您选择在不指定特定可用区的情况下创建实例，那么 Amazon DocumentDB 会代表您选择一个可用区。Amazon DocumentDB 使用一种算法在可用区之间对实例进行负载均衡，以帮助您实现高可用性。例如，如果配置了三个实例，则在默认情况下，将在三个可用区中配置它们，而不会在单个可用区中配置。

建议

- 除非您有特殊原因，否则请始终创建包含三个子网的子网组。这样做可帮助确保包含三个或更多实例的集群能够实现更高的可用性，因为将在三个可用区中预配置实例。
- 始终将实例分散在多个可用区中以实现高可用性。切勿将集群的所有实例放在单个可用区中。
- 由于故障转移事件随时可能发生，您不应假定主实例或副本实例始终位于特定可用区中。

其他先决条件

以下是创建 Amazon DocumentDB 集群的一些其他先决条件：

- 如果您 AWS 使用 AWS Identity and Access Management (IAM) 证书进行连接，则您的 IAM 账户必须具有授予执行 Amazon DocumentDB 操作所需权限的 IAM 策略。

  如果您使用 IAM 账户访问亚马逊 DocumentDB 控制台，则必须先使用您的 IAM 账户登录。 AWS 管理控制台 然后前往亚马逊文档数据库控制台，网址为。https://console.aws.amazon.com/docdb
- 如果要定制您的集群的配置参数，您必须指定集群参数组和具有必需参数设置的参数组。有关创建或修改集群参数组的信息，请参阅 管理 Amazon DocumentDB 集群参数组。
- 您必须确定要为集群指定的 TCP/IP 端口号。有些公司的防火墙不允许连接到 Amazon DocumentDB 的默认端口。如果您公司的防火墙阻止该默认端口，请为您的集群选择其他端口。集群中的所有实例都使用同一端口。

使用创建集群和主实例 AWS 管理控制台

以下过程介绍了如何使用控制台启动包含一个或多个实例的 Amazon DocumentDB 集群。

创建集群：使用默认设置

要使用默认设置创建包含实例的集群，请使用 AWS 管理控制台

1. 登录并打开亚马逊文档数据库控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb

2. 如果您想在美国东部（弗吉尼亚北部）以外 AWS 区域 的其他地区创建集群，请从控制台右上角的列表中选择该区域。
3. 在导航窗格中，选择 Clusters (集群)，然后选择 Create (创建)。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

4. 在创建 Amazon DocumentDB 集群页中，完成配置窗格。

   a. 集群标识符：接受 Amazon DocumentDB 提供的名称，或者为您的集群输入名称，例如 **`sample-cluster`**。

      集群命名约束：

      - 长度为 [1-63] 个字母、数字或连字符。
      - 第一个字符必须是字母。
      - 不能以连字符结尾或包含两个连续的连字符。
      - 每个区域的 Amazon RDS、Neptune 和 Amazon DocumentDB 中的所有集群都必须是唯一 AWS 账户的。

   b. 引擎版本-接受默认引擎版本 5.0.0，或者可以选择选择 8.0.0、4.0.0 或 3.6.0。

   c. 实例类：接受默认值 `db.r5.large`，或者从列表中选择您需要的实例类。

   d. 实例数：在列表中，选择要使用此集群创建的实例的数量。第一个实例为主实例，所有其他实例为只读副本实例。您可以在稍后添加和删除实例（如果需要）。默认情况下，Amazon DocumentDB 集群将以三个实例（一个主实例和两个副本）启动。

5. 完成集群存储配置部分。

   选择亚马逊 DocumentDB 标准（默认）或亚马逊 Document DB。 I/O-Optimized有关更多信息，请参阅 Amazon DocumentDB 集群存储配置。

6. 完成 Authentication (身份验证) 窗格。

   a. 用户名：输入主用户的名称。要登录您的集群，您必须使用主用户名称。

      主用户命名约束：

      - 长度为 [1-63] 个字母数字字符。

- 第一个字符必须是字母。
- 不能是数据库引擎的保留字。

b. 请选择以下密码选项之一：

- 托管于 AWS Secrets Manager-如果您想自动管理您的主用户密码 AWS Secrets Manager ，请选择此选项。

  如果您选择此选项，请通过创建您自己的密钥或使用 Secrets Manager 创建的密钥来配置 KMS 密钥。

- 自行管理：如果您想要自行管理您的主用户密码，请选择此选项。如果选择此选项，请输入主用户的密码，然后确认该密码。要登录您的集群，您必须使用主用户密码。

  密码约束：

  - 长度为 [8-100] 个可打印 ASCII 字符。
  - 可以使用任何可打印 ASCII 字符，以下字符除外：
    - **/**（正斜杠）
    - **"**（双引号）
    - **@**（@ 符号）

7. 在屏幕的底部，选择以下选项之一：

- 要立即创建集群，请选择 Create cluster (创建集群)。
- 要不创建集群，请选择 Cancel (取消)。
- 要在创建集群前进一步配置集群，请选择 Show additional configurations (显示其他配置)，然后在创建集群：其他配置上继续。

  Additional Configurations (其他配置) 部分中包含的配置如下：

  - 网络设置：默认值为使用 `default` VPC 安全组。
  - 集群选项：默认值为使用端口 27017 和默认参数组。
  - 加密：默认值为使用 `(default) aws/rds` 密钥启用加密。

    > **Important**
    >
    > 集群加密后，就不能取消加密。

  - 备份：默认值为保留备份 1 天并让 Amazon DocumentDB 选择备份时段。

- 日志导出-默认为不将审核日志导出到 CloudWatch 日志。
- 维护：默认设置是让 Amazon DocumentDB 选择维护时段。
- 删除保护：防止您的集群被意外删除。使用控制台创建的集群的默认设置为已启用。

如果您现在接受默认设置，则可以稍后通过修改集群来更改其中的大部分。

8. 为集群的安全组启用入站连接。

如果未更改集群的默认设置，则会使用给定区域中的默认 VPC 的默认安全组创建一个集群。要连接到 Amazon DocumentDB，您必须在端口 27017（或所选的端口）上为集群的安全组启用入站连接。

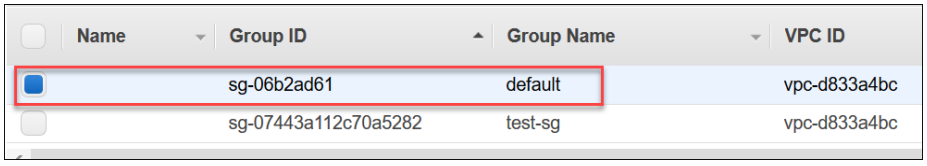
将入站连接添加到集群的安全组

a. 登录 AWS 管理控制台 并打开 Amazon EC2 控制台，网址为https://console.aws.amazon.com/ec2/。

b. 在主窗口的资源部分中，选择安全组。



c. 从安全组列表中，找到您在创建集群时使用的安全组（很可能是 default 安全组），然后选中安全组名称左侧的框。



d. 从操作菜单中，选择编辑入站规则，然后选择或输入规则限制。

i. 类型：从列表中，选择要为网络流量打开的协议。

ii. 协议：从列表中，选择协议类型。

iii. 端口范围：对于自定义规则，请输入端口号或端口范围。确保端口号或范围包括您在创建集群时指定的端口（默认值：27017）。

iv. 源：指定可以到达您的实例的流量。从列表中，选择流量源。如果选择“自定义”，请使用 CIDR 表示法指定单个 IP 地址或 IP 地址范围（例如 203.0.113）。 5/32)。

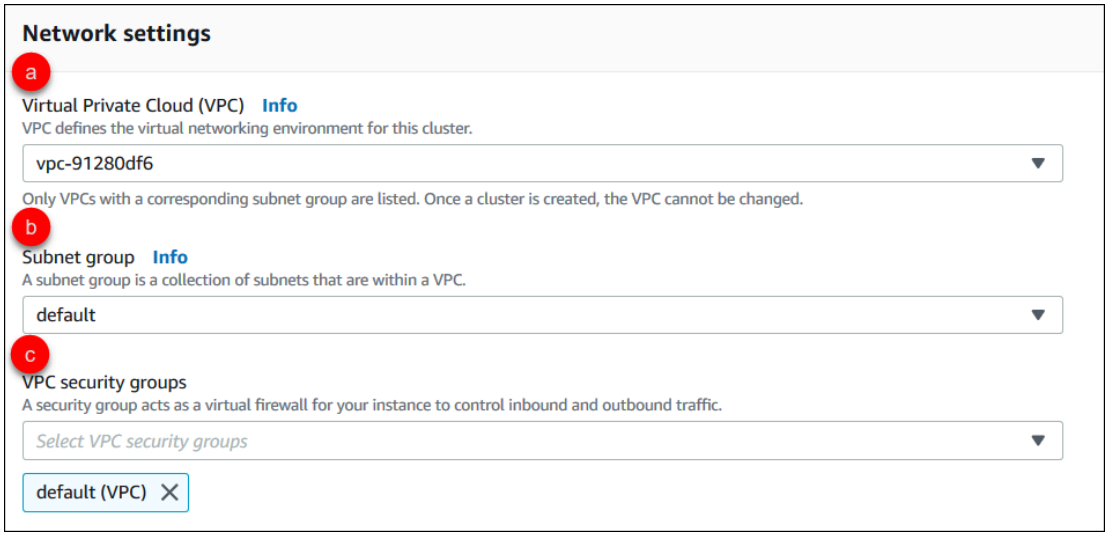
v. 描述：输入该规则的描述。

vi. 在创建完规则时，选择保存。

创建集群：其他配置

如果要接受集群的默认设置，您可以跳过以下步骤并选择 Create cluster (创建集群)。

1. 完成 Network settings (网络设置) 窗格。



a. 虚拟私有云（VPC）：在列表中，选择要在其中启动此集群的 Amazon VPC。

b. 子网组：在列表中，选择要用于此集群的子网组。

c. VPC 安全组：在列表中为此集群选择 VPC 安全组。

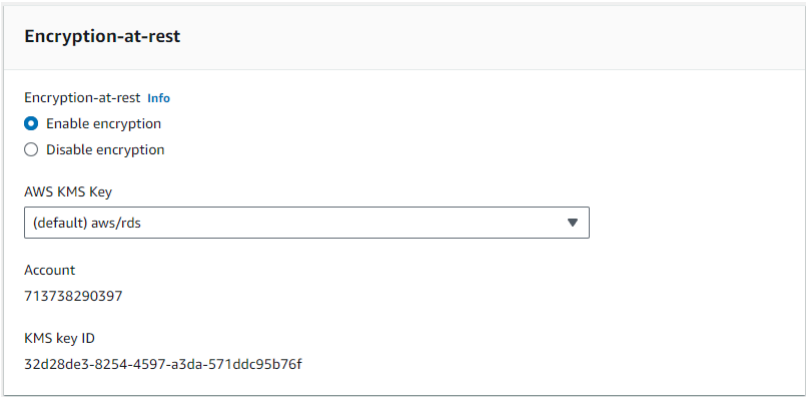
2. 完成 Cluster options (集群选项) 窗格。



a. 数据库端口-使用向上和向下箭头设置应用程序用于连接到您的实例的 TCP/IP 端口。

b. 集群参数组：在参数组列表中，为此集群选择集群参数组。

3. 完成 Encryption (加密) 窗格。



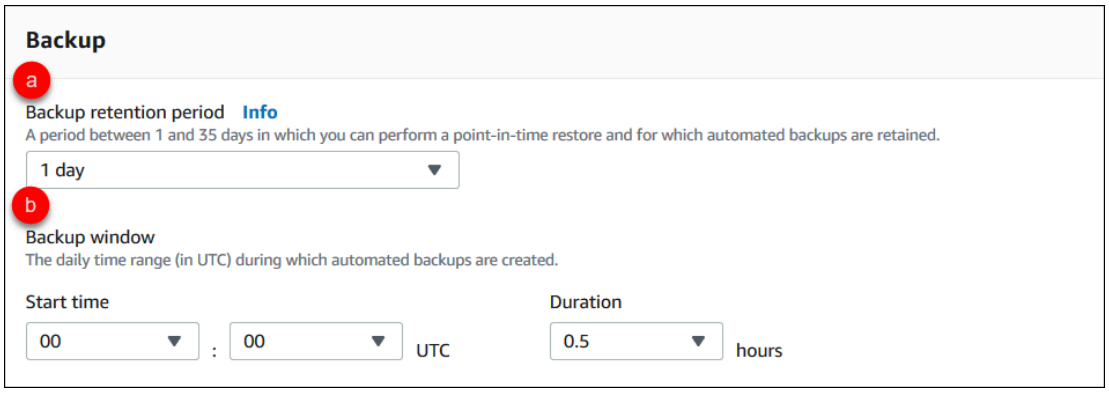
a. Encryption-at-rest—选择以下选项之一：

- 启用加密：默认值。所有静态数据都会加密。如果您选择加密您的数据，则无法撤消此操作。
- 禁用加密：您的数据不会被加密。

b. AWS KMS 密钥-只有在加密数据时才可用。在列表中，选择要用于加密此集群中的数据的密钥。默认值为 `(default) aws/rds`。

如果您选择 Enter a key ARN (输入一个密钥 ARN)，则必须为密钥输入一个 Amazon 资源名称 (ARN)。

4. 完成 Backup (备份) 窗格。



a. 备份留存期：在列表中，选择在删除此集群的自动备份前保留它们的天数。

b. 备份时段：设置 Amazon DocumentDB 要备份此集群的每日时间和持续时间。

i. 开始时间：在第一个列表中，选择开始自动备份的开始时间小时 (UTC)。在第二个列表中，选择您希望自动备份开始的时间（分钟）。

ii. 持续时间：在该列表中，选择要向创建自动备份分配的小时数。

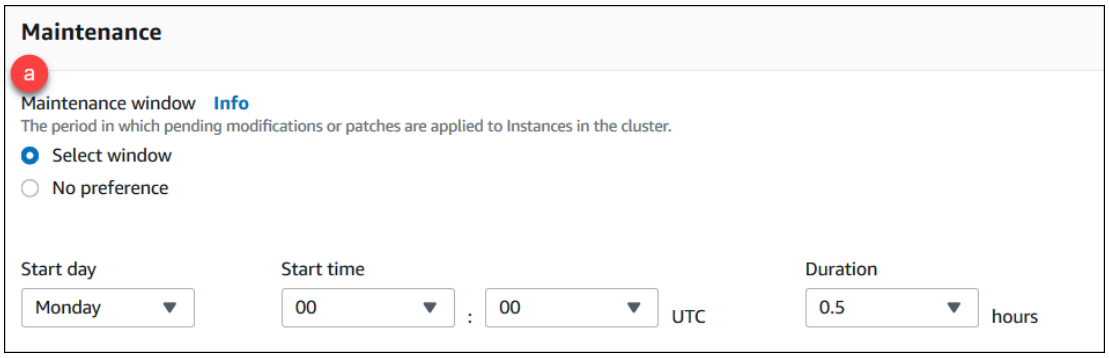
5. 选择要导出到日志的日志类型，完成日志导出 CloudWatch 窗格。



- 审核日志-选择此选项可启用将审核日志导出到 Amazon Lo CloudWatch gs。如果您选择 Audit logs (审计日志)，则必须在集群的自定义参数组中启用 `audit_logs`。有关更多信息，请参阅 审核 Amazon DocumentDB 事件。
- Profiler 日志-选择此选项可启用将操作分析器日志导出到 Amazon Logs。 CloudWatch 如果您选择 Profiler logs (分析器日志)，则还必须在集群的自定义参数组中修改以下参数：
  - `profiler`：设置为 `enabled`。
  - `profiler_threshold_ms`：设置为 `[0-INT_MAX]` 之间的值，以设置分析操作的阈值。
  - `profiler_sampling_rate`：设置为 `[0.0-1.0]` 之间的值，以设置要分析的缓慢操作的百分比。

  有关更多信息，请参阅 分析 Amazon DocumentDB 操作。

6. 完成 Maintenance (维护) 窗格。

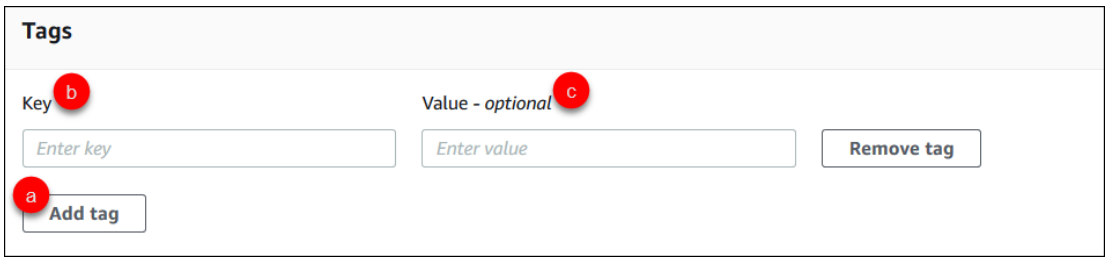


- 选择以下选项之一
  - 选择时段：您可以指定 Amazon DocumentDB 对您的集群执行维护的周日期、UTC 开始时间和持续时间。
    a. 开始日：在列表中，选择开始集群维护的一周中的天。
    b. 开始时间：在列表中，选择开始维护的小时和分钟 (UTC)。

c. 持续时间：在列表中，选择要为集群维护分配的时长。如果无法在指定时间内完成维护，维护过程将在指定时间过后继续进行，直到完成。

- 无首选项：Amazon DocumentDB 为执行维护选择一周中的天、开始时间和持续时间。

7. 如果您要将一个或多个标签添加到此集群，请填写标签窗格。



对于每个要添加到集群的标签，请重复以下步骤。一个集群最多可以有 10 个标签。

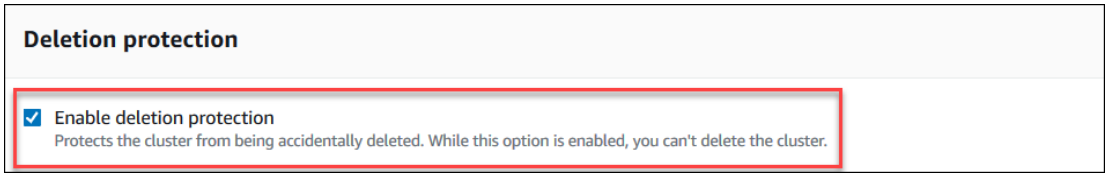
a. 选择添加标签。

b. 键入标签的键。

c. （可选）键入标签的值。

要删除标签，请选择 Remove tag（删除标签）。

8. 在使用控制台创建集群时，将默认启用删除保护。要禁用删除保护，请清除启用删除保护。启用后，删除保护可以防止删除集群。要删除某个受到删除保护的集群，您必须先修改集群以禁用删除保护。



有关删除保护的更多信息，请参阅 删除 Amazon DocumentDB 集群。

9. 要创建集群，请选择创建集群。否则，选择取消。

使用创建集群 AWS CLI

以下过程介绍如何使用启动亚马逊 DocumentDB 集群和创建 Amazon DocumentDB 副本。 AWS CLI

参数

- **`--db-cluster-identifier`**：必填项。标识此集群的一个小写字符串。

集群命名约束：

- 长度为 [1-63] 个字母、数字或连字符。
- 第一个字符必须是字母。
- 不能以连字符结尾或包含两个连续的连字符。
- 每个区域的 AWS 每个账户的所有集群（跨亚马逊 RDS、Amazon Neptune 和 Amazon DocumentDB）必须是唯一的。

- **`--engine`**：必填项。必须是 **`docdb`**。
- **`--deletion-protection | --no-deletion-protection`**：可选。启用删除保护后，可以防止删除集群。使用时 AWS CLI，默认设置是禁用删除保护。

  有关删除保护的更多信息，请参阅 删除 Amazon DocumentDB 集群。
- **`--storage-type standard | iopt1`**：可选。默认值：**`standard`**。集群存储配置。有效值为`standard`（标准）或`iopt1`（I/O-optimized）。
- **`--master-username`**：必填项。用于对用户进行身份验证的用户名。

  主用户命名约束：

  - 长度为 [1-63] 个字母数字字符。
  - 第一个字符必须是字母。
  - 不能是数据库引擎的保留字。
- **`--master-user-password`**：可选。用于对用户进行身份验证的用户密码。

  主密码约束：

  - 长度为 [8-100] 个可打印 ASCII 字符。
  - 可以使用任何可打印 ASCII 字符，以下字符除外：
    - **`/`**（正斜杠）
    - **`"`**（双引号）
    - **`@`**（@ 符号）

- **`--manage-master-user-password`**：可选。Amazon DocumentDB 会生成主用户密码并在其整个生命周期中在 Secrets Manager 中对其进行管理。

有关其他参数，请参阅 createdbCluster。

要启动亚马逊文档数据库集群，请使用 AWS CLI

要创建 Amazon DocumentDB 集群，请调用。`create-db-cluster` AWS CLI以下 AWS CLI 命令创建一个名为`sample-cluster`且启用了删除保护的 Amazon DocumentDB 集群。有关删除保护的更多信息，请参阅 删除 Amazon DocumentDB 集群。

此外，`--engine-version` 是一个默认成主引擎最新版本的可选参数。当前的默认引擎版本为 5.0.0（注意：亚马逊 DocumentDB 8.0 可用，但必须明确指定）。发布主引擎新版本时，`--engine-version` 的默认引擎版本将更新，以反映最近的主引擎版本。因此，对于生产工作负载，尤其是那些依赖脚本、自动化或 CloudFormation 模板的工作负载，我们建议您明确指定预期的主要版本。`--engine-version`

> **Note**
>
> 如果未指定 `db-subnet-group-name` 或 `vpc-security-group-id`，则 Amazon DocumentDB 将使用给定区域的默认子网组和 Amazon VPC 安全组。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "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"
    }
}
```

创建集群需要几分钟时间。您可以使用 AWS 管理控制台 或 AWS CLI 来监控集群的状态。有关更多信息，请参阅 监控 Amazon DocumentDB 集群的状态。

> **Important**
>
> 当您使用创建 Amazon DocumentDB 集群时，不会创建任何实例。 AWS CLI 因此，您必须显式创建主实例和所需的任何副本实例。您可以使用控制台或创建 AWS CLI 实例。有关更多信息，请参阅 向集群添加 Amazon DocumentDB 实例。

有关更多信息，请参阅 Amazon DocumentDB API 参考中的 `CreateDBCluster`。

描述 Amazon DocumentDB 集群

您可以使用 Amazon DocumentDB 管理控制台或查看与您的 Amazon DocumentDB 集群相关的连接终端节点、安全组、VPC 和参数组等详细信息。 AWS CLI

有关更多信息，请参阅下列内容：

- 监控 Amazon DocumentDB 集群的状态
- 查找集群的端点

Using the AWS 管理控制台

通过以下过程使用控制台查看指定 Amazon DocumentDB 集群的详细信息。

1. 登录并打开亚马逊文档数据库控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在集群列表中，选择要查看其详细信息的集群的名称。有关集群的信息分为以下几组：

   - 摘要：有关集群的一般信息，包括引擎版本、集群状态、待定维护及其参数组的状态。

- 连接和安全：连接部分列出要使用 mongo Shell 或应用程序连接到该集群的连接端点。Security Groups (安全组) 部分列出与此集群关联的安全组及其 VPC ID 和描述。
- 配置：集群详细信息部分列出有关集群的详细信息，包括集群的 Amazon 资源名称（ARN）、端点和参数组。其中还列出集群的备份信息、维护详细信息以及安全和网络设置。Cluster instances (集群实例) 部分列出属于该集群的实例，其中包括每个实例的角色和集群参数组状态。
- 监控-此集群的 Amazon CloudWatch 日志指标。有关更多信息，请参阅 使用以下方式监控亚马逊 DocumentDB CloudWatch。
- 事件和标签：近期事件部分列出该集群的近期事件。Amazon DocumentDB 记录与集群、实例、快照、安全组和集群参数组相关的事件。此信息包括与每个事件关联的日期、时间和消息。Tags (标签) 部分列出附加该集群的标签。

Using the AWS CLI

要使用查看您的 Amazon DocumentDB 集群的详细信息 AWS CLI，请使用以下示例所示的`describe-db-clusters`命令。有关更多信息，请参阅 Amazon DocumentDB 资源管理 API 参考中的 `DescribeDBClusters`。

> **Note**
>
> 对于某些管理功能（如集群和实例生命周期管理），Amazon DocumentDB 利用与 Amazon RDS 共享的操作技术。`filterName=engine,Values=docdb` 筛选器参数仅返回 Amazon DocumentDB 集群。

Example

示例 1：列出所有 Amazon DocumentDB 集群

以下 AWS CLI 代码列出了某个区域中所有 Amazon DocumentDB 集群的详细信息。

```
aws docdb describe-db-clusters --filter Name=engine,Values=docdb
```

此操作的输出将类似于下文。

```
{
    "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

示例 2：列出指定 Amazon DocumentDB 集群的所有详细信息

以下 AWS CLI 代码列出了集群的详细信息sample-cluster。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters \
   --filter Name=engine,Values=docdb \
   --db-cluster-identifier sample-cluster
```

对于 Windows：

```
aws docdb describe-db-clusters ^
   --filter Name=engine,Values=docdb ^
   --db-cluster-identifier sample-cluster
```

此操作的输出将类似于下文。

```
{
    "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

示例 3：列出 Amazon DocumentDB 集群的指定详细信息

要使用列出集群详细信息的子集，请添加一个 AWS CLI`--query`，指定要列出`describe-db-clusters`操作要列出的集群成员。`--db-cluster-identifier` 参数是要显示其详细信息的特定集群的标识符。有关查询的更多信息，请参阅 AWS Command Line Interface 用户指南中的如何使用 `--query` 选项筛选输出。

以下示例列出 Amazon DocumentDB 集群中的实例。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters \
    --filter Name=engine,Values=docdb \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterMembers]'
```

对于 Windows：

```
aws docdb describe-db-clusters ^
    --filter Name=engine,Values=docdb ^
    --db-cluster-identifier sample-cluster ^
    --query 'DBClusters[*].[DBClusterMembers]'
```

此操作的输出将类似于下文。

```
[
    [
        [
            {
                "DBInstanceIdentifier": "sample-instance-1",
                "IsClusterWriter": true,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            },
            {
                "DBInstanceIdentifier": "sample-instance-2",
                "IsClusterWriter": false,
                "DBClusterParameterGroupStatus": "in-sync",
                "PromotionTier": 1
            }
```

```
        ]
    ]
]
```

修改 Amazon DocumentDB 集群

要修改集群，该集群必须处于 可用 状态。您无法修改已停止的集群。如果集群已停止，则首先启动集群，等待集群进入可用状态，然后进行所需修改。有关更多信息，请参阅 停止和启动 Amazon DocumentDB 集群。

Using the AWS 管理控制台

通过以下过程使用控制台修改特定的 Amazon DocumentDB 集群。

修改 Amazon DocumentDB 集群

1. 登录并打开亚马逊文档数据库控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。

   > **Tip**
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 通过选择集群名称左侧的按钮，指定要修改的集群。
4. 选择 Actions (操作)，然后选择 Modify (修改)。
5. 在 Modify Cluster: cluster-name (修改集群: cluster-name) 窗格中，进行所需更改。您可以更改以下方面：

   - 集群规范：集群的名称、安全组和凭证管理。
   - 集群存储配置：集群的数据存储模式。在 “标准” 和 “ I/O-Optimized 配置” 之间进行选择。
   - 集群选项：集群的端口和参数组。
   - 备份：集群的备份留存期和备份时段。
   - 日志导出：启用或禁用导出审计日志或分析器日志。
   - 维护：设置集群的维护时段。

- 删除保护：对集群启用或禁用删除保护。默认情况下，将启用删除保护。

6. 完成后，选择 Continue (继续) 以查看更改摘要。
7. 如果您对所做更改满意，可以选择 Modify cluster (修改集群) 以修改集群。或者，您可以选择 Back (返回) 或 Cancel (取消) 来编辑或取消您的更改。

需要几分钟时间才能应用更改。只有在实例状态为 available (可用) 时，才能使用集群。您可以使用控制台或 AWS CLI监控集群状态。有关更多信息，请参阅 监控 Amazon DocumentDB 集群的状态。

Using the AWS CLI

使用 `modify-db-cluster` 操作以使用 AWS CLI修改指定的集群。有关更多信息，请参阅 Amazon DocumentDB API 参考中的 `ModifyDBCluster`。

参数

- **`--db-cluster-identifier`**：必填项。您要修改的 Amazon DocumentDB 集群的标识符。
- **`--backup-retention-period`**：可选。自动备份的保留天数。有效值为 1-35。
- **`--storage-type`**：可选。集群存储配置。有效值为`standard`（标准）或`iopt1`（I/O-optimized）。
- **`--db-cluster-parameter-group-name`** — 可选。用于集群的集群参数组的名称。
- **`--manage-master-user-password`**：可选。Amazon DocumentDB 会生成主用户密码并在其整个生命周期中在 Secrets Manager 中对其进行管理。
- **`--rotate-master-user-password`**：可选。Secrets Manager 会为现有密钥生成一个新的密钥版本。密钥的新版本包含新的主用户密码。Amazon DocumentDB 会更改集群的主用户密码以匹配新密钥版本的密码。

  轮换主密码时必须指定 **`--apply-immediately`** 选项。
- **`--master-user-password`**：可选。主数据库用户的新密码。

  密码约束：
  - 长度为 [8–100] 个可打印 ASCII 字符。
  - 可以使用任何可打印 ASCII 字符，以下字符除外：
    - **`/`**（正斜杠）
    - **`"`**（双引号）

  - **@**（@ 符号）
- **--new-db-cluster-identifier**：可选。重命名集群时集群的新集群标识符。此值以一个小写字符串存储。

  命名约束：

  - 长度为 [1-63] 个字母、数字或连字符。
  - 第一个字符必须是字母。
  - 不能以连字符结尾或包含两个连续的连字符。
  - 每个区域的 Amazon RDS、Amazon Neptune 和 Amazon DocumentDB 中的所有集群都必须是唯一 AWS 账户的。
- **--preferred-backup-window**：可选。创建自动备份的每日时间范围，以通用协调时间 (UTC) 表示。
  - 格式：hh24:mm-hh24:mm
- **--preferred-maintenance-window**：可选。可能进行系统维护的每周时间范围，以 UTC 表示。
  - 格式：ddd:hh24:mm-ddd:hh24:mm
  - 有效值：Sun、Mon、Tue、Wed、Thu、Fri 和 Sat。
- **--deletion-protection** 或 **--no-deletion-protection**：可选。是否应在此集群上启用删除保护。删除保护可以防止集群被意外删除，直到修改集群以禁用删除保护。有关更多信息，请参阅 删除 Amazon DocumentDB 集群。
- **--apply-immediately** 或 **--no-apply-immediately**：使用 --apply-immediately 可立即进行更改。使用 --no-apply-immediately 在集群的下一个维护时段期间进行更改。

Example

以下代码将更改集群 sample-cluster 的备份保留期。

对于 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster \
       --db-cluster-identifier sample-cluster \
       --apply-immediately \
       --backup-retention-period 7
```

对于 Windows：

```
aws docdb modify-db-cluster ^
       --db-cluster-identifier sample-cluster ^
       --apply-immediately ^
       --backup-retention-period 7
```

此操作的输出将类似于下文。

```
{
    "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
    }
}
```

需要几分钟时间才能应用更改。只有在实例状态为 available (可用) 时，才能使用集群。您可以使用控制台或 AWS CLI监控集群状态。有关更多信息，请参阅 监控 Amazon DocumentDB 集群的状态。

确定待处理维护

您可以通过确定是否有待处理的集群维护来确定您是否拥有最新的 Amazon DocumentDB 引擎版本。

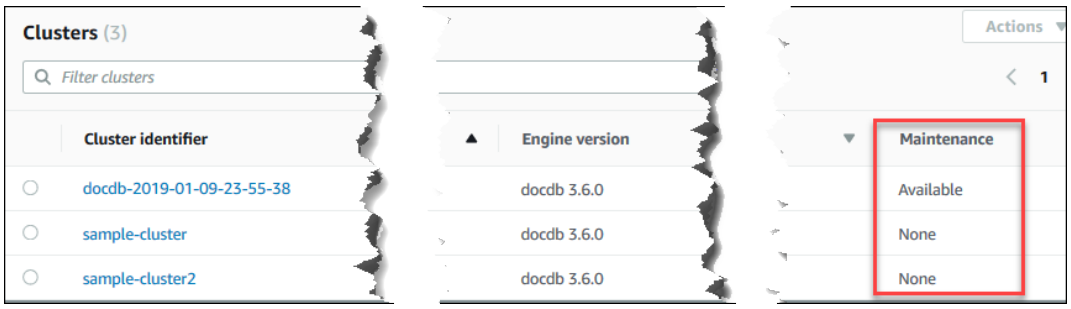
Using the AWS 管理控制台

您可以使用 AWS 管理控制台 来确定集群是否有待维护的状态。

1. 登录并打开亚马逊文档数据库控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 找到 Maintenance (维护) 列以确定集群是否有待处理维护。



None (无) 表示集群运行的是最新的引擎版本。Available (可用) 表示集群有待处理维护，这可能意味着需要升级引擎。

4. 如果您的集群有待处理维护，请继续执行对集群的引擎版本执行补丁更新中的步骤。

Using the AWS CLI

您可以使用带有以下参数的`describe-pending-maintenance-actions`操作 AWS CLI 来确定集群是否具有最新的引擎版本。

参数

- **`--resource-identifier`**：可选。资源（集群）的 ARN。如果省略该参数，则列出所有集群的待处理维护操作。
- **`--region`**：可选。要在其中运行该操作的 AWS 区域，例如 `us-east-1`。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb describe-pending-maintenance-actions \
   --resource-identifier arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster \
   --region us-east-1
```

对于 Windows：

```
aws docdb describe-pending-maintenance-actions ^
   --resource-identifier arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster ^
   --region us-east-1
```

此操作的输出将类似于下文。

```
{
    "PendingMaintenanceActions": [
        {
            "ResourceIdentifier": "arn:aws:rds:us-
east-1:123456789012:cluster:sample-cluster",
            "PendingMaintenanceActionDetails": [
                {
                    "Description": "New feature",
                    "Action": "db-upgrade",
                    "ForcedApplyDate": "2019-02-25T21:46:00Z",
                    "AutoAppliedAfterDate": "2019-02-25T07:41:00Z",
                    "CurrentApplyDate": "2019-02-25T07:41:00Z"
                }
            ]
        }
    ]
}
```

如果您的集群有待处理维护，请继续执行对集群的引擎版本执行补丁更新中的步骤。

对集群的引擎版本执行补丁更新

在本节中，我们将说明如何使用 AWS 管理控制台 或部署补丁更新 AWS CLI。补丁更新是相同引擎版本内的更新（例如，将 3.6 引擎版本更新到较新的 3.6 引擎版本）。您可以立即更新它，也可以在集群的下一个维护时段进行更新。要确定您的引擎是否需要更新，请参阅 确定待处理维护。请注意，当您应用更新时，您的集群将经历一些停机时间。

> **Note**
>
> 如果您正在尝试从主引擎版本升级到另一个引擎版本，例如 3.6 到 5.0 或 5.0 到 8.0，请参阅Amazon DocumentDB 主版本就地升级或使用升级您的亚马逊文档数据库集群 AWS Database Migration Service。

要获取集群的引擎版本的最新补丁更新，需要满足两个配置要求：

- 集群状态必须为 可用。

- 该集群必须运行较早的引擎版本。

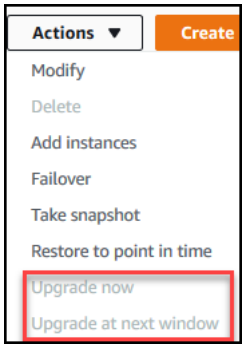
Using the AWS 管理控制台

以下过程使用控制台对集群的引擎版本应用补丁更新。您可以选择立即更新，也可以选择在集群的下一维护时段期间更新。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。在集群列表中，选择要升级的集群左侧的按钮。集群状态必须为可用。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 从 Actions (操作) 菜单中，选择以下选项之一。仅当您选择的集群未运行最新的引擎版本时，才可以选择这些菜单选项。

   - 立即升级：立即启动升级过程。集群升级到最新引擎版本时，集群将离线一段时间。
   - 在下一个时段升级：在集群的下一维护时段内启动升级过程。集群升级到最新引擎版本时，它将离线一段时间。
4. 当确认窗口打开时，选择以下选项之一：
   - 升级：根据上一步中选择的计划将集群升级到最新引擎版本。
   - 取消：取消集群的引擎升级并继续使用集群的当前引擎版本。

Using the AWS CLI

您可以使用 AWS CLI 和带有以下参数的apply-pending-maintenance-action操作将补丁更新应用于您的集群。

参数

- **--resource-identifier**：必填项。您要升级的 Amazon DocumentDB 集群的 ARN。
- **--apply-action**：必填项。允许使用以下值。要升级集群的引擎版本，请使用 db-upgrade。
  - **db-upgrade**
  - **system-update**
- **--opt-in-type**：必填项。允许使用以下值。
  - immediate：立即应用维护操作。
  - next-maintenance：在下一个维护时段内应用维护操作。
  - undo-opt-in：取消任何现有的 next-maintenance 加入请求。

Example

以下示例将 sample-cluster 的引擎版本的补丁更新到版本 4.0.0。

对于 Linux、macOS 或 Unix：

```
aws docdb apply-pending-maintenance-action \
   --resource-identifier arn:aws:rds:us-east-1:123456789012\:cluster:sample-cluster
 \
   --apply-action db-upgrade \
   --opt-in-type immediate
```

对于 Windows：

```
aws docdb apply-pending-maintenance-action ^
   --resource-identifier arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster ^
   --apply-action db-upgrade ^
   --opt-in-type immediate
```

此操作的输出将类似于以下内容：

```
{
```

```
    "ResourcePendingMaintenanceActions": {
        "ResourceIdentifier": "arn:aws:rds:us-
east-1:444455556666:cluster:docdb-2019-01-09-23-55-38",
        "PendingMaintenanceActionDetails": [
            {
                "CurrentApplyDate": "2019-02-20T20:57:06.904Z",
                "Description": "Bug fixes",
                "ForcedApplyDate": "2019-02-25T21:46:00Z",
                "OptInStatus": "immediate",
                "Action": "db-upgrade",
                "AutoAppliedAfterDate": "2019-02-25T07:41:00Z"
            }
        ]
    }
}
```

停止和启动 Amazon DocumentDB 集群

停止和启动 Amazon DocumentDB 集群可以帮助您控制开发和测试环境的成本。当您不需要集群中的所有实例时，您可以暂时停止这些实例，而不是每次使用 Amazon DocumentDB 时创建和删除集群和实例。然后，当您恢复测试时，可以再次启动它们。

主题

- 停止和启动集群概述
- 可以在已停止的集群上执行的操作

停止和启动集群概述

在不需要使用 Amazon DocumentDB 集群期间，您可以同时停止该集群中的所有实例。然后，您可以在需要使用时再次启动集群。启动和停止简化了用于开发、测试或不需要持续可用性的类似活动的集群的设置和停用过程。无论集群中有多少实例，您都可以使用 AWS 管理控制台 或通过单个操作停止和启动集群。 AWS CLI

当您的集群停止后，集群存储卷保持不变。您只需在指定的保留时段内为集群存储、手动快照和自动备份存储付费。您无需为任何实例小时数付费。Amazon DocumentDB 会在七天后自动重新启动您的集群，这样一来您的集群就可以及时获得任何所需的维护更新。当您的集群在 7 天后启动时，您将再次开始为集群中的实例付费。当您的集群停止时，您无法查询您的存储卷，因为查询需要实例处于可用状态。

当 Amazon DocumentDB 集群停止后，不能以任何方式修改该集群或其实例。这包括添加或删除实例，或者删除集群。

Using the AWS 管理控制台

以下过程显示如何停止有一个或多个实例处于可用状态的集群或启动已停止的集群。

停止或启动 Amazon DocumentDB 集群

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。

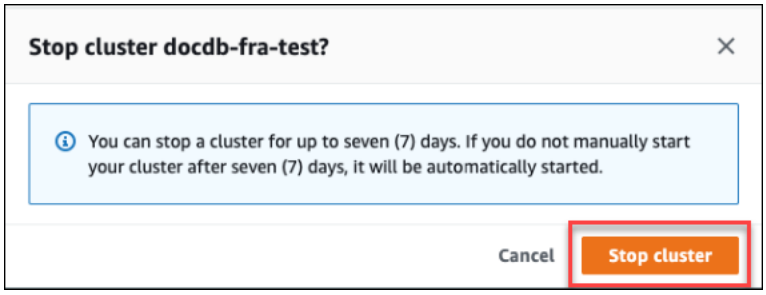
> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在集群列表中，选择要停止或启动的集群名称左侧的按钮。
4. 选择操作，然后选择您要在集群上执行的操作。
   - 如果您要停止集群且集群可用，则：
     a. 选择停止。

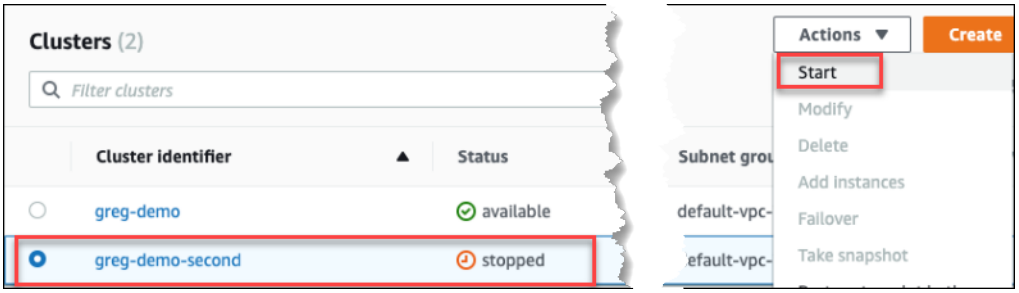


为避免激活故障转移机制，停止操作先停止副本实例，然后停止主实例。

     b. 在确认对话框中，确认您要通过选择 Stop cluster (停止集群) 停止该集群；或者要保持集群运行，则选择取消。



- 如果您要启动集群且集群处于停止状态，则选择启动。



5. 监控集群的状态及其实例。如果您启动集群，则当集群及其实例处于可用时，您可以继续使用集群。有关更多信息，请参阅 确定集群的状态。



Using the AWS CLI

以下代码示例显示如何停止有一个或多个实例处于可用状态的集群或启动已停止的集群。

要使用停止具有一个或多个可用实例的集群 AWS CLI，请使用`stop-db-cluster`操作。要启动已停止的集群，请使用 `start-db-cluster` 操作。这两个操作都使用 `--db-cluster-identifier` 参数。

参数：

- **`--db-cluster-identifier`**：必填项。要停止或启动的集群的名称。

Example— 要停止集群，请使用 AWS CLI

以下代码停止集群 `sample-cluster`。集群必须有一个或多个实例处于可用状态。

对于 Linux、macOS 或 Unix：

```
aws docdb stop-db-cluster \
   --db-cluster-identifier sample-cluster
```

对于 Windows：

```
aws docdb stop-db-cluster ^
   --db-cluster-identifier sample-cluster
```

Example— 要启动集群，请使用 AWS CLI

下面的代码启动集群 `sample-cluster`。集群当前必须处于停止状态。

对于 Linux、macOS 或 Unix：

```
aws docdb start-db-cluster \
   --db-cluster-identifier sample-cluster
```

对于 Windows：

```
aws docdb start-db-cluster ^
   --db-cluster-identifier sample-cluster
```

可以在已停止的集群上执行的操作

在停止 Amazon DocumentDB 集群后，您可以执行时间点还原以还原到指定的自动备份保留时段中的任意时间点。有关执行时间点还原的详细信息，请参阅还原到某个时间点。

在停止 Amazon DocumentDB 集群后，您无法修改该集群或其任何实例的配置。您也无法在该集群中添加或删除实例，或者，如果仍具有任何关联的实例，则无法删除该集群。您必须在执行任何此类管理操作之前启动该集群。

仅当再次启动后， Amazon DocumentDB 才会将任何计划的维护应用于停止的集群。七天后，Amazon DocumentDB 自动启动停止的集群，以使其维护状态不会落后太多。当集群重新启动后，您将再次开始为集群中的实例付费。

集群停止后，Amazon DocumentDB 不会执行任何自动备份，也不会延长备份留存期。

删除 Amazon DocumentDB 集群

您可以使用 AWS 管理控制台 或删除 Amazon DocumentDB 集群。 AWS CLI要删除集群，集群必须处于可用 状态，且不得有任何与其关联的实例。如果集群已停止，则首先启动集群，等待集群进入可用状态，然后删除集群。有关更多信息，请参阅 停止和启动 Amazon DocumentDB 集群。

删除保护

为了防止您的集群遭到意外删除，您可以启用删除保护。在使用控制台创建集群时，将默认启用删除保护。但是，如果您使用 AWS CLI创建集群，将默认禁用删除保护。

Amazon DocumentDB 为集群实施删除保护，不论您是使用控制台还是 AWS CLI来执行删除操作。如果已启用删除保护，则无法删除集群。要删除启用了删除保护的实例，首先请修改该集群并禁用删除保护。

当在集群上启用数据保护的情况下使用控制台时，无法删除该集群的最后一个实例，因为这样也会删除集群。您可以使用 AWS CLI删除受删除保护的集群的最后一个实例。但是，集群本身仍然存在，您的数据将被保留。您可以通过为集群创建新实例来访问数据。有关启用和禁用删除保护的更多信息，请参阅：

- 创建 Amazon DocumentDB 集群
- 修改 Amazon DocumentDB 集群

Using the AWS 管理控制台

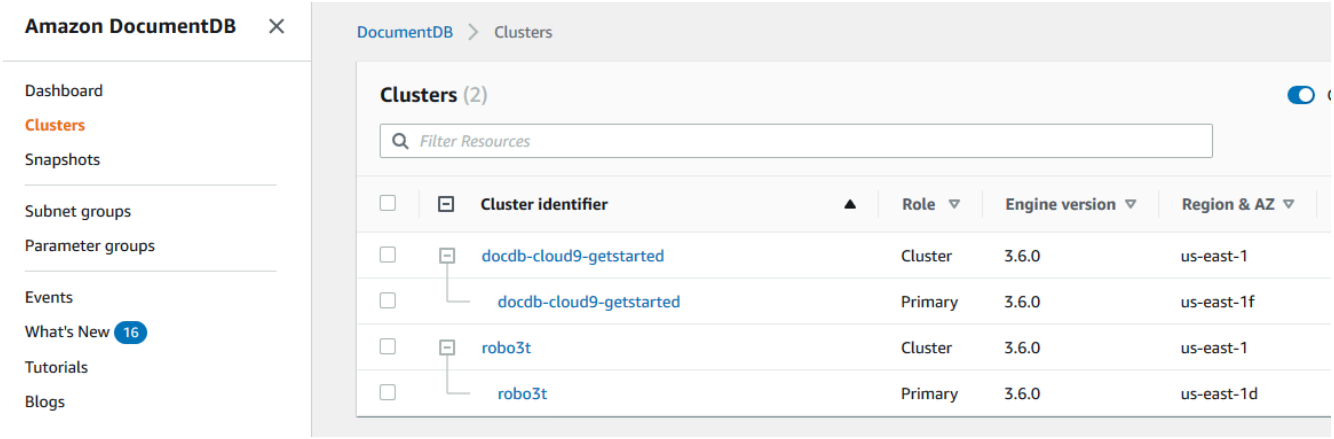
要使用删除集群 AWS 管理控制台，必须禁用删除保护。

确定集群是否已启用删除保护：

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 请注意，在集群导航框中，集群标识符列既显示集群又显示实例。实例列在集群下方，类似于以下屏幕截图。



4. 选择集群的名称，然后选择 Configuration (配置) 选项卡。在 Cluster details (集群详细信息) 部分中，找到 Deletion protection (删除保护)。如果已启用删除保护，则修改集群以禁用删除保护。有关修改集群的信息，请参阅 修改 Amazon DocumentDB 集群。

   禁用删除保护后，您就可以删除该集群了。

删除集群：

1. 在导航窗格中，选择集群。
2. 通过检查集群标识符列下方是否列有实例，来确定集群是否具有任何实例。在删除集群之前，必须先删除其所有实例。有关更多信息，请参阅 删除 Amazon DocumentDB 实例。
3. 根据您的集群是否有任何实例，请执行以下步骤之一。
   - 如果集群没有实例，请选择集群名称左侧的按钮，然后选择 Actions (操作)。从下拉菜单中，选择删除。填写删除 <集群名称> 对话框，然后选择删除。
   - 如果集群具有一个或多个实例，请执行以下操作：
     a. 在导航窗格中，选择集群。
     b. 通过选中集群名称左侧的复选框来删除集群的每个实例。选择 Actions (操作)，然后选择 Delete (删除)。填写删除 <集群名称> 对话框，然后选择删除。

        当您删除最后一个实例时，集群也将被删除。有关删除实例的更多信息，请参阅 删除 Amazon DocumentDB 实例。

删除集群需要几分钟时间。要监控集群的状态，请参阅监控 Amazon DocumentDB 集群的状态。

Using the AWS CLI

您不能删除具有任何关联实例的集群。要确定哪些实例与您的集群关联，请运行 `describe-db-clusters` 命令并删除集群的所有实例。然后，在需要时禁用集群的删除保护，最后删除集群。

1. 首先，删除集群的所有实例。

   要确定需要删除的实例，请运行以下命令。

   ```
   aws docdb describe-db-clusters \
       --db-cluster-identifier sample-cluster \
       --query 'DBClusters[*].
   [DBClusterIdentifier,DBClusterMembers[*].DBInstanceIdentifier]'
   ```

   此操作的输出将类似于下文（JSON 格式）。

   ```
   [
       [
           "sample-cluster",
           [
               "sample-instance-1",
               "sample-instance-2"
           ]
       ]
   ]
   ```

   如果要删除的集群包含任何实例，请如下所示删除它们。

   ```
   aws docdb delete-db-instance \
       --db-instance-identifier sample-instance
   ```

2. 接下来，禁用删除保护。

   使用删除 AWS CLI 集群的所有实例并不能删除集群。您还必须删除集群，但仅当删除保护处于禁用状态时才可以执行此操作。

   要确定集群是否已启用删除保护，请运行以下命令。

> **Tip**
>
> 要查看所有 Amazon DocumentDB 集群的删除保护状态，请省略 `--db-cluster-identifier` 参数。

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].[DBClusterIdentifier,DeletionProtection]'
```

此操作的输出将类似于下文。

```
[
    [
        "sample-cluster",
        "true"
    ]
]
```

如果集群已启用删除保护，则修改集群并禁用删除保护。要对集群禁用删除保护，请运行以下命令。

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --no-deletion-protection \
    --apply-immediately
```

3. 最后，删除集群。

禁用删除保护后，您就可以删除该集群了。要删除集群，请使用带以下参数的 `delete-db-cluster` 操作。

- **`--db-cluster-identifier`**：必填项。要删除的集群的标识符。
- **`--final-db-snapshot-identifier`**：可选。如果您需要最终快照，则必须包含该参数和最终快照的名称。必须包含 `--final-db-snapshot-identifier` 或 `--skip-final-snapshot`。

命名约束：

- 长度为 [1-63] 个字母、数字或连字符。
- 第一个字符必须是字母。
- 不能以连字符结尾或包含两个连续的连字符。
- 每个区域的 Amazon RDS、Amazon Neptune 和 Amazon DocumentDB 中的所有集群都必须是唯一 AWS 账户的。

- **--skip-final-snapshot**：可选。仅当您不想在删除集群之前拍摄最终快照时，才使用此参数。默认设置是拍摄最终快照。必须包含 `--final-db-snapshot-identifier` 或 `--skip-final-snapshot`。

以下 AWS CLI 代码删除`sample-cluster`带有最终快照的集群。如果有任何实例与集群关联，或者如果已启用删除保护，则此操作将失败。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --final-db-snapshot-identifier sample-cluster-final-snapshot
```

对于 Windows：

```
aws docdb delete-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --final-db-snapshot-identifier sample-cluster-final-snapshot
```

Example

以下 AWS CLI 代码在不拍摄最终快照`sample-cluster`的情况下删除集群。

对于 Linux、macOS 或 Unix：

```
aws docdb delete-db-cluster \
    --db-cluster-identifier sample-cluster \
    --skip-final-snapshot
```

对于 Windows：

```
aws docdb delete-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --skip-final-snapshot
```

`delete-db-cluster` 操作的输出是您要删除的集群。

删除集群需要几分钟时间。要监控集群的状态，请参阅监控集群的状态。

扩展 Amazon DocumentDB 集群

Amazon DocumentDB 让您能够根据需求扩展集群中的存储和计算。本节介绍了如何使用存储扩展、实例扩展和读取扩展来管理 Amazon DocumentDB 集群和实例的性能和扩展。

主题

- 存储扩展
- 实例扩展
- 读取扩展
- 写入扩展

存储扩展

Amazon DocumentDB 存储自动使用您的集群卷中的数据进行扩展。当数据量增多时，您的集群卷存储将以 10 GiB 增量递增，最多达 128 TiB。

实例扩展

您可通过修改集群中每个实例的实例类来按需扩展 Amazon DocumentDB 集群。Amazon DocumentDB 支持多个针对 Amazon DocumentDB 进行优化的实例类。

有关更多信息，请参阅 修改 Amazon DocumentDB 实例。

读取扩展

您可以通过在集群中创建最多 15 个 Amazon DocumentDB 副本来实现 Amazon DocumentDB 集群的读取扩展。每个 Amazon DocumentDB 副本从集群卷返回相同的数据，且副本滞后时间最短：通常少于主实例写入更新后的 100 毫秒。当读取流量增大时，可创建额外的 Amazon DocumentDB 副本并直

接连接到这些副本，以便为您的数据库集群分配读取负载。Amazon DocumentDB 副本不必具有与主实例相同的实例类。

有关更多信息，请参阅 向集群添加 Amazon DocumentDB 实例。

要对 Amazon DocumentDB 进行读取扩展，我们建议您以副本集形式连接到集群，并使用驱动程序的内置读取首选项功能将读取操作分布到副本实例。有关更多信息，请参阅 作为副本集连接到 Amazon DocumentDB。

写入扩展

您可以通过增加 Amazon DocumentDB 集群的主实例的大小来扩展该集群上的写入容量。此部分提供了两种方法来根据您的需求扩展集群的主实例。第一种方法旨在最大程度地减小应用程序影响，但需要执行更多步骤才能完成。第二种方法经过了优化，需要的步骤更少，因此更简单，但它会对应用程序产生更多的潜在影响，这需要您做出权衡。

根据您的应用程序，可以从下面选择最适合您的方法。有关可用实例大小和成本的更多信息，请参阅 Amazon DocumentDB 定价页面。

1. 优化以实现高可用性和性能：如果您在副本集模式（推荐）下连接到集群，则可以使用以下过程将扩展主实例时对应用程序产生的影响减至最小。此方法可以最大程度地减小影响，因为它可确保集群的可用性保持一个较高的水平甚至更高，并且读取扩展目标将作为实例添加到集群中，而不是进行就地更新。

   a. 将较大的实例类型的一个或多个副本添加到集群中（请参阅???）。我们建议所有副本具有与主实例相同的实例类型甚至更大的实例类型。这可避免因故障转移到较小的实例类型而意外降低写入性能。对于大多数客户而言，这意味着暂时将其集群中的实例数增加一倍，然后在扩展完成后删除较小的副本。

   b. 将所有新副本上的故障转移层设置为优先级零，并确保较小实例类型的副本具有最高的故障转移优先级。有关更多信息，请参阅 ???。

   c. 启动手动故障转移，这会将其中一个新副本提示为主实例。有关更多信息，请参阅 ???。

      > **Note**
      >
      > 这将导致您的集群停机约 30 秒。相应地做好计划。

   d. 从集群中删除小于新主实例的实例类型的所有副本。

   e. 将所有实例的故障转移层设置回相同的优先级（通常，这意味着将其设置回 1）。

例如，假设您的集群当前包含三个 r5.large 实例（一个主实例和两个副本实例），并且您希望扩展到一个 r5.xlarge 实例类型。为此，您首先将三个 r5.xlarge 副本实例添加到集群中，然后将新 r5.xlarge 副本的故障转移层设置为零。接下来，您将启动手动故障转移（这意味着您的应用程序将停机约 30 秒）。在故障转移完成后，您将从集群中删除所有三个 r5.large 实例，并让集群扩展到 r5.xlarge 实例。

为了帮助优化成本，Amazon DocumentDB 实例以一秒为增量计费，最低收取 10 分钟的费用，然后提供可计费状态更改（例如创建、修改或删除实例）。有关更多信息，请参阅最佳实践文档中的成本优化。

2. 优化以实现简易性：此方法针对简易性进行了优化。它不会扩展和收缩集群，但可能会暂时减少您的读取容量。

   更改副本的实例类可能将会导致该实例在短时间内（从几秒钟到少于 30 秒）无法处理请求。如果您在副本集模式（推荐）下连接到集群，这将在扩展操作期间减少一个副本的读取容量（例如，在 3 节点集群中减少到 66% 的容量，或在 4 节点集群中减少到 75% 的容量等）。

   a. 扩展集群中的副本实例之一。有关更多信息，请参阅 管理实例类。

   b. 等待实例变为可用（请参阅 监控 Amazon DocumentDB 实例的状态）。

      > **Note**
      >
      > 这将导致您的集群停机约 30 秒。相应地做好计划。

   c. 继续执行步骤 1 和 2，直到所有副本实例都逐一实现扩展。

   d. 启动手动失效转移。这会将其中一个副本提升为主实例。有关更多信息，请参阅 Amazon DocumentDB 失效转移。

      > **Note**
      >
      > 这将导致您的集群最多停机 30 秒，但所需的时间通常要更短。相应地做好计划。

   e. 扩展以前的主（现在是副本）实例。

克隆 Amazon DocumentDB 集群卷

通过使用 Amazon DocumentDB 克隆功能，您可以创建一个新集群，该集群使用相同的 Amazon DocumentDB 集群卷并具有与原始集群卷相同的数据。该过程旨在快速且经济高效。我们将新集群及其关联的数据卷称为克隆。与使用其他技术（如还原快照）实际复制数据相比，创建克隆速度更快且空间利用效率更高。

Amazon DocumentDB 支持从预配置的 Amazon DocumentDB 集群创建 Amazon DocumentDB 预配置克隆。使用与源不同的部署配置创建克隆时，源的 Amazon DocumentDB 引擎的最新版本将被用于创建克隆。

当您从 Amazon DocumentDB 集群创建克隆时，将在您的账户（即拥有源 Amazon DocumentDB 集群的同一个账户）中 AWS 创建克隆。

主题

- Amazon DocumentDB 克隆概述
- Amazon DocumentDB 克隆的限制
- Amazon DocumentDB 克隆的工作原理
- 创建 Amazon DocumentDB 克隆

Amazon DocumentDB 克隆概述

Amazon DocumentDB 使用写入时复制协议创建克隆。此机制占用最少的额外空间来创建初始克隆。首次创建克隆时，Amazon DocumentDB 会保留源数据库集群和新（克隆的）Amazon DocumentDB 集群使用的数据的单个副本。只有当源 Amazon DocumentDB 集群或 Amazon DocumentDB 集群克隆对数据（在 Amazon DocumentDB 存储卷上）进行更改时，才会分配额外的存储空间。如需了解有关写入时复制协议的更多信息，请参阅 Amazon DocumentDB 克隆的工作原理。

Amazon DocumentDB 克隆非常适合使用您的生产数据快速设置测试环境，且不会有损坏数据的风险。您可以将克隆用于多种类型的应用程序，例如：

- 对潜在的变化（例如模式变化和参数组变化）进行试验，以评估所有影响。
- 执行工作负载密集型操作，例如导出数据或在克隆上运行分析查询。
- 为开发、测试或其他用途创建生产数据库集群的副本。

您可以从同一个 Amazon DocumentDB 集群创建多个克隆。您还可以从另一个克隆创建多个克隆。

创建 Amazon DocumentDB 克隆后，您可以对 Amazon DocumentDB 实例实现与源 Amazon DocumentDB 集群不同的配置。例如，您可能不需要用于开发目的的克隆来满足与源生产 Amazon

DocumentDB 集群相同的高可用性要求。在这种情况下，您可以使用单个 Amazon DocumentDB 实例来配置克隆，而不是使用 Amazon DocumentDB 集群使用的多个数据库实例。

当克隆完成测试、开发等使用目的时，您可以将其删除。

Amazon DocumentDB 克隆的限制

Amazon DocumentDB 克隆目前具有以下限制：

- 您可以根据需要创建任意数量的克隆，最多为 AWS 区域中允许的最大数据库集群数。但是，在创建 15 个克隆后，下一个克隆是完整副本。克隆操作的作用类似于时间点恢复。
- 您无法在与源 Amazon DocumentDB 集群不同的 AWS 区域创建克隆。
- 您无法从没有数据库实例的 Amazon DocumentDB 集群创建克隆。您只能克隆具有至少一个数据库实例的 Amazon DocumentDB 集群。
- 您可以在与 Amazon DocumentDB 集群不同的虚拟私有云（VPC）中创建克隆。不过，这些 VPC 的子网必须映射到相同的可用区。

Amazon DocumentDB 克隆的工作原理

Amazon DocumentDB 克隆运行于 Amazon DocumentDB 集群的存储层。它使用写入时复制协议，该协议在支持 Amazon DocumentDB 存储卷的底层持久介质方面既快速又节省空间。您可以在 管理 Amazon DocumentDB 集群 中了解有关 Amazon DocumentDB 集群卷的更多信息。

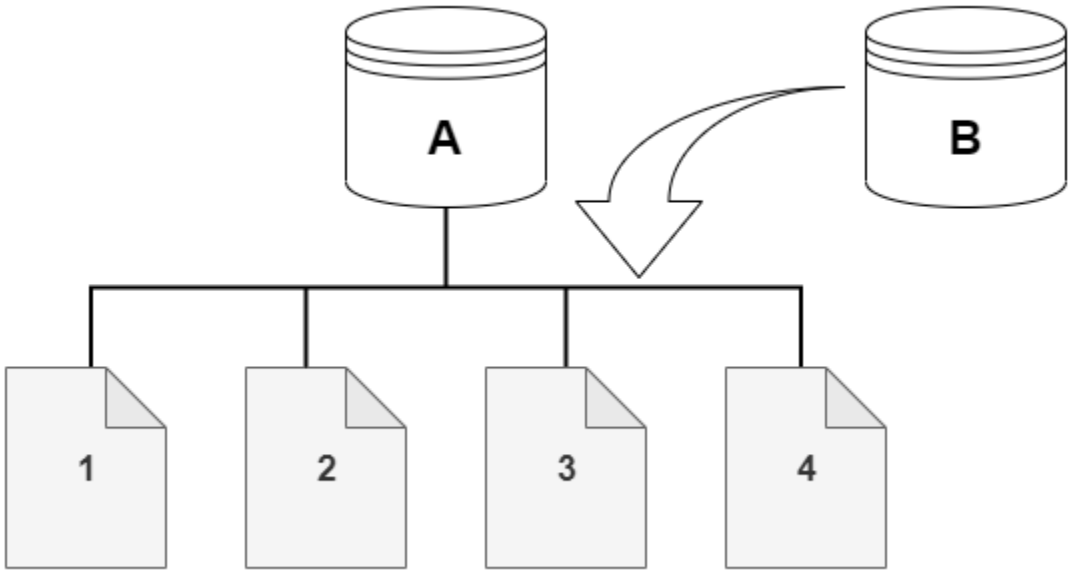
主题

- 了解写入时复制协议
- 删除源集群卷

了解写入时复制协议

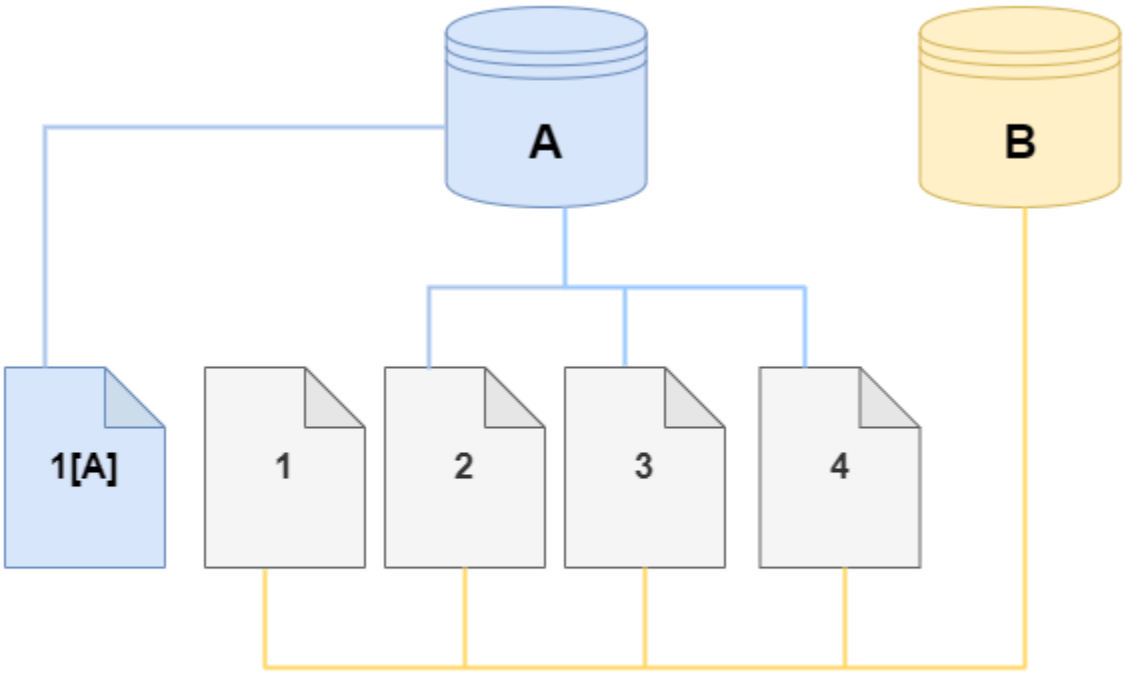
Amazon DocumentDB 集群将数据存储在底层 Amazon DocumentDB 存储卷的页面中。

例如，在下图中，您可以找到拥有四个数据页（1、2、3 和 4）的 Amazon DocumentDB 集群（A）。假设从 Amazon DocumentDB 集群创建了一个克隆 B。创建克隆时，未复制任何数据。相反，克隆指向与源 Amazon DocumentDB 集群相同的页面集。

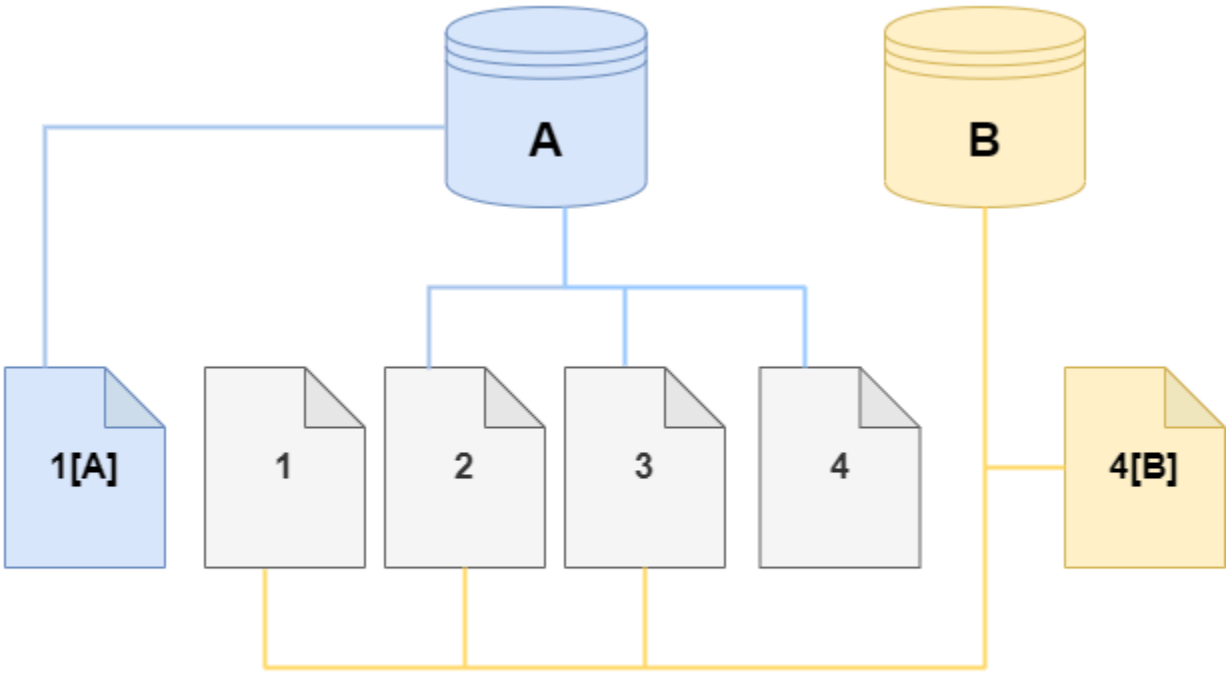


创建克隆时，通常不需要额外的存储空间。写入时复制协议在物理存储介质上使用与源段相同的数据段。只有当源段的容量不足以容纳整个克隆段时，才需要额外的存储空间。如果是这种情况，源段将被复制到另一个物理设备。

在下图中，您可以找到使用相同集群 A 及其克隆 B 的写入操作时复制协议的示例，如前所示。如果您对 Amazon DocumentDB 集群（A）进行更改，那么第 1 页上保存的数据也将随之而发生改变。Amazon DocumentDB 没有写入原始页面 1，而是创建了一个新页面 1[A]。集群（A）的 Amazon DocumentDB 集群卷现在指向页面 1[A]、2、3 和 4，而克隆（B）仍引用原始页面。



在克隆上，对存储卷的第 4 页进行了更改。Amazon DocumentDB 没有写入原始页面 4，而是创建了一个新页面 4[B]。克隆现在指向页面 1、2、3 和页面 4[B]，而集群（A）继续指向 1[A]、2、3 和 4。



随着时间推移，当源 Amazon DocumentDB 集群卷和克隆上出现了更多更改时，因此需要更多存储空间来捕获和存储更改。

删除源集群卷

删除与一个或多个克隆关联的源集群卷时，克隆不会受到影响。克隆继续指向以前由源集群卷拥有的页面。

创建 Amazon DocumentDB 克隆

您可以在与源 Amazon DocumentDB 集群相同的 AWS 账户中创建克隆。为此，您可以使用 AWS 管理控制台 或 AWS CLI 以及以下步骤。

通过使用 Amazon DocumentDB 克隆，您可以从预配置的 Amazon DocumentDB 集群中创建预配置的 Amazon DocumentDB 集群克隆。

Using the AWS 管理控制台

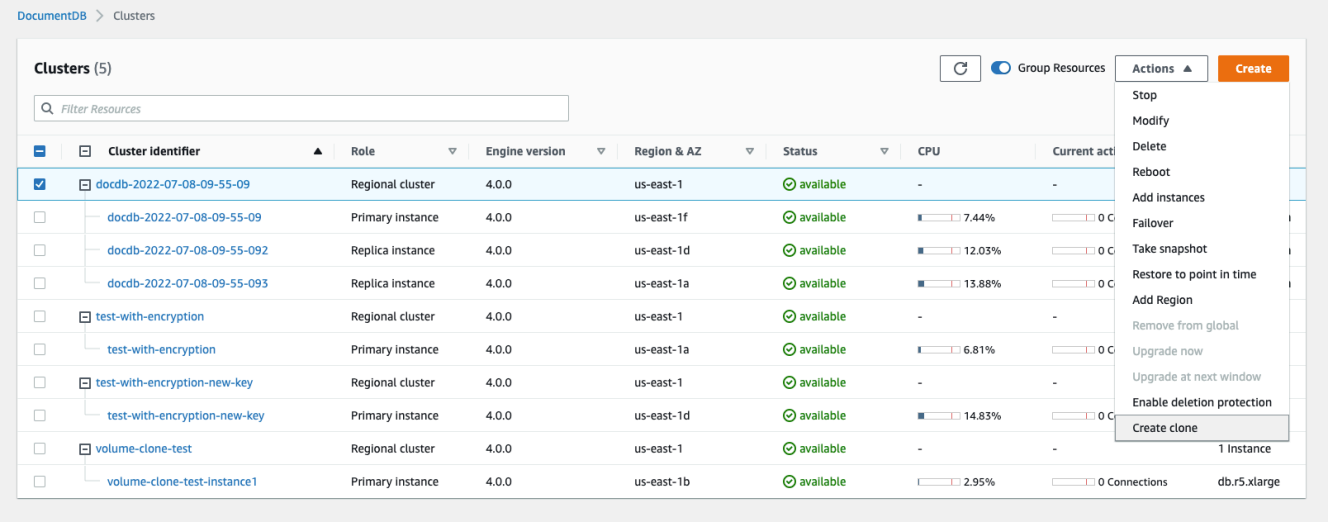
以下过程介绍了如何使用 AWS 管理控制台克隆 Amazon DocumentDB 集群。

使用带有一个 Amazon DocumentDB 实例的 Amazon DocumentDB 集群中的 AWS 管理控制台 结果创建克隆。

这些说明适用于创建克隆的同一个 AWS 账户所拥有的数据库集群。数据库集群必须归同一个 AWS 账户所有，因为 Amazon DocumentDB 不支持跨账户克隆。

创建您拥有的数据库集群的克隆 AWS 账户使用 AWS 管理控制台

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。
3. 从列表中选择您的 Amazon DocumentDB 集群，从操作中选择创建克隆。

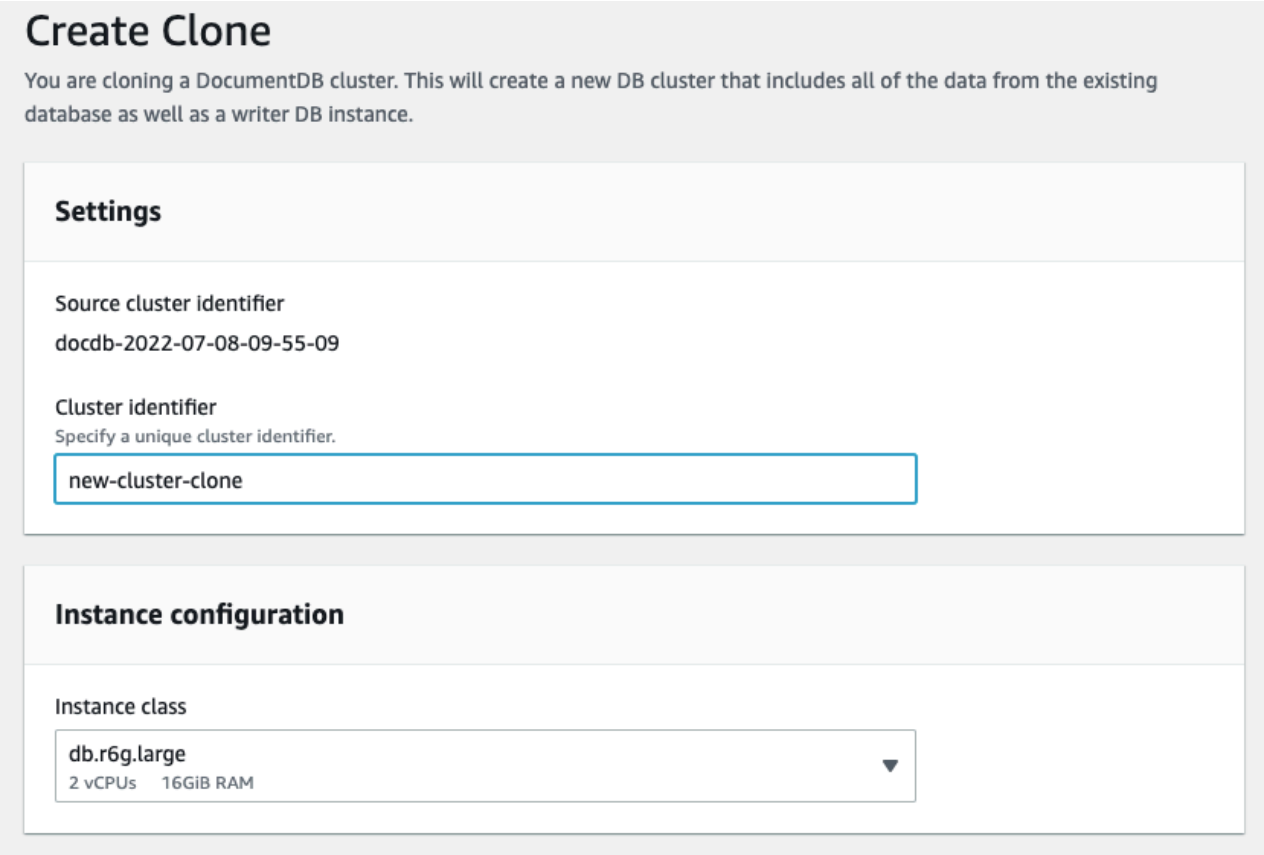


“创建克隆”页面打开后，您可以配置 Amazon DocumentDB 集群克隆的集群标识符和实例类以及其他选项。

4. 在设置部分，执行以下操作：

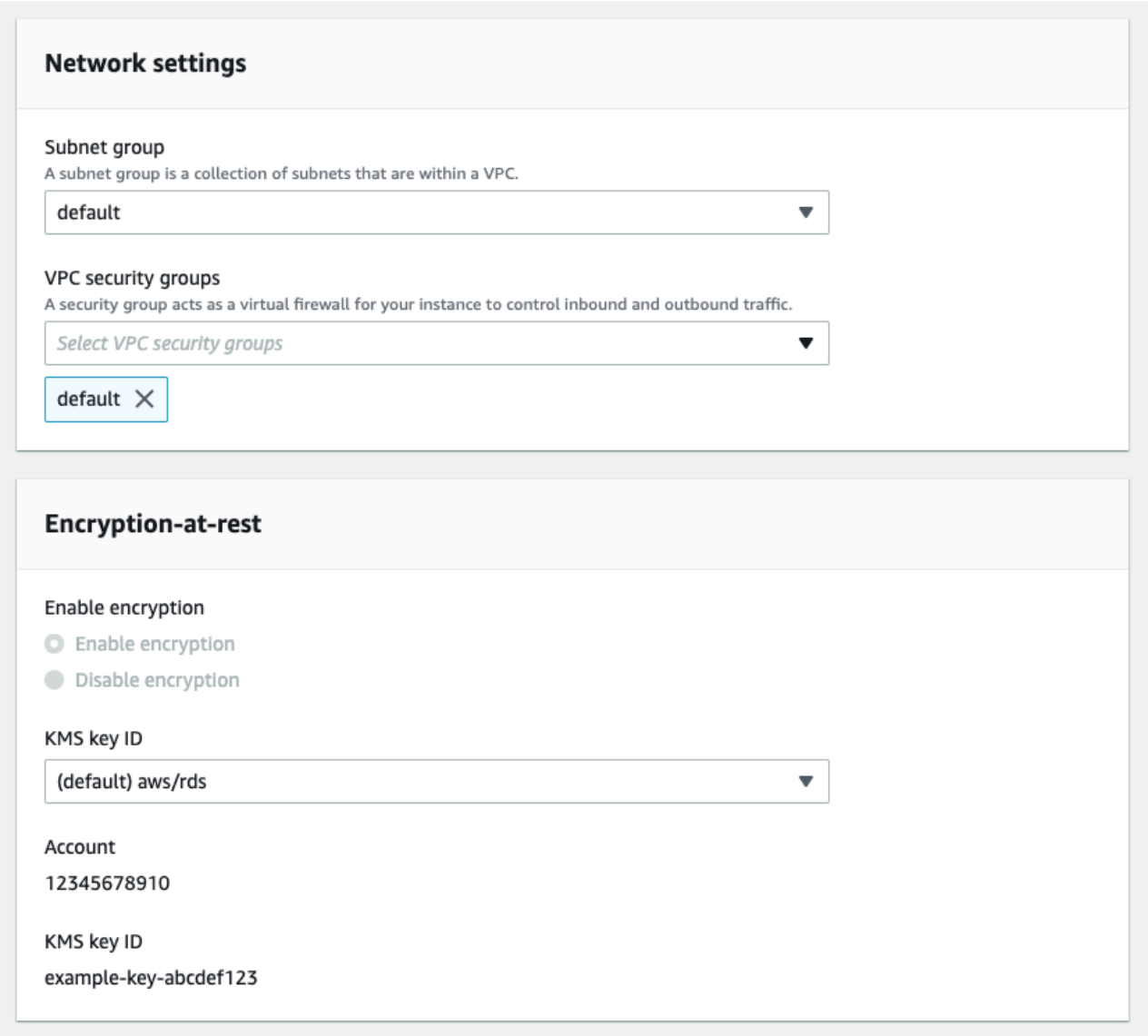
   a. 在集群标识符中，请输入您要为克隆的 Amazon DocumentDB 集群拟定的名称。

   b. 对于实例配置，请为克隆的 Amazon DocumentDB 集群选择适当的实例类。

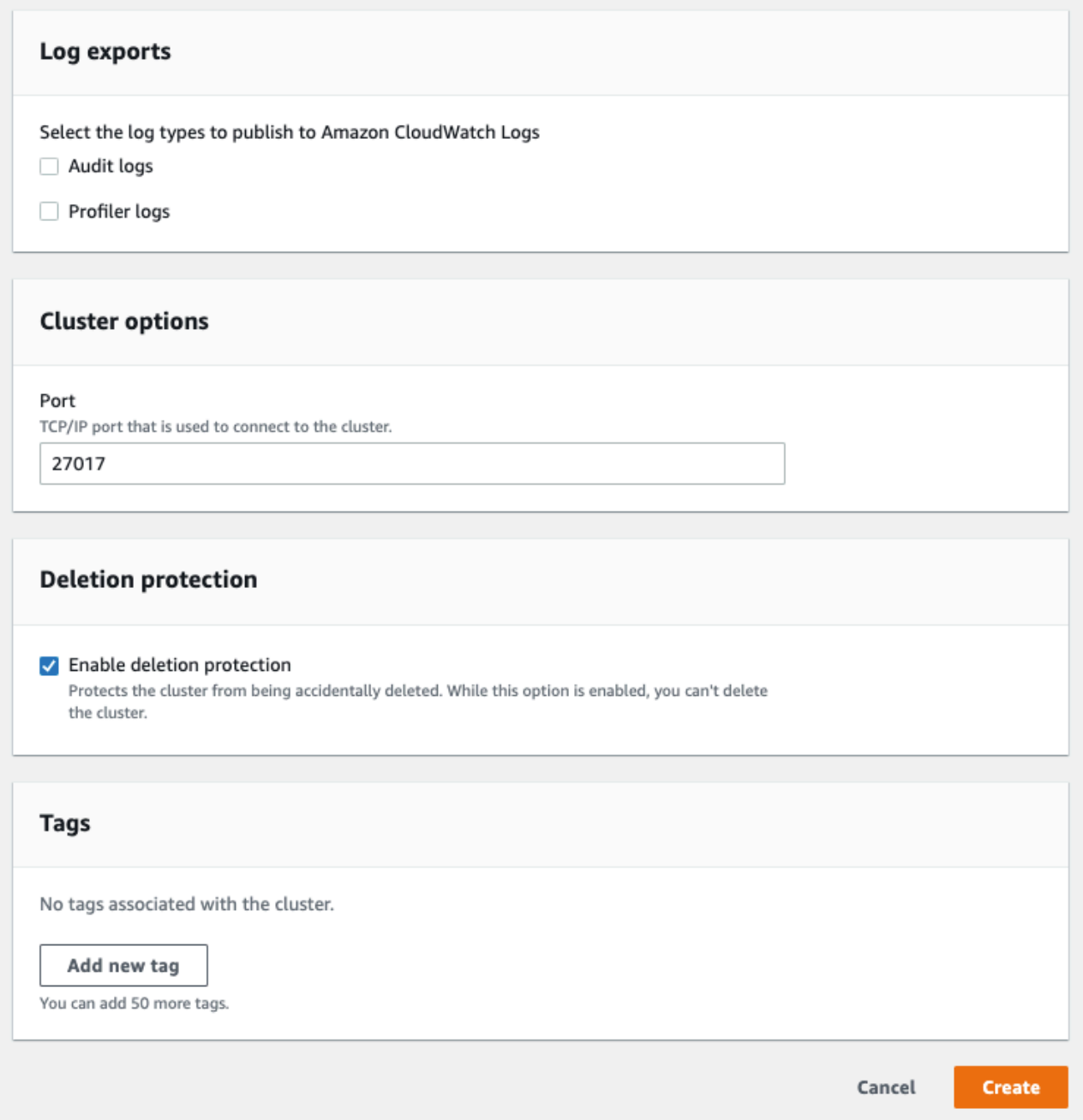


c. 对于网络设置，请选择使用案例的子网组和关联的 VPC 安全组。

d. 因为 Encryption-at-rest，如果源集群（正在克隆的集群）启用了加密，则克隆的集群还必须启用加密。如果这种情况适用，则启用加密选项将显示为灰色（禁用），但选中了启用加密选项。相反，如果源集群未启用加密，则启用加密选项可用，您可以选择启用或禁用加密。



e. 通过选择要导出的日志类型（可选）、输入用于连接到集群的特定端口、以及启用防止意外删除集群的保护（默认启用），完成新的集群克隆配置。



f. 完成 Amazon DocumentDB 集群克隆的所有设置。要了解有关 Amazon DocumentDB 集群和实例设置的更多信息，请参阅 管理 Amazon DocumentDB 集群。

5. 选择创建克隆以启动所选 Amazon DocumentDB 集群的 Amazon DocumentDB 克隆。

克隆创建完成后，它将会与您的其他 Amazon DocumentDB 集群一起列在控制台数据库部分，而且其当前状态也会一起显示。当其状态为可用时，您的克隆即可以使用。

Using the AWS CLI

使用克隆您 AWS CLI 的 Amazon DocumentDB 集群需要几个步骤。

您使用的`restore-db-cluster-to-point-in-time` AWS CLI 命令会生成一个空的 Amazon DocumentDB 集群，其中包含 0 个亚马逊 DocumentDB 实例。也就是说，此命令仅还原 Amazon DocumentDB 集群，而不还原该集群的数据库实例。在克隆可用之后，您可以单独执行此操作。该过程的两个步骤如下：

1. 使用 restore-db-cluster-to-point-in-time CLI 命令创建克隆。与此命令一起使用的参数控制正在创建的空 Amazon DocumentDB 集群（克隆）的容量类型和其他详细信息。
2. 使用 create-db-instance CLI 命令为克隆创建 Amazon DocumentDB 实例，进而在还原的 Amazon DocumentDB 集群中重新创建 Amazon DocumentDB 实例。

以下命令假设 AWS CLI 已将您的 AWS 区域设置为默认区域。这种方法使您无需在每个命令中传递 `--region` 名称。有关更多信息，请参阅配置 AWS CLI。您还可以在后面的每个 CLI 命令中指定 `--region`。

创建克隆

您传递给 `restore-db-cluster-to-point-in-time` CLI 命令的特定参数会有所不同。传递内容取决于您要创建的克隆类型。

使用以下过程从预配置的 Amazon DocumentDB 集群中创建预配置的 Amazon DocumentDB 克隆。

创建与源 Amazon DocumentDB 集群具有相同引擎模式的克隆

- 使用 `restore-db-cluster-to-point-in-time` CLI 命令并指定以下参数的值：
  - `--db-cluster-identifier`：为克隆选择一个有意义的名称。使用 restore-db-cluster-to-point-in-time CLI 命令命名克隆。
  - `--restore-type`：使用 `copy-on-write` 创建源数据库集群的克隆。如果没有此参数，`restore-db-cluster-to-point-in-time` 将还原 Amazon DocumentDB 集群，而不会创建克隆。`restore-type` 的默认值为 `full-copy`。
  - `--source-db-cluster-identifier`：使用要克隆的源 Amazon DocumentDB 集群的名称。

- `--use-latest-restorable-time`：此值指向克隆的最新可还原卷数据。此参数是 `restore-type copy-on-write` 所必需的，但是，您不能将 `restore-to-time parameter` 与它一起使用。

以下示例从名为 `my-source-cluster` 的集群创建一个名为 `my-clone` 的克隆。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

该命令返回包含克隆详细信息的 JSON 对象。在尝试为您的克隆创建数据库实例之前，请检查以确保您的克隆数据库集群可用。有关更多信息，请参阅以下检查状态并获取克隆的详细信息：

检查状态并获取克隆的详细信息

您可以使用以下命令检查新创建的空数据库集群的状态。

```
$ aws docdb describe-db-clusters --db-cluster-identifier my-clone --query '*[].
[Status]' --output text
```

或者，您可以使用以下 AWS CLI 查询获取为克隆创建数据库实例所需的状态和其他值：

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters --db-cluster-identifier my-clone \
  --query '*[].{Status:Status,Engine:Engine,EngineVersion:EngineVersion}'
```

对于 Windows：

```
aws docdb describe-db-clusters --db-cluster-identifier my-clone ^
  --query "*[].{Status:Status,Engine:Engine,EngineVersion:EngineVersion}"
```

此查询返回类似于下述信息的输出：

```
[
  {
        "Status": "available",
        "Engine": "docdb",
        "EngineVersion": "4.0.0",
    }
]
```

为您的克隆创建 Amazon DocumentDB 实例

使用 create-db-instance CLI 命令为您的克隆创建数据库实例。

`--db-instance-class` 参数仅用于预配置的 Amazon DocumentDB 集群。

对于 Linux、macOS 或 Unix：

```
aws docdb create-db-instance \
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
    --db-instance-class  db.r5.4xlarge \
    --engine docdb
```

对于 Windows：

```
aws docdb create-db-instance ^
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class  db.r5.4xlarge ^
    --engine docdb
```

用于克隆的参数

下表总结了与 `restore-db-cluster-to-point-in-time` 一起用于克隆 Amazon DocumentDB 集群的各种参数。

参数	描述
--source-db-cluster-identifier	使用要克隆的源 Amazon DocumentDB 集群的名称。
--db-cluster-identifier	为克隆选择一个有意义的名称。您可以使用 `restore-db-cluster-to-point-in-time` 命令为您的克隆命名。然后将此名称传递给 `create-db-instance` 命令。
--restore-type	将 `copy-on-write` 指定为 `--restore-type` 以创建源数据库集群的克隆，而不是还原源 Amazon DocumentDB 集群。
--use-latest-restorable-time	此值指向克隆的最新可还原卷数据。

了解 Amazon DocumentDB 集群容错能力

Amazon DocumentDB 集群设计为具有容错能力。每个集群的卷跨越单个可用区 AWS 区域，每个可用区都包含集群卷数据的副本。此功能意味着您的集群可容忍可用区故障，而不发生任何数据丢失，只是会短暂中断服务。

如果集群中的主实例失败，Amazon DocumentDB 可通过两种方式之一来自动将失效转移到新的主实例：

- 通过将现有 Amazon DocumentDB 副本提升为根据每个副本的提升层设置所选择的新主实例，然后为以前的主实例创建一个替换项。失效转移到副本实例通常可在不到 30 秒的时间内完成。在此期间，读取和写入操作可能会出现短暂中断。要提高集群的可用性，建议您在两个或更多不同的可用区中创建至少一个或多个 Amazon DocumentDB 副本。
- 创建新的主实例。只有当您的集群中没有副本实例并且可能需要几分钟才能完成时，才会发生这种情况。

如果集群具有一个或多个 Amazon DocumentDB 副本，则 Amazon DocumentDB 副本将在故障事件期间被提升为主实例。故障事件将导致短暂中断，其间的读取和写入操作将失败并引发异常。不过，服务通常会在 120 秒内 (经常在 60 秒内) 还原。要提高集群的可用性，建议您在两个或更多不同的可用区中创建至少一个或多个 Amazon DocumentDB 副本。

您可以通过为每个副本分配一个优先级来自定义发生故障后将 Amazon DocumentDB 副本提升为主实例的顺序。优先级介于 0（最高优先级）和 15（最低优先级）之间。如果主实例失败，则将具有最高优先级的 Amazon DocumentDB 副本提升为新的主实例。您可以随时修改 Amazon DocumentDB 副本的优先级。修改优先级不会触发失效转移。您可以对 `modify-db-instance` 操作使用 `--promotion-tier` 参数。有关自定义实例的故障转移优先级的更多信息，请参阅Amazon DocumentDB 失效转移。

多个 Amazon DocumentDB 副本可同属一个优先级，这会产生提升层问题。如果两个或更多 Amazon DocumentDB 副本具有相同优先级，则最大的副本将被提升为主实例。如果两个或多个 Amazon DocumentDB 副本具有同一优先级和大小，那么将提升同一提升层中的任意副本。

如果集群不包含任何 Amazon DocumentDB 副本，则将在故障事件期间重新创建主实例。故障事件将导致中断，其间的读取和写入操作将失败并引发异常。创建新的主实例时将还原服务，该操作所需的时间通常在 10 分钟内。将 Amazon DocumentDB 副本提升为主实例要比创建新的主实例快得多。

管理 Amazon DocumentDB 实例

以下主题提供可帮助您管理 Amazon DocumentDB 实例的信息。其中包括有关实例类和状态的详细信息，以及如何创建、删除和修改实例。

主题

- 确定实例的状态
- Amazon DocumentDB 实例生命周期
- 管理实例类
- NVMe-backed 实例

确定实例的状态

要查看有效实例的状态、其含义以及如何确定您的实例的状态，请参阅监控 Amazon DocumentDB 实例的状态。

Amazon DocumentDB 实例生命周期

Amazon DocumentDB 实例的生命周期包括创建、修改、维护并升级、执行备份和还原、重启以及删除实例。本节提供有关如何完成这些过程的信息。

主题

- 向集群添加 Amazon DocumentDB 实例
- 描述 Amazon DocumentDB 实例
- 修改 Amazon DocumentDB 实例
- 重启 Amazon DocumentDB 实例
- 删除 Amazon DocumentDB 实例

您可以使用 AWS 管理控制台 或创建新的 Amazon DocumentDB 实例。 AWS CLI要将实例添加到集群，该集群必须处于可用 状态。您无法将实例添加到已停止的集群。如果集群已停止，则首先启动集群，等待集群进入可用 状态，然后添加实例。有关更多信息，请参阅 停止和启动 Amazon DocumentDB 集群。

> **Note**
>
> 如果您使用控制台创建 Amazon DocumentDB 集群，则同时会自动为您创建实例。如果您要创建其他实例，请使用以下过程之一。

向集群添加 Amazon DocumentDB 实例

Using the AWS 管理控制台

使用以下过程通过 Amazon DocumentDB 控制台为集群创建实例。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 要选择希望将实例添加到的集群，请选择该集群名称左侧的按钮。
4. 选择 Actions (操作)，然后选择 Add instance (添加实例)。
5. 在 Add instance to: (将实例添加到:)<cluster-name> 页面中，对要添加到集群的每个实例重复以下步骤。您最多可以有 15。

a. Instance identifier (实例标识符)：您可以输入此实例的唯一标识符，或允许 Amazon DocumentDB 基于集群标识符提供实例标识符。

   实例命名约束：

   - 长度为 [1-63] 个字母、数字或连字符。
   - 第一个字符必须是字母。
   - 不能以连字符结尾或包含两个连续的连字符。
   - 每个区域的 Amazon RDS、Neptune 和 Amazon DocumentDB 中的所有实例都必须是唯一 AWS 账户的。

b. Instance class (实例类)：从下拉列表中，为该实例选择所需的实例类型。

c. Promotion tier (提升层)：从下拉列表中，选择该实例的提升层，或者选择 No preference (无首选项) 以允许 Amazon DocumentDB 为您的实例设置提升层。数字越小，意味着优先级越高。有关更多信息，请参阅 控制失效转移目标。

d. 要添加更多实例，请选择 Add additional instances (添加额外实例)，并重复步骤 a、b 和 c。

6. 完成此操作。

   - 要将实例添加到集群，请选择 Create (创建)。
   - 要取消操作，请选择取消。

创建实例需要几分钟时间。您可以使用控制台或 AWS CLI 查看实例的状态。有关更多信息，请参阅 监控实例的状态。

Using the AWS CLI

使用带有以下参数的`create-db-instance` AWS CLI 操作为您的集群创建主实例。

- **`--db-instance-class`**：必需。实例的计算和内存容量，例如，`db.m4.large`。并非所有实例类在所有 AWS 区域中都可用。
- **`--db-instance-identifier`**：必需。标识实例的一个 字符串。

  实例命名约束：

  - 长度为 [1-63] 个字母、数字或连字符。
  - 第一个字符必须是字母。
  - 不能以连字符结尾或包含两个连续的连字符。

  - 每个区域的 Amazon RDS、Neptune 和 Amazon DocumentDB 中的所有实例都必须是唯一 AWS 账户的。
- **--engine**：必需。必须是 docdb。
- **--availability-zone**：可选。要在其中创建此实例的可用区。使用此参数在不同可用区中找到您的实例，以增强容错能力。有关更多信息，请参阅 Amazon DocumentDB 高可用性和复制。
- **--promotion-tier**：可选。此实例的失效转移优先级层。必须介于 0 与 15 之间，数字越小，优先级越高。有关更多信息，请参阅 控制失效转移目标。

1. 首先，确定您可以在哪些可用区中创建您的实例。

   如果要在创建实例前指定可用区，请运行以下命令以确定哪些可用区可用于您的 Amazon DocumentDB 集群。

   对于 Linux、macOS 或 Unix：

   ```
   aws docdb describe-db-clusters \
         --query 'DBClusters[*].[DBClusterIdentifier,AvailabilityZones[*]]'
   ```

   对于 Windows：

   ```
   aws docdb describe-db-clusters ^
         --query 'DBClusters[*].[DBClusterIdentifier,AvailabilityZones[*]]'
   ```

   此操作的输出将类似于下文。

   ```
   [
       [
           "sample-cluster",
           [
               "us-east-1c",
               "us-east-1b",
               "us-east-1a"
           ]
       ]
   ]
   ```

2. 其次，确定您可以在您的区域中创建哪些实例类。

要确定您所在区域中可用的实例类，请运行以下命令。从输出中，为要添加到 Amazon DocumentDB 集群的实例选择实例类。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-orderable-db-instance-options \
        --engine docdb \
        --query 'OrderableDBInstanceOptions[*].DBInstanceClass'
```

对于 Windows：

```
aws docdb describe-orderable-db-instance-options ^
        --engine docdb ^
        --query 'OrderableDBInstanceOptions[*].DBInstanceClass'
```

此操作的输出将类似于下文。

```
[
    "db.r5.16xlarge",
    "db.r5.2xlarge",
    "db.r5.4xlarge",
    "db.r5.8xlarge",
    "db.r5.large",
    "db.r5.xlarge"
]
```

3. 最后，将实例添加到您的 Amazon DocumentDB 集群。

要将实例添加到您的 Amazon DocumentDB 集群，请运行以下命令。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此操作的输出将类似于下文。

```
{
    "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"
            }
        ],
        "AavailabilityZone": "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"
    }
}
```

创建实例需要几分钟时间。您可以使用控制台或 AWS CLI 查看实例的状态。有关更多信息，请参阅 监控 Amazon DocumentDB 实例的状态。

描述 Amazon DocumentDB 实例

您可以使用 Amazon DocumentDB 管理控制台或 AWS CLI 查看详细信息，例如连接端点、安全组 VPC、证书颁发机构以及与您的 Amazon DocumentDB 实例相关的参数组。

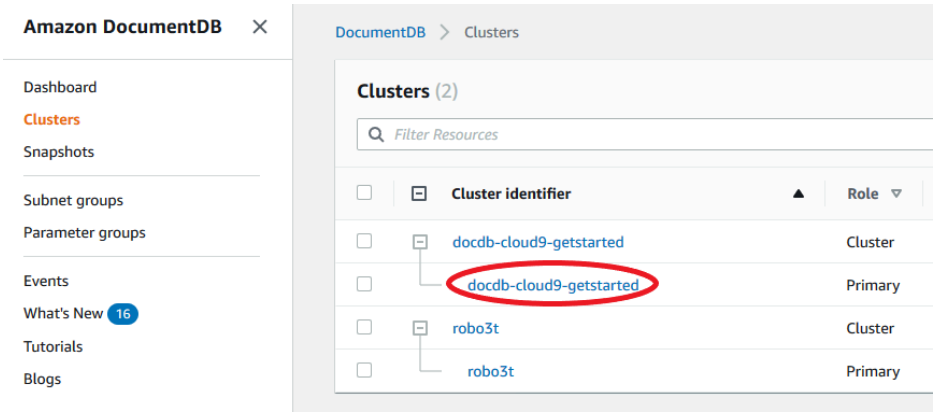
Using the AWS 管理控制台

要使用 AWS 管理控制台查看实例的详细信息，请执行以下步骤。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择 Clusters（集群）。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在集群导航框中，您将看到“集群标识符”列。您的实例列在集群下，类似于以下屏幕截图。



4. 在实例列表中，选择要查看其详细信息的实例的名称。有关实例的信息分为以下几组：

- Summary (摘要)：有关实例的一般信息，包括引擎版本、类、状态和所有待处理维护。
- Connectivity & Security (连接性和安全性)：Connect (连接) 部分列出要使用 mongo Shell 或应用程序连接到此实例的连接端点。Security Groups (安全组) 部分列出与此实例关联的安全组及其 VPC ID 和描述。
- 配置：详情部分列出了实例的配置和状态，包括实例的 Amazon 资源名称 (ARN)、端点、角色、类和证书颁发机构。其中还列出实例的安全性和网络设置以及备份信息。Cluster details (集群详细信息) 部分列出此实例所属集群的详细信息。Cluster instances (集群实例) 部分列出属于集群的所有实例，其中包括每个实例的角色和集群参数组状态。

> **Note**
>
> 您可以通过选择紧邻 集群详情标题的修改，修改与您的实例关联的集群。有关更多信息，请参阅 修改 Amazon DocumentDB 集群。

- 监控- CloudWatch 记录此实例的指标。有关更多信息，请参阅 使用以下方式监控亚马逊 DocumentDB CloudWatch。
- 事件和标签：最近事件部分列出了这个实例的最近事件。Amazon DocumentDB 记录与集群、实例、快照、安全组和集群参数组相关的事件。此信息包括与每个事件关联的日期、时

间和消息。Tags (标签) 部分列出附加该集群的标签。有关更多信息，请参阅 标记 Amazon DocumentDB 资源。

Using the AWS CLI

要使用查看您的 Amazon DocumentDB 实例的详细信息 AWS CLI，请使用以下示例所示的`describe-db-clusters`命令。有关更多信息，请参阅 Amazon DocumentDB 资源管理 API 参考中的 `DescribeDBInstances`。

> **Note**
>
> 对于某些管理功能（如集群和实例生命周期管理），Amazon DocumentDB 利用与 Amazon RDS 共享的操作技术。`filterName=engine,Values=docdb` 筛选条件参数仅返回 Amazon DocumentDB 集群。

1. 列出所有 Amazon DocumentDB 实例。

   以下 AWS CLI 代码列出了某个区域中所有 Amazon DocumentDB 实例的详细信息。

   对于 Linux、macOS 或 Unix：

   ```
   aws docdb describe-db-instances \
       --filter Name=engine,Values=docdb
   ```

   对于 Windows：

   ```
   aws docdb describe-db-instances \
       --filter Name=engine,Values=docdb
   ```

2. 列出指定 Amazon DocumentDB 实例的所有详细信息

   以下代码列出了 `sample-cluster-instance` 的详细信息。将 `--db-instance-identifier` 参数包含在实例名称中，会将输出限制为该特定实例的信息。

   对于 Linux、macOS 或 Unix：

   ```
   aws docdb describe-db-instances \
       --db-instance-identifier sample-cluster-instance
   ```

对于 Windows：

```
aws docdb describe-db-instances \
    --db-instance-identifier sample-cluster-instance
```

此操作的输出将类似于以下内容：

```
{
    "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"
```

```
        }
    ]
}
```

修改 Amazon DocumentDB 实例

您可以使用 AWS 管理控制台 或修改您的 Amazon DocumentDB 实例。 AWS CLI要修改某个实例，该实例必须处于可用 状态。您无法修改已停止的实例。如果集群已停止，则首先启动集群，等待实例进入可用 状态，然后进行所需修改。有关更多信息，请参阅 停止和启动 Amazon DocumentDB 集群。

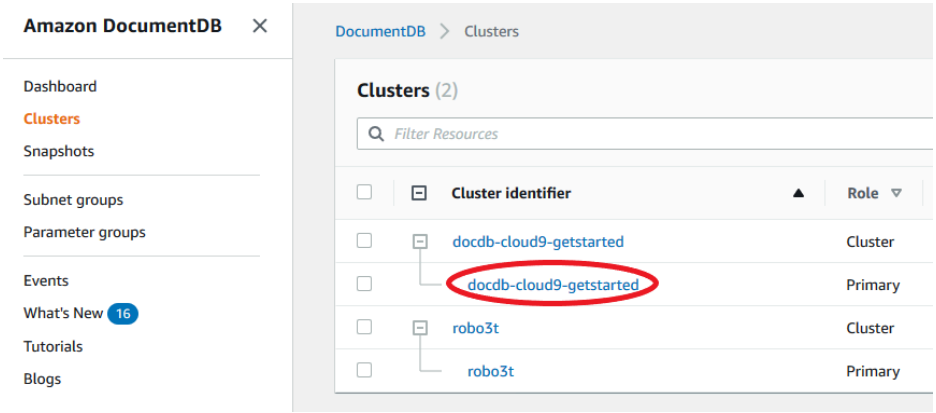
Using the AWS 管理控制台

要使用控制台修改特定的 Amazon DocumentDB 实例，请完成以下步骤。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择 Clusters（集群）。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标
> (≡)。

3. 在集群导航框中，您将看到“集群标识符”列。您的实例列在集群下，类似于以下屏幕截图。



4. 勾选您希望修改的实例左侧的复选框。
5. 选择 Actions (操作)，然后选择 Modify (修改)。

6. 在 Modify instance: <instance-name> (修改实例: <instance-name>>) 窗格中，进行所需更改。您可以进行以下更改：

   - 实例规范：实例标识符和类。实例标识符命名约束：
     - 实例标识符-输入一个名称，该名称对于您在 AWS 账户 当前地区拥有的所有实例都具有唯一性。实例标识符必须包含 [1-63] 个字母数字字符或连字符，以字母作为首字符，并且不能以连字符结尾或不能包含两个连续连字符。
     - 实例类：从下拉菜单中，为您的 Amazon DocumentDB 实例选择一个实例类。有关更多信息，请参阅 管理实例类。
   - 证书颁发机构：此实例的服务器证书。有关更多信息，请参阅 更新您的 Amazon DocumentDB TLS 证书。
   - 失效转移：在失效转移期间，具有最高提升层的实例将被提升为主实例。有关更多信息，请参阅 Amazon DocumentDB 失效转移。
   - 维护：在维护窗口中将待处理的修改或修补程序应用于集群中的实例。
7. 完成后，选择 Continue (继续) 以查看更改摘要。
8. 在确认您的更改后，您可以立即应用这些更改，也可以在 Scheduling of modifications (修改计划) 下的下一个维护时段内应用这些更改。选择 Modify instance (修改实例) 以保存更改。或者，您可以选择 Cancel (取消) 以放弃更改。

需要几分钟时间才能应用更改。只有在实例状态为 available (可用) 时，才能使用实例。您可以使用控制台或 AWS CLI监控实例状态。有关更多信息，请参阅 监控 Amazon DocumentDB 实例的状态。

Using the AWS CLI

要使用修改特定的 Amazon DocumentDB 实例 AWS CLI，请使用`modify-db-instance`带以下参数的。有关更多信息，请参阅 ModifyDBInstance。以下代码为实例 `sample-instance` 将实例类修改为 `db.r5.large`。

参数

- **`--db-instance-identifier`**：必需。要修改的实例的标识符。
- **`--db-instance-class`**：可选。实例的新计算和内存容量；例如，`db.r5.large`。并非所有实例类型在所有 AWS 区域中都可用。如果您修改实例类，则在更改期间会发生中断。更改在下一个维护时段内应用，除非此请求的 `ApplyImmediately` 指定为真实。
- **`--apply-immediately`** 或 **`--no-apply-immediately`**：可选。指定应立即应用此修改，还是等到下一个维护时段。如果省略此参数，则会在下一个维护时段执行修改。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb modify-db-instance \
       --db-instance-identifier sample-instance \
       --db-instance-class db.r5.large \
       --apply-immediately
```

对于 Windows：

```
aws docdb modify-db-instance ^
       --db-instance-identifier sample-instance ^
       --db-instance-class db.r5.large ^
       --apply-immediately
```

此操作的输出将类似于下文。

```
{
    "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"
            ]
        }
    ]
}
```

需要几分钟时间才能应用修改。只有在实例状态为 available (可用) 时，才能使用实例。您可以使用 AWS 管理控制台 或监控实例的状态 AWS CLI。有关更多信息，请参阅 监控 Amazon DocumentDB 实例的状态。

重启 Amazon DocumentDB 实例

有时您可能需要重启 Amazon DocumentDB 实例，这通常是出于维护原因。如果您进行了某些更改，例如更改与集群关联的集群参数组，则必须重启集群中的实例才能使更改生效。您可以使用 AWS 管理控制台 或重启指定的实例 AWS CLI。

重启实例会重新启动数据库引擎服务。重启将导致短暂中断，在此期间，实例的状态将设置为 `rebooting`。重新启动完成后，即会创建 Amazon DocumentDB 事件。

重启实例不会导致失效转移。要对 Amazon DocumentDB 集群进行故障转移，请使用 AWS 管理控制台 或操作。 AWS CLI `failover-db-cluster`有关更多信息，请参阅 Amazon DocumentDB 失效转移。

如果实例未处于 available (可用) 状态，则无法重启该实例。数据库可能会由于几个原因而不可用，例如，以前请求的修改或维护时段操作。有关实例状态的更多信息，请参阅监控 Amazon DocumentDB 实例的状态。

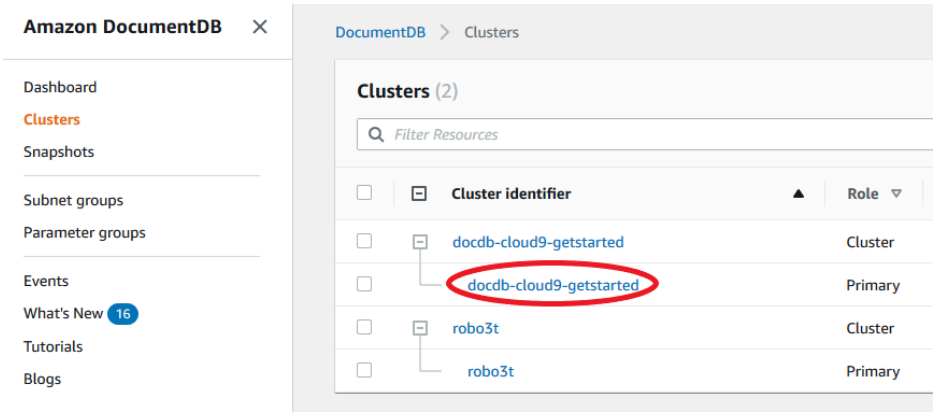
Using the AWS 管理控制台

以下过程将使用控制台重启您指定的实例。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择 Clusters（集群）。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在集群导航框中，您将看到“集群标识符”列。您的实例列在集群下，类似于以下屏幕截图。



4. 勾选您希望重启的实例左侧的复选框。
5. 选择 Actions (操作)、Reboot (重启)，然后选择 Reboot (重启) 以确认重启。

实例重启需要几分钟时间。只有在实例状态为 available (可用) 时，才能使用实例。您可以使用控制台或 AWS CLI监控实例状态。有关更多信息，请参阅 监控 Amazon DocumentDB 实例的状态。

Using the AWS CLI

要重启 Amazon DocumentDB 实例，请使用具有 `--db-instance-identifier` 参数的 `reboot-db-instance` 操作。此参数指定要重启的实例的标识符。

以下代码将重启实例 `sample-instance`。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb reboot-db-instance \
       --db-instance-identifier sample-instance
```

对于 Windows：

```
aws docdb reboot-db-instance ^
       --db-instance-identifier sample-instance
```

此操作的输出将类似于下文。

```
{
    "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"
        ]
    }
}
```

实例重启需要几分钟时间。只有在实例状态为 available (可用) 时，才能使用实例。您可以使用控制台或 AWS CLI监控实例状态。有关更多信息，请参阅 监控 Amazon DocumentDB 实例的状态。

删除 Amazon DocumentDB 实例

您可以使用 AWS 管理控制台 或删除您的 Amazon DocumentDB 实例。 AWS CLI要删除某个实例，该实例必须处于可用 状态。您无法删除已停止的实例。如果包含您的实例的 Amazon DocumentDB 集群已停止，则首先启动集群，等待实例进入可用状态，然后删除实例。有关更多信息，请参阅 停止和启动 Amazon DocumentDB 集群。

> **Note**
>
> Amazon DocumentDB 将所有数据都存储在集群卷中。即使您从集群中删除所有实例，数据仍会保留在该集群卷中。如果您需要再次访问数据，则可随时向该集群中添加实例，并在停止处恢复操作。

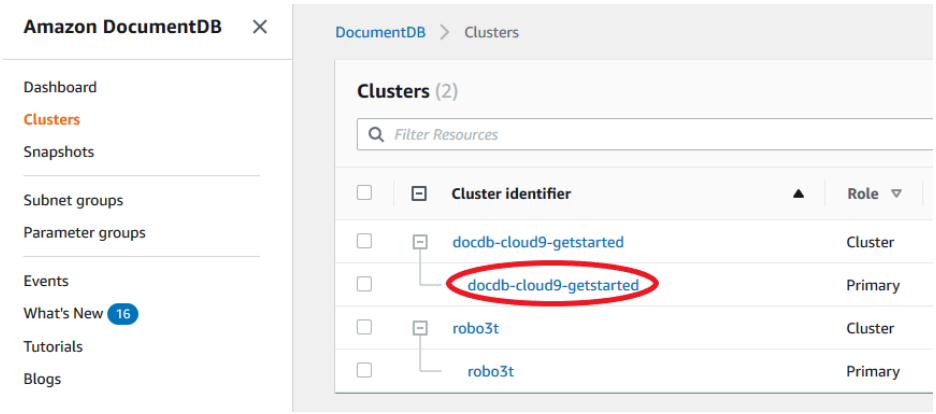
Using the AWS 管理控制台

以下过程使用控制台删除指定的 Amazon DocumentDB 实例。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择 Clusters（集群）。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在集群导航框中，您将看到“集群标识符”列。您的实例列在集群下，类似于以下屏幕截图。



4. 勾选您希望删除的实例左侧的复选框。
5. 选择 Actions (操作)，然后选择 Delete (删除)。

   1. 如果您要删除集群中的最后一个实例：

      - Create final cluster snapshot? (是否创建最终集群快照?) — 如果要在删除集群之前创建最终快照，请选择 Yes (是)。否则，请选择 No。
      - Final snapshot name (最终快照名称)：如果您选择创建最终快照，请输入所创建的新集群快照的集群快照标识符。
      - Delete <instance-name> instance? (是否删除 <instance-name> 实例?) — 在此字段中输入 delete entire cluster (删除整个集群) 短语以确认删除。

   2. 如果您不删除集群中的最后一个实例：

      - Delete <instance-name> instance? (是否删除 <instance-name> 实例?) — 在此字段中输入 delete me (删除我) 短语以确认删除。

6. 选择 Delete (删除) 以删除实例。

删除实例需要几分钟时间。要监控实例的状态，请参阅监控 Amazon DocumentDB 实例的状态。

Using the AWS CLI

以下过程使用 AWS CLI删除 Amazon DocumentDB 实例。

1. 首先，确定您的 Amazon DocumentDB 集群中有多少个实例：

   要确定集群中有多少个实例，请运行 `describe-db-clusters` 命令，如下所示。

   ```
   aws docdb describe-db-clusters \
       --db-cluster-identifier sample-cluster \
       --query 'DBClusters[*].
   [DBClusterIdentifier,DBClusterMembers[*].DBInstanceIdentifier]'
   ```

   此操作的输出将类似于下文。

   ```
   [
       [
           "sample-cluster",
           [
               "sample-instance-1",
               "sample-instance-2"
           ]
       ]
   ]
   ```

2. 如果您的 Amazon DocumentDB 集群中有多个实例：

   要删除指定的 Amazon DocumentDB 实例，请使用带 `--db-instance-identifier` 参数的 `delete-db-instance` 命令，如下所示。删除实例需要几分钟时间。要监控实例的状态，请参阅监控 Amazon DocumentDB 实例的状态。

   ```
   aws docdb delete-db-instance \
          --db-instance-identifier sample-instance-2
   ```

   此操作的输出将类似于下文。

   ```
   {
       "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. 如果要删除的实例是 Amazon DocumentDB 集群中的最后一个实例：

   如果您删除 Amazon DocumentDB 集群中的最后一个实例，则还会删除该集群以及与该集群关联的自动快照和持续备份。

   要删除集群中的最后一个实例，您可以删除此集群并可以选择创建最终快照。有关更多信息，请参阅 删除 Amazon DocumentDB 集群。

删除保护

删除 Amazon DocumentDB 集群的最后一个实例也将删除该集群，以及与该集群关联的自动快照和连续备份。无论您使用还是执行删除操作，Amazon DocumentDB 都会对集群强制执行删除保护。 AWS 管理控制台 AWS CLI如果已启用删除保护，则无法删除集群。

要删除启用了删除保护的实例，首先请修改该集群并禁用删除保护。有关更多信息，请参阅 删除 Amazon DocumentDB 集群。

管理实例类

该实例类确定 Amazon DocumentDB（与 MongoDB 兼容）实例的计算和内存容量。您需要的实例类取决于您的处理能力和内存要求。

Amazon DocumentDB 支持 R4、R5、R6G、R8G、T3 和 T4G 系列的实例类。这些是最新一代的实例类，针对内存密集型应用程序进行了优化。有关这些实例类的规格，请参阅 实例类规格。

主题

- 确定实例类
- 更改实例的类
- 不同区域支持的实例类
- 实例类规格

确定实例类

要确定实例的类别，您可以使用 AWS 管理控制台 或`describe-db-instances` AWS CLI 操作。

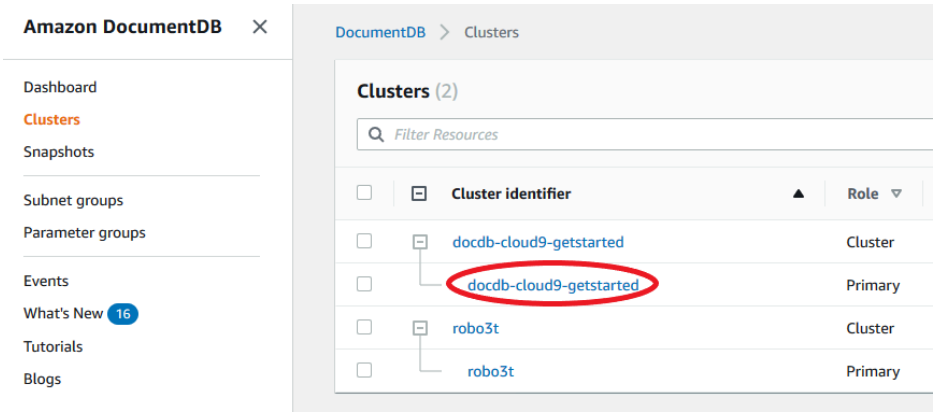
Using the AWS 管理控制台

要确定您的集群实例的实例类，请在控制台中完成以下步骤。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群以找到您感兴趣的实例。

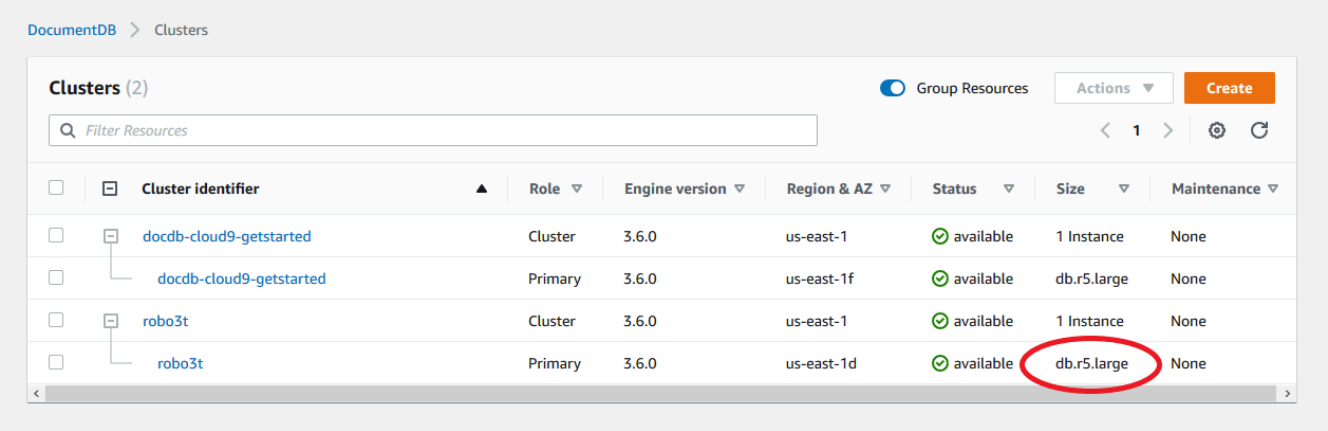
> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在集群导航框中，您将看到“集群标识符”列。您的实例列在集群下，类似于以下屏幕截图。



4. 在实例列表中，展开集群以找到您感兴趣的实例。查找您想要的实例。然后查看实例行的 大小列，以看到实例类。

   在下图中，实例 `robo3t` 的实例类为 `db.r5.4xlarge`。



Using the AWS CLI

要使用确定实例的类别 AWS CLI，请使用带有以下参数的`describe-db-instances`操作。

- **`--db-instance-identifier`**— 可选。指定要查找其实例类的实例。如果省略该参数，`describe-db-instances` 将返回最多 100 个实例的描述。
- **`--query`**：可选。指定结果中要包含的实例成员。如果省略该参数，则返回所有实例成员。

Example

以下示例为实例 `sample-instance-1` 找到实例名称和类。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-instances \
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' \
    --db-instance-identifier sample-instance-1
```

对于 Windows：

```
aws docdb describe-db-instances ^
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' ^
    --db-instance-identifier sample-instance-1
```

此操作的输出将类似于下文。

```
[
```

```
    [
        "sample-instance-1",
        "db.r5.large"
    ]
```

Example

以下示例为多达 100 个 Amazon DocumentDB 实例找到实例名称和类。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-instances \
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' \
    --filter Name=engine,Values=docdb
```

对于 Windows：

```
aws docdb describe-db-instances ^
    --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceClass]' ^
    --filter Name=engine,Values=docdb
```

此操作的输出将类似于下文。

```
[
    [
        "sample-instance-1",
        "db.r5.large"
    ],
    [
        "sample-instance-2",
        "db.r5.large"
    ],
    [
        "sample-instance-3",
        "db.r5.4xlarge"
    ],
    [
        "sample-instance-4",
        "db.r5.4xlarge"
    ]
]
```

有关更多信息，请参阅 描述 Amazon DocumentDB 实例。

更改实例的类

您可以使用 AWS 管理控制台 或更改实例的实例类别 AWS CLI。有关更多信息，请参阅 修改 Amazon DocumentDB 实例。

不同区域支持的实例类

Amazon DocumentDB 支持以下数据库实例类：

- R8G—由 Arm-based AWS Graviton4 处理器提供支持的最新一代内存优化型实例，其性能比 R6G 实例提高多达 30%。
- R6G— 由 Arm-based AWS Graviton2 处理器提供支持的Memory-optimized 实例，其性能比 R5 实例提高多达 30%，成本降低 5%。
- R6GD— Memory-optimized R6G 实例，具有基于本地非易失性存储器 Express (NVMe) 的 Solid-State 硬盘 (SSD) 存储，用于存储临时数据。
- R5— 在相同Memory-optimized 实例成本下，与 R4 实例相比，性能最高可提高 100% 的实例。
- R4：上一代内存优化型实例。
- T4G— 由 Arm-based AWS Graviton2 处理器提供支持的Latest-generation 低成本可突发通用实例类型，可提供基准 CPU 性能水平，性价比比 T3 实例高出 35%，非常适合运行 CPU 使用率中等、使用量暂时激增的应用程序。
- T3：低成本可突增通用实例类型，它提供基准级别的 CPU 性能，有能力根据需要随时突增 CPU 使用率。

有关实例类的详细规格，请参阅 实例类规格。

> **Note**
>
> I/O-Optimized 存储仅在引擎版本 5.0 和 8.0（基于实例的集群）上可用。

特定实例类在给定区域可能受或可能不受支持。下表指定了在每个区域哪些实例类受 Amazon DocumentDB 支持。

不同区域支持的实例类

Region	R8G	R6GD	R6G	R5	R4	T4G	T3	Serverless
美国东部（俄亥俄州）	支持	支持	支持	支持	支持	支持	支持	支持
美国东部（弗吉尼亚州北部）	支持	支持	支持	支持	支持	支持	支持	支持
美国西部（俄勒冈州）	支持	支持	支持	支持	支持	支持	支持	支持
非洲（开普敦）			支持	支持		支持	支持	支持
南美洲（圣保罗）		支持	支持	支持		支持	支持	支持
亚太地区（香港）			支持	支持		支持	支持	支持
亚太地区（海得拉巴）			支持	支持		支持	支持	支持
亚太地区（马来西亚）			支持			支持	支持	
亚太地区（孟买）	支持	支持	支持	支持		支持	支持	支持
亚太地区（大阪）		支持	支持	支持		支持	支持	
亚太地区（首尔）	支持	支持	支持	支持		支持	支持	支持
亚太地区（悉尼）	支持	支持	支持	支持		支持	支持	支持

Region	R8G	R6GD	R6G	R5	R4	T4G	T3	Serverless
亚太地区（雅加达）	支持	支持	支持	支持		支持	支持	
亚太地区（墨尔本）			支持	支持		支持	支持	
亚太地区（新加坡）	支持	支持	支持	支持		支持	支持	支持
亚太地区（泰国）			支持			支持	支持	
亚太地区（东京）	支持	支持	支持	支持		支持	支持	支持
加拿大（中部）	支持	支持	支持	支持		支持	支持	支持
加拿大西部（卡尔加里）			支持	支持		支持	支持	
欧洲地区（法兰克福）	支持	支持	支持	支持		支持	支持	支持
欧洲（苏黎世）		支持	支持	支持		支持	支持	
欧洲地区（爱尔兰）	支持	支持	支持	支持	支持	支持	支持	支持
欧洲地区（伦敦）		支持	支持	支持		支持	支持	支持
欧洲地区（米兰）			支持	支持		支持	支持	支持
欧洲地区（巴黎）		支持	支持	支持		支持	支持	支持

Region	R8G	R6GD	R6G	R5	R4	T4G	T3	Serverless
欧洲（西班牙）	支持	支持	支持	支持		支持	支持	支持
欧洲地区（斯德哥尔摩）	支持	支持	支持	支持		支持	支持	
墨西哥（中部）			支持			支持	支持	
中东（阿联酋）：			支持	支持		支持	支持	支持
中国（北京）		支持	支持	支持		支持	支持	支持
中国（宁夏）			支持	支持		支持	支持	支持
以色列（特拉维夫）			支持	支持		支持	支持	
AWS GovCloud (US-West)	支持	支持	支持	支持			支持	支持
AWS GovCloud (US-East)		支持	支持	支持		支持	支持	支持

实例类规格

下表提供 Amazon DocumentDB 实例类的详细信息，包括每个类支持哪些实例类型。可在表下方找到每个表列的说明。

实例类	vCPU[1]	内存 (GiB)[2]	NVMe SSD 分层缓存（GiB）[3]	最大存储温度 (GiB) 4	基准/突增带宽（Gbps）[5]	支持引擎 [6]
R8G — 基于 Graviton4 的当前一代 Memory-Optimized 实例类 R8G 是最新的实例系列 (Graviton4)，仅在引擎版本 5.0 和 8.0 上可用。尺寸 db.r8g.48xlarge 尚未上市。						

实例类	vCPU[1]	内存 (GiB)[2]	NVMe SSD 分层缓存（GiB）[3]	最大存储温度 (GiB) 4	基准/突增带宽（Gbps）[5]	支持引擎 [6]
db.r8g.large	2	16	-	30	0.937 / 12.5	5.0.0 和 8.0.0
db.r8g.xlarge	4	32	-	60	1.875 / 12.5	5.0.0 和 8.0.0
db.r8g.2xlarge	8	64	-	121	3.75 / 15.0	5.0.0 和 8.0.0
db.r8g.4xlarge	16	128	-	243	7.5 / 15.0	5.0.0 和 8.0.0
db.r8g.8xlarge	32	256	-	488	15	5.0.0 和 8.0.0
db.r8g.12xlarge	48	384	-	732	22	5.0.0 和 8.0.0
db.r8g.16xlarge	64	512	-	987	30	5.0.0 和 8.0.0
db.r8g.24xlarge	96	768	-	1484	40	5.0.0 和 8.0.0
R6G — 基于 Graviton2 的当前一代 Memory-Optimized 实例类						
db.r6g.large	2	16	-	32	0.75/ 10	4.0.0、5.0.0 和 8.0.0
db.r6g.xlarge	4	32	-	63	1.25/ 10	4.0.0、5.0.0 和 8.0.0
db.r6g.2xlarge	8	64	-	126	2.5/ 10	4.0.0、5.0.0 和 8.0.0

实例类	vCPU[1]	内存 (GiB)[2]	NVMe SSD 分层缓存（GiB）[3]	最大存储温度 (GiB) 4	基准/突增带宽（Gbps）[5]	支持引擎 [6]
db.r6g.4xlarge	16	128	-	252	5.0/ 10	4.0.0、5.0.0 和 8.0.0
db.r6g.8xlarge	32	256	-	504	12	4.0.0、5.0.0 和 8.0.0
db.r6g.12xlarge	48	384	-	756	20	4.0.0、5.0.0 和 8.0.0
db.r6g.16xlarge	64	512	-	1008	25	4.0.0、5.0.0 和 8.0.0
R6GD — 基于 Graviton2 的当前一代 NVMe-backed 实例类						
db.r6gd.xlarge	4	32	173	64	1.25/ 10	5.0.0 和 8.0.0
db.r6gd.2xlarge	8	64	346	128	2.5/ 10	5.0.0 和 8.0.0
db.r6gd.4xlarge	16	128	694	256	5.0/ 10	5.0.0 和 8.0.0
db.r6gd.8xlarge	32	256	1388	512	12	5.0.0 和 8.0.0
db.r6gd.12xlarge	48	384	2082	768	20	5.0.0 和 8.0.0
db.r6gd.16xlarge	64	512	2776	1024	25	5.0.0 和 8.0.0
R5 — 上一代 Memory-Optimized 实例类						

实例类	vCPU[1]	内存 (GiB)[2]	NVMe SSD 分层缓存（GiB）[3]	最大存储温度 (GiB) 4	基准/突增带宽（Gbps）[5]	支持引擎 [6]
db.r5.large	2	16	-	31	0.75/ 10	3.6.0、4.0.0、5.0.0 和 8.0.0
db.r5.xlarge	4	32	-	62	1.25/ 10	3.6.0、4.0.0、5.0.0 和 8.0.0
db.r5.2xlarge	8	64	-	124	2.5/ 10	3.6.0、4.0.0、5.0.0 和 8.0.0
db.r5.4xlarge	16	128	-	249	5.0/ 10	3.6.0、4.0.0、5.0.0 和 8.0.0
db.r5.8xlarge	32	256	-	504	10	3.6.0、4.0.0、5.0.0 和 8.0.0
db.r5.12xlarge	48	384	-	748	12	3.6.0、4.0.0、5.0.0 和 8.0.0
db.r5.16xlarge	64	512	-	1008	20	3.6.0、4.0.0、5.0.0 和 8.0.0
db.r5.24xlarge	96	768	-	1500	25	3.6.0、4.0.0、5.0.0 和 8.0.0

实例类	vCPU[1]	内存 (GiB)[2]	NVMe SSD 分层缓存（GiB）[3]	最大存储温度 (GiB) 4	基准/突增带宽（Gbps）[5]	支持引擎 [6]
R4 — 上一代 Memory-Optimized 实例类 只有引擎版本 3.6 支持 R4 实例。请注意，亚马逊 DocumentDB 3.6 将于 2026 年 3 月 30 日终止标准支持。Extended Support（付费）有效期至 2029 年 3 月。						
db.r4.large	2	15.25	-	30	0.75/ 10	仅 3.6.0
db.r4.xlarge	4	30.5	-	60	1.25/ 10	仅 3.6.0
db.r4.2xlarge	8	61	-	120	2.5/ 10	仅 3.6.0
db.r4.4xlarge	16	122	-	240	5.0 /10	仅 3.6.0
db.r4.8xlarge	32	244	-	480	10	仅 3.6.0
db.r4.16xlarge	64	488	-	960	25	仅 3.6.0
T4G：基于 Graviton2 的最新一代可突增性能实例类 T-series 实例在无限 CPU 突发模式下运行。超出基准的突发使用量将额外收费。不支持全局群集。						
db.t4g.medium	2	4	-	8.13	0.256/ 5	4.0.0、5.0.0 和 8.0.0
T3：上一代可突增性能实例类						
db.t3.medium	2	4	-	7.5	0.256/ 5	3.6.0、4.0.0、5.0.0 和 8.0.0

实例类	vCPU[1]	内存 (GiB)[2]	NVMe SSD 分层缓存（GiB）[3]	最大存储温度 (GiB) 4	基准/突增带宽（Gbps）[5]	支持引擎 [6]

1. vCPU：虚拟中央处理单元 (CPU) 数目。虚拟 CPU 是可用于比较实例类的容量单位。您不再购买或租用特定的处理器并用上数月或数年，而是以小时为单位租用容量。我们的目标是提供一致的 CPU 容量，无论实际的底层硬件是什么。
2. 内存 (GiB) 以吉字节计分配给实例的 RAM。内存与 vCPU 之间通常具有一致的比率。
3. NVMe SSD 分层缓存：SSD 卷上的空间，以千兆字节为单位，分配为扩展缓存，用于存储临时数据。此缓存仅在 NVMe-backed 实例中可用。
4. 最大临时存储容量（GiB）：以千兆字节为单位的空间，分配给实例用于非永久性临时文件存储。例如 NVMe-backed ，此存储托管在 NVMe-based SSD 卷上。在所有其他实例中，此存储托管在 Amazon Elastic Block Store（EBS）上。
5. 基准/突发带宽（Gbps）：突发带宽表示最大带宽，以每秒千兆位为单位。除以 8 可获得预期吞吐量，以每秒千兆字节为单位。大小为 4xlarge 及更小的实例具有基准带宽。为了满足额外的需求，他们可以使用网络 I/O 信用机制来突破其基准带宽。实例可以在有限时间内使用突增带宽，通常为 5 到 60 分钟，具体取决于实例的大小。
6. 支持引擎：支持实例类的 Amazon DocumentDB 引擎。

NVMe-backed 实例

NVMe-backed 对于具有超出常规实例内存的大型数据集的应用程序，实例的查询性能最多可提高 7 倍。这些实例利用 r6gd 实例上可用的基于本地非易失性存储 Solid-State 器 Express (NVMe) 的硬盘 (SSD) 存储来存储临时数据，从而减少基于网络的存储访问并改善读取延迟和吞吐量。

本地 SSD 空间分为两个部分：

- 分层缓存：将大约 73% 的本地 SSD 分配为数据库缓存，与仅使用主内存相比，使系统能够存储多达五倍的数据库页面。本地 SSD 充当第二层缓存，而现有的内存缓冲区缓存仍作为第一层缓存。仅当缓冲区缓存和 SSD 缓存都未命中时，查询才会访问外部存储。
- 临时存储：剩余的 27% 用于非永久性临时文件存储，用于涉及排序的复杂查询或资源密集型操作（如索引构建）。在常规实例中，临时空间位于 Amazon Elastic Block Store（EBS）卷上。SSD 上本地托管的临时存储可将涉及排序的查询延迟最多减少一半，并加快索引构建等资源密集型操作的速度。

有关支持的 NVMe-backed 实例类型及其规格的详细信息，请参阅实例类规格。

主题

- 实例的推荐用 NVMe-backed 例
- 在亚马逊文档数据库中使用 NVMe-backed 实例
- 监控 NVMe-backed 实例

实例的推荐用 NVMe-backed 例

我们建议您在以下场景中使用 NVMe-backed 实例：

- Read-heavy 工作负载 — 如果您的工作负载是读取密集型的，并且您的数据集大于缓冲区缓存（由低指标`BufferCacheHitRatio`和高`ReadIOPS`指标表示），则 NVMe-backed 实例可以提供性能优势。
- Update-heavy 工作负载 — 如果您的工作负载是更新密集型的，并且由于网络存储的读取延迟而导致垃圾收集无法跟上，则 NVMe-backed 实例可以帮助缓解问题。

NVMe-backed 实例可以使各种用例受益，包括：

- Internet-scale 应用程序 — 诸如支付处理、计费和电子商务等具有严格性能的应用程序服务级别协议 (SLA) 可以利用 NVMe-backed 实例的性能优势。
- Real-time 报告仪表板 — 运行数百个查询进行 metrics/data 收集的仪表板可以受益于 NVMe-backed 实例的低延迟和高吞吐量。
- 生成式 AI 应用程序 — 使用矢量搜索在数百万个矢量嵌入中查找精确或最近的邻居的应用程序可以利用 NVMe-backed 实例的高性能。

在亚马逊文档数据库中使用 NVMe-backed 实例

要使用亚马逊文档 NVMe-backed 数据库的实例，请执行以下操作：

- 创建一个 Amazon DocumentDB 集群并添加其中一个 NVMe-backed 实例类。有关更多信息，请参阅 创建 Amazon DocumentDB 集群。
- 或者，修改现有的 Amazon DocumentDB 集群以使用其中一个 NVMe-backed 实例类。有关更多信息，请参阅 修改 Amazon DocumentDB 集群。

要查看不同 AWS 区域的 NVMe-backed 实例可用性，请参阅不同区域支持的实例类。

如果您想从 NVMe-backed 实例切换回常规实例，请将 Amazon DocumentDB 实例的数据库实例类修改为不带 NVMe 存储空间的类似实例类。例如，如果当前实例类是“db.r6gd.4xlarge”，请选择“db.r6g.4xlarge”以切换回该实例类。有关更多信息，请参阅 修改 Amazon DocumentDB 集群。

监控 NVMe-backed 实例

除了 Amazon 中提供的常规实例指标外 CloudWatch， NVMe-backed 实例还会发出特定于 NVMe-based SSD 存储、IOPS 和吞吐量的额外指标。

```
NVMeStorageCacheHitRatio
FreeNVMeStorage
ReadIOPSNVMeStorage
ReadLatencyNVMeStorage
ReadThroughputNVMeStorage
WriteIOPSNVMeStorage
WriteLatencyNVMeStorage
WriteThroughputNVMeStorage
```

有关这些指标的更多信息，请参阅 NVMe-backed 实例指标。

管理 Amazon DocumentDB 子网组

虚拟私有云 (VPC) 是专用于您的虚拟网络 AWS 账户。它在逻辑上与 AWS 云中的其他虚拟网络隔离。您可以将您的 AWS 资源（例如亚马逊文档数据库集群）启动到您的亚马逊 VPC 中。您可以为 VPC 指定 IP 地址范围、添加子网、关联安全组以及配置路由表。

子网是您的 Amazon VPC 内的 IP 地址范围。您可以将 AWS 资源启动到指定的子网。对于必须连接到 Internet 的资源，请使用公有 子网。对于不连接到 Internet 的资源，请使用私有 子网。有关公有子网和私有子网的更多信息，请参阅 Amazon Virtual Private Cloud 用户指南中的 VPC 和子网基础知识。

数据库子网组是在 VPC 中创建的一组子网，并随后为集群指定这些子网。通过使用子网组，您可以在创建集群时指定特定的 VPC。如果使用 `default` 子网组，它将涵盖 VPC 中的所有子网。

每个数据库子网组应包含给定 区域中至少两个可用区的子网。在 VPC 中创建数据库集群时，您必须选择一个数据库子网组。Amazon DocumentDB 使用该数据库子网组和首选的可用区选择一个子网以及该子网中的 IP 地址，以便与集群相关联。如果主实例发生故障，Amazon DocumentDB 可将相应的副本实例提升为新的主实例。然后，它可以使用先前主实例所在子网的 IP 地址创建新的副本实例。

当 Amazon DocumentDB 在 VPC 中创建实例时，它使用从数据库子网组中选择的 IP 地址将网络接口分配给集群。我们强烈建议您使用 DNS 名称，因为基本 IP 地址在故障转移期间可能会发生变化。有关更多信息，请参阅 Amazon DocumentDB 端点。

有关创建您自己的 VPC 和子网的信息，请参阅 Amazon Virtual Private Cloud 用户指南中的使用 VPC 和子网。

主题

- 创建 Amazon DocumentDB 子网组
- 描述 Amazon DocumentDB 子网组
- 修改 Amazon DocumentDB 子网组
- 删除 Amazon DocumentDB 子网组

创建 Amazon DocumentDB 子网组

创建 Amazon DocumentDB 集群时，您必须选择一个 Amazon VPC 和该 Amazon VPC 中的对应子网组来启动您的集群。子网确定可用区以及该可用区内要用于启动实例的 IP 范围。

子网组是可让您指定要用于启动 Amazon DocumentDB 实例的子网（或可用区）的命名集。例如，在包含三个实例的集群中，建议在单独的可用区中预置各个实例：这样做优化高可用性。因此，如果单个可用区出现故障，它只会影响单个实例。

目前， Amazon DocumentDB 实例可在多达三个可用区中预置。即使子网组拥有三个以上的子网，您也只能使用这些子网中的三个来创建 Amazon DocumentDB 集群。因此，在创建子网组时，建议仅选择要将实例部署到的三个子网。

例如：创建了一个集群，Amazon DocumentDB 选择可用区 {1A、1B 和 1C}。如果您尝试在可用区 {1D} 中创建实例，API 调用将失败。但是，如果您选择创建实例而不指定特定可用区，则 Amazon DocumentDB 将代表您选择可用区。Amazon DocumentDB 使用一种算法在可用区之间对实例进行负载均衡，以帮助您实现高可用性。如果预置了三个实例，则在默认情况下，将在三个可用区中预置它们，而不会在单一可用区中预置。

最佳实践

- 除非您有特殊原因，否则请始终创建包含三个子网的子网组。这将确保包含三个或更多实例的集群能够实现更高的可用性，因为将在三个可用区中预置实例。
- 始终将实例分散在多个可用区中以实现高可用性。切勿将集群的所有实例放在单个可用区中。
- 由于故障转移事件随时可能发生，您不应假定主实例或副本实例始终位于特定可用区中。

如何创建子网组

您可以使用 AWS 管理控制台 或 AWS CLI 创建 Amazon DocumentDB 子网组：

Using the AWS 管理控制台

使用以下步骤创建 Amazon DocumentDB 子网组。

创建 Amazon DocumentDB 子网组

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择 Subnet groups (子网组)，然后选择 Create (创建)。

   > **Tip**
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在创建子网组页面中：

   a. 在子网组详细信息部分中：

      i. 名称：为子网组输入有意义的名称。

      ii. 描述：输入子网组描述。

   b. 在添加子网部分中：

      i. VPC：在列表中，为该子网组选择一个 VPC。

      ii. 请执行以下操作之一：

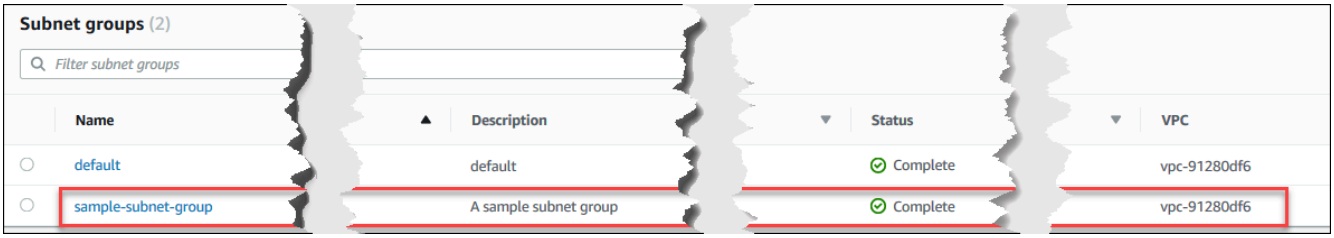
      - 要包括所选的 VPC 中的所有子网，请选择添加与此 VPC 相关的所有子网。
      - 要为该子网组指定子网，请为要包含子网的每个可用区执行以下操作。您必须包含至少两个可用区。

        A. Availability zone (可用区)：在列表中，选择一个可用区。

        B. Subnet (子网)：在列表中，从该子网组所选可用区中选择一个子网。

        C. 选择 Add subnet (添加子网)。

4. 选择创建。创建子网组后，它将与其他子网组一起列出。



Using the AWS CLI

在使用创建子网组之前 AWS CLI，必须先确定哪些子网可用。运行以下 AWS CLI 操作以列出可用区及其子网。

参数：

- **`--db-subnet-group`**：可选。如果指定特定的子网组，将列出该组的可用区和子网。如果省略该参数，将列出所有子网组的可用区和子网。如果指定 `default` 子网组，将列出 VPC 的所有子网。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-subnet-groups \
    --db-subnet-group-name default \
    --query 'DBSubnetGroups[*].[DBSubnetGroupName,Subnets[*].
[SubnetAvailabilityZone.Name,SubnetIdentifier]]'
```

对于 Windows：

```
aws docdb describe-db-subnet-groups ^
    --db-subnet-group-name default ^
    --query 'DBSubnetGroups[*].[DBSubnetGroupName,Subnets[*].
[SubnetAvailabilityZone.Name,SubnetIdentifier]]'
```

此操作的输出将类似于下文（JSON 格式）。

```
[
    [
        "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"
            ]
        ]
    ]
]
```

通过使用上一操作中的输出，您可以创建新的子网组。新子网组必须包含至少两个可用区中的子网。

参数：

- **--db-subnet-group-name**：必填项。该子网组的名称。
- **--db-subnet-group-description**：必填项。该子网组的描述。
- **--subnet-ids**：必填项。要包含在该子网组中的子网的列表。示例：subnet-53ab3636。
- --标签：可选。要附加到该子网组的标签（键/值对）的列表。

以下代码创建具有三个子网（subnet-4e26d263、subnet-afc329f4 和 subnet-b3806e8f）的子网组 sample-subnet-group。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "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"
    }
}
```

描述 Amazon DocumentDB 子网组

您可以使用 AWS 管理控制台 或 AWS CLI 来获取 Amazon DocumentDB 子网组的详细信息。

Using the AWS 管理控制台

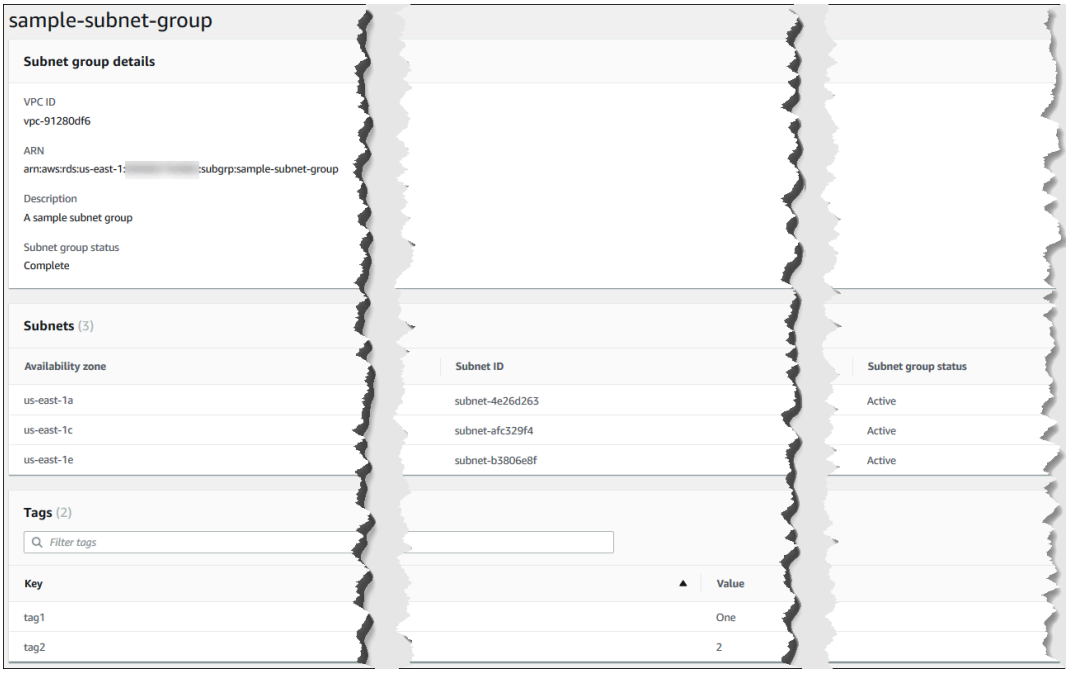
以下过程说明了如何获取 Amazon DocumentDB 子网组的详细信息。

查找子网组的详细信息

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择子网组。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 要查看子网组的详细信息，请选择该子网组的名称。



Using the AWS CLI

要查找 Amazon DocumentDB 子网组的详细信息，请使用具有以下参数的 `describe-db-subnet-groups` 操作。

参数

- `--db-subnet=group-name`：可选。如果包含，则列出指定的子网组的详细信息。如果省略，则列出最多 100 个子网组的详细信息。

Example

以下代码列出在创建 Amazon DocumentDB 子网组部分中创建的 `sample-subnet-group` 子网组的详细信息。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-subnet-groups \
    --db-subnet-group-name sample-subnet-group
```

对于 Windows：

```
aws docdb describe-db-subnet-groups ^
```

```
    --db-subnet-group-name sample-subnet-group
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "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"
    }
}
```

修改 Amazon DocumentDB 子网组

您可以使用或修改子网组的描述 AWS CLI ， AWS 管理控制台 或者在 Amazon DocumentDB 子网组中添加或删除子网。不过，您无法修改 `default` 子网组。

Using the AWS 管理控制台

您可以使用 AWS 管理控制台 来更改子网组的描述或添加和删除子网。请记住，在完成后，您必须具有至少两个与子网组关联的可用区。

修改子网组

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择子网组。然后，选择子网组名称左侧的按钮。请记住，您无法修改 `default` 子网组。

   > **Tip**
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 选择 Actions (操作)，然后选择 Modify (修改)。
4. Description (描述)：要更改子网组描述，请输入新描述。
5. 要更改与子网组关联的子网，请在添加子网部分中执行一个或多个以下操作：
   - 要从该子网组中删除所有子网，请选择全部删除。
   - 要从该子网组中删除特定的子网，请为要删除的每个子网选择删除。
   - 要添加与该 VPC 关联的所有子网，请选择添加与此 VPC 相关的所有子网。
   - 要将特定子网添加到该子网组中，请为要添加子网的每个可用区执行以下操作。
     a. Availability zone (可用区)：在列表中，选择新的可用区。
     b. Subnet (子网)：在列表中，从该子网组所选可用区中选择一个子网。
     c. 选择 Add subnet (添加子网)。
6. 在确认对话框中：
   - 要对子网组进行这些更改，请选择 Modify (修改)。
   - 要将子网组保持不变，请选择取消。

Using the AWS CLI

您可以使用 AWS CLI 来更改子网组的描述或添加和删除子网。请记住，在完成后，您必须具有至少两个与子网组关联的可用区。您无法修改 `default` 子网组。

参数：

- `--db-subnet-group-name`：必填项。要修改的 Amazon DocumentDB 子网组的名称。
- `--subnet-ids`：必填项。完成此更改后，子网组中所需的所有子网的列表。

> **Important**
>
> 将从子网组中删除当前位于子网组中并且未包含在该列表中的任何子网。如果要保留当前位于子网组中的任何子网，您必须将其包含在该列表中。

- `--db-subnet-group-description`：可选。子网组的描述。

Example

以下代码修改描述并将现有子网替换为 `subnet-991cb8d0`、`subnet-53ab3636` 和 `subnet-29ab1025` 子网。

对于 Linux、macOS 或 Unix：

```
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"
```

对于 Windows：

```
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"
```

此操作的输出将类似于下文（JSON 格式）。请注意，这是在创建 Amazon DocumentDB 子网组部分中创建的子网组。但是，此子网组中的子网将替换为 `modify-db-subnet-group` 操作中列出的子网。

```
{
    "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"
    }
}
```

删除 Amazon DocumentDB 子网组

您可以使用 AWS 管理控制台 或 AWS CLI 删除 Amazon DocumentDB 子网组。不过，您无法删除 `default` 子网组。

Using the AWS 管理控制台

您可以使用 AWS 管理控制台 来删除子网组。但是，您无法删除 `default` 子网组。

删除子网组

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb

2. 在导航窗格中，选择子网组。然后，选择子网组名称左侧的按钮。请记住，您无法删除 `default` 子网组。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 选择操作，然后选择删除。
4. 在确认对话框中：
   - 要删除子网组，请选择删除。
   - 要保留子网组，请选择取消。

Using the AWS CLI

要使用删除 Amazon DocumentDB 子网组 AWS CLI，请使用带以下参数的`delete-db-subnet-group`操作。

参数

- `--db-subnet-group-name`：必填项。要删除的 Amazon DocumentDB 子网组的名称。请记住，您无法删除 `default` 子网组。

Example

以下代码删除 `sample-subnet-group`。

对于 Linux、macOS 或 Unix：

```
aws docdb delete-db-subnet-group \
    --db-subnet-group-name sample-subnet-group
```

对于 Windows：

```
aws docdb delete-db-subnet-group ^
    --db-subnet-group-name sample-subnet-group
```

该操作不会生成任何输出。

Amazon DocumentDB 高可用性和复制

通过使用副本实例，可以在 Amazon DocumentDB（与 MongoDB 兼容）中实现高可用性和读取扩展。单个 Amazon DocumentDB 集群支持单个主实例和最多 15 个副本实例。这些实例可以分布在集群区域的多个可用区中。主实例接受读取和写入流量，而副本实例仅接受读取请求。

集群卷由该集群的多个数据副本组成。不过，集群卷中的数据，对于主实例和数据库集群中的 Amazon DocumentDB 副本表示为单个逻辑卷。副本实例具有最终一致性。它们返回具有最短副本滞后的查询结果：通常远远少于主实例写入更新后的 100 毫秒。副本滞后因数据库更改速率而异。也就是说，在对数据库执行大量写入操作期间，您可能发现副本滞后时间变长。

读取扩展

Amazon DocumentDB 副本十分适用于读取扩展，因为它们完全专用于集群卷上的读取操作。写入操作由主实例进行管理。集群卷在集群中的所有实例之间共享。因此，您不必为每个 Amazon DocumentDB 副本复制和维护数据副本。

高可用性

在创建 Amazon DocumentDB 集群时，根据子网组中的可用区数（必须至少为两个），Amazon DocumentDB 将在这些可用区中预置实例。在集群中创建实例时，Amazon DocumentDB 会自动在子网组的可用区中分发实例来平衡集群。此操作还会阻止所有实例位于同一可用区中。

示例

为了说明这一点，请考虑以下示例：创建一个集群，其中包含一个具有三个可用区（AZ1、AZ2 和 AZ3）的子网组。

在创建集群中的第一个实例时，它是主实例并位于其中的一个可用区中。在本示例中，它位于 AZ1 中。创建的第二个实例是副本实例，它位于另外两个可用区之一中，例如，AZ2。创建的第三个实例是副本实例，它位于剩余的可用区 AZ3 中。如果创建更多的实例，它们将分布在多个可用区中，以便在集群中实现平衡。

如果在主实例 (AZ1) 中发生故障，则会触发失效转移，并将其中的一个现有副本提升为主实例。在旧的主实例恢复时，它将变为预置它的同一可用区 (AZ1) 中的副本。当您配置一个三实例集群时，Amazon DocumentDB 继续保留这个三实例集群。Amazon DocumentDB 自动处理实例故障的检测、失效转移和恢复，无需任何手动干预。

在 Amazon DocumentDB 执行失效转移并恢复实例时，恢复的实例保留在最初预置它的可用区中。但是，此实例的角色可能从主实例变为副本。这样做是为了防止出现以下情况：一系列失效转移可能导致所有实例位于同一可用区中。

您可以将 Amazon DocumentDB 副本指定为失效转移目标。即，如果主实例发生故障，指定的 Amazon DocumentDB 副本或某个层中的副本将提升为主实例。提升过程只造成短暂的中断，在此期间，对主实例发出的读写请求将失败，并且会出现异常。如果您的 Amazon DocumentDB 集群不包含任何 Amazon DocumentDB 副本，则会在主实例失败期间重新创建主实例。提升 Amazon DocumentDB 副本要比重新创建主实例快得多。

对于高可用性场景，建议您创建一个或多个 Amazon DocumentDB 副本。您的 Amazon DocumentDB 集群应该与主实例具有相同的实例类，并且位于不同可用区中。

有关更多信息，请参阅下列内容：

- 了解 Amazon DocumentDB 集群容错能力
- Amazon DocumentDB 失效转移
  - 控制失效转移目标

在 Amazon DocumentDB 版本 5.0 和 8.0 中，副本实例在主实例重启时不会重启。它们在主实例重启期间继续为读取请求提供服务，从而保持更高的可用性。

全局集群可用性高

为了实现跨多个集群的高可用性 AWS 区域，您可以设置 Amazon DocumentDB 全球集群。每个全球集群跨越多个区域，从而实现低延迟的全局读取和跨区域中断灾难恢复。 AWS 区域 Amazon DocumentDB 自动处理从主区域到每个辅助区域的数据复制以及更新。

添加 副本

添加到集群的第一个实例是主实例。在第一个实例之后添加的每一个实例都是副本实例。除主实例之外，每个集群最多可拥有 15 个副本实例。

使用创建集群时 AWS 管理控制台，会同时自动创建主实例。创建集群和主实例的同时还要创建副本，请选择 Create replica in different zone (在其他区域创建副本)。有关更多信息，请参阅 创建 Amazon DocumentDB 集群 中的步骤 4.d。要将更多副本添加到 Amazon DocumentDB 集群，请参阅向集群添加 Amazon DocumentDB 实例 。

使用创建集群时，必须明确创建主实例和副本实例。 AWS CLI 有关更多信息，请参阅以下主题中的“使用 AWS CLI” 部分：

- 创建 Amazon DocumentDB 集群
- 向集群添加 Amazon DocumentDB 实例

Amazon DocumentDB 失效转移

在某些情况下，例如某些类型的计划维护，或出现不太可能出现的主节点或可用区故障时，Amazon DocumentDB（与 MongoDB 兼容）会检测故障并替换主节点。在故障转移期间，写入停机时间将最小化。因为主节点的角色故障转移到一个只读副本，而不是必须创建和设置新的主节点。此故障检测和副本提升可确保在提升完成后，您可以马上继续写入新的主节点。

要使失效转移起作用，您的集群必须至少具有两个实例：一个主实例和至少一个副本实例。

> **Note**
>
> 本主题仅适用于 Amazon DocumentDB 基于实例的原始集群。不适用于弹性集群或全局集群。

控制失效转移目标

Amazon DocumentDB 提供失效转移层作为方法来控制在发生失效转移时，将哪个副本实例提升为主实例。

故障转移层

每个副本实例都与一个失效转移层 (0–15) 关联。当因为维护或意外硬件故障而发生失效转移时，主实例将故障转移到具有最高优先级（最低编号层）的副本。如果多个副本具有相同的优先级层，则主实例将故障转移到大小与前主实例最接近的层的副本。

通过将一组选择副本的故障转移层设置为 0（最高优先级），可以确保故障转移将提升该组中的副本之一。您可以通过为特定副本分配低优先级层（编号高）来有效阻止这些副本在发生故障转移时被提升为主实例。当特定副本被应用程序大量使用并且故障转移到其中一个副本会对关键应用程序产生不利影响时，此方法很有用。

您可以在创建实例时或稍后通过修改实例来设置它的故障转移层。通过修改实例来设置实例故障转移不会触发故障转移。有关更多信息，请参阅以下主题：

- 向集群添加 Amazon DocumentDB 实例
- 修改 Amazon DocumentDB 实例

手动启动故障转移时，可通过两种方法来控制将哪些副本实例提升为主实例：前面介绍的故障转移层和 `--target-db-instance-identifier` 参数。

--target-db-#####

为了进行测试，您可以使用 `failover-db-cluster` 操作强制故障转移事件。您可以使用 `--target-db-instance-identifier` 参数指定将哪个副本实例提升为主实例。使用 `--target-db-instance-identifier` 参数取代故障转移优先级层。如果您不指定 `--target-db-instance-identifier` 参数，主故障转移将与故障转移优先级层保持一致。

失效转移过程中发生的事件

Amazon DocumentDB 将自动处理失效转移，以便应用程序尽快恢复数据库操作而无需管理干预。

- 如果您在相同或不同的可用区中有 Amazon DocumentDB 副本实例，当进行故障转移时：Amazon DocumentDB 会翻转您的实例的规范名称记录（CNAME），以指向运行状态正常的副本；相应地，此副本会晋升为新的主实例。从开始到结束，故障转移通常会在 30 秒内完成。
- 如果您没有 Amazon DocumentDB 副本实例（例如，单个实例集群）：Amazon DocumentDB 将尝试在与原始实例相同的可用区中创建新实例。原实例会尽量替换，但可能不会成功，例如出现全面影响该可用区的问题时。

您的应用程序应在连接丢失时重试数据库连接。

测试失效转移

集群的失效转移会将集群中的 Amazon DocumentDB 副本之一（只读实例）提升为主实例（集群写入器）。

当主实例发生故障时，Amazon DocumentDB 将自动故障转移到 Amazon DocumentDB 副本（如果存在）。当您要模拟主实例的故障以进行测试时，可以强制进行故障转移。集群中的每个实例都有自己的终端节点地址。因此，您需要在故障转移完成后清理并重新建立使用这些终端节点地址的任何现有连接。

要强制进行故障转移，请使用带以下参数的 `failover-db-cluster` 操作。

- `--db-cluster-identifier`：必填项。要进行故障转移的集群的名称。
- `--target-db-instance-identifier`：可选。要提升为主实例的实例的名称。

Example

以下操作将强制对 sample-cluster 集群进行故障转移。它不指定哪个实例要成为新的主实例，因此 Amazon DocumentDB 将根据失效转移层优先级来选择实例。

对于 Linux、macOS 或 Unix：

```
aws docdb failover-db-cluster \
   --db-cluster-identifier sample-cluster
```

对于 Windows：

```
aws docdb failover-db-cluster ^
   --db-cluster-identifier sample-cluster
```

以下操作将强制 sample-cluster 集群进行故障转移，同时指定将 sample-cluster-instance 提升为主角色。（请注意输出中的 "IsClusterWriter": true。）

对于 Linux、macOS 或 Unix：

```
aws docdb failover-db-cluster \
   --db-cluster-identifier sample-cluster \
   --target-db-instance-identifier sample-cluster-instance
```

对于 Windows：

```
aws docdb failover-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --target-db-instance-identifier sample-cluster-instance
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "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"
    }
}
```

复制滞后

复制延迟通常为 50 毫秒或更短。副本延迟增长的最常见原因是：

- 主服务器上高写入速率导致只读副本落后于主副本。
- 长时间运行的查询（例如，大型顺序扫描、聚合查询）与传入性写入复制之间的只读副本争用。
- 只读副本上有数目非常庞大的并发查询。

要最大限度减少复制延迟，请尝试以下故障排除技术：

- 如果您的写入速率高或 CPU 使用率较高，我们建议您纵向扩展集群中的实例。
- 如果您的只读副本上有长时间运行的查询，并且所查询的文档更新非常频繁，请考虑更改长时间运行的查询，或者针对副本运行这些查询，以避免只读 primary/write 副本争用。
- 如果只读副本上数目非常庞大的并发查询，或者仅只读副本上 CPU 使用率高，则另一种选项是横向扩展只读副本的数目以分散工作负载。
- 由于复制延迟是由高写入吞吐量和长时间运行查询造成的，因此我们建议将 DBClusterReplicaLagMaximum CW 指标与慢速查询记录器和`WriteThroughput`/`WriteIOPS`指标结合使用，对复制延迟进行故障排除。

通常，我们建议您的所有副本都属于相同实例类型，从而群集失效转移将不导致性能下降。

如果您正在在纵向扩展与横向扩展（例如，六个较小实例与三个较大实例）之间选择，我们通常建议您在扩展之前尝试纵向扩展（较大实例），因为您将获得每个 DB 实例更大的缓冲区缓存。

您应该前瞻性设置复制延迟警报，并将其阈值设置成这样一个值，您认为该值是副本实例上您的数据在它开始影响应用程序的功能之前可以落后（或“过时”）多久的上限。通常，由于工作负载转瞬即逝，我们将建议报警之前复制延迟阈值应被超过几个数据点。

> **Note**
>
> 此外，我们建议您为超过 10 秒的复制延迟设置另一个警报。如果您超越这个阈值延续多个数据点，我们建议您纵向扩展您的实例或降低主实例上的写入吞吐量。

管理 Amazon DocumentDB 索引

主题

- Amazon DocumentDB 索引创建
- 维护 Amazon DocumentDB 索引

Amazon DocumentDB 索引创建

在 Amazon DocumentDB 中创建索引需要做出一些决定：

- 需要多久完成？
- 在构建过程中无法访问该集合吗？
- 实例的计算能力中有多少可以分配给构建？
- 应该创建什么类型的索引？

本节帮助您回答这些问题，并提供在基于实例的集群集合上创建 Amazon DocumentDB 索引的命令和监控示例。

指南要求

以下指南包括创建新索引时的基本限制和配置权衡：

- Amazon DocumentDB 版本支持-虽然所有版本的亚马逊 DocumentDB 都支持单工作人员索引，但只有亚马逊 DocumentDB 版本 4.0、5.0 和 8.0 支持多工作程序索引。
- 性能权衡：在索引创建过程中增加工作进程数量会增加 Amazon DocumentDB 数据库主实例上的 CPU 利用率和读取 IO。创建新索引所需的资源将无法用于正在运行的工作负载。
- 弹性集群：Amazon DocumentDB 弹性集群不支持并行索引。
- 最大工作进程：您可以配置的最大工作进程取决于数据库集群中主实例的大小。其等于数据库集群中主实例上 vCPU 总数的一半。例如，在有 64 个 vCPU 的 db.r6g.16xlarge 实例上，你最多可以运行 32 个工作进程。

> **Note**
>
> 2xlarge 及更低版本的实例类不支持并行工作进程。

- 最少工作进程：您可以配置的最小工作进程为 1。在基于实例的集群上创建索引的默认设置是两个工作进程。但是，您可以使用“工作线程”选项将工作进程减少到一个。这将使用单工作进程运行该进程。
- 索引压缩：Amazon DocumentDB 不支持索引压缩。索引的数据大小可能比您使用其他选项时更大。
- 索引多个集合：数据库集群主实例上一半的 vCPU 可用于已配置的工作进程对多个集合执行索引创建。
- 索引类型：有关 Amazon DocumentDB 上支持的索引类型的完整说明，请参阅此博客文章。

开始使用

要开始在集合上创建索引，请使用 `createIndexes` 命令。默认情况下，该命令将运行两个并行工作进程，可将索引创建过程的速度提高两倍。

例如，以下命令过程演示了如何为文档中的 “user_name” 字段创建索引，并将索引处理速度提至四个工作进程：

1. 在集群上使用两个并行工作进程创建索引：

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx"}]})
```

2. 要优化索引创建过程的速度，您可以使用 `db.runCommand createIndexes` 命令中的“工作线程”选项（`"workers":<number>`）指定工作进程数。

   将处理速度提至四个并行工作进程：

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx", "workers":4}]})
```

> **Note**
>
> 工作进程数越多，则索引创建速度越快。但是，工作进程数增幅越大，主实例的 vCPU 的加载及 IO 的读取增速越快。确保您的集群配置充足，足以应对增加的负担，不会使其他工作负载降级。

索引进度状态

索引创建过程的工作原理是初始化、扫描集合、排序键，最后通过索引生成器插入键。在前台运行时，该过程最多有六个阶段，在后台运行时，则最多有九个阶段。您可以逐阶段查看状态指标，例如完成百分比、扫描的存储块总数、已排序键和已插入键。

使用 mongo Shell 中的`db.currentOp()`命令监控索引过程进度。最后一个阶段的 100% 已完成表示所有索引都已成功创建：

```
db.currentOp({"command.createIndexes": { $exists : true } })
```

> **Note**
>
> 仅支持在 Amazon DocumentDB 5.0 上查看索引进度状态。

索引构建类型

索引构建的四种类型是：

- 前台：在创建索引之前，前台索引构建将阻止所有其他数据库操作。Amazon DocumentDB 前台构建由五个阶段组成。
- 前台（唯一）：单个文档（唯一）前台索引构建将阻止其他数据库操作，例如常规前台构建。与基本前台构建不同，唯一构建使用额外的阶段（排序键 2）来查找重复的键。前台（唯一）构建由六个阶段组成。
- 后台：后台索引构建允许创建索引时在前台运行其他数据库操作。Amazon DocumentDB 后台构建由八个阶段组成。
- 后台（唯一）：单个文档（唯一）后台索引构建允许在创建索引时在前台运行其他数据库操作。与基本背景构建不同，唯一构建使用额外的阶段（排序键 2）来查找重复的键。后台（唯一）构建由九个阶段组成。

索引构建阶段

暂存区	前台	前台（唯一）	背景	背景（唯一）
INITIALIZING	1	1	1	1
构建索引：初始化	2	2	2	2
构建索引：扫描集合	3	3	3	3
构建索引：排序键 1	4	4	4	4
构建索引：排序键 2		5		5
构建索引：插入键	5	6	5	6
验证：扫描索引			6	7
验证：排序元组			7	8
验证：扫描集合			8	9

- 正在初始化：createIndex 正在准备索引生成器。该阶段应该非常短暂。
- 构建索引：正在初始化：索引生成器正在准备创建索引。该阶段应该非常短暂。
- 构建索引：扫描集合：索引生成器正在执行集合扫描以收集索引秘钥。测量单位是“块”。

> **Note**
>
> 如果为索引构建配置了多个工作进程，则在此阶段将显示该工作进程。“扫描集合”阶段是在索引构建过程中使用多个工作进程的唯一阶段。所有其他阶段将显示单工作进程。

- 构建索引：排序键 1：索引生成器正在排序已收集的索引键。测量单位是“键”。

- 构建索引：排序键 2：索引生成器正在对收集的与死元组相对应的索引键进行排序。此阶段仅适用于唯一索引构建。测量单位是“键”。
- 构建索引：插入键：索引生成器正在将索引键插入到新索引中。测量单位是“键”。
- 验证：扫描索引：createIndex 正在扫描索引以查找需要验证的键。测量单位是“块”。
- 验证：排序元组：createIndex 正在排序索引扫描阶段的输出。
- 验证：扫描集合：CreateIndex 正在扫描集合以验证在前两个阶段中找到的索引键。测量单位是“块”。

索引构建输出示例

在以下输出示例（前台索引构建）中，显示了索引创建的状态。“msg”字段通过指示构建的阶段和完成百分比来汇总构建进度。“工作进程”字段表示在该索引构建阶段使用的工作进程数。“进度”字段显示用于计算完成百分比的实际数字。

> **Note**
>
> 亚马逊 DocumentDB 版本 4.0 不支持 “当前IndexBuildName”、“消息” 和 “进度” 字段。

```
{
    "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
}
```

维护 Amazon DocumentDB 索引

主题

- 索引膨胀
- 使用 re Index 维护索引

索引膨胀

Amazon DocumentDB 使用 Multi-Version 并发控制 (MVCC) 来管理并发交易。删除或更新文档后，其先前版本将作为“失效”版本保留在集合和索引中。垃圾回收过程会自动从这些失效版本中回收空间以用于未来的操作。

当集合的索引因失效或过时的索引条目累积或页面内的碎片累积而变大时，就会出现索引膨胀。报告的百分比表示未来索引条目可以使用的索引空间量。此膨胀会同时消耗缓冲区缓存和存储空间中的空间。如果要消除膨胀，则需要重建索引。

Example示例

运行以下命令以确定索引未使用的存储空间：

```
db.coll.aggregate({$indexStats:{}});
```

结果类似如下：

```
{
    "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
    }
}
```

使用 reIndex 命令无需停机即可重建索引，该命令需要扫描整个集合。请参阅使用 re Index 维护索引。

使用 **re** Index 维护索引

reIndex 是用于重建索引的命令。通常在索引损坏或效率低下时使用。随着时间的推移，索引会因为多次更新、插入或删除而积累未使用的空间，从而导致性能下降。重新编制索引有助于删除此类未使用的空间并恢复索引的效率。

######

- reIndex 目前仅在 Amazon DocumentDB 5.0 上受支持。
- Amazon DocumentDB 支持在后台进行单个索引的 reindex，以便多个工作人员操作。在 reIndex 程序运行时，查询可以使用旧索引。
- Amazon DocumentDB 支持通过 currentOp 发布索引进度报告。您可以看到与创建索引时看到的索引构建阶段 类似的索引构建阶段。唯一的区别是 reIndex 始终有八个阶段，无论它是否唯一。没有“构建索引：排序键 2”阶段。
- reIndex 可以与同一个集合上的任何命令同时运行，但与索引相关的下列命令除外：createIndexes、dropIndexes、collMod 和 renameCollection。
- reIndex 目前不支持文本索引、地理空间索引、向量索引和部分索引。

######

使用以下命令来重建索引：

```
db.runCommand({ reIndex: "collection-name", index: "index-name"})
```

您也可以选择控制分配给重建过程的工作人员的人数：

```
db.runCommand({ reIndex: "collection-name", index: "index-name", workers: number })
```

管理集合级文档压缩

Amazon DocumentDB 5.0 馆藏级文档压缩允许您通过压缩馆藏中的文档来降低存储和 IO 成本。您可以在集合层面启用文档压缩，并通过用压缩指标（例如已压缩文档的存储大小和压缩状态）衡量存储增益，根据需要查看压缩指标。亚马逊 DocumentDB 5.0 使用 LZ4 压缩算法来压缩文档。

Amazon DocumentDB 从版本 5.0 开始支持文档压缩。以下是集合级文档压缩功能：

- 默认行为-5.0 集群上新集合的默认压缩设置由群集参数确定`default_collection_compression`。默认情况下，该参数设置为“禁用”。
- 压缩现有集合：可以使用 `collMod` 命令更改现有集合的压缩设置。
- 更改压缩阈值：默认压缩阈值为 2 KB。可以使用 `createCollection` 命令为新集合指定此值，也可以使用 `collMod` 命令为现有集合更改此值。

> **Note**
>
> Amazon DocumentDB 版本 3.6 和 4.0 不支持 Amazon DocumentDB 文档压缩。

主题

- 管理文档压缩
- 监控文档压缩

管理文档压缩

在集合中启用文档压缩

使用db.createCollection()以下方法在 Amazon DocumentDB 5.0 上创建馆藏时启用文档压缩：

```
db.createCollection( sample_collection,{
    storageEngine : {
        documentDB: {
            compression:{enable: <true | false>}
        }
    }
})
```

在集群中启用文档压缩

默认情况下，可以通过将集群参数 default_collection_compression 设置为“启用”，为所有新集合启用集群级的文档压缩。当此参数设置为“启用”时，集群上新创建的集合将默认启用压缩功能，压缩阈值为 2 KB。

压缩现有集合

您还可以使用 collMod 操作并指定以下 storageEngine 配置来修改现有集合的压缩设置。请注意，使用此命令所做的更改将仅适用于新 inserted/updated 文档，以前插入的文档的压缩效果不会改变。

```
db.runCommand({
    collMod: "orders",
    storageEngine: {
        documentDB: {compression: {enable: <true | false>} }
    }
})
```

设置压缩阈值

在默认情况下，压缩集合的压缩阈值为 2032 字节。在创建一个启用压缩的新集合时，可以在 createCollection 命令中设置此阈值：

```
db.createCollection( sample_collection, {
    storageEngine : {
        documentDB: {
```

```
            compression: {
                enable: true,
                threshold: <128 - 8000>
            }
        }
    }
})
```

您还可以使用 collMod 操作并指定以下 storageEngine 配置来修改现有压缩集合的压缩阈值：

```
db.runCommand({
    collMod: "orders",
    storageEngine: {
        documentDB: {
            compression: {
                enable: true,
                threshold: <128 - 8000>
            }
        }
    }
})
```

请注意，压缩阈值只能设置为 128 到 8000 字节之间的值。此外，在指定压缩阈值时，enable 选项需要设置为“true”。

监控文档压缩

您可以检查某集合是否已压缩如下计算其压缩率。

通过运行来自 mongo Shell 的 db.printCollectionStats() 或 db.collection.stats() 命令查看压缩统计量。输出向您显示原始大小和压缩后的大小，您可以比较这些大小来分析来自文档压缩的存储增益。在此示例中，集合“sample_collection”的统计量显示如下：下面使用 1024*1024 的缩放因子来输出 size 和 storageSize 值，单位为 MB。

```
db.sample_collection.stats(1024*1024)
```

以下是上述命令的输出示例：

```
{
    "ns" : "test.sample_collection",
    "count" : 1000000,
    "size" : 3906.3,
```

```
    "avgObjSize" : 4096,
    "storageSize" : 1953.1,
    compression:{"enabled" : true,"threshold" : 2032},
    ...
}
```

- 大小 -文档集合原始大小。
- avg ObjSize-压缩前的平均文档大小四舍五入到小数点后第一位。计量单位是字节。
- storageSize：压缩后集合的存储空间大小。计量单位是字节。
- 启用：指示压缩是启用还是禁用。

要计算实际压缩率，请将集合大小除以存储大小 (size/storageSize)。对于上面的示例，计算结果为 3906。 3/1953.1 表示压缩比为 2:1。

在 Amazon DocumentDB 8.0 中管理基于字典的压缩

亚马逊 DocumentDB 8.0 引入了一种新的文档压缩算法 zstd，作为 LZ4 的改进替代方案。当您通过选择 Zstd 作为压缩算法对 Amazon DocumentDB 8.0 集合启用字典压缩时，将对您的馆藏中的文档进行采样以创建自定义压缩字典。使用此字典和 zstd 算法压缩新的和更新的文档。与标准压缩方法相比，这种方法通常可以实现更高的压缩率，特别是对于具有一致文档架构或重复字段名的集合。

Lz4 是一种专为快速压缩和解压缩而设计的算法。众所周知，它对CPU的影响很小，同时可以实现明显的压缩。Zstd 是一种通用算法，在默认设置下，它使用的 CPU 更多，但压缩率比 lz4 更好。字典的使用进一步提高了大多数 JSON 文档的压缩率。Zstd 算法的一些优点是：

- 降低存储成本：更高的压缩比意味着更少的存储使用量和更低的成本。
- 较低 I/O：压缩后的文档需要更少 I/O，从而有可能提高性能。
- 针对您的馆藏进行了优化：该词典专门针对您的馆藏的数据模式进行了训练。

> **Note**
>
> Dictionary-based 亚马逊 DocumentDB 3.6、4.0 和 5.0 版本不支持压缩。

性能注意事项

Zstd 压缩涉及以下权衡：

- 存储与 CPU：与 LZ4 压缩相比，Zstd 压缩可实现更好的压缩率，但使用的 CPU 资源可能会稍多一些。
- 初始压缩：在插入足够的文档来训练有效的字典之前，新集合可能无法实现最佳压缩。当前，如果字典收集了至少 100 个文档，则会对其进行训练。
- 工作 Read-intensive 负载类型：由于解压缩开销，将全部数据存入缓冲区缓存的工作负载的延迟和 CPU 使用率可能会增加。

Zstd 压缩对于包含小型文档、文档数组和重复字段名的集合特别有效。

启用基于字典的压缩

对于新的集合，您可以使用以下命令启用 Zstd 压缩：

```
db.createCollection("myCollection",
    {
        storageEngine: {
            documentDB: {
                compression: {
                    enable: true,
                    algorithm: "zstd"
                }
            }
        }
    }
 )
```

您也可以对现有集合启用或修改压缩：

```
db.runCommand({
    collMod: "myCollection",
    storageEngine: {
        documentDB: {
            compression: {
                enable: true,
                algorithm: "zstd"
            }
        }
    }
```

```
})
```

要在集群上的所有集合中启用 Zstd 算法，您可以修改集群参数组以选择 “zstd” 作为参数 “default_collection_compression” 的值。

开始使用

亚马逊 DocumentDB 8.0 默认开启 Zstd 压缩。您可以随时通过在集群参数组中将 “default_compression” 的值设置为禁用来将其关闭。必须注意的是，从 Amazon DocumentDB 8.0 开始，“启用” 不再是默认压缩的有效选择，您必须从 Zstd 和 LZ4 中进行选择。

监控

您可以使用以下命令之一查看集合的压缩信息：

- db.runCommand ({collstats: “myCollection”}) 或
- db.collection.stats ()

以下命令返回可用于计算压缩率的密钥统计信息：

- 压缩算法：使用的算法（“lz4” 或 “zstd”）
- 存储大小：压缩后集合使用的实际存储空间。请注意，此数字包括碎片（即数据库页面中未使用的空间）
- avgObjSize：解压缩后的集合文档的平均逻辑大小。请注意，如果您的馆藏包含超过 2 万个文档，则此数字将是基于 2 万个文档样本的近似值。
- size：未压缩的集合的逻辑大小。这个数字是通过将平均值乘以集合中的文档总数ObjSize 来获得的，因此，如果 avg ObjSize 是近似值，则该数字也将是近似值。
- 计数：馆藏中的文档数

在评估基于字典的压缩时，以下 CloudWatch 指标可能会有所帮助：

- CPUUtilization
- FreeableMemory
- VolumeBytesUsed
- VolumeReadIOPs
- VolumeWriteIOPs

CollStats 指标：

- storageSize
- size

此外，跟踪特定于您的应用程序的指标（例如延迟和每种查询类型或 API 的吞吐量）也很有用。

管理 Amazon DocumentDB 事件

Amazon DocumentDB（与 MongoDB 兼容）记录与集群、实例、快照、安全组和集群参数组相关的事件。此信息包含事件的日期和时间、事件的源名称和源类型以及一条与事件相关的消息。

> **Important**
>
> 对于某些管理功能，Amazon DocumentDB 使用与 Amazon RDS 和 Amazon Neptune 共享的操作技术。区域限制是指在区域级别管理，在 Amazon DocumentDB、Amazon RDS 和 Amazon Neptune 之间共有的限制。有关更多信息，请参阅 区域配额。

主题

- 查看 Amazon DocumentDB 事件类别
- 查看 Amazon DocumentDB 事件

查看 Amazon DocumentDB 事件类别

每种 Amazon DocumentDB 资源类型具有与其关联的特定类型的事件。您可以使用该 AWS CLI `describe-event-categories`操作来查看事件类型与 Amazon DocumentDB 资源类型之间的映射关系。

参数

- **`--source-type`**：可选。可以使用 `--source-type` 参数查看特定源类型的事件类别。以下是允许的值：
  - `db-cluster`
  - `db-instance`
  - `db-parameter-group`

  - db-security-group
  - db-cluster-snapshot
- **--filters**：可选。要仅查看 Amazon DocumentDB 的事件类别，请使用筛选器 --filter Name=engine,Values=docdb。

Example

以下代码列出了与集群关联的事件类别。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-event-categories \
    --filter Name=engine,Values=docdb \
    --source-type db-cluster
```

对于 Windows：

```
aws docdb describe-event-categories ^
    --filter Name=engine,Values=docdb ^
    --source-type db-cluster
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "EventCategoriesMapList": [
        {
            "EventCategories": [
                "notification",
                "failure",
                "maintenance",
                "failover"
            ],
            "SourceType": "db-cluster"
        }
    ]
}
```

以下代码列出了与每种 Amazon DocumentDB 源类型关联的事件类别。

```
aws docdb describe-event-categories
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "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"
            ]
        }
    ]
}
```

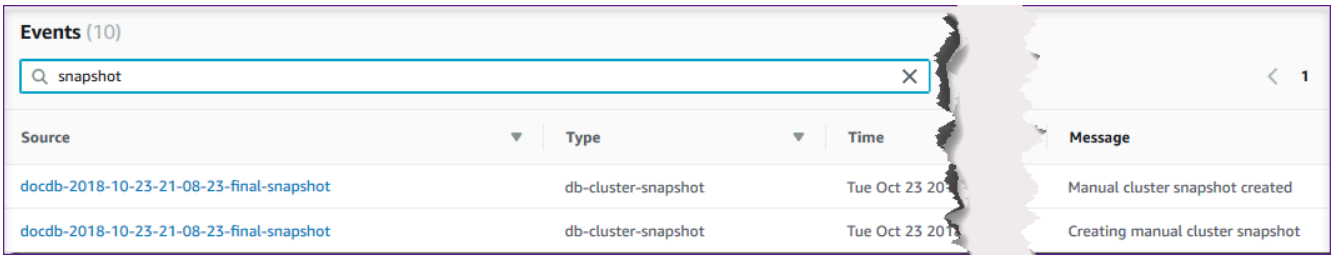
查看 Amazon DocumentDB 事件

您可通过 Amazon DocumentDB 控制台（显示过去 24 小时的事件）检索 Amazon DocumentDB 资源的事件。您还可以使用 d escribe-events AWS CLI 命令或 Amazon DocumentDB API 操作来检索亚马逊文档DescribeEvents数据库资源的事件。如果您使用 AWS CLI 或 Amazon DocumentDB API 查看事件，则最多可以检索过去 14 天的事件。

Using the AWS 管理控制台

查看过去 24 小时的所有 Amazon DocumentDB 实例事件

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择 Events (事件)。列表中显示可用的事件。
3. 使用 Filter (筛选条件) 列表按类型筛选事件。在文本框中输入期限以进一步筛选结果。例如，以下屏幕截图显示针对快照事件筛选所有 Amazon DocumentDB 事件。



Using the AWS CLI

查看过去 7 天的所有 Amazon DocumentDB 实例事件

通过运行描述事件 AWS CLI 操作，将--duration参数设置为（10,080 分钟），您可以查看过去 7 天内的所有 Amazon DocumentDB 实例事件。10080

```
aws docdb describe-events --duration 10080
```

筛选 Amazon DocumentDB 事件

要查看特定 Amazon DocumentDB 事件，请使用带有以下参数的 `describe-events` 操作。

参数

- **`--filter`**：必需，用于限制 Amazon DocumentDB 事件的返回值。使用 **`Name=engine,Values=docdb`** 仅筛选 Amazon DocumentDB 的所有事件。
- **`--source-identifier`**：可选。为其返回事件的事件源的标识符。如果省略，则在结果中包含来自所有源的事件。
- **`--source-type`**：可选，除非提供 `--source-identifier`，则为必需。如果提供 `--source-identifier`，则 `--source-type` 必须与 `--source-identifier` 的类型一致。以下是允许的值：
  - `db-cluster`
  - `db-instance`
  - `db-parameter-group`
  - `db-security-group`
  - `db-cluster-snapshot`

以下示例列出您的所有 Amazon DocumentDB 事件。

```
aws docdb describe-events --filters Name=engine,Values=docdb
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "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": [
```

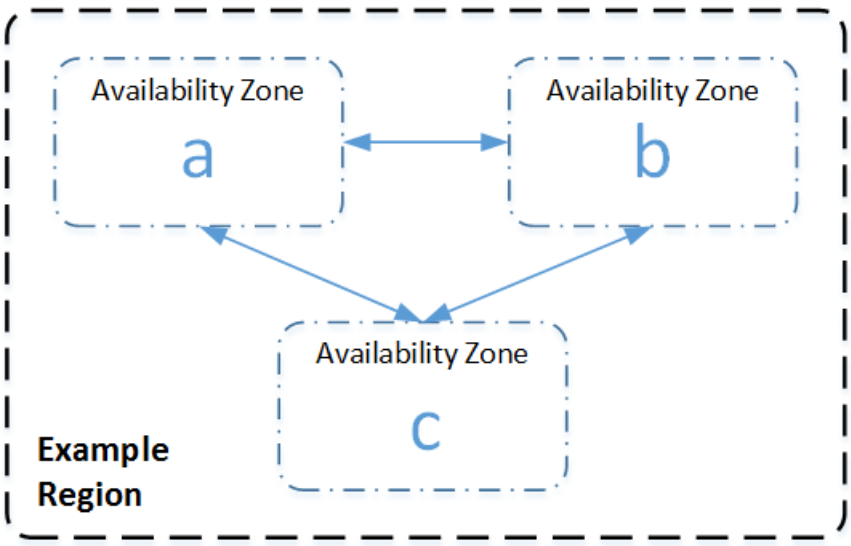
```
                "creation"
            ]
        },
        {
            "SourceArn": "arn:aws:rds:us-
east-1:123SAMPLE012:db:docdb-2018-12-11-21-08-23",
            "Message": "instance shutdown",
            "SourceType": "db-instance",
            "Date": "2018-12-11T21:25:01.245Z",
            "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"
            ]
        }
    ]
}
```

有关更多信息，请参阅 审核 Amazon DocumentDB 事件。

选择区域和可用区

Amazon 云计算资源在全球多个位置托管。这些位置包括 AWS 区域 和可用区。每个 AWS 区域 都是一个单独的地理区域。每个区域都有多个相互隔离的位置，称为 可用区。借助 Amazon DocumentDB，您可以将资源（如实例）和数据放置在多个位置。 AWS 区域 除非你特别这样做，否则资源不会被复制到其他地方。

Amazon 运行着具有高可用性的高级数据中心。数据中心有时会发生影响托管于同一位置的所有实例的可用性的故障，虽然这种故障极少发生。如果您将所有实例都托管在受此类故障影响的同一个位置，则您的所有实例都将不可用。下图显示了 AWS 区域 具有三个可用区域的。



需要牢记的是，每个区域都是独立的，这一点非常重要。您启动的任何 Amazon DocumentDB 活动（例如创建实例或列出可用实例）都只会在您当前的默认 AWS 区域运行。您可以通过设置 `EC2_REGION` 环境变量在控制台上更改默认区域。或者，通过使用 AWS CLI中的 `--region` 参数进行覆盖。有关更多信息，请参阅配置（具体 AWS Command Line Interface而言）有关环境变量和命令行选项的部分。

当您使用 Amazon DocumentDB 控制台创建集群并选择在不同的可用区中创建副本时，Amazon DocumentDB 会创建两个实例。它在一个可用区中创建主实例，在另一个可用区中创建副本实例。集群卷始终跨三个可用区复制。

要创建或使用特定的 Amazon DocumentDB 实例 AWS 区域，请使用相应的区域服务终端节点。

区域可用性

亚马逊 DocumentDB 可在以下 AWS 区域使用。

Amazon DocumentDB 支持的区域

区域名称	Region	可用区（计算）
美国东部（俄亥俄州）	`us-east-2`	3
美国东部（弗吉尼亚州北部）	`us-east-1`	6

区域名称	Region	可用区（计算）
美国西部（俄勒冈州）	us-west-2	4
非洲（开普敦）	af-south-1	3
南美洲（圣保罗）	sa-east-1	3
亚太地区（香港）	ap-east-1	3
亚太地区（海得拉巴）	ap-south-2	3
亚太地区（马来西亚）	ap-southeast-5	3
亚太地区（孟买）	ap-south-1	3
亚太地区（大阪）	ap-northeast-3	3
亚太地区（首尔）	ap-northeast-2	4
亚太地区（新加坡）	ap-southeast-1	3
亚太地区（悉尼）	ap-southeast-2	3
亚太地区（雅加达）	ap-southeast-3	3
亚太地区（墨尔本）	ap-southeast-4	3
亚太地区（泰国）	ap-southeast-7	3
亚太地区（东京）	ap-northeast-1	3
加拿大（中部）	ca-central-1	3
加拿大西部（卡尔加里）	ca-west-1	3
中国（北京）区域	cn-north-1	3

区域名称	Region	可用区（计算）
中国（宁夏）	cn-northwest-1	3
欧洲地区（法兰克福）	eu-central-1	3
欧洲（苏黎世）	eu-central-2	3
欧洲地区（爱尔兰）	eu-west-1	3
欧洲地区（伦敦）	eu-west-2	3
欧洲地区（米兰）	eu-south-1	3
欧洲地区（巴黎）	eu-west-3	3
欧洲（西班牙）	eu-south-2	3
欧洲地区（斯德哥尔摩）	eu-north-1	3
墨西哥（中部）	mx-central-1	3
中东（阿联酋）：	me-central-1	3
以色列（特拉维夫）	il-central-1	3
AWS GovCloud (US-West)	us-gov-west-1	3
AWS GovCloud (US-East)	us-gov-east-1	3

默认情况下，Amazon DocumentDB 数据库群集的时区是协调世界时 (UTC)。

有关查找特定区域中集群和实例的连接端点的信息，请参阅了解 Amazon DocumentDB 端点。

管理 Amazon DocumentDB 集群参数组

您可以通过使用集群参数组中的参数来管理 Amazon DocumentDB 引擎配置。集群参数组是 Amazon DocumentDB 配置值的集合，这个集合让管理 Amazon DocumentDB 集群的参数更轻松。集群参数组就像是引擎配置值的容器，这些值可应用于集群中的所有实例。

本节介绍如何创建、查看和修改集群参数组。它还介绍了您可以怎样确定哪个集群参数组与给定集群关联。

主题

- 描述 Amazon DocumentDB 集群参数组
- 创建 Amazon DocumentDB 集群参数组
- 修改 Amazon DocumentDB 集群参数组
- 修改 Amazon DocumentDB 集群以使用自定义集群参数组
- 复制 Amazon DocumentDB 集群参数组
- 重置 Amazon DocumentDB 集群参数组
- 删除 Amazon DocumentDB 集群参数组
- Amazon DocumentDB 集群参数参考

描述 Amazon DocumentDB 集群参数组

当您在新引擎中 AWS 区域 或使用新引擎创建第一个 Amazon DocumentDB 集群时，会自动创建集群参数组。`default`在同一区域创建且引擎版本相同的后续集群是使用`default`集群参数组创建的。

主题

- 描述 Amazon DocumentDB 集群参数组详情
- 确定 Amazon DocumentDB 集群的参数组
- 确定与 Amazon DocumentDB 集群参数组关联的集群和实例

描述 Amazon DocumentDB 集群参数组详情

要描述给定群集参数组的详细信息，请使用 AWS 管理控制台 或 AWS Command Line Interface (AWS CLI) 完成以下步骤。

Using the AWS 管理控制台

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择参数组。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在集群参数组窗格中，选择要查看其详细信息的参数组的名称。
4. 所得页面显示参数组的参数、最近活动和标签。

   - 在集群参数下，您可以看到该参数的名称、当前值、允许值、该参数是否可修改、其应用值、数据类型和描述。您可以修改各个参数，方法是：选择该参数，然后在集群参数部分，选择编辑。有关更多信息，请参阅 修改 Amazon DocumentDB 集群参数。
   - 在近期事件下，您可以看到该参数组的最新事件。您可以使用本部分的搜索栏筛选这些事件。有关更多信息，请参阅 管理 Amazon DocumentDB 事件。
   - 在 Tags (标签)下，您可以看到此集群参数组上的标签。您可以通过在标签部分选择编辑来添加或移除标签。有关更多信息，请参阅 标记 Amazon DocumentDB 资源。

Using the AWS CLI

您可以使用`describe-db-cluster-parameter-groups` AWS CLI 命令查看您为 Amazon DocumentDB 拥有的单个集群参数组或所有集群参数组的亚马逊资源名称 (ARN)、系列、描述和名称。您还可以使用`describe-db-cluster-parameters` AWS CLI 命令查看单个集群参数组中的参数及其详细信息。

- **`--describe-db-cluster-parameter-groups`**：查看所有集群参数组的列表及其详细信息。
  - **`--db-cluster-parameter-group-name`**：可选。您想描述的集群参数组的名称。如果忽略此参数，则将描述所有集群参数组。
- **`--describe-db-cluster-parameters`**：列出参数组中的所有参数及其值。
  - **`--db-cluster-parameter-group name`**：必需。您想描述的集群参数组的名称。

Example

以下代码列出了最多 100 个集群参数组及其 ARN、族、描述和名称。

```
aws docdb describe-db-cluster-parameter-groups
```

此操作的输出将类似于下文（JSON 格式）。

```
{
          "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

以下代码列出了 sample-parameter-group 的 ARN、族、描述和名称。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameter-groups \
          --db-cluster-parameter-group-name sample-parameter-group
```

对于 Windows：

```
aws docdb describe-db-cluster-parameter-groups ^
          --db-cluster-parameter-group-name sample-parameter-group
```

此操作的输出将类似于下文（JSON 格式）。

```
{
          "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

以下代码列出了 sample-parameter-group 中参数的值。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

对于 Windows：

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name sample-parameter-group
```

此操作的输出将类似于下文（JSON 格式）。

```
{
   "Parameters": [
         {
            "ParameterName": "audit_logs",
            "ParameterValue": "disabled",
            "Description": "Enables auditing on cluster.",
            "Source": "system",
```

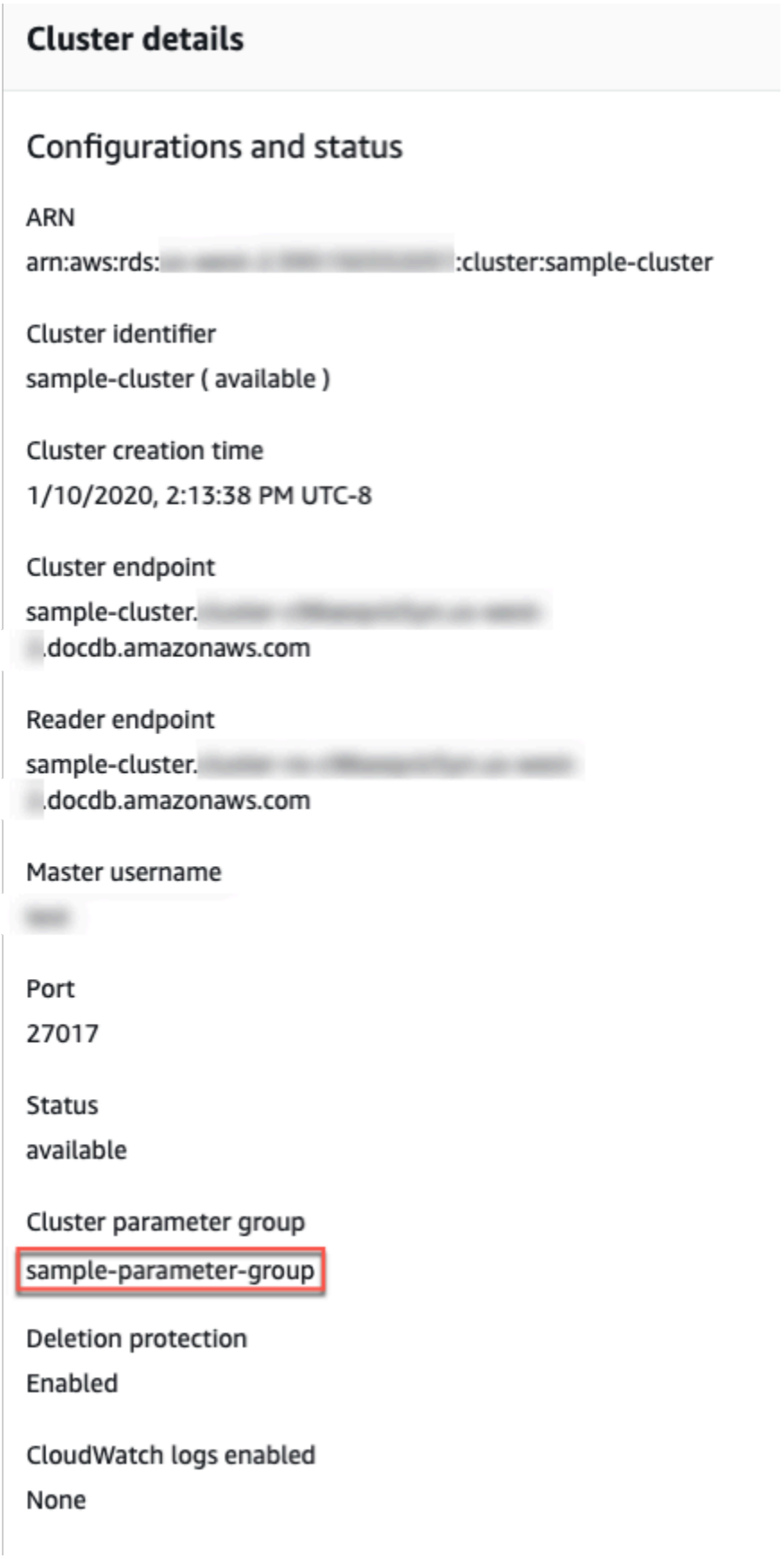
```
            "ApplyType": "dynamic",
            "DataType": "string",
            "AllowedValues": "enabled,disabled",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot"
         },
         {
            "ParameterName": "change_stream_log_retention_duration",
            "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"
         }
   ]
}
```

确定 Amazon DocumentDB 集群的参数组

要确定哪个参数组与特定集群相关联，请使用 AWS 管理控制台 或完成以下步骤 AWS CLI。

Using the AWS 管理控制台

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在左侧导航窗格中，选择集群。
3. 从集群列表中，选择您感兴趣的集群的名称。
4. 生成的页面将显示所选集群的详细信息。向下滚动到 Cluster details (集群详细信息)。在此部分的底部，在 Cluster parameter group (集群参数组) 的下方找到参数组的名称。


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

以下 AWS CLI 代码确定哪个参数组正在控制集群sample-cluster。

```
aws docdb describe-db-clusters \
      --db-cluster-identifier sample-cluster \
      --query 'DBClusters[*].[DBClusterIdentifier,DBClusterParameterGroup]'
```

此操作的输出将类似于下文（JSON 格式）。

```
[
       [
           "sample-cluster",
           "sample-parameter-group"
       ]
]
```

确定与 Amazon DocumentDB 集群参数组关联的集群和实例

要确定与每个参数组关联的集群和实例数量，请使用完成以下步骤 AWS CLI。

Using the AWS CLI

以下 AWS CLI 代码确定了与多少集群和实例相关联sample-parameter-group。

对于 Linux、macOS 或 Unix：

```
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
      }) | .[]'
```

对于 Windows：

```
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
      }) | .[]'
```

此操作的输出将类似于下文（JSON 格式）。

```
{
   "parameter_group_name": "sample-parameter-group",
   "total_clusters": 50,
   "total_instances": 150
}
```

创建 Amazon DocumentDB 集群参数组

默认集群参数组（例如default.docdb5.0default.docdb4.0default.docdb3.6、或）是在使用新引擎版本和新引擎版本创建集群时创建的 AWS 区域。在同一区域创建的具有相同引擎版本的后续default集群将继承集群参数组。一经创建，default 参数组就无法删除或重命名。您可以通过创建带有首选参数值的自定义参数组并将其附加到 Amazon DocumentDB 集群来修改集群实例的引擎行为。

以下过程指导您完成集群参数组的创建。然后，您可以修改该参数组中的参数。

> **Note**
>
> 创建一个集群参数组之后，您应至少等 5 分钟，再使用特定的集群参数组。这样，在将集群参数组用于新集群之前，Amazon DocumentDB 可以完成全部 create 操作。您可以使用 AWS 管理控制台 或describe-db-cluster-parameter-groups AWS CLI 操作来验证您的集群参数组是否已创建。有关更多信息，请参阅 描述 Amazon DocumentDB 集群参数组。

Using the AWS 管理控制台

要创建集群参数组

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择参数组。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在集群参数组窗格中，选择创建。
4. 在创建集群参数组窗格中，输入以下内容：

   a. 新集群参数组名称：为集群参数组输入一个名称。例如 `sample-parameter-group`。集群参数组具有以下命名限制：

      - 长度为 [1-255] 个字母数字字符。
      - 第一个字符必须是字母。
      - 不能以连字符结尾或包含两个连续的连字符。

   b. 系列：选择要用于集群的 DocumentDB 版本。

   c. 描述：为此集群参数组提供一个描述。
5. 要创建集群参数组，请选择创建。要取消操作，请选择取消。
6. 选择创建后，以下文本将显示于页面顶部，以验证集群参数组是否已成功创建：

```
Successfully created cluster parameter group 'sample-parameter-group'.
```

Using the AWS CLI

要为 Amazon DocumentDB 4.0 集群创建新的集群参数组，请使用带有以下参数的 AWS CLI `create-db-cluster-parameter-group`操作：

- **`--db-cluster-parameter-group-name`**：自定义集群参数组的名称。例如 `sample-parameter-group`。
- **`--db-cluster-parameter-group-family`**：用作自定义集群参数组模板的集群参数组族。

- **--description**：用户提供的此集群参数组的描述。下面的示例使用了“Custom docdb4.0 parameter group”。

对于 Linux、macOS 或 Unix：

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"
```

对于 Windows：

```
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"
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "sample-parameter-group",
        "DBParameterGroupFamily": "docdb5.0",
        "Description": "Custom docdb4.0 parameter group",
        "DBClusterParameterGroupArn": "sample-parameter-group-arn"
    }
}
```

修改 Amazon DocumentDB 集群参数组

本节介绍如何修改自定义 Amazon DocumentDB 参数组。在 Amazon DocumentDB 中，您无法修改在新区域中首次使用新引擎版本创建集群时创建的集群参数组。default如果您的 Amazon DocumentDB 集群正使用默认集群参数组，而您想要修改其中的值，则您必须首先创建新的参数组或复制现有参数组，修改它，并且将修改的参数组应用于您的集群。

请完成以下步骤以修改自定义集群参数组。修改操作可能耗费一段时间来传播。等待修改后的集群参数组可用，然后再将其附加到您的集群。您可以使用 AWS 管理控制台 或 AWS CLI describe-db-

`cluster-parameters`操作来验证您的集群参数组是否已被修改。有关更多信息，请参阅 描述集群参数组。

Using the AWS 管理控制台

遵循以下步骤修改自定义 Amazon DocumentDB 参数组。您无法修改 `default` 参数组。如果要修改 `default` 参数组中的值，您可以复制默认的集群参数组，修改它，然后将修改后的参数组应用于您的集群。有关将参数组应用于您的集群的更多信息，请参阅 修改 Amazon DocumentDB 集群。

修改自定义集群参数组

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在控制台左侧的导航窗格中，选择参数组。在参数组列表中，选择要修改的参数组的名称。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 对于要修改的参数组中的每个参数，执行以下操作：

   a. 找到要修改的参数，并通过核查该参数是否在可修改列下作为 `true` 列出，验证该参数是否可修改。

   b. 如果该参数可修改，请选择它并从控制台页面的右上角选择编辑。

   c. 在修改 **<parameter-name>** 对话框中，进行所需的更改。然后，选择修改集群参数或选择取消以放弃更改。

Using the AWS CLI

您可以使用 AWS CLI，修改 Amazon DocumentDB 集群参数组中任何可修改参数的 `ParameterValue`、`Description` 或 `ApplyMethod`。您无法直接对默认的集群参数组进行修改。

要修改自定义集群参数组的参数，请使用带以下参数的 `modify-db-cluster-parameter-group` 操作。

- **`--db-cluster-parameter-group-name`**：必需。您正在修改的集群参数组的名称。

- **--parameters**：必需。您正在修改的参数。有关适用于 Amazon DocumentDB 集群中所有实例的参数列表，请参阅 Amazon DocumentDB 集群参数参考。每个参数条目必须包含以下内容：
  - **ParameterName**：您正在修改的参数的名称。
  - **ParameterValue**：此参数的新值。
  - **ApplyMethod**：您希望如何应用对此参数的更改。允许的值为 `immediate` 和 `pending-reboot`。

> **Note**
>
> 带 `static` 的 `ApplyType` 参数必须具有 `pending-reboot` 的 `ApplyMethod`。

Example- 修改参数的值

在此示例中，您列出 `sample-parameter-group` 的参数值并修改 `tls` 参数。然后，等待 5 分钟后，再次列出 `sample-parameter-group` 的参数值以查看更改后的参数值。

1. 列出 `sample-parameter-group` 的参数及其值。

   对于 Linux、macOS 或 Unix：

   ```
   aws docdb describe-db-cluster-parameters \
          --db-cluster-parameter-group-name sample-parameter-group
   ```

   对于 Windows：

   ```
   aws docdb describe-db-cluster-parameters ^
          --db-cluster-parameter-group-name sample-parameter-group
   ```

   此操作的输出将类似于下文（JSON 格式）。

   ```
   {
          "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. 修改 `tls` 参数，以便其值为 `disabled`。

   您无法修改 `ApplyMethod`，因为 `ApplyType` 是 `static`。

   对于 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group \
       --parameters
 "ParameterName"=tls,"ParameterValue"=disabled,"ApplyMethod"=pending-reboot
```

   对于 Windows：

```
aws docdb modify-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group ^
       --parameters
 "ParameterName"=tls,"ParameterValue"=disabled,"ApplyMethod"=pending-reboot
```

   此操作的输出将类似于下文（JSON 格式）。

```
{
       "DBClusterParameterGroupName": "sample-parameter-group"
   }
```

3. 至少等 5 分钟。
4. 列出 `sample-parameter-group` 的参数值以验证 `tls` 参数是否修改过。

   对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameters \
       --db-cluster-parameter-group-name sample-parameter-group
```

   对于 Windows：

```
aws docdb describe-db-cluster-parameters ^
       --db-cluster-parameter-group-name sample-parameter-group
```

   此操作的输出将类似于下文（JSON 格式）。

```
{
       "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"
           }
       ]
}
```

修改 Amazon DocumentDB 集群以使用自定义集群参数组

当您创建 Amazon DocumentDB 集群时，将自动为该集群创建一个 `default.docdb4.0` 参数值。您不能修改 `default` 集群参数组。相反，您可以修改您的 Amazon DocumentDB 集群，以将新的定制化参数组与其关联。

本节介绍如何使用 AWS 管理控制台 和 AWS Command Line Interface ()AWS CLI修改现有 Amazon DocumentDB 集群以使用自定义集群参数组。

Using the AWS 管理控制台

修改 Amazon DocumentDB 集群以使用新的非默认集群参数组

1. 开始之前，请确保您已创建一个 Amazon DocumentDB 集群和集群参数组。有关更多说明，请参阅 创建 Amazon DocumentDB 集群 和 创建 Amazon DocumentDB 集群参数组。
2. 创建集群参数组后，请在 https://console.aws.amazon.com/docdb 打开 Amazon DocumentDB 控制台。在导航窗格中，选择 集群以将新参数组添加到集群。
3. 选择您想将其与您的参数组关联起来的集群。选择操)，然后选择 修改以修改您的集群。
4. 在集群选项下，选择要与您的集群关联的新参数组。
5. 选择 Continue (继续) 以查看修改摘要。
6. 在确认您的更改后，您可以立即应用这些更改，也可以在 Scheduling of modifications (修改计划) 下的下一个维护时段内应用这些更改。
7. 选择 Modify cluster (修改集群) 以使用新参数组更新您的集群。

Using the AWS CLI

开始之前，请确保您已创建一个 Amazon DocumentDB 集群和集群参数组。您可以使用操作创建 Amazon DocumentDB 集群。 AWS CLI `create-db-cluster`您可以使用 AWS CLI `create-db-cluster-parameter-group`操作创建集群参数组。

要将新的集群参数组添加到集群，请使用带有以下参数的 AWS CLI `modify-db-cluster`操作。

- --db-cluster-identifier：您的集群的名称（例如，`sample-cluster`）。
- --db-cluster-parameter-group-name：您要与自己的集群关联的参数组的名称（例如，`sample-parameter-group`）。

Example

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --db-cluster-parameter-group-name sample-parameter-group
```

此操作的输出将类似于下文（JSON 格式）。

```
"DBCluster": {
    "AvailabilityZones": [
            "us-west-2c",
            "us-west-2b",
            "us-west-2a"
    ],
    "BackupRetentionPeriod": 1,
    "DBClusterIdentifier": "sample-cluster",
    "DBClusterParameterGroup": "sample-parameter-group",
    "DBSubnetGroup": "default",
   ...
}
```

复制 Amazon DocumentDB 集群参数组

您可以使用 AWS 管理控制台 或 AWS Command Line Interface ()AWS CLI复制 Amazon DocumentDB 中的集群参数组。

Using the AWS 管理控制台

以下规程指导您通过复制现有集群参数组创建新的集群参数组。

复制一个集群参数组

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择参数组。
3. 在集群参数组窗格中，选择您要复制的集群参数组的名称。
4. 选择操作，然后选择复制以复制该参数组。
5. 在复制选项下，输入新集群参数组的名称和描述。然后选择复制以保存更改。

Using the AWS CLI

要复制一个集群参数组，请使用带以下参数的 `copy-db-cluster-parameter-group` 操作。

- **`--source-db-cluster-parameter-group-identifier`**：必需。您想要复制的集群参数组的名称或 Amazon 资源名称（ARN）。

  如果源集群参数组和目标集群参数组相同 AWS 区域，则标识符可以是名称或 ARN。

  如果源集群参数组和目标集群参数组不同 AWS 区域，则标识符必须是 ARN。

- **`--target-db-cluster-parameter-group-identifier`**：必需。集群参数组副本的名称或 ARN。

  约束：

  - 不能为 null 或空。
  - 必须包含 1-255 个字母、数字或连字符。
  - 第一个字符必须是字母。
  - 不能以连字符结尾或包含两个连续的连字符。

- **`--target-db-cluster-parameter-group-description`**：必需。用户提供的对集群参数组的描述。

Example

以下代码将创建 `sample-parameter-group` 的副本，将副本命名为 `sample-parameter-group-copy`。

对于 Linux、macOS 或 Unix：

```
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"
```

对于 Windows：

```
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"
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "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"
    }
}
```

重置 Amazon DocumentDB 集群参数组

您可以使用或 AWS Command Line Interface (AWS CLI) 重置集群参数组，将 Amazon DocumentDB 集群参数组的部分 AWS 管理控制台 或全部参数值重置为默认值。

Using the AWS 管理控制台

按照以下步骤将将部分或全部集群参数组的参数值重置为默认值。

要重置集群参数组的参数值

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在控制台左侧的导航窗格中，选择参数组。
3. 在集群参数组窗格中，选择要对其复置的集群参数组的名称。
4. 选择操作，然后选择重置以重置该参数组。
5. 在所得的集群参数组重置确认页面上，确认您想要将该参数组的所有集群参数重置为默认值。然后，选择重置以重置您的参数组。您也可以选择取消以放弃所做更改。

Using the AWS CLI

要将部分或全部集群参数组的参数值重置为其默认值，请使用带有以下参数的 `reset-db-cluster-parameter-group` 操作。

- **`--db-cluster-parameter-group-name`**：必需。要重置的集群参数组的名称。

- **--parameters**：可选。集群参数组中要重置为其默认值的 ParameterName 和 ApplyMethod 列表。静态参数必须设置成 pending-reboot 才能在下次实例重启或 reboot-db-instance 请求时生效。对于集群中您希望更新的静态参数应用于的每个实例，您必须调用 reboot-db-instance。

  此参数和 --reset-all-parameters 相互排斥：您可以使用这两个参数之一，但不能同时使用二者。

- **--reset-all-parameters** 或 **--no-reset-all-parameters**：可选。指定是否将所有参数 (--reset-all-parameters) 或仅部分参数 (--no-reset-all-parameters) 重置为其默认值。--reset-all-parameters 参数和 --parameters 相互排斥：您可以使用这两个参数之一，但不能同时使用二者。

  当您重置整个组时，动态参数立即更新。将静态参数设置成 pending-reboot，以便在下次实例重启或 reboot-db-instance 请求时生效。对于集群中您希望更新的静态参数应用于的每个实例，您必须调用 reboot-db-instance。

Example

示例 1：将所有参数重置为其默认值

以下代码将集群参数组 sample-parameter-group 中的所有参数重置成它们的默认值。

对于 Linux、macOS 或 Unix：

```
aws docdb reset-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group \
       --reset-all-parameters
```

对于 Windows：

```
aws docdb reset-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group ^
       --reset-all-parameters
```

示例 2：将指定的参数重置为其默认值

以下代码将集群参数组 sample-parameter-group 中的 tls 参数重置成它的默认值。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此操作的输出将类似于下文（JSON 格式）。

```
{
       "DBClusterParameterGroupName": "sample-parameter-group"
   }
```

重启集群实例

在更改静态参数的值之前，必须重新启动集群实例。重启集群中您希望将更新的静态参数应用到的每个实例。

对于 Linux、macOS 或 Unix：

```
aws docdb reboot-db-instance \
       --db-instance-identifier sample-cluster-instance
```

对于 Windows：

```
aws docdb reboot-db-instance ^
       --db-instance-identifier sample-cluster-instance
```

删除 Amazon DocumentDB 集群参数组

您可以使用 AWS 管理控制台 或 AWS Command Line Interface ()AWS CLI删除自定义 Amazon DocumentDB 集群参数组。您不能删除 `default.docdb4.0` 集群参数组。

Using the AWS 管理控制台

删除集群参数组

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择参数组。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在参数组窗格中，选择要删除的集群参数组左侧的单选按钮。
4. 选择操作，然后选择删除。
5. 在删除确认窗格中，选择删除以删除集群参数组。要保留集群参数组，请选择取消。

Using the AWS CLI

要删除集群参数组，请使用带以下参数的 `delete-db-cluster-parameter-group` 操作。

- **`--db-cluster-parameter-group-name`**：必需。要删除的集群参数组的名称。它必须是现有集群参数组。您无法删除*`default.docdb4.0`* 集群参数组。

Example删除集群参数组

以下示例引导您通过三个步骤来删除集群参数组：

1. 找到您要删除的集群参数组的名称。
2. 删除指定的集群参数组。
3. 正在验证集群参数组是否已删除。

1. 找到您要删除的集群参数组的名称。

以下代码列出了所有集群参数组的名称。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameter-groups \
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

对于 Windows：

```
aws docdb describe-db-cluster-parameter-groups ^
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

上一个操作的输出是集群参数组名称的一个列表，与以下（JSON 格式）类似。

```
[
       [
           "default.docdb4.0"
       ],
       [
           "sample-parameter-group"
       ],
       [
           "sample-parameter-group-copy"
       ]
   ]
```

2. 删除特定的集群参数组。

以下代码将删除集群参数组 `sample-parameter-group-copy`。

对于 Linux、macOS 或 Unix：

```
aws docdb delete-db-cluster-parameter-group \
       --db-cluster-parameter-group-name sample-parameter-group-copy
```

对于 Windows：

```
aws docdb delete-db-cluster-parameter-group ^
       --db-cluster-parameter-group-name sample-parameter-group-copy
```

没有来自此操作的输出。

3. 验证指定的集群参数组是否已删除。

以下代码列出了所有其余集群参数组的名称。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameter-groups \
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

对于 Windows：

```
aws docdb describe-db-cluster-parameter-groups ^
       --query 'DBClusterParameterGroups[*].[DBClusterParameterGroupName]'
```

上一个操作的输出是集群参数组的一个列表，与以下（JSON 格式）类似。您刚刚删除的集群参数组不应在列表中。

此操作的输出将类似于下文（JSON 格式）。

```
[
       [
           "default.docdb4.0"
       ],
       [
           "sample-parameter-group"
       ]
   ]
```

Amazon DocumentDB 集群参数参考

当您更改动态参数并保存集群参数组时，更改将立即应用，而不管立即应用的设置如何。当您更改静态参数并保存集群参数组时，参数更改将在您手动重启实例后生效。您可以使用 Amazon DocumentDB 控制台或通过明确调用 `reboot-db-instance` 来重启实例。

下表显示了适用于 Amazon DocumentDB 集群中所有实例的参数。

Amazon DocumentDB 集群级参数

参数	默认 值	有效值	可修改	应用类型	数据类型	说明
audit_logs	disabled	已启用、已禁用、ddl、dml_read、dml_wr	是	动态	字符串	定义是否启用 Amazon CloudWatc

参数	默认 值	有效值	可修改	应用类型	数据类型	说明
		ite、全部、无				h 审计日志。 • **enabled**—已启用 CloudWatch 审核日志。 • **disabled**—CloudWatch 审核日志已禁用。 • **ddl**：对 DDL 事件的审核已启用。 • **dml_read**：对 DML 读取事件的审计已启用。 • **dml_write**：对 DML 写入事件的审核已启用。 • **all**：对所有数据库事件的审核已启用。

参数	默认 值	有效值	可修改	应用类型	数据类型	说明
						• **none**：审核已禁用。
change_stream_log_retention_duration	10800	3600-604800	是	动态	整数	定义变更流日志保留和可以使用的以秒计的时间长度。

参数	默认 值	有效值	可修改	应用类型	数据类型	说明
default_collection_compression	disabled	已启用、已禁用（亚马逊 DocumentDB 5.0）/ zstd、lz4、无（亚马逊）DocumentDB8.0	是	动态	字符串	定义集群中新集合的默认压缩设置 • **enabled**—亚马逊 DocumentDB 5.0 中默认启用 Lz4 压缩。亚马逊 DocumentDB 8.0 中默认启用 zstd 压缩。 • **disabled, none**—默认情况下压缩处于禁用状态（在亚马逊 DocumentDB 5.0 中禁用，亚马逊 DocumentDB 8.0 中禁用）。

参数	默认 值	有效值	可修改	应用类型	数据类型	说明
profiler	disabled	启用，禁用	是	动态	字符串	为慢速操作启用分析。 • **enabled**—花费的时间超过客户定义的阈值（例如 100 毫秒）的操作将记录到 Amazon Lo CloudWatch gs 中。 • **disabled**—慢速操作不会记录到 CloudWatch 日志中。
profiler_sampling_rate	1.0	0.0-1.0	是	动态	浮点型	定义已记录操作的采样率。

参数	默认 值	有效值	可修改	应用类型	数据类型	说明
profiler_threshold_ms	100	50-2147483646	是	动态	整数	为 profiler 定义阈值。 • 所有大于的操作profiler_threshold_ms 都将记录到 CloudWatch 日志中。

参数	默认 值	有效值	可修改	应用类型	数据类型	说明
planner_version	3.0	1.0、2.0、3.0	是	动态	浮点型	定义用于查询的查询计划器版本。 • **1.0**— 在亚马逊 DocumentDB 5.0 中支持。 • **2.0**— 改进了查找和更新运算符。亚马逊 DocumentDB 5.0 支持。 • **3.0**— 对查找、更新和聚合运算符的改进。亚马逊 DocumentDB 8.0 中的默认查询计划器。

参数	默认 值	有效值	可修改	应用类型	数据类型	说明
tls	已启用	enabled、disabled、fips-140-3、tls1.2+、tls1.3+	是	静态	字符串	定义是否需要传输层安全性 (TLS) 连接。 • **enabled**：需要 TLS 连接才能进行连接。对于亚马逊 DocumentDB 5.0 及更早版本，这将启用 TLS 1.0 到 1.3，但我们建议使用 TLS 1.2 或更高版本。从次要版本 5.0.1、亚马逊 DocumentDB 8.0 及更高版本开始，亚马逊 DocumentDB 5.0 及更高版本仅支持

参数	默认 值	有效值	可修改	应用类型	数据类型	说明
						TLS 1.2 及更高版本。 • **disabled**：TLS 连接不能用于连接。 • **fips-140-3** ：要求具有联邦信息处理标准 (FIPS) 属性的 TLS 连接才连接。根据 FIPS 出版物 140-3，集群仅接受安全的连接。只有以下地区的亚马逊文档数据库 5.0（引擎版本 3.0.3727）集群才支持此功能：ca-centra

参数	默认 值	有效值	可修改	应用类型	数据类型	说明
						l-1、us-we st-2、us-e ast-1、us-east-2、us-east-2、us-gov-eas t-1、us-gov-east-1、us-gov-west-1。 • **tls1.2+**：需要使用 TLS 版本 1.2 及更高版本的 TLS 连接才能进行连接。从 Amazon DocumentDB 4.0（引擎版本 2.0.10980）和 Amazon DocumentDB 5.0（引擎版本

参数	默认 值	有效值	可修改	应用类型	数据类型	说明
						3.0.11051）开始才支持此功能。 • **tls1.3+**：需要使用 TLS 版本 1.3 及更高版本的 TLS 连接才能进行连接。从 Amazon DocumentDB 4.0（引擎版本 2.0.10980）和 Amazon DocumentDB 5.0（引擎版本 3.0.11051）开始才支持此功能。

参数	默认 值	有效值	可修改	应用类型	数据类型	说明
ttl_monitor	已启用	启用，禁用	是	动态	字符串	定义是否为集群启用有效时间 (TTL) 监控。 • **enabled**：启用 TTL 监控。 • **disabled**：禁用 TTL 监控。

修改 Amazon DocumentDB 集群参数

在 Amazon DocumentDB 中，集群参数组由适用于您在该集群中创建的所有实例的参数组成。对于自定义集群参数组，您可以随时修改参数值，也可以将所有参数值重置为您创建的参数组的默认值。本节介绍如何查看构成 Amazon DocumentDB 集群参数组的参数及其值，以及您可以如何更改或更新这些值。

参数可以是动态或静态的。当您更改动态参数并保存集群参数组时，更改将立即应用，而不管 `Apply Immediately` 的设置如何。当您更改静态参数并保存集群参数组时，参数更改仅在您手动重启实例后生效。

查看 Amazon DocumentDB 集群参数组的参数

您可以使用或查看 Amazon DocumentDB 集群的参数及其值。 AWS 管理控制台 AWS CLI

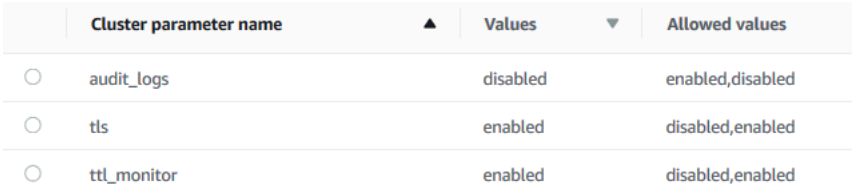
Using the AWS 管理控制台

要查看集群参数组的详细信息

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择参数组。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在 Parameter groups (参数组) 窗格中，选择要查看其详细信息的集群参数组的名称。
4. 生成的页面显示每个参数的以下值：参数的名称、当前值、允许的值、参数是否可修改、应用类型、数据类型和描述。



Cluster parameter name	Values	Allowed values
audit_logs	disabled	enabled,disabled
tls	enabled	disabled,enabled
ttl_monitor	enabled	disabled,enabled

Using the AWS CLI

要查看集群参数组的参数及其值，请使用带有以下参数的 `describe-db-cluster-parameters` 操作。

- **`--db-cluster-parameter-group-name`**：必需。您想要详细参数列表的集群参数组的名称。
- **`--source`**：可选。如果提供特定源，仅返回参数。参数源可以是 `engine-default`、`system` 或 `user`。

Example

以下代码列出了 `custom3-6-param-grp` 参数组的所有参数及其值。有关参数组的更多信息，请省略 `--query` 行。有关所有参数组的信息，请省略 `--db-cluster-parameter-group-name` 行。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameters \
   --db-cluster-parameter-group-name custom3-6-param-grp \
   --query 'Parameters[*].[ParameterName,ParameterValue]'
```

对于 Windows：

```
aws docdb describe-db-cluster-parameters ^
   --db-cluster-parameter-group-name custom3-6-param-grp ^
   --query 'Parameters[*].[ParameterName,ParameterValue]'
```

此操作的输出将类似于下文（JSON 格式）。

```
[
    [
        "audit_logs",
        "disabled"
    ],
    [
        "tls",
        "enabled"
    ],
    [
        "ttl_monitor",
        "enabled"
    ]
]
```

修改 Amazon DocumentDB 集群参数组的参数

您可以使用 AWS 管理控制台 或修改参数组的参数 AWS CLI。

Using the AWS 管理控制台

要更新集群参数组的参数

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择参数组。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在参数组 窗格中，选择要更新其参数的集群参数组的名称。
4. 所得页面显示该集群参数组的参数及它们的相应详情。选择要更新的参数。

5. 在页面右上角，选择编辑以更改参数的值。有关集群参数类型的更多信息，请参阅 Amazon DocumentDB 集群参数参考。
6. 进行更改，然后选择修改集群参数以保存更改。要放弃更改，请选择取消。

Using the AWS CLI

要修改集群参数组的参数，请使用带以下参数的 `modify-db-cluster-parameter-group` 操作：

- **`--db-cluster-parameter-group-name`**：必需。您正在修改的集群参数组的名称。
- **`--parameters`**：必需。您正在修改的参数。每个参数条目必须包含以下内容：
  - **`ParameterName`**：您正在修改的参数的名称。
  - **`ParameterValue`**：此参数的新值。
  - **`ApplyMethod`**：您希望如何应用对此参数的更改。允许的值为 `immediate` 和 `pending-reboot`。

> **Note**
>
> 带 `static` 的 `ApplyType` 参数必须具有 `pending-reboot` 的 `ApplyMethod`。

要更改群集参数组的参数值 (AWS CLI)

以下示例将更改 `tls` 参数。

1. 列出 **`sample-parameter-group`** 的参数及其值。

   对于 Linux、macOS 或 Unix：

   ```
   aws docdb describe-db-cluster-parameters \
       --db-cluster-parameter-group-name sample-parameter-group
   ```

   对于 Windows：

   ```
   aws docdb describe-db-cluster-parameters ^
       --db-cluster-parameter-group-name sample-parameter-group
   ```

   此操作的输出将类似于下文（JSON 格式）。

```
{
    "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. 修改 **tls** 参数，以便其值为 **disabled**。您无法修改 ApplyMethod，因为 ApplyType 是 static。

   对于 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name sample-parameter-group \
    --parameters
 "ParameterName"=tls,ParameterValue=disabled,ApplyMethod=pending-reboot"
```

   对于 Windows：

```
aws docdb modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name sample-parameter-group ^
```

```
    --parameters "ParameterName=tls,ParameterValue=disabled,ApplyMethod=pending-
reboot"
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "DBClusterParameterGroupName": "sample-parameter-group"
}
```

3. 至少等 5 分钟。
4. 列出 **sample-parameter-group** 的参数值。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

对于 Windows：

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name sample-parameter-group
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "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"
        }
    ]
}
```

了解 Amazon DocumentDB 端点

您可以使用 Amazon DocumentDB（与 MongoDB 兼容）端点连接到集群或实例。Amazon DocumentDB 有三种不同类型的端点，各自具有自身用途。

主题

- 查找集群的端点
- 查找实例的端点
- 连接到端点

集群端点

集群端点是用于 Amazon DocumentDB 集群的端点，连接到该集群的当前主实例。每个 Amazon DocumentDB 集群都具有单个集群端点和一个主实例。在失效转移情况下，集群端点重新映射到新的主实例。

读取器端点

读取器端点是 Amazon DocumentDB 集群的一个端点，连接到该集群的可用副本之一。每个 Amazon DocumentDB 集群都具有一个读取器端点。如果有多个副本，则读取器端点会将每个连接请求定向到 Amazon DocumentDB 副本之一。

实例端点

实例终端节点是连接到特定实例的终端节点。集群中的每个实例（不论其是主实例还是副本实例）都有各自唯一的实例终端节点。最好不要使用应用程序中的实例端点。这是因为它们在发生失效转移时会更改角色，从而需要在应用程序中更改代码。

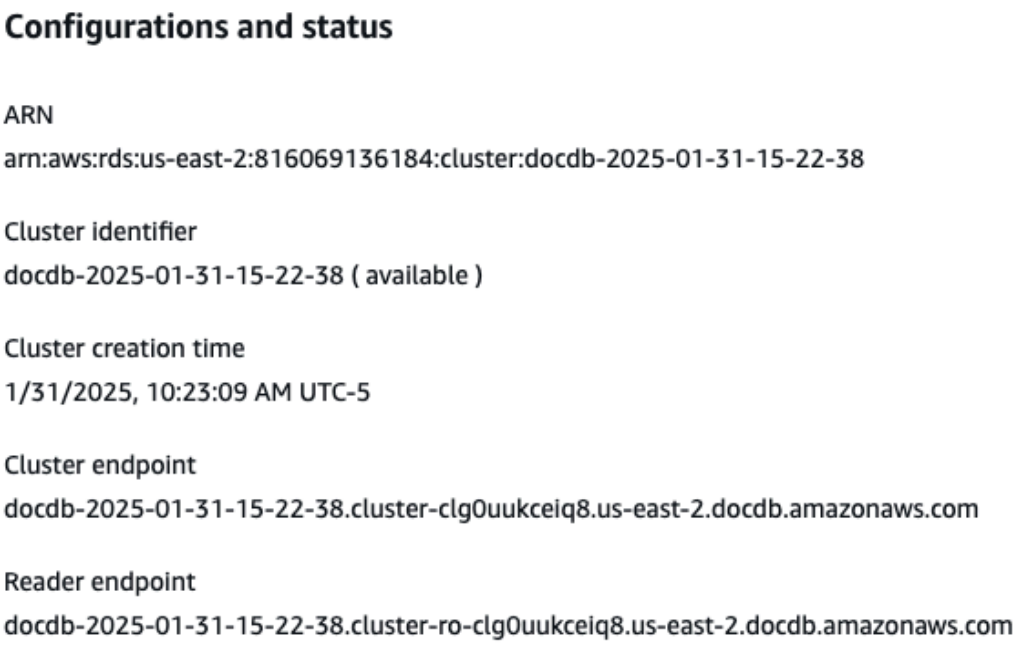
查找集群的端点

您可以使用 Amazon DocumentDB 控制台或 AWS CLI查找集群的集群端点和读取器端点。

Using the AWS 管理控制台

要使用控制台查找集群的端点：

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。
3. 从集群列表中，选择感兴趣的集群的名称。
4. 在集群详细信息页面上，选择配置选项卡。您将在配置和状态部分中找到集群端点和读取器端点。



5. 要连接到此集群，请选择连接和安全选项卡。查找 mongo Shell 的连接字符串以及可在应用程序代码中用于连接到集群的连接字符串。



Using the AWS CLI

要使用查找集群的集群和读取器终端节点 AWS CLI，请使用这些参数运行`describe-db-clusters`命令。

参数

- **`--db-cluster-identifier`**：可选。指定要返回端点的集群。如果省略，则返回多达 100 个集群的端点。
- **`--query`**：可选。指定要显示的字段。因减少您为查找端点所需查看的数据量而有益。如果省略，则返回有关集群的全部信息。
- **`--region`**：可选。使用 `--region` 参数指定要将命令应用到的区域。如果省略，则使用默认区域。

Example

以下示例将返回 `sample-cluster` 的 `DBClusterIdentifier`、终端节点（集群终端节点）和 `ReaderEndpoint`。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters \
   --region us-east-1 \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[DBClusterIdentifier,Port,Endpoint,ReaderEndpoint]'
```

对于 Windows：

```
aws docdb describe-db-clusters ^
```

```
    --region us-east-1 ^
    --db-cluster-identifier sample-cluster ^
    --query 'DBClusters[*].[DBClusterIdentifier,Port,Endpoint,ReaderEndpoint]'
```

此操作的输出将类似于下文（JSON 格式）。

```
[
  [
     "sample-cluster",
     27017,
     "sample-cluster.cluster-corlsfccjozr.us-east-1.docdb.amazonaws.com",
     "sample-cluster.cluster-ro-corlsfccjozr.us-east-1.docdb.amazonaws.com"
  ]
]
```

现在您已拥有集群端点，可以使用 mongo 或 mongodb 连接到集群。有关更多信息，请参阅 连接到端点。

查找实例的端点

您可使用 Amazon DocumentDB 控制台或 AWS CLI查找实例的端点。

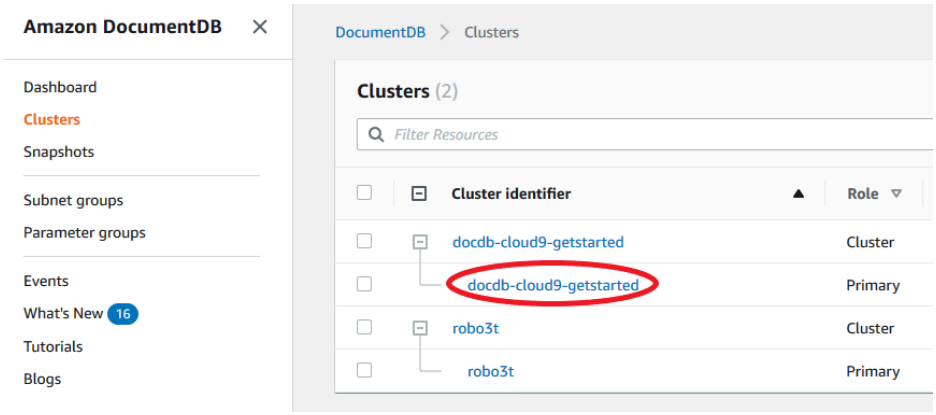
Using the AWS 管理控制台

要使用控制台查找实例的端点

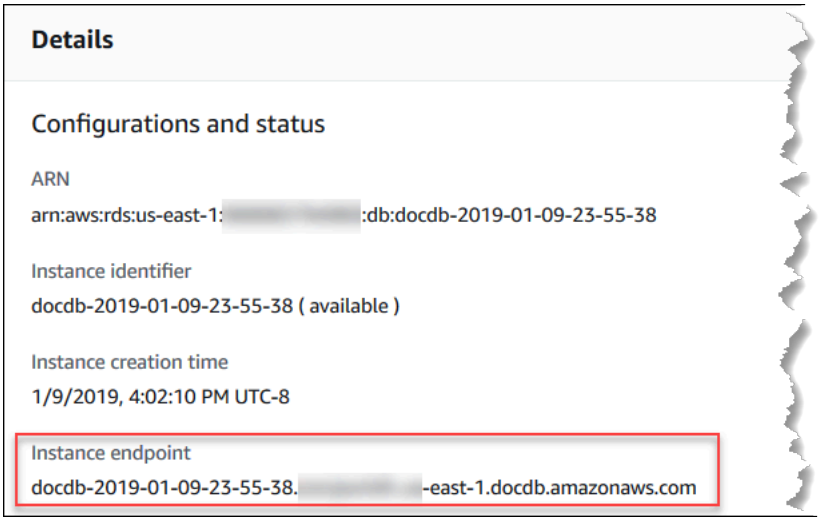
1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在集群导航框中，您将看到“集群标识符”列。您的实例列在集群下，类似于以下屏幕截图。



4. 选中您感兴趣的实例左侧的框。
5. 向下滚动至 Details (详细信息) 部分，然后找到实例终端节点。



6. 要连接到这个实例，请向上滚动到连接部分。定位 mongo Shell 的连接字符串和一个可在应用程序代码中用来连接到您实例的连接字符串。



Using the AWS CLI

要使用查找实例终端节点 AWS CLI，请使用这些参数运行以下命令。

参数

- **--db-instance-identifier**：可选。指定要返回端点的实例。如果省略， 将返回多达 100 个实例的端点。
- **--query**：可选。指定要显示的字段。因减少您为查找端点所需查看的数据量而有益。如果省略，则返回有关实例的所有信息。Endpoint 字段有三个成员，因此在查询中列出它则返回所有三个成员，如下例所示。如果您只对部分 Endpoint 成员感兴趣，请将查询中的 Endpoint 替换为您感兴趣的成员，如第二个示例中所示。
- **--region**：可选。使用 --region 参数指定要将命令应用到的区域。如果省略，则使用默认区域。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-instances \
    --region us-east-1 \
    --db-instance-identifier sample-cluster-instance \
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint]'
```

对于 Windows：

```
aws docdb describe-db-instances ^
    --region us-east-1 ^
    --db-instance-identifier sample-cluster-instance ^
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint]'
```

此操作的输出将类似于下文（JSON 格式）。

```
[
    [
        "sample-cluster-instance",
        {
            "Port": 27017,
            "Address": "sample-cluster-instance.corcjozrlsfc.us-
east-1.docdb.amazonaws.com",
            "HostedZoneId": "Z2R2ITUGPM61AM"
        }
    ]
```

```
]
```

减少输出以消除终端节点的 HostedZoneId，可通过指定 Endpoint.Port 和 Endpoint.Address 来修改查询。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-instances \
    --region us-east-1 \
    --db-instance-identifier sample-cluster-instance \
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint.Port,Endpoint.Address]'
```

对于 Windows：

```
aws docdb describe-db-instances ^
    --region us-east-1 ^
    --db-instance-identifier sample-cluster-instance ^
    --query 'DBInstances[*].[DBInstanceIdentifier,Endpoint.Port,Endpoint.Address]'
```

此操作的输出将类似于下文（JSON 格式）。

```
[
    [
        "sample-cluster-instance",
        27017,
        "sample-cluster-instance.corcjozrlsfc.us-east-1.docdb.amazonaws.com"
    ]
]
```

现在您已拥有实例端点，可以使用 mongo 或 mongodb 连接到实例。有关更多信息，请参阅 连接到端点。

连接到端点

当您拥有自己的端点（集群或实例）时，可以使用 mongo Shell 或连接字符串连接到该端点。

使用 mongo Shell 连接

使用以下结构来构造您需要使用 mongo Shell 连接到自身集群或实例的字符串：

```
mongo \
```

```
    --ssl \
    --host Endpoint:Port \
    --sslCAFile global-bundle.pem \
    --username UserName \
    --password Password
```

mongo 外壳示例

连接到集群：

```
mongo \
    --ssl \
    --host sample-cluster.corcjozrlsfc.us-east-1.docdb.amazonaws.com:27017 \
    --sslCAFile global-bundle.pem \
    --username UserName \
    --password Password
```

连接到实例：

```
mongo \
    --ssl \
    --host sample-cluster-instance.corcjozrlsfc.us-east-1.docdb.amazonaws.com:27017 \
    --sslCAFile global-bundle.pem \
    --username UserName \
    --password Password
```

使用连接字符串连接

使用以下结构来构造您需要连接到自身集群或实例的连接字符串：

```
mongodb://UserName:Password@endpoint:port?replicaSet=rs0&ssl_ca_certs=global-
bundle.pem
```

连接字符串示例

连接到集群：

```
mongodb://UserName:Password@sample-cluster.cluster-corlsfccjozr.us-
east-1.docdb.amazonaws.com:27017?replicaSet=rs0&ssl_ca_certs=global-bundle.pem
```

连接到实例：

```
mongodb://UserName:Password@sample-cluster-instance.cluster-corlsfccjozr.us-
east-1.docdb.amazonaws.com:27017?replicaSet=rs0&ssl_ca_certs=global-bundle.pem
```

了解 Amazon DocumentDB Amazon 资源名称（ARN）

您在中创建的每个资源 AWS 都使用亚马逊资源名称 (ARN) 进行唯一标识。对于某些 Amazon DocumentDB（与 MongoDB 兼容）操作，您必须通过指定其 ARN 来唯一标识 Amazon DocumentDB 资源。例如，当您向资源中添加标签时，必须提供该资源的 ARN。

主题

- 为 Amazon DocumentDB 资源构造 ARN
- 查找 Amazon DocumentDB 资源 ARN

为 Amazon DocumentDB 资源构造 ARN

您可以使用以下语法为 Amazon DocumentDB 资源构造 ARN。Amazon DocumentDB 共享 Amazon Relational Database Service（Amazon RDS）ARN 的格式。Amazon DocumentDB ARN 包含 `rds` 而不是含 `docdb`。

`arn:aws:rds:region:account_number:resource_type:resource_id`

区域名称	Region	可用区（计算）
美国东部（俄亥俄州）	us-east-2	3
美国东部（弗吉尼亚州北部）	us-east-1	6
美国西部（俄勒冈州）	us-west-2	4
非洲（开普敦）	af-south-1	3
南美洲（圣保罗）	sa-east-1	3

区域名称	Region	可用区（计算）
亚太地区（香港）	ap-east-1	3
亚太地区（海得拉巴）	ap-south-2	3
亚太地区（马来西亚）	ap-southeast-5	3
亚太地区（孟买）	ap-south-1	3
亚太地区（大阪）	ap-northeast-3	3
亚太地区（首尔）	ap-northeast-2	4
亚太地区（新加坡）	ap-southeast-1	3
亚太地区（悉尼）	ap-southeast-2	3
亚太地区（雅加达）	ap-southeast-3	3
亚太地区（墨尔本）	ap-southeast-4	3
亚太地区（泰国）	ap-southeast-7	3
亚太地区（东京）	ap-northeast-1	3
加拿大（中部）	ca-central-1	3
加拿大西部（卡尔加里）	ca-west-1	3
中国（北京）区域	cn-north-1	3
中国（宁夏）	cn-northwest-1	3
欧洲地区（法兰克福）	eu-central-1	3
欧洲（苏黎世）	eu-central-2	3

区域名称	Region	可用区（计算）
欧洲地区（爱尔兰）	eu-west-1	3
欧洲地区（伦敦）	eu-west-2	3
欧洲地区（米兰）	eu-south-1	3
欧洲地区（巴黎）	eu-west-3	3
欧洲（西班牙）	eu-south-2	3
欧洲地区（斯德哥尔摩）	eu-north-1	3
墨西哥（中部）	mx-central-1	3
中东（阿联酋）：	me-central-1	3
以色列（特拉维夫）	il-central-1	3
AWS GovCloud (US-West)	us-gov-west-1	3
AWS GovCloud (US-East)	us-gov-east-1	3

> **Note**
>
> Amazon DocumentDB 架构将存储和计算分开。在存储层方面，Amazon DocumentDB 会在三个 AWS 可用区 (AZ) 中复制六个数据副本。上表中列出的 AZ 是指，您可以在给定区域中预置计算实例时使用的 AZ 数。例如，如果要在 ap-northeast-1 中启动一个 Amazon DocumentDB 集群，将在三个 AZ 之间的 6 个方向复制存储，但仅在两个 AZ 中具有计算实例。

下表显示在构造特定 Amazon DocumentDB 资源的 ARN 时应使用的格式。Amazon DocumentDB 共享 Amazon RDS ARN 的格式。Amazon DocumentDB ARN 包含 `rds` 而不是含 `docdb`。

资源类型	ARN 格式/示例
实例 (db)	arn:aws:rds: *region*:*account_number* :db:*resource_id* `arn:aws:rds:us-east-1: 1234567890 :db:sample-db-instance`
集群 (cluster)	arn:aws:rds: *region*:*account_number* :cluster: *resource_id* `arn:aws:rds:us-east-1: 1234567890 :cluster: sample-db-cluster`
集群参数组 (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`
安全组 (secgrp)	arn:aws:rds: *region*:*account_number* :secgrp:*resource_id* `arn:aws:rds:us-east-1: 1234567890 :secgrp:sample-public-secgrp`
集群快照 (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`
子网组 (subgrp)	arn:aws:rds: *region*:*account_number* :subgrp:*resource_id*

资源类型	ARN 格式/示例
	`arn:aws:rds:us-east-1: 1234567890 :subgrp:sample-subnet-10`

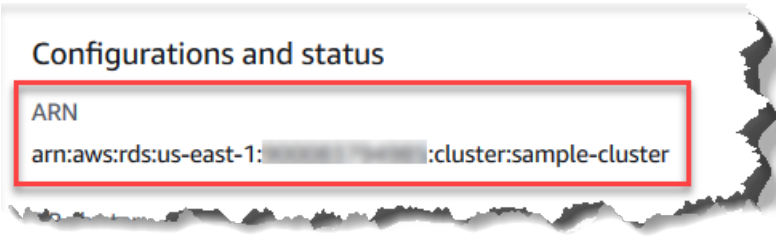
查找 Amazon DocumentDB 资源 ARN

您可以使用或查找 Amazon DocumentDB 资源的 ARN。 AWS 管理控制台 AWS CLI

Using the AWS 管理控制台

要使用控制台查找 ARN，请导航到要获取其 ARN 的资源，然后查看该资源的详细信息。

例如，您可以通过选择集群详细信息页面上的配置选项卡来获取集群的 ARN。如以下屏幕截图所示，您可以在配置和状态部分中找到 ARN。



Using the AWS CLI

要使用特定的 Amazon DocumentDB 资源获取 ARN，请`describe`对该资源使用操作。 AWS CLI 下表显示了每项 AWS CLI 操作以及用于获取 ARN 的操作的 ARN 属性。

AWS CLI 命令	ARN 属性
`describe-db-instances`	`DBInstanceArn`
`describe-db-clusters`	`DBClusterArn`
`describe-db-parameter-groups`	`DBParameterGroupArn`
`describe-db-cluster-parameter-groups`	`DBClusterParameterGroupArn`
`describe-db-security-groups`	`DBSecurityGroupArn`

AWS CLI 命令	ARN 属性
describe-db-snapshots	DBSnapshotArn
describe-db-cluster-snapshots	DBClusterSnapshotArn
describe-db-subnet-groups	DBSubnetGroupArn

Example- 查找集群的 ARN

以下 AWS CLI 操作查找集群的 ARN。sample-cluster

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterArn'
```

对于 Windows：

```
aws docdb describe-db-clusters ^
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[*].DBClusterArn'
```

此操作的输出将类似于下文（JSON 格式）。

```
[
    "arn:aws:rds:us-east-1:123456789012:cluster:sample-cluster"
]
```

Example- 查找多个参数组的 ARN

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameter-groups \
   --query 'DBClusterParameterGroups[*].DBClusterParameterGroupArn'
```

对于 Windows：

```
aws docdb describe-db-cluster-parameter-groups ^
```

```
    --query 'DBClusterParameterGroups[*].DBClusterParameterGroupArn'
```

此操作的输出将类似于下文（JSON 格式）。

```
[
    "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"
]
```

标记 Amazon DocumentDB 资源

您可以使用 Amazon DocumentDB（与 MongoDB 兼容）标签将元数据添加到 Amazon DocumentDB 资源。这些标签可以与 AWS Identity and Access Management (IAM) 策略配合使用，以管理对 Amazon DocumentDB 资源的访问权限并控制可以对资源应用哪些操作。您还可以将具有类似标签的资源的费用分组在一起，使用标签来跟踪成本。

您可以标记以下 Amazon DocumentDB 资源：

- 集群
- 实例
- 快照
- 集群快照
- 参数组
- 集群参数组
- 安全组
- 子网组

Amazon DocumentDB 资源标签概述

Amazon DocumentDB 标签是由您定义的名称-值对，与某种 Amazon DocumentDB 资源关联。此名称也叫键。为键提供值为可选操作。可使用标签向 Amazon DocumentDB 资源分配任意信息。例如，您可以使用标签键定义一个类别，而标签值可以是该类别中的一个项目。例如，定义 `project` 的标签键和 `Salix` 的标签值 ，表示将 Amazon DocumentDB 资源分配给 Salix 项目。您也可以使用标签通过 `environment=test` 或 `environment=production` 等键指定 Amazon DocumentDB 资源用于测

试或生产。我们建议使用一组具有一致性的标签键，以使跟踪与 Amazon DocumentDB 资源关联的元数据变得更轻松。

您可以使用标签来整理 AWS 账单，以反映您自己的成本结构。为此，请注册以获取包含标签键值的 AWS 账户 账单。然后，如需查看组合资源的成本，请按有同样标签键值的资源组织您的账单信息。例如，您可以将特定的应用程序名称用作几个资源的标签，然后组织账单信息，以查看在数个服务中的使用该应用程序的总成本。有关更多信息，请参阅AWS 账单和成本管理用户指南中的使用成本分配标签。

每个 Amazon DocumentDB 资源都有一组标签，其中包含分配给该资源的所有标签。一个标签集可以包含多达 10 个标签，也可以为空。如果给 Amazon DocumentDB 资源添加一个标签，而该标签的键与资源上某个现有的标签相同，则新值覆盖旧值。

AWS 不对您的标签应用任何语义含义；标签严格解释为字符串。Amazon DocumentDB 可以在实例或其他 Amazon DocumentDB 资源上设置标签，具体取决于您创建资源时使用的设置。例如，Amazon DocumentDB 可能添加一个标签来指示实例用于生产或测试。

您可以将标签添加到快照中，但您的账单不会反映此分组。

您可以使用 AWS 管理控制台 或在 Amazon DocumentDB 资源上添加、列出和删除标签。 AWS CLI 使用时 AWS CLI，您必须为要使用的资源提供亚马逊资源名称 (ARN)。有关 Amazon DocumentDB ARN 的更多信息，请参阅 了解 Amazon DocumentDB Amazon 资源名称（ARN）。

标签约束

以下约束适用于 Amazon DocumentDB 标签：

- 每个资源的最大标签数：10
- 最大键长度：128 个 Unicode 字符
- 最大值长度：256 个 Unicode 字符
- 键和值的有效字符-字符集中的大写和小写字母、数字、空 UTF-8 格和以下字符：`_ . : / = + -`和 @ (Java regex:) `"^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$"`
- 标签键和值区分大小写。
- 前缀 `aws:` 无法用于标签键或值；将其保留供 AWS使用。

为 Amazon DocumentDB 资源添加和更新标签

您可以使用 AWS 管理控制台 或向资源添加最多 10 个标签 AWS CLI。

Using the AWS 管理控制台

无论您将标签添加到哪个资源，向资源添加标签的过程都是相似的。在本示例中，您向集群添加标签。

要使用控制台向集群添加标签或更新标签

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 从导航窗格中，选择集群。
3. 选择要将标签添加到的集群的名称。
4. 向下滚动到标签部分，然后选择 编辑。
5. 对于每个要添加到此资源的标签，请执行以下操作：
   a. 要添加新标签，请在值框中输入标签的名称。要更改标签的值，请在值列中找到标签的名称。
   b. 要赋予标签新建或更新的值，在值框中，为标签输入一个值。
   c. 如果要添加多个标签，请选择添加。否则，完成后，选择保存。

Using the AWS CLI

无论您将标签添加到哪个资源，向资源添加标签的过程都是相似的。在本示例中，您向集群添加三个标签。第二个标签 key2 没有值。

使用`add-tags-to-resource`带有这些参数的 AWS CLI 操作。

参数

- **`--resource-name`**：要将标签添加到的 Amazon DocumentDB 资源的 ARN。
- **`--tags`**：要添加到格式为 `Key=`*`key-name`*`,Value=`*`tag-value`* 的此资源的标签（键/值对）的列表。

Example

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此 `add-tags-to-resource` 操作不会生成任何输出。要查看操作结果，请使用 `list-tags-for-resource` 操作。

列出 Amazon DocumentDB 资源上的标签

您可以使用 AWS 管理控制台 或 AWS CLI 来获取 Amazon DocumentDB 资源的标签列表。

Using the AWS 管理控制台

无论您将标签添加到哪个资源，在资源上列出标签的过程都是相似的。在本示例中，您为集群列出标签。

要使用控制台列出集群上的标签

1. 打开 Amazon DocumentDB 控制台，网址为。https://console.aws.amazon.com/docdb
2. 从导航窗格中，选择集群。
3. 选择要为之列出标签的集群的名称。
4. 要查看此资源上的标签列表，向下滚动到标签部分。

Using the AWS CLI

无论您为哪个资源列出标签，在资源上列出标签的过程都是相似的。在本示例中，您在集群上列出标签。

使用`list-tags-for-resource`带有这些参数的 AWS CLI 操作。

参数

- **`--resource-name`**：必填项。您要列出其标签的 Amazon DocumentDB 资源的 ARN。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb list-tags-for-resource \
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster
```

对于 Windows：

```
aws docdb list-tags-for-resource ^
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "TagList": [
        {
            "Key": "key1",
            "Value": "value1"
        },
        {
            "Key": "key2",
            "Value": ""
        },
        {
            "Key": "key3",
            "Value": "value3"
        }
    ]
}
```

从 Amazon DocumentDB 资源中删除标签

您可以使用 AWS 管理控制台 或从 Amazon DocumentDB 资源中移除标签。 AWS CLI

Using the AWS 管理控制台

无论您将标签添加到哪个资源，从资源中删除标签的过程都是相似的。在本示例中，您从集群中删除标签。

使用控制台从集群中删除标签

1. 打开 Amazon DocumentDB 控制台，网址为。https://console.aws.amazon.com/docdb
2. 从导航窗格中，选择集群。
3. 选择要从中删除标签的集群的名称。
4. 向下滚动到标签部分，然后选择 编辑。
5. 如果您要从该资源中删除所有标签，请选择 Remove all（删除所有）。否则，对于每个要从此资源删除的标签，请执行以下操作：

   a. 查找键列中的标签的名称。

   b. 在同一行上选择删除作为标签值。

   c. 完成后，选择保存。

Using the AWS CLI

无论您从哪个资源删除标签，从资源中删除标签的过程都是相似的。在本示例中，您从集群中删除标签。

使用`remove-tags-from-resource`带有这些参数的 AWS CLI 操作。

- **`--resource-name`**：必填项。要从中删除标签的 Amazon DocumentDB 资源的 ARN。
- **`--tag-keys`**：必填项。要从此资源中删除的标签值列表。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb remove-tags-from-resource \
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster \
    --tag-keys key1 key3
```

对于 Windows：

```
aws docdb remove-tags-from-resource ^
    --resource-name arn:aws:rds:us-east-1:1234567890:cluster:sample-cluster \
    --tag-keys key1 key3
```

此 `removed-tags-from-resource` 操作不会生成任何输出。要查看操作结果，请使用 `list-tags-for-resource` 操作。

维护 Amazon DocumentDB

亚马逊 DocumentDB 定期执行两种维护：

- 集群维护会更新数据库引擎。引擎更新包含安全修复、错误修复、新功能和其他引擎增强功能。
- 实例维护会更新实例上的操作系统 (OS)。

引擎补丁和操作系统更新使用相同的三个生命周期类别（可选、必需和强制），每个类别的通知和应用行为都相同。引擎版本还有第四类：次要版本，您可以手动升级到次要版本。这些类别是：

- 可选-包含非关键改进。没有自动申请日期，也没有 AHD 通知；在适合你的时候申请。（对于操作系统更新，您可以订阅`RDS-EVENT-0230`以便在操作系统更新可用时收到通知。）
- 必填项-包含安全修复和其他重要补丁。您会通过 Health Dashboard (AHD) 和电子邮件收到通知。在集群或实例维护窗口之后，必需的操作会自动生效。`AutoAppliedAfterDate`您可以通过在该日期之前更改维护时段来推迟。
- 强制 — 一种罕见的、高度关键的修复方法。 Auto-applies 之后在维护窗口之外`ForcedApplyDate`。只有在没有其他选项可用时，Amazon DocumentDB 才会指定强制执行的操作。
- 次要版本（仅限引擎版本）— 主版本之上的带编号的引擎版本（例如`5.0.1`）。 User-driven：您可以通过修改集群的引擎版本进行升级。永远不要自动应用；没有 AHD 通知。5.0 之前的主要版本不发布次要版本。

引擎补丁按单一类别（可选、必需或强制）发布，并保持不变。操作系统更新进度：大多数操作系统以可选形式启动，如果未应用，则过渡到必需并最终强制更新。确切的时间取决于补丁，并发布在 AHD 通知和返回的日期字段中`describe-pending-maintenance-actions`（参见应用日期）。亚马逊 DocumentDB 发行说明在宣布引擎变更时使用这些类别名称。

应用任何引擎补丁都会使集群短暂脱机。本主题的其余部分将介绍维护窗口的工作原理、如何查找待处理的工作、如何应用引擎补丁和次要版本、操作系统更新的工作原理以及全局集群的特殊处理。

主题

- 引擎版本号
- 管理您的 Amazon DocumentDB 维护窗口

- Amazon DocumentDB 引擎补丁通知
- 查看待处理的 Amazon DocumentDB 维护操作
- Amazon DocumentDB 引擎更新
- 次要版本升级
- Amazon DocumentDB 操作系统更新
- User-initiated 更新
- 全球集群修补

引擎版本号

亚马逊 DocumentDB 使用两个不同的版本标识符：

- 引擎版本-表格中由三部分组成的数字*major*.*major*.*minor*（例如，5.0.0或5.0.1）。前两部分 (5.0) 是 MongoDB 兼容版本；第三部分是次要版本，当 Amazon DocumentDB 发布包含错误修复和不间断改进的次要版本时，次要版本会增加。这是您在创建或升级集群时指定的版本。
- 引擎补丁版本 — 一个由三部分组成的单独数字*major*.0.*patch*（例如3.0.17983），用于标识应用于集群的补丁级别。中间的数字始终是0。补丁版本包含重要的安全性和稳定性修复程序。

您可以根据引擎补丁版本的前缀确定引擎版本，如下表所示。

引擎补丁版本前缀	亚马逊 DocumentDB 引擎版本
1.0.*x*	3.6
2.0.*x*	4.0
3.0.*x*	5.0
4.0.*x*	8.0

要检查您的集群正在运行的补丁版本，请连接并运行db.runCommand({getEngineVersion: 1})。

有关已发布的引擎补丁版本列表以及每个版本包含的内容，请参阅发布说明。

管理您的 Amazon DocumentDB 维护窗口

每个集群和每个实例都有自己的每周 30 分钟维护窗口，即定期修改和软件补丁运行的时段。大多数活动会在 30 分钟内完成；规模较大的活动可能会持续更长时间。

如果您在创建资源时没有选择窗口，Amazon DocumentDB 会在为该地区定义的每日 8 小时区块内，在随机选择的某一天随机分配一个窗口。选择能最大限度地减少对应用程序影响的窗口，例如晚间或周末。

对于数据库引擎升级，Amazon DocumentDB 使用集群的窗口，而不是单个实例的窗口。

下表显示了每个区域的默认时段。

区域名称	Region	UTC 时间数据块
美国东部（俄亥俄州）	us-east-2	03:00-11:00
美国东部（弗吉尼亚州北部）	us-east-1	03:00-11:00
美国西部（俄勒冈州）	us-west-2	06:00-14:00
非洲（开普敦）	af-south-1	03:00–11:00
亚太地区（香港）	ap-east-1	06:00-14:00
亚太地区（海得拉巴）	ap-south-2	06:30–14:30
亚太地区（马来西亚）	ap-southeast-5	13:00-21:00
亚太地区（孟买）	ap-south-1	06:00-14:00
亚太地区（大阪）	ap-northeast-3	12:00-20:00
亚太地区（首尔）	ap-northeast-2	13:00-21:00
亚太地区（新加坡）	ap-southeast-1	14:00-22:00
亚太地区（悉尼）	ap-southeast-2	12:00-20:00
亚太地区（雅加达）	ap-southeast-3	08:00-16:00
亚太地区（墨尔本）	ap-southeast-4	11:00-19:00

区域名称	Region	UTC 时间数据块
亚太地区（泰国）	ap-southeast-7	15:00-23:00
亚太地区（东京）	ap-northeast-1	13:00-21:00
加拿大（中部）	ca-central-1	03:00-11:00
加拿大西部（卡尔加里）	ca-west-1	18:00-02:00
中国（北京）	cn-north-1	06:00-14:00
中国（宁夏）	cn-northwest-1	06:00-14:00
欧洲地区（法兰克福）	eu-central-1	21:00-05:00
欧洲（苏黎世）	eu-central-2	02:00-10:00
欧洲地区（爱尔兰）	eu-west-1	22:00-06:00
欧洲（伦敦）	eu-west-2	22:00-06:00
欧洲地区（米兰）	eu-south-1	02:00-10:00
欧洲地区（巴黎）	eu-west-3	23:59-07:29
欧洲（西班牙）	eu-south-2	02:00–10:00
欧洲地区（斯德哥尔摩）	eu-north-1	04:00 — 12:00
墨西哥（中部）	mx-central-1	03:00-11:00
中东（阿联酋）：	me-central-1	05:00–13:00
南美洲（圣保罗）	sa-east-1	00:00-08:00
以色列（特拉维夫）	il-central-1	04:00-12:00
AWS GovCloud (US-East)	us-gov-east-1	17:00-01:00
AWS GovCloud (US-West)	us-gov-west-1	06:00-14:00

更改您的 Amazon DocumentDB 维护窗口

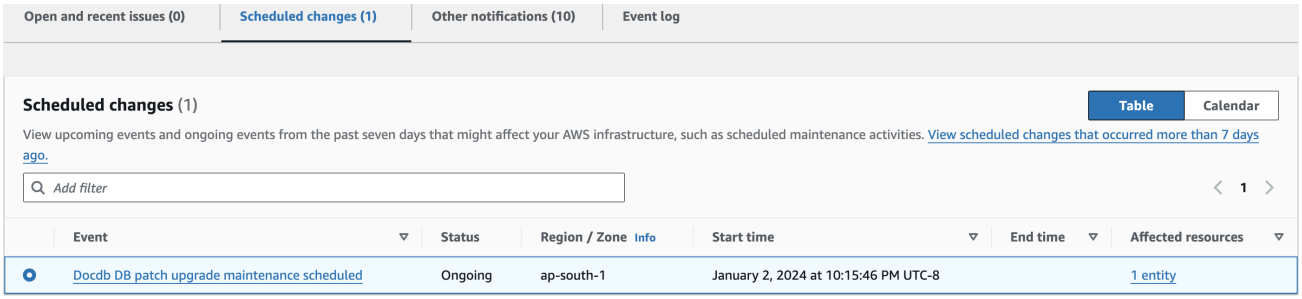
尽你所能选择最低流量窗口，并随着交通模式的变化而随着时间的推移进行调整。只有当系统更改（例如扩展存储操作或实例类别更改）需要中断时，集群或实例才会在窗口期间不可用，并且只能在变更实际需要的时间内持续下去。

更改维护时段

- 对于集群：请参阅修改 Amazon DocumentDB 集群。
- 对于实例：请参阅修改 Amazon DocumentDB 实例。

Amazon DocumentDB 引擎补丁通知

当所需的引擎补丁在某个 AWS 地区可用时，在该地区拥有受影响的 Amazon DocumentDB 集群的每个 AWS 账户都会通过 Health Dashboard (AHD) 和电子邮件（发送到该 AWS 账户的根用户地址）收到通知。每个受影响的 Amazon DocumentDB 引擎版本都会发送一条通知。您可以在 AHD 的 “预定更改” 下找到它们。每份通知都列出了补丁可用时间、自动应用时间表、受影响的集群和发布说明。



通知会在自动申请窗口打开前大约两天发出。例如，周一 00:00 UTC 发布的必需补丁有资格在周三 00:00 UTC 自动应用。如果您的集群的维护时段是星期三 12:00 UTC，则补丁将在该星期三自动应用，也就是自动应用窗口打开大约 12 小时后。如果您的维护时段是星期二 12:00 UTC，则补丁会等待整整一周才能自动应用。

收到通知后，您有两种选择：在自动应用日期之前自行应用补丁，或者在即将到来的维护时段内等待补丁自动应用（默认）。要自行申请，请打开集群的 “维护和备份” 选项卡并查找类型的`system-update`条目。

> **Note**
>
> 在 Amazon DocumentDB 发布另一个包含新补丁版本的引擎补丁之前，通知在 AHD 中的状态一直保持不变。

应用补丁后，集群的引擎补丁版本会更新以匹配通知中的版本。通过运行来验证新版本`db.runCommand({getEngineVersion: 1})`。

可选补丁和新的次要版本不会生成 AHD 或电子邮件通知。要追踪它们，请查看亚马逊 DocumentDB 发行说明。

强制补丁（最稀有的类别，保留用于最关键的安全修复）也通过AHD和电子邮件公布。与必需的补丁不同，它们在维护时段之外适用，因此上面的自动应用计时示例不适用。

以编程方式对补丁通知做出反应

AWS Health 与 Amazon 集成 EventBridge，允许您跨超过 20 个目标构建事件驱动的应用程序，包括 AWS Lambda 亚马逊简单队列服务 (SQS) Simple Queue Service。要以编程方式响应引擎补丁的可用性，请 EventBridge 针对事件进行配置。`AWS_DOCDB_DB_PATCH_UPGRADE_MAINTENANCE_SCHEDULED`从那里你可以捕获事件数据、引发其他事件、通过发送推送通知或采取任何其他你需要的操作。 AWS Console Mobile Application

如果 Amazon DocumentDB 取消补丁（罕见），您将收到 AHD 通知和一封关于取消补丁的电子邮件。使用 Amazon `AWS_DOCDB_DB_PATCH_UPGRADE_MAINTENANCE_CANCELLED` 的事件代码 EventBridge 来处理此案例。有关编写规则的更多信息，请参阅 Amazon EventBridge 用户指南。

查看待处理的 Amazon DocumentDB 维护操作

使用 AWS 管理控制台 或查看 AWS CLI 集群或实例的待维护情况。

待定更新显示为操作类型`system-update`，包括引擎补丁和操作系统更新。

当更新处于待定状态时，您可以：

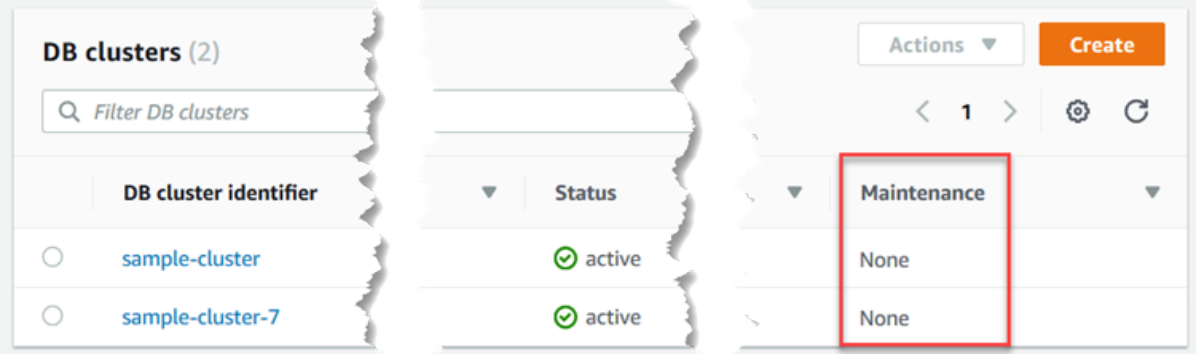
- 立即涂抹。
- 将其安排在下一个维护时段。
- 通过更改维护时段来推迟（仅限引擎补丁和操作系统更新）`AutoAppliedAfterDate`。该日期过后，该操作将在下一个维护时段内自动生效。一旦`ForcedApplyDate`通过，就不可能再延期。

> **Note**
>
> 如果您不采取任何行动，则在即将到来的维护时段内会自动应用所需的维护操作，例如所需的引擎补丁。可选补丁和次要版本永远不会自动应用。

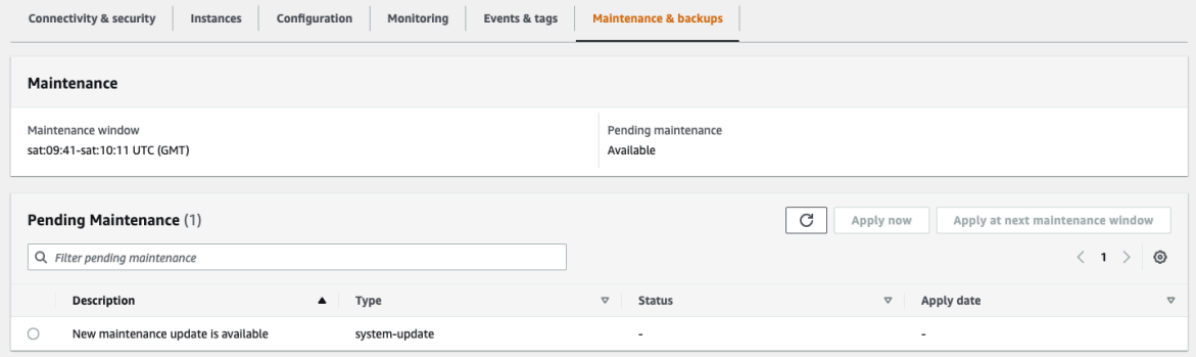
维护窗口控制待处理的操作何时开始，而不是它们需要多长时间才能完成。

Using the AWS 管理控制台

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。
3. 当更新处于待定状态时，集群的 “维护” 列会显示 “可用”、“必需” 或 “下一个窗口”。



4. 打开群集，然后选择 “维护和备份” 以查看待定维护项目并对其进行操作。



Using the AWS CLI

跑describe-pending-maintenance-actions去看看还有什么待处理的。以下示例显示了一个没有待处理操作的账户。

```
aws docdb describe-pending-maintenance-actions
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "PendingMaintenanceActions": []
```

```
}
```

有待处理操作的账户会返回如下所示的输出：

```
{
    "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"
                }
            ]
        }
    ]
}
```

您可以在表单中使用`--filters`将列表范围限定到特定的集群`Name=`*`filter-name`*`,Values=`*`resource-id`*`,...`。接受的筛选条件`Name`是`db-cluster-id`，它采用集群标识符或 ARN 的列表。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb describe-pending-maintenance-actions \
   --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

对于 Windows：

```
aws docdb describe-pending-maintenance-actions ^
   --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

应用日期

每个待处理的维护操作最多有三个申请日期。它们出现在的 AWS CLI 输出中，`describe-pending-maintenance-actions`并指示操作何时运行。字段`null`用于可选维护。

- **CurrentApplyDate**— 当操作计划运行时，无论是现在还是下一个维护时段。为必需操作和强制操作填充。
- **AutoAppliedAfterDate**— 在集群或实例维护时段内开始自动应用的日期。已填充以执行所需的操作。
- **ForcedApplyDate**— 艰难的最后期限。在此日期之后，无论您的维护时段如何，操作都会自动运行。已填充以进行强制操作。

要推迟待处理的操作，请将维护时段移至前一天晚些时候AutoAppliedAfterDate。AutoAppliedAfterDate通过后，该操作将在下一个维护时段内自动生效。一旦ForcedApplyDate通过，就不可能再延期。确切的延期窗口因补丁而异；日期发布在AHD通知和输出中 AWS CLI 。

Amazon DocumentDB 引擎更新

确定待处理的引擎补丁后，请使用以下过程之一来应用或安排修补程序。您可以从 AWS 管理控制台或运行这些过程 AWS CLI。

Using the AWS 管理控制台

管理集群的更新

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。
3. 选择要更新的集群。
4. 从 “操作” 菜单中，选择以下任一选项：
   - 立即升级-立即运行待处理的维护。
   - 在下一个窗口升级-在集群的下一个维护窗口中运行它。

   您也可以使用集群 “维护和备份” 选项卡的 “待定维护” 部分中的 “立即申请” 或 “在下一个维护时段申请”（请参阅查看待处理的 Amazon DocumentDB 维护操作）。

   > **Note**
   >
   > 如果没有待处理的项目，则所有这些选项都处于非活动状态。

Using the AWS CLI

使用应用待处理的更新`apply-pending-maintenance-action`。

参数

- **`--resource-identifier`**— 待处理操作目标资源的亚马逊 DocumentDB 亚马逊资源名称 (ARN)。
- **`--apply-action`**— 待执行的维护操作。有效值：`system-update`、`db-upgrade`。
- **`--opt-in-type`**— 选择加入请求的类型，或者是否撤消一个请求。有效值：
  - `immediate`—立即申请。提交后无法撤消。
  - `next-maintenance`—在资源的下一个维护时段内应用。
  - `undo-opt-in`—取消现有的`next-maintenance`选择加入。

Example

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

在修补期间读取可用性

当集群有多个实例时，Amazon DocumentDB 引擎 5.0 和 8.0 在修补期间保留读取可用性。Amazon DocumentDB 以滚动方式修补阅读器实例，分为三组，因此其余的读取器可以继续提供流量。写入器在修补时短暂不可用。要实现零读取停机时间，请设置读取首选项，以便读取可以回退到写入

器：secondaryPreferred或primaryPreferred工作；primary或者secondary单独读取可能会导致读取停机。

阅读偏好模式	在写入器升级期间	在读卡器升级期间	读取停机时间为零所需的读取器数量最少
primary	Read/write 停机时间	无影响	N/A
primaryPreferred	写入停机时间	无影响	1
secondary	写入停机时间	读取停机时间（如果只有一个读取器）	2
secondaryPreferred	写入停机时间	无影响	1
nearest	写入停机时间	无影响	1

当读取器修补时，集群的整体读取吞吐量会暂时下降。要保持吞吐量稳定，请在升级之前配置其他读取器，并在升级完成后将其删除。

在引擎 3.6 和 4.0 上，这些读取可用性功能不适用：引擎补丁会导致更长的停机时间，从而影响读取和写入。要升级到支持的主要版本，请参阅Amazon DocumentDB 主版本就地升级。

补丁停机时间

Engine-patch 停机时间各不相同。最大的因素是修补时实例的 CPU 利用率和内存压力，因此正确调整实例大小很重要。为了最大限度地减少停机时间，请运行最新的 Amazon DocumentDB 主引擎版本，并将实例分布到多个可用区。

补丁更新和替换

Amazon DocumentDB 会在补丁发布后进行监控。在发现问题的极少数情况下，Amazon DocumentDB 会在准备更新版本时暂停部署。发生这种情况时，尚未收到补丁的集群将不再将其视为可用的维护操作，并且中相应的计划变更通知将被撤回。 Health Dashboard 已经运行受影响版本的集群可以继续正常运行，无需您采取任何操作。

不久将发布更新的补丁。当它在您所在的地区推出时，您会通过 Health Dashboard 和电子邮件收到新的通知，如中所述Amazon DocumentDB 引擎补丁通知。

次要版本升级

Amazon DocumentDB 在主版本 5.0 及更高版本的基础上发布次要版本（例如，5.0.1）。5.0 之前的主要版本不发布次要版本。次要版本的行为与必需和可选的引擎补丁不同：

- 它们不会显示为待处理的维护操作，也不会自动应用。
- 它们不生成 AHD 或电子邮件通知。新的次要版本将在亚马逊 DocumentDB 发行说明中公布。
- 要升级，您需要修改集群的引擎版本（立即或在下一个维护时段内）。次要版本升级需要短暂的停机时间，并且是单向的，您无法降级到较早的次要版本。对于全局群集，请先升级辅助群集，然后再升级主群集。

阅读更多:亚马逊 DocumentDB 次要版本升级.

Amazon DocumentDB 操作系统更新

实例偶尔需要更新操作系统。Amazon DocumentDB 更新操作系统以提高性能并加强安全性。操作系统更新使集群引擎版本和实例类保持不变。与引擎补丁一样，操作系统更新使用本主题顶部描述的可选/必需/强制生命周期；与引擎补丁不同，如果您推迟，操作系统更新可能会随着时间的推移在这些类别中过渡。在操作系统更新可用时立即应用这些更新，并将您的实例维护窗口设置为适合您业务的时间。

要在新的可选更新到来时获取事件，请在安全补丁事件类别RDS-EVENT-0230中订阅。有关详细信息，请参阅订阅亚马逊 DocumentDB 活动订阅。收到通知后，您可以自行将操作系统补丁应用于每个实例。

修补集群时，先更新读取器实例，最后更新写入器实例。避免同时修补读取器和写入器——补丁期间的故障转移可能会延长停机时间。主实例的维护会触发故障转移，因此每个集群运行多个实例以保持可用状态。有关更多信息，请参阅 Amazon DocumentDB 失效转移。

> **Important**
>
> 为了升级操作系统，您的 Amazon DocumentDB 实例将处于离线状态。 Multi-instance 群集可以最大限度地减少影响。如果您运行的是单实例集群，则可以临时添加辅助集群进行升级，然后将其删除。辅助设备存在时会产生通常的费用。

> **Note**
>
> 为了合规，可能需要及时了解可选和必需的更新。在维护时段内定期应用可用更新。

操作系统更新与特定的引擎版本和实例类别相关联，因此不同的实例在不同的时间符合资格。当实例符合条件时，更新将显示在控制台中；您也可以通过 AWS CLI `describe-pending-maintenance-actions`命令或 `DescribePendingMaintenanceActions` API 进行查看。

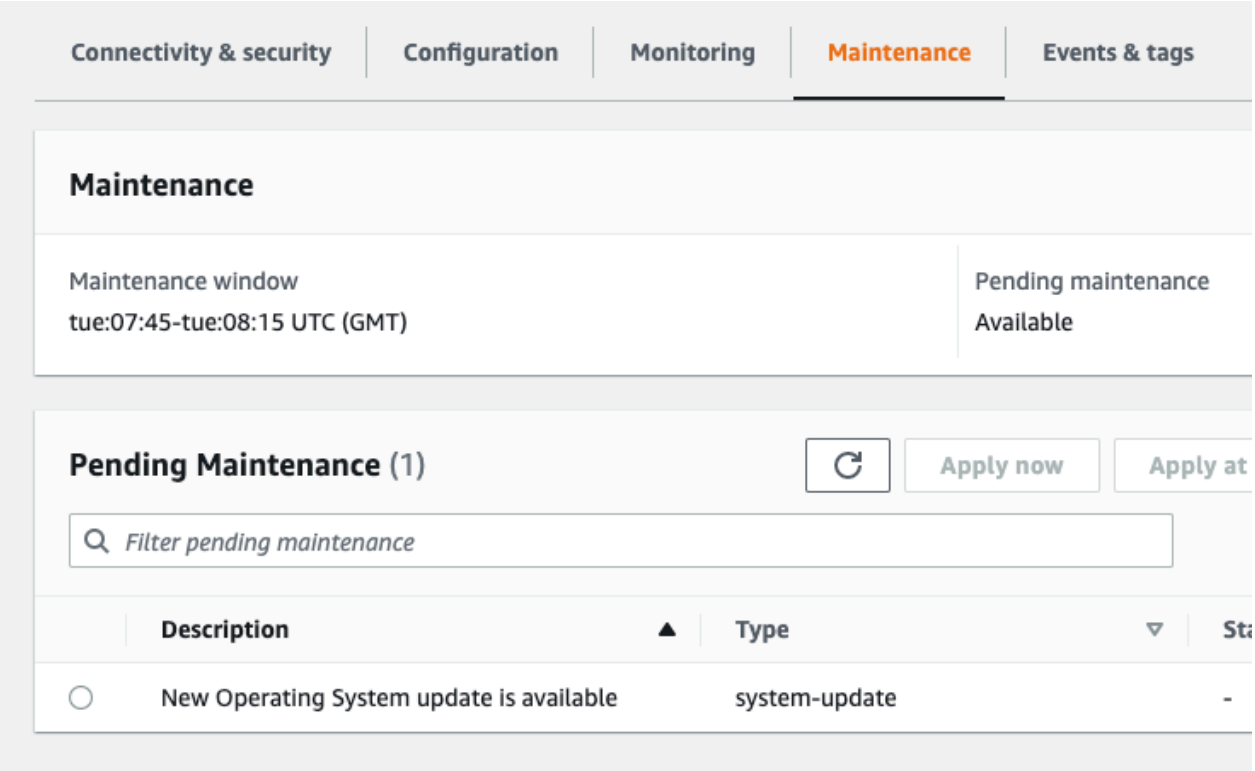
> **Note**
>
> 如果您的集群未使用其 Amazon DocumentDB 引擎的最新补丁版本，则操作系统更新可能显示为不可用。先应用最新的引擎补丁，然后再次检查。

使用 AWS 管理控制台 或 AWS CLI 检查更新是否可用。

Using the AWS 管理控制台

要从控制台查看操作系统更新，请执行以下操作：

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。该列表显示了集群及其中的实例，通过角色列进行区分。
3. 选择角色为实例的行（不是群集行）。操作系统更新适用于实例，不适用于集群。
4. 选择 维护。
5. 在 “待维护” 下查看操作系统更新。



从 “待定维护” 部分，选择操作系统更新，然后选择 “立即应用” 或 “在下一个维护时段应用”。如果维护值为下一个窗口，则只要更新尚未开始，就可以通过延迟升级来推迟更新。

您也可以从集群列表中执行此操作：在导航窗格中，选择集群，选择角色为 “实例” 的行，然后从 “操作” 菜单中选择 “立即申请” 或 “在下一个维护时段应用”。

Using the AWS CLI

从 AWS CLI中运行`describe-pending-maintenance-actions`：

```
aws docdb describe-pending-maintenance-actions
```

```
{
  "PendingMaintenanceActions": [
    {
      "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:db:sample-cluster-
instance-1",
      "PendingMaintenanceActionDetails": [
        {
          "Action": "system-update",
          "Description": "New Operating System update is available"
        }
```

```
      ]
    }
  ]
}
```

User-initiated 更新

有些更改是您自己开始的，例如，将实例类换成内存或多或少的实例类，或者更改集群的参数组。Amazon DocumentDB 对待这些内容的方式与其启动的更新不同。有关详细信息，请参阅：

- 修改 Amazon DocumentDB 集群
- 修改 Amazon DocumentDB 实例

要列出用户发起但仍处于待处理状态的更改，请执行以下操作：

Example

列出用户对您的实例发起的待处理更改

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-instances \
    --query 'DBInstances[*].
[DBClusterIdentifier,DBInstanceIdentifier,PendingModifiedValues]'
```

对于 Windows：

```
aws docdb describe-db-instances ^
    --query 'DBInstances[*].
[DBClusterIdentifier,DBInstanceIdentifier,PendingModifiedValues]'
```

此操作的输出将类似于下文（JSON 格式）。

在此示例中，`sample-cluster-instance`有待更改为`db.r5.xlarge`；没`sample-cluster-instance-2`有。

```
[
    [
```

```
        "sample-cluster",
        "sample-cluster-instance",
        {
            "DBInstanceClass": "db.r5.xlarge"
        }
    ],
    [
        "sample-cluster",
        "sample-cluster-instance-2",
        {}
    ]
]
```

全球集群修补

在全局群集中，每个成员群集（主群集和辅助群集）都会在自己的维护时段内进行升级。当每个地区都有所需的引擎补丁可用时，您会收到 AHD 和电子邮件通知。可选补丁和新的次要版本不会生成通知；请查看 Amazon DocumentDB 发行说明了解这些通知。

如果您自行申请，请务必先修补次要补丁，最后修补主补丁。此顺序使故障转移和切换在整个部署过程中都可用。

> **Important**
>
> 如果您错误地先修补了主版本，请尽快将所有辅助版本调至相同的版本。在每个集群都使用相同版本之前，故障转移和切换将保持禁用状态。

如果您不采取任何行动，则补丁会在每个集群的下一个维护时段内自动应用：首先是辅助群集，然后在辅助群集完成后在其窗口中应用主群集。

保持主数据库集群和辅助数据库集群的版本相同。只有当每个集群共享相同的引擎版本和补丁级别时，托管跨区域故障转移才适用于全局数据库。如果您添加一个使用比主版本更新的引擎版本的新辅助服务器，则同样适用，即在主服务器的版本上创建新的辅助服务器，然后再将其加入全局数据库。

收到补丁通知后，请尽早将主版本和辅助版本升级到最新版本，以保持故障转移和切换正常运行。如果故障转移或切换请求被拒绝，请比较各集群的引擎补丁版本；如果它们不匹配，请在滞后的集群上应用可用的补丁。

了解服务相关角色

亚马逊 DocumentDB（兼容 MongoDB）使用 AWS Identity and Access Management (IAM) 服务相关角色。服务相关角色是一种独特的 IAM 角色，直接链接到 Amazon DocumentDB。 Service-linked 角色由 Amazon DocumentDB 预定义，包括该服务代表您调用其他 AWS 服务所需的所有权限。

服务关联角色使得可以更轻松地设置 Amazon DocumentDB，因为您不必手动添加必要的权限。Amazon DocumentDB 定义其服务相关角色的权限，除非另外定义，否则只有 Amazon DocumentDB 可以代入该角色。定义的权限包括信任策略和权限策略，而且权限策略不能附加到任何其他 IAM 实体。

只有在首先删除角色的相关资源后，才能删除角色。这将保护您的 Amazon DocumentDB 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其它服务的信息，请参阅使用 IAM 的AWS 服务并查找Service-Linked 角色列中显示为是的服务。请选择是与查看该服务的服务关联角色文档的链接。

Amazon DocumentDB 的服务相关角色权限

亚马逊 DocumentDB（兼容 MongoDB）使用名为的服务相关角色来允许 AWSServiceRoleForRDSAmazon DocumentDB 代表您的集群调用服务。 AWS

AWSServiceRoleForRDS 服务相关角色信任以下服务来代入该角色：

- `docdb.amazonaws.com`

角色权限策略允许 Amazon DocumentDB 对指定资源完成以下操作：

- 对 `ec2` 的操作：
  - `AssignPrivateIpAddresses`
  - `AuthorizeSecurityGroupIngress`
  - `CreateNetworkInterface`
  - `CreateSecurityGroup`
  - `DeleteNetworkInterface`
  - `DeleteSecurityGroup`
  - `DescribeAvailabilityZones`
  - `DescribeInternetGateways`

  - DescribeSecurityGroups
  - DescribeSubnets
  - DescribeVpcAttribute
  - DescribeVpcs
  - ModifyNetworkInterfaceAttribute
  - RevokeSecurityGroupIngress
  - UnassignPrivateIpAddresses
- 对 sns 的操作：
  - ListTopic
  - Publish
- 对 cloudwatch 的操作：
  - PutMetricData
  - GetMetricData
  - CreateLogStream
  - PullLogEvents
  - DescribeLogStreams
  - CreateLogGroup

> **Note**
>
> 您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务关联角色。您可能会遇到以下错误消息：
> Unable to create the resource. Verify that you have permission to create service linked role。Otherwise wait and try again later.
> 如果您看到此错误，请确保您已启用以下权限：
>
> ```
> {
>     "Action": "iam:CreateServiceLinkedRole",
>     "Effect": "Allow",
>     "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
> AWSServiceRoleForRDS",
>     "Condition": {
>         "StringLike": {
>             "iam:AWSServiceName":"rds.amazonaws.com"
> ```

```
        }
    }
}
```

有关更多信息，请参阅 IAM 用户指南中的Service-Linked 角色权限。

创建 Amazon DocumentDB 服务相关角色

您无需手动创建服务关联角色。当您创建集群时，Amazon DocumentDB 将为您创建服务相关角色。

如果删除此服务相关角色然后需要再次创建它，则可以使用相同的流程在您的账户中重新创建此角色。当您创建集群时，Amazon DocumentDB 将再次为您创建服务相关角色。

修改 Amazon DocumentDB 服务关联角色

Amazon DocumentDB 不允许您修改 AWSServiceRoleForRDS 服务相关角色。创建服务关联角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。不过，您可以使用 IAM 修改角色的说明。有关更多信息，请参阅 IAM 用户指南中的编辑 Service-Linked 角色。

删除 Amazon DocumentDB 服务相关角色

如果不再需要使用某个需要服务关联角色的功能或服务，我们建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。但是，您必须先删除所有 集群，然后才能删除服务相关角色。

清除 Amazon DocumentDB 服务相关角色

必须先确认服务相关角色没有活动会话并删除该角色使用的任何资源，然后才能使用 IAM 删除服务相关角色。

要使用控制台检查服务相关角色是否具有活动会话

1. 登录 AWS 管理控制台 并打开 IAM 控制台，网址为https://console.aws.amazon.com/iam/。
2. 在 IAM 控制台的导航窗格中，选择角色，然后选择AWSServiceRoleForRDS角色的名称（不是复选框）。
3. 在所选角色的 Summary 页面上，选择 Access Advisor 选项卡。
4. 在 Access Advisor (访问顾问) 选项卡上，查看服务相关角色的近期活动。

> **Note**
>
> 如果您不确定 Amazon DocumentDB 是否 AWSServiceRoleForRDS 在使用该角色，可以尝试删除该角色。如果服务正在使用该角色，则删除操作会失败，并且您可以查看正在使用该角色的 区域。如果该角色已被使用，则您必须等待会话结束，然后才能删除该角色。您无法撤销服务相关角色对会话的权限。

如果要移除该 AWSServiceRoleForRDS 角色，则必须先删除所有实例和集群。有关删除实例和集群的信息，请参阅以下主题：

- 删除 Amazon DocumentDB 实例
- 删除 Amazon DocumentDB 集群

Amazon DocumentDB 服务相关角色支持的区域

Amazon DocumentDB 支持在该服务可用的所有区域中使用服务相关角色。有关更多信息，请参阅 https://docs.aws.amazon.com/documentdb/latest/devguide/regions-and-azs.html#regions-and-azs-availability。

将变更流与 Amazon DocumentDB 结合使用

Amazon DocumentDB（与 MongoDB 兼容）中的变更流功能提供按时间顺序排列的更改事件，这些事件在您的集群集合内发生。您可以从变更流中读取事件，以实现许多不同的使用案例，包括以下情况：

- 更改通知
- Full-text 使用亚马逊 OpenSearch 服务（OpenSearch 服务）进行搜索
- 使用 Amazon Redshift 分析

应用程序可以使用变更流在各个集合中订阅数据变更。变更流事件在集群上发生时按顺序排列，并在记录事件之后存储 3 个小时（默认情况下）。使用 `change_stream_log_retention_duration` 参数可以将保留期延长至 7 天。要修改更改流的保留期，请参阅 Modifying the Change Stream Log Retention Duration。

主题

- 支持的操作
- 计费
- 限制
- 启用变更流
- 示例：在 Python 中使用变更流
- 完整文档查找
- 恢复变更流
- 使用 start 恢复变更流 AtOperationTime
- 使用帖子恢复变更流 BatchResumeToken
- 变更流中的事务
- 修改变更流日志保留期限
- 在辅助实例上使用变更流

支持的操作

Amazon DocumentDB 支持以下变更流操作：

- MongoDB `db.collection.watch()`、`db.watch()` 和 `client.watch()` API 中支持的所有变更事件。
- 查找完整文档以获取更新。
- 聚合阶段：`$match`、`$project`、`$redact`、`$addFields` 和 `$replaceRoot`。
- 从简历令牌恢复变更流
- 使用 `startAtOperation` 从时间戳恢复变更流（适用于 Amazon DocumentDB 4.0+）

计费

默认情况下，Amazon DocumentDB 变更流功能处于禁用状态，并且在启用该功能之前不会产生任何额外费用。在集群中使用变更流时，会产生额外的读取和写入 IOs 和存储成本。可以使用 `modifyChangeStreams` API 为集群启用此功能。有关定价的更多信息，请参阅 Amazon DocumentDB 定价

限制

变更流在 Amazon DocumentDB 中存在以下限制：

- 在 Amazon DocumentDB 3.6 和 Amazon DocumentDB 4.0 上，变更流只能通过与 Amazon DocumentDB 集群主实例的连接打开。Amazon DocumentDB 3.6 和 Amazon DocumentDB 4.0 不支持从副本实例上的变更流中读取信息。在调用 `watch()` API 操作时，您必须指定 `primary` 读取首选项，以确保所有读取都定向到主实例（请参阅示例部分）。
- 在 Amazon DocumentDB 5.0 上，可以从主实例和辅助实例（包括全局集群）打开变更流。可以指定辅助读取首选项，将变更流重定向至辅助实例。有关其他最佳实践和限制，请参阅 在辅助实例上使用变更流。
- 写入集合的变更流的事件最多可在 7 天（默认为 3 小时）内使用。变更流数据将在日志保留时段过后删除，即使没有发生新更改也是如此。
- 对 `updateMany` 或 `deleteMany` 之类的集合执行长时间运行的写入操作时，会暂时延迟变更流事件的写入，直至长时间运行的写入操作完成为止。
- Amazon DocumentDB 不支持 MongoDB 操作日志 (oplog)。
- 使用 Amazon DocumentDB，您必须明确在给定集合上启用变更流。
- 如果变更流事件的总大小（包括变更数据，在请求的情况下还包括完整文档）大于 16 MB，客户端将在变更流上遇到读取失败情况。
- 当使用 `db.watch()` 和 `client.watch()` 搭配 Amazon DocumentDB 3.6 时，目前不支持 Ruby 驱动程序。
- 当该字段的更新值与之前的值相同时，在 Amazon DocumentDB 中，`updateDescription` 命令在变更流中的输出与 MongoDB 中不同：
  - 如果 `$set` 命令指定了所提供的字段，并且其目标值已经等于源值，则 Amazon DocumentDB 不会在 `updateDescription` 输出中返回字段。
  - 即使指定值等于当前值，MongoDB 也会在输出中返回此字段。

启用变更流

您可以为给定数据库中的所有集合启用 Amazon DocumentDB 变更流，也可以只针对选定集合启用。下面是如何使用 mongo Shell 为不同使用案例启用变更流的示例。在指定数据库和集合名称时，将空字符串视为通配符。

```
//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});
```

如果满足以下任意条件，则将为集合启用变更流：

- 数据库和集合均已明确启用。
- 包含该集合的数据库已启用。
- 所有数据库均已启用。

如果父数据库也启用了变更流，或者集群中的所有数据库都已启用，则从数据库中删除集合不会禁用该集合的变更流。如果创建了与已删除收藏同名的新收藏集，则将为该集合启用变更流。

您可以使用 `$listChangeStreams` 聚合管道阶段列出集群所有已启用的变更流。Amazon DocumentDB 支持的所有聚合阶段都可以在管道中用于额外处理。如果以前启用的某个集合被禁用，则该集合将不会显示在 `$listChangeStreams` 输出中。

```
//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:{}}));
```

示例：在 Python 中使用变更流

以下是在集合级别使用带有 Python 的 Amazon DocumentDB 变更流的示例。

```
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}}}
"""
```

以下是在数据库级别使用带有 Python 的 Amazon DocumentDB 变更流的示例。

```
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'}
"""
```

完整文档查找

更新变更事件不包括完整文档；只包括已执行的变更。如果您的使用案例需要用到受更新影响的完整文档，则可以在打开流时启用完整文档查找。

更新变更流事件的 `fullDocument` 文档会指明文档查找时已更新文档的最新版本。如果在更新操作与 `fullDocument` 查找之间发生了变更，则 `fullDocument` 文档可能无法指明更新时的文档状态。

要创建启用更新查找的流对象，请使用以下示例：

```
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()
```

流对象输出类似如下：

```
{'_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}}}
```

恢复变更流

您可以在以后通过使用恢复令牌来恢复变更流，该令牌相当于上次检索的变更事件文档的 `_id` 字段。

```
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}}
"""
```

start ##### AtOperationTime

您可以稍后使用 startAtOperationTime 从特定时间戳恢复变更流。

> **Note**
>
> Amazon DocumentDB 4.0+ 提供了使用 startAtOperationTime 的功能。使用 startAtOperationTime 时，变更流光标将仅返回在指定时间戳或之后发生的更改。startAtOperationTime 和 resumeAfter 命令是互斥的，因此不能一起使用。

```
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}}
"""
```

BatchResumeToken

Amazon DocumentDB 变更流现在会返回一个名为 postBatchResumeToken 的附加字段。此字段由 $changestream 命令和 getMore 命令返回。

Python 中的 $changestream 命令示例：

```
db.command({"aggregate": "sales", "pipeline": [{ "$changeStream": {}}], "cursor":
 {"batchSize": 1}})
```

预期输出：

```
cursor" : {
   "firstBatch" : [ ],
   "postBatchResumeToken" : {"_data" : "0167c8cbe60000000004"},
   "id" : NumberLong("9660788144470"),
   "ns" : "test.sales"
}
```

Python 中的 getMore 命令示例：

```
db.command({"getMore": NumberLong(<cursor id>), "collection": "sales", "batchSize":
 1 })
```

预期输出

```
cursor" : {
   "nextBatch" : [ ],
   "postBatchResumeToken" : {"_data" : "0167c8cbe60000000004"},
   "id" : NumberLong("9660788144470"),
   "ns" : "test.sales"
}
```

该 postBatchResumeToken 字段可用于在 resumeAfter 字段中打开新的变更流光标，其使用方式与恢复令牌的类似。

打开一个在所选 postBatchResumeToken 后开始的流：

```
post_batch_resume_token = output['cursor']['postBatchResumeToken']
stream = db.watch(full_document='updateLookup', resume_after=post_batch_resume_token)
```

常规恢复令牌始终对应于反映实际事件的操作日志（oplog）条目，与常规恢复令牌不同，postBatchResumeToken 对应于变更流在服务器上扫描到的操作日志条目，该条目未必是匹配的变更。

尝试使用旧的常规恢复令牌进行恢复将强制数据库扫描指定时间戳与当前时间之间的所有操作日志条目。这可能会在内部生成大量查询，每个子查询都会扫描一小段时间。这将导致 CPU 使用率激增和数据库性能降低。使用最后一个 postBatchResumeToken 进行恢复时，将跳过扫描不匹配的操作日志条目。

变更流中的事务

更改流事件将不包含来自未提交的中 and/or 止事务的事件。例如，如果您启动的事务包含一个 INSERT 操作和一个 UPDATE 操作，当 INSERT 操作成功但 UPDATE 操作失败时，该事务将回滚。由于此事务已回滚，您的变更流将不包含此事务的任何事件。

修改变更流日志保留期限

您可以使用 AWS 管理控制台 或将更改流日志的保留期限修改为 1 小时到 7 天之间 AWS CLI。

Using the AWS 管理控制台

修改变更流日志保留期限的步骤

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择参数组。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在参数组内，选择与集群相关联的集群参数组。要识别与集群关联的集群参数组，请参阅 确定 Amazon DocumentDB 集群的参数组。

4. 所得页面显示您的集群参数组的参数及它们的相应详情。选择 change_stream_log_retention_duration 参数。
5. 在页面右上角，选择编辑以更改参数的值。可以将 change_stream_log_retention_duration 参数修改为 1 小时到 7 天之间。
6. 进行更改，然后选择修改集群参数以保存更改。要放弃更改，请选择取消。

Using the AWS CLI

要修改集群参数组的 change_stream_log_retention_duration 参数，请使用带以下参数的 modify-db-cluster-parameter-group 操作：

- **--db-cluster-parameter-group-name**：必需。您正在修改的集群参数组的名称。要识别与集群关联的集群参数组，请参阅 确定 Amazon DocumentDB 集群的参数组。
- **--parameters**：必需。您正在修改的参数的名称。每个参数条目必须包含以下内容：
  - **ParameterName**：您正在修改的参数的名称。在本例中，为 change_stream_log_retention_duration。
  - **ParameterValue**：此参数的新值。
  - **ApplyMethod**：您希望如何应用对此参数的更改。允许的值为 immediate 和 pending-reboot。

> **Note**
>
> 带 static 的 ApplyType 参数必须具有 pending-reboot 的 ApplyMethod。

1. 要更改参数 change_stream_log_retention_duration 的值，请运行以下命令并替换 parameter-value 为要修改参数的值。

   对于 Linux、macOS 或 Unix：

```
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"
```

   对于 Windows：

```
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"
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "DBClusterParameterGroupName": "sample-parameter-group"
}
```

2. 至少等待 5 分钟。
3. 列出 sample-parameter-group 参数值以确保已进行更改。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

对于 Windows：

```
aws docdb describe-db-cluster-parameters ^
    --db-cluster-parameter-group-name sample-parameter-group
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "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**
>
> 在日志大小大于 (>) 51,200MB 之前，变更流日志保留期不会删除早于配置 `change_stream_log_retention_duration` 值的日志。

在辅助实例上使用变更流

要开始在辅助实例上使用变更流，请以 `readPreference` 作为辅助实例打开变更流指针。

可以打开变更流指针来监视集群或数据库中特定集合或所有集合上的变更事件。您可以在任何 Amazon DocumentDB 实例上打开变更流指针，并从写入器和读取器实例获取变更流文档。您可以在写入器和读取器实例上打开的不同变更流指针之间共享变更流令牌（例如 `resumeToken` 或 `startOperationTime`）。

示例

```
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)
```

辅助实例上的变更流准则和限制

- 变更流事件需要从主实例复制到辅助实例。您可以在 Amazon 中根据该DBInstanceReplicaLag指标监控延迟 CloudWatch。
- 辅助实例上的时间戳可能并非一直与主实例同步。在这种情况下，预计辅助实例的时间戳会产生延迟，这样它才能赶得上。作为最佳实践，我们建议使用 `startAtOperationTime` 或 `resumeToken` 在辅助实例上启动监视。
- 如果文档太大，您在进行 `fullDocumentLookup`，而主实例上的并发写入工作负载很高，那么辅助实例上的吞吐量可能低于主实例。作为最佳实践，我们建议在辅助实例上监控缓冲区的缓存命中率，并确保缓冲区的缓存命中率很高。

在 Amazon DocumentDB 8.0 中使用排序规则

亚马逊 DocumentDB 8.0 现在支持排序规则。排序规则允许您配置特定语言的字符串比较规则。使用排序规则，您可以指定区分大小写的比较规则，也可以指定语言区域。在 DocumentDB 8.0 中，可以在集合级别或索引级别配置排序规则。在 DocumentDB 中使用排序规则时，将在内部使用以下参数创建一个归类文档。

```
{
locale: string,
caseLevel: boolean,
caseFirst: string,
strength: int,
numericOrdering: boolean,
alternate: string,
```

```
maxVariable: string,
backwards: boolean,
normalization: boolean
}
```

限制

归类在 Amazon DocumentDB 中有以下限制：

- 排序规则与亚马逊 DocumentDB 8.0 中提供的计划器 v3 兼容。切换到 plannerv2 或 plannerv1 可能会导致行为不一致，包括 “未找到索引” 错误。
- 由于库固有的差异，如果导出带有排序规则的 mongodb 集合，则需要在迁移之前更新 metadata.bson 文件并将其版本从 57.1 更改为 60.2。
- 在极少数情况下，您的排序规则设置可能会违反字符数的内部限制，从而导致以下错误。“错误：归类文档的非默认属性超出了支持的范围。请减少选项的数量。” 在这种情况下，请尝试减少您在归类文档中提供的选项，或者您可以尽可能尝试使用默认值。

在 Amazon DocumentDB 8.0 中使用视图

- 亚马逊 DocumentDB 8.0 现在支持视图。视图充当虚拟集合，根据指定的聚合操作呈现数据。创建视图时，您可以定义一个用于转换来自一个或多个源集合的数据的查询。每次访问视图时，Amazon DocumentDB 8.0 都会执行此查询，而不会消耗额外的存储资源。与标准集合不同，Amazon DocumentDB 8.0 中的视图不将文档存储在磁盘上，因此它们是向应用程序呈现转换后或经过筛选的数据的有效解决方案。要在亚马逊文档数据库中创建视图，你可以使用 createView 命令或 db.createView () 帮助器方法：

  ```
  db.createView("viewName","sourceCollection",
  [
      { $match: { status: "active" } },
      { $project: { _id: 1, name: 1, email: 1 } }
  ]
  )
  ```

  这将创建一个基于 “SourceCollection” 的名为 “ViewName” 的视图，该视图仅包含活动文档和项目，仅包括_id、name和电子邮件字段。Amazon DocumentDB 中的视图为只读视图。对视图进行写入操作将返回错误。要在大型数据集中获得最佳性能，您可以构建视图管道以最大限度地提高效率。对于

复杂的聚合管道，建议使用 $match 阶段作为管道的第一阶段或早期阶段，以减少后续阶段需要处理的文档数量，从而提高查询性能。

最佳实践

下面列出了视图应遵循的一些最佳实践。

- 尽早过滤：在视图管道中尽早使用 $match 阶段来减少处理的数据量。
- 避免复杂聚合：对于经常访问且具有复杂聚合的视图，请考虑使用预先计算的结果创建一个单独的集合，并定期更新。
- 索引规划：确保视图管道中使用的字段，尤其是在 $match 和 $sort 操作中使用的字段，已在源集合上正确编制索引。
- 查询优化：使用 explain 命令来了解视图查询是如何执行的，并进行相应的优化。
- 视图的替代方案：鉴于 Amazon DocumentDB 和 MongoDB 视图之间的功能差异，在遇到限制时，可以考虑使用带有计划更新的常规集合作为视图的替代方案。

聚合器运算符兼容性

Amazon DocumentDB 在视图定义中支持许多聚合运算符，同时继续扩展兼容性。使用视图时，请重点关注以下支持的运算符：

- $match 用于筛选文档
- $project 用于字段选择和转换
- $addFields 用于添加计算字段
- $sort 用于对结果进行排序
- $limit 和 $skip 用于分页

一些专门的运算符，例如 $currenTop、$replaceRoot 和 $geoNear，目前在直接聚合查询中工作，而不是视图定义。

利用索引和视图

Amazon DocumentDB 8.0 中的视图使用基础集合的索引。因此，您无法直接在视图上创建、删除或重建索引。但是，在源集合上精心设计的索引可以显著提高视图查询性能。以下是一些优化视图查询性能的步骤：

- 确保视图管道中使用的源集合字段上存在适当的索引，尤其是在$match和$sort操作中
- 使用 explain () 方法分析查询执行计划并验证索引使用情况。
  E.g。 ，`db.viewName.find({...}).explain()`

使用 AWS Lambda 有变更流

Amazon DocumentDB 已与集成， AWS Lambda 因此您可以使用 Lambda 函数来处理变更流中的记录。Lambda 事件源映射是一种资源，它可以用来调用 Lambda 函数，以便处理不直接调用 Lambda 的 Amazon DocumentDB 事件。以 Amazon DocumentDB 变更流作为事件源，您可以构建响应数据变化的事件驱动型应用。例如，您可以使用 Lambda 函数来处理新文档、跟踪现有文档的更新或记录已删除的文档。

您可以配置事件源映射以发送来自 Amazon DocumentDB 变更流的记录至 Lambda 函数。事件可以一次一个发送或批量发送以提高效率，将按顺序处理。您可以根据特定的时间窗持续时间（0-300 秒）或批处理记录计数（最大限值为 10,000 条记录），配置事件源映射的批处理行为。您可以创建多个事件源映射，以使用多个 Lambda 函数处理相同的数据，或使用单个函数处理来自多个流的不同项目。

如果您的函数返回错误，则 Lambda 将重试批处理，直到它成功处理。在变更流中事件已过期的情况下，Lambda 将禁用事件源映射。在这种情况下，您可以创建新的事件源映射，并用您选择的起始位置对其配置。由于轮询器的分布式特性，Lambda 事件源映射至少处理一次事件。因此，在极少数情况下，Lambda 函数可能会收到重复的事件。遵循使用 AWS Lambda 函数的最佳实践，构建等性函数，以避免与重复事件相关的问题。有关更多信息，请参阅AWS Lambda 开发者指南中的AWS Lambda console 与亚马逊 DocumentDB 配合使用。

作为性能最佳实践，Lambda 函数需要短时间运行。为避免引入不必要的处理延迟，它也不应执行复杂的逻辑。特别是对于高速流，最好是触发异步后处理 Step Function 工作流，而不是长时间运行的 Lambda 函数。有关的更多信息 AWS Lambda，请参阅《AWS Lambda 开发人员指南》。

限制

以下是使用亚马逊 DocumentDB 时需要考虑的限制，以及： AWS Lambda

- AWS Lambda 目前仅支持亚马逊 DocumentDB 4.0 和 5.0。
- AWS Lambda 弹性集群或全局集群目前不支持。
- AWS Lambda 有效载荷大小不能超过 6MB。有关 Lambda 批量大小的更多信息，请参阅AWS Lambda 开发者指南中 Lambda 事件源映射部分的“批处理行为”。

Amazon DocumentDB 弹性集群

Amazon DocumentDB 弹性集群支持 reads/writes 每秒数百万和存储容量为 PB 的工作负载。弹性集群还通过取消选择、管理或升级实例的需求，简化开发人员如何与 Amazon DocumentDB 互动。

> **Note**
>
> 弹性集群使用兼容 MongoDB 5.0 的有线协议。它们在 Amazon DocumentDB 引擎版本 8.0 上不受支持，也不适用于中国地区或。 AWS GovCloud

Amazon DocumentDB 弹性集群创建用于：

- 为客户提供解决方案，这些客户寻求提供几乎无限扩展、附带丰富查询功能和 MongoDB API 兼容性的数据库。
- 为客户提供更高连接限值，缩减修补造成的停机时间。
- 继续为 JSON 工作负载投资于云原生、弹性和一流的架构。

主题

- 弹性集群用例
- 弹性集群的优势
- 弹性集群区域和版本可用性
- 限制
- Amazon DocumentDB 弹性集群：工作原理
- 开始使用 Amazon DocumentDB 弹性集群
- Amazon DocumentDB 弹性集群最佳实践
- 管理 Amazon DocumentDB 弹性集群
- 用于 Amazon DocumentDB 弹性集群的静态数据加密
- Service-linked 弹性集群中的角色

弹性集群用例

对于需要一个灵活架构以实现快速迭代开发的工作负载来说，文档数据库很有用。有关示例性 Amazon DocumentDB 用例，请参阅 文档数据库使用案例。

以下是用例的一些示例， 弹性集群可以为这些用例提供显著的优势：

用户资料

由于文档数据库具有灵活架构，因此它们可以大规模地存储具有不同属性和数据值的文档。弹性集群是在线资料的实际解决方案，其中不同的用户提供不同类型的信息。假设您的应用支持数亿个用户配置文件。您可以使用弹性集群支持此类应用，因为它们可以横向和纵向扩展，以支持对这些用户配置文件的数百万次读写操作。您也可以在非高峰时段缩减，以降低成本。

内容管理和历史记录

要有效地管理内容，您必须能够从各种来源收集和汇总内容，然后将其交付给客户。由于其灵活的架构，文档数据库非常适合用于收集和存储任何类型的数据。您可以使用它们来创建和引入新类型的内容，包括用户生成的内容，如图像、评论和视频。随着时间推移，数据库可能需要更多存储空间。借助弹性集群，您可以将数据分发到更多的存储卷上，让您能能够在单一集群中存储数千万亿字节数据。

弹性集群的优势

AWS 服务集成

亚马逊 DocumentDB 弹性集群与其他 AWS 服务集成的方式与亚马逊 DocumentDB 的集成方式相同：

- 迁移 — 您可以使用 AWS Database Migration Service (DMS) 从 MongoDB 和其他关系数据库迁移到 Amazon DocumentDB 弹性集群。
- 监控 — 您可以使用 Amazon 监控弹性集群的运行状况和性能 CloudWatch。
- 安全 — 您可以通过 AWS Identity and Access Management (IAM) 设置身份验证和授权，以管理您的弹性集群资源，例如创建和更新弹性集群，并使用 Amazon VPC 进行安全 VPC-only 连接。不支持对弹性集群使用 IAM 身份验证登录。
- 数据管理 — 您可以使用 AWS Glue from/to 其他 AWS 服务（例如亚马逊 S3、Amazon Redshift 和亚马逊服务）来导入和导出数据。 OpenSearch

弹性集群区域和版本可用性

弹性集群的区域可用性

下表显示了当前可用 Amazon DocumentDB 弹性集群的 AWS 区域以及每个区域的终端节点。

区域名称	Region	可用区
美国东部（弗吉尼亚州北部）	us-east-1	5
美国东部（俄亥俄州）	us-east-2	3
美国西部（俄勒冈州）	us-west-2	3
亚太地区（香港）	ap-east-1	3
亚太地区（孟买）	ap-south-1	3
亚太地区（首尔）	ap-northeast-2	3
亚太地区（新加坡）	ap-southeast-1	3
亚太地区（悉尼）	ap-southeast-2	3
亚太地区（东京）	ap-northeast-1	3
加拿大（中部）	ca-central-1	3
南美洲（圣保罗）	sa-east-1	3
欧洲地区（法兰克福）	eu-central-1	3
欧洲地区（爱尔兰）	eu-west-1	3
欧洲地区（伦敦）	eu-west-2	3
欧洲地区（米兰）	eu-south-1	3
欧洲地区（巴黎）	eu-west-3	3

版本可用性

弹性集群支持兼容 MongoDB 5.0 的性线路协议。有关基于 Amazon DocumentDB 4.0 实例的集群与弹性集群之间的差异，请参阅 Amazon DocumentDB 4.0 与弹性集群之间的功能差异。注意：亚马逊 DocumentDB 8.0 不支持弹性集群

限制

弹性集群管理

此版本不支持以下集群管理特性和功能：

- 自定义端口（仅支持 27017）
- 能够使用 IAM 用户和角色对数据库进行身份验证
- 创建全局集群的能力
- 现有 Amazon DocumentDB 事件和事件订阅
- 范围分片
- 对现有集合分片
- Multi-field 碎片密钥
- 更改分片密钥
- Point-in-time 恢复
- 克隆
- Performance Insights

> **Note**
>
> 有关弹性集群限值的信息，请参阅 Amazon DocumentDB 配额和限制。

查询和写入操作

此版本不支持以下查询和写入操作命令和功能：

- 扩展操作期间的 DDL 命令
- Profiler
- 参数组
- AWS Config
- AWS Backup

集合和索引管理

此版本不支持以下集合和索引管理功能：

- 唯一索引
- 部分索引
- 文本索引
- 向量索引
- 文档压缩

管理和诊断

此版本不支持以下管理和诊断命令和功能：

- AWS Secrets Manager
- Role-based-access-control (RBAC) 自定义角色。
- 连接时，不支持 0 写入关注。
- 更改当前未分配给现有弹性集群的 VPC 所属的子网。

Opt-in features

此版本不支持以下 Amazon DocumentDB 选择加入功能：

- ACID 事务
- DDL/DML 审计
- 变更流
- 会话命令

Amazon DocumentDB 弹性集群：工作原理

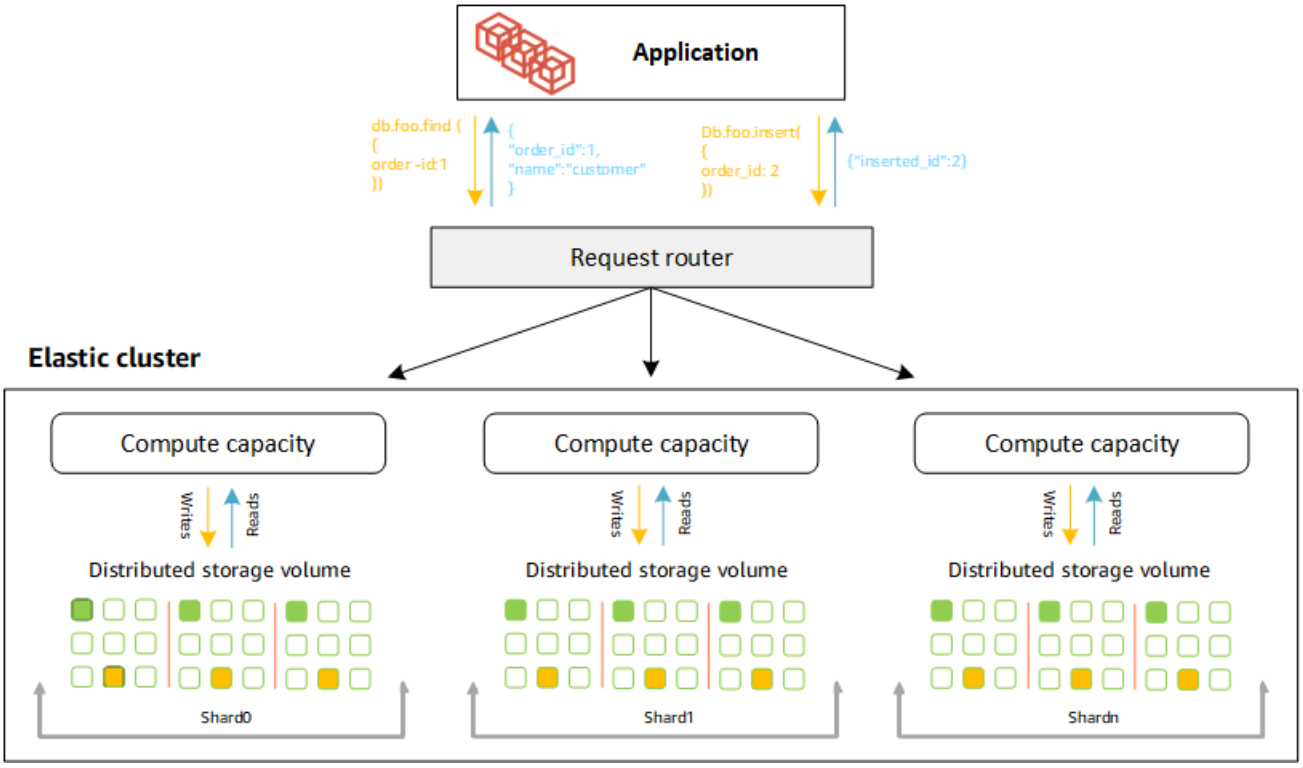
本节的主题提供有关支持 Amazon DocumentDB 弹性集群的机制和功能的信息。

主题

- Amazon DocumentDB 弹性集群分片
- 弹性集群迁移
- 弹性集群扩展
- 弹性集群可靠性
- 弹性集群存储和可用性
- Amazon DocumentDB 4.0 与弹性集群之间的功能差异

Amazon DocumentDB 弹性集群分片

Amazon DocumentDB 弹性集群使用基于哈希的分片在分布式存储系统中对数据进行分区。分片（也称为分区）将大型数据集拆分为跨多个节点的小型数据集，使您能够横向扩展数据库超出垂直扩展限值。弹性集群使用 Amazon DocumentDB 中计算和存储的分离或“解耦”，使您能够相互独立地扩展。弹性集群不是通过在计算节点之间移动小块数据进行集合重新分区，而是在分布式存储系统中高效复制数据。



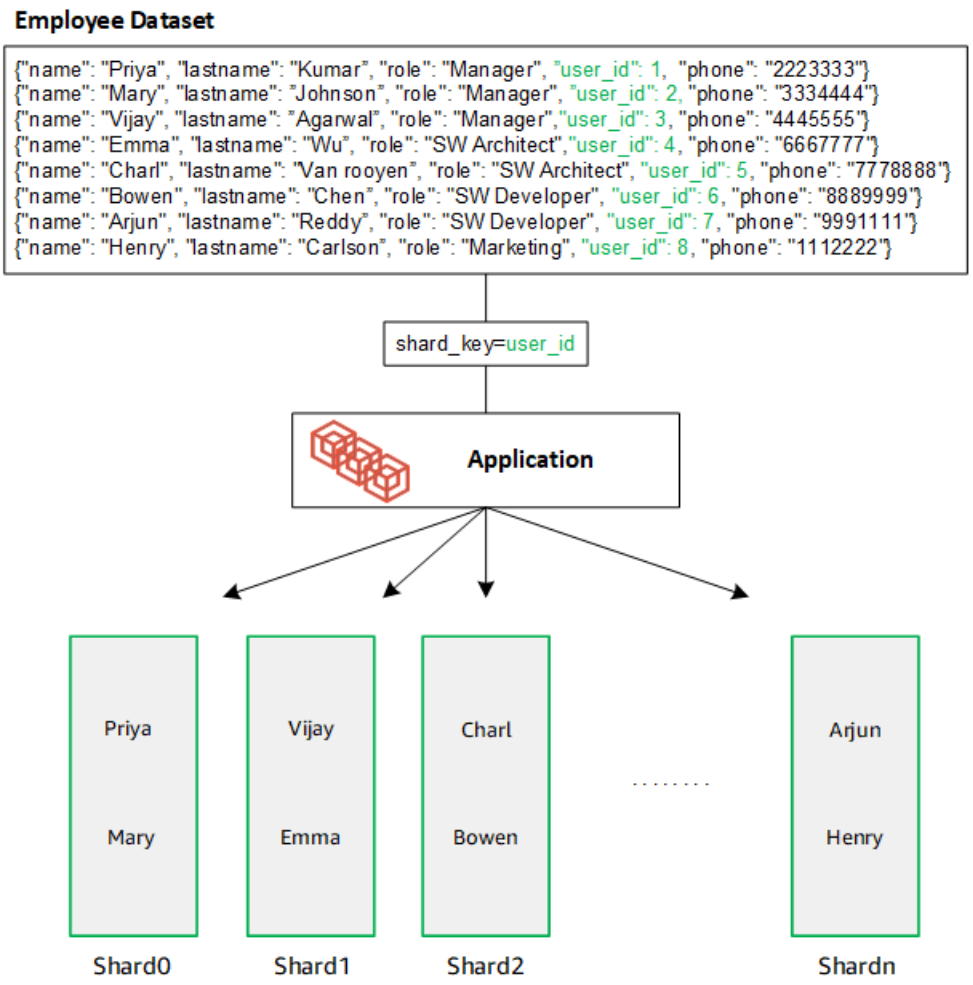
分片定义

分片命名法的定义：

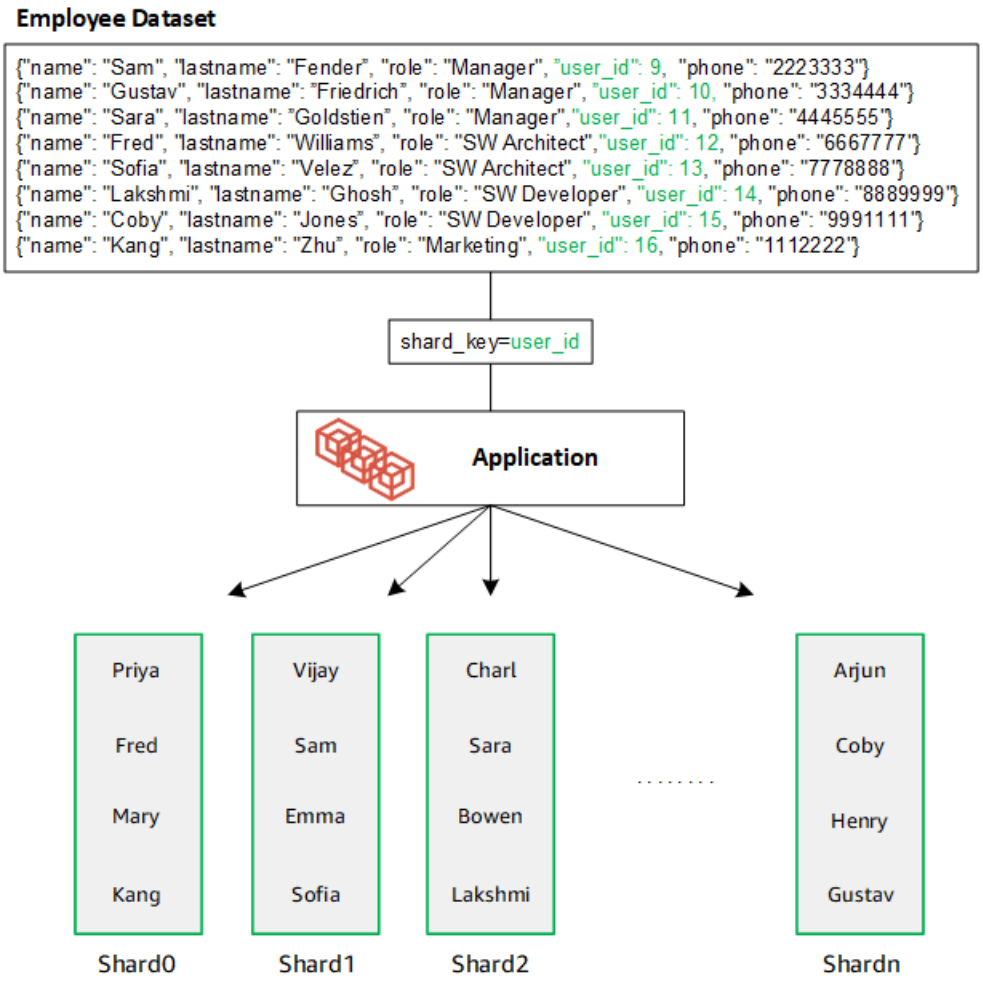
- 分片：一个分片为弹性集群提供计算。分片将有一个写入器实例和 0 到 15 个只读副本。默认情况下，一个分片将有两个实例：一个写入器和一个只读副本。您可以配置最多 32 个分片，每个分片实例可以有最多 64 个 vCPU。
- 分片密钥：分片密钥是已分片集合的 JSON 文档中的一个必填字段，弹性集群使用该字段向匹配的分片分配读写流量。
- 分片集合：分片集合是其数据跨数据分区中弹性集群分布的一个集合。
- 分区：分区是已分片数据的逻辑部分。创建已分片集合时，数据将根据分片密钥自动组织到每个分片内的分区中。每个分片都有多个分区。

跨已配置分片分配数据

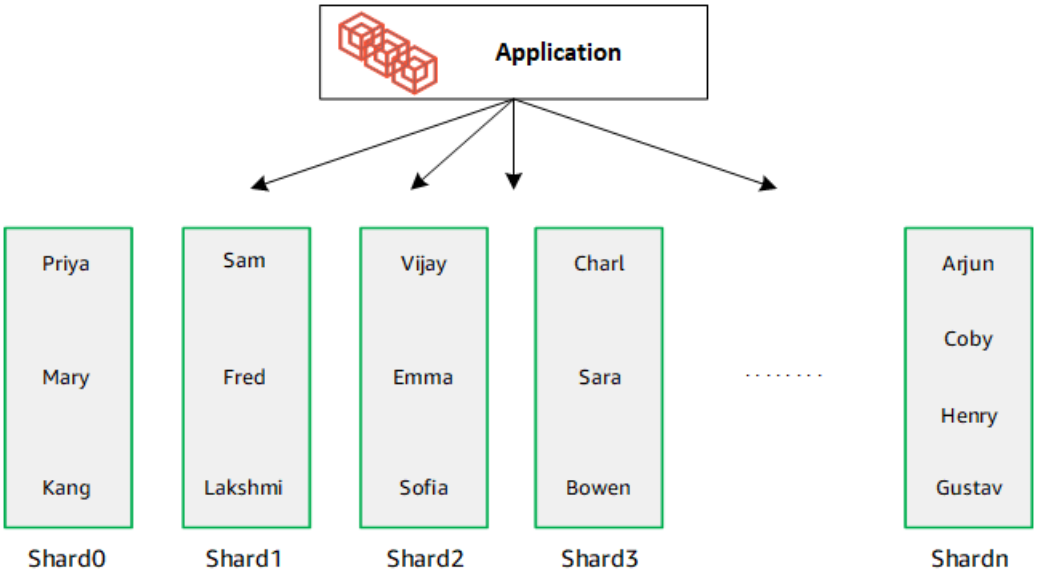
创建具有许多唯一值的分片密钥。良好分片密钥可以将您的数据跨底层分片均匀分配，从而让您的工作负载有最佳吞吐量和性能。以下示例是使用名为“user_id”的分片密钥的员工姓名数据：



DocumentDB 使用哈希分片跨底层分片进行数据分区。其他数据按相同方式插入和分配：



当您通过添加额外分片横向扩展数据库时，Amazon DocumentDB 会自动重新分配数据：



弹性集群迁移

Amazon DocumentDB 支持将分片的 MongoDB 数据迁移到弹性集群。支持离线、在线和混合迁移方法。有关更多信息，请参阅 迁移到 Amazon DocumentDB。

弹性集群扩展

Amazon DocumentDB 弹性集群可以增加弹性集群中分片数目（横向扩展）以及应用于每个分片的 vCPU 数目（纵向扩展）。您还可以根据需要减少分片数目和计算容量 (vCPU)。

有关扩展的最佳实践，请参阅 扩展弹性集群。

> **Note**
>
> Cluster-level 缩放也可用。有关更多信息，请参阅 扩展 Amazon DocumentDB 集群。

弹性集群可靠性

Amazon DocumentDB 的设计具有可靠、持久和容错的特点。为提高可用性，弹性集群按每个分片部署两个节点，跨不同可用区布置。Amazon DocumentDB 包括多种自动功能，使其成为很可靠的数据库解决方案。有关更多信息，请参阅 Amazon DocumentDB 可靠性。

弹性集群存储和可用性

Amazon DocumentDB 数据存储在集群卷中，该集群卷是使用固态驱动器 (SSD) 的单个虚拟卷。一个集群卷由六个数据副本组成，这些副本会在单个 AWS 区域的多个可用区之间自动复制。此复制有助于确保您的数据具有高持久性，减少数据丢失的可能性。它还有助于确保在故障转移期间您的集群具有更高可用性，因为您的数据副本已存在于其他可用区中。有关存储、高可用性和复制的更多详情，请参阅 Amazon DocumentDB：工作方式。

Amazon DocumentDB 4.0 与弹性集群之间的功能差异

Amazon DocumentDB 4.0 与弹性集群之间存在以下功能差异。

- 来自 `top` 和 `collStats` 的结果按分片分区。对于分片集合，数据分布在多个分区之间，`collStats` 报告了来自各分区的聚合 `collScans`。
- 当改变集群分片计数时，来自已分片集合的 `top` 和 `collStats` 的集合统计数据将重置。

- 备份内置角色现在支持 serverStatus。操作：具有备份角色的开发人员和应用程序可以收集有关 Amazon DocumentDB 集群状态的统计量。
- SecondaryDelaySecs 字段替换 replSetGetConfig 输出中的 slaveDelay。
- hello 命令替换 isMaster：hello 返回描述弹性集群角色的一个文档。
- 弹性集群中的 $elemMatch 运算符仅匹配某数组第一个嵌套级别中的文档。在 Amazon DocumentDB 4.0 中，运算符在返回已匹配文档之前遍历所有级别。例如：

```
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 4.0 中的“$”投影返回带所有字段的所有文档。对于弹性集群，带“$”投影的 find 命令返回匹配查询参数的文档，该参数仅含有匹配“$”投影的字段。
- 在弹性集群中，带 $regex 和 $options 查询参数的 find 命令返回一个错误：“无法同时在 $regex 和 $options 中设置选项。”
- 对于弹性集群，以下情况时，$indexOfCP 现在返回“-1”：

- 未在 `string expression` 中找到子字符串，或
- `start` 是一个大于 `end` 的数字，或
- `start` 是一个大于字符串字节长度的数字。

在 Amazon DocumentDB 4.0 中，当 `start` 位置是一个大于 `end` 或字符串字符串字节长度的数字时，`$indexOfCP` 返回“0”。

- 对于弹性集群，`_id fields` 中的投影操作，例如 `{"_id.nestedField" : 1}`，返回仅包含所投影字段的文档。同时，在 Amazon DocumentDB 4.0 中，嵌套字段投影命令不会筛选掉任何文档。

开始使用 Amazon DocumentDB 弹性集群

这个入门部分将向您介绍如何创建和查询您的第一个弹性集群。

有许多连接和开始使用 Amazon DocumentDB 的方式。以下过程是用户开始使用我们强大文档数据库的最快捷、最简便的方法。本指南使用 AWS CloudShell 直接从 AWS 管理控制台连接和查询 Amazon DocumentDB 集群。符合 AWS 免费套餐条件的新客户可以免费使用亚马逊 DocumentDB CloudShell。如果您的 AWS CloudShell 环境或 Amazon DocumentDB 弹性集群使用了超出免费套餐范围的资源，则您需要为这些资源支付正常 AWS 费率。本指南将在不到 5 分钟的时间内让你开始使用亚马逊 DocumentDB。

主题

- 先决条件
- 第 1 步：创建弹性集群
- 步骤 2：连接到您的弹性集群
- 步骤 3：对您的集合分片，插入和查询数据
- 第 4 步：探索

先决条件

在创建第一个 Amazon DocumentDB 集群之前，您必须执行以下操作：

已创建 Amazon Web Services（AWS）账户

在开始使用 Amazon DocumentDB 之前，您必须拥有 Amazon Web Services（AWS）账户。该 AWS 账户是免费的。您只需为使用的服务和资源付费。

如果您没有 AWS 账户，请完成以下步骤来创建一个。

报名参加 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明操作。

   在注册时，将接到电话或收到短信，要求使用电话键盘输入一个验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

设置所需的 AWS Identity and Access Management (IAM) 权限。

要管理集群、实例和集群参数组等 Amazon DocumentDB 资源，需要 AWS 能够对您的请求进行身份验证的证书。有关更多信息，请参阅 适用于 Amazon DocumentDB 的 Identity and Access Management。

1. 在的搜索栏中 AWS 管理控制台，键入 IAM，然后在下拉菜单中选择 IAM。
2. 一旦您进入 IAM 控制台，就从导航窗格中选择用户。
3. 选择您的用户名。
4. 选择添加权限。
5. 选择直接附加策略。
6. 在搜索栏中键入 `AmazonDocDBElasticFullAccess`，并且一旦它出现在搜索结果中就选择之。
7. 选择下一步。
8. 选择添加权限。

> **Note**
>
> 您的 AWS 账户在每个区域都包含一个默认 VPC。如果您选择使用 Amazon VPC，请按照《Amazon VPC 用户指南》的创建 Amazon VPC 主题中的步骤进行操作。

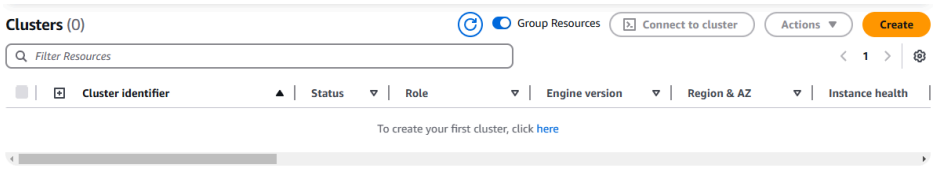
第 1 步：创建弹性集群

在本节中，我们将说明如何使用 AWS 管理控制台 或 AWS CLI 并按照以下说明创建全新的弹性集群。

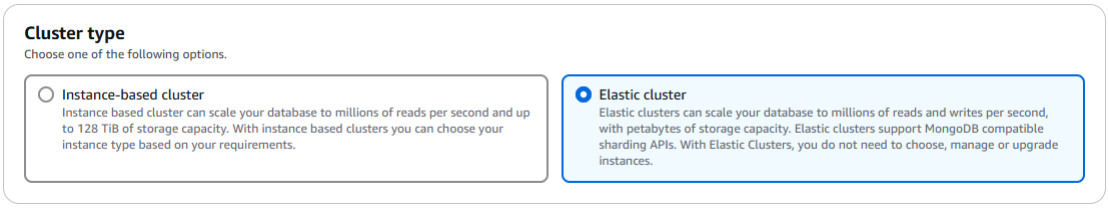
Using the AWS 管理控制台

使用 AWS 管理控制台创建弹性集群配置：

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在 Amazon DocumentDB 管理控制台上，集群下，选择创建。



3. 在创建 Amazon DocumentDB 集群页面上，在集群类型部分中选择弹性集群。



4. 在配置部分中，配置以下内容：

   a. 在集群名称字段中，输入唯一集群标识符（遵循字段下方的命名要求）。

   b. 在分片计数字段中，输入您在自身集群中想要的分片的数目。每个集群的最大分片数目为 32。

   > **Note**
   >
   > 将对每个分片部署两个节点。两个节点将具有相同的分片容量。

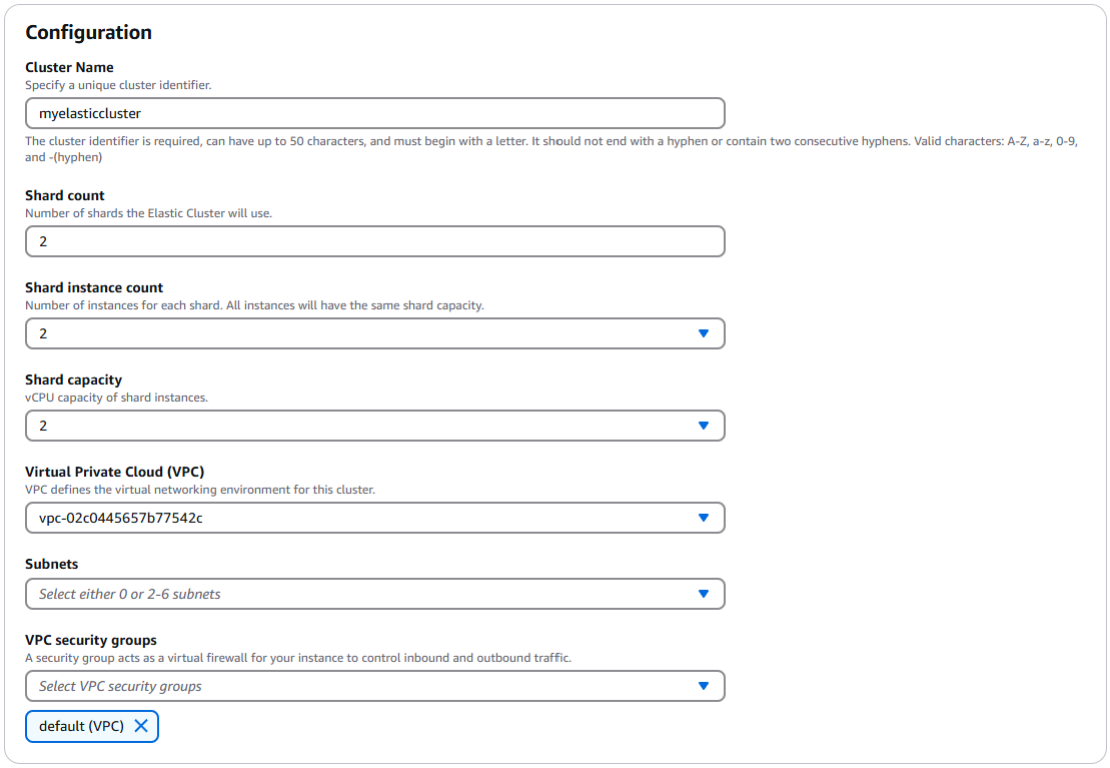
   c. 在分片实例数字段中，选择想要与每个分片关联的副本实例的数量。分片实例的最大数量为 16 个，以 1 个为增量。所有副本实例都有相同的分片容量，具体定义见以下字段。出于测试目的，默认值 2 应已足够。

> **Note**
>
> 副本实例的数量适用于弹性集群中的所有分片。分片实例计数值为 1 表示有一个写入器实例，其他任何实例都是可用于读取和提高可用性的副本。出于测试目的，默认值 2 应已足够。

d. 在分片容量字段中，选择您想要与每个分片实例关联的虚拟 CPU (vCPU) 的数目。每个分片实例的最大 vCPU 数目为 64。允许值为 2、4、8、16、32、64。出于测试目的，默认值 2 应已足够。

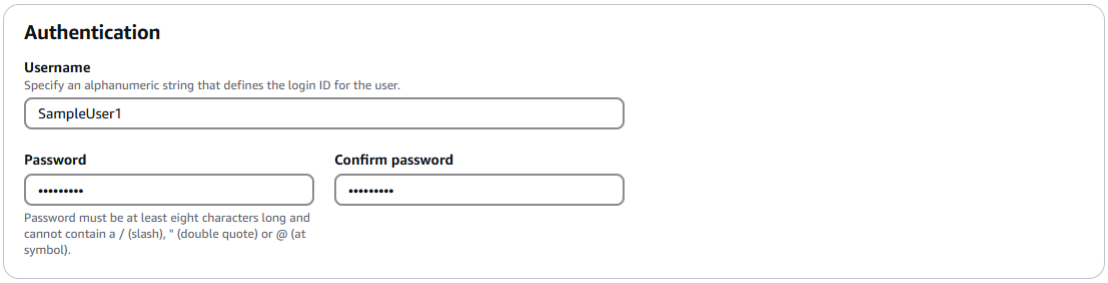
e. 在虚拟私有云（VPC）字段中，从下拉列表中选择一个 VPC。

f. 对于子网和 VPC 安全组，您可以使用默认值或选择您选定的三个子网和多达三个 VPC 安全组（最少一个）。



5. 在身份验证部分，在用户名字段中输入确定主用户登录名称的字符串。

   在密码字段中，输入符合说明的唯一密码，然后确认该密码。



6. 在加密部分中，保留默认设置（默认密钥）。

   或者，您可以输入自己创建的 AWS KMS key ARN。有关更多信息，请参阅 [用于 Amazon DocumentDB 弹性集群的静态数据加密](#)。

   > **Important**
   >
   > 必须对弹性集群启用加密。

7. 在备份部分，根据您的备份要求编辑字段。出于测试目的，您可以保留默认设置。



   a. 备份留存期：在列表中，选择在删除此集群的自动备份前保留它们的天数。

   b. 备份时段：设置 Amazon DocumentDB 要备份此集群的每日时间和持续时间。

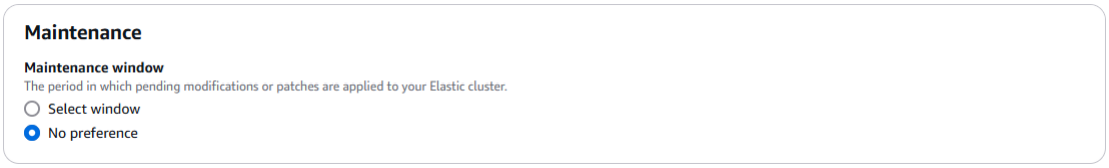
      i. 如果要配置创建备份的时间和时长，请选择选择时段。

         开始时间：在第一个列表中，选择开始自动备份的开始时间小时 (UTC)。在第二个列表中，选择您希望自动备份开始的时间（分钟）。

         持续时间：在该列表中，选择要向创建自动备份分配的小时数。

      ii. 如果想要 Amazon DocumentDB 选择创建备份的时间和时长，请选择无首选项。

8. 在维护部分中，选择对集群进行修改或修补的日期、时间和持续时间。出于测试目的，您可以保留默认设置。



9. 选择创建集群。

弹性集群现正在预配置。此过程可能需要数分钟完成。当弹性集群状态在集群列表中显示为可用时，可以连接到您的集群。

Using the AWS CLI

要使用创建弹性集群 AWS CLI，请使用带有以下参数的`create-cluster`操作：

- --cluster-name：必填项。创建期间输入或上次修改的弹性扩展集群的当前名称。
- --shard-capacity：必填项。分配给每个分片的 vCPU 的数目。最大值为 64。允许值为 2、4、8、16、32、64。
- --shard-count：必填项。分配给集群的分片的数目。最大值为 32。
- --shard-instance-count：可选。适用于此集群中所有分片的副本实例数量。最大值为 16。
- --admin-user-name：必填项。与管理用户关联的用户名。
- --admin-user-password：必填项。与管理用户关联的密码。
- --auth-type：必填项。用于确定从何处获取用于访问弹性集群的密码的身份验证类型。有效类型为 `PLAIN_TEXT` 和 `SECRET_ARN`。
- --vpc-security-group-ids：可选。要与此集群关联的 EC2 VPC 安全组的列表。
- --preferred-maintenance-window：可选。配置可进行系统维护的每周时间范围（采用通用协调时间（UTC））。

  格式为：`ddd:hh24:mi-ddd:hh24:mi`。有效值 (ddd)：
  Mon、Tue、Wed、Thu、Fri、Sat、Sun

  默认值为每个 Amazon Web Services 区域 8 小时的时间段中随机选择的 30 分钟时段（随机选取周中的某天进行）。

  至少 30 分钟的窗口。
- --kms-key-id：可选。配置已加密集群的 KMS 密钥标识符。

KMS 密钥标识符是 AWS KMS 加密密钥的亚马逊资源名称 (ARN)。如果使用拥有用于加密新集群的 KMS 加密密钥的同一 Amazon Web Services 账户创建集群，则可以使用 KMS 密钥别名而不是 KMS 加密密钥的 ARN。

如果中未指定加密密钥， KmsKeyId 且`StorageEncrypted`参数为真，则 Amazon DocumentDB 将使用您的默认加密密钥。

- --preferred-backup-window：可选。创建自动备份的每日首选时间范围。默认值是从 8 小时的时间段中随机选择一个 30 分钟的窗口。 AWS 区域
- --backup-retention-period：可选。自动备份的保留天数。默认值是 1。
- --storage-encrypted：可选。配置集群是已加密还是未加密。

  `--no-storage-encrypted` 指定集群未加密。
- --subnet-ids：可选。配置网络子网 ID。

在以下示例中，将每个 *user input placeholder* 替换为您自己的信息。

> **Note**
>
> 以下示例包括创建特定 KMS 密钥。要使用默认 KMS 密钥，请不要包含 `--kms-key-id` 参数。

对于 Linux、macOS 或 Unix：

```
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
```

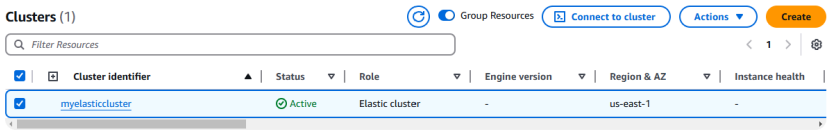
对于 Windows：

```
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
```

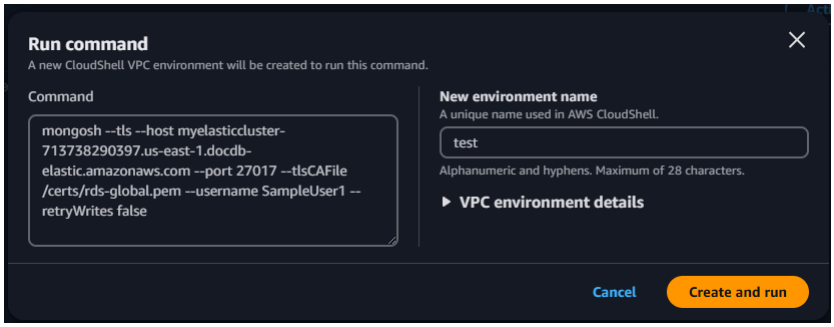
步骤 2：连接到您的弹性集群

使用连接到您的亚马逊 DocumentDB 弹性集群。 AWS CloudShell

1. 在 Amazon DocumentDB 管理控制台上的集群下，找到您创建的弹性集群。单击集群旁边的复选框，选择您的集群。



2. 选择 Connect to cluster（位于“操作”下拉菜单旁边）。只有在您单击集群旁边的复选框并且集群的状态显示为可用后，才会启用此按钮。将出现“ CloudShell 运行命令”屏幕。
3. 在新环境名称字段中，输入唯一的名称，例如“test”，然后选择创建并运行。将自动为您的 Amazon DocumentDB 数据库配置 VPC 环境详细信息。



4. 出现提示时，输入您在“步骤 1：创建 Amazon DocumentDB 弹性集群”（子步骤 5）中创建的密码。



在您输入密码并且提示符变成 `direct: mongos] <env-name>>` 后，表示您已成功连接到您的 Amazon DocumentDB 集群

> **Note**
>
> 有关对 流进行问题排查的更多信息，请参阅 Amazon DocumentDB 问题排查。

步骤 3：对您的集合分片，插入和查询数据

弹性集群增加对 Amazon DocumentDB 中分片过程的支持。既然您已连接到集群，您可以对集群分片、插入数据并运行数个查询。

1. 要对集合分片，请输入以下：

```
sh.shardCollection("db.Employee1" , { "Employeeid" : "hashed" })
```

2. 要插入单个文档，请输入以下内容：

```
db.Employee1.insertOne({"Employeeid":1, "Name":"Joe", "LastName": "Bruin", "level":
 1 })
```

以下输出显示：

```
WriteResult({ "nInserted" : 1 })
```

3. 要阅读您编写的文档，请输入以下`findOne()`命令（它返回单一文档）：

```
db.Employee1.findOne()
```

以下输出显示：

```
{
"_id" : ObjectId("61f344e0594fe1a1685a8151"),
"EmployeeID" : 1,
"Name" : "Joe",
"LastName" : "Bruin",
"level" : 1
}
```

4. 要执行若干更多查询，请考虑游戏配制文件用例。首先，将几个条目插入标题为“员工”的集合。输入以下信息：

```
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 }
])
```

以下输出显示：

```
{ acknowledged: true,
     insertedIds: {
        '0': ObjectId('679d02cd6b5a0581be78bcbd'),
        '1': ObjectId('679d02cd6b5a0581be78bcbe'),
        '2': ObjectId('679d02cd6b5a0581be78bcbf'),
        '3': ObjectId('679d02cd6b5a0581be78bcc0')
    }
}
```

5. 要返回配制文件集合中的所有文档，请输入find () 命令：

```
db.Employee.find()
```

您在步骤 4 中输入的数据显示。

6. 要查询单一文档，请纳入过滤器（例如：“Katie”）。输入以下信息：

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

以下输出显示：

```
[
   {
     _id: ObjectId('679d02cd6b5a0581be78bcc0'),
     Employeeid: 4,
     name: 'Katie',
     lastname: 'Schaper',
     level: 3
   }
]
```

7. 要查找配置文件并对其进行修改，请输入findAndModify命令。在此示例中，给予员工“Matt”更高等级，即 “14”：

```
db.Employee.findAndModify({
    query: { "Employeeid" : 1, "name" : "Matt"},
    update: { "Employeeid" : 1, "name" : "Matt", "lastname" : "Winkle", "level" :
 14 }
})
```

以下输出显示（请注意，级别尚未更改）：

```
{
   _id: ObjectId('679d02cd6b5a0581be78bcbd'),
   Employeeid: 1,
   name: 'Matt',
```

```
   lastname: 'Winkle',

   level: 12

}
```

8. 要验证级别提高，请输入以下查询：

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

以下输出显示：

```
[
   {

   _id: ObjectId('679d02cd6b5a0581be78bcbd'),

   Employeeid: 1,

   name: 'Matt',

   lastname: 'Winkle',

   level: 14

   }
]
```

第 4 步：探索

恭喜您！您已成功完成 Amazon DocumentDB 弹性集群的入门过程。

接下来做什么？ 了解如何充分利用这个数据库及其热门功能：

- Amazon DocumentDB 弹性集群最佳实践
- 管理 Amazon DocumentDB 弹性集群

> **Note**
>
> 您在此入门过程中创建的弹性集群将继续产生费用，除非您将其删除。有关指导，请参阅删除弹性集群。

Amazon DocumentDB 弹性集群最佳实践

了解使用 Amazon DocumentDB 弹性集群的最佳实践。所有基于实例的 Amazon DocumentDB 集群的最佳实践也适用于弹性集群。随着新的最佳实践的确定，此节将不断更新。

主题

- 选择分片键
- 连接管理
- 未分片的集合
- 扩展弹性集群
- 监控弹性集群

选择分片键

以下列表描述了创建分片键的指导原则。

- 使用均匀分发的哈希键分发数据遍及您集群中的所有分片（避免热键）。
- 在所有 read/update /delete 请求中使用分片密钥，以避免分散收集查询。
- 执行 read/update /delete 操作时，请避免使用嵌套分片键。
- 进行批量操作时，请将 `ordered` 设置成虚假，从而使所有分片可以并行运行并改善延迟。

连接管理

以下列表描述了管理与您数据库连接的指导原则。

- 监控您的连接计数以及新连接开闭的频率。
- 将您的连接分布到按您的应用程序配置的所有子网。如果您的集群配置在多个子网中，但您只使用这些子网的子集，则您可能在最大连接数上遇到瓶颈。

未分片的集合

以下内容描述了用于未分片集合的指导原则。

- 在处理未分片的集合时，为了分配负载，请尝试将高度利用的未分片集合保留在不同的数据库上。Amazon DocumentDB 弹性集群跨不同分片安置数据库，并将同一数据库的未分片集合共同定位在同一个分片上。

扩展弹性集群

以下列表描述了扩展弹性集群的指导原则。

- 扩展操作可能会导致短时间的间断性数据库和网络错误。如果可能，请避免在高峰时段扩展。尝试在维护窗口期间扩展。
- 相比增加或减少分片计数，首选向上和向下扩展分片容量（更改每个分片的 vCPU 计数）以增加计算能力，因为它更快且具有更短的间断性数据库和网络错误持续时间。
- 在预测增长时，最好增加分片计数，而非扩展分片容量。这使您能够通过增加需要快速扩展的场景的分片容量，来扩展您的集群。
- 监控您的客户端重试策略并使用指数回退和抖动进行重试，以避免在扩展情况下出现错误时数据库过载。
- 更改分片数量可能需要时间，因为数据必须安全地从一个分片移动到另一个分片。为了最大限度地降低出错风险，我们建议在更改过程中不要停止集群状态或对其进行任何其他更改。

监控弹性集群

以下列表描述了监控弹性集群的指导原则。

- 跟踪每个分片指标的峰值与平均值的比率，以确定您的流量是否不均衡（有热点）。key/hot跟踪峰值/平均值比率的关键指标是：
  - `PrimaryInstanceCPUUtilization`
    - 这可以在每个分片层面监控。
    - 在集群层面，您可以监视平均到 p99 的偏差。
  - `PrimaryInstanceFreeableMemory`
    - 这可以在每个分片层面监控。
    - 在集群层面，您可以监视平均到 p99 的偏差。

- `DatabaseCursorsMax`
  - 这应在每个分片层面监控以确定偏差。
- `Documents-Inserted/Updated/Returned/Deleted`
  - 这应在每个分片层面监控以确定偏差。

管理 Amazon DocumentDB 弹性集群

要管理 Amazon DocumentDB 弹性集群，您必须拥有带适当 Amazon DocumentDB 控制层面的 IAM policy略。这些权限使您能够创建、修改和删除集群。亚马逊 DocumentDBFullAccess 策略提供了管理亚马逊 DocumentDB 弹性集群所需的所有权限。

以下主题显示使用 Amazon DocumentDB 弹性集群时如何执行各种任务。

主题

- 修改弹性集群配置
- 监控弹性集群
- 删除弹性集群
- 管理弹性集群快照
- 停止和启动 Amazon DocumentDB 弹性集群
- 维护 Amazon DocumentDB 弹性集群

修改弹性集群配置

在本节中，我们将说明如何使用 AWS 管理控制台 或 AWS CLI 并按照以下说明修改弹性集群。

修改集群的主要用途是通过增加或减少分片计数 and/or 分片计算容量来扩展分片。

Using the AWS 管理控制台

要使用以下命令修改弹性集群配置 AWS 管理控制台：

1. 登录 AWS 管理控制台 并打开 Amazon DocumentDB 控制台。
2. 在导航窗格中，选择集群。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在导航窗格左上角选择菜单图标。

3. 在集群标识符列中选择您想要修改的集群的名称。
4. 选择 Modify(修改)。
5. 编辑您想要更改的字段，然后选择修改群集。

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**
>
> 或者，您可以通过以下方式访问修改集群对话框：转到集群页面，勾选紧邻您集群的复选框，选择操作，然后选择修改。

Using the AWS CLI

要使用修改弹性集群配置 AWS CLI，请使用带有以下参数的`update-cluster`操作：

- **`--cluster-arn`**：必填项。要删除的集群的 ARN 标识符。
- **`--shard-capacity`**：可选。分配给每个分片的 vCPU 的数目。最大值为 64。允许值为 2、4、8、16、32、64。
- **`--shard-count`**：可选。分配给集群的分片的数目。最大值为 32。
- **`--shard-instance`**-count：可选。适用于此集群中所有分片的副本实例数量。最大值为 16。
- **`--auth-type`**：可选。用于确定从何处获取用于访问弹性集群的密码的身份验证类型。有效类型为 `PLAIN_TEXT` 和 `SECRET_ARN`。
- **`--admin-user-password`**：可选。与管理用户关联的密码。
- **`--vpc-security-group-ids`**：可选。配置与这个集群待关联的一系列 Amazon EC2 安全组和 Amazon VPC (VPC) 安全组。
- **`--preferred-maintenance-window`**：可选。配置可进行系统维护的每周时间范围（采用通用协调时间（UTC））

  格式为：`ddd:hh24:mi-ddd:hh24:mi`。有效值 (ddd)：
  Mon、Tue、Wed、Thu、Fri、Sat、Sun

  默认值为每个 Amazon Web Services 区域 8 小时的时间段中随机选择的 30 分钟时段（随机选取周中的某天进行）。

  至少 30 分钟的窗口。
- **`--subnet-ids`**：可选。配置网络子网 ID。

在以下示例中，将每个 *`user input placeholder`* 替换为您自己的信息。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

要在修改后监控弹性集群的状态，请参阅监控弹性集群。

监控弹性集群

在本节中，我们将说明如何使用 AWS 管理控制台 或 AWS CLI 以及以下说明监控您的弹性集群。

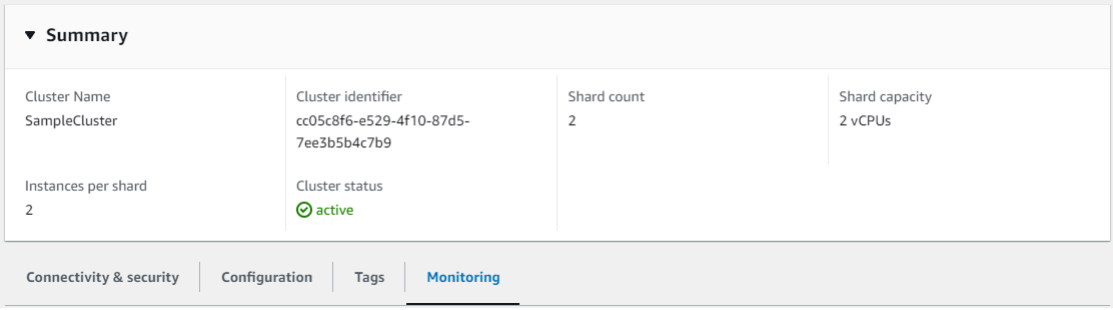
Using the AWS 管理控制台

要使用以下方法监控弹性集群配置 AWS 管理控制台：

1. 登录 AWS 管理控制台 并打开 Amazon DocumentDB 控制台。
2. 在导航窗格中，选择集群。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在导航窗格左上角选择菜单图标。

3. 在集群标识符列中选择您想要监控的集群的名称。
4. 选择监控选项卡。



Amazon CloudWatch 为以下监控类别显示了许多图表：

- 资源利用率
- 吞吐量
- 操作
- 系统

您也可以 CloudWatch 通过访问 Amazon AWS 管理控制台 ，为您的弹性集群设置自己的监控环境。

Using the AWS CLI

要使用监控特定的弹性集群配置 AWS CLI，请使用带有以下参数的`get-cluster`操作：

- **`--cluster-arn`**：必填项。要获取其信息的集群的 ARN 标识符。

在以下示例中，将每个 *`user input placeholder`* 替换为您自己的信息。

对于 Linux、macOS 或 Unix：

```
aws docdb-elastic get-cluster \
    --cluster-arn arn:aws:docdb-elastic:us-west-2:123456789012:cluster:/68ffcdf8-e3af-40a3-91e4-24736f2dacc9
```

对于 Windows：

```
aws docdb-elastic get-cluster ^
    --cluster-arn arn:aws:docdb:-elastic:us-west-2:123456789012:cluster:/68ffcdf8-e3af-40a3-91e4-24736f2dacc9
```

此操作的输出将类似于以下内容：

```
"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",
        ...
 }
```

有关更多信息，请参阅 Amazon DocumentDB 资源管理 API 参考中的 `DescribeClusterSnapshot`。

要使用查看所有弹性集群的详细信息 AWS CLI，请使用带有以下参数的`list-clusters`操作：

- **`--next-token`**：可选。如果项目输出的数量（`--max-results`）少于基础 API 调用所返回的项目总数，则输出将包含您可传递到后续命令的 `NextToken` 以检索下一组项目。
- **`--max-results`**：可选。命令的输出中要返回的项目总数。如果存在的记录数超过了指定的`max-results` 值，则在响应中包含分页记号（`next-token`），以便检索剩余的结果。
  - 默认值：100
  - 最小值 20，最大值 100

在以下示例中，将每个 *`user input placeholder`* 替换为您自己的信息。

对于 Linux、macOS 或 Unix：

```
aws docdb-elastic list-clusters \
    --next-token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ== \
    --max-results 2
```

对于 Windows：

```
aws docdb-elastic list-clusters ^
    --next-token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ== ^
    --max-results 2
```

此操作的输出将类似于以下内容：

```
{
   "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"
       }
   ]
}
```

删除弹性集群

在本节中，我们将说明如何使用 AWS 管理控制台 或 AWS CLI 以及以下说明删除弹性集群。

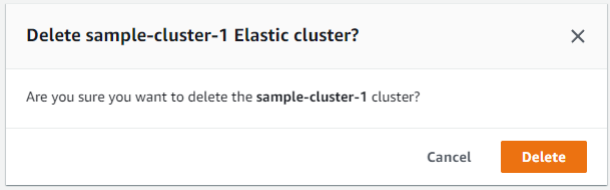
Using the AWS 管理控制台

要使用 AWS 管理控制台删除弹性集群配置：

1. 登录 AWS 管理控制台 并打开 Amazon DocumentDB 控制台。
2. 在导航窗格中，选择集群。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在导航窗格左上角选择菜单图标。

3. 在集群列表表中，选择在您想要删除的集群名称左侧的复选框，然后选择操作。从下拉菜单中，选择删除。
4. 在是否删除“集群名称” 弹性集群？对话框中，选择删除。



删除集群需要几分钟时间。要监控集群的状态，请参阅 监控 Amazon DocumentDB 集群的状态。

Using the AWS CLI

要使用删除弹性集群 AWS CLI，请使用带有以下参数的`delete-cluster`操作：：

- **`--cluster-arn`**：必填项。要删除的集群的 ARN 标识符。
- **`--no-skip-final-backup`**：可选。如果您需要最终备份，则必须包含该参数和最终备份的名称。必须包含 `--final-backup-identifier` 或 `--skip-final-backup`。
- **`--skip-final-backup`**：可选。仅当您不想在删除集群之前拍摄最终备份时，才使用此参数。默认设置是拍摄最终快照。

以下 AWS CLI 代码示例删除 ARN 为 arn: aws: docdb: us-west-2:123456789012: sharded-cluster: sample-cluster 的集群，并进行最终备份。

在以下示例中，用您自己的信息替换每个*`user input placeholder`*示例。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

以下 AWS CLI 代码示例在不进行最终备份的情况下删除 ARN 为 arn: aws: docdb: us-west-2:123456789012: sharded-cluster: sample-cluster 的集群。

在以下示例中，将每个 *user input placeholder* 替换为您自己的信息。

对于 Linux、macOS 或 Unix：

```
aws docdb-elastic delete-cluster \
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-cluster \
    --skip-final-backup \
```

对于 Windows：

```
aws docdb-elastic delete-cluster ^
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-cluster ^
    --skip-final-backup ^
```

`delete-cluster` 操作的输出是您要删除的集群的展示。

删除集群需要几分钟时间。要监控集群的状态，请参阅 监控 Amazon DocumentDB 集群的状态。

管理弹性集群快照

已经创建弹性集群后可以手动拍摄快照。在创建弹性集群快照的那一刻就会创建自动备份。

> **Note**
>
> 您的集群必须处于 `Available` 状态，才能拍摄手动快照。

这个部分解释如何创建、查看、恢复和删除弹性集群快照。

以下主题显示使用 Amazon DocumentDB 弹性集群快照时如何执行各种任务。

主题

- 创建手动弹性集群快照
- 查看弹性集群快照
- 从快照还原弹性集群
- 复制弹性集群快照
- 删除弹性集群快照
- 管理弹性集群快照的自动备份

创建手动弹性集群快照

在本节中，我们将说明如何使用 AWS 管理控制台 或 AWS CLI 以及以下说明创建手动弹性集群快照。

Using the AWS 管理控制台

要使用 AWS 管理控制台创建手动弹性集群快照：

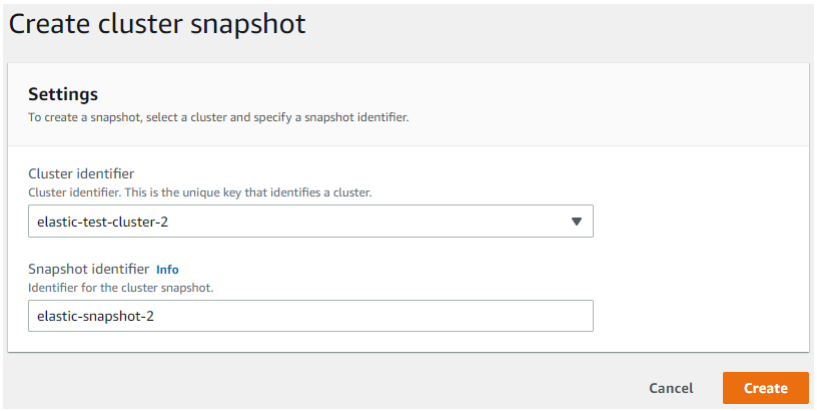
1. 登录 AWS 管理控制台 并打开 Amazon DocumentDB 控制台。
2. 在导航窗格中，选择快照。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在导航窗格左上角选择菜单图标。

3. 在 Snapshots (快照) 页面上，选择 Create (创建)。
4. 在创建集群快照页面上集群标识符字段中，从下拉列表选择您的弹性集群。

   在快照标识符字段中，输入您弹性集群的唯一标识符。

   选择创建。



> **Note**
>
> 或者，您可以通过以下方式访问创建集群快照对话框：转到集群页面，勾选紧邻您集群的复选框，然后选择操作，然后拍摄快照。

您的弹性集群快照现正在预配置。此过程可能需要数分钟完成。当状态作为 `Available` 在快照列表中显示时，您可以查看快照并从其中恢复。

Using the AWS CLI

要使用创建手动弹性集群快照 AWS CLI，请使用带有以下参数的`create-cluster-snapshot`操作：

- **`--snapshot-name`**：必填项。用来创建新集群的快照的名称。
- **`--cluster-arn`**：必填项。您想要创建其快照的集群的 ARN 标识符。

在以下示例中，将每个 *`user input placeholder`* 替换为您自己的信息。

对于 Linux、macOS 或 Unix：

```
aws docdb-elastic create-cluster-snapshot \
    --snapshot-name sample-snapshot-1 \
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster
```

对于 Windows：

```
aws docdb-elastic create-cluster-snapshot ^
```

```
    --snapshot-name sample-snapshot-1 ^
    --cluster-arn arn:aws:docdb:us-west-2:123456789012:sharded-cluster:sample-
cluster
```

查看弹性集群快照

在本节中，我们将说明如何使用 AWS 管理控制台 或 AWS CLI 以及以下说明查看弹性集群快照信息。

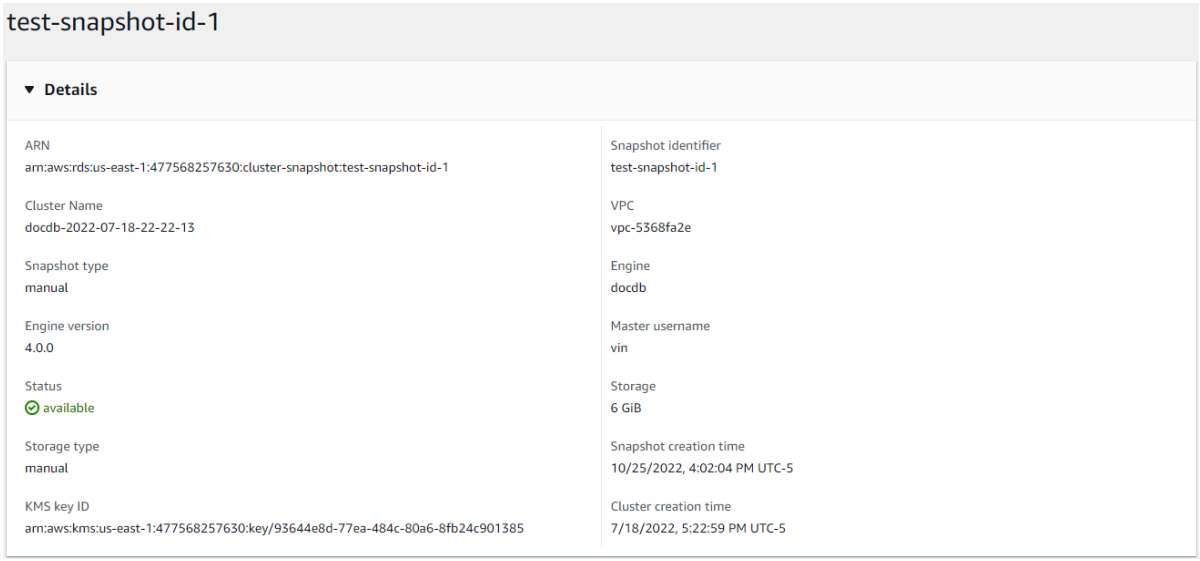
Using the AWS 管理控制台

要查看有关特定弹性集群快照的信息，请使用 AWS 管理控制台：

1. 登录 AWS 管理控制台 并打开 Amazon DocumentDB 控制台。
2. 在导航窗格中，选择快照。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在导航窗格左上角选择菜单图标。

3. 在快照页面上，通过点击快照标识符列中的名称，从列表选择您的快照。
4. 在详情中查看您的快照信息。



Using the AWS CLI

要使用查看有关特定弹性集群快照的信息 AWS CLI，请使用带有以下参数的`get-cluster-snapshot`操作：

- **--snapshot-arn**：必填项。您要获取其信息的快照的 ARN 标识符。

在以下示例中，将每个 *user input placeholder* 替换为您自己的信息。

对于 Linux、macOS 或 Unix：

```
aws docdb-elastic get-cluster-snapshot \
    --snapshot-arn sampleResourceName
```

对于 Windows：

```
aws docdb-elastic get-cluster-snapshot ^
    --snapshot-arn sampleResourceName
```

要使用查看有关特定弹性集群快照的信息 AWS CLI，请使用带有以下参数的`get-cluster-snapshot`操作：

- **--snapshot-arn**：必填项。您要获取其信息的快照的 ARN 标识符。

在以下示例中，将每个 *user input placeholder* 替换为您自己的信息。

对于 Linux、macOS 或 Unix：

```
aws docdb-elastic get-cluster-snapshot \
    --snapshot-arn sampleResourceName
```

对于 Windows：

```
aws docdb-elastic get-cluster-snapshot ^
    --snapshot-arn sampleResourceName
```

要使用查看有关所有弹性集群快照的信息 AWS CLI，请使用带有以下参数的`list-cluster-snapshots`操作：

- **--snapshot-type**—可选。要返回的数据库集群快照的类型。可以指定以下值之一：
  - `automated`-返回 Amazon DocumentDB 自动为您的 AWS 账户创建的所有集群快照。

- manual-返回您为 AWS 账户手动创建的所有集群快照。
- shared-返回已共享到您的 AWS 账户的所有手动集群快照。
- public：返回已标记为公有的所有集群快照。

- **--next-token**：可选。由之前的请求提供的可选分页标记。如果指定此参数，则响应仅包含令牌之外的记录，最大数量为 max-results 指定的值。
- **--max-results**：可选。包括在响应中的最大记录数。如果存在的记录数超过了指定的max-results 值，则在响应中包含分页记号（next-token），以便检索剩余的结果。
  - 默认值：100
  - 最小值 20，最大值 100

在以下示例中，将每个 *user input placeholder* 替换为您自己的信息。

对于 Linux、macOS 或 Unix：

```
aws docdb-elastic list-cluster-snapshots \
    --snapshot-type  value \
    --next-token  value \
    --max-results 50
```

对于 Windows：

```
aws docdb-elastic list-cluster-snapshots ^
    --snapshot-type  value ^
    --next-token  value ^
    --max-results 50
```

从快照还原弹性集群

在本节中，我们将按照以下说明说明如何使用 AWS 管理控制台 或 AWS CLI 从快照恢复弹性集群。

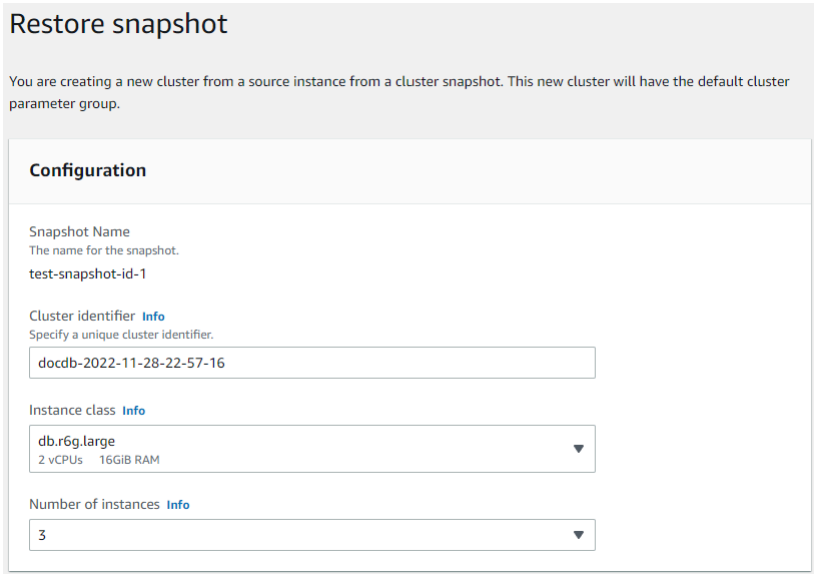
Using the AWS 管理控制台

使用 AWS 管理控制台从快照还原弹性集群：

1. 登录 AWS 管理控制台 并打开 Amazon DocumentDB 控制台。
2. 在导航窗格中，选择快照。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在导航窗格左上角选择菜单图标。

3. 在快照标识符 中，选择要用于还原集群的快照左侧的按钮。
4. 选择操作，然后还原。



5. 在还原快照页面上，在集群标识符字段中输入新集群的名称。

> **Note**
>
> 对于任何手动快照恢复，您必须创建一个新集群。

6. 在虚拟私有云 (VPC)字段中，从下拉列表中选择一个 VPC。
7. 对于子网和 VPC 安全组，您可以使用默认值或选择您选定的三个子网和多达三个 VPC 安全组（最少一个）。
8. 如果您对集群配置满意，请选择 Restore cluster (还原集群) 并等待集群还原。

Using the AWS CLI

要使用从快照恢复弹性集群 AWS CLI，请使用带有以下参数的`restore-cluster-from-snapshot`操作：

- **`--cluster-name`**：必填项。创建期间输入或上次修改的弹性集群的当前名称。

- **--snapshot-arn**：必填项。正用于恢复集群的快照的 ARN 标识符。
- **--vpc-security-group-ids**：可选。与该集群待关联的一个或多个 Amazon EC2 安全组和 Amazon VPC (VPC) 安全组。
- **--kms-key-id**：可选。配置已加密集群的 KMS 密钥标识符。

  KMS 密钥标识符是 AWS KMS 加密密钥的亚马逊资源名称 (ARN)。如果使用拥有用于加密新集群的 KMS 加密密钥的同一 Amazon Web Services 账户创建集群，则可以使用 KMS 密钥别名而不是 KMS 加密密钥的 ARN。

  如果中未指定加密密钥， KmsKeyId 且StorageEncrypted参数为真，则 Amazon DocumentDB 将使用您的默认加密密钥。
- **--subnet-ids**—可选。网络子网 ID。

在以下示例中，用您自己的信息替换每个*user input placeholder*示例。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

复制弹性集群快照

在 Amazon DocumentDB 中，您可以在同一区域和同一账户中复制手动和自动弹性集群快照。在本节中，我们将说明如何使用 AWS 管理控制台 或复制弹性集群快照 AWS CLI。

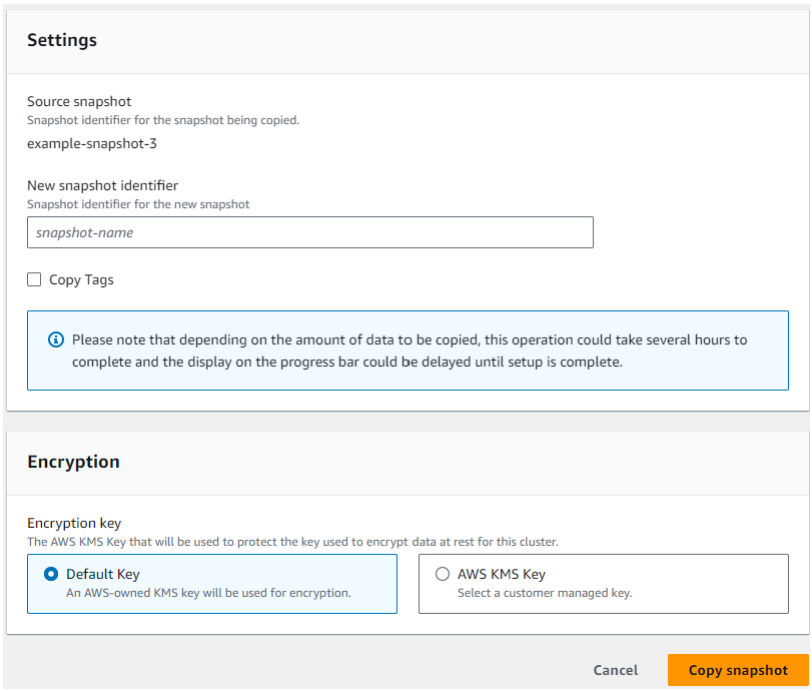
Using the AWS 管理控制台

要使用以下方法复制弹性集群快照 AWS 管理控制台：

1. 登录 AWS 管理控制台 并打开 Amazon DocumentDB 控制台。
2. 在导航窗格中，选择快照。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在导航窗格左上角选择菜单图标。

3. 在快照标识符中，选择要复制的快照左侧的按钮。
4. 选择操作和复制。



5. 对于新快照标识符，输入新快照的名称。
6. 对于复制标签，如果要将源弹性集群快照的所有标签复制到目标弹性集群快照，请勾选此方框。
7. 对于加密，请选择默认 AWS KMS 密钥或 KMS 密钥。在第二个选项中，您可以选择已创建的现有 KMS 密钥，也可以创建新的密钥。
8. 完成后选择复制快照。

Using the AWS CLI

要使用复制弹性集群快照 AWS CLI，请使用带有以下参数的`copy-cluster-snapshot`操作：

- **`--source-db-cluster-snapshot-identifier`**：必填项。当前复制的现有弹性集群快照的标识符。弹性集群快照必须存在并且处于可用状态。此参数不区分大小写。
- **`--target-db-cluster-snapshot-identifier`**：必填项。要从现有集群快照创建的新弹性集群快照标识符。此参数不区分大小写。

  目标快照名称约束：

  - 不能是现有快照的名称。
  - 长度为 [1-63] 个字母、数字或连字符。
  - 第一个字符必须是字母。
  - 不能以连字符结尾或包含两个连续的连字符。

在以下示例中，将每个 *`user input placeholder`* 替换为您自己的信息。

对于 Linux、macOS 或 Unix：

```
aws docdb-elastic copy-cluster-snapshot \
     --source-cluster-snapshot-arn <sample ARN> \
     --target-cluster-snapshot-name my-target-copied-snapshot
```

对于 Windows：

```
aws docdb-elastic copy-cluster-snapshot ^
     --source-cluster-snapshot-arn <sample ARN> ^
     --target-cluster-snapshot-name my-target-copied-snapshot
```

删除弹性集群快照

在本节中，我们将说明如何使用 AWS 管理控制台 或删除弹性集群快照 AWS CLI。

Using the AWS 管理控制台

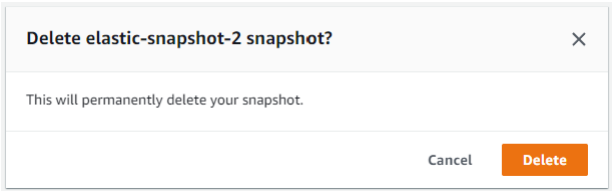
使用 AWS 管理控制台从快照还原弹性集群：

1. 登录 AWS 管理控制台 并打开 Amazon DocumentDB 控制台。

2. 在导航窗格中，选择快照。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在导航窗格左上角选择菜单图标。

3. 在快照标识符 中，选择要用于还原集群的快照左侧的按钮。
4. 选择操作，然后选择删除。



5. 在删除“快照名称”快照对话框中，选择删除。

Using the AWS CLI

要使用删除弹性集群快照 AWS CLI，请使用带有以下参数的`delete-cluster-snapshot`操作：

- **`--snapshot-arn`**：必填项。正用于恢复集群的快照的 ARN 标识符。

在以下示例中，用您自己的信息替换每个*`user input placeholder`*示例。

对于 Linux、macOS 或 Unix：

```
aws docdb-elastic delete-cluster-snapshot \
     --snapshot-arn sampleResourceName
```

对于 Windows：

```
aws docdb-elastic delete-cluster-snapshot ^
     --snapshot-arn sampleResourceName
```

管理弹性集群快照的自动备份

Amazon DocumentDB 每天都会对弹性集群拍摄快照。您可以在新的或现有弹性集群快照配置中，指定首选备份时段和备份保留期。在本节中，我们将说明如何使用 AWS 管理控制台 或在弹性集群快照中设置自动备份参数 AWS CLI。

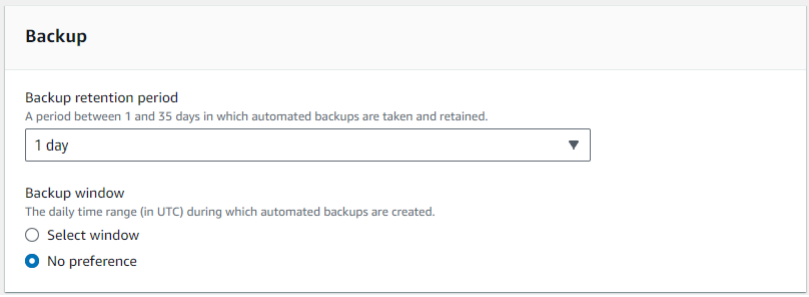
Using the AWS 管理控制台

要为新的弹性集群快照设置自动备份，请使用 AWS 管理控制台：

1. 登录 AWS 管理控制台 并打开 Amazon DocumentDB 控制台。
2. 在导航窗格中，选择集群。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在导航窗格左上角选择菜单图标。

3. 在集群标识符中，选择要更改其备份设置的集群左侧的按钮。
4. 依次选择操作和修改。
5. 在备份部分，根据您的备份要求编辑字段。



   a. 备份留存期：在列表中，选择在删除此集群的自动备份前保留它们的天数。
   b. 备份时段：设置 Amazon DocumentDB 要备份此集群的每日时间和持续时间。
      i. 如果要配置创建备份的时间和时长，请选择选择时段。

         开始时间：在第一个列表中，选择开始自动备份的开始时间小时 (UTC)。在第二个列表中，选择您希望自动备份开始的时间（分钟）。

         持续时间：在该列表中，选择要向创建自动备份分配的小时数。
      ii. 如果想要 Amazon DocumentDB 选择创建备份的时间和时长，请选择无首选项。
6. 完成后选择修改集群。

Using the AWS CLI

要使用为新的弹性集群快照设置自动备份 AWS CLI，请使用带有以下参数的`create-cluster-snapshot`操作：

- **--preferred-backup-window**：可选。创建自动备份的每日首选时间范围。默认值是从 8 小时的时间段中随机选择一个 30 分钟的窗口。 AWS 区域

  约束：

  - 必须采用 hh24:mi-hh24:mi 格式。
  - 必须采用通用协调时间（UTC）。
  - 不得与首选维护时段冲突。
  - 必须至少为 30 分钟。
- **--backup-retention-period**：可选。自动备份的保留天数。默认值是 1。

  约束：

  - 必须指定最小值 1。
  - 范围从 1 到 35。

> **Note**
>
> 只有当集群处于“活动”状态时才进行自动备份。

> **Note**
>
> 您也可以使用 `aws docdb-elastic update-cluster` 命令修改现有弹性集群的 `preferred-backup-window` 和 `backup-retention-period` 参数。

在以下示例中，将每个 *user input placeholder* 替换为您自己的信息。

以下`create-cluster`示例创建了 Amazon DocumentDB 弹性集群*sample-cluster*，其自动备份的保留期为*7*几天，首选备份窗口为。*18:00-18:30 UTC*

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

停止和启动 Amazon DocumentDB 弹性集群

停止和启动 Amazon DocumentDB 弹性集群可以帮助您控制开发和测试环境的成本。当不需要时，您可以暂时停止弹性集群，而不是在每次使用 Amazon DocumentDB 时创建和删除集群。当您恢复测试时，可以再次启动。

主题

- 停止和启动弹性集群概述
- 可以在已停止的弹性集群上执行的操作

停止和启动弹性集群概述

在不需要使用 Amazon DocumentDB 弹性集群期间，您可以停止该集群。然后，您可以在需要使用时再次启动集群。启动和停止简化了用于开发、测试或不需要持续可用性的类似活动的弹性集群的设置和停用过程。您可以使用 AWS 管理控制台 或通过单个操作停止和启动弹性集群。 AWS CLI

当您的弹性集群停止后，集群存储卷保持不变。您只需在指定的保留时段内为集群存储、手动快照和自动备份存储付费。Amazon DocumentDB 会在七天后自动重新启动您的弹性集群，这样一来您的集群就可以及时获得任何所需的维护更新。当您的集群在七天后启动时，您将再次开始为使用弹性集群付费。当您的集群停止时，您无法查询您的存储卷，因为查询需要集群处于可用状态。

当 Amazon DocumentDB 弹性集群停止后，不能以任何方式修改该集群。这包括删除集群。

Using the AWS 管理控制台

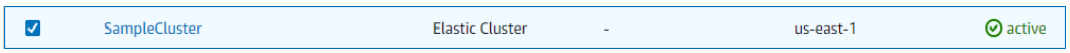
以下过程显示如何停止处于可用状态的弹性集群或启动已停止的弹性集群。

停止或启动 Amazon DocumentDB 弹性集群

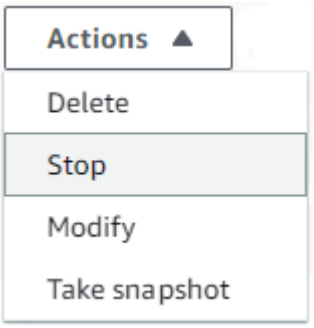
1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

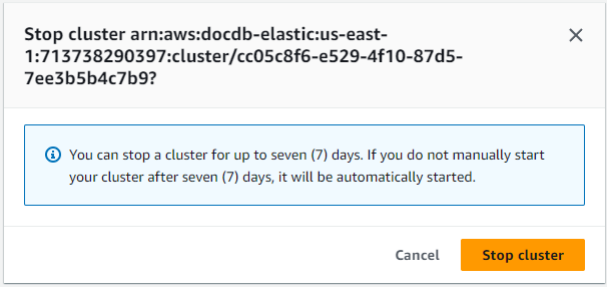
3. 在集群列表中，选择要停止或启动的集群名称左侧的按钮。



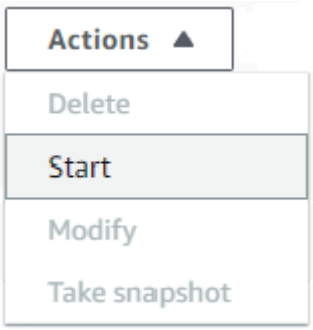
4. 选择操作，然后选择您要在集群上执行的操作。
   - 如果您要停止集群且集群可用，则：
     a. 选择停止。



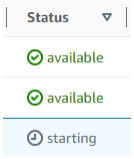
     b. 在确认对话框中，确认您要通过选择停止集群来停止该弹性集群；或者要保持集群运行，则选择取消。



- 如果您要启动集群且集群处于停止状态，则选择启动。



5. 监控弹性集群的状态。如果您启动集群，则当集群可用时，您可以继续使用集群。有关更多信息，请参阅 确定集群的状态。



Using the AWS CLI

以下代码示例显示如何停止处于活动或可用状态的弹性集群或启动已停止的弹性集群。

要使用停止弹性集群 AWS CLI，请使用`stop-cluster`操作。要启动已停止的集群，请使用 `start-cluster` 操作。这两个操作都使用 `--cluster-arn` 参数。

参数：

- **`--cluster-arn`**：必填项。要停止或启动的弹性集群的 ARN 标识符。

Example— 要停止弹性集群，请使用 AWS CLI

在以下示例中，将每个 *`user input placeholder`* 替换为您自己的信息。

以下代码可停止 ARN 为 `arn:aws:docdb-elastic:us-east-1:477568257630:cluster/b9f1d489-6c3e-4764-bb42-da62ceb7bda2` 的弹性集群。

> **Note**
>
> 弹性集群必须为活动或可用状态。

对于 Linux、macOS 或 Unix：

```
aws docdb-elastic stop-cluster \
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

对于 Windows：

```
aws docdb-elastic stop-cluster ^
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

Example— 要启动弹性集群，请使用 AWS CLI

在以下示例中，将每个 *user input placeholder* 替换为您自己的信息。

以下代码可启动 ARN 为 arn:aws:docdb-elastic:us-east-1:477568257630:cluster/b9f1d489-6c3e-4764-bb42-da62ceb7bda2 的弹性集群。

> **Note**
>
> 弹性集群必须处于停止状态。

对于 Linux、macOS 或 Unix：

```
aws docdb-elastic start-cluster \
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

对于 Windows：

```
aws docdb-elastic start-cluster ^
   --cluster-arn arn:aws:docdb-elastic:us-east-1:477568257630:cluster/
b9f1d489-6c3e-4764-bb42-da62ceb7bda2
```

可以在已停止的弹性集群上执行的操作

在 Amazon DocumentDB 弹性集群停止后，您无法修改该集群的配置。您必须在执行任何此类管理操作之前启动该集群。

仅当再次启动后， Amazon DocumentDB 才会将任何计划的维护应用于停止的弹性集群。七天后，Amazon DocumentDB 自动启动停止的弹性集群，以使其维护状态不会落后太多。当弹性集群重新启动后，您将再次开始为集群中的分片付费。

弹性集群停止后，Amazon DocumentDB 不会执行任何自动备份，也不会延长备份留存期。

维护 Amazon DocumentDB 弹性集群

主题

- 查看待处理的弹性集群维护操作
- 弹性集群引擎更新
- 弹性集群操作系统更新

Amazon DocumentDB 会定期对 Amazon DocumentDB 弹性集群资源执行维护。维护最常涉及对数据库引擎（弹性集群维护）或弹性集群的底层操作系统的更新（操作系统更新）。数据库引擎更新是必需的补丁，包括安全补丁、错误修复以及数据库引擎增强功能。虽然大多数操作系统补丁都是可选的，但如果在一段时间内未应用这些补丁，就可能变成必需补丁并自动应用，以保持您的安全状况。因此，我们建议在操作系统更新可用时立即将其应用于 Amazon DocumentDB 弹性集群。

数据库引擎补丁需要使 Amazon DocumentDB 弹性集群脱机一小段时间。一旦这些补丁可用后，系统会自动安排在即将到来的 Amazon DocumentDB 弹性集群计划维护时段内应用补丁。

弹性集群有各自的维护时段。您选择不立即应用的弹性集群修改会在维护时段内应用。默认情况下，在创建弹性集群时，Amazon DocumentDB 会为弹性集群分配维护时段。您可以在创建弹性集群时选择维护时段。也可以随时修改维护时段以适应您的业务计划或实践。通常，建议选择尽量减少对应用程序的影响的维护时段（例如，晚上或周末）。

查看待处理的弹性集群维护操作

您可以使用 AWS CLI来查看维护更新是否对弹性集群可用。

如果有可用更新，您可以执行以下操作之一：

- 推迟目前计划在下一个维护窗口内执行的维护操作（仅适用于操作系统补丁）。
- 立即应用维护操作。
- 计划下一个维护时段内要开始的维护操作。
- 计划所选应用时段内要开始的维护操作。

维护时段确定待处理的操作何时开始，但不限制这些操作的总执行时间。

使用以下 AWS CLI 操作来确定哪些维护操作处于待处理状态。列出所有待处理的维护操作：

```
aws docdb-elastic list-pending-maintenance-actions
```

此操作的输出将类似如下（JSON 格式）：

```
{
'ResourcePendingMaintenanceActions': [
    {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'ENGINE_UPDATE',
                'AutoAppliedAfterDate': 'string',
                'ForcedApplyDate': 'string',
                'OptInStatus': 'string',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            },
        ]
    },
],
'NextToken': 'string'
}
```

获取给定 `resourceArn` 的待处理维护操作（如果有）：

```
aws docdb-elastic get-pending-maintenance-action --resource-arn string-arn
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    'ResourcePendingMaintenanceAction': {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'ENGINE_UPDATE',
                'AutoAppliedAfterDate': 'string',
                'ForcedApplyDate': 'string',
```

```
                'OptInStatus': 'string',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            }
        ]
    }
}
```

参数：

- ResourceArn：待处理的维护操作应用于的资源的 Amazon DocumentDB Amazon 资源名称（ARN）。
- Action：将应用于资源的待处理维护操作。

  有效值：

  - ENGINE_UPDATE
  - ENGINE_UPGRADE
  - SECURITY_UPDATE
  - OS_UPDATE
  - MASTER_USER_PASSWORD_UPDATE
- AutoAppliedAfterDate：此日期之后的第一个维护时段。在此情况下将忽略 NEXT_MAINTENANCE OPT_IN。
- ForcedApplyDate：无论维护时段如何都应用。在此情况下将忽略 IMMEDIATE OPT_IN。
- OptInStatus：用于指定加入请求类型或撤消加入请求的值。不能撤消 IMMEDIATE 类型的加入请求。

  有效值：

  - IMMEDIATE：立即应用维护操作。
  - NEXT_MAINTENANCE：在资源的下一个维护时段内应用维护操作。
  - APPLY_ON：在指定的应用日期应用维护操作，而无论资源的下一个维护时段如何。
  - UNDO_OPT_IN：取消任何现有的 NEXT_MAINTENANCE 或 APPLY_ON 选择加入请求。
- CurrentApplyDate：如果 opt-in-type 为 APPLY_ON，则显示此项。
- Description：维护操作的选项描述。

弹性集群引擎更新

通过 Amazon DocumentDB，您可以选择何时应用维护操作。使用 AWS CLI时，您可以决定 Amazon DocumentDB 何时应用更新。

应用待处理的维护操作：

```
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]
```

参数：

- **--resource-arn**：待处理的维护操作应用于的资源的 Amazon DocumentDB Amazon 资源名称（ARN）。
- **--apply-action**：应用于此资源的待处理的维护操作。

  有效值：

  - ENGINE_UPDATE
  - ENGINE_UPGRADE
  - SECURITY_UPDATE
  - OS_UPDATE
  - MASTER_USER_PASSWORD_UPDATE
- **--opt-in-type**：用于指定加入请求类型或撤消加入请求的值。不能撤消 IMMEDIATE 类型的加入请求。

  有效值：

  - IMMEDIATE：立即应用维护操作。
  - NEXT_MAINTENANCE：在资源的下一个维护时段内应用维护操作。
  - APPLY_ON：在指定的应用日期应用维护操作，而无论资源的下一个维护时段如何。
  - UNDO_OPT_IN：取消任何现有的 NEXT_MAINTENANCE 或 APPLY_ON 选择加入请求。
- **[--apply-on]**：如果 opt-in-type 为 APPLY_ON，则为必需项。格式：yyyy/MM/dd HH:mm-yyyy/MM/dd HH:mm（此选项使用 UTC 时间。 开始时间可以是将来的任何时间，最少 30 分钟，最多 14 天，也可以是待处理操作的 force/apply 日期，以较早者为准。 开始到结束的时间范围可以是最少 30 分钟，最长可以是 8 小时。 ）

此操作的输出将类似如下（JSON 格式）：

```
{
 'ResourcePendingMaintenanceAction': {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'SECURITY_UPDATE',
                'AutoAppliedAfterDate': 'string',
                'ForcedApplyDate': 'string',
                'OptInStatus': 'IMMEDIATE',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            },
        ]
 }
}
```

参数：

- ResourceArn：待处理的维护操作应用于的资源的 Amazon DocumentDB Amazon 资源名称（ARN）。
- Action：将应用于资源的待处理维护操作。

  有效值：

  - ENGINE_UPDATE
  - ENGINE_UPGRADE
  - SECURITY_UPDATE
  - OS_UPDATE
  - MASTER_USER_PASSWORD_UPDATE
- AutoAppliedAfterDate：此日期之后的第一个维护时段。在此情况下将忽略 NEXT_MAINTENANCE OPT_IN。
- ForcedApplyDate：无论维护时段如何都应用。在此情况下将忽略 IMMEDIATE OPT_IN。
- OptInStatus：用于指定加入请求类型或撤消加入请求的值。不能撤消 IMMEDIATE 类型的加入请求。

  有效值：

  - IMMEDIATE：立即应用维护操作。

  - NEXT_MAINTENANCE：在资源的下一个维护时段内应用维护操作。
  - APPLY_ON：在指定的应用日期应用维护操作，而无论资源的下一个维护时段如何。
  - UNDO_OPT_IN：取消任何现有的 NEXT_MAINTENANCE 或 APPLY_ON 选择加入请求。
- CurrentApplyDate：如果 opt-in-type 为 APPLY_ON，则显示此项。
- Description：维护操作的选项描述。

应用日期

每个维护操作都有一个相应的应用日期，您可以在描述待处理的维护操作时找到它们。当您阅读中待处理的维护操作的输出时 AWS CLI，会列出三个日期：

- CurrentApplyDate：将立即应用或在下一个维护时段期间应用维护操作的日期。如果维护是可选的，则该值可以为 null。
- ForcedApplyDate：自动应用维护的日期，与维护时段无关。
- AutoAppliedAfterDate：将在该日期后的集群维护时段期间应用维护。

User-created 维护操作

作为 Amazon DocumentDBelastic 集群用户，您可以启动对集群配置的更新。

更新集群主密码

```
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
```

参数：

- **--cluster-arn**：维护操作将应用于的资源的 Amazon DocumentDB Amazon 资源名称（ARN）。
- **[--admin-user-password]**：与管理员用户关联的密码。

- **[--auth-type]**：身份验证类型，用于确定从何处获取用于访问弹性集群的密码。有效类型为 PLAIN_TEXT 和 SECRET_ARN。
- **[--apply-method]**：一个值，指定所应用方法的类型。支持的值包括 IMMEDIATE 和 APPLY_ON。默认值为 IMMEDIATE。
- **[--apply-on]**：如果 apply-method 为 APPLY_ON，则为必需项。格式：yyyy/MM/dd HH:mm-yyyy/MM/dd HH:mm（此选项使用 UTC 时间。开始时间可以是未来的任何时间，最小值为 30 分钟，最大值为 14 天。开始到结束的时段可以最少为 30 分钟，最长为 8 小时。）

此操作的输出将类似如下（JSON 格式）：

```
{
 'ResourcePendingMaintenanceAction': {
        'ResourceArn': 'string-arn',
        'PendingMaintenanceActionDetails': [
            {
                'Action': 'MASTER_USER_PASSWORD_UPDATE',
                'OptInStatus': 'APPLY_ON',
                'CurrentApplyDate': 'string',
                'Description': 'string'
            },
        ]
 }
}
```

更改您的 Amazon DocumentDB 维护窗口

维护时段应当选在使用量最小的时段上，因而可能必须不时予以更改。您的弹性集群只会在应用系统更改（例如，扩展存储操作更改）并且需要中断的期间出现不可用现象，且持续时间只是这些必要更改所需的最少时间。

默认值为每个 Amazon Web Services 区域 8 小时的时间段中随机选择的 30 分钟时段（随机选取周中的某天进行）。

要更改维护时段，请参阅 修改弹性集群配置。

弹性集群操作系统更新

Amazon DocumentDB 弹性集群偶尔需要操作系统更新。Amazon DocumentDB 将操作系统升级到更新的版本，以提高数据库性能和客户的整体安保状况。操作系统更新不会更改 Amazon DocumentDB 弹性集群的集群引擎版本。

Amazon DocumentDB 弹性集群的大多数操作系统更新都是可选的，没有固定的应用日期。但是，如果在一段时间内未应用这些更新，最终可能会变成必需的更新，并在集群维护时段内自动应用。这是为了帮助维持数据库的安全状况。为避免任何意外停机，我们建议在操作系统更新可用时，尽快将其应用于 Amazon DocumentDB 弹性集群，并根据业务需求在方便的时间设置集群维护时段。

用于 Amazon DocumentDB 弹性集群的静态数据加密

以下主题可帮助您了解、创建和监控亚马逊 DocumentDB 弹性集群的 AWS Key Management Service 加密密钥：

主题

- Amazon DocumentDB 弹性集群如何在中使用授权 AWS KMS
- 创建客户托管密钥
- 监控您的 Amazon DocumentDB 弹性集群加密密钥
- 了解详情

Amazon DocumentDB 弹性集群会自动与 AWS Key Management Service (AWS KMS) 集成以进行密钥管理，并使用一种称为信封加密的方法来保护您的数据。有关信封加密的更多信息，请参阅 AWS Key Management Service 开发人员指南中的信封加密。

A AWS KMS key 是密钥的逻辑表示形式。KMS 密钥包含元数据，如密钥 ID、创建日期、描述和密钥状态。KMS 密钥还包含用于加密和解密数据的密钥材料。有关 KMS 密钥的更多信息，请参阅 AWS Key Management Service 开发人员指南中的 AWS KMS keys。

Amazon DocumentDB 弹性集群支持采用两种类型的密钥加密：

- AWS 自有密钥 — Amazon DocumentDB 弹性集群默认使用这些密钥来自动加密个人身份数据。您无法查看、管理或使用 AWS自有密钥，也无法审核其使用情况。但是，无需采取任何措施或更改任何计划即可保护用于加密数据的密钥。有关更多信息，请参阅 AWS Key Management Service 开发人员指南中的 AWS 自有密钥。
- Customer-managed 密钥 — 由您创建、拥有和管理 AWS KMS keys 的 Symmetric。由于您可以完全控制这层加密，因此可以执行以下任务：
  - 制定和维护关键策略
  - 建立和维护 IAM 策略和授权
  - 启用和禁用密钥策略

- 轮换加密材料
- 添加 标签
- 创建密钥别名
- 安排密钥删除

有关更多信息，请参阅 AWS Key Management Service 开发人员指南中的客户托管密钥。

> **Important**
>
> 您必须使用对称加密 KMS 密钥加密您的集群，因为 Amazon DocumentDB 仅支持对称加密 KMS 密钥。请勿使用非对称 KMS 密钥尝试对 Amazon DocumentDB elastic 集群中的数据进行加密。有关更多信息，请参阅AWS Key Management Service 开发人员指南中的非对称密钥 AWS KMS。
>
> 如果 Amazon DocumentDB 不再能够有权访问集群的加密密钥（例如，在撤销密钥访问权限时），加密的集群将进入终末状态。在此情况下，您只能从备份还原集群。对于 Amazon DocumentDB，备份始终启用 1 天。此外，如果您禁用已加密 Amazon DocumentDB 集群的密钥，您最终将失去对该集群的读写访问权限。如果 Amazon DocumentDB 遇到用它无法访问的密钥加密的集群，则它会使该集群进入最终状态。在此状态下，集群不再可用，并且数据库的当前状态无法恢复。若要还原集群，您必须重新启用对 Amazon DocumentDB 的加密密钥的访问，然后从备份还原集群。

> **Important**
>
> 在已创建加密集群的 KMS 密钥后，您无法更改它。请确保先确定您的加密密钥要求，然后再创建加密的弹性集群。

Amazon DocumentDB 弹性集群如何在中使用授权 AWS KMS

Amazon DocumentDB 弹性集群需要 授权 来使用客户托管密钥。

当您创建使用客户托管密钥加密的集群时，Amazon DocumentDB 弹性集群会通过向发送`CreateGrant`请求来代表您创建授权。 AWS KMS中的授权 AWS KMS 用于向 Amazon DocumentDB 弹性集群提供访问客户账户中的 KMS 密钥的权限。

Amazon DocumentDB 弹性集群要求该授权对以下内部操作使用您的客户托管密钥：

- 向发送DescribeKey请求， AWS KMS 以验证在创建跟踪器或地理围栏集合时输入的对称客户管理的 KMS 密钥 ID 是否有效。
- 向发送GenerateDataKey请求 AWS KMS 以生成由您的客户托管密钥加密的数据密钥。
- 向发送解密加密数据密钥的Decrypt请求，以便这些密钥可用于加密您的数据。 AWS KMS
- 您可以随时撤销授予访问权限，或删除服务对客户托管密钥的访问权限。如果您这样做，Amazon DocumentDB 弹性集群将不能访问由客户托管密钥加密的任何数据，这影响依赖于该数据的操作。

创建客户托管密钥

您可以使用 AWS 管理控制台 或 AWS KMS API 创建对称的客户托管密钥。

对称客户托管密钥创建

遵循 AWS Key Management Service 开发人员指南中创建对称性客户托管密钥的步骤。

密钥策略

密钥策略控制对客户自主管理型密钥的访问。每个客户托管式密钥必须只有一个密钥策略，其中包含确定谁可以使用密钥以及如何使用密钥的声明。创建客户托管式密钥时，可以指定密钥策略。有关更多信息，请参阅位于AWS Key Management Service 开发人员指南AWS Key Management Service 的概述中的 KMS 密钥访问信息。

要将您的客户托管密钥配合 Amazon DocumentDB 弹性集群资源一起使用，则必须在密钥策略中允许以下 API 操作：

- kms:CreateGrant：向客户托管密钥添加授权。授予对指定 KMS 密钥的控制访问权限，这允许 Amazon Location Service 要求的授权操作。有关授权的更多信息，请参阅 AWS Key Management Service 开发人员指南中的授权 AWS KMS。
- kms:DescribeKey：提供客户托管式密钥详细信息以允许 Docdb Elastic 验证密钥。
- kms:Decrypt：允许 Docdb Elastic 使用存储的已加密数据密钥访问已加密数据。
- kms:GenerateDataKey：允许 Docdb Elastic 生成并存储已加密的数据密钥，因为数据密钥并不立即用于加密。

有关更多信息，请参阅AWS Key Management Service 开发人员指南中的密钥策略中的 AWS 服务权限和密钥访问疑难解答。

通过 IAM 策略限制客户访问托管密钥

除了 KMS 密钥策略外，您还可以在 IAM policy 略中限制 KMS 密钥权限。

您可以通过各种方式使 IAM 策略更严格。例如，要允许客户管理密钥仅用于源自 Amazon DocumentDB 弹性集群的请求，您可以将 kms:ViaService 条件键 与 docdb-elastic.<region-name>.amazonaws.com 值结合使用。

有关更多信息，请参阅《AWS Key Management Service 开发人员指南》中的 Allowing users in other accounts to use a KMS key。

监控您的 Amazon DocumentDB 弹性集群加密密钥

当您将 AWS KMS key 客户托管密钥与 Docdb Elastic 资源一起使用时，您可以使用 AWS CloudTrail 或 Amazon CloudWatch Logs 来跟踪 Docdb Elastic 发送到的请求。 AWS KMS

以下示例是CreateGrant、GenerateDataKeyWithoutPlainTextDecrypt、和DescribeKey监控 Amazon DocumentDB 弹性集群为访问由您的客户托管密钥加密的数据而调用的 AWS KMS key 操作 AWS CloudTrail 的事件：

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"
}
```


了解详情

以下资源提供有关静态数据加密的更多信息：

- 有关 AWS KMS 概念的更多信息，请参阅《AWS Key Management Service 开发者指南》中的AWS Key Management Service 基本概念。

- 有关 AWS KMS 安全的更多信息，请参阅《AWS Key Management Service 开发人员指南》 AWS Key Management Service中的安全最佳实践。

Service-linked 弹性集群中的角色

亚马逊 DocumentDB 弹性集群使用 AWS Identity and Access Management (IAM) 服务相关角色。服务相关角色是一种独特的 IAM 角色，直接链接到 Amazon DocumentDB 弹性集群。 Service-linked 角色由 Amazon DocumentDB 弹性集群预定义，包括该服务代表您调用其他 AWS 服务所需的所有权限。

服务关联角色使得可以更轻松地设置 Amazon DocumentDB 弹性集群，因为您不必手动添加必要的权限。Amazon DocumentDB 弹性集群定义了其服务关联角色的权限，除另有定义外，只有 Amazon DocumentDB 弹性集群可以担当该角色。定义的权限包括信任策略和权限策略，而且权限策略不能附加到任何其他 IAM 实体。只有在首先删除角色的相关资源后，才能删除角色。这样可以保护您的 Amazon DocumentDB 弹性集群资源，因为您不会无意中移除访问资源所需的权限。

有关支持服务相关角色的其他服务的信息，请参阅与 IAM 配合使用的AWS 服务，并在 “Service-Linked 角色” 列中查找标有 “是” 的服务。请选择是与查看该服务的服务关联角色文档的链接。

Service-linked 弹性集群的角色权限

Amazon DocumentDB 弹性集群使用名为的服务相关角色AWS ServiceRoleForDocDB-Elastic来允许 Amazon DocumentDB 弹性集群代表您的集群调用 AWS 服务。

此服务相关角色附加了一个名为 AmazonDocDB-ElasticServiceRolePolicy 的权限策略，授予其在您的账户中操作的权限。角色权限策略允许 Amazon DocumentDB 弹性集群对指定资源完成以下操作：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*",
```

```
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": [
                        "AWS/DocDB-Elastic"
                    ]
                }
            }
        }
    ]
}
```

> **Note**
>
> 您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务关联角色。如果您遇到以下错误消息：“不能创建资源。Verify that you have permission to create service linked role。否则，请等待并稍后再试。”确保您已启用以下权限：

```
{
"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"
        }
    }
}
```

有关更多信息，请参阅《Identity and A ccess Managem AWS ent 用户指南》中的Service-linked 角色权限。

创建 Amazon DocumentDB 弹性集群的服务关联角色

您无需手动创建服务关联角色。创建数据库实例时，Amazon DocumentDB 弹性集群将为您创建一个服务关联角色。

编辑 Amazon DocumentDB 弹性集群的服务关联角色

Amazon DocumentDB 弹性集群不允许您编辑 AWS ServiceRoleForDocDB-Elastic 服务关联角色。创建服务关联角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅 AWS 身份和访问用户指南中的编辑服务关联角色。

删除 Amazon DocumentDB 弹性集群的服务关联角色

如果您不再需要使用某个需要服务相关角色的功能或服务，我们建议您删除该角色。这样您就没有未被主动监控或维护的未使用实体。但是，您必须先删除所有 集群，然后才能删除服务相关角色。

清除服务相关角色

必须先确认服务相关角色没有活动会话并删除该角色使用的任何资源，然后才能使用 IAM 删除服务相关角色。

在 IAM 控制台中检查服务相关角色是否具有活动会话

1. 登录到 AWS 管理控制台 并打开 IAM 控制台。
2. 在 IAM 控制台的导航窗格中，选择角色。然后选择 AWS ServiceRoleForDocDB-Elastic 角色的名称（不是复选框）。
3. 在所选角色的 Summary (摘要) 页面上，选择 Access Advisor (访问顾问) 选项卡。

> **Note**
>
> 如果您不确定 Amazon DocumentDB 弹性集群是否在使用 AWS ServiceRoleForDocDB-Elastic 角色，可以尝试删除该角色。如果服务正在使用该角色，则删除将失败，您可以查看该角色的使用 AWS 区域 位置。如果该角色已被使用，则您必须等待会话结束，然后才能删除该角色。您无法撤销服务相关角色对会话的权限。
> 如果您要删除 AWS ServiceRoleForDocDB-Elastic 角色，则必须先删除您的所有集群。

删除所有集群

在 Amazon DocumentDB 控制台删除集群

1. 登录 AWS 管理控制台 并打开 Amazon DocumentDB 控制台。
2. 在导航窗格中，选择集群。
3. 选择要删除的集群。

4. 对于操作，选择删除。
5. 如果系统提示您是否创建最终快照？，请选择是或否。
6. 如果您在上一步中选择了 Yes (是)，请为 Final snapshot name (最终快照名称) 输入最终快照的名称。
7. 选择删除。

> **Note**
>
> 您可以使用 IAM 控制台、IAM CLI 或 IAM API 来删除 `AWS ServiceRoleForDocDB-Elastic` 服务关联角色。有关更多信息，请参阅 AWS 身份和访问用户指南中的删除服务关联角色。

监控 Amazon DocumentDB

监控您的 AWS 服务是保持系统健康和最佳运行的重要组成部分。最好从 AWS 解决方案的各个部分收集监控数据，以便在发生故障或性能下降时更容易调试或修复。在开始监控您的 AWS 解决方案之前，我们建议您思考并回答以下问题：

- 监控目的是什么？
- 您要监控什么资源？
- 您将以什么样的频率监控这些资源？
- 您将使用哪些监控工具？
- 由谁负责执行监控？
- 发生错误时通知谁？以何种方式发送通知？

要了解当前的性能模式，判断性能异常表现，并构想问题解决方法，您应该针对不同时间和不同负载条件设定基准性能指标。当您监控 AWS 解决方案时，我们建议您存储历史监控数据，这些数据既可以供日后参考，也可帮助您设定基准。

通常，性能指标的可接受值取决于您的基准性能以及应用程序执行的操作。应调查相对于基准性能的一致或趋势性变化。有关特定指标类型的建议如下：

- 高 CPU 或 RAM 使用：较高的 CPU 或 RAM 使用值可能是正常情况，只要它们符合您的应用程序的目标（如吞吐量或并发度）并且是预期情况即可。
- 存储卷消耗：如果使用的空间始终不低于总存储卷空间的 85%，调查存储消耗量 (`VolumeBytesUsed`)。确定是可以从存储卷中删除数据还是可以将数据存档到其他系统以释放空间。有关更多信息，请参阅Amazon DocumentDB 存储和Amazon DocumentDB 配额和限制。
- 网络流量：对于网络流量，请与系统管理员讨论，以了解域网络和互联网连接的预期吞吐量。如果吞吐量始终低于预期，则应调查网络流量。
- 数据库连接：如果发现用户连接数较高，同时实例性能下降并且响应时间延长，请考虑约束数据库连接。实例的最佳用户连接数因实例类和所执行操作的复杂性而异。
- IOPS 指标：IOPS 指标的预期值取决于磁盘规格和服务器配置，因此，请使用您的基准来了解典型状况。调查一下值是否始终与您的基准不同。为获得最佳 IOPS 性能，请确保典型工作集与内存大小相适，以最大限度地减少读取和写入操作。

亚马逊 DocumentDB（兼容 MongoDB）提供了各种亚马逊指标，您可以监控这些 CloudWatch 指标，以确定亚马逊 DocumentDB 集群和实例的运行状况和性能。您可以使用各种工具查看亚马逊 DocumentDB 指标，包括亚马逊 DocumentDB 控制台 AWS CLI CloudWatch 、API 和 Performance Insights。

主题

- 监控 Amazon DocumentDB 集群的状态
- 监控 Amazon DocumentDB 实例的状态
- 查看 Amazon DocumentDB 推荐
- 使用 Amazon DocumentDB 事件订阅
- 使用以下方式监控亚马逊 DocumentDB CloudWatch
- 使用记录亚马逊 DocumentDB API 调用 AWS CloudTrail
- 分析 Amazon DocumentDB 操作
- 使用 Performance Insights 进行监控

监控 Amazon DocumentDB 集群的状态

集群的状态表示集群的运行状况。您可以使用 Amazon DocumentDB 控制台或命令查看集群的 AWS CLI `describe-db-clusters`状态。

主题

- 集群状态值
- 监控集群的状态

集群状态值

下表列出集群状态的有效值。

集群状态	说明
`active`	集群为活动状态。此状态仅适用于弹性集群。
`available`	集群状态良好且可用。此状态仅适用于基于实例的集群。

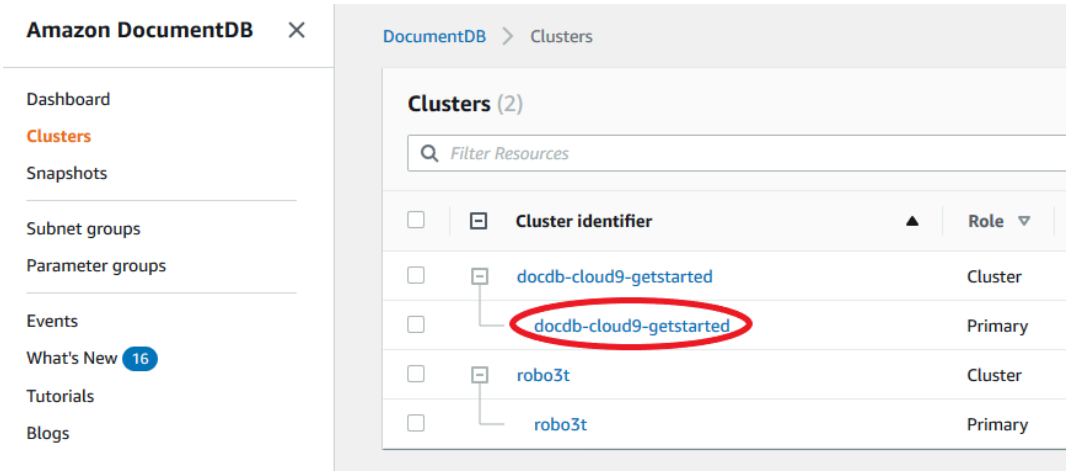
集群状态	说明
backing-up	当前正在备份集群。
creating	正在创建集群。正在创建的过程中无法访问。
deleting	正在删除集群。正在删除的过程中无法访问。
failing-over	正在执行从主实例到 Amazon DocumentDB 副本的失效转移。
inaccessible-encryption-credentials	无法 AWS KMS 访问用于加密或解密集群的密钥。
maintenance	正在对集群应用维护更新。此状态用于 Amazon DocumentDB 预先计划的集群级别维护。
migrating	正将集群快照还原给集群。
migration-failed	迁移失败。
modifying	正在按照客户请求修改集群。
renaming	正在按照客户请求重命名集群。
resetting-master-credentials	正在按照客户请求重置集群的主凭证。
upgrading	集群引擎版本正在升级。

监控集群的状态

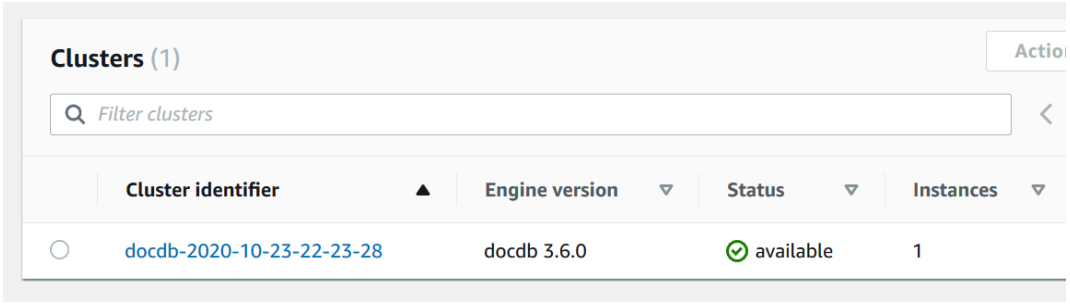
Using the AWS 管理控制台

使用 AWS 管理控制台 来确定群集的状态时，请按以下步骤操作。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。
3. 在集群导航框中，您将看到集群标识符列。您的实例列在集群下，类似于以下屏幕截图。



4. 在集群标识符列中，找到您感兴趣的实例的名称。然后，要查找该实例的状态，请跨该行阅读至状态列，如下所示。



Using the AWS CLI

使用 AWS CLI 来确定集群的状态时，请使用`describe-db-clusters`操作。以下代码可查找集群 `sample-cluster` 的状态。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster  \
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

对于 Windows：

```
aws docdb describe-db-clusters ^
    --db-cluster-identifier sample-cluster  ^
    --query 'DBClusters[*].[DBClusterIdentifier,Status]'
```

此操作的输出将类似于下文。

```
[
    [
        "sample-cluster",
        "available"
    ]
]
```

监控 Amazon DocumentDB 实例的状态

Amazon DocumentDB 提供有关数据库中每个已配置实例当前状况的信息。

您可以对 Amazon DocumentDB 实例查看的状态有三个类型：

- 实例状态：此状态显示在中集群表的 “状态” 列中， AWS 管理控制台 并显示实例的当前生命周期状况。状态中显示的值源自 `DescribeDBCluster` API 响应的 `Status` 字段。
- 实例运行状况：此状态显示在中集群表的 “实例运行状况” 列中， AWS 管理控制台 并显示数据库引擎（负责管理和检索数据的组件）是否正在运行。“实例运行状况” 列中显示的值基于 Amazon CloudWatch `EngineUptime` 系统指标。
- 维护状态：此状态显示在中集群表的 “维护” 列中，表示需要应用于实例的任何维护事件的状态。AWS 管理控制台 维护状态独立于其他实例的状态，并且源自 `PendingMaintenanceAction` API。有关维护状态的更多信息，请参阅维护 Amazon DocumentDB。

主题

- 实例状态值
- 使用监控实例状态 AWS 管理控制台 或者 AWS CLI
- 实例运行状况值
- 使用监控实例的运行状况 AWS 管理控制台

实例状态值

下表列出实例的可能状态值以及如何对每个状态计费。其中显示是否对实例和存储计费、只对存储向您计费，还是不向您计费。对于所有实例状态，始终会针对备份用量向您计费。

实例状态	已计费	说明
available	计费	实例正常和可用。
backing-up	计费	当前正在备份实例。
configuring-log-exports	计费	此实例已启用或禁用将 CloudWatch 日志文件发布到 Amazon Logs。
creating	不计费	正在创建实例。无法访问正在创建的实例。
deleting	不计费	正在删除实例。
failed	不计费	实例已失败，Amazon DocumentDB 无法恢复其。要恢复数据，请将时间点还原至上次还原实例的时间。
inaccessible-encryption-credentials	不计费	无法访问用于加密或解密实例的密 AWS KMS 钥。
incompatible-network	不计费	Amazon DocumentDB 正尝试对实例执行恢复操作，但无法执行此操作，因为 VPC 正处于一种阻止此操作完成的状态。例如，如果子网中的所有可用 IP 地址都在使用中，并且 Amazon DocumentDB 无法为实例获取 IP 地址，就会出现此状态。
maintenance	计费	Amazon DocumentDB 正在对此实例应用维护更新。此状态用于 Amazon DocumentDB 预先计划的实例级别维护。我们将通过此状态评估向客户公开其他维护操作的方式。
modifying	计费	按照请求正在修改实例。

实例状态	已计费	说明
rebooting	计费	按照请求或需要重启实例的 Amazon DocumentDB 过程正在重启实例。
renaming	计费	按照请求正在重命名实例。
resetting-master-credentials	计费	按照请求正在重置实例的主凭证。
restore-error	计费	实例在尝试还原到某个时间点或从快照还原时遇到错误。
starting	对存储计费	实例正在启动。
stopped	对存储计费	实例已停止。
stopping	对存储计费	正在停止实例。
storage-full	计费	实例超出了其存储分配容量。这是一种严重状态，应立即修复；请通过修改实例来扩展存储。将 Amazon CloudWatch 警报设置为在存储空间不足时向您发出警告，这样您就不会遇到这种情况。

使用监控实例状态 AWS 管理控制台 或者 AWS CLI

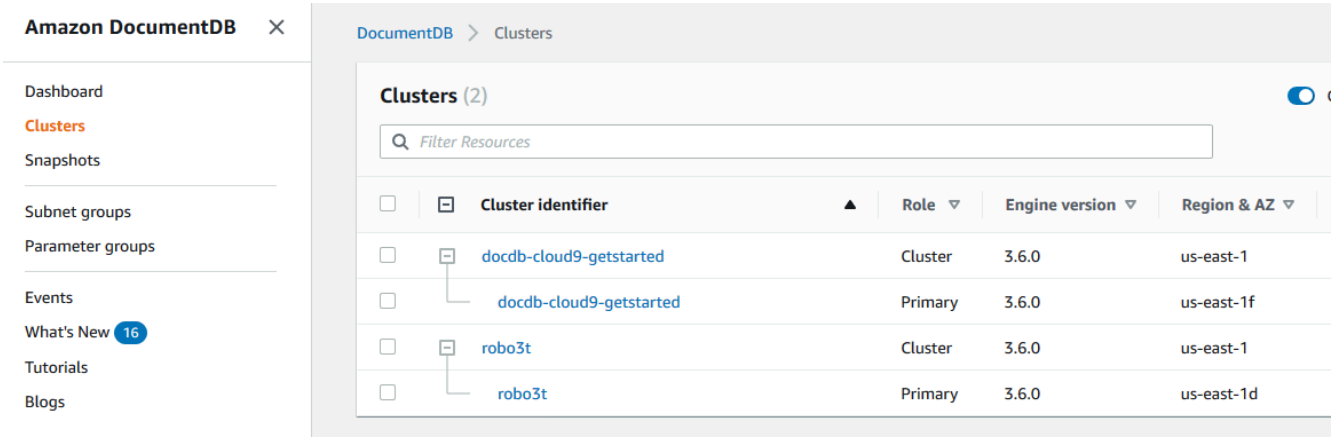
使用 AWS 管理控制台 或 AWS CLI 来监控您的实例的状态。

Using the AWS 管理控制台

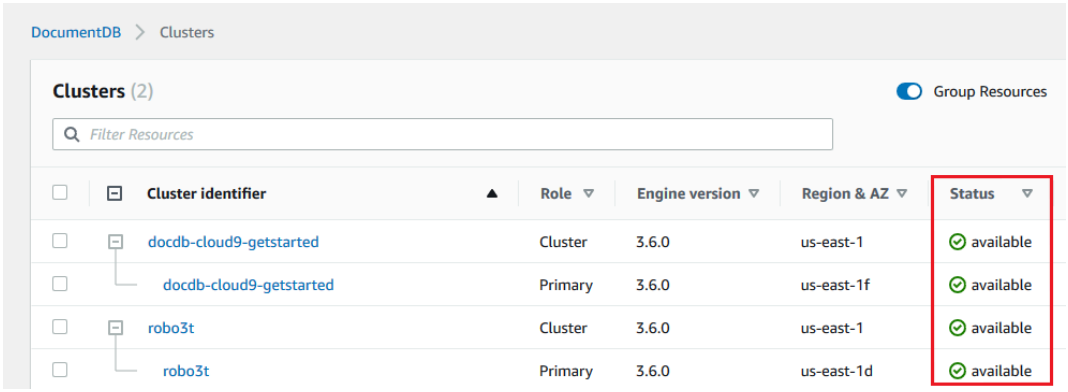
使用 AWS 管理控制台 来确定群集的状态时，请按以下步骤操作。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。

> **Note**
>
> 请注意，在集群导航框中，集群标识符列既显示集群又显示实例。实例列在集群下方，如下图所示。



3. 查找您感兴趣的实例的名称。然后，要查找实例的状态，请跨该行阅读至 Status (状态) 列，如下所示。



Using the AWS CLI

使用 AWS CLI 来确定集群的状态时，请使用`describe-db-instances`操作。以下代码可查找实例 `sample-cluster-instance-01` 的状态。

对于 Linux、macOS 或 Unix：

```
aws docdb describe-db-instances \
          --db-instance-identifier sample-cluster-instance-01  \
```

```
          --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceStatus]'
```

对于 Windows：

```
aws docdb describe-db-instances ^
          --db-instance-identifier sample-cluster-instance-01  ^
          --query 'DBInstances[*].[DBInstanceIdentifier,DBInstanceStatus]'
```

此操作的输出将类似于下文。

```
[
          [
              "sample-cluster-instance-01",
              "available"
          ]
      ]
```

实例运行状况值

下表列出了实例的可能运行状况值。实例运行状况列位于的 Clu st ers 表中 AWS 管理控制台，显示数据库引擎（负责存储、管理和检索数据的组件）是否运行正常。此列还指明中 CloudWatch提供的`EngineUptime`系统指标是否显示每个实例的运行状况。

实例运行状况	说明
正常	数据库引擎正在 Amazon DocumentDB 实例中运行。
运行状况不佳	数据库引擎未在运行或已在不到一分钟前重启。

使用监控实例的运行状况 AWS 管理控制台

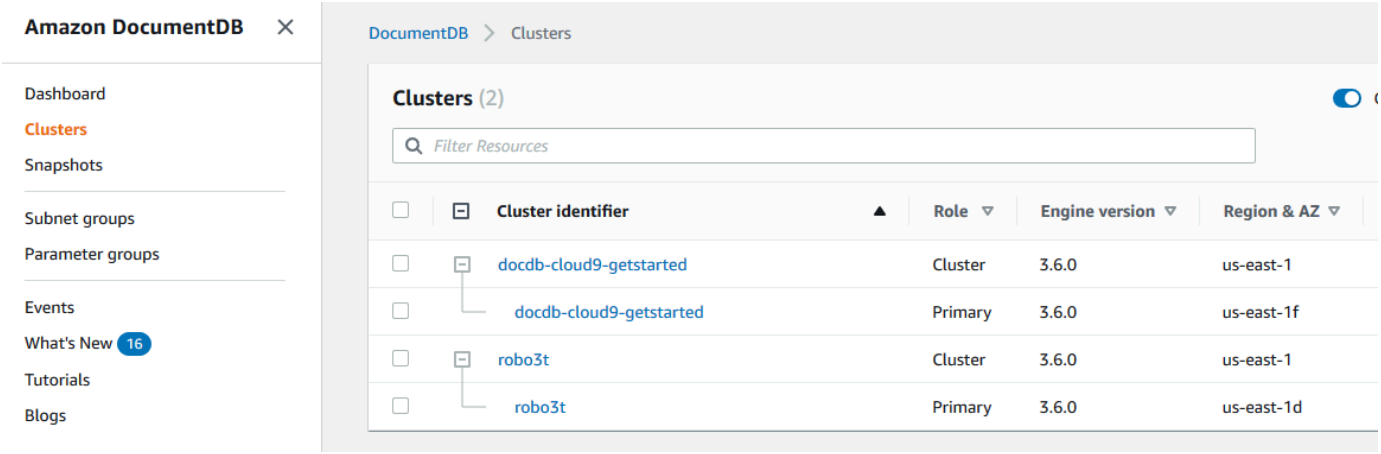
使用 AWS 管理控制台 来监控您的实例的运行状况。

使用时 AWS 管理控制台，请按照以下步骤了解实例的运行状况。

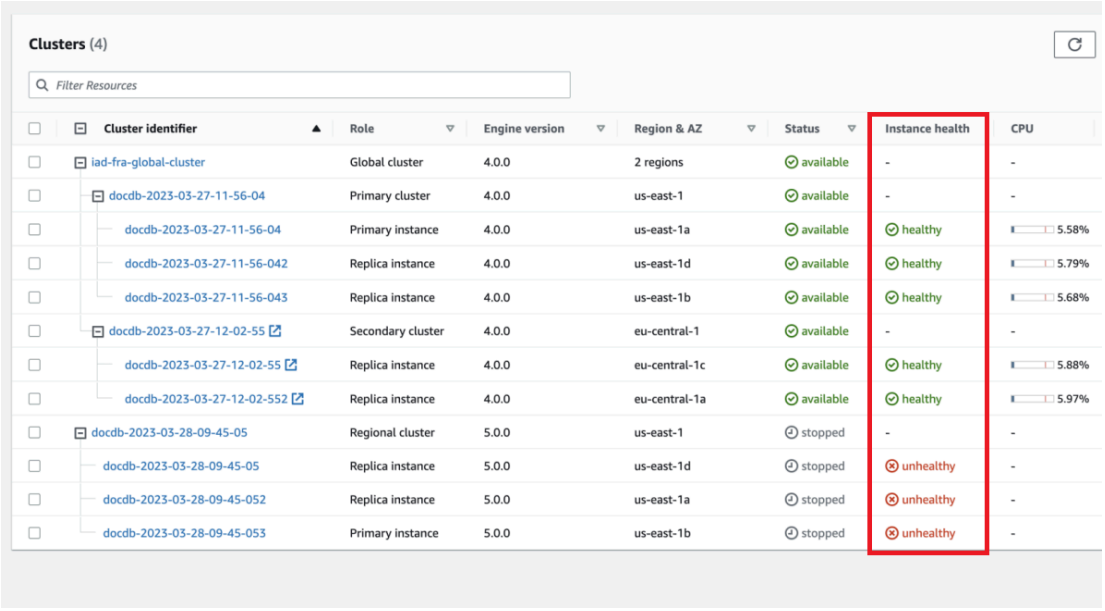
1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb

2. 在导航窗格中，选择集群。

> **Note**
>
> 在集群导航框中，集群标识符列既显示集群又显示实例。实例列在集群下方，如下图所示。



3. 查找您感兴趣的实例的名称。然后，要查找实例的状态，请跨该行阅读至 实例运行状况列，如下图所示：



> **Note**
>
> 实例运行状况轮询每 60 秒进行一次，轮询基于 CloudWatch `EngineUptime`系统指标。实例运行状况列中的值自动更新。

查看 Amazon DocumentDB 推荐

Amazon DocumentDB 为数据库资源（如实例和集群）提供了一系列自动化推荐。这些推荐通过分析集群和实例配置来提供最佳实践指南。

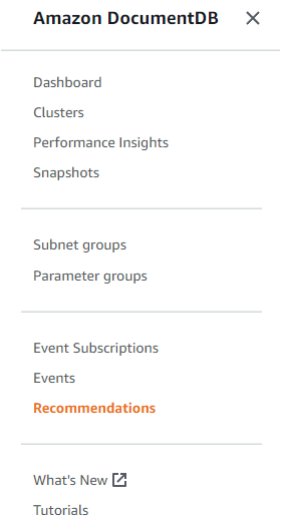
有关这些推荐的示例，请参阅以下内容：

Type	描述	建议	附加信息
1 个 实例	集群仅包含一个实例	性能和可用性：我们建议添加另一个位于不同可用区中的具有相同数据库实例类的实例。	Amazon DocumentDB 高可用性和复制

Amazon DocumentDB 在创建或修改资源时，为资源生成建议。Amazon DocumentDB 还定期扫描您的资源并生成建议。

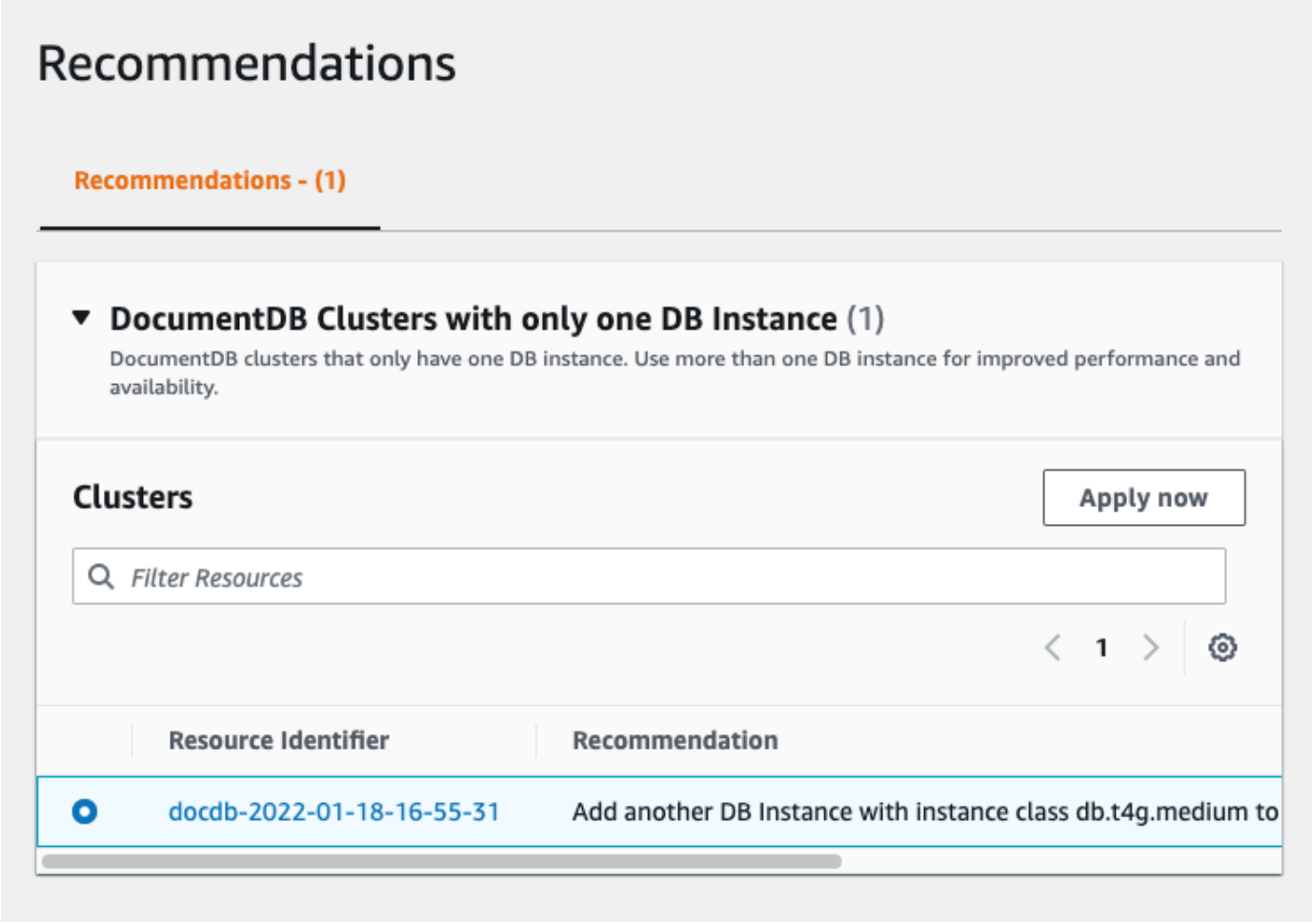
要查看 Amazon DocumentDB 推荐并依照其采取行动

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择推荐：



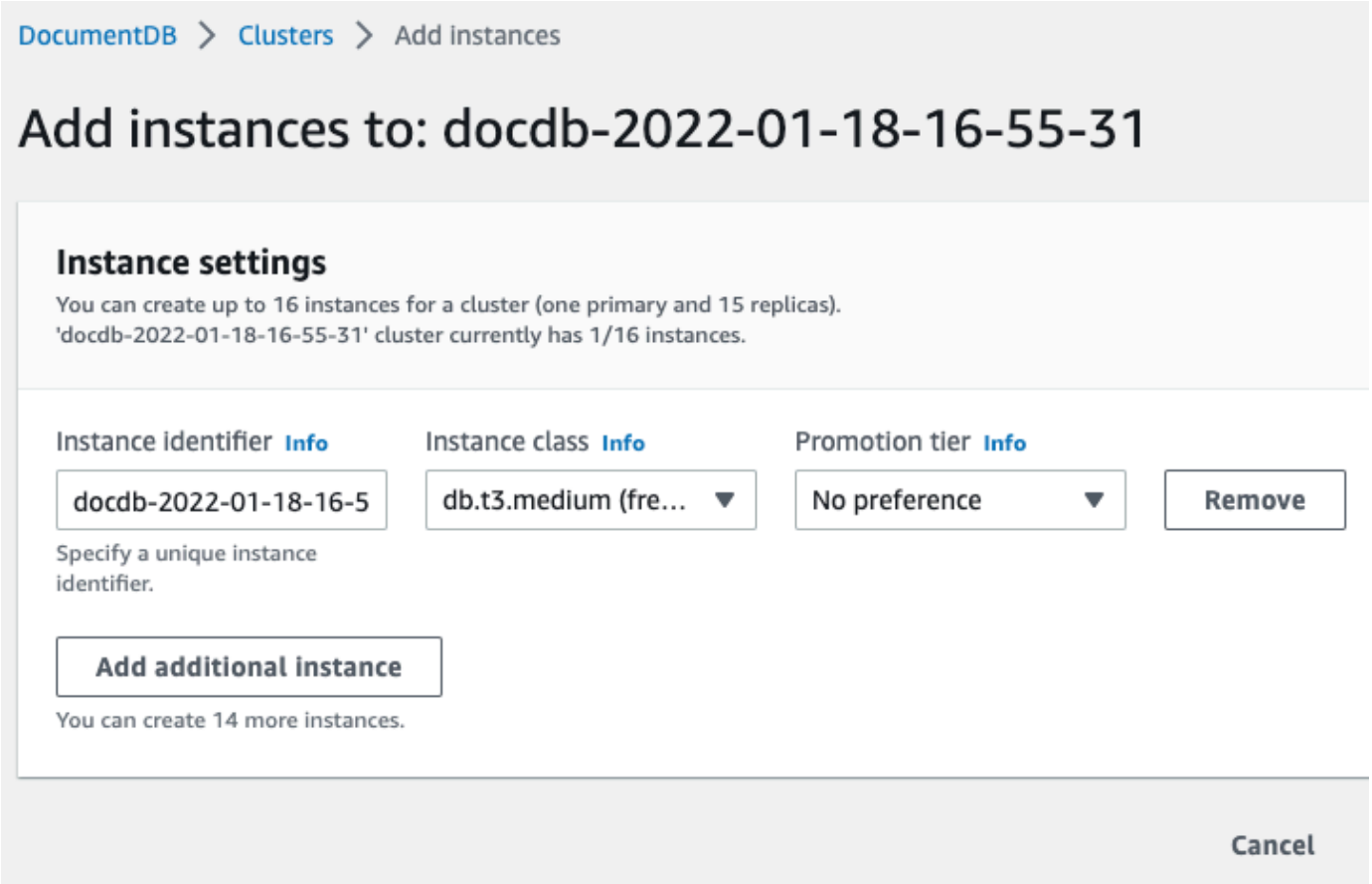
3. 在推荐对话框中，展开目的部分并选择推荐的任务。

   在以下示例中，推荐的任务适用于只有一个实例的 Amazon DocumentDB 集群。推荐应添加另一个实例以改善性能和可用性。



4. 选择 Apply now (立即应用)。

   对于此示例，将出现添加实例对话框：



5. 修改新实例的设置并选择创建。

使用 Amazon DocumentDB 事件订阅

Amazon DocumentDB 使用 Amazon Simple Notification Service (Amazon SNS) 在发生 Amazon DocumentDB 事件时提供通知。这些通知可以采用 Amazon SNS 支持的任何形式 AWS 区域，例如电子邮件、短信或对 HTTP 终端节点的调用。

Amazon DocumentDB 将这些事件分组为您可以订阅的类型，以便您在出现该类事件时收取通知。您可以针对实例、集群、快照、集群快照或参数组订阅事件类别。例如，如果您订阅给定实例的 Backup 类别，那么无论何时出现影响该实例的备份相关事件，您都将收到通知。您还将在事件订阅更改时收到通知。

事件在集群和实例级别发生。所以，如果您针对集群或实例进行订阅，将可收到事件。

事件订阅会发送到您在创建订阅时提供的地址。您可能希望创建多个不同的订阅，如使用一个订阅接收所有事件通知，并使用另一个订阅仅接收针对生产数据库实例的关键事件。您无需删除订阅即可轻松关闭通知。为此，请在 Amazon DocumentDB 控制台中将“启用” 单选按钮设置为“否”。

> **Important**
>
> Amazon DocumentDB 不保证在事件流中发送的事件的顺序。事件顺序可能会发生变化。

Amazon DocumentDB 使用 Amazon SNS 主题的 Amazon 资源名称 (ARN) 标识每个订阅。Amazon DocumentDB 控制台在您创建订阅时为您创建 ARN。

通过 Amazon SNS 对 Amazon DocumentDB 活动订阅计费。Amazon SNS 费用在使用事件通知时适用。有关更多信息，请参阅 Amazon Simple Notification Service Pricing。除了 Amazon SNS 费用外，Amazon DocumentDB 对活动订阅不计费。

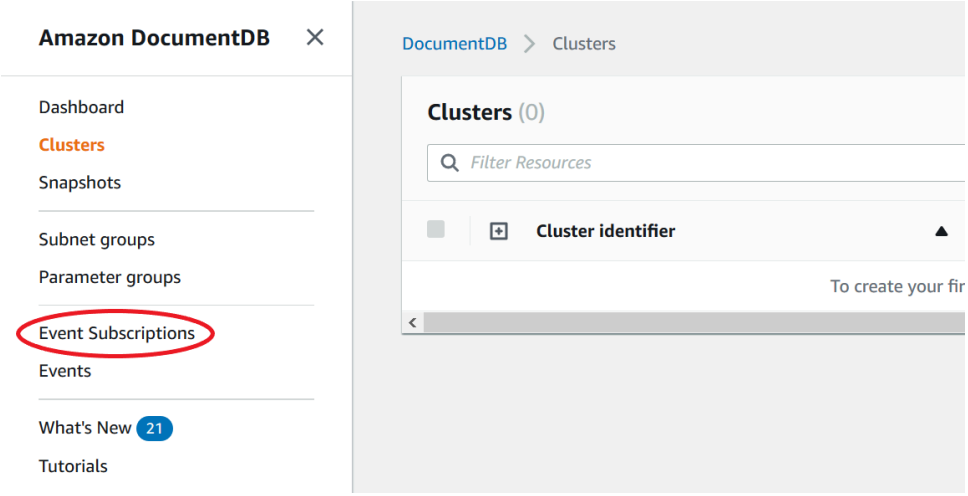
主题

- 订阅 Amazon DocumentDB 事件
- 管理 Amazon DocumentDB 事件通知订阅
- Amazon DocumentDB 事件类别和消息

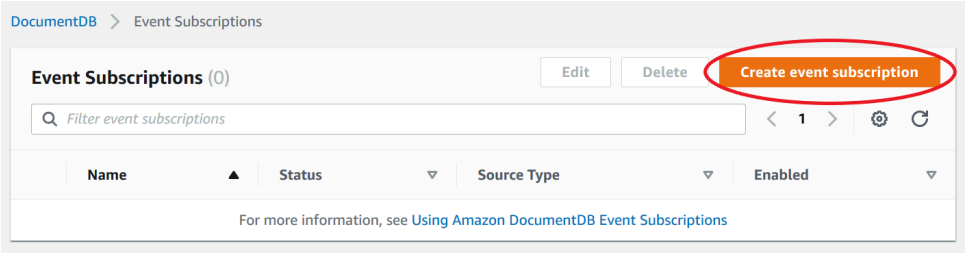
订阅 Amazon DocumentDB 事件

您可以如下使用 Amazon DocumentDB 控制台订阅活动订阅：

1. 登录到 a AWS 管理控制台 t https://console.aws.amazon.com/docdb。
2. 在导航窗格中，选择事件订阅。

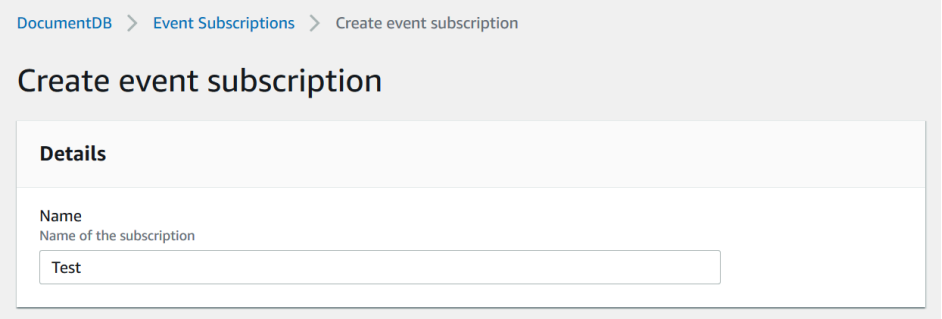


3. 在事件订阅窗格中，选择创建事件订阅。

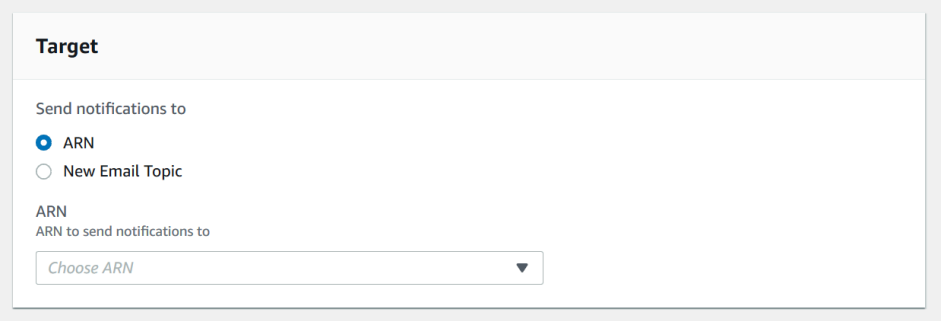


4. 在创建事件订阅对话框中，请执行以下操作：

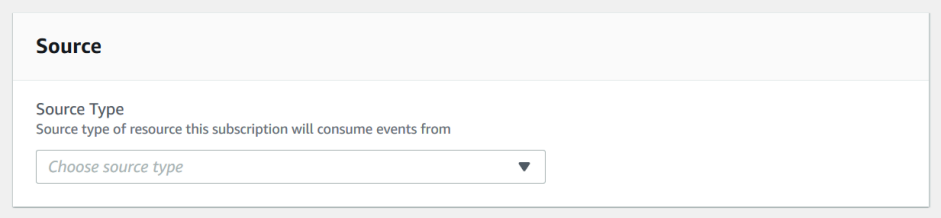
   - 对于名称，输入事件通知订阅的名称。



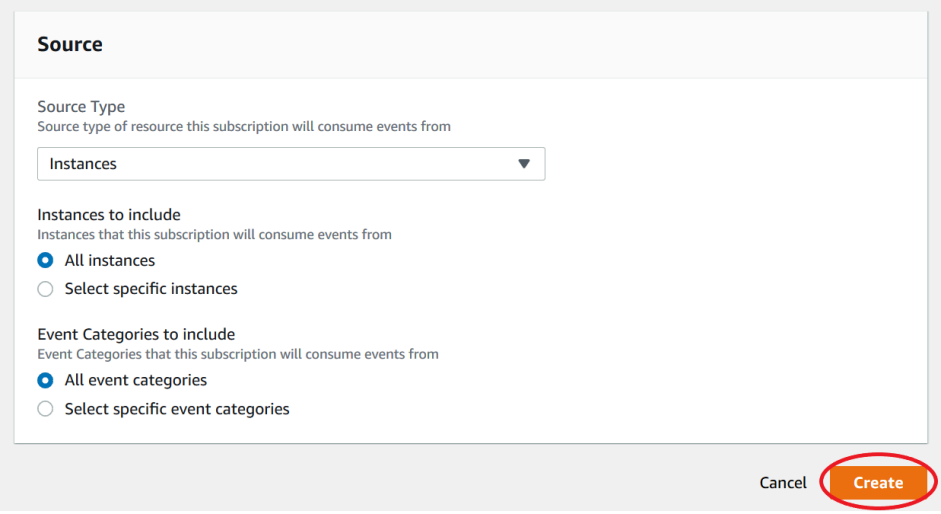
   - 对于目标，选择您想要发送通知到何处。您可以选择现有 ARN 或者选择 “新建电子邮件主题”来输入主题的名称和收件人列表。



- 对于源，请选择一种源类型。根据选定源类型的情况，选择您希望接收来自 事件通知的事件类别和源。



- 选择创建。

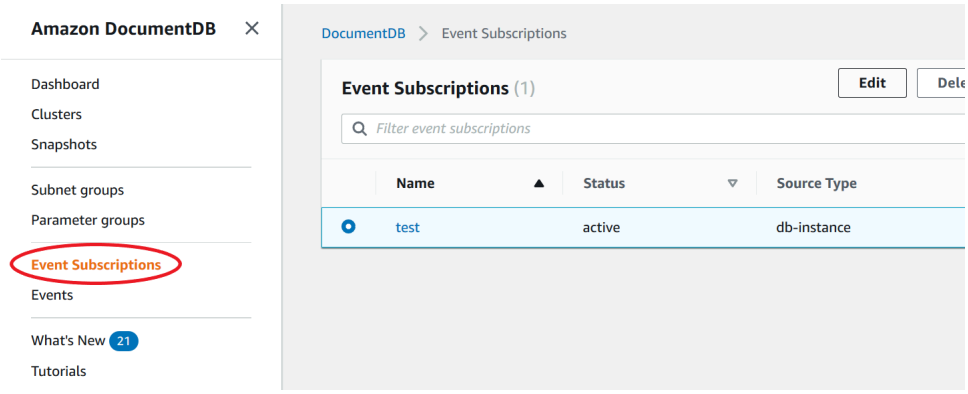


管理 Amazon DocumentDB 事件通知订阅

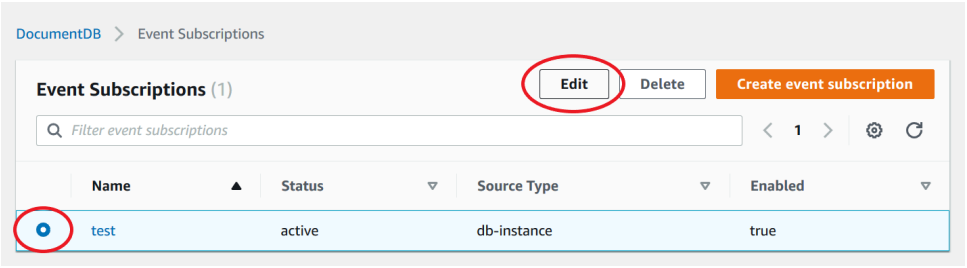
如果您在 Amazon DocumentDB 控制台的导航窗格中选择“活动订阅”，则可以查看订阅类别和您当前订阅的列表。您也可以修改或删除特定的订阅。

列出当前的 Amazon DocumentDB 事件通知订阅

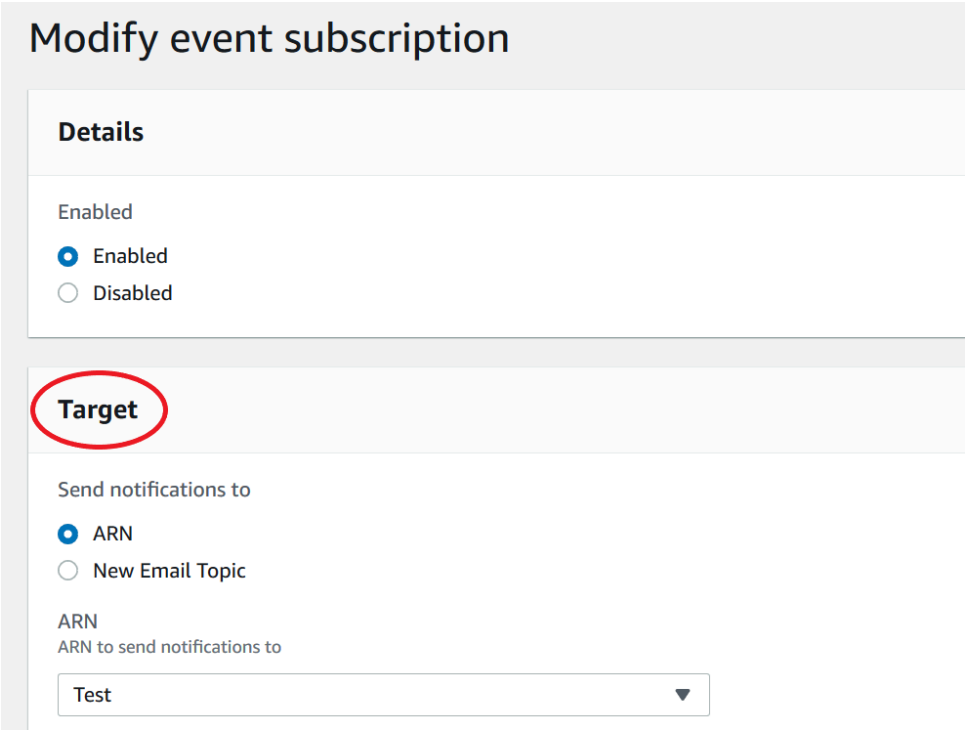
1. 登录到 a AWS 管理控制台 t https://console.aws.amazon.com/docdb。
2. 在导航窗格中，选择事件订阅。事件订阅窗格中会显示您的所有事件通知订阅。



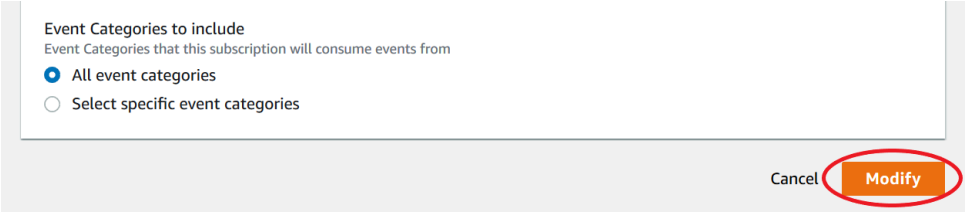
3. 在事件订阅窗格中，选择您要修改的订阅，然后选择编辑。



4. 在目标或来源部分中对订阅进行更改。您可以通过在源部分选择或取消选择源标识符来添加或删除它们。



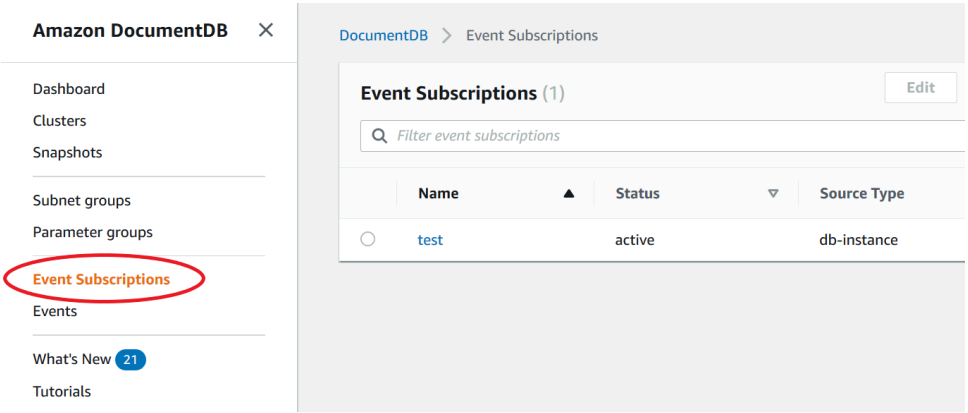
5. 选择 Modify(修改)。Amazon DocumentDB 控制台会表明正在修改订阅。



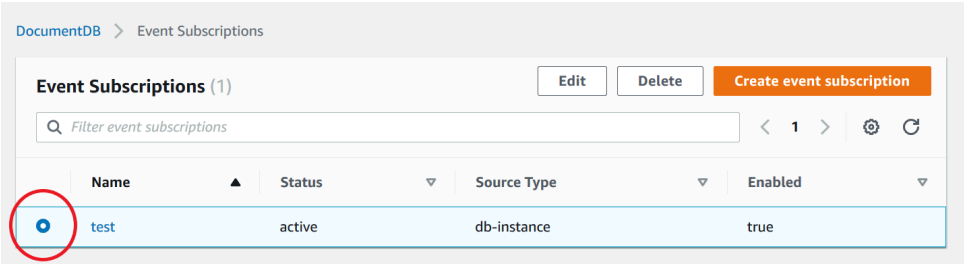
删除 Amazon DocumentDB 事件通知订阅

当您不再需要时，可以删除订阅。该主题的所有用户都将再也不会收到订阅指定的事件通知。

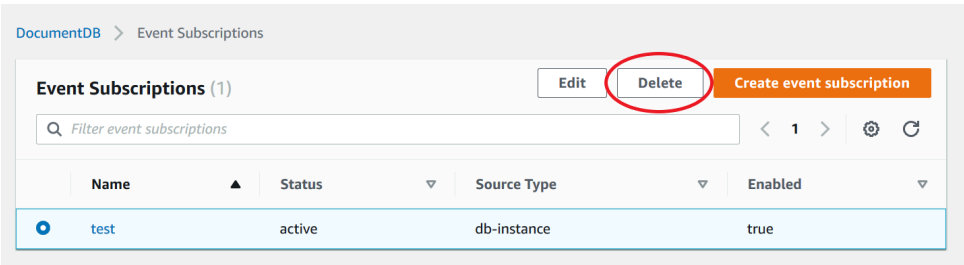
1. 登录到 a AWS 管理控制台 t https://console.aws.amazon.com/docdb。
2. 在导航窗格中，选择事件订阅。



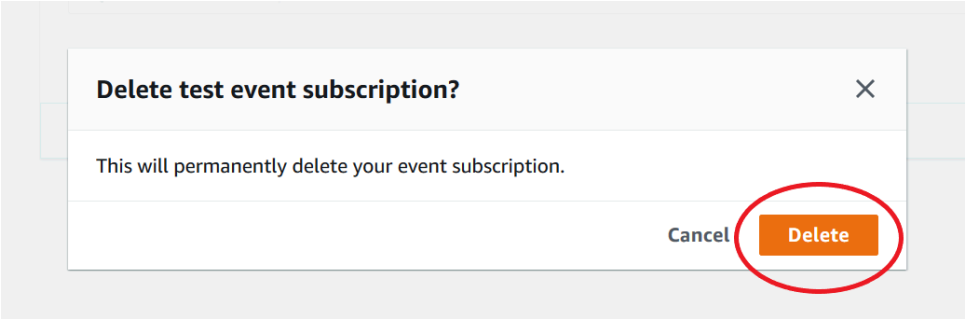
3. 在 事件订阅窗格中，选择您希望删除的订阅。



4. 选择删除。



5. 将出现一个弹出窗口，询问您是否要永久删除此通知。选择删除。



Amazon DocumentDB 事件类别和消息

Amazon DocumentDB 会在各种类型中生成许多事件，您可以使用 控制台对它们进行订阅。每个类别应用于一种源类型，可以是实例、集群、快照或参数组。

> **Note**
>
> Amazon DocumentDB 使用现有的 Amazon RDS 活动定义和 ID。

源自实例的 Amazon DocumentDB 活动

类别	说明
可用性	实例已重启。
可用性	实例已关闭。
配置更改	将修改应用于实例类。
配置更改	已完成将修改应用于实例类。
配置更改	重置主凭证。
创建	实例已创建。
删除	已删除实例
失败	由于某个不兼容配置或底层存储问题，实例已失败。开始对实例进行时间点还原。

类别	说明
notification	实例已停止。
notification	实例已启动。
notification	实例由于它超过最大允许停止的时间而正被启动。
恢复	已启动实例的还原。恢复时间会随待恢复数据量的变化而变化。
恢复	实例的恢复已完成。
安全修补	操作系统更新可用于您的实例。有关应用更新的信息，请参阅维护 Amazon DocumentDB。

源自集群的 Amazon DocumentDB 活动

类别	说明
创建	已创建集群
删除	已删除集群。
故障转移	再次提升之前的主实例。
故障转移	已完成到实例的失效转移。
故障转移	已开始失效转移到数据库实例：%s
故障转移	已开始将同一可用区失效转移到实例：%s
故障转移	已开始跨可用区失效转移到实例：%s
maintenance	集群已修补。
maintenance	数据库集群处于无法升级的状态：%s
notification	已停止集群。

类别	说明
notification	已启动库集群。
notification	集群停止失败。
notification	集群将由于它超过最大允许停止的时间而正被启动。
notification	将群集从 %s 重命名为 %s。

源自集群快照的 Amazon DocumentDB 活动

下表显示了 Amazon DocumentDB 集群快照为源类型时的事件类别和事件列表。

类别	说明
备份	正在创建手动集群快照。
备份	已创建手动集群快照。
备份	正在创建自动集群快照。
备份	已创建自动集群快照。

源自参数组的 Amazon DocumentDB 活动

下表显示的是参数组为源类型时的事件类型和事件列表。

类别	说明
配置更改	借助应用方法 %s 将参数 %s 更新为 %s

使用以下方式监控亚马逊 DocumentDB CloudWatch

Amazon DocumentDB（兼容 MongoDB）与亚马逊集成， CloudWatch 因此您可以收集和分析集群的运行指标。您可以使用 CloudWatch 控制台、Amazon DocumentDB 控制台、 AWS Command Line Interface (AWS CLI) 或 API 来监控这些指标。 CloudWatch

CloudWatch 还允许您设置警报，以便在指标值违反您指定的阈值时收到通知。您甚至可以设置 Amazon Ev CloudWatch ents，以便在发生违规行为时采取纠正措施。有关使用 CloudWatch 和警报的更多信息，请参阅 Amazon CloudWatch 文档。

主题

- Amazon DocumentDB 指标
- 查看 CloudWatch 数据
- Amazon DocumentDB 维度
- 监控 Opcounter 指标
- 监控数据库连接

Amazon DocumentDB 指标

要监控 Amazon DocumentDB 集群和实例的运行状况和性能，您可以在 Amazon DocumentDB 控制台中查看以下指标。

> **Note**
>
> 下表中的指标适用于基于实例的集群和弹性集群。

主题

- 资源利用率指标
- 延迟指标
- NVMe-backed 实例指标
- 操作指标
- 吞吐量指标
- 系统指标
- T3 实例指标

资源利用率指标

指标	说明
BackupRetentionPeriodStorageUsed	用于支持 Amazon DocumentDB 的保留时段内的时间点恢复功能的备份存储总量（以字节为单位）。包含在 TotalBackupStorageBilled 指标报告的总数中。针对每个 Amazon DocumentDB 集群单独计算。
ChangeStreamLogSize	集群用于存储变更流日志的存储量（以兆字节为单位）。此值是集群总存储量的子集 (VolumeBytesUsed)，将影响集群的成本。有关存储定价信息，请参阅 Amazon DocumentDB 产品页面。变更流日志大小取决于集群上发生了多少更改以及变更流日志的保留时间。有关变更流的更多信息，请参阅将变更流与 Amazon DocumentDB 结合使用。
CPUUtilization	实例占用的 CPU 百分比。
DatabaseConnections	在以 1 分钟频率拍摄的实例上打开的连接（活动和空闲）数。
DatabaseConnectionsMax	1 分钟内在实例上打开的最大数据库连接（活动和空闲）数。

指标	说明
DatabaseConnectionsLimit	在任何给定时间，实例上允许的最大并发数据库连接（活动和空闲）数。
DatabaseCursors	在以 1 分钟频率拍摄的实例上打开的光标数。
DatabaseCursorsMax	1 分钟内实例上打开的最大光标数。
DatabaseCursorsLimit	在任何给定时间，实例上允许的最大光标数。
DatabaseCursorsTimedOut	在 1 分钟内超时的光标数。
FreeableMemory	随机存取内存的可用量 (以字节为单位)。
FreeLocalStorage	此指标报告每个实例中可用于临时表和日志的存储量。此值取决于实例类。您可通过为实例选择较大的实例类来增加对实例可用的存储空间量。（这不适用于 DocumentDB 无服务器。）
LowMemThrottleQueueDepth	由于可用内存不足而受到限制的请求的队列深度，频率为 1 分钟。
LowMemThrottleMaxQueueDepth	1 分钟内由于可用内存不足而受到限制的请求的最大队列深度。
LowMemNumOperationsThrottled	1 分钟内由于可用内存不足而受到限制的请求数量。

指标	说明
SnapshotStorageUsed	给定 Amazon DocumentDB 集群的所有快照在其备份保留时段外消耗的备份存储总量（以字节为单位）。包含在 TotalBackupStorageBilled 指标报告的总数中。针对每个 Amazon DocumentDB 集群单独计算。
SwapUsage	实例上使用的交换空间的大小。
TotalBackupStorageBilled	为给定 Amazon DocumentDB 集群计费时所针对的备份存储总量（以字节为单位）。包含由 BackupRetentionPeriodStorageUsed 和 SnapshotStorageUsed 指标度量的备份存储。针对每个 Amazon DocumentDB 集群单独计算。
TransactionsOpen	在以 1 分钟频率拍摄的实例上打开的事务数量。
TransactionsOpenMax	1 分钟内在实例上打开的最大事务数量。
TransactionsOpenLimit	在任何给定时间，实例上允许的最大并发事务数。
VolumeBytesUsed	您的集群使用的存储量（以字节为单位）。此值将影响集群的成本。有关定价信息，请参阅 Amazon DocumentDB 定价页面。

延迟指标

指标	说明	
DBClusterReplicaLagMaximum	数据库集群中主实例和每个 Amazon DocumentDB 实例之间的最大滞后量（以毫秒为单位）。	
DBClusterReplicaLagMinimum	集群中主实例和每个副本实例之间的最小滞后量（以毫秒为单位）。	
DBInstanceReplicaLag	在从主实例向副本实例复制更新时的滞后总量（以毫秒为单位）。	
ReadLatency	每次磁盘 I/O 操作所花费的平均时间。	
WriteLatency	每次磁盘操作所用的平均时间，以毫秒为单位。 I/O	

NVMe-backed 实例指标

指标	说明	
NVMeStorageCacheHitRatio	分层缓存所提供请求的百分比。	
FreeNVMeStorage	可用临时 NVMe 存储的大小。	
ReadIOPSNVMeStorage	对临时 NVMe 存储进行磁盘读取 I/O 操作的平均次数。	
ReadLatencyNVMeStorage	Ephemeral NVMe 存储每次磁盘读取 I/O 操作所花费的平均时间。	

指标	说明
ReadThroughputNVMeStorage	每秒从磁盘向临时 NVMe 存储读取的平均字节数。
WriteIOPSNVMeStorage	对临时 NVMe 存储进行磁盘写入 I/O 操作的平均次数。
WriteLatencyNVMeStorage	临时 NVMe 存储每次磁盘写入 I/O 操作所花费的平均时间。
WriteThroughputNVMeStorage	临时 NVMe 存储每秒写入磁盘的平均字节数。

操作指标

指标	说明
DocumentsDeleted	1 分钟内删除的文档数量。
DocumentsInserted	1 分钟内插入的文档数量。
DocumentsReturned	1 分钟内返回的文档数量。
DocumentsUpdated	1 分钟内更新的文档数量。
OpcountersCommand	1 分钟内发出的命令数。
OpcountersDelete	1 分钟内发出的删除操作数。
OpcountersGetmore	1 分钟内发出的 getmore 数。
OpcountersInsert	1 分钟内发出的插入操作数。
OpcountersQuery	1 分钟内发出的查询数。
OpcountersUpdate	1 分钟内发出的更新操作数。

指标	说明
TransactionsStarted	1 分钟内在实例上启动的事务数量。
TransactionsCommitted	1 分钟内在实例上提交的事务数量。
TransactionsAborted	1 分钟内在实例上中止的事务数量。
TTLDeletedDocuments	TTLMonitor 在 1 分钟内删除的文档数量。

吞吐量指标

指标	说明
NetworkReceiveThroughput	集群中每个实例从客户端接收的网络吞吐量（以每秒字节数为单位）。此吞吐量不包括集群中的实例与集群卷之间的网络流量。
NetworkThroughput	Amazon DocumentDB 集群中每个实例从客户端接收和发送到客户端的网络吞吐量（以每秒字节数为单位）。此吞吐量不包括集群中的实例与集群卷之间的网络流量。
NetworkTransmitThroughput	集群中每个实例发送到客户端的网络吞吐量（以每秒字节数为单位）。此吞吐量不包括集群中的实例与集群卷之间的网络流量。

指标	说明
ReadIOPS	每秒磁盘读取 I/O 操作的平均次数。Amazon DocumentDB 每分钟分别报告一次读取和写入 IOPS。
ReadThroughput	每秒从磁盘读取的平均字节数。
StorageNetworkReceiveThroughput	集群中每个实例从 Amazon DocumentDB 集群存储卷接收的网络吞吐量（以每秒字节数为单位）。
StorageNetworkTransmitThroughput	集群中每个实例发送到 Amazon DocumentDB 集群存储卷的网络吞吐量（以每秒字节数为单位）。
StorageNetworkThroughput	Amazon DocumentDB 集群中每个实例接收自和发送到 Amazon DocumentDB 集群存储卷的网络吞吐量（以每秒字节数为单位）。

指标	说明	
VolumeReadIOPs	集群卷的平均计费读取 I/O 操作数，每隔 5 分钟报告一次。计费读取操作数是在集群卷级别计算的，由集群中的所有实例聚合而来，然后每隔 5 分钟报告一次。此值是通过采用 5 分钟以上的读取操作数指标的值计算得来的。您可通过采用计费读取操作数指标的值并除以 300 秒来确定每秒的计费读取操作数。 例如，如果 VolumeReadIOPs 返回 13,686，则每秒的计费读取操作数为 45 (13,686 / 300 = 45.62)。 您累积请求不在缓冲区缓存中因而必须从存储加载的数据库页的查询的计费读取操作数。您可能看到计费读取操作数出现峰值，因为查询结果是从存储中读取然后加载到缓冲区缓存中的。	

指标	说明	
VolumeWriteIOPs	集群卷的平均计费写入 I/O 操作数，每隔 5 分钟报告一次。计费写入操作数是在集群卷级别计算的，由集群中的所有实例聚合而来，然后每隔 5 分钟报告一次。此值是通过采用 5 分钟以上的写入操作数指标的值计算得来的。您可通过采用计费写入操作数指标的值并除以 300 秒来确定每秒的计费写入操作数。 例如，如果 VolumeWriteIOPs 返回 13686，则每秒的计费写入操作数为 45 (13686 / 300 = 45.62)。 请注意，VolumeReadIOPs 和 VolumeWriteIOPs 指标是由 DocumentDB 存储层计算的，其中包括主实例和副本实例执行的 IO。数据每 20-30 分钟聚合一次，然后每隔 5 分钟报告一次，因此该时间段内该指标的数据点相同。如果您正在寻找与 1 分钟间隔内的插入操作相关联的指标，则可以使用实例级别 WriteIOPs 指标。该指标可在您的 Amazon DocumentDB 主实例的“monitoring”（监控）选项卡中找到。	

指标	说明
WriteIOPS	每秒磁盘写入 I/O 操作的平均次数。在集群级别使用时，WriteIOPs 会对集群中的所有实例进行评估。每分钟分别报告一次读取和写入 IOPS。
WriteThroughput	每秒写入磁盘的平均字节数。

系统指标

指标	说明
AvailableMVCCIds	一个计数器，显示在达到零之前剩余的可用写入操作数。当此计数器达到零时，您的集群将进入只读模式，直至 ID 被回收并循环利用。计数器会在每次写入操作后递减，并在垃圾回收利用旧的 MVCC ID 后递增。
BufferCacheHitRatio	缓冲区缓存提供的请求的百分比。
DiskQueueDepth	等待写入磁盘或从磁盘读取的 I/O 操作数。
EngineUptime	实例已运行的时间长度（以秒为单位）。
IndexBufferCacheHitRatio	缓冲区缓存提供的指数请求的百分比。删除索引、集合或数据库后，您可能会立即看到该指标的峰值超过 100%。60 秒

指标	说明
	后自动更正。此限制将在未来的补丁更新中得到修复。
LongestActiveGCRuntime	最长活动垃圾回收过程的持续时间（以秒为单位）。每分钟更新一次，仅跟踪活动操作，不包括在一分钟时段内完成的进程。

T3 实例指标

指标	说明
CPUCreditUsage	在测量周期内花费的 CPU 积分数。
CPUCreditBalance	实例产生的 CPU 积分数量。在 CPU 突增以及 CPU 积分的花费速度比获得速度快时，该余额将用完。
CPUSurplusCreditBalance	当该 CPUCreditBalance 值为零时，为维持 CPU 性能而花费的剩余 CPU 积分数。
CPUSurplusCreditsCharged	超过可在 24 小时内获得的 CPU 积分数上限的超额 CPU 积分数，因而会产生额外的费用。有关更多信息，请参阅 监控您的 CPU 积分。

查看 CloudWatch 数据

您可以使用 CloudWatch 控制台、亚马逊 DocumentDB 控制台 AWS Command Line Interface (AWS CLI) 或 API 查看亚马逊 CloudWatch 数据。 CloudWatch

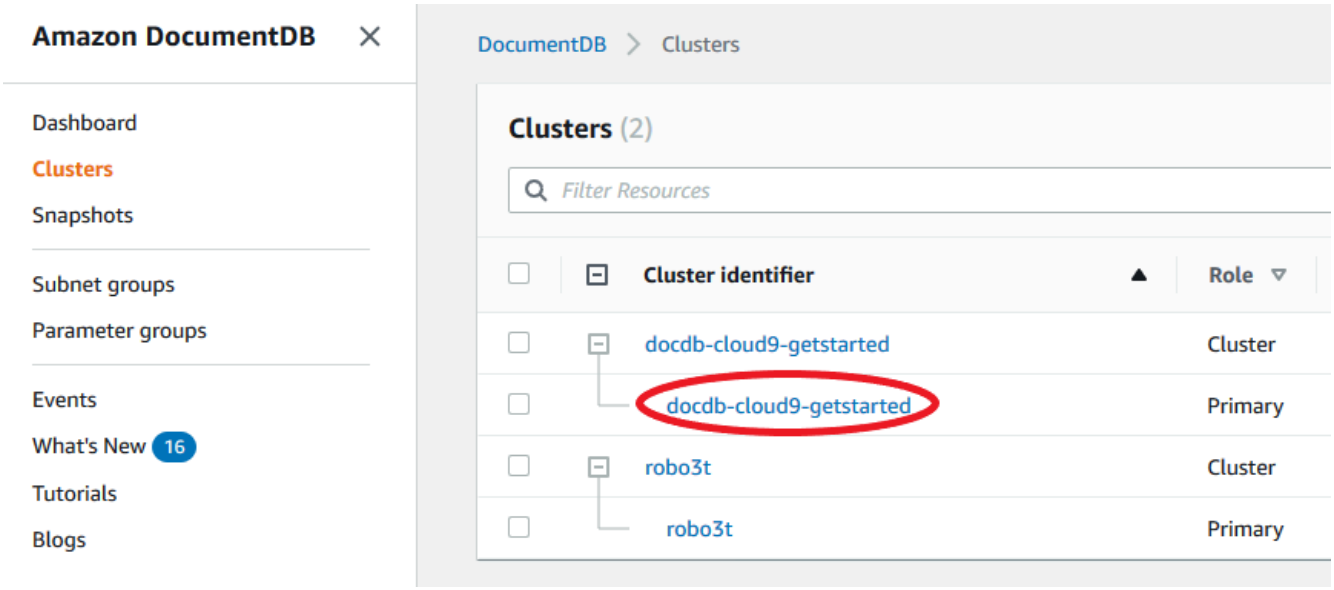
Using the AWS 管理控制台

要使用 Amazon DocumentDB 管理控制台查看 CloudWatch 指标，请完成以下步骤。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。

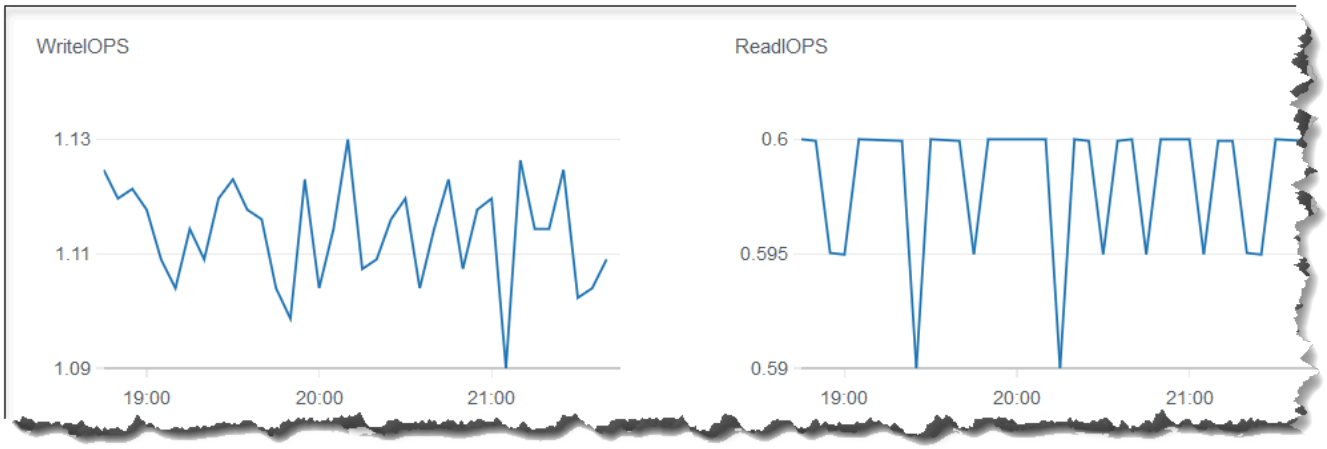
> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

3. 在集群导航框中，您将看到“集群标识符”列。您的实例列在集群下，类似于以下屏幕截图。



4. 从实例列表中选择要获取其指标的实例的名称。
5. 在生成的实例摘要页面中，选择 Monitoring（监控）选项卡，查看您的 Amazon DocumentDB 实例指标的图形表示。由于必须为每个指标生成图表，因此可能需要几分钟才能填充CloudWatch图表。

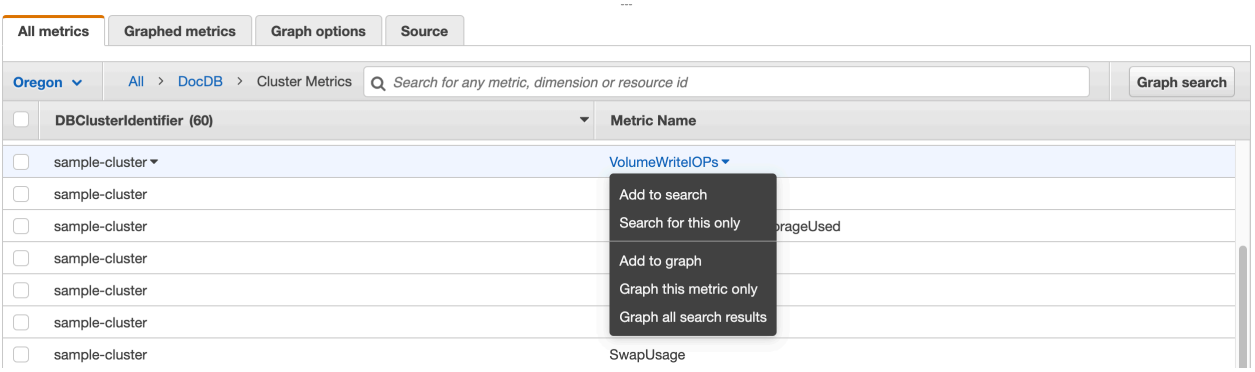
   下图显示了 Amazon DocumentDB 控制台中两个 CloudWatch 指标的图形表示，`WriteIOPS`以及。`ReadIOPS`



Using the CloudWatch Management Console

要使用 CloudWatch 管理控制台查看 CloudWatch 指标，请完成以下步骤。

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/cloudwatch
2. 在导航窗格中，选择指标。然后，从服务名称列表中选择 DocDB。
3. 选择指标维度（例如 集群指标）。
4. All metrics 选项卡显示 DocDB 中该维度的所有指标。
   a. 要对表进行排序，请使用列标题。
   b. 要为指标绘制图表，请选中该指标旁的复选框。要选择所有指标，请选中表的标题行中的复选框。
   c. 要按指标筛选，请将鼠标悬停在指标名称上，然后选择指标名称旁边的下拉箭头。然后，选择 “添加到搜索”，如下图所示。



Using the AWS CLI

要查看 Amazon DocumentDB CloudWatch 的数据，请使用带有以下参数的 CloudWatch `get-metric-statistics`操作。

参数

- **`--namespace`**：必需。需要其 CloudWatch 指标的服务命名空间。对于 Amazon DocumentDB 来说，必须为 `AWS/DocDB`。
- **`--metric-name`**：必需。需要其数据的指标的名称。
- **`--start-time`**：必需。用于确定要返回的第一个数据点的时间戳。

  包含指定的值；结果包括具有指定时间戳的数据点。时间戳必须采用 ISO 8601 UTC 格式（例如，2016-10-03）。T23:00:00Z
- **`--end-time`**：必需。用于确定要返回的最后一个数据点的时间戳。

  包含指定的值；结果包括具有指定时间戳的数据点。时间戳必须采用 ISO 8601 UTC 格式（例如，2016-10-03）。T23:00:00Z
- **`--period`**：必需。返回的数据点的粒度（以秒为单位）。对于具有常规精度的指标，期间可以短到一分钟（60 秒），并且必须为 60 的倍数。对于以小于一分钟的间隔收集的高精度指标，期间可以是 1、5、10、30、60 或 60 的任意倍数。
- **`--dimensions`**— 可选。如果该指标包含多个维度，则必须为每个维度包含一个值。CloudWatch 将每个唯一的维度组合视为一个单独的指标。如果未发布某个特定的维度组合，则无法检索该组合的统计数据。您必须指定创建指标时使用的同一维度。
- **`--statistics`**：可选。百分位数之外的指标统计数据。对于百分位数统计数据，请使用 `ExtendedStatistics`。调用 `GetMetricStatistics` 时，必须指定 `Statistics` 或 `ExtendedStatistics`，但不能同时指定两者。

  允许的值：

  - `SampleCount`
  - `Average`
  - `Sum`
  - `Minimum`
  - `Maximum`

- **--extended-statistics**：可选。percentile 统计数据。指定介于 p0.0 到 p100 之间的值。调用 GetMetricStatistics 时，必须指定 Statistics 或 ExtendedStatistics，但不能同时指定两者。
- **--unit**：可选。给定指标的单位。可以用多个单位报告指标。如果不提供单位，将返回所有单位。如果您仅指定指标不报告的单位，调用的结果将为空。

  可能的值：

  - 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

以下示例查找 2 小时时段的最大 CPUUtilization，每隔 60 秒采样一次。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此操作的输出类似于以下内容：

```
{
       "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 维度

Amazon DocumentDB 的指标由账户或操作的值来限定。您可以使用 CloudWatch 控制台检索按下表中任意维度筛选的 Amazon DocumentDB 数据。

维度	说明
DBClusterIdentifier	筛选您为特定 Amazon DocumentDB 集群请求的数据。
DBClusterIdentifier, Role	筛选您为特定 Amazon DocumentDB 集群请求的数据，按实例角色汇总指标 ()。WRITER/READER例如，您可以聚合属于某个群集的所有 READER 实例的指标。
DBInstanceIdentifier	筛选您为特定数据库实例请求的数据。

监控 Opcounter 指标

对于空闲集群，Opcounter 指标具有非零值（通常约为 50）。这是因为 Amazon DocumentDB 会定期执行运行状况检查、内部操作和指标收集任务。

监控数据库连接

当你使用数据库引擎命令查看连接数时`db.runCommand( { serverStatus: 1 })`，你看到的连接数可能比你看到的`DatabaseConnections`多达 10 个 CloudWatch。发生这种情况的原因是，Amazon DocumentDB 执行定期运行状况检查和指标收集任务，而这些任务不记入`DatabaseConnections`。`DatabaseConnections` 仅显示客户启动的连接数。

使用记录亚马逊 DocumentDB API 调用 AWS CloudTrail

Amazon DocumentDB（兼容 MongoDB）与一项服务集成，可记录用户 AWS CloudTrail、角色或 AWS 亚马逊文档数据库中的服务所采取的操作（兼容 MongoDB）。 CloudTrail 将亚马逊文档数据库的所有 AWS CLI API 调用捕获为事件，包括来自亚马逊 DocumentDB 控制台的调用和对亚马逊 DocumentDB 软件开发工具包的代码调用。如果您创建跟踪，则可以允许将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括亚马逊 DocumentDB 的事件。如果您未配置跟踪，您仍然可以在 CloudTrail 主机上的事件历史记录中查看最新的事件。使用收集的信息 CloudTrail，您可以确定向 Amazon DocumentDB 发出的请求（兼容 MongoDB）、发出请求的 IP 地址、谁发出了请求、何时发出请求以及其他详细信息。

> **Important**
>
> 对于某些管理功能，Amazon DocumentDB 使用与 Amazon Relational Database Service (Amazon RDS) 共享的操作技术。亚马逊 DocumentDB 控制台和 API 调用记录为对亚马逊 RDS API 的调用。 AWS CLI

要了解更多信息 AWS CloudTrail，请参阅《AWS CloudTrail 用户指南》。

中的亚马逊 DocumentDB 信息 CloudTrail

CloudTrail 在您创建账户 AWS 账户 时已在您的账户上启用。当 Amazon DocumentDB（兼容 MongoDB）中发生活动时，该活动会与其他 AWS 服务 CloudTrail 事件一起记录在事件历史记录中。您可以在中查看、搜索和下载最近发生的事件 AWS 账户。有关更多信息，请参阅使用事件历史记录查看 CloudTrail 事件。

要持续记录您的事件 AWS 账户，包括亚马逊 DocumentDB 的事件（兼容 MongoDB），请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。默认情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有 AWS 区域。此跟踪记录在 AWS 分区中记录所有区域中的事件，并将日

志文件传送至您指定的 Amazon S3 存储桶。此外，您可以配置其他 AWS 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅《AWS CloudTrail 用户指南》中的以下主题：

- 创建跟踪概述
- CloudTrail 支持的服务和集成
- 配置 Amazon SNS 通知 CloudTrail
- 接收来自多个区域的 CloudTrail 日志文件
- 从多个账户接收 CloudTrail 日志文件

每个事件或日志条目都包含有关生成请求的人员的信息。身份信息有助于您确定以下内容：

- 请求是使用根凭证还是用户凭证发出的。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

分析 Amazon DocumentDB 操作

可以使用 Amazon DocumentDB（与 MongoDB 兼容）中的分析器来记录在您集群上执行的操作的执行时间和详细信息。对于监控集群上速度最慢的操作以帮助您提高单个查询的性能和整体集群性能，分析器非常有用。

默认情况下，分析器功能处于禁用状态。启用后，Profiler 会将花费超过客户定义的阈值（例如 100 毫秒）的操作记录到 Ama CloudWatch zon Logs 中。记录的详细信息包括分析的命令、时间、计划摘要和客户端元数据。将操作记录到 CloudWatch 日志后，您可以使用 CloudWatch Logs Insights 来分析、监控和存档您的 Amazon DocumentDB 分析数据。常见查询 部分中提供了常见的查询。

在启用时，分析器会使用集群中的其他资源。我们建议您从较高的阈值（例如，500 毫秒）开始，然后逐步降低该值以确定缓慢的操作。对于高吞吐量应用程序，从 50 毫秒阈值开始会导致集群性能问题。分析器在集群级别启用，并对集群中的所有实例和数据库执行分析。Amazon DocumentDB 会尽力将操作记录到亚马逊 CloudWatch 日志。

尽管 Amazon DocumentDB 不会为启用分析器收取任何额外费用，但您需要按标准费率支付日志使用费。 CloudWatch 有关 CloudWatch 日志定价的信息，请参阅 Amazon CloudWatch 定价。

主题

- 支持的操作
- 限制
- 启用 Amazon DocumentDB 分析器
- 禁用 Amazon DocumentDB 分析器
- 禁用分析器日志导出
- 访问您的 Amazon DocumentDB 分析器日志
- 常见查询

支持的操作

Amazon DocumentDB 分析器支持以下操作：

- `aggregate`
- `count`
- `delete`
- `distinct`
- `find`（OP_QUERY 和命令）
- `findAndModify`
- `insert`
- `update`

限制

仅当查询的整个结果集能够容纳在一个批处理中，并且结果集小于 16MB（最大 BSON 大小）时，慢速查询分析器才能够生成分析器日志。大于 16MB 的结果集会自动拆分为多个批处理。

大多数驱动程序或 Shell 可能会设置一个较小的默认批处理大小。您可以在查询中指定批处理大小。为了捕获慢速查询日志，我们建议设置一个超过您预期结果集大小的批处理大小。如果不确定结果集大小，或者结果集大小不同，也可以将批处理大小设置为较大的数字（例如，100k）。

但是，使用较大的批大小意味着在将响应发送到客户端之前，必须从数据库中检索更多结果。对于某些查询，这可能会在获得结果之前造成更长的延迟。如果您不打算使用整个结果集，则可能会花更多的钱 I/Os 来处理查询并丢弃结果。

启用 Amazon DocumentDB 分析器

在集群上启用分析器的过程包含三个步骤。确保所有步骤都已完成，否则分析日志将不会发送到 CloudWatch 日志。分析器在集群级别设置，对集群的所有数据库和实例执行分析。

在集群上启用分析器

1. 由于您无法修改默认集群参数组，请确保您有可用的自定义集群参数组。有关更多信息，请参阅创建 Amazon DocumentDB 集群参数组。
2. 使用可用的自定义集群参数组，修改以下参数：`profiler`、`profiler_threshold_ms` 和 `profiler_sampling_rate`。有关更多信息，请参阅 修改 Amazon DocumentDB 集群参数组。
3. 创建或修改您的集群以使用自定义集群参数组并启用将`profiler`日志导出到 CloudWatch 日志的功能。

以下各节介绍如何使用 AWS 管理控制台 和 AWS Command Line Interface (AWS CLI) 实现这些步骤。

Using the AWS 管理控制台

1. 开始之前，请先创建一个 Amazon DocumentDB 集群和一个自定义集群参数组（如果您还没有）。有关更多信息，请参阅创建 Amazon DocumentDB 集群参数组和创建 Amazon DocumentDB 集群。
2. 使用可用的自定义集群参数组，修改以下参数。有关更多信息，请参阅 修改 Amazon DocumentDB 集群参数组。

   - `profiler`：启用或禁用查询分析。允许的值为 `enabled` 和 `disabled`。默认值为 `disabled`。要启用分析，请将值设置为 `enabled`。
   - `profiler_threshold_ms`— 如果设置`profiler`为`enabled`，则所有花费时间超过其时间的命令 `profiler_threshold_ms`都将被记录到 CloudWatch。允许的值为 `[50-INT_MAX]`。默认值为 `100`。
   - `profiler_sampling_rate`：应该分析或记录的缓慢操作的部分。允许的值为 `[0.0-1.0]`。默认值为 `1.0`。

3. 修改您的集群以使用自定义集群参数组，并将分析器日志导出设置为发布到 Amazon CloudWatch。

   a. 在导航窗格中，选择 Clusters (集群) 以将自定义参数组添加到集群。

b. 选择要与您的参数组关联的集群名称左边的按钮。选择 Actions (操作)，然后选择 Modify (修改) 以修改您的集群。

c. 在 Cluster options (集群选项) 下，选择上一步中的自定义参数组以将其添加到集群中。

d. 在“日志导出”下，选择要发布到 Amazon CloudWatch 的 Profiler 日志。

e. 选择 Continue (继续) 以查看修改摘要。

f. 在确认您的更改后，您可以立即应用这些更改，也可以在 Scheduling of modifications (修改计划) 下的下一个维护时段内应用这些更改。

g. 选择 Modify cluster (修改集群) 以使用新参数组更新您的集群。

Using the AWS CLI

以下过程对集群 `sample-cluster` 上的所有支持操作启用分析器。

1. 在开始之前，请运行以下命令，并查看对于名称中不包含 `default` 且具有 `docdb3.6` 作为参数组系列的集群参数组的输出，以确保您拥有可用的自定义集群参数组。如果您没有非默认集群参数组，请参阅创建 Amazon DocumentDB 集群参数组。

```
aws docdb describe-db-cluster-parameter-groups \
    --query 'DBClusterParameterGroups[*].
[DBClusterParameterGroupName,DBParameterGroupFamily]'
```

在以下输出中，仅 `sample-parameter-group` 满足这两个条件。

```
[
       [
           "default.docdb3.6",
           "docdb3.6"
       ],
       [
           "sample-parameter-group",
           "docdb3.6"
       ]
]
```

2. 使用您的自定义集群参数组，修改以下参数。

- `profiler`：启用或禁用查询分析。允许的值为 `enabled` 和 `disabled`。默认值为 `disabled`。要启用分析，请将值设置为 `enabled`。

- profiler_threshold_ms— 如果设置profiler为enabled，则所有命令花费的时间profiler_threshold_ms都超过了记录到的时间 CloudWatch。允许的值为 [50-INT_MAX]。默认值为 100。
- profiler_sampling_rate：应该分析或记录的缓慢操作的部分。允许的值为 [0.0-1.0]。默认值为 1.0。

```
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. 修改您的 Amazon DocumentDB 集群，使其使用上一步中提到的 sample-parameter-group 自定义集群参数组，并将参数 --enable-cloudwatch-logs-exports 设置为 profiler。

   以下代码修改群集sample-cluster以使用上一步sample-parameter-group中的，并profiler添加到已启用的 CloudWatch 日志导出中。

```
aws docdb modify-db-cluster \
       --db-cluster-identifier sample-cluster \
       --db-cluster-parameter-group-name sample-parameter-group \
       --cloudwatch-logs-export-configuration '{"EnableLogTypes":["profiler"]}'
```

   此操作的输出将类似于下文。

```
{
    "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
    }
}
```

禁用 Amazon DocumentDB 分析器

要禁用探查器，您可以禁用`profiler`参数和将日志导出到`profiler`日 CloudWatch 志。

禁用分析器

您可以使用 AWS 管理控制台 或禁用该`profiler`参数 AWS CLI，如下所示。

Using the AWS 管理控制台

以下过程使用来禁用 Amazon Documen `profiler` tDB。 AWS 管理控制台

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择参数组。然后选择您要在其上禁用分析器的集群参数组的名称。
3. 在生成的 Cluster parameters (集群参数) 页面中，选择 `profiler` 参数左侧的按钮，然后选择 Edit (编辑)。
4. 在 Modify Profiler (修改分析器) 对话框中，在列表中选择 `disabled`。
5. 选择 Modify cluster parameter (修改集群参数)。

Using the AWS CLI

要使用 AWS CLI在集群上禁用 `profiler`，请如下所示修改集群。

```
aws docdb modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name sample-parameter-group \
    --parameters
 ParameterName=profiler,ParameterValue=disabled,ApplyMethod=immediate
```

禁用分析器日志导出

您可以使用 AWS 管理控制台 或禁用将`profiler` CloudWatch 日志导出到日志 AWS CLI，如下所示。

Using the AWS 管理控制台

以下过程使用禁用 Amazon DocumentDB AWS 管理控制台 将日志导出到。 CloudWatch

1. 打开 Amazon DocumentDB 控制台，网址为。https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择集群。选择要禁用导出日志的集群名称左侧的按钮。
3. 在 Actions (操作) 菜单上，选择 Modify (修改)。
4. 向下滚动到 Log exports (日志导出) 部分并取消选择 Profiler logs (分析器日志)。
5. 选择继续。
6. 检查更改，然后选择何时将该更改应用到集群：
   - Apply during the next scheduled maintenance window (在下一个计划的维护时段内应用)
   - Apply immediately (立即应用)
7. 选择修改集群。

Using the AWS CLI

以下代码修改集群`sample-cluster` 并禁用 CloudWatch 探查器日志。

Example

对于 Linux、macOS 或 Unix：

```
aws docdb modify-db-cluster \
   --db-cluster-identifier sample-cluster \
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["profiler"]}'
```

对于 Windows：

```
aws docdb modify-db-cluster ^
   --db-cluster-identifier sample-cluster ^
   --cloudwatch-logs-export-configuration '{"DisableLogTypes":["profiler"]}'
```

此操作的输出将类似于下文。

```
{
    "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
    }
}
```

访问您的 Amazon DocumentDB 分析器日志

按照以下步骤访问您在Amazon上的个人资料日志 CloudWatch。

1. 打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 确保您与 Amazon DocumentDB 集群位于同一区域。
3. 在导航窗格中，选择日志。
4. 要查找集群的分析器日志，请在列表中选择 /aws/docdb/*yourClusterName*/profiler。

   此时，每个实例名称的下方将显示该实例的分析日志。

常见查询

以下是您可以用来分析您的已分析命令的常见查询。有关 Lo CloudWatch gs Insights 的更多信息，请参阅使用 CloudWatch 日志见解和示例查询分析日志数据。

获取指定集合上最慢的 10 个操作

```
filter ns="test.foo" | sort millis desc | limit 10
```

获取集合上用时超过 60 毫秒的所有更新操作

```
filter millis > 60 and op = "update"
```

获取上个月最慢的 10 个操作

```
sort millis desc | limit 10
```

获取具有 COLLSCAN 计划摘要的所有查询

```
filter planSummary="COLLSCAN"
```

使用 Performance Insights 进行监控

Performance Insights 添加到现有的 Amazon DocumentDB 监控功能中，以展示您的集群性能并帮助您分析影响集群性能的任何问题。利用 Performance Insights 控制面板，您可以可视化数据库负载并按等待状态、查询语句、主机或应用来筛选负载。

> **Note**
>
> Performance Insights 仅适用于亚马逊 DocumentDB 3.6、4.0、5.0 和 8.0 基于实例的集群。

它有何用处？

- 可视化数据库性能：可视化负载以确定负载在数据库上的时间和位置
- 确定导致数据库负载的原因：确定哪些查询、主机和应用程序导致了实例上的负载
- 确定数据库何时出现负载 ：放大 Performance Insights 控制面板以关注特定事件，或缩小以查看更大时间跨度的趋势
- 数据库加载警报 — 自动访问新的数据库负载指标，您可以从中 CloudWatch 监控数据库负载指标以及其他 Amazon DocumentDB 指标，并对其设置警报

Amazon DocumentDB Performance Insights 有哪些局限性？

- AWS GovCloud (US-East) 和 AWS GovCloud (US-West) 区域的 Performance Insights 不可用
- Performance Insights for Amazon DocumentDB 最多可保留 7 天的性能数据
- 长度超过 1024 字节的查询不会在性能详情中聚合

主题

- Performance Insights 概念
- 启用和禁用 Performance Insights
- 为 Performance Insights 配置访问策略
- 使用 Performance Insights 控制面板分析指标
- 使用 Performance Insights API 检索指标
- Performance Insights 的亚马逊 CloudWatch 指标
- Performance Insights 的计数器指标

Performance Insights 概念

主题

- 平均活动会话数
- Dimensions
- 最大 vCPU

平均活动会话数

数据库负载（数据库负载）衡量数据库中的活动级别。Performance Insights 的关键指标是 DB Load，每秒收集一次。DBLoad 指标的单位是 Amazon DocumentDB 实例的平均活动会话数 (AAS)。

活动会话是已将作业提交到 Amazon DocumentDB 实例并且正在等待响应的连接。例如，如果您将查询提交到 Amazon DocumentDB 实例，则数据库会话在实例处理该查询时将处于活动状态。

为了获取平均活动会话数，Performance Insights 会对同时运行查询的会话数进行采样。平均活动会话数是会话总数除以样本总数。下表显示了正在运行的查询的五个连续示例。

示例	运行查询的会话数	AAS	计算
1	2	2	2 个会话/1 个样本
2	0	1	2 个会话/2 个样本
3	4	2	6 个会话/3 个样本
4	0	1.5	6 个会话/4 个样本
5	4	2	10 个会话/5 个样本

在上一示例中，1-5 时间间隔的数据库负载为 2 AAS。数据库负载的增加意味着，平均而言数据库上运行的会话更多。

Dimensions

DB Load 指标不同于其他时间序列指标，因为您可以将它分为称为维度的子组件。您可以将维度视为 DB Load 指标的不同特征的类别。诊断性能问题时，最有用的维度是等待状态和主要查询。

等待状态

等待状态 会导致查询语句等待特定事件发生，然后才能继续运行。例如，查询语句可能会一直等到已锁定的资源得到解锁。通过结合使用 DB Load 和等待状态，您可以全面了解会话状态。以下是各种 Amazon DocumentDB 等待状态：

Amazon DocumentDB 等待状态	等待状态描述
Latch	当会话等待分页缓冲池时，就会出现 Latch 等待状态。当系统频繁处理大型查询、集合扫描或缓冲池太小而无法处理工作集时，频繁分页和退出缓冲池的情况可能会更频繁。
CPU	当会话在 CPU 上等待时，就会出现 CPU 等待状态。
CollectionLock	当会话 CollectionLock 等待获取集合锁定时，就会出现等待状态。当对集合进行 DDL 操作时，就会发生这些事件。
DocumentLock	当会话 DocumentLock 等待获取文档锁时，会出现等待状态。对同一文档进行大量并发写入将导致该文档的 DocumentLock等待状态增加。
SystemLock	当 SystemLock 会话在系统上等待时，就会出现等待状态。当系统上频繁出现长时间运行的查询、长时间运行的事务或高并发时，可能会发生这种情况。
IO	当会话等待 IO 完成时，就会出现 IO 等待状态。
BufferLock	当会话 BufferLock 等待获取缓冲区中共享页面的锁时，就会出现等待状态。 BufferLock如果其他进程在请求的页面上持有打开的游标，则等待状态可能会延长。
LowMemThrottle	由于 Amazon DocumentDB 实例的内存压力过大而导致会话处于 LowMemThrottle 等待状态

Amazon DocumentDB 等待状态	等待状态描述
	时，就会出现等待状态。如果此状态持续很长时间，请考虑纵向扩展实例以提供额外的内存。有关更多信息，请参阅资源管理器。
BackgroundActivity	当会话正在 BackgroundActivity 等待内部系统进程时，会出现等待状态。
其他	其他等待状态是内部等待状态。如果此状态持续很长时间，请考虑终止此查询。有关更多信息，请参阅如何查找并终止长时间运行或受阻的查询？

主要查询

等待状态太显示瓶颈，主要查询则显示哪些查询对数据库负载的贡献最大。例如，当前可能正在数据库上运行许多查询，但单个查询可能会占用 99% 的数据库负载。在这种情况下，高负载可能表示查询存在问题。

最大 vCPU

在控制面板中，数据库负载图表会收集、聚合和显示会话信息。要查看活动会话是否超过最大 CPU，请查看它们与最大 vCPU 线的关系。最大 vCPU 值由 Amazon DocumentDB 实例的 vCPU（虚拟 CPU）内核数决定。

如果数据库负载经常高于最大 vCPU 线并且主要等待状态为 CPU，则表示 CPU 过载。在这种情况下，您可能需要限制与实例的连接数，优化具有高 CPU 负载的任何查询，或考虑使用更大的实例类。如果始终有大量实例处于任何等待状态，则表示可能存在要解决的瓶颈或资源争用问题。即使数据库负载未越过最大 vCPU 线，也可能会出现此问题。

启用和禁用 Performance Insights

要使用 Performance Insights，请在数据库实例中启用它。如果需要，您可以稍后将其禁用。启用和禁用 Performance Insights 不会导致停机、重新启动或故障转移。

性能详情代理占用数据库主机上有限的 CPU 和内存。当数据库负载较高时，代理将通过降低收集数据的频率来限制性能影响。

在创建集群时启用 Performance Insights

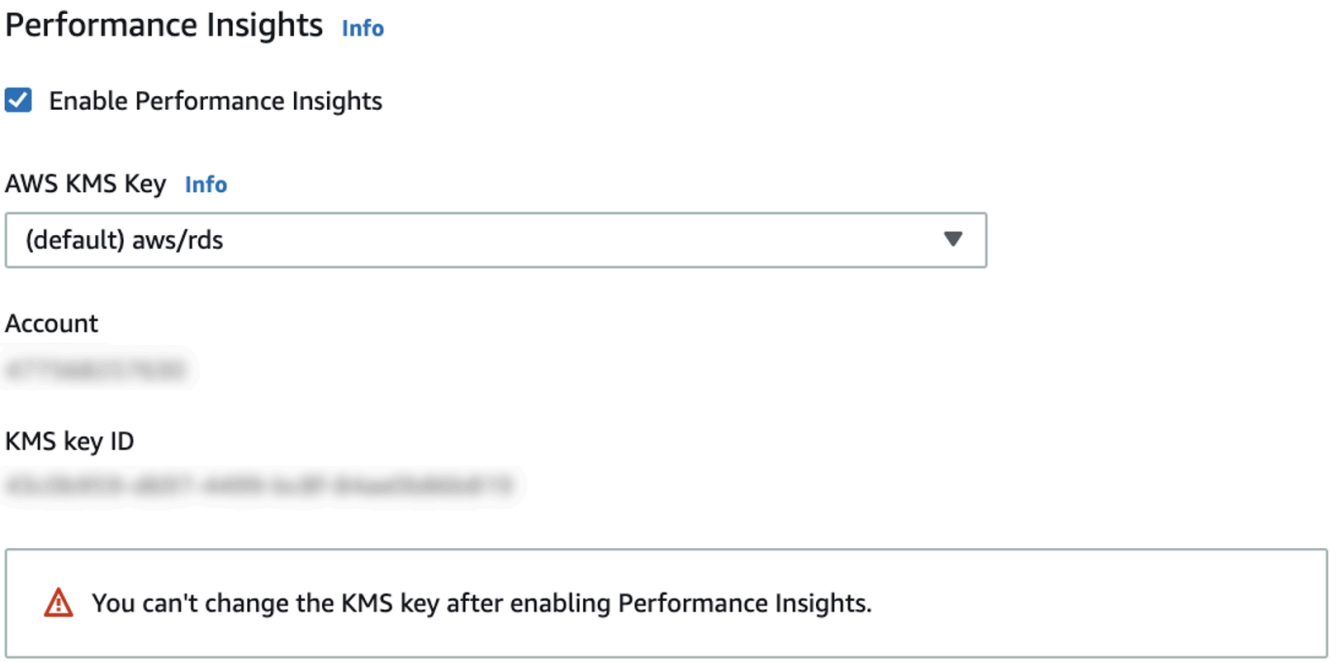
在控制台中，您可以在创建或修改新数据库实例时启用或禁用 Performance Insights。

使用 AWS 管理控制台

在控制台中，您可以在创建 Amazon DocumentDB 集群时启用 Performance Insights。在创建新 Amazon DocumentDB 集群时，通过在 Performance Insights 部分中选择启用 Performance Insights 以启用 Performance Insights。

控制台说明

1. 有关创建集群的说明，请参阅创建 Amazon DocumentDB 集群中的说明。
2. 在 Performance Insights 部分中选择启用 Performance Insights。



> **Note**
>
> Performance Insights 的数据留存期将为七天。

AWS KMS 密钥-指定您的 AWS KMS 密钥。Performance Insights 使用您的 AWS KMS 密钥加密所有潜在的敏感数据。正在传输的数据和静态数据都会被加密。有关更多信息，请参阅为 Performance Insights 配置 AWS AWS KMS 策略。

修改实例时启用和禁用

您也可以修改数据库实例以使用控制台或 AWS CLI启用或禁用 Performance Insights。

Using the AWS 管理控制台

控制台说明

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 选择集群。
3. 选择一个数据库实例，然后选择修改。
4. 在 Performance Insights 部分，选择启用 Performance Insights 或禁用 Performance Insights。

> **Note**
>
> 如果选择 “启用 Performance Insights”，则可以指定 AWS AWS KMS 密钥。Performance Insights 使用您的 AWS KMS 密钥加密所有潜在的敏感数据。正在传输的数据和静态数据都会被加密。有关更多信息，请参阅加密 Amazon DocumentDB 静态数据。

5. 选择继续。
6. 对于修改计划，选择立即应用。如果您选择在下一个计划的维护时段内应用，则您的实例将忽略此设置并立即启用 Performance Insights。
7. 选择修改实例。

Using the AWS CLI

使用`create-db-instance`或`modify-db-instance` AWS AWS CLI 命令时，您可以通过指定来启用 Performance Insights`--enable-performance-insights`，也可以通过指定将其禁用`--no-enable-performance-insights`。

以下过程介绍如何使用 AWS AWS CLI为数据库实例启用或禁用 Performance Insights。

AWS AWS CLI 指令

调用`modify-db-instance` AWS AWS CLI 命令并提供以下值：

- `--db-instance-identifer`：数据库实例的名称
- `--enable-performance-insights` 以启用，或 `--no-enable-performance-insights` 以禁用

Example

以下示例为 `sample-db-instance` 启用 Performance Insights：

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
```

为 Performance Insights 配置访问策略

要访问 Performance Insights，您必须拥有 AWS Identity and Access Management （IAM）的相应权限。您可以使用以下选项来授予访问权限：

- 将 `AmazonRDSPerformanceInsightsReadOnly` 托管式策略附加到权限集或角色。
- 创建自定义 IAM policy 并将其附加到权限集或角色。

此外，如果您在启用 Performance Insights 时指定了客户托管密钥，请确保账户中的用户对 KMS 密钥具有 `kms:Decrypt` 和 `kms:GenerateDataKey` 权限。

> **Note**
>
> 为了使用 AWS KMS 密钥进行静态加密和安全组管理，Amazon DocumentDB 利用了与 Amazon RDS 共享的操作技术。

将 AmazonRDSPerformanceInsightsReadOnly 策略附加到 IAM 委托人

AmazonRDSPerformanceInsightsReadOnly是一项 AWS托管策略，允许访问亚马逊 DocumentDB Performance Insights API 的所有只读操作。目前，此 API 中的所有操作均为只读。如果将 AmazonRDSPerformanceInsightsReadOnly 附加到权限集或角色，接收人可以使用 Performance Insights 以及其他控制台功能。

为 Performance Insights 创建自定义 IAM policy

对于没有 AmazonRDSPerformanceInsightsReadOnly 策略的用户，您可以通过创建或修改用户托管 IAM policy 来授予对 Performance Insights 的访问权限。当您将策略附加到一个权限集或角色时，接收人可以使用 Performance Insights。

创建自定义策略

1. 使用 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在导航窗格中，选择策略。
3. 选择 Create policy (创建策略)。
4. 在创建策略页面上，选择“JSON”选项卡。
5. 复制并粘贴以下文本，*us-east-1*替换为您 AWS 所在地区的*111122223333*名称和您的客户账号。

   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. 选择查看策略。
7. 为策略提供名称并可以选择提供描述，然后选择创建策略。

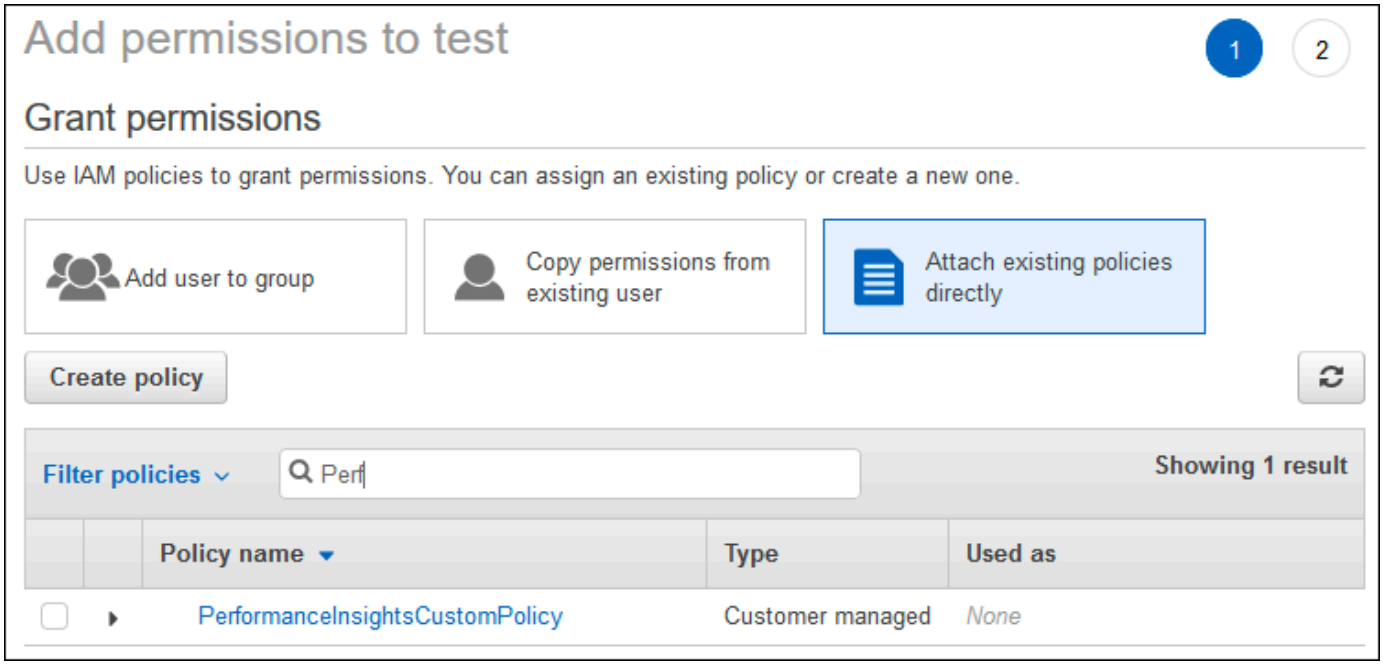
现在，可以将策略附加到权限集或角色。以下过程假设您已经有一个可用于此目的的用户。

将策略附加到用户

1. 使用 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在导航窗格中，选择 Users。
3. 从列表中选择现有用户。

> **Important**
>
> 要使用 Performance Insights，请确保除了自定义策略之外，您还有权访问 Amazon DocumentDB。例如，AmazonDocDBReadOnlyAccess预定义策略提供对 Amazon 的只读访问权限。 DocDB.For 更多信息，请参阅使用策略管理访问权限。

4. 在 Summary (摘要) 页上，选择 Add permissions (添加权限)。
5. 选择直接附加现有策略。对于 Search，键入策略名称的前几个字符，如下所示。



6. 选择策略，然后选择 Next: Review。
7. 选择 Add permissions (添加权限)。

配置一个 AWS KMS Performance Insight

Performan AWS KMS key ce Insights 使用加密敏感数据。当您通过 API 或控制台启用 Performance Insights 时，您可以选择以下选项：

- 选择默认值 AWS 托管式密钥。

  Amazon DocumentDB 使用您的新数据库实例。 AWS 托管式密钥 亚马逊 DocumentDB 会 AWS 托管式密钥 为您的 AWS 账户创建一个。您的亚马逊文档数据库 AWS 账户在每个 AWS 区域都有不同的 AWS 托管式密钥 账户。
- 选择客户托管密钥。

如果您指定一个客户托管密钥，则您账户中调用 Performance Insights API 的用户需要在 KMS 密钥具有 `kms:Decrypt` 和 `kms:GenerateDataKey` 权限。您可以通过 IAM policy 配置这些权限。但是，我们建议您通过 KMS 密钥策略来管理这些权限。有关更多信息，请参阅在 AWS KMS 中使用密钥策略。

Example

以下示例密钥策略显示了如何将语句添加到 KMS 密钥策略。这些语句可以访问 Performance Insights。根据您的使用方式 AWS KMS，您可能需要更改一些限制。在将语句添加到您的策略之前，请删除所有注释。

使用 Performance Insights 控制面板分析指标

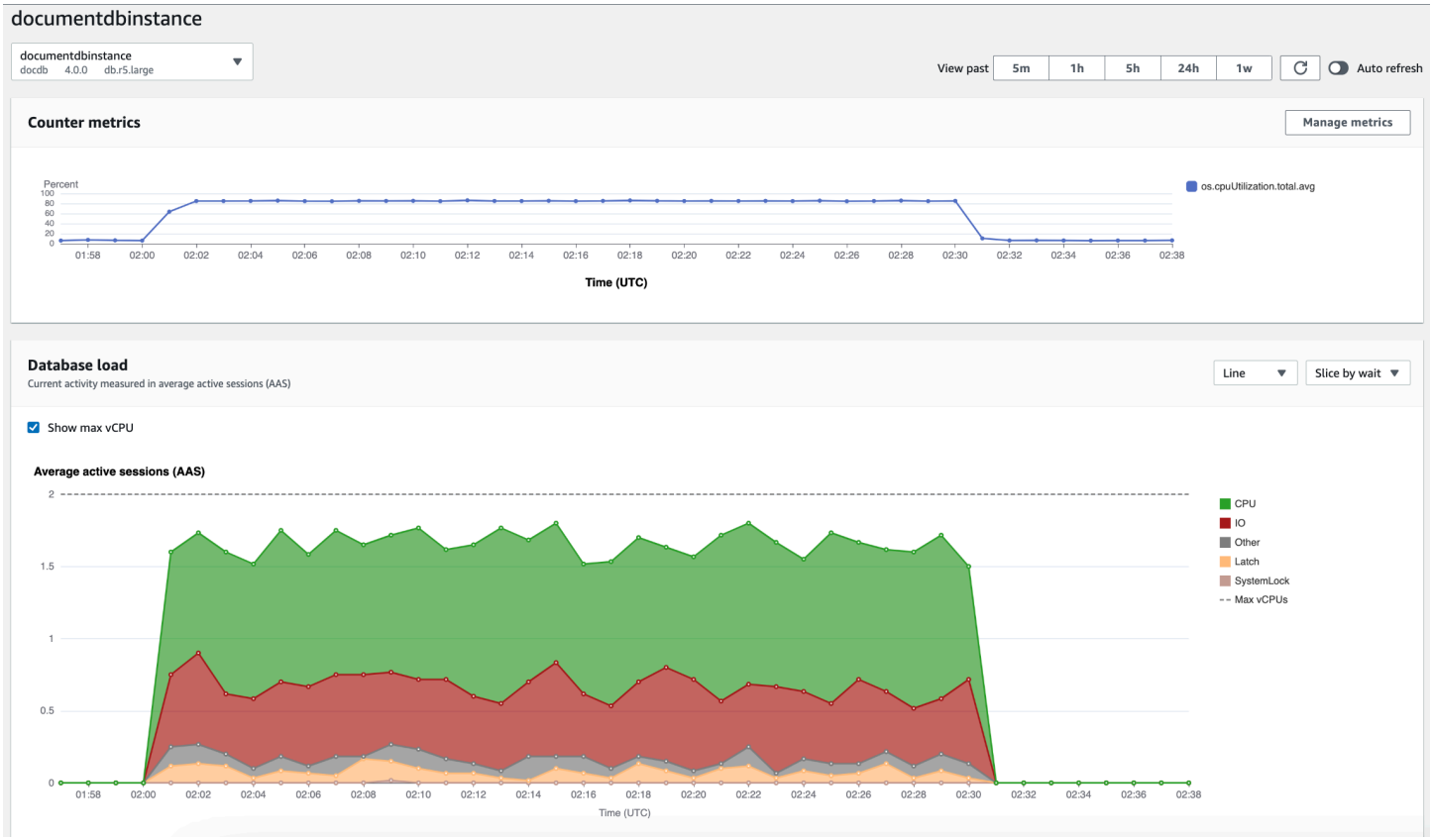
Performance Insights 控制面板包含帮助您分析和排查性能问题的数据库性能信息。在主控制面板页面上，可以查看有关数据库负载（DB 负载）的信息。您可以按维度（例如等待状态或查询）对数据库负载进行“切片”。

主题

- Performance Insights 控制面板概览
- 打开 Performance Insights 控制面板
- 通过等待状态分析数据库负载
- 主要查询选项卡概览
- 放大数据库负载图表

Performance Insights 控制面板概览

与 Performance Insights 进行交互的最简单方式即为控制面板。以下示例显示了 Amazon DocumentDB 实例的控制面板。默认情况下，Performance Insights 控制面板将显示最近一小时的数据。



控制面板分为以下几个部分：

1. 计数器指标：显示特定性能计数器指标的数据。
2. 数据库负载：显示数据库负载与最大 vCPU 线表示的数据库实例容量的比较情况。
3. 主要维度：显示对数据库负载影响最大的主要维度。这些维度包括 `waits`、`queries`、`hosts`、`databases` 和 `applications`。

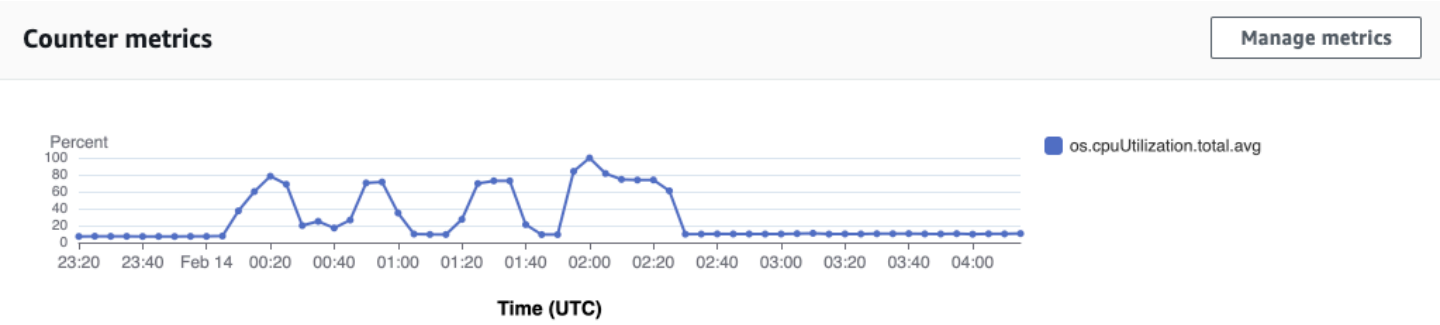
主题

- 计数器指标图表
- 数据库负载图表
- 主要维度表

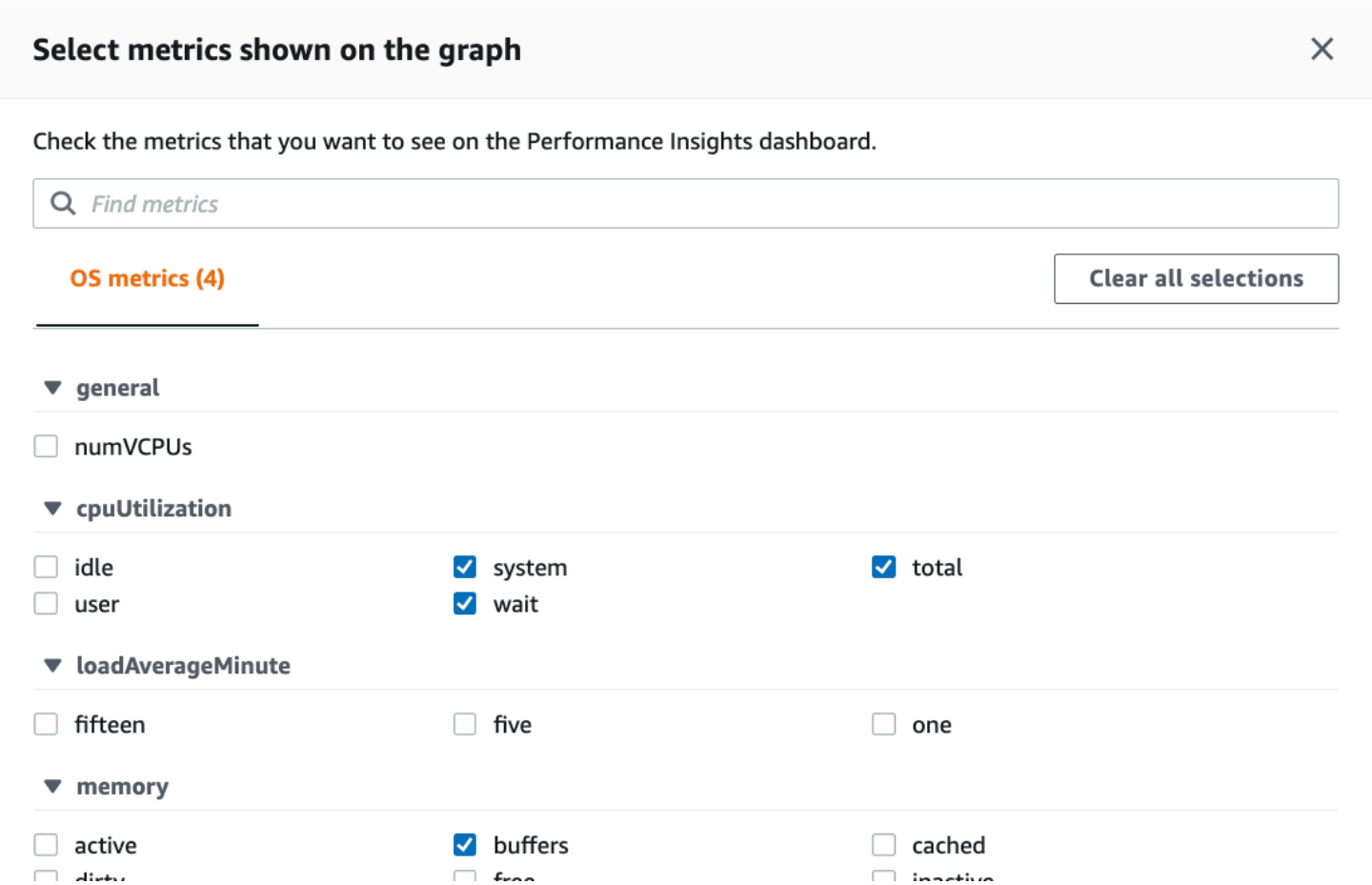
计数器指标图表

使用计数器指标，您可以自定义 Performance Insights 控制面板来包括最多 10 个其他图表。这些图表显示了所选的数十个操作系统指标。您可将此信息与数据库负载相关联，以帮助识别和分析性能问题。

计数器指标图表显示了性能计数器的数据。



要更改性能计数器，请选择管理指标。您可以选择多个 OS 指标，如以下屏幕截图所示。要查看任何指标的详细信息，请将鼠标悬停在相应指标名称上。



数据库负载图表

数据库负载图表显示数据库负载与最大 vCPU 线表示的实例容量的比较情况。预设情况下，堆叠折线图将以每单位时间的平均活动会话数表示数据库负载。数据库负载按等待状态进行切片（分组）。



按维度切片的数据库负载

您可以选择按任何受支持维度分组的活动会话显示负载。下图显示了 Amazon DocumentDB 实例的维度。

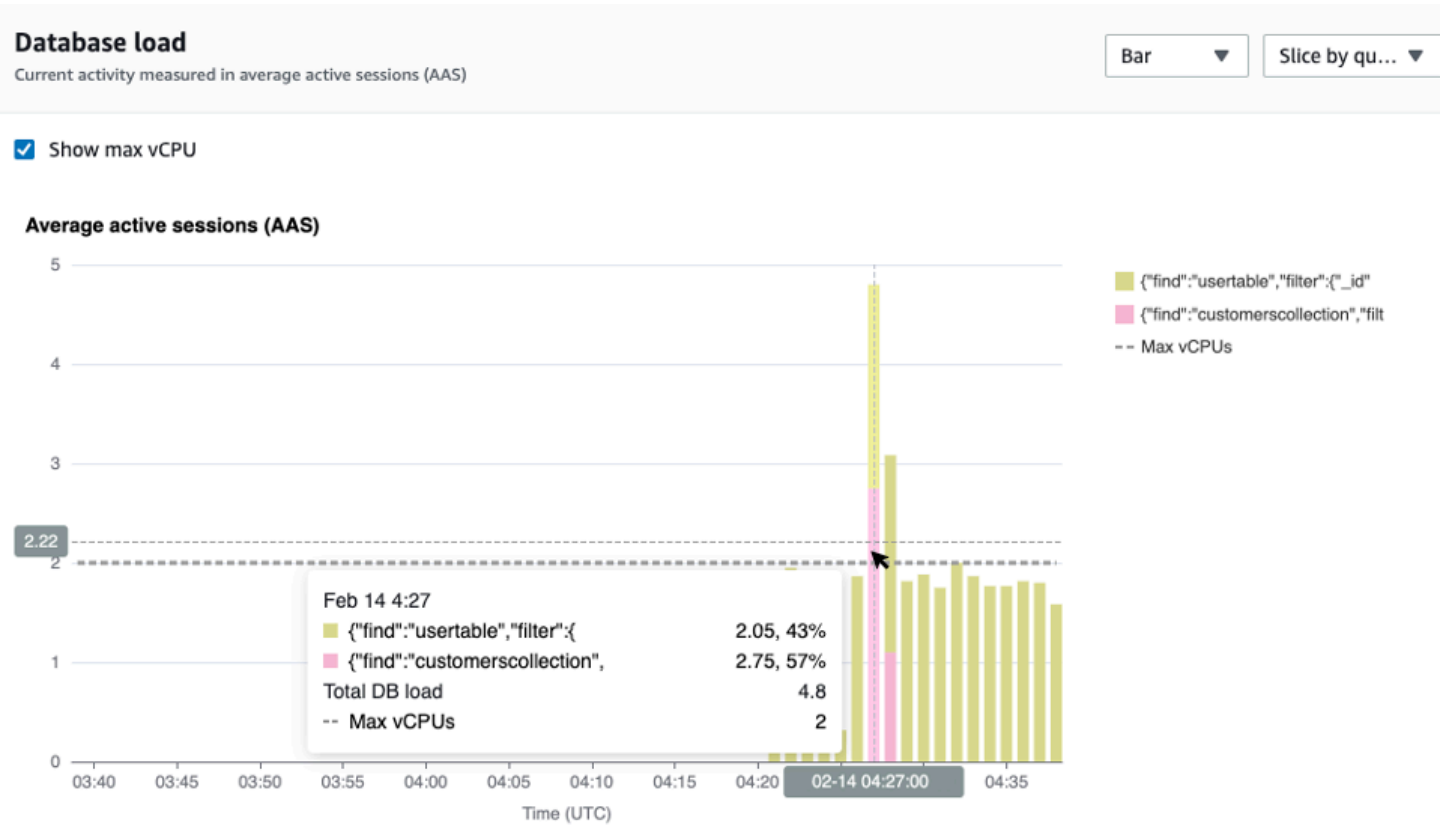


维度项目的数据库负载详细信息

要查看维度中数据库负载项目的详细信息，请将光标悬停在相应项目名称上。下图显示了查询语句的详细信息。



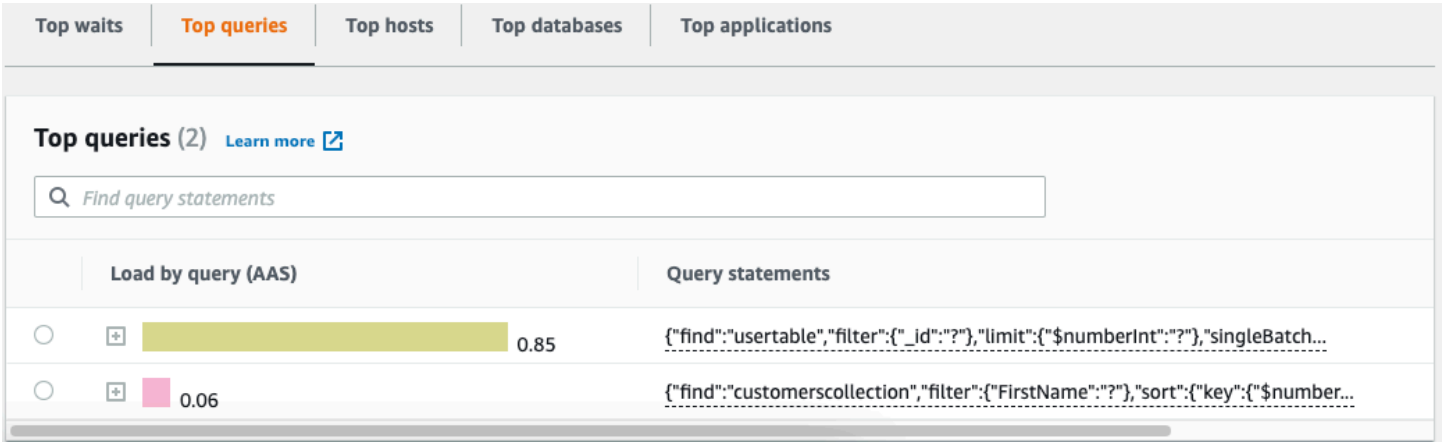
要在图例中查看任何项目在选定时间段内的详细信息，请将鼠标悬停在相应项目上。



主要维度表

主要维度表将按不同的维度切割数据库负载。维度是数据库负载不同特征的类别或“切片依据”。如果维度为查询，则主要查询显示了对数据库负载影响最大的查询语句。

请选择以下任何一个维度选项卡。



下表简要说明了每个选项卡。

选项卡

数据等待端正在等待的事件

当前正

选项卡

查询

行的查询语句

前要求

接客户端的主机IP和端口

选项卡

客户端

连接的数据库

应用程序

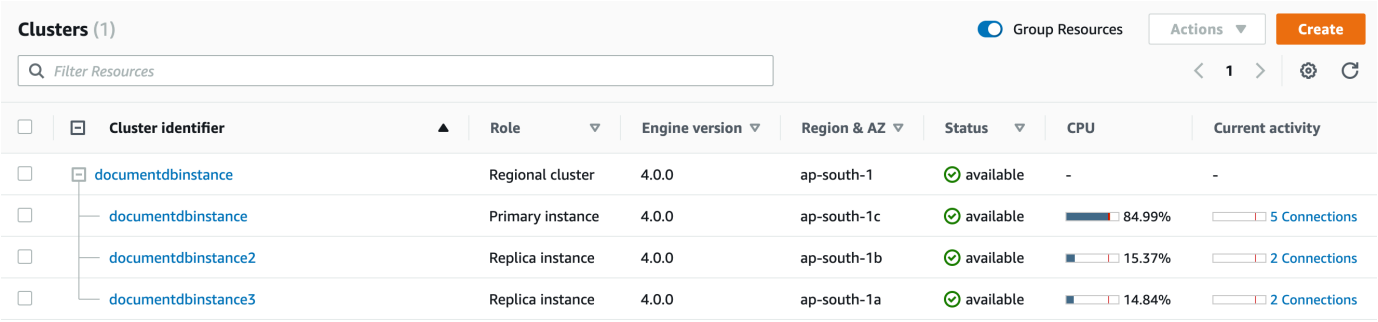
要了解如何使用主要查询选项卡分析查询，请参阅 主要查询选项卡概览。

打开 Performance Insights 控制面板

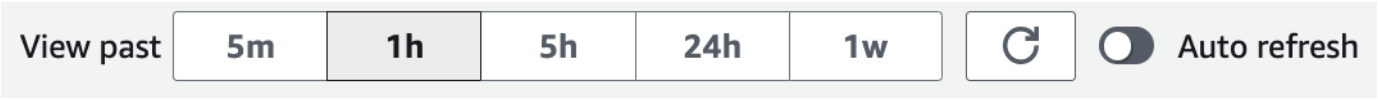
要查看 Performance Insights 控制面板 AWS 管理控制台，请使用以下步骤：

1. 在上打开 Performance Insight https://console.aws.amazon.com/docdb/s 控制台
2. 选择一个数据库实例。将为该 Amazon DocumentDB 实例显示 Performance Insights 控制面板。

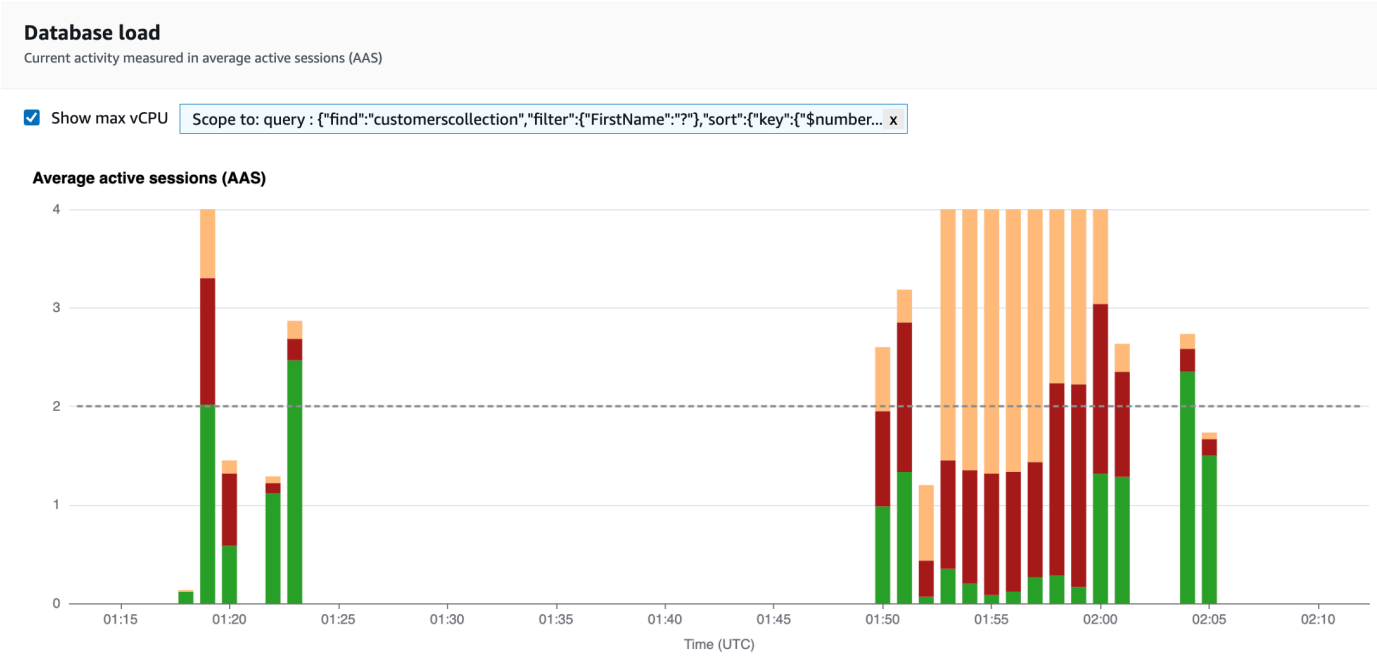
   对于启用 Performance Insights 的 Amazon DocumentDB 实例，您还可以通过选择实例列表中的会话项目来访问控制面板。在当前活动下，会话项目显示在过去五分钟内平均活跃会话中的数据库负载。条形图显示负载量。当条形图为空时，实例处于空闲状态。随着负载的增加，条形图会以蓝色填充。当负载超过实例类上的虚拟 CPU (vCPU) 数量时，条形图变为红色，表示可能出现瓶颈。



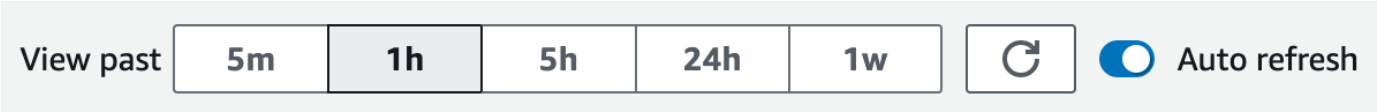
3. （可选）通过选择右上角的按钮来选择不同的时间间隔。例如，要将间隔更改为 1 小时，请选择 1 小时。



   在以下屏幕截图中，数据库负载间隔为 1 小时。



4. 要自动刷新数据，请启用自动刷新。



Performance Insights 控制面板自动刷新新的数据。刷新速率取决于所显示的数据量：

- 5 分钟则每 5 秒刷新一次。
- 1 小时则每分钟刷新一次。
- 5 小时则每分钟刷新一次。
- 每 5 分钟刷新 24 小时一次。
- 每小时刷新一周一次。

通过等待状态分析数据库负载

如果数据库负载（DB 负载）图表显示了一个瓶颈，您可以找出负载的来源。为此，请查看数据库负载图表下方的主要负载项目。选择特定项目 (如查询或应用) 以深入了解该项目并查看有关该项目的详细信息。

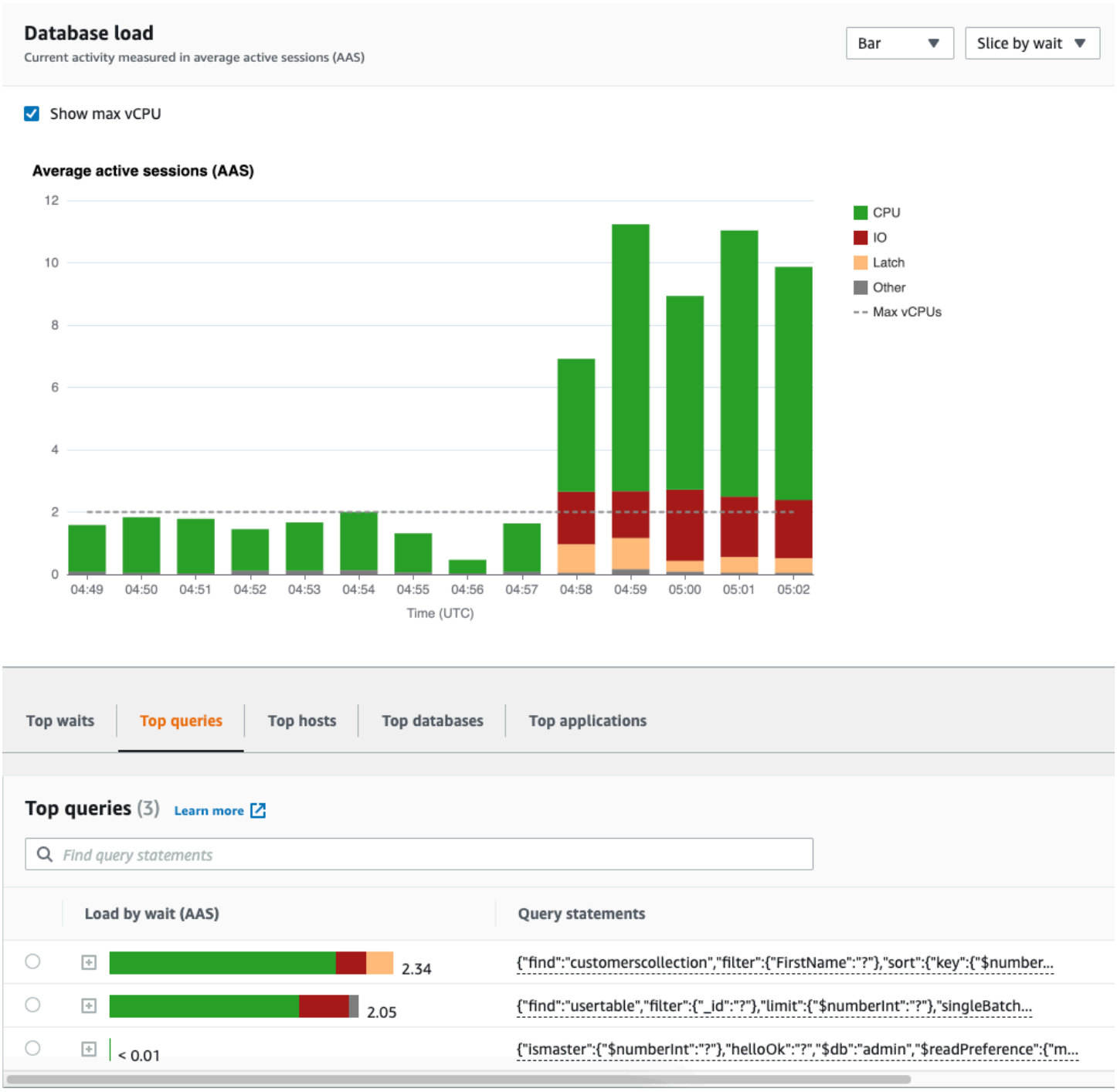
按等待状态和主要查询分组的数据库负载通常可以提供对性能问题的最深入了解。按等待状态分组的数据库负载显示了数据库中是否存在任何资源瓶颈或并发瓶颈。在这种情况下，“主要负载项目”表的主要查询选项卡显示了增大该负载的查询。

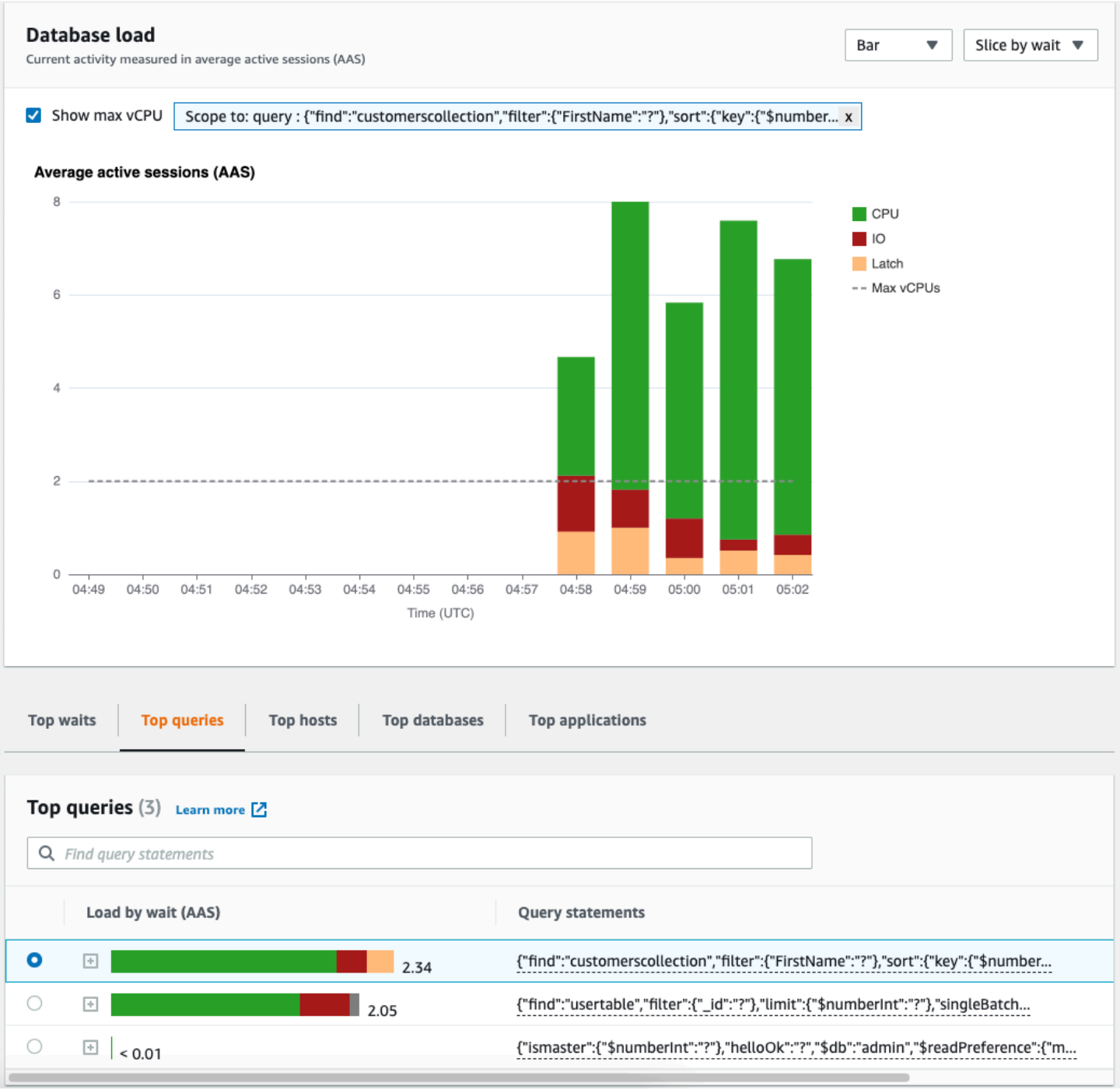
诊断性能问题的典型工作流程如下：

1. 查看数据库负载图表并了解是否存在数据库负载的事件越过了 Max CPU 线。
2. 如果有，请查看数据库负载图表并确定负主要责任的等待状态。
3. 通过以下方式确定导致负载的摘要查询：查看“主要负载项目”表上的主要查询选项卡中的哪个查询对于导致这些等待状态所起的作用最大。可通过 按等待状态排列的负载 (AAS) 列加以识别。
4. 在主要查询选项卡中选择这些摘要查询之一以展开它并查看它包含的子查询。

您还可以分别选择热门主机或热门应用程序来查看哪些主机或应用程序造成的负载最大。应用程序名称在 Amazon DocumentDB 实例的连接字符串中指定。Unknown 表示未指定应用程序字段。

例如，在下面的控制面板中，CPU 等待状态占大部分数据库负载。选择主要查询下的排名靠前的查询会将数据库负载图表的范围限定为重点关注选择查询贡献的最大负载。


Database load
Current activity measured in average active sessions (AAS)
Bar
Slice by wait
Show max vCPU
Average active sessions (AAS)
12
10
8
6
4
2
0
04:49
04:50
04:51
04:52
04:53
04:54
04:55
04:56
04:57
04:58
04:59
05:00
05:01
05:02
Time (UTC)
CPU
IO
Latch
Other
Max vCPUs
Top waits
Top queries
Top hosts
Top databases
Top applications
Top queries (3)
Learn more
Find query statements
Load by wait (AAS)
Query statements
2.34
{"find":"customerscollection","filter":{"FirstName":"?"},"sort":{"key":{"$number...
2.05
{"find":"usertable","filter":{"_id":"?"},"limit":{"$numberInt":"?"},"singleBatch...
< 0.01
{"ismaster":{"$numberInt":"?"},"helloOk":"?","$db":"admin","$readPreference":{"m...



主要查询选项卡概览

原定设置情况下，主要查询选项卡将显示对数据库负载影响最大的 25 个 SQL 查询。您可以分析查询文本，帮助调整您的查询。

主题

- 查询摘要

- 按等待状态排列的负载 (AAS)
- 查看详细的查询信息
- 访问语句查询文本
- 查看和下载语句查询文本

查询摘要

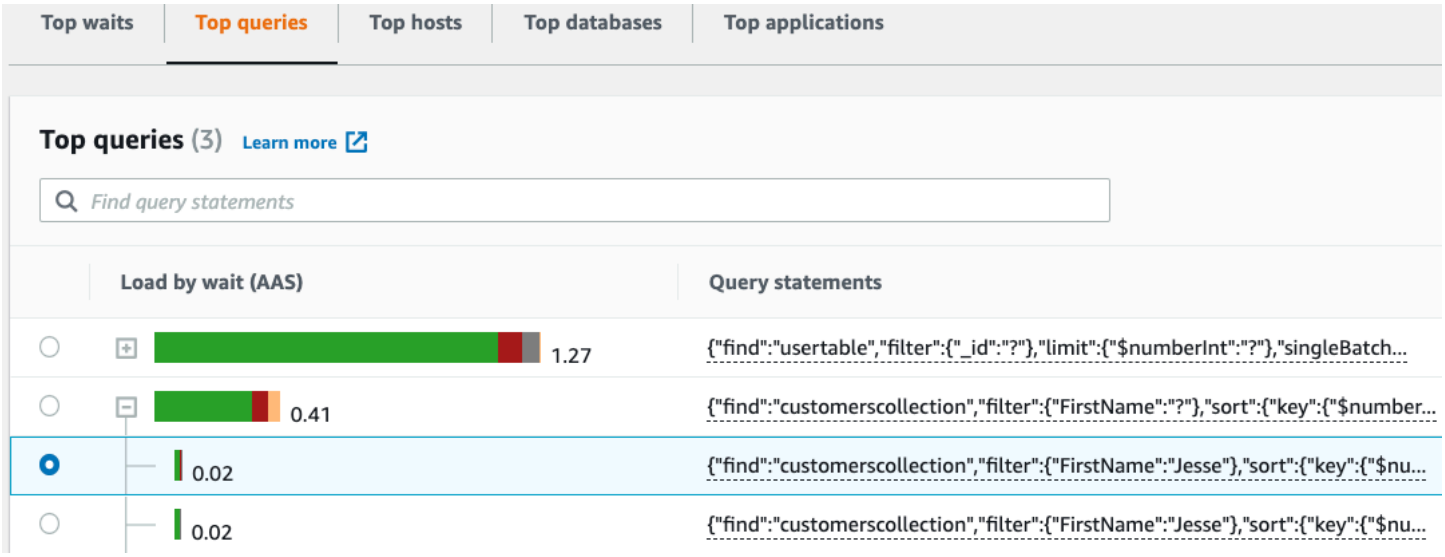
查询摘要是多个结构上相似但可能具有不同文本值的实际查询的组合。摘要用问号替换硬编码值。例如，查询摘要可能如下所示：

```
{"find":"customerscollection","filter":{"FirstName":"?"},"sort":{"key":
{"$numberInt":"?"}},"limit":{"$numberInt":"?"}}
```

此摘要可能包含以下子查询：

```
{"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"}}
```

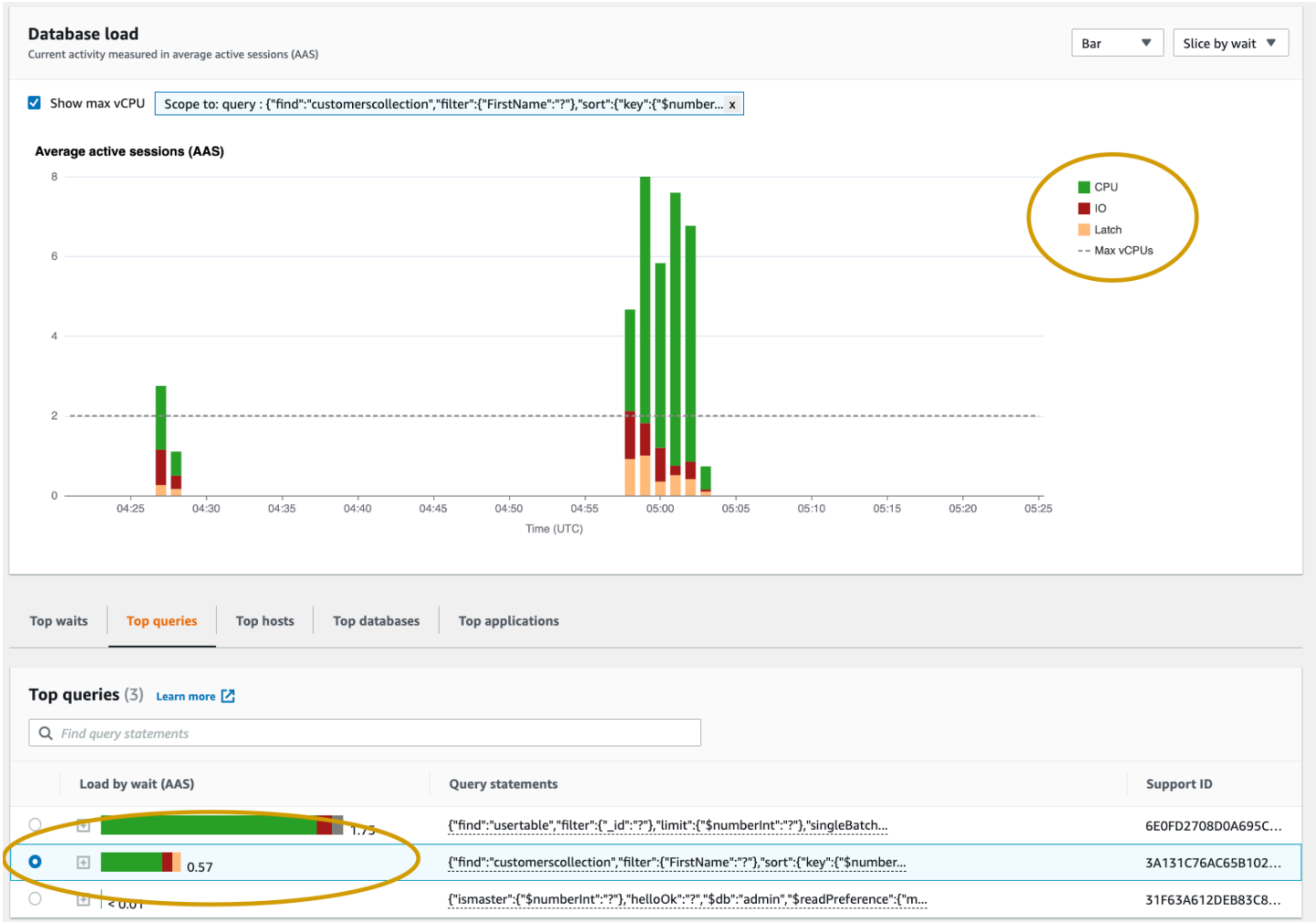
要查看摘要中的文字查询语句，请选择查询，然后选择加号 (+)。在下面的屏幕截图中，选定的查询是摘要。



> **Note**
>
> 查询摘要将相似的查询语句进行分组，但不会编辑敏感信息。

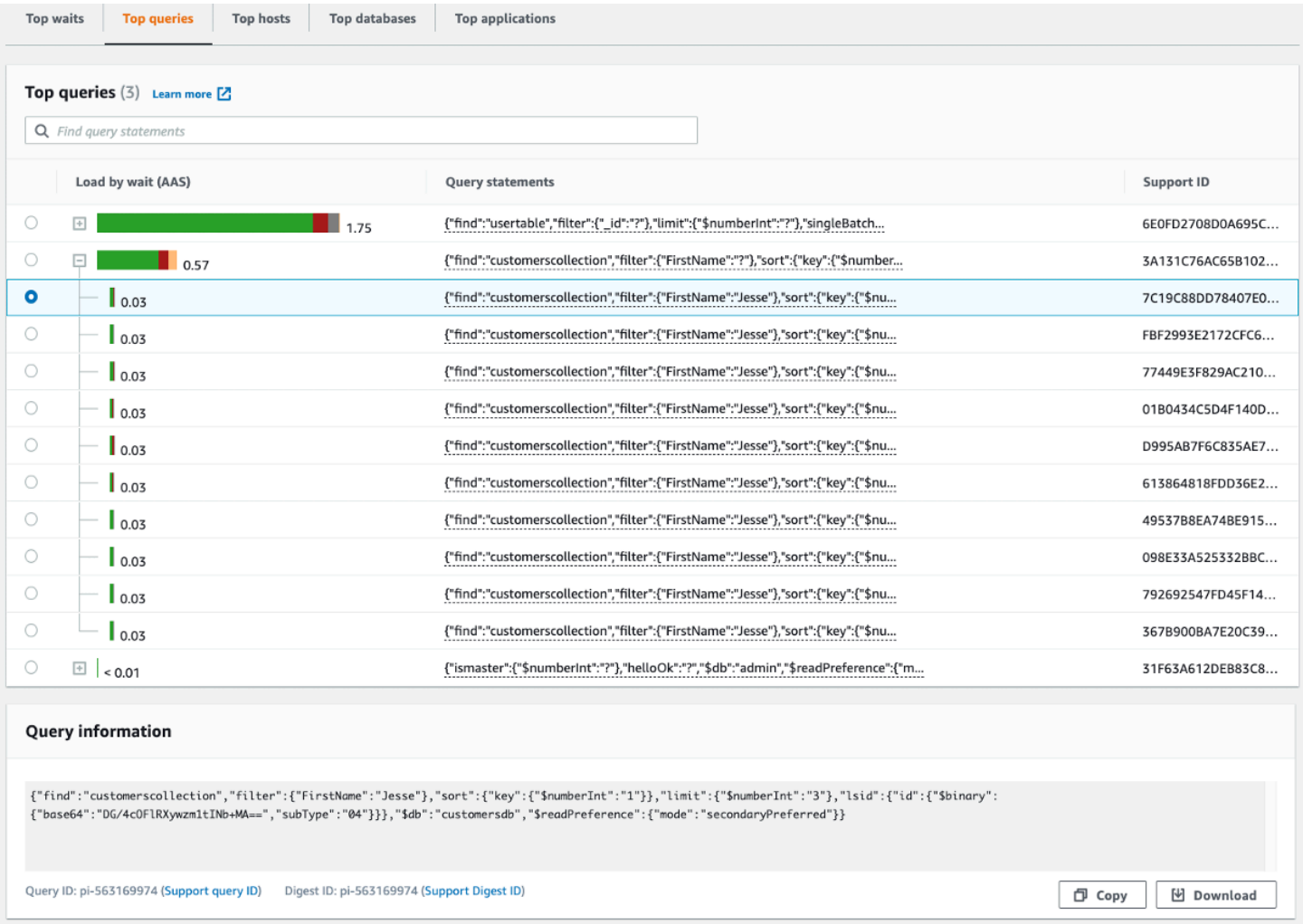
按等待状态排列的负载 (AAS)

在主要查询中，按等待状态排列的负载 (AAS) 列说明了与每个主要负载项目关联的数据库负载的百分比。此列按当前在数据库负载图表中选择的分组方式反映该项目的负载。例如，您可以按等待状态对数据库负载图表进行分组。在这种情况下，系统将对 DB Load by Waits (按等待状态排列的数据库负载) 栏进行大小调整、分段和颜色编码，以显示该查询在导致给定等待状态方面所起的作用大小，它还会显示哪些等待状态正在影响选定的查询。



查看详细的查询信息

在主要查询表中，您可以打开一条摘要语句以查看其信息。信息将显示在底部窗格中。



以下类型的标识符 (ID) 与查询语句相关联：

1. 支持查询 ID：查询 ID 的哈希值。此值仅用于在使用 Support 时引用 AWS 查询 ID。 AWS Support 无法访问您的实际查询 ID 和查询文本。
2. 支持摘要 ID：摘要 ID 的哈希值。此值仅用于在使用 Support 时引用摘要 ID。 AWS AWS Support 无法访问您的实际摘要 ID 和查询文本。

访问语句查询文本

原定设置情况下，主要查询表中的每行为每条查询语句显示 500 字节的查询文本。当摘要语句超过 500 字节时，可通过在 Performance Insights 控制面板中打开该语句来查看更多文本。在这种情况下，显示的查询的最大长度为 1 KB。如果查看完整的查询语句，也可以选择下载。

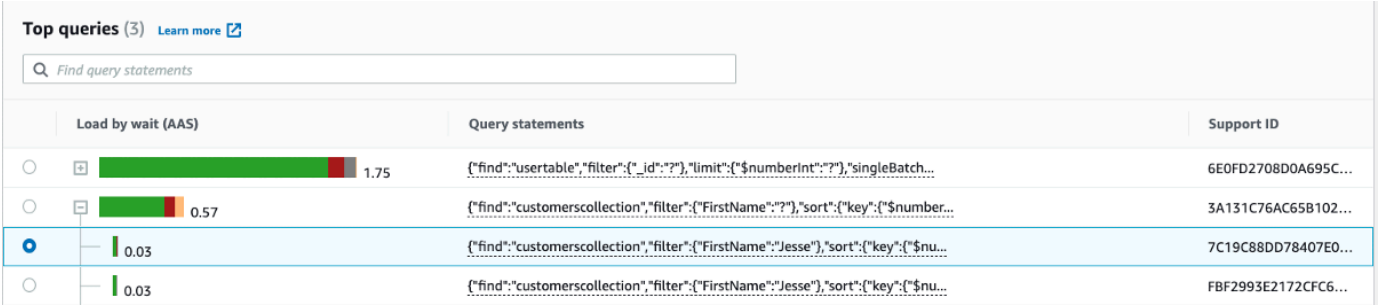
查看和下载语句查询文本

在 Performance Insights 控制面板中，您可以查看或下载查询文本。

在 Performance Insights 控制面板中查看更多查询文本

1. 打开 Amazon DocumentDB 控制台，网址为：https://console.aws.amazon.com/docdb/
2. 在导航窗格中，选择性能详情。
3. 选择一个数据库实例。将为该数据库实例显示 Performance Insights 控制面板。

   具有大于 500 字节的文本的查询语句如下图所示。



4. 检查查询信息部分以查看更多的查询文本。



Performance Insights 控制面板可以为每个完整的查询语句最多显示 1 KB。

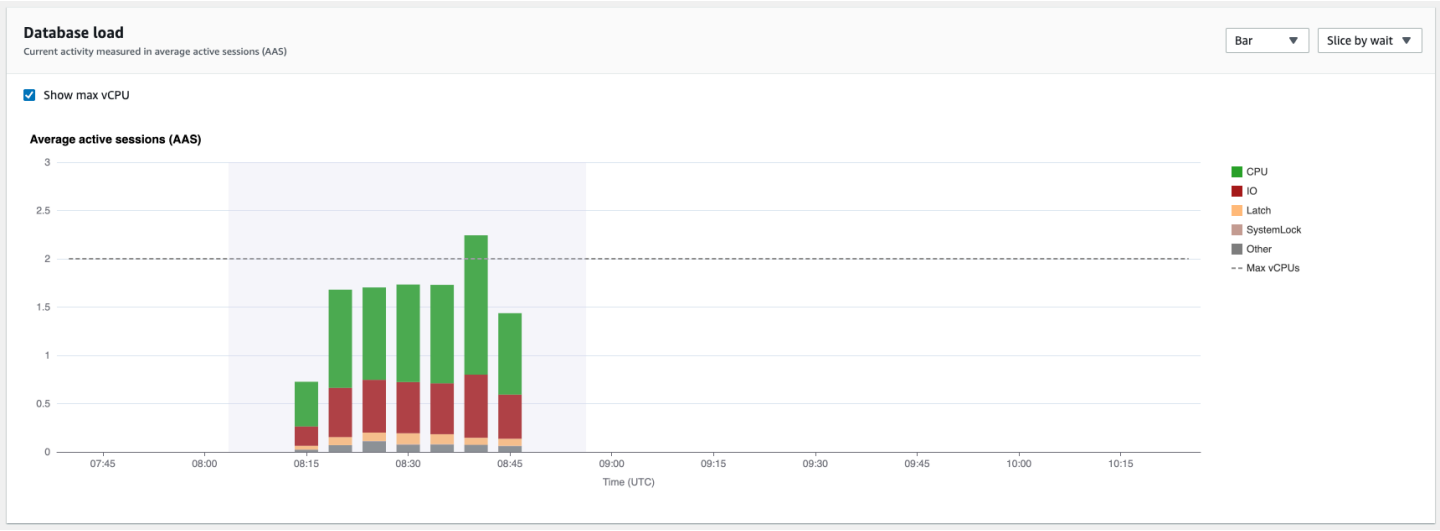
> **Note**
>
> 要复制或下载查询语句，请禁用弹出窗口阻止程序。

放大数据库负载图表

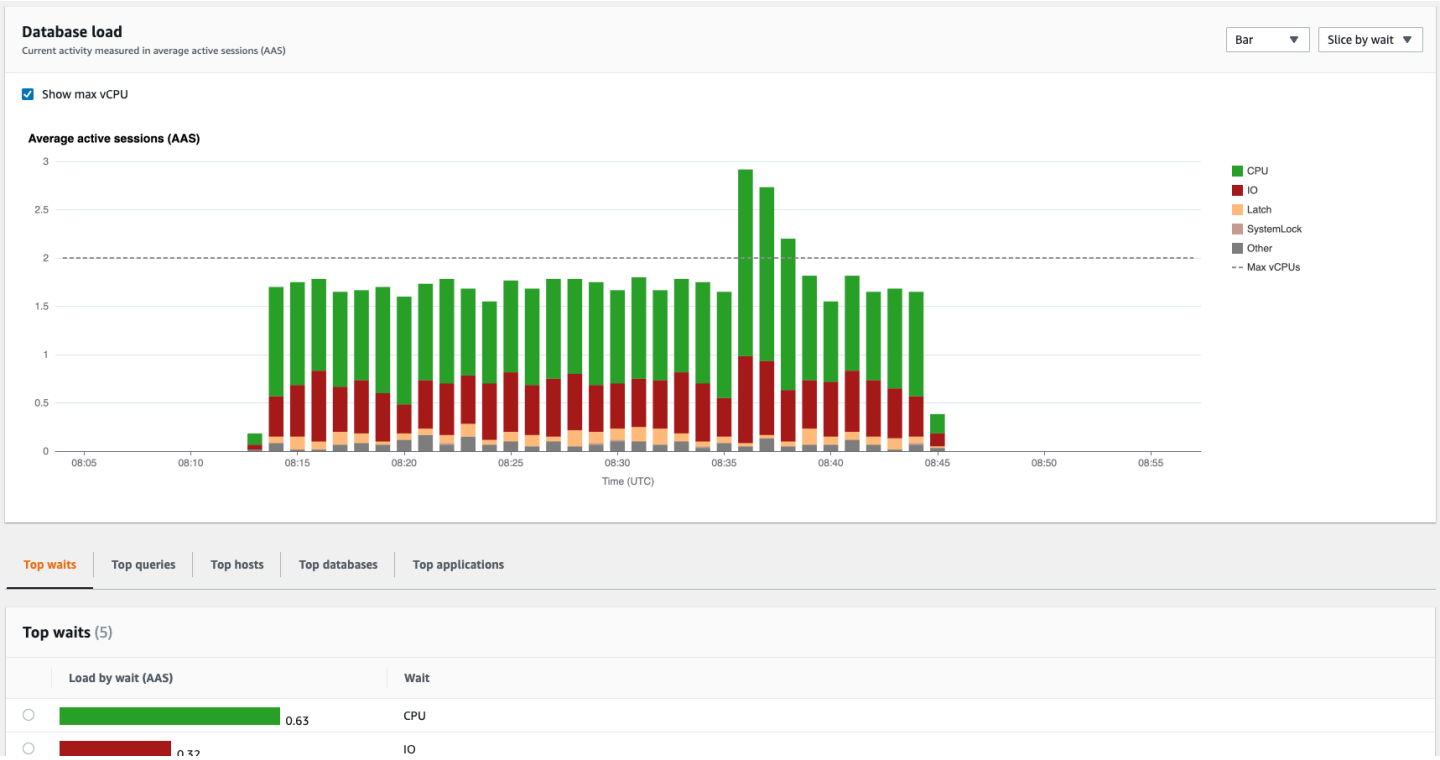
您可以使用 Performance Insights 用户界面的其他功能来帮助分析性能数据。

Click-and-Drag 放大

在 Performance Insights 界面中，您可以选择负载图表的一小部分并放大细节。



要放大负载图表的一部分，请选择开始时间并拖动到所需时间段的结尾。执行该操作时，所选区域将突出显示。释放鼠标时，负载图表上的所选区域将放大，并重新计算主要项目表。



使用 Performance Insights API 检索指标

启用 Performance Insights 后，API 将提供实例性能的可见性。Ama CloudWatch zon Logs 为 AWS 服务的销售监控指标提供了权威来源。

Performance Insights 提供了按平均活动会话 (AAS) 衡量的数据库负载的特定于域的视图。对 API 使用者而言，此指标看起来像是二维时间序列数据集。数据的时间维度提供所查询时间范围的每个时间点的数据库负载数据。每个时间点将分解与所请求维度相关的整体负载，如相应时间点测量的 `Query`、`Wait-state`、`Application` 或 `Host`。

Amazon DocumentDB Performance Insights 用于监控您的 Amazon DocumentDB 数据库实例，使您可以分析数据库性能和排查数据库性能问题。查看 Performance Insights 数据的一种方法是在 AWS 管理控制台中。Performance Insights 还提供公有 API，以便您可以查询自己的数据。您可以使用 API 来执行以下操作：

- 将数据卸载到数据库中
- 将 Performance Insights 数据添加到现有监控控制面板
- 构建监控工具

要使用 Performance Insights API，请在您的 Amazon DocumentDB 实例之一上启用 Performance Insights。有关启用 Performance Insights 的信息，请参阅 启用和禁用 Performance Insights。有关性能详情 API 的更多信息，请参阅性能详情 API 参考。

Performance Insights API 提供以下操作。

Performance Insights 操作	AWS CLI 命令	说明
`DescribeDimensionKeys`	`aws pi describe-dimension-keys`	对于特定的时间段，检索指标的前 N 个维度键。
`GetDimensionKeyDetails`	`aws pi get-dimension-key-details`	检索数据库实例或数据来源的指定维度组的属性。例如，如果您指定了查询 ID，并且有维度详细信息，则 `GetDimensionKeyDetails` 将检索与此 ID 关联的维度 `db.query.statement` 的全文。此操作很有用，因为 `GetResourceMetrics` 和 `DescribeDimensionKeys` 不支持检索大型查询语句文本。

Performance Insights 操作	AWS CLI 命令	说明
GetResourceMetadata	aws pi get-resource-metadata	检索不同功能的元数据。例如，元数据可以表明特定数据库实例上的某个功能已打开或关闭。
GetResourceMetrics	aws pi get-resource-metrics	检索一组数据来源在一段时间内的 Performance Insights 指标。您可以提供特定维度组和维度，并为每个组提供聚合和筛选条件。
ListAvailableResourceDimensions	aws pi list-available-resource-dimensions	检索特定实例上每个特定指标类型可查询的维度。
ListAvailableResourceMetrics	aws pi list-available-resource-metrics	检索指定指标类型的所有可用指标，指定数据库实例可用该指标进行查询。

主题

- AWS CLI 获取性能见解
- 检索时间序列指标
- AWS CLI 性能 Insights 的示例

AWS CLI 获取性能见解

您可以使用 AWS CLI查看 Performance Insights 数据。可以通过在命令行上输入以下内容来查看 Performance Insights 的 AWS CLI 命令的帮助。

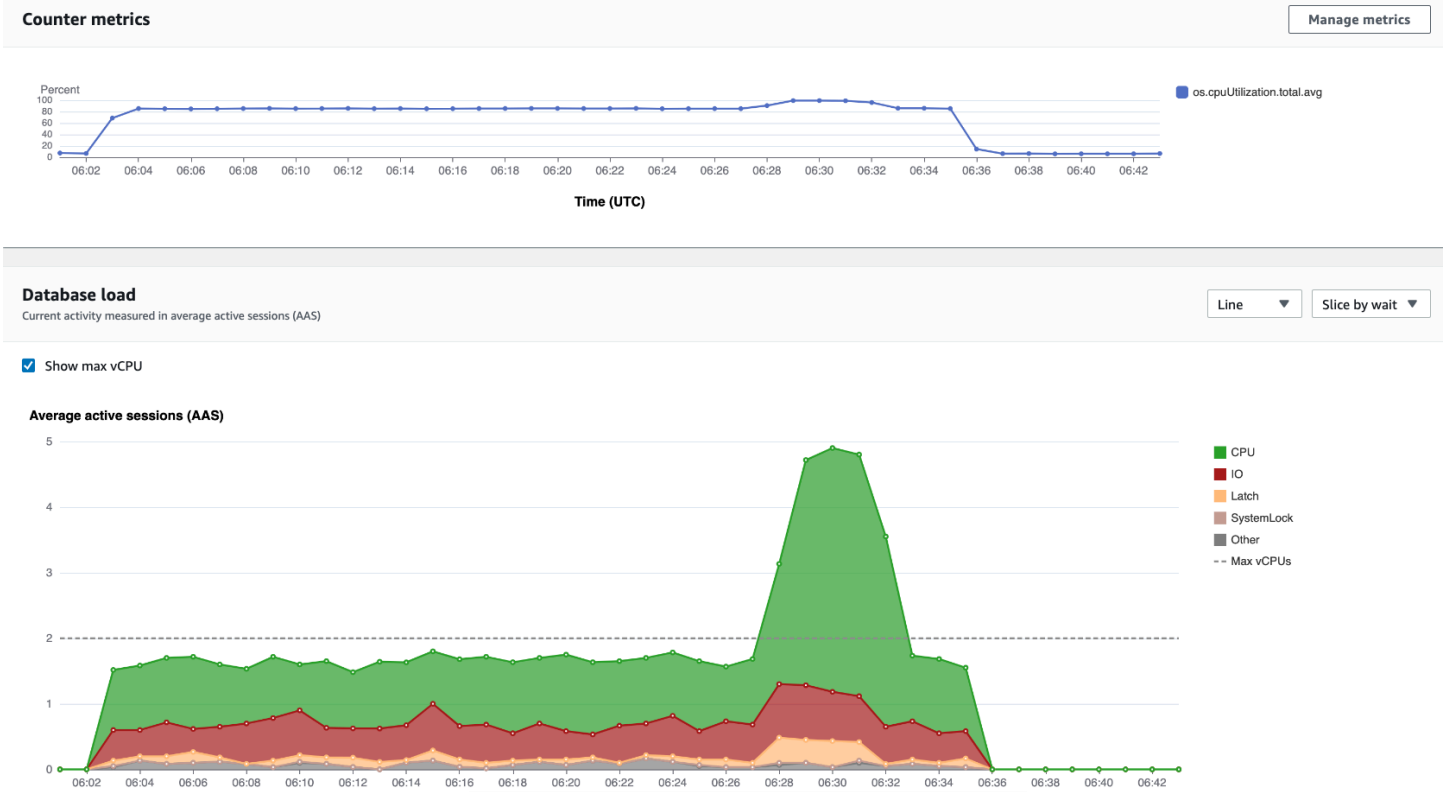
```
aws pi help
```

如果您尚未 AWS CLI 安装，请参阅《AWS CLI 用户指南》中的 “安装 AWS 命令行界面”，了解有关安装命令行界面的信息。

检索时间序列指标

GetResourceMetrics 操作从 Performance Insights 数据中检索一个或多个时间序列指标。GetResourceMetrics 需要指标和时间段，并返回包含数据点列表的响应。

例如， AWS 管理控制台 用于GetResourceMetrics填充 “计数器指标” 图表和 “数据库负载” 图表，如下图所示。



GetResourceMetrics 返回的所有指标都是标准的时间序列指标，但 db.load 除外。此指标显示在 Database Load (数据库负载) 图表中。db.load 指标不同于其他时间序列指标，因为您可以将它分为称为维度的子组件。在上图中，按组成 db.load 的等待状态对 db.load 进行细分和分组。

> **Note**
>
> GetResourceMetrics 也可以返回 db.sampleload 指标，但 db.load 指标在大多数情况下是合适的。

有关 GetResourceMetrics 返回的计数器指标的信息，请参阅Performance Insights 的计数器指标。

指标支持以下计算：

- 平均值：指标在一段时间内的平均值。在指标名称后面附加 .avg。
- 最小值：指标在一段时间内的最小值。在指标名称后面附加 .min。
- 最大值：指标在一段时间内的最大值。在指标名称后面附加 .max。
- 总计：指标值在一段时间内的总计。在指标名称后面附加 .sum。
- 样本数：在一段时间内收集指标的次数。在指标名称后面附加 .sample_count。

例如，假定在 300 秒（5 分钟）时段内收集指标，并且每分钟收集一次指标。各分钟的值为 1、2、3、4 和 5。在本例中，返回以下计算：

- 平均值：3
- 最小值：1
- 最大值：5
- 总计：15
- 样本数：5

有关使用该get-resource-metrics AWS CLI 命令的信息，请参见get-resource-metrics。

对于 --metric-queries 选项，请指定一个或多个要获取其结果的查询。每个查询包括必需的 Metric 和可选的 GroupBy 和 Filter 参数。以下是 --metric-queries 选项规范的示例。

```
{
   "Metric": "string",
   "GroupBy": {
     "Group": "string",
     "Dimensions": ["string", ...],
     "Limit": integer
   },
   "Filter": {"string": "string"
     ...}
```

AWS CLI 性能 Insights 的示例

以下示例说明了如何使用获取 Performanc AWS CLI e Insights。

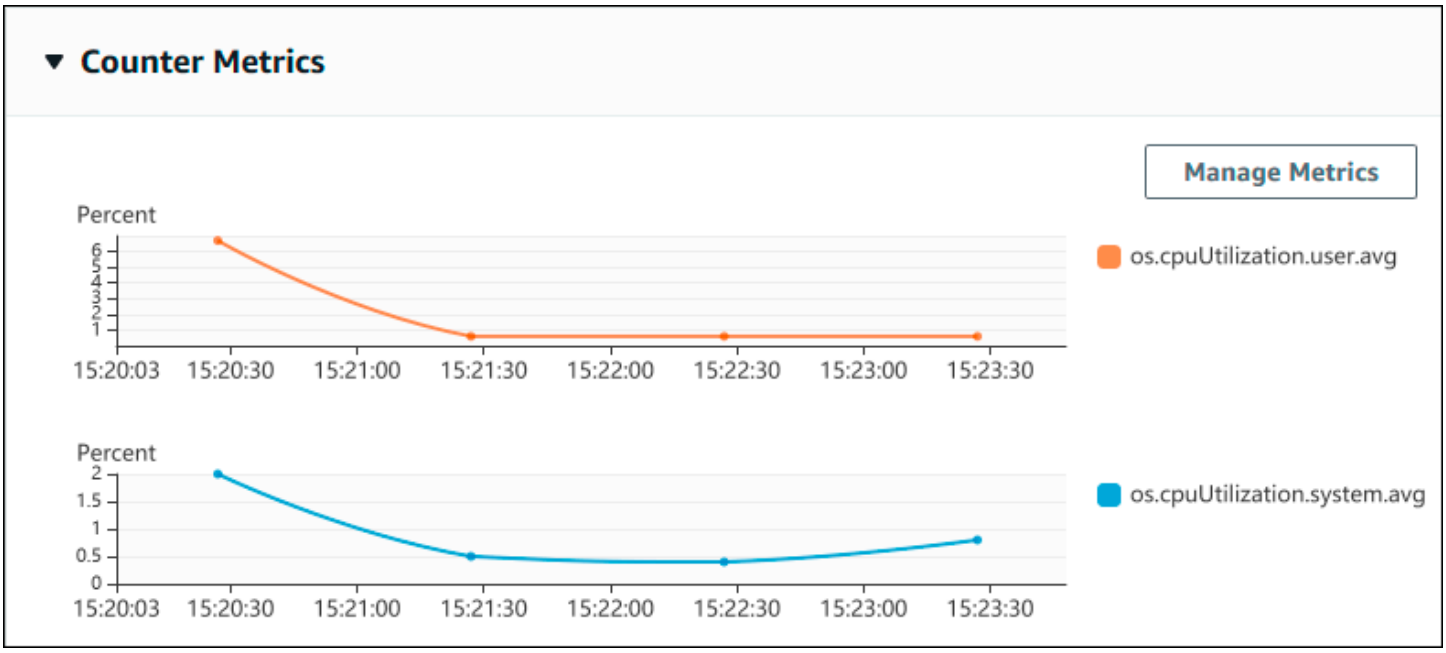
主题

- 检索计数器指标
- 检索首要等待状态的数据库负载平均值

- 检索主要查询的数据库负载平均值
- 检索按查询筛选的数据库负载平均值

检索计数器指标

以下屏幕截图显示 AWS 管理控制台中的两个计数器指标图表。



以下示例显示如何收集 AWS 管理控制台 用于生成两个计数器指标图表的相同数据。

对于 Linux、macOS 或 Unix：

```
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"}]'
```

对于 Windows：

```
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"}]'
```

还可以通过为 `--metrics-query` 选项指定文件来使命令更易于读取。以下示例为该选项使用名为 query.json 的文件。此文件具有以下内容。

```
[
    {
        "Metric": "os.cpuUtilization.user.avg"
    },
    {
        "Metric": "os.cpuUtilization.idle.avg"
    }
]
```

运行以下命令来使用此文件。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

上一个示例为各选项指定了以下值：

- `--service-type`：DOCDB 适用于 Amazon DocumentDB

- `--identifier`：数据库实例的资源 ID
- `--start-time` 和 `--end-time`：要查询的期间的 ISO 8601 DateTime 值，支持多种格式

它查询一小时时间范围：

- `--period-in-seconds`：对于每分钟查询来说为 60
- `--metric-queries`：两个查询的数组，每个查询只用于一个指标。

  指标名称使用点在有用的类别中分类指标，最后一个元素是函数。在示例中，对于每个查询来说，此函数是 avg。与 Amazon 一样 CloudWatch，支持的函数有`min`、`max`、`total`、和`avg`。

响应类似于以下内容。

```
{
    "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
```

响应具有 Identifier、AlignedStartTime 和 AlignedEndTime。但 --period-in-seconds 值为 60，开始和结束时间已与分钟对齐。如果 --period-in-seconds 为 3600，则开始和结束时间已与小时对齐。

响应中的 MetricList 具有许多条目，每个条目具有 Key 和 DataPoints 条目。每个 DataPoint 具有 Timestamp 和 Value。每个 Datapoints 列表具有 60 个数据点，因为查询针对一小时内的每分钟数据，具有 Timestamp1/Minute1、Timestamp2/Minute2 等，一直到 Timestamp60/Minute60。

因为查询用于两个不同的计数器指标，响应 MetricList 中有两个元素。

检索首要等待状态的数据库负载平均值

以下示例与 AWS 管理控制台 用于生成堆叠面积折线图的查询相同。此示例检索按前七个等待状态划分负载的最后一个小时的 db.load.avg。命令与 检索计数器指标 中的命令相同。不过，query.json 文件具有以下内容。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_state", "Limit": 7 }
    }
]
```

运行以下命令。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此示例指定指标 db.load.avg 和前七个等待状态的 GroupBy。有关此示例有效值的详细信息，请参阅 Performance Insights API 参考DimensionGroup中的。

响应类似于以下内容。

```
{
    "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
```

在此响应中，MetricList 中有三个条目。有一个有关总 db.load.avg 的条目，还有三个条目，其中每个条目关于按前三个等待状态之一划分的 db.load.avg。由于具有分组维度（与第一个示例不同），所以必须具有一个用于每个指标分组的键。不能像在基本计数器指标使用案例中那样每个指标只有一个键。

检索主要查询的数据库负载平均值

以下示例按前 10 个查询语句对 db.wait_state 进行分组。有两个不同的查询语句组：

- db.query：完整的查询语句，例如 {"find":"customers","filter": {"FirstName":"Jesse"},"sort":{"key":{"$numberInt":"1"}}}
- db.query_tokenized：令牌化的查询语句，例如 {"find":"customers","filter": {"FirstName":"?"},"sort":{"key":{"$numberInt":"?"}},"limit": {"$numberInt":"?"}}

在分析数据库性能时，将仅参数不同的查询语句视为一个逻辑项目很有用。因此，您在查询时可以使用 db.query_tokenized。不过，尤其在您对 explain() 感兴趣时，查看带参数的完整查询语句会更有用。令牌化和完整查询之间存在父-子关系，多个完整查询（子级）分组在同一令牌化查询（父级）下。

此示例中的命令类似于 检索首要等待状态的数据库负载平均值 中的命令。不过，query.json 文件具有以下内容。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.query_tokenized", "Limit": 10 }
    }
]
```

下面的示例使用了 db.query_tokenized。

对于 Linux、macOS 或 Unix：

```
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
```

对于 Windows：

```
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
```

此示例查询 1 小时，以秒为单位的期间为一分钟。

此示例指定指标 db.load.avg 和前七个等待状态的 GroupBy。有关此示例有效值的详细信息，请参阅 Performance Insights API 参考DimensionGroup中的。

响应类似于以下内容。

```
{
    "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
```

此响应的 MetricList 中具有 11 个条目（1 个总计，10 个首要令牌化查询），其中每个条目具有 24 个每小时 DataPoints。

对于令牌化查询，每个维度列表中具有三个条目：

- db.query_tokenized.statement：令牌化的查询语句。
- db.query_tokenized.db_id ：Performance Insights 为您生成的合成 ID。此示例返回 pi-1064184600 合成 ID。

- db.query_tokenized.id：Performance Insights 中的查询的 ID。

  在中 AWS 管理控制台，此 ID 被称为 Support ID。之所以这样命名，是因为 ID 是 Su AWS pport 可以检查的数据，以帮助您解决数据库问题。 AWS 非常重视数据的安全性和隐私性，几乎所有数据都使用您的数据加密存储 AWS KMS key。因此，里面没有人 AWS 可以查看这些数据。在上一个示例中，tokenized.statement 和 tokenized.db_id 都进行了加密存储。如果您的数据库出现问题，Su AWS pport 可以通过引用 Support ID 来帮助您。

在查询时，在 Group 中指定 GroupBy 可能很方便。不过，要更精细地控制返回的数据，请指定维度列表。例如，如果所需的所有内容是 db.query_tokenized.statement，则可将 Dimensions 属性添加到 query.json 文件中。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": {
            "Group": "db.query_tokenized",
            "Dimensions":["db.query_tokenized.statement"],
            "Limit": 10
        }
    }
]
```

检索按查询筛选的数据库负载平均值

此示例中的相应 API 查询类似于 检索主要查询的数据库负载平均值 中的命令。不过，query.json 文件具有以下内容。

```
[
 {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_state", "Limit": 5  },
        "Filter": { "db.query_tokenized.id": "AKIAIOSFODNN7EXAMPLE" }
    }
]
```

在此响应中，根据 query.json 文件中指定的令牌化查询 AKIAIOSFODNN7EXAMPLE 的贡献筛选所有值。键还可能遵循与没有筛选条件的查询不同的顺序，因为前五个等待状态影响了筛选的查询。

Performance Insights 的亚马逊 CloudWatch 指标

Performance Insights 会自动向亚马逊发布指标 CloudWatch。可以从 Performance Insights 中查询相同的数据，但是将指标包含在里面可以很 CloudWatch 容易地添加 CloudWatch 警报。还可以轻松地将指标添加到现有 CloudWatch 控制面板中。

指标	描述
DBLoad	Amazon DocumentDB 的活动会话数。通常，您需要活动会话的平均数量数据。在 Performance Insights 中，作为 `db.load.avg` 查询此数据。
DBLoadCPU	等待状态类型为 CPU 的活动会话的数量。在 Performance Insights 中，作为 `db.load.avg` 查询此数据，按等待状态类型 CPU 进行筛选。
DBLoadNonCPU	等待状态类型不为 CPU 的活动会话的数量。

> **Note**
>
> CloudWatch 仅当数据库实例有负载时，才会将这些指标发布到。

您可以使用 CloudWatch 控制台 AWS CLI、或 CloudWatch API 来检查这些指标。

例如，可以通过运行 get-metric-statistics 命令来获取 DBLoad 指标的统计数据。

```
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
```

该示例将生成与下类似的输出。

```
{
    "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"
}
```


您可以使用 CloudWatch 控制台中的DB_PERF_INSIGHTS指标数学函数来查询 Amazon DocumentDB Performance Insights 计数器指标。DB_PERF_INSIGHTS 函数还包括以亚分钟为间隔的 DBLoad 指标。您可以对这些指标设置 CloudWatch 警报。有关如何创建警报的更多详细信息，请参阅针对AWS数据库中的 Performance Insights 计数器指标创建警报。

有关的更多信息 CloudWatch，请参阅 Amazon 是什么 CloudWatch？ 在《亚马逊 CloudWatch 用户指南》中。

Performance Insights 的计数器指标

计数器指标是 Performance Insights 控制面板中的操作系统指标。为帮助确定和分析性能问题，您可将计数器指标与数据库负载相关联。

Performance Insights 操作系统计数器

以下操作系统计数器可用于 Amazon DocumentDB Performance Insights。

计数器	类型	指标
active	memory	os.memory.active
buffers	memory	os.memory.buffers
cached	memory	os.memory.cached
dirty	memory	os.memory.dirty
free	memory	os.memory.free
inactive	memory	os.memory.inactive
mapped	memory	os.memory.mapped
pageTables	memory	os.memory.pageTables
slab	memory	os.memory.slab
total	memory	os.memory.total
writeback	memory	os.memory.writeback
idle	cpuUtilization	os.cpu Utilization.idle
system	cpuUtilization	os.cpu Utilization.system
total	cpuUtilization	os.cpu Utilization.total

计数器	类型	指标
用户	cpuUtilization	os.cpu Utilization.user
wait	cpuUtilization	os.cpu Utilization.wait
one	加载 AverageMinute	os.load AverageMinute.one
fifteen	加载 AverageMinute	os.load AverageMinute.fifteen
five	加载 AverageMinute	os.load AverageMinute.five
cached	swap	os.swap.cached
free	swap	os.swap.free
in	swap	os.swap.in
out	swap	os.swap.out
total	swap	os.swap.total
rx	network	os.network.rx
tx	network	os.network.tx
numVCPUs	general	os.general.numVCPUs

Zero-ETL 与亚马逊 OpenSearch 服务集成

主题

- 以亚马逊 OpenSearch 服务为目的地
- 限制

以亚马逊 OpenSearch 服务为目的地

OpenSearch 与 Amazon DocumentDB 的服务集成使您能够将满负荷事件和更改数据事件流式传输到 OpenSearch 域中。摄取基础设施以摄取管道的形式托管，它提供了一种高规模、低延迟的机制，可以持续流式传输来自 Amazon Doc OpenSearch umentDB 馆藏的数据。

在满载期间，Zero-ETL 集成首先 OpenSearch 使用摄取管道提取历史满载数据。载入满载数据后，摄取管道将开始从 Amazon Doc OpenSearch umentDB 变更流中读取数据，并最终赶上，以保持 Amazon DocumentDB 和之间近乎实时的数据一致性。 OpenSearch OpenSearch 将文档存储在索引中。从 Amazon DocumentDB 集合传入的数据可以发送到一个索引，也可以分割成不同的索引。采集管道会将 Amazon DocumentDB 集合中的所有创建、更新和删除事件与相应的文档创建、更新和删除 OpenSearch 事件同步，以使两个数据系统保持同步。可以将摄取管道配置为从一个集合中读取数据并写入一个索引，或者从一个集合中读取数据并按条件路由到多个索引。

可以将摄取管道配置为使用以下方法将数据从 Amazon DocumentDB 流式传输到 OpenSearch 亚马逊服务：

- 仅完全加载
- 从 Amazon DocumentDB 流式传输变更流事件而不完全加载
- 完全加载，然后是来自 Amazon DocumentDB 的变更流

要设置摄取管道，请执行以下步骤：

步骤 1：创建 Amazon OpenSearch 服务域名或 OpenSearch 无服务器集合

需要具有相应数据读取权限的 Amazon OpenSearch 服务集合。要创建集合，请参阅《亚马逊 OpenSearch 服务开发者指南》中的 “亚马逊 OpenSearch 服务入门” 或 “亚马逊 OpenSearch 无服务器入门”。请参阅《亚马逊 OpenSearch 服务开发者指南》中的 Amazon OpenSearch Ingest ion，创建具有访问集合或域名写入数据的正确权限的 AIM 角色。

步骤 2：在 Amazon DocumentDB 集群上启用变更流

确保在 Amazon DocumentDB 集群中的所需集合上启用变更流。有关更多信息，请参阅[将变更流与 Amazon DocumentDB 结合使用]。

步骤 3：设置管道角色，使其拥有写入 Amazon S3 存储桶和目标域或集合的权限

在创建 Amazon DocumentDB 集合并启用变更流后，设置要在管道配置中使用的管道角色，并在该角色中添加以下权限：

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/*"
            ]
        }
    ]
}
```

为了使 OpenSearch 管道能够将数据写入 OpenSearch 域，该域必须具有允许 sts_role_arn 管道角色访问该域的域级访问策略。以下示例域访问策略允许您在上一步中创建的名为 `pipeline-role` 的管道角色向名为 `ingestion-domain` 的域写入数据：

```
{
  "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}/*"
    }
  ]
}
```

步骤 4：添加要创建的管道角色所需的权限 X-ENI

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" }
            }
        }
    ]
}
```

步骤 5：创建管道

配置一个将亚马逊文档数据库指定为来源的 OpenSearch 摄取管道。此示例管道配置假设采用变更流获取机制。有关更多信息，请参阅 OpenSearch 《亚马逊服务开发者指南》中的在 Amazon DocumentDB 中使用采集管道。 OpenSearch

限制

以下限制适用于亚马逊 DocumentDB 集成 OpenSearch ：

- 每个管道仅支持将一个 Amazon DocumentDB 集合作为源。
- Cross-region 不支持数据摄取。您的 Amazon DocumentDB 集群和 OpenSearch 域必须位于同一 AWS 区域。
- Cross-account 不支持数据摄取。您的 Amazon DocumentDB 集群和 OpenSearch 采集管道必须位于同一个账户中。 AWS
- 不支持 Amazon DocumentDB 弹性集群。仅支持 Amazon DocumentDB 基于实例的集群。
- 确保 Amazon DocumentDB 集群启用了使用 AWS 密钥的身份验证。 AWS 机密是唯一支持的身份验证机制。
- 无法更新现有的管道配置以从不同的数据库 and/or 中提取不同集合的数据。要更新管道的数据库 and/or 集合名称，必须创建新的管道。

使用 Amazon DocumentDB 无服务器

Amazon DocumentDB 无服务器是一种按需自动扩缩配置，可根据应用程序需求自动执行动态调整 Amazon DocumentDB 数据库容量的过程。您只需为集群使用的资源付费。因此，DocumentDB 无服务器可以帮助您保持在预算范围内，避免为不使用的计算机资源付费。

这种自动化对于具有高度可变且不可预测工作负载的环境（例如多租户数据库、分布式数据库、开发和测试系统）尤为重要。

主题

- DocumentDB 无服务器使用案例
- Amazon DocumentDB 无服务器的优势
- Amazon DocumentDB 无服务器的工作原理
- DocumentDB 无服务器的要求和限制
- 创建使用 Amazon DocumentDB 无服务器的集群
- 迁移到 Amazon DocumentDB 无服务器
- 管理 Amazon DocumentDB 无服务器
- Amazon DocumentDB 无服务器实例限制
- Amazon DocumentDB 无服务器扩缩配置
- 监控 Amazon DocumentDB 无服务器

DocumentDB 无服务器使用案例

Amazon DocumentDB 预置集群和 DocumentDB 无服务器均支持多种类型的数据库工作负载，从开发和测试环境到最严苛的业务关键型应用程序（需要高扩展性和高可用性）。但是 DocumentDB 无服务器为客户工作负载增加了另一个维度，即能够支持工作负载不可预测的网站和应用程序。

DocumentDB 无服务器对于以下使用案例特别有用：

- 可变工作负载：您运行的工作负载会突然出现无法预测的活动增加。例如，在开始下雨时显示活动涌现的流量网站。另一个例子是当您提供销售或特别促销活动时流量会增加的电子商务网站。借助 DocumentDB 无服务器，您的数据库可自动扩展容量以满足应用程序的峰值负载需求，然后在活动涌现结束时再缩减。借助 DocumentDB 无服务器，您不再需要为峰值或平均容量进行预置。您可以指定容量上限来应对最糟糕情况，除非确实需要该容量，否则不使用该容量。

  - DocumentDB 无服务器中的扩缩粒度帮助您使容量与数据库的需求紧密匹配。对于预置的集群，扩展需要添加一个全新的实例。当仅需少量额外容量时，DocumentDB 无服务器可以增加半个 DCU。根据处理工作负载增加所需的额外容量，可以增加 0.5、1、1.5、2 或额外的半个 DCU。当工作负载减少且不再需要该容量时，可以删除 0.5、1、1.5、2 或额外的半个 DCU。
- Multi-tenant 应用程序 — 使用 DocumentDB 无服务器，您无需单独管理队列中每个应用程序的数据库容量。DocumentDB 无服务器为您管理单个数据库容量。
  - 您可为每个租户创建一个集群。这样，您就可以视情况使用克隆和快照恢复等功能为每个租户增强高可用性和灾难恢复。
  - 根据一天中的时间、一年中的时间、促销活动等，每个租户可能有特定的忙碌和空闲时段。每个集群都有宽容量范围。这样，活动量较少的集群将产生更低的实例费用。任何集群都可以快速扩展以应对活动较多的时期。
- 新应用程序：您正在部署新应用程序，但不确定所需的实例大小。通过使用 DocumentDB 无服务器，您可以设置具有一个或多个实例的集群，并让数据库根据应用程序的容量需求自动扩展。
- Mixed-use 应用程序-假设您有一个在线事务处理 (OLTP) 应用程序，但查询流量会定期出现峰值。通过在集群中指定 DocumentDB 无服务器实例的提升层，您可以将集群配置为让读取器实例可以独立于写入器实例进行扩展以处理额外负载。当使用高峰消退时，读取器实例会缩减以匹配写入器实例的容量。
- 容量规划：假设您通常通过修改集群中所有实例的实例类来调整数据库容量，或者验证工作负载的最佳数据库容量。借助 DocumentDB 无服务器，您可以避免产生此管理开销。您可以通过运行工作负载并检查实例实际扩展的程度来确定适当的最小和最大容量。
  - 您可以将现有实例从预置修改为 DocumentDB 无服务器或者从 DocumentDB 无服务器修改为预置。在此类情况下，您无需创建新集群或新实例。
- 开发和测试：除了运行要求最严苛的应用程序外，您还可以将 DocumentDB 无服务器用于开发和测试环境。借助 DocumentDB 无服务器，您可以创建最低容量较低的实例，而不是使用可突增的 db.t* 实例类。您可以将最大容量设置得足够高，以便这些实例仍然可以运行大量工作负载，而不会出现内存不足的情况。当数据库未在使用时，所有实例都会缩减以避免产生不必要的费用。

使用 Amazon DocumentDB 无服务器处理现有的预置工作负载

假设您已在预置集群上运行 DocumentDB 应用程序。您可以将一个或多个 DocumentDB 无服务器实例作为读取器实例添加到现有集群，检查应用程序将如何与 DocumentDB 无服务器配合使用。您可以查看读取器实例扩展和缩减的频率。您可以使用 DocumentDB 故障转移机制将 DocumentDB 无服务器实例提升为写入器，并检查它如何处理工作负载。 read/write 这样，您可以在最短的停机时间内进行切

换，而且无需更改客户端应用程序使用的端点。有关将现有集群转换为 DocumentDB 无服务器的过程的详细信息，请参阅 迁移到 Amazon DocumentDB 无服务器。

Amazon DocumentDB 无服务器的优势

DocumentDB 无服务器适用于可变或“高峰”工作负载。对于此类不可预测的工作负载，您可能难以规划何时更改数据库容量。使用熟悉的机制（例如添加实例或更改实例类），您可能也难以足够快地进行容量更改。DocumentDB 无服务器提供了以下优势来帮助处理此类使用案例：

- 比预置更简单的容量管理：DocumentDB 无服务器减少了规划实例大小和随着工作负载变化调整实例大小的工作量。此外，还可以减少为集群中的所有实例维持一致容量的工作量。

- 在活动较多的时期更快、更轻松地扩展：DocumentDB 无服务器可根据需要扩展计算和内存容量，而无需中断客户端事务或总体工作负载。借助 DocumentDB 无服务器，能够使用读取器实例，除了垂直扩缩外，还可以帮助您利用水平扩缩。能够使用 DocumentDB 全局集群意味着您可以将您的 DocumentDB 无服务器读取工作负载分散到多个集群。 AWS 区域此功能比预置集群的扩缩机制更便利。

- Cost-effective 在@@ 活动量低的时期 — DocumentDB 无服务器可帮助您避免实例过度配置。当实例扩展时，DocumentDB 无服务器以细粒度增量添加资源。您只需为所使用的数据库资源付费。DocumentDB 无服务器资源使用情况按秒计量。这样，当实例缩减时，会立即注册减少的资源使用量。

- 与预置同等的功能：您可以在 DocumentDB 无服务器中使用所有 DocumentDB 功能。例如，使用 DocumentDB 无服务器，您可以使用读取器实例、 AWS Identity and Access Management (IAM) 数据库身份验证和 Performance Insights。

  特别是，借助 DocumentDB 无服务器，您可以利用预置集群的以下功能：

  - 读取器实例：DocumentDB 无服务器可以利用读取器实例进行横向扩展。当集群包含一个或多个读取器实例时，如果写入器实例出现问题，集群可以立即进行失效转移。
  - Multi-AZ 集群 — 您可以将集群的 DocumentDB 无服务器实例分发到多个可用区 (AZ)。即使在影响整个可用区的极少数问题中，设置 Multi-AZ 集群也有助于确保业务连续性。
  - 全局群集 — 您可以将 DocumentDB 无服务器与 DocumentDB 全局群集结合使用，在其他集群中创建其他 AWS 区域 群集的只读副本，以用于灾难恢复。

Amazon DocumentDB 无服务器的工作原理

主题

- 概述
- Amazon DocumentDB 集群的配置
- Amazon DocumentDB 无服务器扩缩容量
- Amazon DocumentDB 无服务器扩缩
- 空闲状态（0.5 个 DCU）

概述

Amazon DocumentDB 无服务器适用于要求严苛、变化很大的工作负载。例如，您的数据库使用量在短时间内可能很大，紧接着就是长时间的少量活动或完全没有活动。其中一些例子是定期举办促销活动的零售、游戏或体育网站，以及根据需要生成报告的数据库。其他一些例子包括开发和测试环境，以及使用量可能会迅速增加的新应用程序。对于类似这些情况和许多其他情况，使用预置模型并不总是能提前正确地配置容量。如果您过度预置并且保留不使用的容量，也可能导致更高的成本。

相比之下，DocumentDB 预置集群适合稳定的工作负载。对于已配置的集群，您可以选择具有预定义内存量、CPU 功耗、 I/O 带宽等的实例类别。如果您的工作负载发生变化，可以手动修改写入器和读取器的实例类。如果您可以在预期的消费模式之前调整容量，并且在更改集群中写入器和读取器的实例类时，出现短暂的中断是可以接受的，则预置模式很适合。

DocumentDB 无服务器专为支持即时可扩展的无服务器集群而从头开始构建。DocumentDB 无服务器的设计旨在提供与预置的写入器和读取器相同程度的安全性和隔离性。这些方面在多租户无服务器云环境中至关重要。动态扩缩机制的开销非常小，因此它可以快速响应数据库工作负载的变化。它还足够强大，可以满足处理需求的急剧增长。

通过使用 DocumentDB 无服务器，您可以创建 DocumentDB 集群，而无需将每个写入器和读取器限制在特定的数据库容量。您指定最小和最大容量范围。DocumentDB 会在该容量范围内扩缩集群中的每个 DocumentDB 无服务器写入器或读取器。通过使用每个写入器或读取器都可以动态扩展的 Multi-AZ 集群，您可以利用动态扩展和高可用性。

DocumentDB 无服务器根据您的最小和最大容量规格自动扩缩数据库资源。扩缩速度很快，因为大多数扩缩事件操作将写入器或读取器保持在同一主机上。在极少数情况下，DocumentDB 无服务器写入器或读取器从一个主机移动到另一个主机，DocumentDB 无服务器自动管理连接。您无需更改数据库客户端应用程序代码或数据库连接字符串。

与预置集群一样，使用 DocumentDB 无服务器时，存储容量和计算容量是分开的。当我们提到 DocumentDB 无服务器容量和扩缩时，增加或减少的总是计算容量。因此，即使 CPU 和内存容量缩减至低水平，您的集群仍会包含许多 TB 的数据。

您可以指定数据库容量，而不是调配和管理数据库服务器。每个 DocumentDB 无服务器写入器或读取器的实际容量会随着时间的推移而发生变化，具体取决于您的工作负载。有关该机制的详细信息，请参阅 Amazon DocumentDB 无服务器扩缩。

Amazon DocumentDB 集群的配置

对于每个 Amazon DocumentDB 集群，您可以选择 DocumentDB 无服务器容量、预置容量或两者的任意组合。

您可以设置一个同时包含 DocumentDB 无服务器和预置容量的集群，称为混合配置集群。例如，假设您需要的 read/write 容量超过了 DocumentDB 无服务器写入器的可用容量。在这种情况下，您可以使用非常大的预置写入器来设置集群。然后，您仍然可以对读取器使用 DocumentDB 无服务器。或者假设集群的写入工作负载会变化，但读取工作负载保持稳定。在这种情况下，您可以使用 DocumentDB 无服务器写入器和一个或多个预置读取器设置集群。

您还可以设置由 DocumentDB 无服务器管理所有容量的集群。为此，您可以创建新的集群并从头开始使用 DocumentDB 无服务器。或者，您可以将现有集群中的所有预置容量替换为 DocumentDB 无服务器。对于使用 DocumentDB 无服务器创建新集群或者将现有集群切换到 DocumentDB 无服务器的过程，请参阅 创建使用 Amazon DocumentDB 无服务器的集群 和 迁移到 Amazon DocumentDB 无服务器。

如果您在集群中根本不使用 DocumentDB 无服务器，则集群中的所有写入器和读取器都为预置。这是大多数用户都熟悉的最常见的集群类型。预置容量是恒定的。费用相对更容易预测。但是，您必须提前预测所需的容量。在某些情况下，您的预测可能不准确，或者容量需求可能会发生变化。在这些情况下，您的集群可能会变得预置不足（比您期望的更慢）或预置过度（比您期望的更昂贵）。

Amazon DocumentDB 无服务器扩缩容量

Amazon DocumentDB 无服务器的计量单位是 DocumentDB 容量单位（DCU）。DocumentDB 无服务器扩缩容量与您用于预置集群的实例类无关。

每个 DCU 是约 2GiB 的内存、相应的 CPU 和网络的组合。您可以使用此计量单位指定数据库容量范围。`ServerlessDatabaseCapacity` 和 `DCUUtilization` CloudWatch 指标帮助您确定数据库实际使用的容量以及该容量在指定范围内的位置。

在任何时候，每个 DocumentDB 无服务器写入器或读取器有一个容量。该容量是代表 DCU 的浮点数。每当写入器或读取器扩缩时，容量会增加或减少。此值每秒测量一次。对于您打算使用 DocumentDB 无服务器的每个集群，定义一个容量范围：每个 DocumentDB 无服务器写入器或读取器可以在其间进行扩缩的最小容量和最大容量值。集群中的每个 DocumentDB 无服务器写入器或读取器的容量范围都相同。每个 DocumentDB 无服务器写入器或读取器都有自己的容量，该容量处于该范围内的某个位置。

DocumentDB 5.0 和 8.0 支持 DocumentDB 无服务器，容量范围为 0.5-256 个 DCU。

您可以定义的最小 DocumentDB 无服务器容量是 0.5 个 DCU。如果小于或等于最大支持容量值，您可以指定一个较高的数值。将最小容量设置为一个很小的数字，可以让负载较轻的集群消耗最少的计算资源。同时，他们随时准备立即接受连接，并在变得忙碌时进行扩展。

我们建议将最小值设置为让每个写入器或读取器可以在缓冲池中保留应用程序工作集的值。这样，在空闲期间，缓冲池的内容不会被丢弃。有关选择扩缩容量范围时的所有注意事项，请参阅 为 DocumentDB 无服务器集群选择扩缩容量范围。

根据您在 Multi-AZ 部署中配置读取器的方式，读取器的容量可以与写入器的容量挂钩，也可以独立使用。有关如何执行此操作的详细信息，请参阅 查看和修改无服务器读取器的提升层。

监控 DocumentDB 无服务器涉及随时间推移测量集群中的写入器和读取器的容量值。如果数据库没有缩减至最小容量，则可以采取行动，例如调整最小值和优化数据库应用程序。如果数据库持续达到最大容量，则可以采用行动，例如增大最大容量。您还可以优化数据库应用程序，并将查询负载分散到更多读取器中。

DocumentDB 无服务器容量的费用按以下标准衡量。 DCU-hours有关如何计算 DocumentDB 无服务器费用的信息，请参阅 Amazon DocumentDB 定价。假设集群中写入器和读取器的总数为 n。在这种情况下，当您未运行任何数据库操作时，集群会消耗大约 n x 最小 DCU。Amazon DocumentDB 本身可能会运行监控或维护操作，从而产生少量负载。当数据库以全部容量运行时，该集群消耗不超过 n x 最大 DCU。

有关选择适当的最小和最大 DCU 值的更多详细信息，请参阅 为 DocumentDB 无服务器集群选择扩缩容量范围。您指定的最小和最大 DCU 值也会影响一些 Amazon DocumentDB 实例限制。有关容量范围与实例限制之间的相互影响的详细信息，请参阅 Amazon DocumentDB 无服务器实例限制。

Amazon DocumentDB 无服务器扩缩

对于每个 DocumentDB 无服务器写入器或读取器，Amazon DocumentDB 会持续跟踪 CPU、内存和网络等资源的使用率。这些测量统称为负载。负载包括应用程序执行的数据库操作。还包括数据库服务器的后台处理和 Amazon DocumentDB 管理任务。当容量受到上述任何限制时，DocumentDB

无服务器就会扩展。当 DocumentDB 无服务器检测到可以通过扩展予以解决的性能问题时，也会扩展。您可以使用 监控 Amazon DocumentDB 无服务器 中的过程监控资源利用率以及了解其如何影响 DocumentDB 无服务器扩缩。

集群中的写入器和读取器的负载可能会有所不同。写入器处理写入操作，并执行针对集群卷的所有数据修改。读取器可以处理只读请求。

扩缩是为数据库增大或减少 DocumentDB 无服务器容量的操作。借助 DocumentDB 无服务器，每个写入器和读取器都有其自己的当前容量值，以 DCU 为单位。当写入器或读取器的当前容量太低而无法处理负载时，DocumentDB 无服务器可将其扩展到更高的容量。当写入器或读取器的当前容量高于所需容量时，可以将写入器或读取器缩减至更低的容量。

DocumentDB 无服务器可以逐步增加容量。当您的工作负载需求开始达到写入器或读取器的当前数据库容量时，DocumentDB 无服务器会增加该写入器或读取器的 DCU 数量。DocumentDB 无服务器会按所需的增量扩展容量，以便为所消耗的资源提供最佳性能。扩缩按小至 0.5 个 DCU 的增量进行。当前容量越大，扩缩增量就越大，因此可以更快地进行扩缩。

由于 DocumentDB 无服务器扩缩非常频繁、精细且无中断，因此不会在 AWS 管理控制台中导致离散事件。相反，您可以衡量 Amazon CloudWatch 指标（例如`serverlessDatabaseCapacity`和）`DCUUtilization`，并跟踪它们在一段时间内的最小值、最大值和平均值。要了解有关监控 DocumentDB 无服务器的更多信息，请参阅 监控 Amazon DocumentDB 无服务器。

扩展或缩减可能是由以下原因引起的：

- 内存利用率
- CPU 使用率
- 网络使用率
- 存储使用率

您可以监控这些在 DocumentDB 无服务器实例 up/down 上扩容的原因。有关更多信息，请参阅 监控 Amazon DocumentDB 无服务器。

您可以选择使读取器与关联写入器同时扩缩，也可以独立于写入器进行扩缩。您可以通过为该读取器指定提升层来完成扩缩。

- 提升层 0 和 1 中的 DocumentDB 无服务器读取器与写入器同时扩缩。这种扩缩行为使得优先级层 0 和 1 中的读取器非常容易获得。这是因为它们的大小总是调整为适当的容量，以便在失效转移情况下接管来自写入器的工作负载。

- 提升层 2-15 中的读取器可以独立于写入器进行扩缩。每个读取器都保持在您为集群指定的最小和最大 DCU 值范围内。当读取器独立于关联的写入器数据库进行扩缩时，它会变为空闲并缩减，同时写入器继续处理大量事务。如果在较低的提升层中没有其他读取器可用，它仍可作为失效转移目标。但是，如果它被提升为写入器，则可能需要扩展以处理写入器的全部工作负载。

有关查看和更改无服务器实例的提升层的详细信息，请参阅 查看和修改无服务器读取器的提升层。

DocumentDB 无服务器扩缩可能在数据库连接打开、事务处理正在进行等情况下发生。DocumentDB 无服务器不会等到安静点才开始扩缩。扩缩不会中断任何正在进行的数据库运营。

如果您的工作负载需要的读取容量超过单个写入器和单个读取器可提供的读取容量，则可以向集群添加多个 DocumentDB 无服务器读取器。每个 DocumentDB 无服务器读取器可以在您为集群指定的最小和最大容量值范围内扩缩。您可以使用集群的读取器端点将只读会话定向到读取器并减少写入器上的负载。

DocumentDB 无服务器是否执行扩缩，以及扩缩在启动后的速度也取决于集群的最小和最大 DCU 设置。此外，它还取决于读取器是配置为随写入器一起扩缩还是独立于写入器进行扩缩。有关扩缩配置的详细信息，请参阅 Amazon DocumentDB 无服务器扩缩配置。

空闲状态（0.5 个 DCU）

当 Amazon DocumentDB 无服务器写入器或读取器处于空闲状态时，如果集群配置为 0.5，DocumentDB 无服务器实例支持缩减到 0.5 DCU 的空闲状态。 MinCapacity

在空闲状态下，DocumentDB 无服务器实例虽然没有足够的 CPU 计算容量来支持大多数生产工作负载，但可以随时快速扩展以支持新的工作负载。在非空闲状态下，DocumentDB 无服务器实例通常需要至少 1.0 到 2.5 个 DCU。因此，当 DocumentDB 无服务器实例从空闲状态扩展到非空闲状态时，它们将直接扩展到 1.0-2.5 DCU（ MaxCapacity 如果值较低，则为的值）。

为了支持在空闲时缩减到 0.5 个 DCU，如果配置为小于或等于 1.0 DCU， MinCapacity 则会对实例限制设定上限。有关 MinCapacity 配置如何影响限制的更多信息，请参阅Amazon DocumentDB 无服务器实例限制。

DocumentDB 无服务器的要求和限制

功能要求

区域可用性

Amazon DocumentDB 无服务器实例类型已在以下区域推出：

以下 AWS CLI 命令可用于验证特定区域提供的确切的 DocumentDB 无服务器实例选项：

```
aws docdb describe-orderable-db-instance-options \
        --region my_region \
        --db-instance-class db.serverless \
        --engine docdb
```

引擎版本可用性

亚马逊 DocumentDB 5.0 和 8.0 引擎版本支持 DocumentDB 无服务器。它不适用于引擎版本 3.6 或 4.0。

> **Note**
>
> DocumentDB 无服务器仅在较新的补丁版本 5.0.0 上才受支持。确保您的集群已更新到最新的引擎补丁版本。有关维护 Amazon DocumentDB 集群的更多信息，请参阅 维护 Amazon DocumentDB

集群配置

在将任何 Amazon DocumentDB 无服务器实例添加到 Amazon DocumentDB 集群之前，必须先为集群设置 `ServerlessV2ScalingConfiguration` 参数。该参数定义集群中 DocumentDB 无服务器实例的容量范围。有关扩缩配置的更多信息，请参阅 Amazon DocumentDB 无服务器扩缩配置。

特定 Amazon DocumentDB 功能的最小扩缩容量范围设置

有一些 Amazon DocumentDB 功能适用于 DocumentDB 无服务器，但如果您的容量范围低于特定工作负载下的这些功能的内存需求所需的容量，则可能会导致问题。在这种情况下，数据库可能无法像往常一样运行，或者可能会遇到内存不足错误。

以下功能需要配置更高的`MinCapacity` and/or `MaxCapacity`值才能实现最佳操作：

- 性能详情
- 全局群集（仅适用于主群集 AWS 区域）
- 在数据量大的集群上创建无服务器实例

  这包括在集群恢复过程中创建无服务器实例。

有关设置适当容量范围的建议（如果您正在使用此功能），请参阅 为 DocumentDB 无服务器集群选择扩缩容量范围。如果您的数据库因容量范围配置错误而遇到内存不足错误，如需故障排除信息，请参阅避免内存不足错误。

创建使用 Amazon DocumentDB 无服务器的集群

创建 Amazon DocumentDB 无服务器集群

借助 Amazon DocumentDB 无服务器，您的集群可以与预置集群互换。您的集群可以有一些实例使用无服务器，有些实例为预置实例。

确认所需区域和引擎版本支持 DocumentDB 无服务器。请参阅DocumentDB 无服务器的要求和限制。

要创建可以在其中添加无服务器实例的 Amazon DocumentDB 集群，请按照 创建 Amazon DocumentDB 集群 中的相同过程进行操作。唯一区别在于还必须提供 `ServerlessV2ScalingConfiguration` 参数。

`ServerlessV2ScalingConfiguration` 参数指定 DocumentDB 无服务器实例的扩缩容量范围。由适用于集群中所有 DocumentDB 无服务器实例的最小和最大 DocumentDB 容量单位（DCU）值组成：

- `MinCapacity` 值指定最小扩缩容量。
- `MaxCapacity` 值指定最大扩缩容量。

有关扩展的更多信息，请参阅Amazon DocumentDB 无服务器扩缩配置。

Using the AWS 管理控制台

以下 AWS 管理控制台 配置示例显示了如何创建 DocumentDB 无服务器集群。

1. 登录 AWS 管理控制台 并打开 Amazon DocumentDB 控制台。
2. 在导航窗格中，选择集群。

> **Tip**
>
> 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

将显示集群表。

3. 选择创建。

   将显示创建 Amazon DocumentDB 集群页面。

4. 在创建 Amazon DocumentDB 集群页面的集群类型部分，选择Instance-based 集群（这是默认选项）。

5. 在集群配置部分中：

   a. 在集群标识符中，输入唯一名称，例如 **myserverlesscluster**。请注意，无论如何输入，控制台都会将所有集群的名称更改为小写。

   b. 对于引擎版本，选择 5.0.0（这是默认选项）。

6. 在集群存储配置部分，选择 Amazon DocumentDB 标准（此为默认选项）。

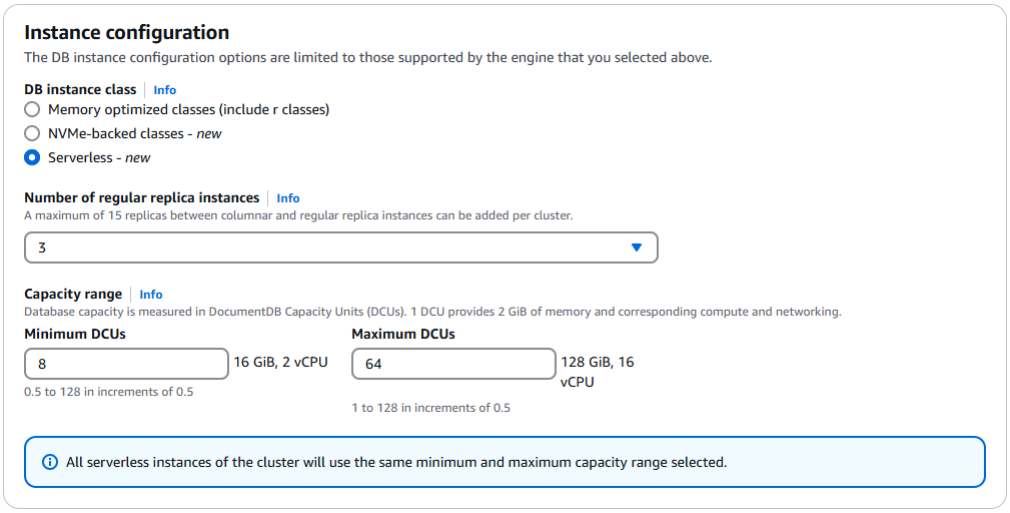
   > **Note**
   >
   > 此类别中的另一个选项是 Amazon Doc I/O-Optimized umentDB。要了解有关任一选项的更多信息，请参阅 Amazon DocumentDB 集群存储配置

7. 在实例配置部分：

   a. 对于数据库实例类，选择无服务器。

   b. 对于常规副本实例数，选择 3（这是默认选项）。

   c. 在容量范围部分中，保留最小 DCU 和最大 DCU 的默认值。有关设置这些参数的信息，请参阅 Amazon DocumentDB 无服务器实例限制。



8. 在连接部分中，保留默认设置：不连接到 EC2 计算资源。
9. 在身份验证部分中，输入主要用户的用户名，然后选择自行管理。输入密码，然后确认密码。

   如果您改为在中选择 “托管” AWS Secrets Manager，使用 Amazon DocumentDB 进行密码管理以及 AWS Secrets Manager请参阅，了解更多信息。
10. 使其他所有选项保持默认，并选择创建集群。

Using the AWS CLI

在以下示例中，*user input placeholder*使用您自己的信息或配置参数替换每个示例。

要使用创建与 DocumentDB 无服务器实例兼容的集群， AWS CLI请参阅。使用创建集群 AWS CLI

在 `create-db-cluster` 命令中包含以下附加参数：

```
--serverless-v2-scaling-configuration
     MinCapacity=minimum_capacity,MaxCapacity=maximum_capacity
```

示例：

```
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
```

添加 Amazon DocumentDB 无服务器实例

要添加 DocumentDB 无服务器实例，请按照 向集群添加 Amazon DocumentDB 实例 中的相同步骤操作，确保将 db.serverless 指定为实例类。

使用添加无服务器实例 AWS 管理控制台.

要使用控制台添加 DocumentDB 无服务器实例，请参阅 向集群添加 Amazon DocumentDB 实例，并选择使用 AWS 管理控制台 选项卡。

使用添加无服务器实例 AWS CLI

要使用添加 DocumentDB 无服务器实例 AWS CLI，请参阅向集群添加 Amazon DocumentDB 实例并选择使用选项卡。 AWS CLI

使用以下实例类 CLI 参数：

```
--db-instance-class db.serverless
```

示例：

```
aws docdb create-db-instance \
      --db-cluster-identifier sample-cluster \
      --db-instance-identifier sample-instance \
      --db-instance-class db.serverless \
      --engine docdb
```

迁移到 Amazon DocumentDB 无服务器

主题

- 将现有的 DocumentDB 集群迁移到无服务器
- 从 MongoDB 迁移到 DocumentDB 无服务器

将现有的 DocumentDB 集群迁移到无服务器

升级集群的引擎版本

如果您的预置集群运行的是不支持 DocumentDB 无服务器的较低引擎版本，则需要先将集群升级到支持的引擎版本。有关更多信息，请参阅 Amazon DocumentDB 主版本就地升级。

将预置集群迁移到 DocumentDB 无服务器

要将预置集群切换为使用 DocumentDB 无服务器，请按照以下步骤操作：

1. 检查是否需要升级预置集群的引擎版本以与 DocumentDB 无服务器配合使用。请参阅DocumentDB 无服务器的要求和限制。

   > **Note**
   >
   > 如果预置集群运行的是不适用于 DocumentDB 无服务器的引擎版本，则升级集群的引擎版本。请参阅Amazon DocumentDB 主版本就地升级。

2. 配置集群的扩缩配置。有关选择扩缩配置的详细信息，请参阅 为 DocumentDB 无服务器集群选择扩缩容量范围。要修改集群的扩缩配置，请参阅 查看和修改集群的扩缩容量范围配置。
3. 配置任何其他集群属性以满足 DocumentDB 无服务器的要求和限制 中的 DocumentDB 无服务器要求和限制。
4. 将一个或多个 DocumentDB 无服务器实例添加到该集群。按照添加 Amazon DocumentDB 无服务器实例中的程序操作。

   > **Note**
   >
   > 在某些情况下，您的集群中可能已经拥有一个或多个预置读取器实例。如果已经拥有，您可以选择将其中一个读取器转换为 DocumentDB 无服务器实例，而不是创建新的实例。为此，请按照更改实例的类中的过程操作

5. （可选）执行失效转移操作，以使 DocumentDB 无服务器实例成为集群写入器。请参阅Amazon DocumentDB 失效转移。
6. （可选）将任何剩余的预置 Amazon DocumentDB 实例转换为 DocumentDB 无服务器实例（请参阅 更改实例的类），或者将其从集群中移除（请参阅 删除 Amazon DocumentDB 实例）。

Using the AWS 管理控制台

以下 AWS 管理控制台 配置示例显示了使用运行 Amazon DocumentDB 5.0.0 的 Amazon DocumentDB 预配置集群的迁移过程，该集群不需要升级引擎版本即可开始使用 DocumentDB 无服务器。集群名为 `sample`，从名为 `sample`、`sample2` 和 `sample3` 的三个预置实例开始。在此示例中，将这三个实例替换为三个无服务器实例。

1. 登录 AWS 管理控制台 并打开 Amazon DocumentDB 控制台。
2. 在导航窗格中，选择集群。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标
   > (≡)。

   将显示集群表。
3. 在集群表中，选中要向其添加无服务器实例的集群的复选框。
4. 选择 Actions (操作)，然后选择 Add instance (添加实例)。
5. 在添加实例对话框中，为要创建的每个新无服务器实例选择数据库实例类部分中的无服务器。
6. 对于无服务器容量设置，根据对话框中的容量描述设置扩缩配置。
7. （可选）要添加其他实例，请选择添加实例。继续添加实例，直至达到所需的新实例数量。

   在此示例中，创建了三个新无服务器实例。
8. 选择创建。

   创建实例需要几分钟时间。您可以使用控制台或 AWS CLI 查看实例的状态。有关更多信息，请参阅监控 Amazon DocumentDB 集群的状态。
9. 返回至集群表，选中所有三个原始预置实例的复选框。
10. 选择操作，然后选择删除。

在删除过程中，将自动执行失效转移，以将剩余的实例之一提升为写入器。数分钟后删除过程完成。现有集群现在包含三个 DocumentDB 无服务器实例（如大小列中所定义）。

Using the AWS CLI

以下 AWS CLI 配置示例显示了使用运行 Amazon DocumentDB 5.0.0 的 Amazon DocumentDB 预配置集群的迁移过程，该集群不需要升级引擎版本即可开始使用 DocumentDB 无服务器。集群名为 `sample`，从名为 `sample`、`sample2` 和 `sample3` 的三个预置实例开始。在此示例中，将这三个实例替换为三个无服务器实例。集群名为 `sample-cluster`，从名为 `sample-provisioned-instance-1` 和 `sample-provisioned-instance-2` 的两个预置实例（一个写入器实例和一个读取器实例）开始。

在以下示例中，*user input placeholder*使用您自己的信息或配置参数替换每个示例。

使用 `aws docdb describe-db-clusters` 操作确定集群的状态。以下代码将查找集群 `sample-cluster` 的状态并以表格形式输出结果：

```
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             |
+---------------------------------+---------------------------------
+------------------+----------------+
```

添加集群的扩缩配置：

```
aws docdb modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16
```

添加无服务器实例。在此示例中，添加了名为 sample-serverless-instance-1 和 sample-serverless-instance-2 的新无服务器实例：

```
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
```

输入以下内容，等待无服务器实例可用后再继续：

```
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
```

执行失效转移以使新的 sample-serverless-instance-1 实例成为集群写入器。

```
aws docdb failover-db-cluster \
    --db-cluster-identifier sample-cluster \
    --target-db-instance-identifier sample-serverless-instance-1
```

失效转移需要几秒钟才能完成，之后 sample-serverless-instance-1 将成为集群写入器。使用以下输入来验证这一点：

```
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             |
+---------------------------------+---------------------------------
+------------------+----------------+
```

最后，删除原始的预置实例：

```
aws docdb delete-db-instance \
    --db-instance-identifier sample-provisioned-instance-1

aws docdb delete-db-instance \
    --db-instance-identifier sample-provisioned-instance-2
```

从 MongoDB 迁移到 DocumentDB 无服务器

您可以将 MongoDB 数据库迁移到 DocumentDB 无服务器，就像使用预置 Amazon DocumentDB 一样。有关更多信息，请参阅 迁移到 Amazon DocumentDB。

管理 Amazon DocumentDB 无服务器

查看和修改集群的扩缩容量范围配置

`ServerlessV2ScalingConfiguration` 参数指定 DocumentDB 无服务器实例的扩缩容量范围。由适用于集群中所有 DocumentDB 无服务器实例的最小和最大 DocumentDB 容量单位（DCU）值组成。

- **`MinCapacity`**：集群中任何 DocumentDB 无服务器实例的最小扩缩容量。
- **`MaxCapacity`**：集群中任何 DocumentDB 无服务器实例的最大扩缩容量。

> **Note**
>
> 以下扩缩配置修改需要重新启动实例以反映新的 `MinCapacity` 和 `MaxCapacity` 值：
>
> - 对 `MaxCapacity` 值的任何更改
> - 将 `MinCapacity` 值从较高的值更改为 1.0 或更低值
> - 将 `MinCapacity` 值从较低的值更改为大于 1.0 的值

有关扩缩配置以及如何选择适当的扩缩容量限制的更多信息，请参阅 Amazon DocumentDB 无服务器扩缩配置。

Using the AWS 管理控制台

以下 AWS 管理控制台 配置示例显示了如何查看和编辑 DocumentDB 无服务器集群的扩展配置设置。

1. 登录 AWS 管理控制台 并打开 Amazon DocumentDB 控制台。
2. 在导航窗格中，选择集群。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标
   > (≡)。

   将显示集群表。
3. 在集群表中，选中要修改扩缩容量的集群的复选框。
4. 选择 Actions (操作)，然后选择 Modify (修改)。

   将显示修改集群对话框。
5. 找到无服务器容量设置部分，根据对话框中的容量描述设置扩缩配置（容量范围）。

   有关扩缩和容量范围的更多信息，请参阅 Amazon DocumentDB 无服务器扩缩配置。
6. 选择继续。
7. 对于修改计划，选择立即应用。
8. 选择修改集群。

9. 修改完成后，应重启每个无服务器实例。要最大限度地减少写入器不可用情况，请执行以下操作序列：

   a. 重启每个无服务器读取器实例。

      i. 选择读取器实例，选择操作，然后选择重启。

      ii. 等待实例状态恢复为可用。

   b. 对重启的无服务器实例执行失效转移。

      i. 选择集群，选择操作，然后选择失效转移。

      ii. 等待失效转移操作完成。

   c. 重启剩余的无服务器实例。

      i. 选择剩余的实例，选择操作，然后选择重启。

      ii. 等待实例状态恢复为可用。

Using the AWS CLI

以下 AWS CLI 配置示例显示了当前的扩展配置。

在以下示例中，*user input placeholder*使用您自己的信息或配置参数替换每个示例。

可以使用以下`describe-db-clusters` AWS CLI 命令查看集群的当前扩展配置：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[0].ServerlessV2ScalingConfiguration'
```

此命令的输出如下：

```
{
    "MinCapacity": 0.5,
    "MaxCapacity": 16.0
}
```

可以使用 `modify-db-cluster` 命令修改集群的扩缩配置：

```
aws docdb modify-db-cluster \
```

```
    --db-cluster-identifier sample-cluster \
    --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=32
```

完成后，应重启每个无服务器实例。要最大限度地减少写入器不可用情况，可以执行以下操作序列：

```
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
```

查看和修改无服务器读取器的提升层

对于包含多个 DocumentDB 无服务器实例或混用预置实例和无服务器实例的集群，请注意每个无服务器实例的提升层设置。此设置控制的无服务器实例行为比预置实例更多。

对于预置实例，0-15 层的选择仅决定在失效转移操作期间 Amazon DocumentDB 选择将读取器实例提升为写入器的顺序。但是，对于无服务器实例，层编号还决定实例是扩展以匹配写入器实例的容量，还是根据自己的工作负载独立扩展。第 0 层或第 1 层的无服务器读取器实例的最小容量至少与写入器实例一样高。这样，读取器实例就可以准备好在发生失效转移的情况下接管写入器实例。如果写入器实例是预置实例，Amazon DocumentDB 会估算等效的 DocumentDB 无服务器容量。其使用该估计值作为无服务器读取器实例的最小容量。

第 2-15 层中的 DocumentDB 无服务器读取器实例对其最小容量没有这样的限制。当其处于空闲状态时，可以缩减到集群的容量范围中指定的最小 DocumentDB 容量单位（DCU）值。

Using the AWS 管理控制台

以下 AWS 管理控制台 配置示例显示了如何查看和修改 DocumentDB 无服务器实例读取器的促销等级设置。

1. 登录 AWS 管理控制台并 打开 Amazon DocumentDB 控制台。

   每个实例的提升层都显示在 AWS 管理控制台的提升层列中。
2. 在导航窗格中，选择集群。

   > **Tip**
   >
   > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

   将显示集群表。
3. 在集群表中，选中要修改提升层的实例的复选框。
4. 选择 Actions (操作)，然后选择 Modify (修改)。

   将显示修改实例对话框。
5. 找到失效转移部分，将提升层设置为所需级别。
6. 选择继续。
7. 对于修改计划，选择立即应用。
8. 选择修改实例。

Using the AWS CLI

以下 AWS CLI 配置示例显示了当前的扩展配置。

在以下示例中，*user input placeholder*使用您自己的信息或配置参数替换每个示例。

可以使用`describe-db-clusters` AWS CLI 以下命令查看集群中所有实例的升级等级：

```
aws docdb describe-db-clusters \
    --db-cluster-identifier sample-cluster \
    --query 'DBClusters[0].DBClusterMembers' \
    --output table
```

此命令的输出如下：

```
-----------------------------------------------------------------------------------------------
|                                          DescribeDBClusters
                    |
+---------------------------------+---------------------------------
+------------------+----------------+
|  DBClusterParameterGroupStatus |      DBInstanceIdentifier       | IsClusterWriter
  | PromotionTier  |
+---------------------------------+---------------------------------
+------------------+----------------+
|  in-sync                        |  sample-serverless-instance-2   |  False
  |  1             |
|  in-sync                        |  sample-serverless-instance-1   |  True
  |  1             |
+---------------------------------+---------------------------------
+------------------+----------------+
```

可以使用 `modify-db-instance` 命令修改特定实例的提升层：

```
aws docdb modify-db-instance \
    --db-instance-identifier sample-serverless-instance-2 \
    --promotion-tier 3
```

Amazon DocumentDB 无服务器实例限制

对于 DocumentDB 无服务器实例，以下每个实例限制取决于实例当前的扩缩容量：

- 实例内存 (GiB)
- 连接（全部）
- 光标限制
- 打开事务。
- 连接（活动）

以下各表描述了无服务器实例的每个实例限制如何根据实例当前的扩缩容量进行扩缩。

请注意，限制值会有所不同，具体取决于集群的 `MinCapacity` 扩缩配置是否已设置为大于二（2）。如果将 `MinCapacity` 设置为小于或等于 2，则光标限制、未完成事务和连接（活动）的限制值上限为

较低的最大值。（设置上限的技术原因在于支持缩减到“空闲”状态。）有关更多信息，请参阅 空闲状态（0.5 个 DCU）。

如果 ≤1，db.serverless 实例将受到限制 MinCapacity

当前容量（DCU）	实例内存 (GiB)	连接（全部）	光标限制	打开事务。	连接（活动）
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

如果大于 1，db.serverless 实例将受到限制 MinCapacity

当前容量（DCU）	实例内存 (GiB)	连接（全部）	光标限制	打开事务。	连接（活动）
1.5	3	7250	18	36	124
2	4	1000	24	48	173
4	8	2000	48	96	416
8	16	4000	96	192	1071

当前容量（DCU）	实例内存 (GiB)	连接（全部）	光标限制	打开事务。	连接（活动）
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

您可以使用以下 CloudWatch 指标监控每个实例的限制并发出警报。有关亚马逊 DocumentDB CloudWatch 指标的更多信息，请参阅。使用以下方式监控亚马逊 DocumentDB CloudWatch

资源	CloudWatch 限制指标	CloudWatch 使用量指标（最大 1 分钟）	CloudWatch 使用量指标
实例内存	-	-	FreeableMemory
连接（全部）	DatabaseConnectionsLimit	DatabaseConnectionsMax	DatabaseConnections
游标	DatabaseCursorsLimit	DatabaseCursorsMax	DatabaseCursors
事务	TransactionsOpenLimit	TransactionsOpenMax	TransactionsOpen

Amazon DocumentDB 无服务器扩缩配置

主题

- 为 DocumentDB 无服务器集群选择扩缩容量范围
- 为 DocumentDB 无服务器集群选择 MinCapacity 设置
- 为 DocumentDB 无服务器集群选择 MaxCapacity 设置
- 避免内存不足错误

- 为什么我的无服务器实例无法缩减？

为 DocumentDB 无服务器集群选择扩缩容量范围

在将任何 DocumentDB 无服务器实例添加到 Amazon DocumentDB 集群之前，必须先为集群设置 `ServerlessV2ScalingConfiguration` 参数。

`ServerlessV2ScalingConfiguration` 参数由两个值组成，这两个值定义了集群中任何无服务器实例的无服务器扩缩容量范围：

- **`MinCapacity`**：集群中任何 DocumentDB 无服务器实例的最小扩缩容量。
- **`MaxCapacity`**：集群中任何 DocumentDB 无服务器实例的最大扩缩容量。

为 DocumentDB 无服务器集群选择 `MinCapacity` 设置

最好是始终为 `MinCapacity` 选择 0.5。该值允许实例在完全空闲时缩减至最小容量，同时保持活动状态。但是，根据您使用该集群的方式和配置的其他设置，采用一个不同的最小容量可能更有效。选择最小容量设置时，请考虑以下因素：

- DocumentDB 无服务器实例的扩缩率取决于其当前容量。当前容量越大，扩展速度就越快。如果您需要实例快速扩展到非常高的容量，请考虑将最小容量设置为其扩缩率可满足要求的值。
- 如果您通常在预期工作负载特别高或特别低的情况下修改实例的实例类，则可以利用这个经验粗略估计等效的 DocumentDB 无服务器容量范围。要确定预置 Amazon DocumentDB 实例类型的内存大小，请参阅 实例限制。

  例如，假设您在集群的工作负载较低时使用 `db.r6g.xlarge` 实例类。该实例类具有 32GiB 的内存。因此，您可以将 `MinCapacity` 指定为 16，以设置一个可以缩减至大约相同容量的无服务器实例。那是因为每个 DCU 对应大约 2GiB 的内存。如果 `db.r6g.xlarge` 实例有时未充分利用，您可以指定一个稍低的值，以便进一步缩减实例。
- 如果当实例在缓冲区缓存中有一定量的数据时，应用程序的运行效率最高，请考虑指定一个最小 DCU 设置，让内存足够大，可以容纳经常访问的数据。否则，当无服务器实例缩减至较小的内存大小时，一些数据会从缓冲区缓存中移出。然后，随着时间的推移，当实例再扩展回来时，信息会读回到缓冲区缓存中。如果 I/O 要将数据带回缓冲区缓存的数量很大，则选择更高的最小 DCU 值可能会更有效。有关更多信息，请参阅 实例大小调整。
- 如果您的 DocumentDB 无服务器实例大部分时间都以特定容量运行，请考虑指定低于该基准但不要太低的最小容量设置。当前容量没有远低于所需容量时，无服务器实例可以最有效地估计扩展的规模和速度。

- 如果您的预置工作负载的内存要求对于 T3 或 T4g 等小型实例类而言太高，请选择可提供与 R5 或 R6g 实例的内存相当的最低 DCU 设置。
- 特别是，我们建议在使用指定功能时使用以下最低 MinCapacity（这些建议可能会发生变化）：
  - 性能详情：2 个 DCU
  - 全局集群 — 8 个 DCU（仅适用于主 AWS 区域集群）
- 在 Amazon DocumentDB 中，复制发生在存储层，因此读取器容量不会直接影响复制。但是，对于独立扩展的 DocumentDB 无服务器读取器实例，请确保在有密集写入操作期间，最低容量足以处理工作负载，从而避免查询延迟。如果提升层 2-15 的读取器实例遇到性能问题，请考虑增大集群的最小容量。有关更改读取器实例是随写入器扩缩还是独立扩缩的详细信息，请参阅 查看和修改无服务器读取器的提升层。

  如果您的集群包含 DocumentDB 无服务器读取器实例，则当读取器的提升层不是 0 或 1 时，读取器不会随写入器实例一起扩展。在这种情况下，设置较低的最小容量会导致复制滞后过大。这是因为在数据库忙碌时，读取器可能没有足够的容量来应用来自写入器的更改。建议您将最小容量设置为一个值，该值表示与写入器实例相当的内存量和 CPU 量。
- DocumentDB 无服务器实例从最小容量扩展到最大容量所花的时间取决于其最小和最大 DCU 值之间的差异。与实例从小容量开始的情况相比，当实例的当前容量很大时，DocumentDB 无服务器会以更大的增量扩展。因此，如果您指定了相对较大的最大容量，且实例的大部分运行时间都接近该容量，则请考虑增大最小 DCU 设置。这样，空闲实例可以更快地扩展回到最大容量。
- 某些实例限制由无服务器实例的当前容量决定，例如连接限制、光标限制和未完成事务限制。如果实例的当前容量很小，则限制也将相应地很小。如果您的无服务器实例缩减到其 MinCapacity 值时存在这些限制问题，则可以考虑将 MinCapacity 增加到更高的值。有关更多信息，请参阅 Amazon DocumentDB 无服务器实例限制。
- 此外，如果将 MinCapacity 设置为小于或等于 1.0 个 DCU，则某些实例限制的上限为较低的最大值，例如活动连接限制、光标限制和未完成事务限制。如果这些上限限制不足以应付您的工作负载，请使用至少 1.5 个 DCU 的MinCapacity值。有关更多信息，请参阅 Amazon DocumentDB 无服务器实例限制。

有关如何修改集群扩缩配置的说明，请参阅 管理 Amazon DocumentDB 无服务器。

为 DocumentDB 无服务器集群选择 **MaxCapacity** 设置

最好是始终为最大 DocumentDB 无服务器容量设置选择一些较高的值。较大的最大容量让实例可以在运行密集型工作负载时最大限度地扩展。使用较小的值可以避免产生意外费用。根据您使用该集群的方式以及配置的其他设置，最有效的值可能会比原先想象的值更高或更低。选择最大容量设置时，请考虑以下因素：

- 最大容量必须至少与最小容量相同。可以将最小容量和最大容量设置为完全相同。但是，在这种情况下，容量永远不会扩展或缩减。因此，除了在测试状况下，对最小和最大容量使用完全相同的值并不合适。
- 最大容量必须至少为 1.0 个 DCU，并且最多不得超过 256 个 DCU。
- 我们建议监控无服务器实例的扩缩和资源使用情况。如果您的无服务器实例经常扩展到最大容量并达到资源限制（例如，当 `DCUUtilization` 指标为 100.0 时），我们建议您选择更高的 `MaxCapacity` 值。有关更多信息，请参阅 监控 Amazon DocumentDB 无服务器。
- 如果您通常在预期工作负载特别高或特别低的情况下修改预置实例的实例类，则可以利用这个经验估计等效的 DocumentDB 无服务器容量范围。要确定预置 Amazon DocumentDB 实例的内存大小，请参阅 实例限制。

  例如，假设您在集群的工作负载较高时使用 `db.r6g.4xlarge` 实例类。该实例类具有 128GiB 的内存。因此，您可以将最大 DCU 设置指定为 64，从而设置一个可以扩展至大约相同容量的无服务器实例。那是因为每个 DCU 对应大约 2GiB 的内存。如果 `db.r6g.4xlarge` 实例有时没有足够的容量来有效处理工作负载，则可以指定一个稍高的值来让实例进一步扩展。
- 如果您的数据库使用有预算上限，请选择一个即使所有无服务器实例始终以最大容量运行时仍能保持在该上限内的值。请记住，如果您的集群中有 n 个无服务器实例，则该集群在任何时刻可以使用的理论最大无服务器容量是 n 乘以集群的最大 DCU 设置。（例如，如果一些读取器独立于写入器进行扩展，则实际消耗量可能更少。）
- 如果您使用无服务器读取器实例从写入器实例中分流一些只读工作负载，您可以选择较低的最大容量设置。这样做是为了反映这一点，每个读取器实例不需要像集群只包含一个实例那样扩展得那么高。
- 假设您想防止由于应用程序中的数据库参数配置错误或低效查询而导致过度使用。在这种情况下，您可以通过选择一个最大容量设置，使之低于可以设置的绝对最高容量，从而避免意外过度使用。
- 如果由于实际用户活动造成的峰值很少见但确实会发生，那么在选择最大容量设置时可以考虑这些情况。如果优先级是使应用程序以全面的性能和可扩展性保持运行，则可以指定比正常使用情况下的容量更大的最大容量设置。如果在非常极端的活动高峰期间应用程序可以降低吞吐量运行，则可以选择稍低的最大容量设置。确保选择的设置仍然有足够的内存和 CPU 资源以使应用程序保持运行。
- 如果您在集群中启用了增加每个实例内存使用量的设置，请在决定最大 DCU 值时将该内存考虑在内。此类设置包括 Performance Insights 和全局集群的设置。确保最大 DCU 值允许无服务器实例扩展到足以在使用这些功能时处理工作负载。有关排除由于低最大 DCU 设置和增加内存开销的 Amazon DocumentDB 功能组合而引起的问题的信息，请参阅下面的 避免内存不足错误。
- 特别是，我们建议在使用指定功能时使用以下最低 `MaxCapacity`（这些建议可能会发生变化）：
  - 在数据量较大的集群上创建无服务器实例：2 个 DCU（这包括在集群恢复过程中创建无服务器实例。）

- 某些实例限制由实例的当前容量决定，例如连接限制、光标限制和未完成事务限制。在为工作负载选择 `MaxCapacity` 值时，请务必牢记这些实例限制，以避免因其中某项限制而成为瓶颈。有关更多信息，请参阅 Amazon DocumentDB 无服务器实例限制。

有关如何修改集群扩缩配置的说明，请参阅 管理 Amazon DocumentDB 无服务器。

避免内存不足错误

如果您的其中一个 DocumentDB 无服务器实例持续达到最大容量的限制，Amazon DocumentDB 会通过将实例设置为 incompatible-parameters 状态来表明这种状况。实例处于 incompatible-parameters 状态时，某些操作会被阻止。例如，您无法升级引擎版本。有关 Amazon DocumentDB 实例状态的更多信息，请参阅监控 Amazon DocumentDB 实例的状态。

通常，当由于内存不足错误而频繁重新启动时，实例会进入此状态。发生这种类型的重新启动时，Amazon DocumentDB 会记录一个事件。要查看资源事件，请参阅 查看 Amazon DocumentDB 事件。由于开启性能详情等设置会产生开销，因此通常会出现高内存使用率。实例上存在繁重的工作负载，或者要管理与大量架构对象相关联的元数据时，也会导致出现高内存使用率。

如果内存压力变得更低，以致实例不会经常达到最大容量，Amazon DocumentDB 会自动将实例状态更改回可用。

要从这种情况中恢复，您可以采取以下部分或全部操作：

- 通过更改集群的最小 DocumentDB 容量单位（DCU）值，提高无服务器实例的容量下限。这样做可以避免空闲数据库缩减到某个容量，以至于内存太少，无法满足集群中启用的功能所需的内存。更改集群的 DCU 设置后，重启无服务器实例。这样做会评估 Amazon DocumentDB 是否可以将状态重置回可用。
- 通过更改集群的最大 DCU 值，提高无服务器实例的容量上限。这样做可避免繁忙的数据库无法扩展到某个容量，以至于没有足够的内存来满足集群中启用的功能和数据库工作负载所需的内存。更改集群的 DCU 设置后，重启无服务器实例。这样做会评估 Amazon DocumentDB 是否可以将状态重置回可用。
- 关闭需要内存开销的配置设置。例如，假设您已启用性能详情等功能，但尚未使用此功能。如果是这样，您可以将其关闭。或者，您可以将集群的最小和最大容量值调高，以便考虑到这些类型的功能使用的内存。有关选择最小和最大容量设置的指引，请参阅为 DocumentDB 无服务器集群选择扩缩容量范围。
- 减少实例上的工作负载。例如，您可以将读取器实例添加到集群中，以便将只读查询的负载分散到更多实例中。

为什么我的无服务器实例无法缩减？

在某些情况下，即使数据库上没有负载，DocumentDB 无服务器也不会缩减至最小容量。出现这种情况的原因如下：

- 性能详情可以增加资源使用量并防止数据库缩减到最低容量。这些功能如下所示：
- 如果读取器实例未缩减到最小值，并保持在与写入器实例相同或更高的容量，请检查读取器实例的优先级层。第 0 层或第 1 层的 DocumentDB 无服务器读取器实例的最小容量至少与写入器实例一样高。将读取器的优先级层更改为 2 或更高，这样它就可以独立于写入器而纵向扩展和缩减。有关更多信息，请参阅 Amazon DocumentDB 无服务器扩缩。
- 繁重的数据库工作负载可能会增加资源使用量。
- 较大的数据库卷可能会增加资源使用量。Amazon DocumentDB 使用内存和 CPU 资源进行集群管理。Amazon DocumentDB 需要更多的 CPU 和内存来管理具有更大数据库卷的集群。如果您的集群的最小容量低于集群管理所需的最小容量，则您的集群不会缩减到最小容量。
- 后台维护活动可能会定期增加资源使用量。

如果数据库仍无法缩减到配置的最小容量，则停止并重新启动数据库，以回收可能随着时间推移而累积的所有内存碎片。停止和启动数据库会导致停机，因此我们建议谨慎执行此操作。

监控 Amazon DocumentDB 无服务器

要了解有关在 Amazon DocumentDB 中监控的更多信息，请参阅 监控 Amazon DocumentDB。

主题

- 内存不足：incompatible-parameters 状态
- DocumentDB 无服务器的亚马逊 CloudWatch 指标
- 使用性能详情监控 DocumentDB 无服务器性能

内存不足：incompatible-parameters 状态

如果您的其中一个无服务器实例持续达到最大容量的限制，Amazon DocumentDB 会通过将实例设置为 incompatible-parameters 状态来表明这种状况。有关更多信息，请参阅 避免内存不足错误。

DocumentDB 无服务器的亚马逊 CloudWatch 指标

要详细了解如何使用亚马逊 Docu CloudWatch mentDB，请参阅。使用以下方式监控亚马逊 DocumentDB CloudWatch

您可以在中查看您的无服务器实例 CloudWatch ，通过该`ServerlessDatabaseCapacity`指标监控每个实例消耗的容量。您还可以监控所有标准的 DocumentDB CloudWatch 指标，例如`DatabaseConnections`和查询。有关您可以监控的 Amazon DocumentDB CloudWatch 指标的完整列表，请参阅。使用以下方式监控亚马逊 DocumentDB CloudWatch监控以下 CloudWatch 实例级指标非常重要，有助于您了解您的 DocumentDB 无服务器实例是如何向上和向下扩展的。所有这些指标每秒计算一次。这样，您就可以监控无服务器实例的当前状态。您可以设置警报，以便在任何无服务器实例接近与容量相关的指标阈值时通知您。您可以确定最小和最大容量设置是否合适，或者是否需要调整它们。您可以确定将精力集中在哪方面来优化数据库的效率。

- **`ServerlessDatabaseCapacity`**：作为实例级指标，会报告当前实例容量所代表的 DCU 数量。作为集群级指标，它代表集群中所有 DocumentDB 无服务器实例的 `ServerlessDatabaseCapacity` 值的平均值。
- **`DCUUtilization.`**：此指标是 DocumentDB 无服务器中的新指标。此值以百分比表示。其计算方法是 `ServerlessDatabaseCapacity` 指标的值除以集群的最大 DCU 值。解读此指标并采取行动时考虑以下准则：
  - 如果此指标接近值 `100.0`，则实例已扩展到能达到的最大容量。考虑增大集群的最大 DCU 设置。这样，写入器和读取器实例都可以扩展到更高的容量。
  - 假设只读工作负载导致读取器实例的 `DCUUtilization` 接近 `100.0`，而写入器实例未接近其最大容量。在这种情况下，请考虑向集群添加额外的读取器实例。这样，您可以将工作负载的只读部分分散到更多实例上，从而减少每个读取器实例的负载。
  - 假设您正在运行生产应用程序，其中性能和可扩展性是主要考虑因素。在这种情况下，您可以将集群的最大 DCU 值设置为较高的数字。您的目标是让 `DCUUtilization` 指标始终低于 100.0。使用较高的最大 DCU 值，您可以确信有足够的空间来应对数据库活动中出现的意外高峰。您只需为实际使用的数据库容量付费。
- **`CPUUtilization`**：该指标在 DocumentDB 无服务器中的解读不同于预置实例中的解读。对于 DocumentDB 无服务器，此值是一个百分比，其计算方法是当前使用的 CPU 量除以在集群的最大 DCU 值基础上提供的 CPU 容量。当实例持续使用其 CPU 容量的很大一部分时，Amazon DocumentDB 会自动监控此值并扩展无服务器实例。

  如果此指标接近值 `100.0`，则该实例已达到其最大 CPU 容量。考虑增大集群的最大 DCU 设置。如果读取器实例上的这个指标接近值 `100.0`，请考虑向集群添加额外的读取器实例。这样，您可以将工作负载的只读部分分散到更多实例上，从而减少每个读取器实例的负载。

- **FreeableMemory**：此值表示当 DocumentDB 无服务器实例已扩展到其最大容量时，提供的未使用内存量。对于当前容量低于最大容量的每个 DCU，此值增加大约 2GiB。因此，在实例扩展到其可以达到的最大值之前，此指标不会接近零。

  如果此指标接近值零，则实例已扩展到其可以达到的最大值，并且已接近其可用内存的限制。考虑增大集群的最大 DCU 设置。如果读取器实例上的这个指标接近值零，请考虑向集群添加额外的读取器实例。这样，工作负载的只读部分可以分散到更多实例上，从而减少每个读取器实例的内存使用。

- **TempStorageIops**：在连接到实例的本地存储上完成的 IOPS 数。它包括读取和写入的 IOPS。此指标表示计数，每秒测量一次。这是 DocumentDB 无服务器的新指标。有关更多信息，请参阅 使用以下方式监控亚马逊 DocumentDB CloudWatch。

- **TempStorageThroughput**：与实例关联的本地存储的传入和传出数据量。此指标表示字节数，每秒测量一次。这是 DocumentDB 无服务器的新指标。有关更多信息，请参阅 使用以下方式监控亚马逊 DocumentDB CloudWatch。

通常情况下，DocumentDB 无服务器实例的大多数扩展是因内存使用和 CPU 活动而引起。TempStorageIops 和 TempStorageThroughput 指标可以帮助您诊断实例与本地存储设备之间传输的网络活动导致意外容量增加的罕见情况。要监控其他网络活动，您可以使用以下现有指标：

- NetworkReceiveThroughput
- NetworkThroughput
- NetworkTransmitThroughput
- StorageNetworkReceiveThroughput
- StorageNetworkThroughput
- StorageNetworkTransmitThroughput

DocumentDB 无服务器 CloudWatch 指标如何应用于你的 AWS 物料

AWS 账单上的 DocumentDB 无服务器费用是根据您可以监控的相同ServerlessDatabaseCapacity指标计算的。如果您仅使用了部分小时 DocumentDB 无服务器容量，则计费机制可能与该指标的计算 CloudWatch 平均值不同。如果系统问题导致 CloudWatch 指标在短时间内不可用，情况也可能有所不同。因此，你在账单 DCU-hours 上看到的值可能会与你自己根据ServerlessDatabaseCapacity平均值计算该数字时略有不同。

DocumentDB 无服务器指标的 Amazon CloudWatch CLI 命令示例

以下 AWS CLI 示例演示了如何监控与 DocumentDB 无服务器相关的最重要 CloudWatch 指标。在每种情况下，请将 `--dimensions` 参数的 `Value=` 字符串替换为您自己的 DocumentDB 无服务器实例的标识符。

以下 Linux 示例显示了实例的最小容量、最大容量和平均容量值，在一小时内每 10 分钟测量一次。Linux 日期命令指定相对于当前日期和时间的开始时间和结束时间。`--query` 参数中的 `sort_by` 函数根据 `Timestamp` 字段按时间顺序对结果进行排序。

```
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
```

以下 Linux 示例演示了监控集群中实例的容量。将衡量实例的最小容量、最大容量和平均容量利用率。在三个小时内，每小时进行一次测量。这些示例使用 `DCUUtilization` 指标表示 DCU 上限的百分比，而不是使用 `ServerlessDatabaseCapacity` 表示固定数量的 DCU。这样，您就不需要知道容量范围内的最小和最大 DCU 值的实际数字。您可以看到 0 到 100 之间的百分比。

```
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
```

下面的 Linux 示例执行的测量与之前的示例类似。在本例中，测量值针对 CPUUtilization 指标。在 1 小时内，每 10 分钟进行一次测量。这些数字表示已使用的可用 CPU 的百分比，基于实例的最大容量设置中可用的 CPU 资源。

```
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
```

下面的 Linux 示例执行的测量与之前的示例类似。在本例中，测量值针对 FreeableMemory 指标。在 1 小时内，每 10 分钟进行一次测量。

```
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
```

使用性能详情监控 DocumentDB 无服务器性能

您可以使用性能详情来监控 DocumentDB 无服务器实例的性能。有关性能详情的程序，请参阅使用 Performance Insights 进行监控。

以下新性能详情计数器适用于 DocumentDB 无服务器实例：

- **os.general.serverlessDBCapacity**：实例的当前容量，以 DCU 为单位。该值对应于实例的ServerlessDatabaseCapacity CloudWatch 指标。

- **os.general.dcuUtilization**：当前容量占最大配置容量的百分比。该值对应于实例的DCUUtilization CloudWatch 指标。
- **os.general.maxConfiguredDcu**：您为此 DocumentDB 无服务器实例配置的最大容量。以 DCU 为单位。
- **os.general.minConfiguredDcu**：您为此 DocumentDB 无服务器实例配置的最小容量。以 DCU 为单位。

有关性能详情计数器的完整列表，请参阅 Performance Insights 的计数器指标。

在性能详情中为 DocumentDB 无服务器实例显示 vCPU 值时，这些值表示基于实例的 DCU 值的估计值。默认时间间隔为 1 分钟，任何小数 vCPU 值将向上舍入到最接近的整数。对于更长的时间间隔，显示的 vCPU 值是每分钟的整数 vCPU 值的平均值。

使用 Amazon DocumentDB 开发

以下各节介绍使用 Amazon DocumentDB 进行开发。

主题

- 连接到 Amazon DocumentDB
- 使用 Amazon DocumentDB 进行编程

连接到 Amazon DocumentDB

Amazon DocumentDB 提供多种连接选项，包括使用 MongoDB 驱动程序进行编程连接、通过 SSH 隧道或 VP AWS N 从 VPC 外部进行连接，以及 Studio 3T 等热门工具。 DataGrip该服务支持副本集连接以实现高可用性。针对分析和报告，您可以使用 JDBC 或 ODBC 驱动程序进行连接。同一 VPC 中的 EC2 实例可以直接连接到 DocumentDB 集群。所有连接都需要 TLS 和适当的连接 AWS credentials/authentication。

主题

- 以编程方式连接到 Amazon DocumentDB
- 从 Amazon VPC 外部连接到 Amazon DocumentDB 集群
- 作为副本集连接到 Amazon DocumentDB
- 从 Studio 3T 连接到 Amazon DocumentDB 集群
- 使用连接亚马逊 DocumentDB DataGrip
- 使用 Amazon EC2 进行连接
- 使用 Amazon DocumentDB JDBC 驱动程序进行连接
- 使用 Amazon DocumentDB ODBC 驱动程序进行连接

以编程方式连接到 Amazon DocumentDB

本部分包含说明了如何使用多种不同语言连接到 Amazon DocumentDB（与 MongoDB 兼容）的代码示例。根据连接的集群是否启用传输层安全性 (TLS)，这些示例分为两个部分。默认情况下，Amazon DocumentDB 集群启用了 TLS。但是，您可以根据需要关闭 TLS。有关更多信息，请参阅 加密传输中数据。

如果您尝试从集群所在的 VPC 外部连接到您的 Amazon DocumentDB，请参阅。从 Amazon VPC 外部连接到 Amazon DocumentDB 集群

在连接到集群之前，您必须知道集群是否启用了 TLS。下一部分介绍如何使用 AWS 管理控制台 或 AWS CLI确定集群的 `tls` 参数的值。之后，您可以查找和应用适当的代码示例。

主题

- 确定 tls 参数的值
- 启用了 TLS 的情况下的连接
- 禁用了 TLS 的情况下的连接

确定 tls 参数#值

确定您的集群是否启用了 TLS 是一个两步过程，您可以使用 AWS 管理控制台 或 AWS CLI来执行。

1. 确定管理集群的参数组。

   Using the AWS 管理控制台

   1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
   2. 在左侧导航窗格中，选择集群。
   3. 在集群列表中，选择您的集群的名称。
   4. 生成的页面将显示所选集群的详细信息。选择 Configuration（配置）选项卡。在配置和状态部分中，在集群参数组下找到参数组的名称。

   Using the AWS CLI

   以下 AWS CLI 代码确定哪个参数控制您的集群。请确保将 *sample-cluster* 替换为您的集群的名称。

   ```
   aws docdb describe-db-clusters \
       --db-cluster-identifier sample-cluster \
       --query 'DBClusters[*].[DBClusterIdentifier,DBClusterParameterGroup]'
   ```

   此操作的输出将类似于以下内容：

   ```
   [
   ```

```
        [
            "sample-cluster",
            "sample-parameter-group"
        ]
]
```

2. 确定您的集群参数组中的 **tls** 参数的值。

   Using the AWS 管理控制台

   1. 在导航窗格中，选择参数组。
   2. 在集群参数组窗口中，选择您在步骤 1d 中创建的集群参数组名称。
   3. 打开的页面上会显示您的集群参数组中包含的参数。您可以在其中查看 `tls` 参数的值。有关修改此参数的信息，请参阅修改 Amazon DocumentDB 集群参数组。

   Using the AWS CLI

   您可以使用`describe-db-cluster-parameters` AWS CLI 命令查看集群参数组中参数的详细信息。

   - **`--describe-db-cluster-parameters`**：列出参数组中的所有参数及其值。
     - **`--db-cluster-parameter-group name`**：必需。您的集群参数组的名称。

   在以下示例中，将每个*`user input placeholder`*示例替换为集群的信息。

```
aws docdb describe-db-cluster-parameters \
    --db-cluster-parameter-group-name sample-parameter-group
```

   此操作的输出将类似于以下内容：

```
{
        "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 DocumentDB 支持以下区域中从 Amazon DocumentDB 5.0（引擎版本 3.0.3727）集群开始的 FIPS 140-3 端点：ca-central-1、us-west-2、us-east-1、us-east-2、us-gov-east-1、us-gov-west-1。

确定 `tls` 参数的值后，即可使用以下部分中的代码示例之一继续连接到您的集群。

- 启用了 TLS 的情况下的连接
- 禁用了 TLS 的情况下的连接

启用了 TLS 的情况下的连接

要查看以编程方式连接到 TLS-enabled Amazon DocumentDB 集群的代码示例，请为要使用的语言选择相应的选项卡。

要加密传输中数据，请使用以下操作下载名为 `global-bundle.pem` 的 Amazon DocumentDB 公钥。

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

如果您的应用程序在 Microsoft Windows 上并且需要 PKCS7 文件，则可以下载 PKCS7 证书捆绑包。此捆绑包包含中间证书和根证书https://truststore.pki.rds.amazonaws.com/global/global-bundle.p7b。

Python

以下代码说明了如何在启用了 TLS 的情况下使用 Python 连接到 Amazon DocumentDB。

在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
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

以下代码演示了在启用 TLS Node.js 时如何使用连接到亚马逊文档数据库。

> **Important**
>
> 版本早于 6.13.1 的 Node.js 驱动程序存在已知限制，Amazon DocumentDB 的 IAM 身份验证目前不支持这些限制。 Node.js 使用驱动程序（例如 mongosh）的 Node.js 驱动程序和工具必须升级为使用 Node.js 驱动程序版本 6.13.1 或更高版本。

在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
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

以下代码说明了如何在启用了 TLS 的情况下使用 PHP 连接到 Amazon DocumentDB。

在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
<?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

以下代码说明了如何在启用了 TLS 的情况下使用 Go 连接到 Amazon DocumentDB。

> **Note**
>
> 从版本 1.2.1 开始，MongoDB Go 驱动程序将仅使用在 `sslcertificateauthorityfile` 中找到的第一个 CA 服务器证书。以下代码示例通过

将在 sslcertificateauthorityfile 中找到的所有服务器证书手动附加到在创建客户端期间使用的自定义 TLS 配置来解决此限制。

在以下示例中，将每个*user input placeholder*示例替换为集群的信息。

```
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

从 Java 应用程序连接到 TLS-enabled Amazon DocumentDB 集群时，您的程序必须使用 AWS提供的证书颁发机构 (CA) 文件来验证连接。要使用 Amazon RDS CA 证书，请执行以下操作：

1. 从 https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem 下载 Amazon RDS CA 文件。
2. 通过执行以下命令，使用该文件中包含的 CA 证书来创建信任存储。请务必将更改*truststore-password*为其他内容。如果您要访问同时包含旧 CA 证书 (`rds-ca-2015-root.pem`) 和新 CA 证书 (`rds-ca-2019-root.pem`) 的信任存储，可以将证书捆绑包导入该信任存储。

   下面是一个示例 Shell 脚本，它将证书捆绑包导入 Linux 操作系统上的信任存储。在以下示例中，用您自己的信息替换每个*user input placeholder* 示例。最值得注意的是，无论示例目录 “*mydir*” 位于脚本中的哪个位置，都要将其替换为为此任务创建的目录。

```
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
```

下面是一个示例 Shell 脚本，它将证书捆绑包导入 macOS 上的信任存储。

```
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. 请先在您的应用程序中设置以下系统属性，以便在该程序中使用 keystore，然后再连接到 Amazon DocumentDB 集群。

```
javax.net.ssl.trustStore: truststore
```

```
javax.net.ssl.trustStorePassword: truststore-password;
```

4. 以下代码说明了如何在启用了 TLS 的情况下使用 Java 连接到 Amazon DocumentDB。

   在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
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

以下代码说明了如何在启用了 TLS 的情况下使用 C# / .NET 连接到 Amazon DocumentDB。

在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
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

以下代码演示了如何在启用 TLS 的情况下，使用最新版本 mongosh 或先前的 mongo Shell 版本连接和查询 Amazon DocumentDB。

使用 mongosh 连接到 Amazon DocumentDB

> **Important**
>
> 版本早于 6.13.1 的 Node.js 驱动程序存在已知限制，Amazon DocumentDB 的 IAM 身份验证目前不支持这些限制。 Node.js 使用驱动程序（例如 mongosh）的 Node.js 驱动程序和工具必须升级为使用 Node.js 驱动程序版本 6.13.1 或更高版本。

在以下示例中，将每个*user input placeholder*示例替换为集群的信息。

```
mongosh --tls --host cluster-end-point:27017 --tlsCAFile  global-bundle.pem --
username sample-user --password password --retryWrites false
```

使用先前的 mongo Shell 版本连接到 Amazon DocumentDB

如果您使用 IAM，则必须使用先前版本的 mongo Shell。输入以下命令选项之一：

```
mongo --ssl --host cluster-end-point:27017 --sslCAFile global-bundle.pem --
username sample-user --password password
```

如果您使用的版本等于或高于 4.2，请使用以下代码进行连接。Amazon DocumentDB 不支持可重试写入。如果您使用旧版 mongo Shell（而非 mongosh），不要在任何代码字符串中包含 `retryWrites=false` 命令。默认情况下，禁用可重试写入。包含 `retryWrites=false` 可能导致正常读取命令失败。

```
mongo --tls --host cluster-end-point:27017 --tlsCAFile global-bundle.pem --
username sample-user --password password
```

测试连接

1. 插入单个文档。

   ```
   db.myTestCollection.insertOne({'hello':'Amazon DocumentDB'})
   ```

2. 查找以前插入的文档。

   ```
   db.myTestCollection.find({'hello':'Amazon DocumentDB'})
   ```

R

以下代码说明了如何在启用了 TLS 的情况下使用 mongolite (https://jeroen.github.io/mongolite/) 通过 R 连接到 Amazon DocumentDB。

在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
#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

以下代码说明了如何在启用了 TLS 的情况下使用 Ruby 连接到 Amazon DocumentDB。

在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
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
```

禁用了 TLS 的情况下的连接

要查看以编程方式连接到 TLS-disabled Amazon DocumentDB 集群的代码示例，请选择要使用的语言选项卡。

Python

以下代码说明了如何在禁用 TLS 的情况下使用 Python 连接到 Amazon DocumentDB。

在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
## 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

以下代码演示了在禁用 TLS Node.js 时如何使用连接到 Amazon DocumentDB。

> **Important**
>
> 版本早于 6.13.1 的 Node.js 驱动程序存在已知限制，Amazon DocumentDB 的 IAM 身份验证目前不支持这些限制。 Node.js 使用驱动程序（例如 mongosh）的 Node.js 驱动程序和工具必须升级为使用 Node.js 驱动程序版本 6.13.1 或更高版本。

在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
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

以下代码说明了如何在禁用了 TLS 的情况下使用 PHP 连接到 Amazon DocumentDB。

在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
<?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

以下代码说明了如何在禁用了 TLS 的情况下使用 Go 连接到 Amazon DocumentDB。

在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
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

以下代码说明了如何在禁用了 TLS 的情况下使用 Java 连接到 Amazon DocumentDB。

在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
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

以下代码说明了如何在禁用了 TLS 的情况下使用 C# / .NET 连接到 Amazon DocumentDB。

在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
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

以下代码演示了如何在禁用 TLS 的情况下，使用最新版本 mongosh 或先前的 mongo Shell 版本连接和查询 Amazon DocumentDB。

使用 mongosh 连接到 Amazon DocumentDB

> **Important**
>
> 版本早于 6.13.1 的 Node.js 驱动程序存在已知限制，Amazon DocumentDB 的 IAM 身份验证目前不支持这些限制。 Node.js 使用驱动程序（例如 mongosh）的 Node.js 驱动程序和工具必须升级为使用 Node.js 驱动程序版本 6.13.1 或更高版本。

在以下示例中，将每个*user input placeholder*示例替换为集群的信息。

```
mongosh --host cluster-end-point:27017 --username sample-user --password password --
retryWrites false
```

使用先前的 mongo Shell 版本连接到 Amazon DocumentDB

如果您使用 IAM，则必须使用先前版本的 mongo Shell。输入以下命令选项之一：

```
mongo --host cluster-end-point:27017 --username sample-user --password password
```

如果您使用的版本等于或高于 4.2，请使用以下代码进行连接。Amazon DocumentDB 不支持可重试写入。如果您使用旧版 mongo Shell（而非 mongosh），不要在任何代码字符串中包含 `retryWrites=false` 命令。默认情况下，禁用可重试写入。包含 `retryWrites=false` 可能导致正常读取命令失败。

```
mongo --host cluster-end-point:27017 --username sample-user --password password
```

测试连接

1. 插入单个文档。

   ```
   db.myTestCollection.insertOne({'hello':'Amazon DocumentDB'})
   ```

2. 查找以前插入的文档。

   ```
   db.myTestCollection.find({'hello':'Amazon DocumentDB'})
   ```

R

以下代码说明了如何在禁用了 TLS 的情况下使用 mongolite (https://jeroen.github.io/mongolite/) 通过 R 连接到 Amazon DocumentDB。

在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
#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

以下代码说明了如何在禁用了 TLS 的情况下使用 Ruby 连接到 Amazon DocumentDB。

在以下示例中，将每个*user input placeholder*替换为集群的信息。

```
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
```

从 Amazon VPC 外部连接到 Amazon DocumentDB 集群

Amazon DocumentDB（与 MongoDB 兼容）集群部署在 Amazon Virtual Private Cloud (Amazon VPC) 中。部署在同一 Amazon VPC 中的 Amazon EC2 实例或其他 AWS 服务可以直接访问它们。此外，同一 AWS 区域 区域或其他区域的不同 VPC 中的 EC2 实例或其他 AWS 服务可以通过 VPC 对等互连访问 Amazon DocumentDB。

但是，假设您的使用案例要求您（或您的应用程序）从集群的 VPC 外部访问您的 Amazon DocumentDB 资源。在这种情况下，您可以使用 SSH 隧道（也称为端口转发）访问您的 Amazon DocumentDB 资源。

深入讨论 SSH 隧道超出了本主题的范围。有关 SSH 隧道的更多信息，请参阅以下内容：

- SSH 隧道
- SSH 端口转发示例，尤其是本地转发部分

要创建 SSH 隧道，您需要一个与您的 Amazon DocumentDB 集群在同一 Amazon VPC 中运行的 Amazon EC2 实例。您可以使用同一 VPC 中的现有 EC2 实例作为集群，或创建一个集群。有关更多信息，请参阅适合您的操作系统的主题：

- Amazon EC2 Linux 实例入门
- Amazon EC2 Windows 实例入门

您可能通常会使用以下命令连接到 EC2 实例：

```
ssh -i "ec2Access.pem" ubuntu@ec2-34-229-221-164.compute-1.amazonaws.com
```

如果是这样，您可以通过在本地计算机上运行以下命令设置到 Amazon DocumentDB 集群 `sample-cluster.node.us-east-1.docdb.amazonaws.com` 的 SSH 隧道。`-L` 标志用于转发本地端口。使用 SSH 隧道时，我们建议您使用集群端点连接到集群，而不要尝试以副本集模式（即在连接字符串中指定 `replicaSet=rs0`）进行连接，因为这会导致错误。

```
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
```

创建 SSH 隧道后，您发布到 `localhost:27017` 的任何命令都会转发到在 Amazon VPC 中运行的 Amazon DocumentDB 集群 `sample-cluster`。如果您的 Amazon DocumentDB 集群上启用了传输层安全性协议 (TLS)，则需要从 https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem 下载 Amazon DocumentDB 的公有密钥。以下操作将下载此文件：

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

> **Note**
>
> 默认情况下会对新的 Amazon DocumentDB 集群启用 TLS。但是，您可以将其禁用。有关更多信息，请参阅 管理 Amazon DocumentDB 集群 TLS 设置。

要从 Amazon VPC 外部连接到您的 Amazon DocumentDB 集群，请使用以下命令。

```
mongo --sslAllowInvalidHostnames --ssl --sslCAFile global-bundle.pem --username
 <yourUsername> --password <yourPassword>
```

作为副本集连接到 Amazon DocumentDB

在针对 Amazon DocumentDB（与 MongoDB 兼容）进行开发时，我们建议您以副本集形式连接到集群，并使用驱动程序的内置读取首选项功能将读取操作分布到副本实例。本节将更深入地探讨其含义，并介绍如何使用适用于 Python 的软件开发工具包作为副本集连接到您的 Amazon DocumentDB 集群。

Amazon DocumentDB 有三个可用于连接集群的端点：

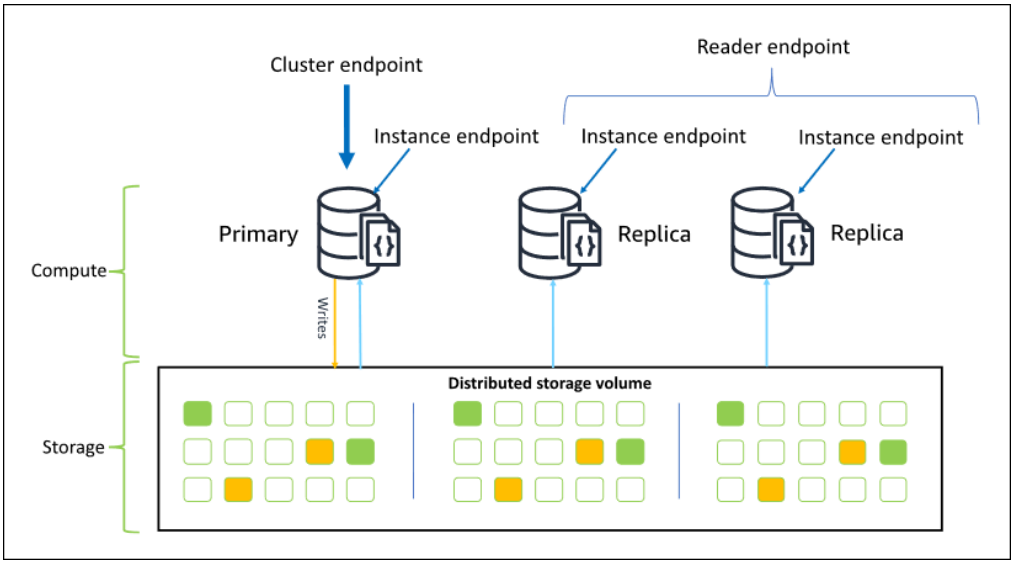
- 集群端点

- 读取器端点
- 实例端点

在大多数情况下，当您连接到 Amazon DocumentDB 时，我们建议使用集群端点。这是指向集群中主实例的 CNAME，如下图所示。

使用 SSH 隧道时，我们建议您使用集群端点连接到集群，而不要尝试以副本集模式（即在连接字符串中指定 `replicaSet=rs0`）进行连接，因为这会导致错误。

> **Note**
>
> 有关 Amazon DocumentDB 网站端点的更多信息，请参阅 Amazon DocumentDB 端点。



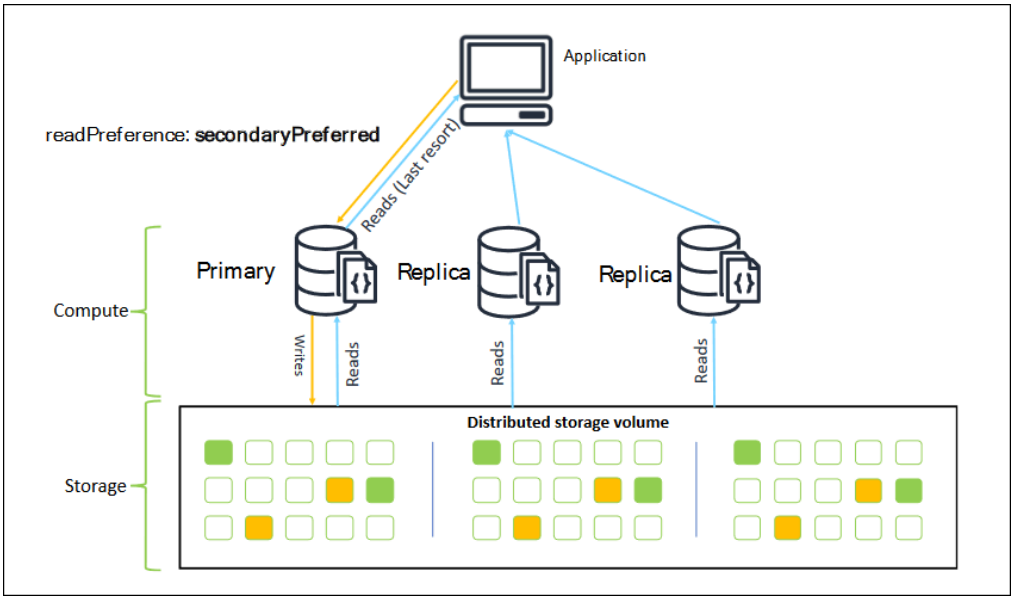
使用集群端点，可以以副本集模式连接到您的集群。然后，您可以使用内置的读取首选项驱动程序功能。在以下示例中，指定 `/?replicaSet=rs0` 表示要作为副本集连接到软件开发工具包。如果省略 `/?replicaSet=rs0'`，则客户端会将所有请求路由到集群端点，即您的主实例。

```
## 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')
```

作为副本集连接的优势在于，它使您的软件开发工具包能够自动发现集群地形，包括何时在集群中增加或移除实例。然后，您可以通过将读取请求路由到副本实例，从而更高效地使用集群。

作为副本集连接时，可以指定连接的 readPreference。如果您将读取首选项指定为 secondaryPreferred，则客户端会将读取查询路由到您的副本，并将写入查询路由到您的主实例（如下图所示）。这样可以更好地利用您的集群资源。有关更多信息，请参阅 读取首选项选项。

```
## 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')
```



从 Amazon DocumentDB 副本中读取的内容具有最终一致性。它们返回数据的顺序与在主实例上写入数据的顺序相同，而且复制延迟时间通常不到 50 毫秒。您可以使用 Amazon CloudWatch 指标DBInstanceReplicaLag和监控集群的副本延迟DBClusterReplicaLagMaximum。有关更多信息，请参阅 使用以下方式监控亚马逊 DocumentDB CloudWatch。

与传统的单体数据库架构不同，Amazon DocumentDB 将存储和计算分离开来。鉴于这种现代架构，我们鼓励您在副本实例上进行读取扩展。对副本实例的读取不会阻止从主实例复制写入。您可以在集群中增加多达 15 个只读副本实例，并横向扩展到每秒数百万次读取。

作为副本集连接并将读取分配给副本的主要好处是，它增加了集群中可用于应用程序工作的总体资源。作为最佳实践，我们建议以副本集的形式进行连接。此外，我们最常建议在以下情况下使用它：

- 您在主实例上使用了将近 100% 的 CPU。
- 缓冲区缓存命中率接近零。
- 您已达到单个实例的连接或光标限制。

扩展集群实例大小是一种选择，在某些情况下，这可能是纵向扩展集群的最佳方式。但是，您也应该考虑如何更好地使用集群中已有的副本。这使您可以扩展规模，而不会因为使用更大的实例类型而增加成本。我们还建议您使用警报监控这些限制（即CPUUtilizationDatabaseConnections、和BufferCacheHitRatio）并 CloudWatch 发出警报，以便知道何时资源被大量使用。

有关更多信息，请参阅以下主题：

- Amazon DocumentDB 的最佳实践
- Amazon DocumentDB 配额和限制

使用集群连接

请考虑使用集群中所有连接的情况。例如，r5.2xlarge 实例的连接数限制为 4,500（以及 450 个打开的游标）。如果您创建了三个实例的 Amazon DocumentDB 集群，并且只使用集群端点连接到主实例，则打开的连接和游标的集群限制分别为 4,500 和 450。如果您正在构建的应用程序要使用许多在容器中启动的工作程序，那么可能会达到这些限制。容器同时打开多个连接并使集群饱和。

相反，您可以作为副本集连接到您的 Amazon DocumentDB 集群，并将读取分配给副本实例。然后，您可以有效地将集群中可用连接和光标数量增加三倍，分别达到 13,500 和 1,350。向集群增加更多实例只会增加读取工作负载的连接和光标数量。如果您需要增加写入集群的连接数量，我们建议您增加实例大小。

> **Note**
>
> large、xlarge、和 2xlarge 实例的连接数随实例大小上升到 4,500 而增加。对于 4xlarge 或更大的实例，每个实例的最大连接数为 4,500。有关不同实例类型限制的更多信息，请参阅 实例限制。

一般而言，我们不建议使用 secondary 的读取首选项连接到集群。这是因为如果您的集群中没有副本实例，则会读取失败。例如，假设您具有一个具有两个实例的 Amazon DocumentDB 集群，它具有一个主集和一个副本。如果副本出现问题，则从设置为 secondary 失败的连接池中读取请求。secondaryPreferred 的优点在于，如果客户端找不到合适的副本实例进行连接，则它会回退到主实例进行读取。

多个连接池

在某些情况下，应用程序中的读取需要具有先写后读的一致性，这只能从 Amazon DocumentDB 的主实例提供。在这些情况下，您可以创建两个客户端连接池：一个用于写入，另一个用于需要先写后读一致性的读取。为此，您的代码如下所示。

```
## 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')
```

另一种选择是创建单个连接池，并覆盖给定集合的读取首选项。

```
##Specify the collection and set the read preference level for that collection
col = db.review.with_options(read_preference=ReadPreference.SECONDARY_PREFERRED)
```

Summary

为了更好地使用集群中的资源，我们建议您使用副本集模式连接到集群。如果它适合您的应用程序，则可以通过将读取分配给副本实例来扩展您的应用程序。

从 Studio 3T 连接到 Amazon DocumentDB 集群

Studio 3T 是一款受欢迎的面向使用 MongoDB 的开发人员和数据工程师的 GUI 和 IDE。它提供了多种强大的数据树视图、表格视图和 JSON 视图、CSV、JSON、SQL 视图以及灵活的查询选项 BSON/mongodump、可视化的拖放界面、带自动完成功能的内置 mongo shell、聚合管道编辑器以及 SQL 查询支持。 import/export

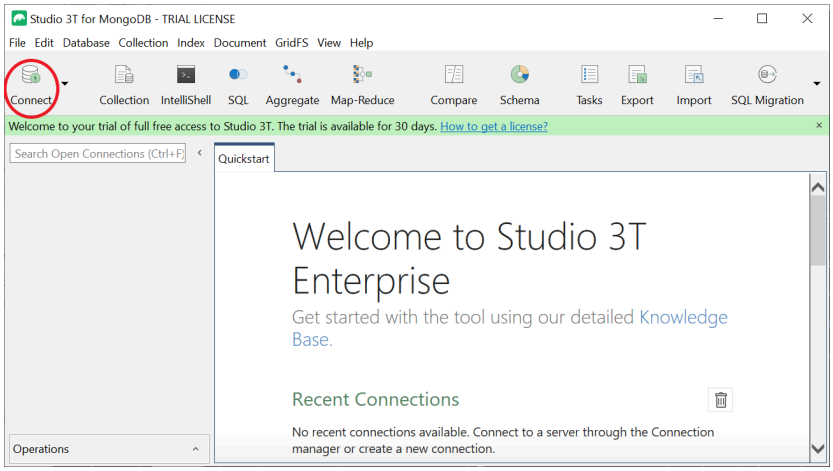
先决条件

- 如果您还没有使用亚马逊 EC2 作为 bastion/jump 主机的亚马逊 DocumentDB 集群，请按照有关如何连接亚马逊 EC2 的说明进行操作。

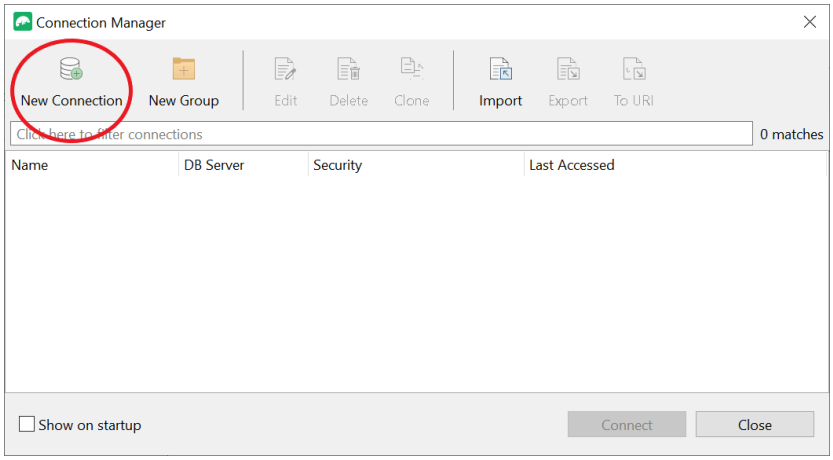
- 如果你没有 Studio 3T，请下载并安装它。

用 Studio 3T 连接

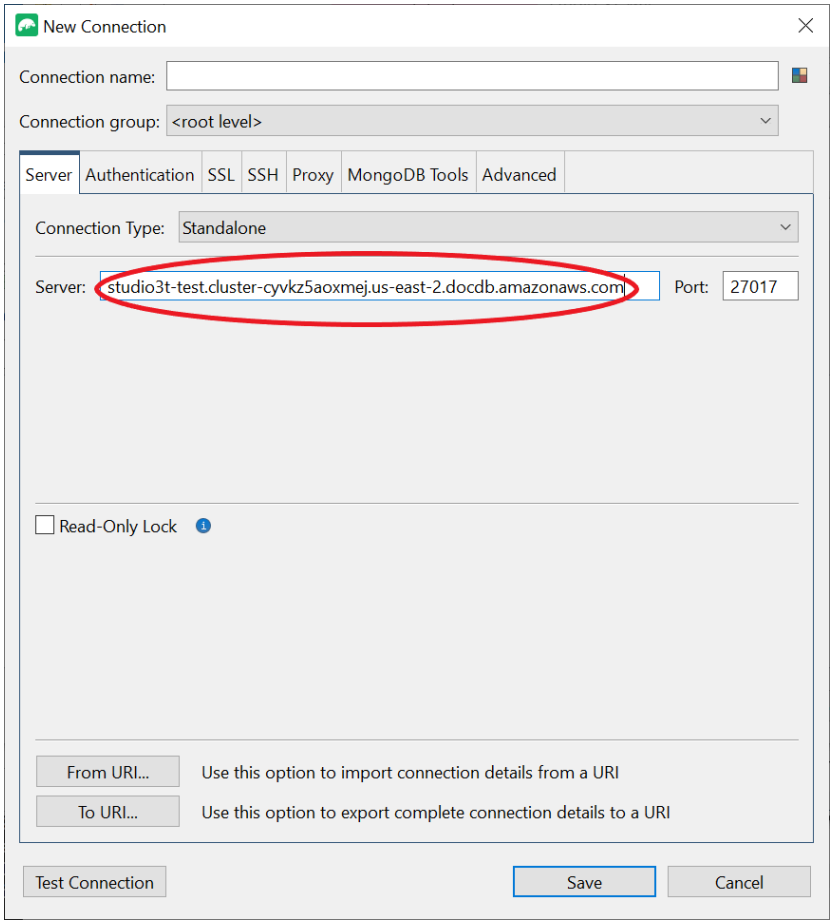
1. 选择工具栏左上角的连接。



2. 选择工具栏左上角的新连接。

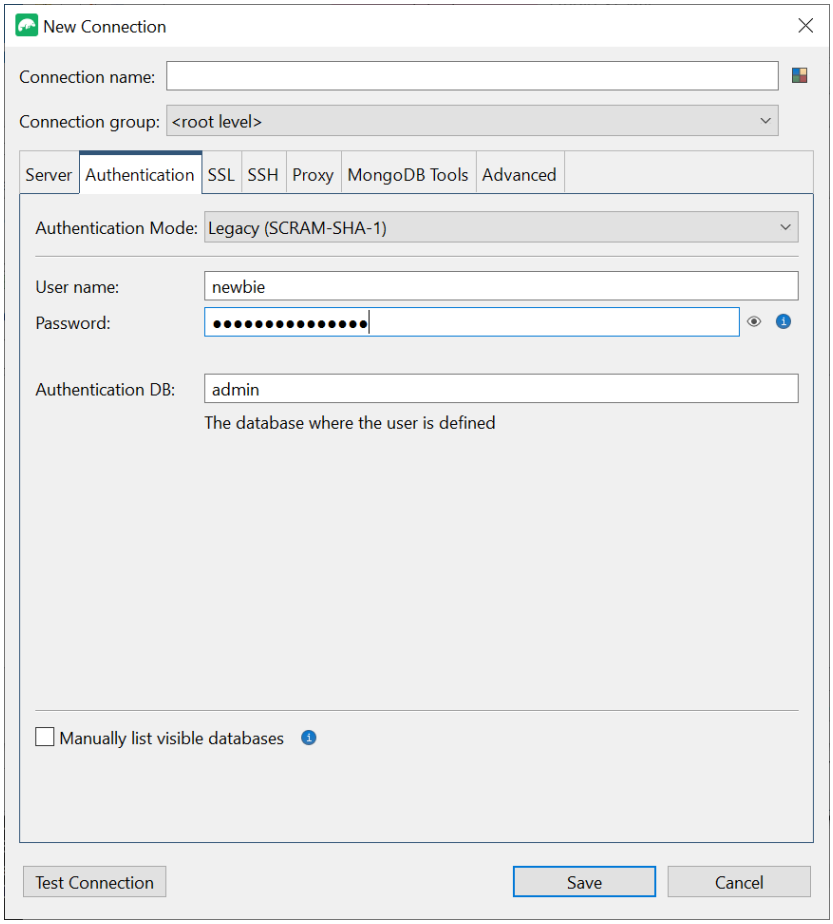


3. 在服务器选项卡，在服务器字段中输入集群端点信息。

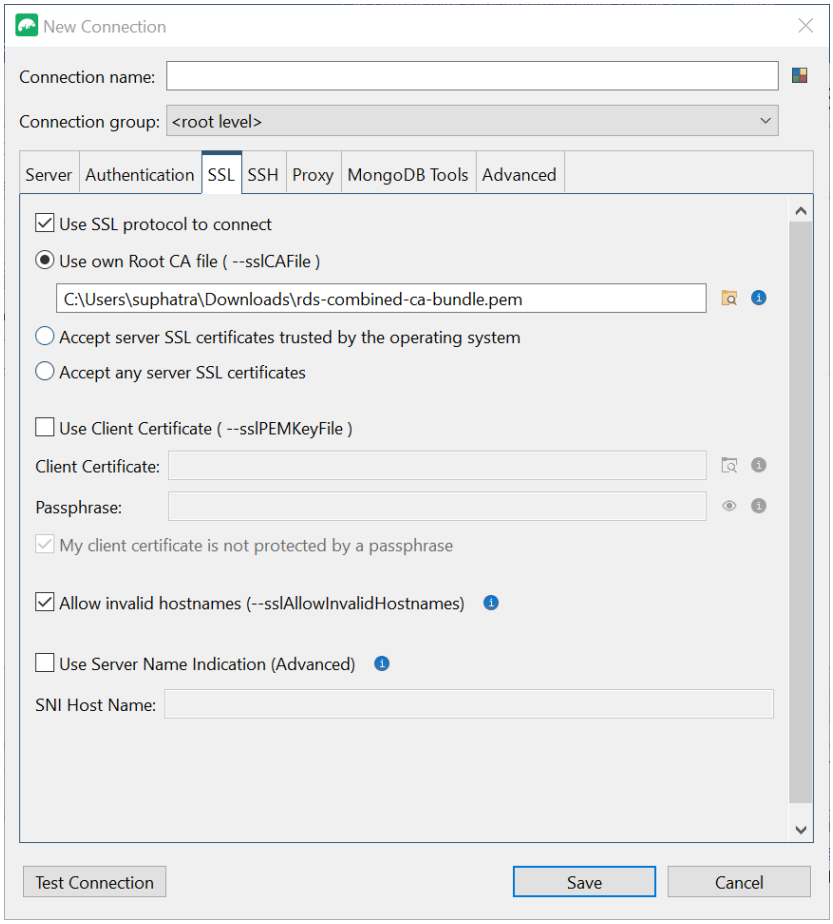


> **Note**
>
> 无法找到您的集群端点？ 只要遵循此处步骤即可。

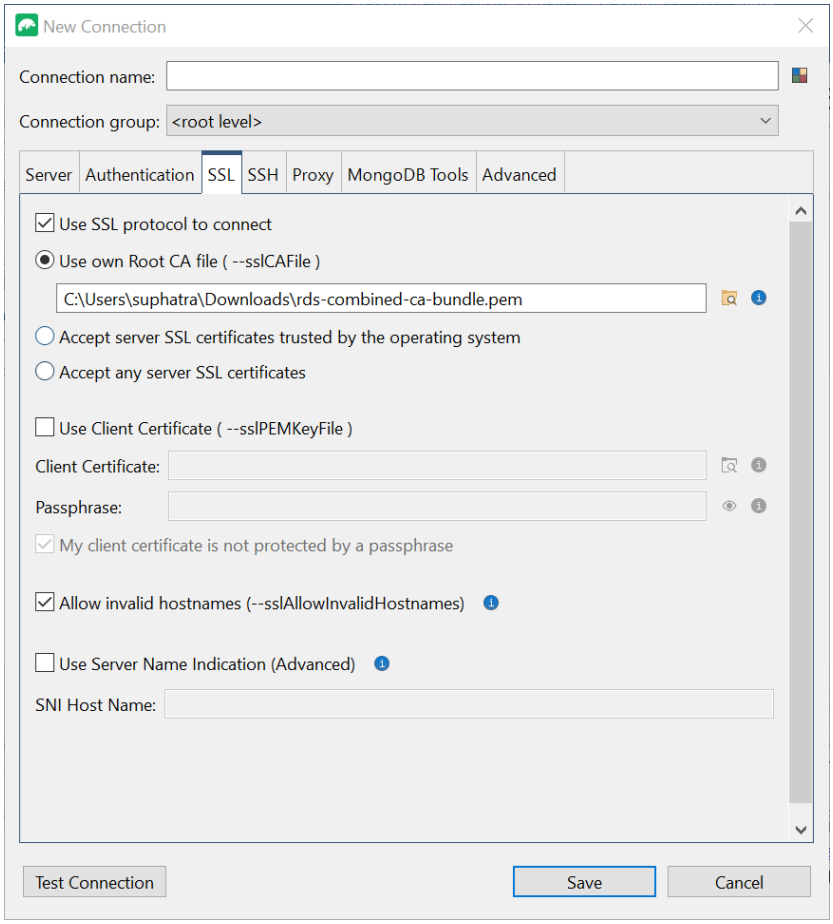
4. 选择身份验证选项卡，并且在身份验证模式的下拉菜单中选择旧版。



5. 在用户名和密码字段中输入您的用户名和凭证。
6. 选择SSL选项卡，然后勾选使用 SSL 协议连接复选框。



7. 选择使用自己的根 CA 文件。然后添加 Amazon DocumentDB 证书（如果 SSL 在您的 DocumentDB 集群上已禁用，则可以跳过此步骤）。勾选该复选框以允许无效主机名。



> **Note**
>
> 没有证书？ 您可以使用以下命令下载：
>
> ```
> wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
> ```

8. 如果您正从 Amazon VPC 之外的客户机连接，则需要创建 SSH 隧道。您将在SSH选项卡中创建。

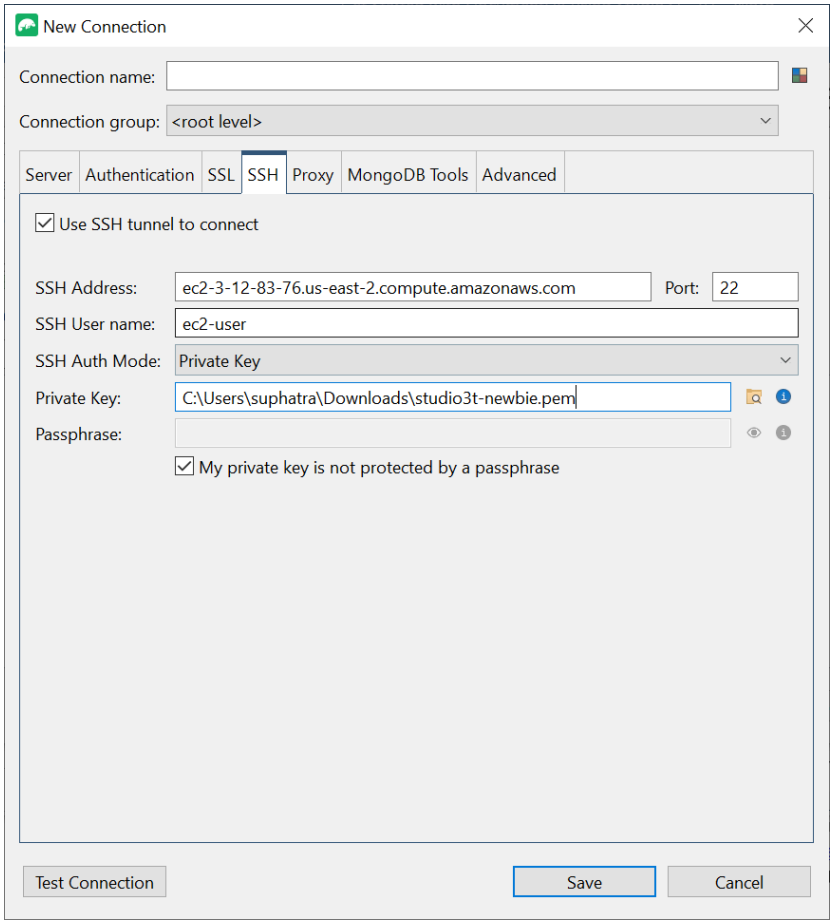
   a. 勾选使用 SSH 隧道复选框并在SSH 地址字段中输入 SSH 地址。这是您的实例公有 DNS (IPV4)。您可以从您的 Amazon EC2 管理控制台获取此 URL。

   b. 输入您的用户名。这是您的 Amazon EC2 实例用户名

   c. 对于 SSH 身份验证模式，请选择私钥。在私钥字段中，选择文件查找器图标以定位并选择 Amazon EC2 实例的私钥。这是您在Amazon EC2 控制台中创建实例时保存过的.pem 文件（密钥对）。

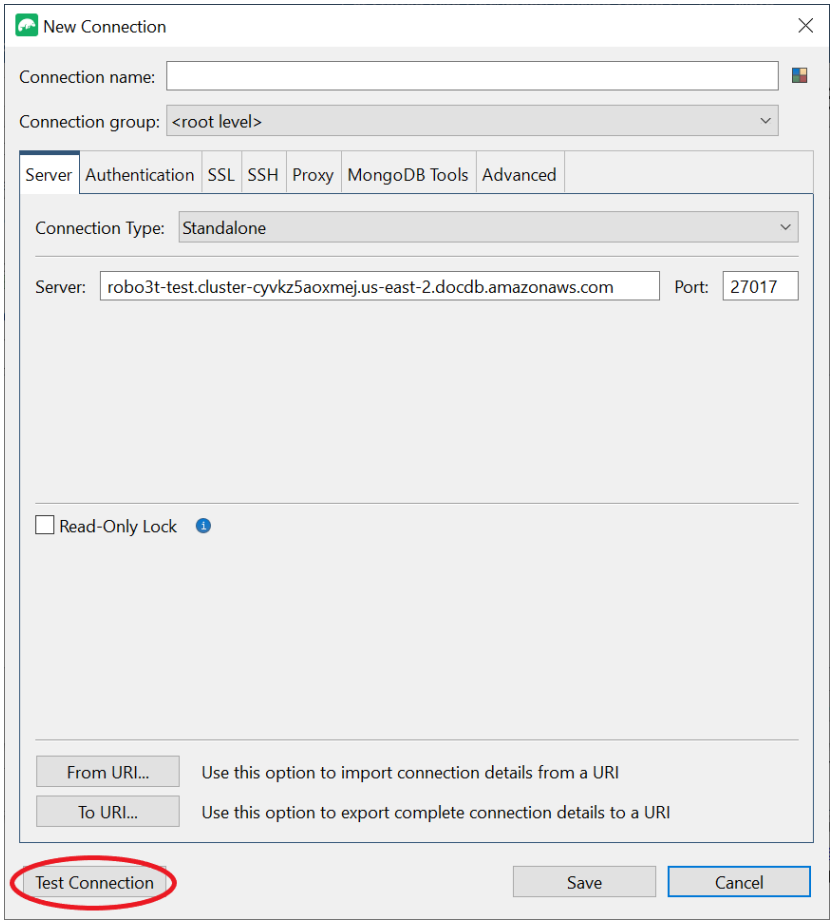
   d. 如果您在 Linux/macOS 客户端计算机上，则可能需要使用以下命令更改私钥的权限：

```
chmod 400 /fullPathToYourPemFile/<yourKey>.pem
```

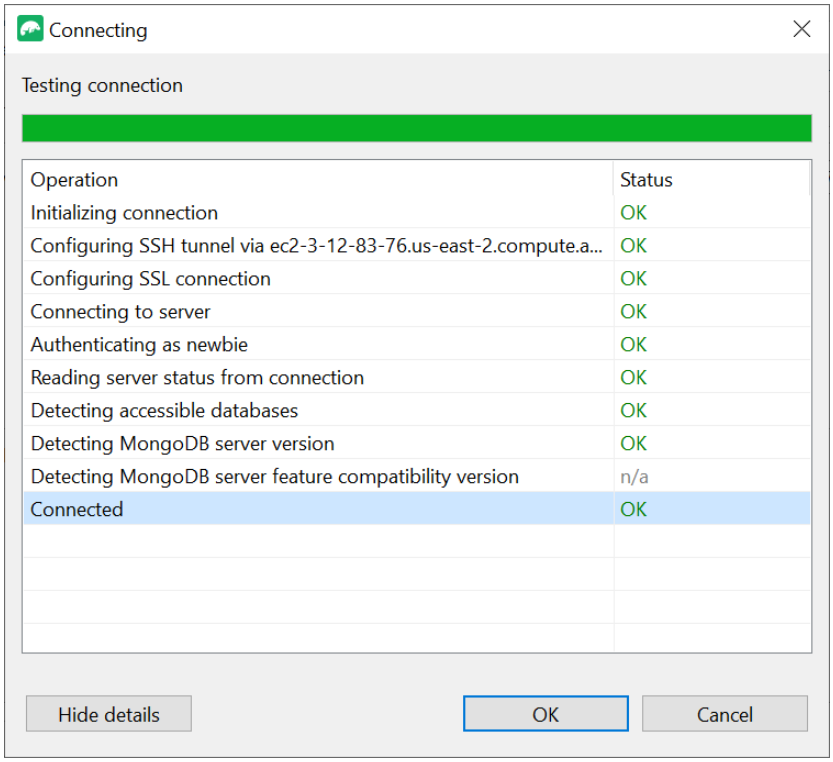


> **Note**
>
> 这个 Amazon EC2 实例应与您的 DocumentDB 集群处于相同的 Amazon VPC 和安全组中。您可以从自己的 Amazon EC2 管理控制台获取 SSH 地址、用户名和私钥。

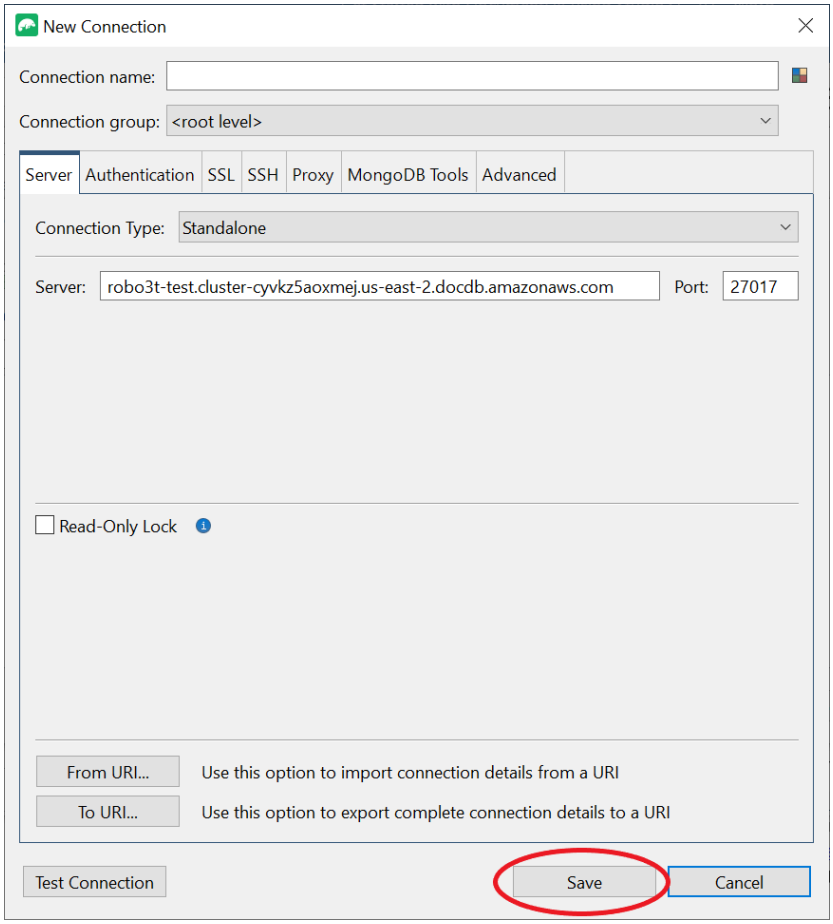
9. 现在，通过选择测试连接按钮测试您的配置。



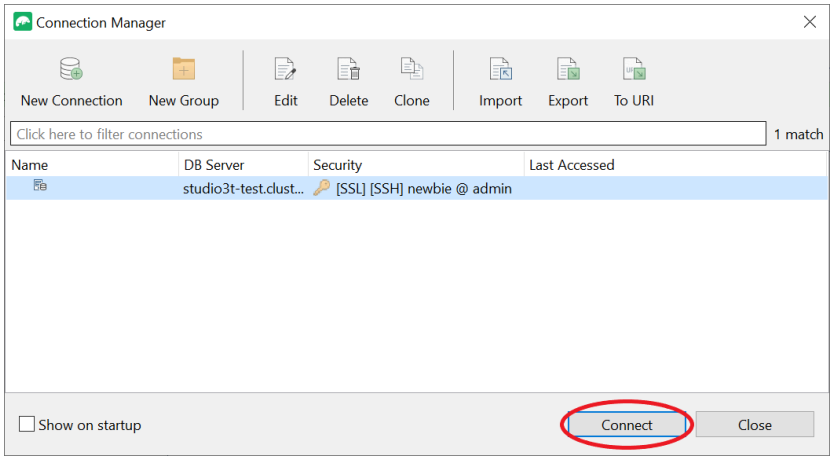
10. 诊断窗口应加载绿色条以指示测试成功。现在选择 OK 关闭诊断窗口。



11. 选择保存来保存您的连接以供将来使用。



12. 现在选择您的集群并选择连接。



恭喜您！您已成功通过 Studio 3T 连接到您的 Amazon DocumentDB 集群。

使用连接亚马逊 DocumentDB DataGrip

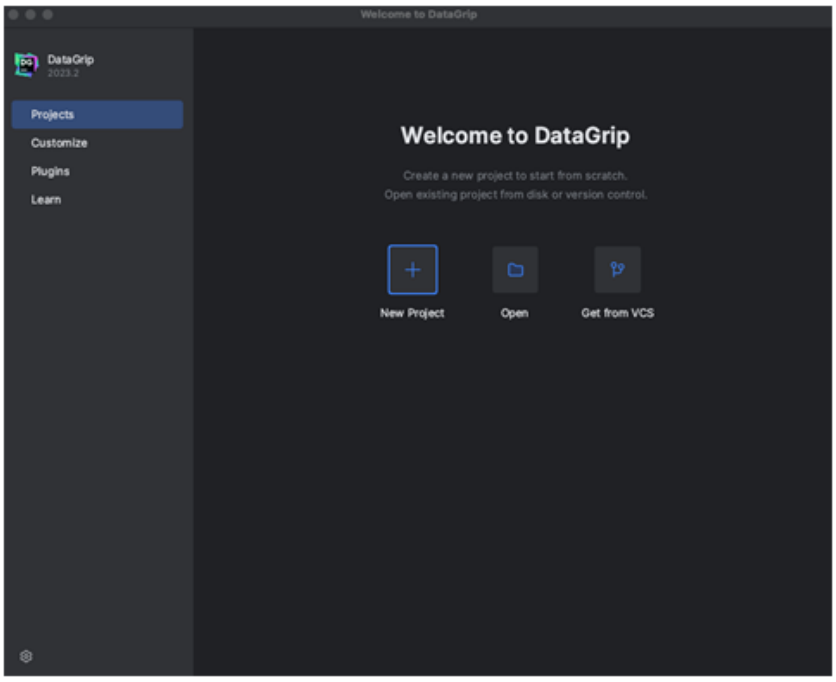
DataGrip是一个强大的集成开发环境 (IDE)，支持各种数据库系统，包括 Amazon DocumentDB。本节将引导您完成使用连接到 Amazon DocumentDB 集群的步骤 DataGrip，使您能够使用图形界面轻松管理和查询数据。

先决条件

- DataGrip IDE 已安装在您的计算机上。你可以从中下载JetBrains。
- 在与您的 Amazon DocumentDB 集群相同的 VPC 中运行的 Amazon EC2 实例。您将使用此实例建立从您的本地机到 Amazon DocumentDb 集群的安全隧道。要了解如何 使用 Amazon EC2 进行连接，请遵循以下说明：
- Amazon EC2 实例、VPN 连接的替代方案，或者如果您已经在使用安全 VPN 访问 AWS 基础设施。如果您更喜欢此选项，请按照说明使用安全地访问亚马逊文档数据库。 AWS Client VPN

使用 Connect DataGrip

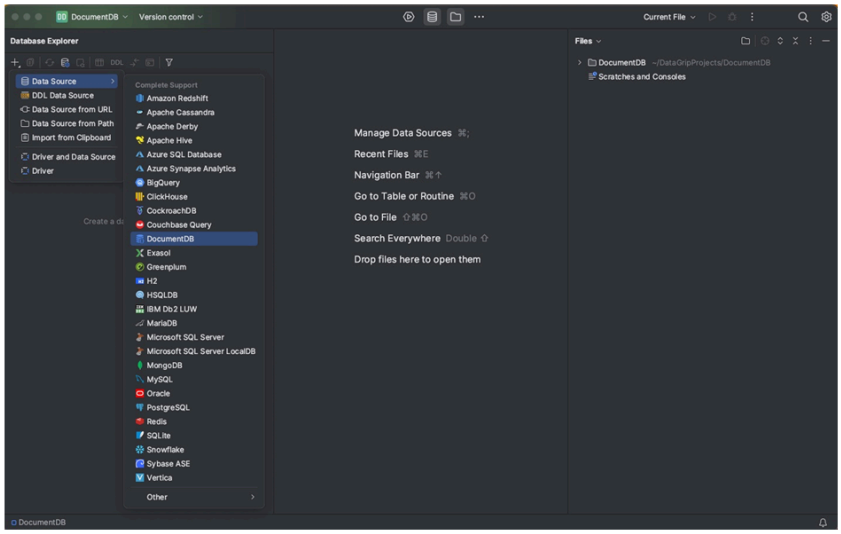
1. DataGrip 在电脑上启动并创建一个新项目。



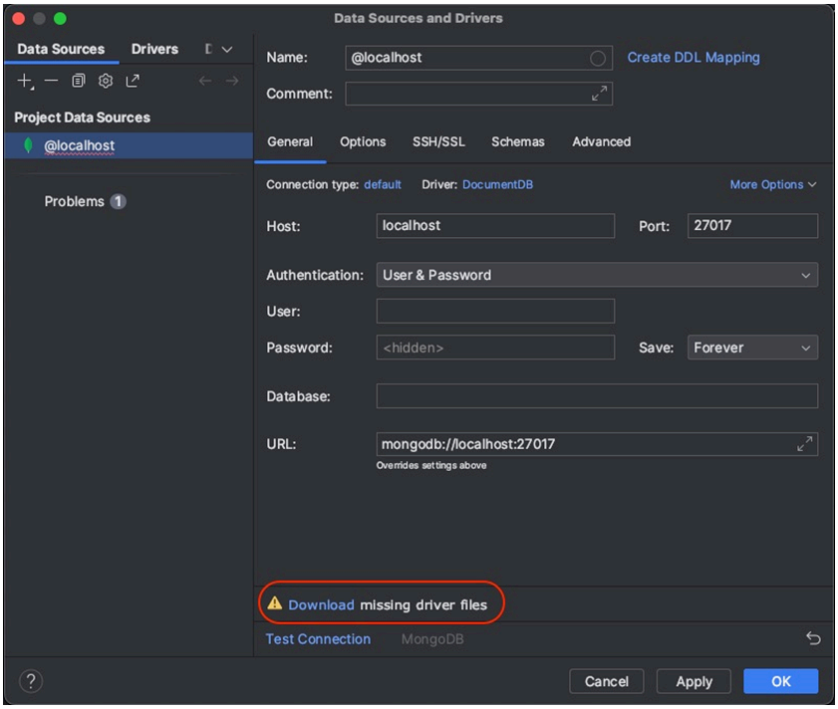
2. 通过以下方式之一添加新数据来源：

   a. 从主菜单中导航到 文件 – 新建 – 数据来源，然后选择 DocumentDB

   b. 在数据库资源管理器中，选择工具栏中的新图标 (+)。导航到数据来源，然后选择DocumentDB。



3. 在数据来源页面上的常规选项卡中，检查在连接设置区底部是否有下载缺少的驱动程序文件链接。选择此链接可下载与数据库交互所需的驱动程序。有关直接下载链接，请参阅 JetBrains JDBC 驱动程序。



4. 在常规选项卡中，指定连接详情：

   a. 在主机字段中，指定 Amazon DocumentDB 集群端点。

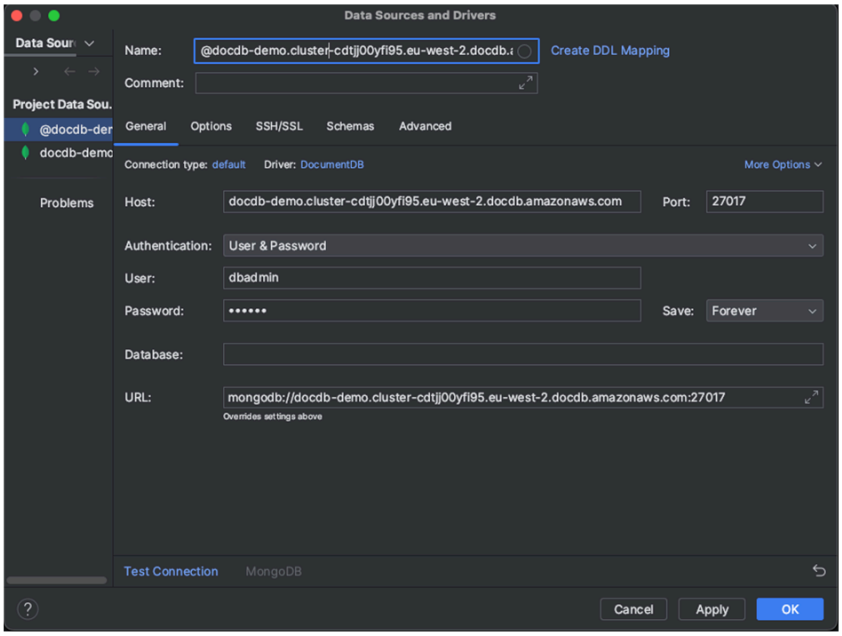
   b. 端口已设置成 27017。如果您的集群部署在一个不同端口上，请更改之。

   c. 对于身份验证，请选择用户 & 密码。

   d. 输入您的用户名和密码信息。

e. 数据库字段为可选项。您可以指定您想要连接的数据库。

f. 添加上述详细信息时，URL 字段会自动完成。



5. 在SSH/SSL选项卡中，启用 “使用 SSH 隧道”，然后选择图标打开 “SSH 配置” 对话框。输入以下信息：

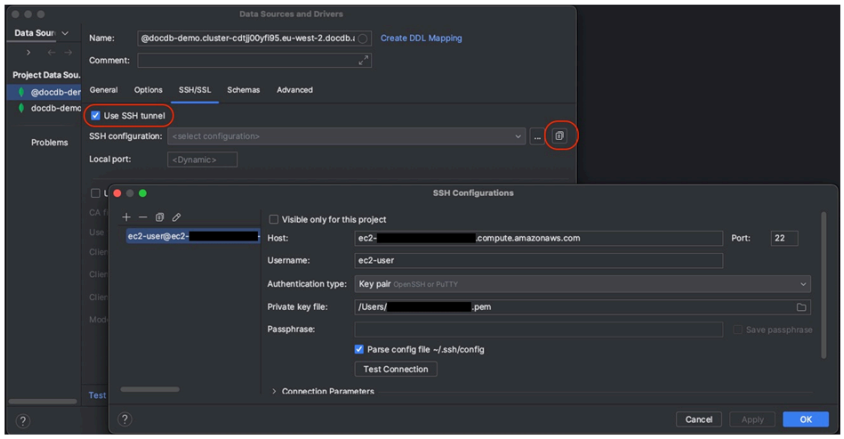
a. 在主机字段中，输入您的 Amazon EC2 实例的主机名。

b. 为您的 Amazon EC2 实例输入用户名和密码。

c. 对于 Authentication Type（身份验证类型），选择 密钥对。

d. 输入您的私钥文件。

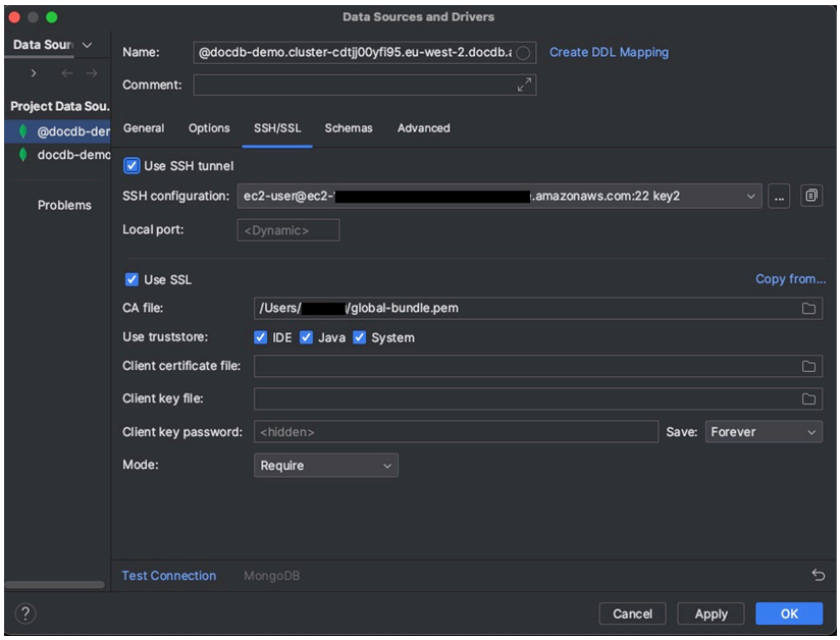
> **Note**
>
> 如果您正使用 VPN 选项，则无需配置 SSH 隧道。



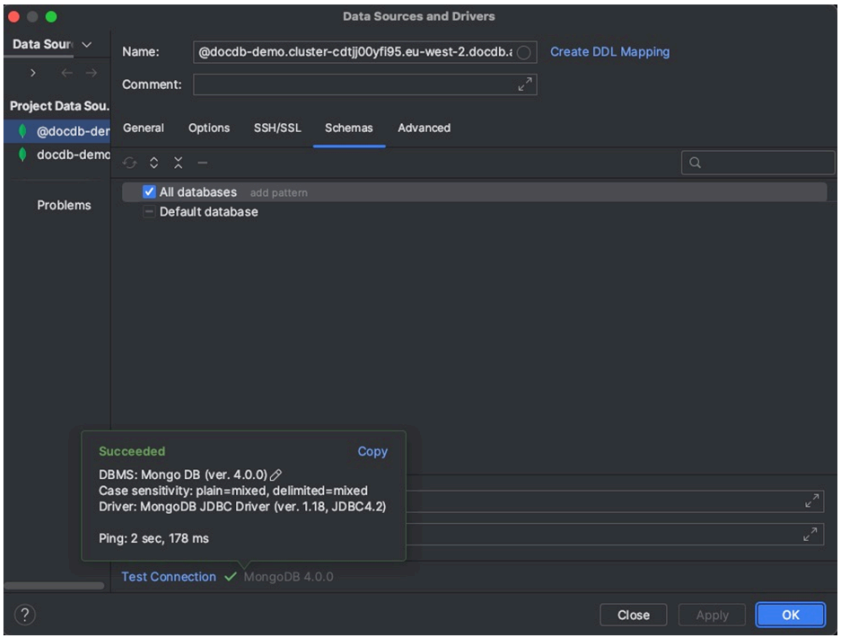
6. 在SSH/SSL选项卡中，启用 “使用 SSL”。在 CA 文件字段中，输入您计算机上到达 `global-bundle.pem` 文件的位置。对于模式，保留需要选项。

> **Note**
>
> 你可以从这个位置或用这个命令：wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem下载证书

> **Note**
>
> 如果您要连接到 Amazon DocumentDB 弹性集群，则无需指定 CA 文件。保留勾选使用 SSL 选项及所有其他选项处于其默认值。



7. 在架构选项卡中，选择所有数据库或在架构模式字段中输入筛选器“*:*”。选择 “测试连接” 以测试连接。



8. 成功测试连接后，选择确定以保存数据源配置。

DataGrip features

DataGrip 提供了多种功能来帮助您高效地使用 Amazon DocumentDB：

- SQL 编辑器-使用中的 SQL 编辑器编写和执行对您的 DocumentDB 集合的 SQL-like 查询。DataGrip
- 可视性查询生成器- 使用可视性查询生成器以图形方式创建查询，而无需编写 SQL 代码。
- 架构管理：轻松管理您的数据库架构，包括创建、更改和删除集合。
- 数据可视化-使用中提供的各种可视化工具查看和分析您的数据 DataGrip。
- 导出和导入数据-使用导出和导入功能在 Amazon Documen DataGrip tDB 和其他数据库之间传输数据。

有关使用 Amazon DocumentDB 和其他数据库系统的更多高级功能和技巧，请参阅官方DataGrip 文档。

使用 Amazon EC2 进行连接

本节介绍如何设置 Amazon DocumentDB 集群和 Amazon EC2 之间的连接，以及如何从 Amazon EC2 实例访问 Amazon DocumentDB 集群。

有两种选项可用于配置 EC2 连接：

- EC2 实例自动连接到 Amazon DocumentDB 数据库：使用 EC2 控制台中的自动连接功能，自动配置 EC2 实例与新的或现有 Amazon DocumentDB 数据库之间的连接。此连接允许在 EC2 实例与 Amazon DocumentDB 数据库之间传输流量。这一选项通常用于测试和创建新的安全组。
- 手动连接 EC2 实例到 Amazon DocumentDB 数据库：通过手动配置和分配安全组来配置 EC2 实例与 Amazon DocumentDB 数据库之间的连接，从而重现由自动连接功能创建的配置。这一选项通常用于更改更高级的设置以及使用现有的安全组。

先决条件

不管选择哪种选项，在创建第一个 Amazon DocumentDB 集群之前，您必须执行以下操作：

已创建 Amazon Web Services（AWS）账户

在开始使用 Amazon DocumentDB 之前，您必须拥有 Amazon Web Services（AWS）账户。该 AWS 账户是免费的。您只需为使用的服务和资源付费。

如果您没有 AWS 账户，请完成以下步骤来创建一个。

报名参加 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明操作。

   在注册时，将接到电话或收到短信，要求使用电话键盘输入一个验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

（可选）设置所需的 AWS Identity and Access Management (IAM) 权限。

要管理集群、实例和集群参数组等 Amazon DocumentDB 资源，需要 AWS 能够对您的请求进行身份验证的证书。有关更多信息，请参阅 适用于 Amazon DocumentDB 的 Identity and Access Management。

1. 在的搜索栏中 AWS 管理控制台，键入 IAM，然后在出现的下拉菜单中选择 IAM。
2. 一旦您进入 IAM 控制台，就从导航窗格中选择用户。
3. 选择您的用户名。
4. 选择添加权限。
5. 选择 Attach existing policies directly（直接附加现有策略）。
6. 在搜索栏中键入 `AmazonDocDBFullAccess`，并且一旦它出现在搜索结果中就选择之。
7. 选择 Next: Review（下一步：审核）。
8. 选择 Add permissions (添加权限)。

创建 Amazon Virtual Private Cloud（Amazon VPC）

根据您所在的位置， AWS 区域 您可能已经创建了默认 VPC，也可能没有。如果您没有默认 VPC，请完成 Amazon VPC 用户指南中 Amazon VPC 入门 的步骤 1。这将耗时不到五分钟。

自动连接 Amazon EC2

主题

- 将 EC2 实例自动连接到新的 Amazon DocumentDB 数据库
- 将 EC2 实例自动连接到现有的 Amazon DocumentDB 数据库

- 与 EC2 实例的自动连接概述
- 查看连接的计算资源

在设置 EC2 实例与新 Amazon DocumentDB 数据库之间的连接之前，请确保满足 与 EC2 实例的自动连接概述 中所述的要求。如果您在配置连接后更改安全组，则这些更改可能会影响 EC2 实例与 Amazon DocumentDB 数据库之间的连接。

> **Note**
>
> 您只能使用 AWS 管理控制台自动设置 EC2 实例与 Amazon DocumentDB 数据库之间的连接。您无法使用 AWS CLI 或亚马逊 DocumentDB API 自动建立连接。

将 EC2 实例自动连接到新的 Amazon DocumentDB 数据库

下面的程序假定您已经完成 先决条件 主题中的步骤。

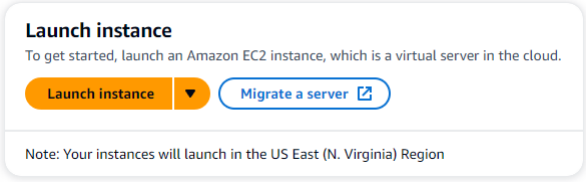
Steps

- 步骤 1：创建 Amazon EC2 实例
- 步骤 2：创建 Amazon DocumentDB 集群
- 步骤 3：连接 Amazon EC2 实例
- 步骤 4：安装 MongoDB Shell
- 步骤 5：管理 Amazon DocumentDB TLS
- 步骤 6：连接到 Amazon DocumentDB 集群
- 步骤 7：插入和查询数据
- 步骤 8：探索

步骤 1：创建 Amazon EC2 实例

在这个步骤，您将在同一区域和 Amazon VPC 中创建一个 Amazon EC2 实例，稍后您将使用这个实例配置您的 Amazon DocumentDB 集群。

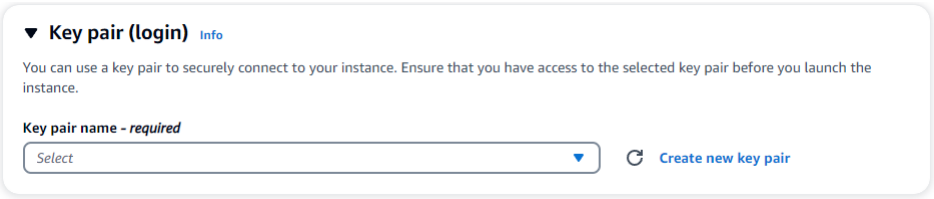
1. 从 Amazon EC2 控制台控制中，选择启动实例。



2. 在名称和标签部分的名称字段中，输入名称或标识符。
3. 在亚马逊机器映像 (AMI) 下拉列表中，找到并选中 Amazon Linux 2 AMI。

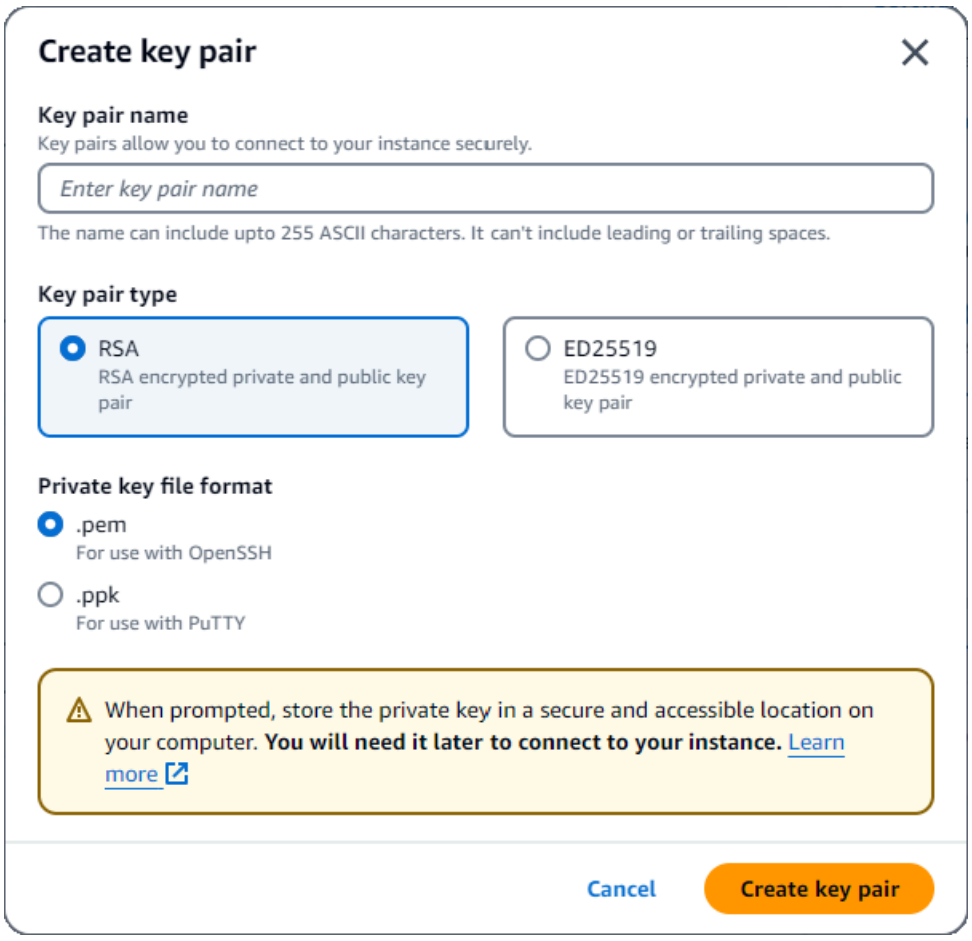


4. 在实例类型下拉列表中，找到并选中 t3.micro。
5. 在密钥对（登录）部分，输入现有密钥对的标识符，或选择新建密钥对。



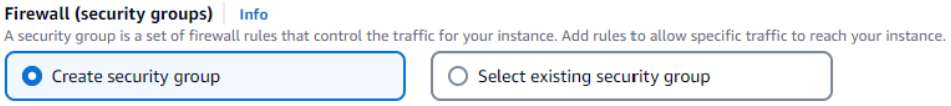
您必须提供 Amazon EC2 密钥对。

- 如果您的确有 Amazon EC2 密钥对：
  1. 选定一个密钥对，从列表中选择您的密钥对。
  2. 您必须有可用的私有密钥文件（.pem 或 .ppk 文件）才能登录 Amazon EC2 实例。
- 如果您没有 Amazon EC2 密钥对：
  1. 选择新建密钥对，随后出现创建密钥对对话框。
  2. 在密钥对名称字段中输入名称。
  3. 选择密钥对类型和私有密钥文件格式。
  4. 选择 Create key pair (创建密钥对)。



> **Note**
>
> 为安全起见，我们强烈建议使用密钥对进行 EC2 实例的 SSH 和互联网连接。

6. 可选：在网络设置部分的防火墙（安全组）下，选择创建安全组。



选择创建安全组（选中所有适用于您的 EC2 连接的流量允许规则）。

> **Note**
>
> 如果您想要使用现有安全组，请按照 手动连接 Amazon EC2 中的说明进行操作。

7. 在摘要部分中，查看 EC2 配置，如果配置正确，选择启动实例。

步骤 2：创建 Amazon DocumentDB 集群

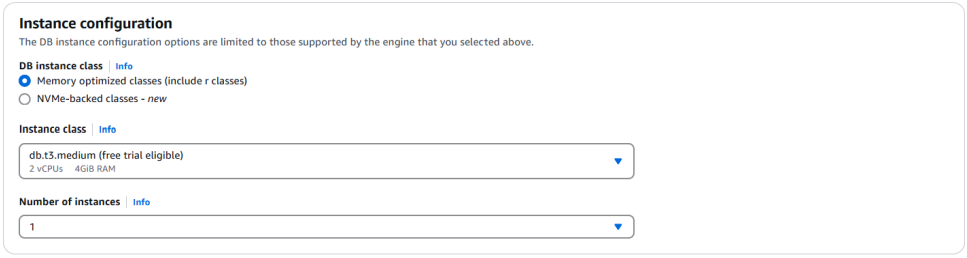
在预配置 Amazon EC2 实例的同时，创建您的 Amazon DocumentDB 集群。

1. 导航至 Amazon DocumentDB 控制台并且从导航窗格中选择集群。
2. 选择创建。
3. 将集群类型设置保留为默认的基于实例的集群。
4. 在集群配置中，对于集群标识符，请输入唯一名称。请注意，无论如何输入，控制台都会将所有集群的名称更改为小写。

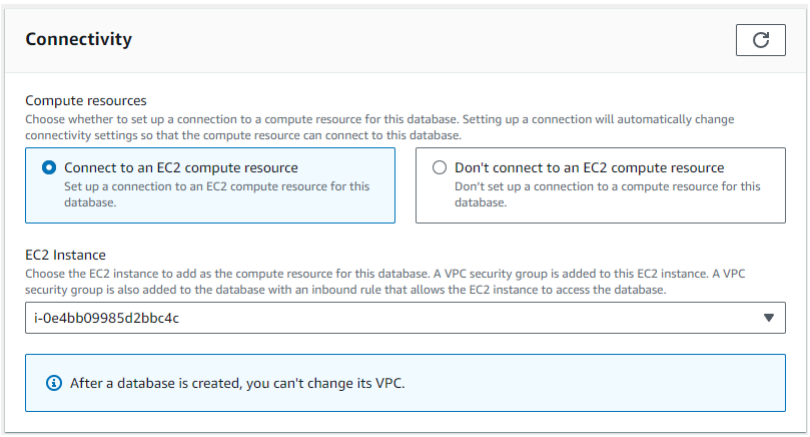
   将引擎版本保留为默认值 5.0.0。
5. 对于集群存储配置，请保留 Amazon DocumentDB 标准的默认设置。
6. 在实例配置中：

   - 对于数据库实例类，选择内存优化类（包括 r 类）（这是默认值）。

     另一个实例选项是NVMe-backed 类。要了解更多信息，请参阅NVMe-backed 实例。
   - 对于实例类，请选择符合您需求的实例类型。有关实例类的更详细说明，请参阅 实例类规格。
   - 对于实例数量，请选择最能反映您的需求的数量。请记住，数字越低，成本越低，集群可以管理的 read/write 容量也越低。



7. 对于连接，选择连接到 EC2 计算资源。这是您在步骤 1 中创建的 EC2 实例。



> **Note**
>
> 连接到 EC2 计算资源会自动为 EC2 计算资源到 Amazon DocumentDB 集群的连接创建一个安全组。创建完集群后，如果想查看新创建的安全组，请导航到集群列表并选择集群的标识符。在连接和安全选项卡中，转到安全组，然后在安全组名称 (ID) 下找到您的安全组。它将包含集群的名称，其外观类似于此：`docdb-ec2-docdb-2023-12-11-21-33-41:i-0e4bb09985d2bbc4c (sg-0238e0b0bf0f73877)`。

8. 在身份验证部分中，输入主要用户的用户名，然后选择自行管理。输入密码，然后确认密码。

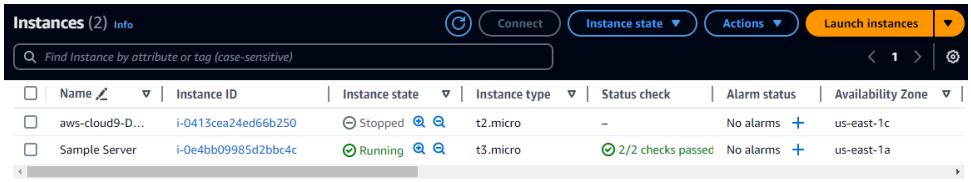
   如果您改为在中选择 “托管” AWS Secrets Manager，使用 Amazon DocumentDB 进行密码管理以及 AWS Secrets Manager请参阅，了解更多信息。

9. 选择创建集群。

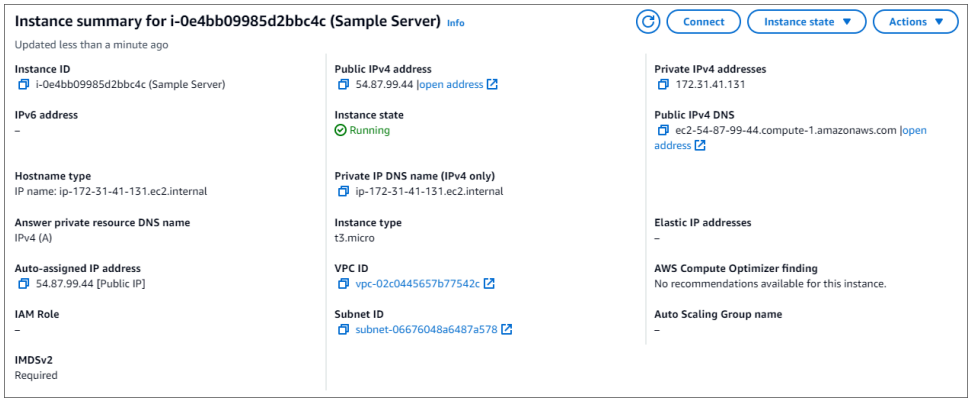
步骤 3：连接 Amazon EC2 实例

要安装 Mongo Shell，您必须首先连接到您的 Amazon EC2 实例。安装 Mongo Shell 使您能够连接到并查询您的 Amazon DocumentDB 集群。完成以下步骤：

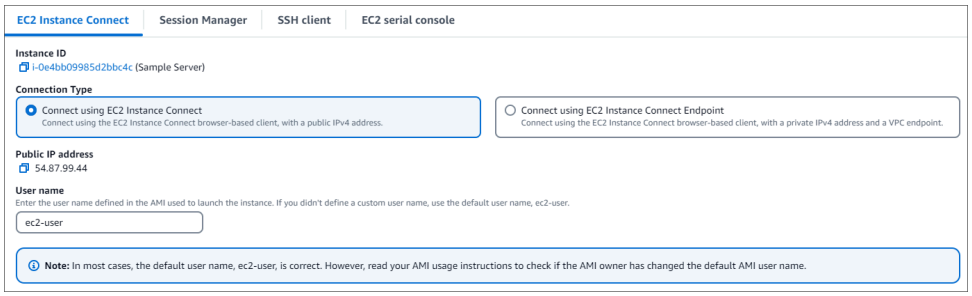
1. 在 Amazon EC2 控制台上，导航到您的实例并且查看您刚才创建的实例是否正在运行。如果是，请单击实例 ID 选择实例。



2. 选择连接。



3. 连接方法有四个选项卡选项：Amazon EC2 Instance Connect、会话管理器、SSH 客户端或 EC2 Serial Console。您必须选择一个选项并遵循其说明。完成后，选择连接。



> **Note**
>
> 如果您开始这次演练后自己的 IP 地址发生变化，或者您稍后正要返回您的环境，则必须更新您的 demoEC2 安全组入站规则，以启用来自新 API 地址的入站流量。

步骤 4：安装 MongoDB Shell

您现在可以安装 MongoDB Shell，它是一个命令行实用程序，用于连接和查询 Amazon DocumentDB 集群。目前有两个版本的 MongoDB Shell：最新版本 mongosh 和先前版本 mongo Shell。

> **Important**
>
> 版本早于 6.13.1 的 Node.js 驱动程序存在已知限制，Amazon DocumentDB 的 IAM 身份验证目前不支持这些限制。 Node.js 使用驱动程序（例如 mongosh）的 Node.js 驱动程序和工具必须升级为使用 Node.js 驱动程序版本 6.13.1 或更高版本。

遵循以下说明为您的操作系统安装 MongoDB Shell。

On Amazon Linux

在 Amazon Linux 上安装 MongoDB Shell

如果您未使用 IAM 身份验证，并且想要使用最新的 MongoDB Shell（mongosh）连接到您的 Amazon DocumentDB 集群，请按照以下步骤操作：

1. 创建存储库文件。在您创建的 EC2 实例的命令行中，执行以下命令：

```
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. 完成后，在命令提示符处使用以下两个命令选项之一安装 mongosh：

   选项 1：如果您在 Amazon EC2 配置期间选择了默认的 Amazon Linux 2023，请输入以下命令：

```
sudo yum install -y mongodb-mongosh-shared-openssl3
```

   选项 2：如果您在 Amazon EC2 配置期间选择了 Amazon Linux 2，请输入以下命令：

```
sudo yum install -y mongodb-mongosh
```

如果您使用的是 IAM 身份验证，则必须使用先前版本的 MongoDB Shell（5.0）连接到您的 Amazon DocumentDB 集群，请按照以下步骤操作：

1. 创建存储库文件。在您创建的 EC2 实例的命令行中，执行以下命令：

```
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. 完成后，在命令提示符处使用以下命令选项安装 mongodb 5.0 Shell：

```
sudo yum install -y mongodb-org-shell
```

On Ubuntu

在 Ubuntu 上安装 mongosh

1. 导入包管理系统将使用的公有密钥。

```
curl -fsSL https://pgp.mongodb.com/server-5.0.asc | sudo gpg --dearmor -o /usr/
share/keyrings/mongodb-server-5.0.gpg
```

2. 使用适合您的 Ubuntu 版本的命令创建用于 MongoDB 的列表文件 `mongodb-org-5.0.list`。

```
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. 使用以下命令导入并更新本地程序包数据库：

```
sudo apt-get update
```

4. 安装 mongosh。

```
sudo apt-get install -y mongodb-mongosh
```

有关在您的 Ubuntu 系统上安装早期版本的 MongoDB 的信息，请参阅在 Ubuntu 中安装 MongoDB Community Edition。

On other operating systems

要在其他操作系统上安装 mongo Shell，请参阅 MongoDB 文档中的安装 MongoDB Community Edition。

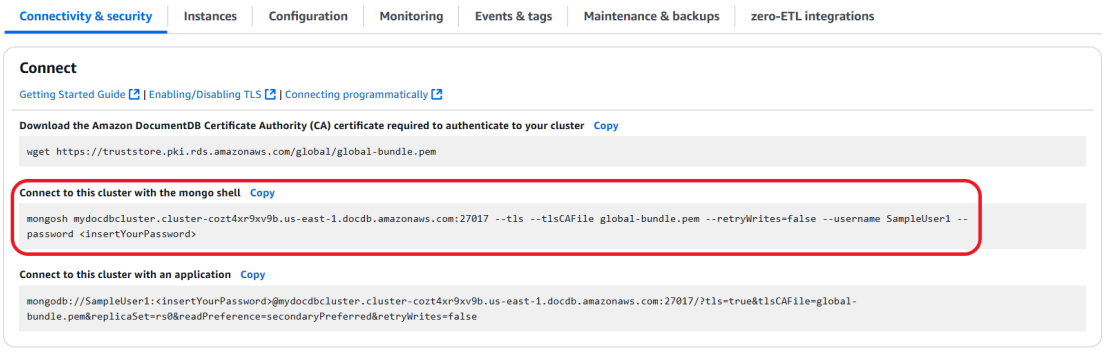
步骤 5：管理 Amazon DocumentDB TLS

用以下代码下载 Amazon DocumentDB 的 CA 证书：`wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem`

> **Note**
>
> 传输层安全性协议 (TLS)默认对所有新的 Amazon DocumentDB 集群启用。有关更多信息，请参阅管理 Amazon DocumentDB Cluster TLS 设置。

步骤 6：连接到 Amazon DocumentDB 集群

1. 在 Amazon DocumentDB 数据库控制台上的集群下，定位您的集群。通过单击该集群的集群标识符，选择您创建的集群。
2. 在连接和安全选项卡中，在连接方框中找到使用 mongo Shell 连接到此集群：



   复制所提供的连接字符串，并将其粘贴到您的终端中。

   对其进行以下更改：

   a. 确保字符串中的用户名正确。
   b. 省略 <insertYourPassword> 从而 mongo Shell 在您连接时提示您输入密码。
   c. 可选：如果您使用的是 IAM 身份验证，或者使用的是先前版本的 MongoDB Shell，请按以下方式修改您的连接字符串：

      ```
      mongo --ssl --host mydocdbcluster.cluster-cozt4xr9xv9b.us-east-1.docdb.amazonaws.com:27017 --sslCAFile global-bundle.pem --username SampleUser1 --password
      ```

      将 mydocdbcluster.cluster-cozt4xr9xv9b.us-east-1 替换为您的集群中的相同信息。

3. 在您的终端中按回车。现在，系统将提示您输入密码。输入您的密码。
4. 当输入密码并可以看到 rs0 [direct: primary] <env-name>> 提示时，您已成功连接到您的 Amazon DocumentDB 集群。

连接时遇到问题？ 参见 Amazon DocumentDB 故障排除。

步骤 7：插入和查询数据

现在，您已连接到自己的集群，您可以运行几个查询来熟悉如何使用文档数据库。

1. 要插入单个文档，请输入以下内容：

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

您会得到以下输出：

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

2. 您可以读取您用 `findOne()` 命令编写过的文档（因为它只返回单个文档）。输入以下：

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

您会得到以下输出：

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

3. 要执行若干更多查询，请考虑游戏个人资料用例。首先，将几个条目插入标题为 `profiles` 的集合。输入以下：

```
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 }
])
```

您会得到以下输出：

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

4. 使用 `find()` 命令返回个人资料集合中的所有文档。输入以下：

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

您将获得与您在步骤 3 中键入的数据相匹配的输出。

5. 利用筛选器对单个文档使用查询。输入以下：

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

您会得到以下输出：

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

6. 现在，让我们尝试查找个人资料并使用 `findAndModify` 命令修改它。我们将用以下代码向用户 Matt 给予额外的 10 分：

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

你得到以下输出（请注意，他的分数尚未增加）：

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

7. 你可以借助以下查询验证他的分数是否已变化：

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

您会得到以下输出：

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

步骤 8：探索

恭喜您！您已成功完成 Amazon DocumentDB 快速入门指南。

接下来做什么？ 了解如何充分利用这款强大的数据库及其热门功能：

- 管理 Amazon DocumentDB
- 扩展
- 备份和还原

> **Note**
>
> 为了节省成本，您可以停用您的 Amazon DocumentDB 集群以降低成本，也可以删除该集群。默认情况下，在闲置 30 分钟后，您的 AWS Cloud9 环境将停止底层 Amazon EC2 实例。

将 EC2 实例自动连接到现有的 Amazon DocumentDB 数据库

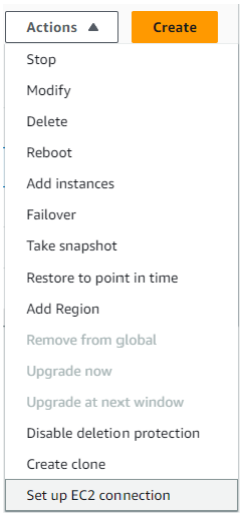
下面的程序假定存在一个现有 Amazon DocumentDB 集群和一个现有 Amazon EC2 实例。

访问 Amazon DocumentDB 集群并设置 Amazon EC2 连接

1. 访问 Amazon DocumentDB 集群。
   a. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
   b. 在导航窗格中，选择集群。

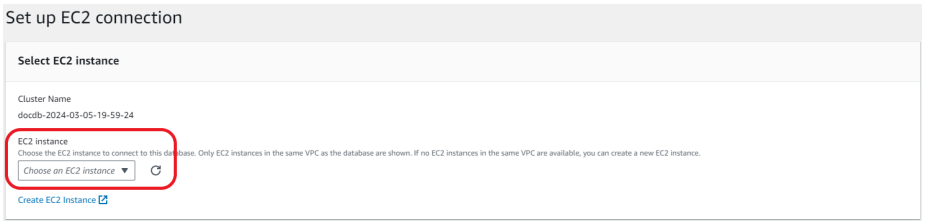
      > **Tip**
      >
      > 如果您在屏幕左侧没有看到导航窗格，请在页面左上角选择菜单图标 (≡)。

   c. 通过选择集群名称左侧的按钮，指定需要的集群。
2. 设置 Amazon EC2 连接。
   a. 选择操作，然后选择设置 EC2 连接。



将出现设置 EC2 连接页面。

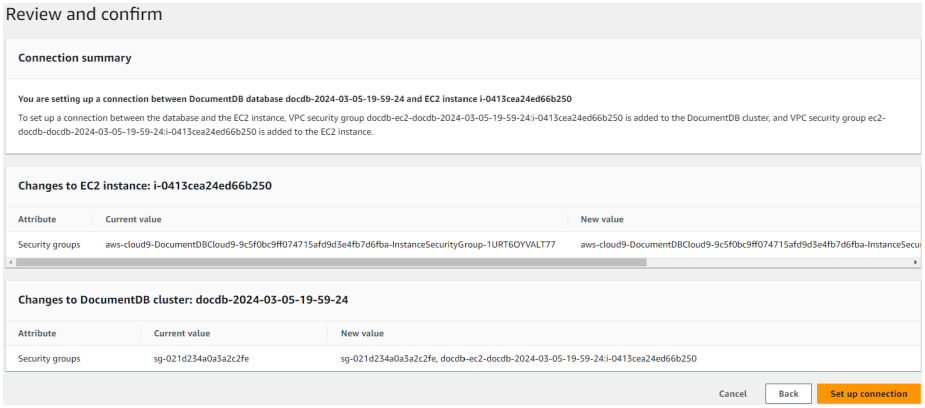
b. 在 EC2 实例字段中，选择要连接到集群的 EC2 实例。



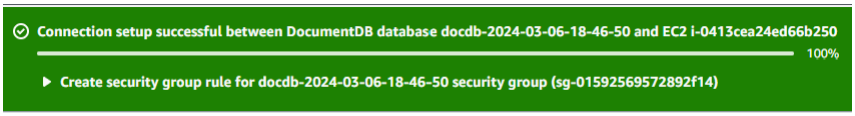
c. 选择继续。

将出现检查并确认页面。

d. 确保更改正确。然后选择设置连接。



成功后会显示以下验证：



与 EC2 实例的自动连接概述

当您设置 EC2 实例和 Amazon DocumentDB 数据库之间的连接时，Amazon DocumentDB 会自动为您的 EC2 实例和 Amazon DocumentDB 数据库配置 VPC 安全组。

以下是将 EC2 实例与 Amazon DocumentDB 数据库连接的要求：

- EC2 实例必须与 Amazon DocumentDB 数据库存在于同一 VPC 中。

  如果同一 VPC 中不存在任何 EC2 实例，则控制台将提供创建一个此类实例的链接。
- 设置连接的用户必须具有执行以下 Amazon EC2 操作的权限：
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeSecurityGroups`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

如果数据库实例和 EC2 实例位于不同的可用区，则您的账户可能会产生跨可用区成本。

当您设置与 EC2 实例的连接时，Amazon DocumentDB 会根据与 Amazon DocumentDB 数据库和 EC2 实例关联的安全组的当前配置采取操作，如下表所述。

当前 Amazon DocumentDB 安全组配置	当前 EC2 安全组配置	Amazon DocumentDB 操作
有一个或多个安全组与 Amazon DocumentDB 数据库关联，该数据库的名称与模式 `DocumentDB-ec2-n` 相匹配。尚未修改与此模式匹配的	有一个或多个安全组与 EC2 实例关联，此实例的名称与模式 `DocumentDB-ec2-n` （其中 n 是数字）相匹配。尚未修改与此模式匹配的安全组。该	Amazon DocumentDB 不执行任何操作。已在 EC2 实例和 Amazon DocumentDB 数据库之间自动配置了连接。由于 EC2 实例和 Amazon

当前 Amazon DocumentDB 安全组配置	当前 EC2 安全组配置	Amazon DocumentDB 操作
安全组。该安全组只具有一条以 EC2 实例的 VPC 安全组作为源的入站规则。	安全组只具有一条以 Amazon DocumentDB 数据库的 VPC 安全组作为源的出站规则。	DocumentDB 数据库之间已经存在连接，因此不会修改安全组。
以下任一条件适用： • 没有与 Amazon DocumentDB 数据库（其名称与模式 `DocumentDB-ec2-n` 匹配）关联的安全组。 • 有一个或多个安全组与 Amazon DocumentDB 关联，该数据库的名称与模式 `DocumentDB-ec2-n` 相匹配。但是，Amazon DocumentDB 不能使用这些安全组中的任何一个来连接 EC2 实例。如果安全组没有一条以 EC2 实例的 VPC 安全组作为源的入站规则，则 Amazon DocumentDB 无法使用该安全组。Amazon DocumentDB 也无法使用经过修改的安全组。修改示例包括添加规则或更改现有规则的端口。	以下任一条件适用： • 没有与 EC2 实例（其名称与模式 `ec2-DocumentDB-n` 匹配）关联的安全组。 • 有一个或多个安全组与 EC2 实例关联，此实例的名称与模式 `ec2-DocumentDB-n` 相匹配。但是，Amazon DocumentDB 不能将其中任何安全组用于连接 Amazon DocumentDB 数据库。如果安全组没有一条以 Amazon DocumentDB 数据库的 VPC 安全组作为源的出站规则，则 Amazon DocumentDB 无法使用该安全组。Amazon DocumentDB 也无法使用经过修改的安全组。	Amazon DocumentDB 操作：新建安全组

当前 Amazon DocumentDB 安全组配置	当前 EC2 安全组配置	Amazon DocumentDB 操作
有一个或多个安全组与 Amazon DocumentDB 数据库关联，该数据库的名称与模式 `DocumentDB-ec2-n` 相匹配。尚未修改与此模式匹配的安全组。该安全组只具有一条以 EC2 实例的 VPC 安全组作为源的入站规则。	有一个或多个安全组与 EC2 实例（名称与模式 `ec2-DocumentDB-n` 匹配）关联。但是，Amazon DocumentDB 不能将其中任何安全组用于连接 Amazon DocumentDB 数据库。如果安全组没有一条以 Amazon DocumentDB 数据库的 VPC 安全组作为源的出站规则，则 Amazon DocumentDB 无法使用该安全组。Amazon DocumentDB 也无法使用经过修改的安全组。	Amazon DocumentDB 操作：新建安全组
有一个或多个安全组与 Amazon DocumentDB 数据库关联，该数据库的名称与模式 `DocumentDB-ec2-n` 相匹配。尚未修改与此模式匹配的安全组。该安全组只具有一条以 EC2 实例的 VPC 安全组作为源的入站规则。	存在用于连接的有效 EC2 安全组，但它与 EC2 实例不关联。此安全组的名称与模式 `DocumentDB-ec2-n` 相匹配。尚未修改它。它只具有一条以 Amazon DocumentDB 数据库的 VPC 安全组作为源的出站规则。	Amazon DocumentDB 操作：关联 EC2 安全组

当前 Amazon DocumentDB 安全组配置	当前 EC2 安全组配置	Amazon DocumentDB 操作
以下任一条件适用： • 没有与 Amazon DocumentDB 数据库（其名称与模式 DocumentDB-ec2-n 匹配）关联的安全组。 • 有一个或多个安全组与 Amazon DocumentDB 数据库关联，该数据库的名称与模式 DocumentDB-ec2-n 相匹配。但是，Amazon DocumentDB 不能使用这些安全组中的任何一个来连接 EC2 实例。如果安全组没有一条以 EC2 实例的 VPC 安全组作为源的入站规则，则 Amazon DocumentDB 无法使用该安全组。Amazon DocumentDB 也无法使用经过修改的安全组。	有一个或多个安全组与 EC2 实例（名称与模式 DocumentDB-ec2-n 匹配）关联。尚未修改与此模式匹配的安全组。该安全组只具有一条以 Amazon DocumentDB 数据库的 VPC 安全组作为源的出站规则。	Amazon DocumentDB 操作：新建安全组

Amazon DocumentDB 操作：新建安全组

Amazon DocumentDB 执行以下操作：

- 创建与模式 DocumentDB-ec2-n 匹配的新安全组。该安全组具有一条以 EC2 实例的 VPC 安全组作为源的入站规则。此安全组与 Amazon DocumentDB 数据库关联，并允许 EC2 实例访问 Amazon DocumentDB 数据库。
- 创建与模式 ec2-DocumentDB-n 匹配的新安全组。该安全组具有一条以 Amazon DocumentDB 数据库的 VPC 安全组作为源的出站规则。该安全组与 EC2 实例相关联，并允许 EC2 实例向 Amazon DocumentDB 数据库发送流量。

Amazon DocumentDB 操作：关联 EC2 安全组

Amazon DocumentDB 将有效的现有 EC2 安全组与 EC2 实例关联。该安全组允许 EC2 实例向 Amazon DocumentDB 数据库发送流量。

查看连接的计算资源

您可以使用 AWS 管理控制台 来查看连接到 Amazon DocumentDB 数据库的计算资源。显示的资源包括自动设置的计算资源连接。您可以通过以下方式自动设置与计算资源的连接：

- 您可以在创建数据库时选择计算资源。有关更多信息，请参阅创建 Amazon DocumentDB 集群和创建 Multi-AZ 数据库集群。
- 您可以在现有数据库和计算资源之间设置连接。有关更多信息，请参阅 自动连接 Amazon EC2。

列出的计算资源不包括手动连接到数据库的计算资源。例如，您可以通过向与数据库关联的 VPC 安全组添加规则来允许计算资源手动访问数据库。

要列出计算资源，必须满足以下条件：

- 与计算资源关联的安全组的名称与模式 `ec2-DocumentDB-n` 相匹配（其中 n 是数字）。
- 与计算资源关联的安全组具有出站规则，其端口范围设置为 Amazon DocumentDB 数据库使用的端口。
- 与计算资源关联的安全组具有出站规则，源设置为与 Amazon DocumentDB 数据库关联的安全组。
- 与 Amazon DocumentDB 数据库关联的安全组的名称与模式 `DocumentDB-ec2-n`（其中 n 是数字）相匹配。
- 与 Amazon DocumentDB 数据库关联的安全组具有一条出站规则，其端口范围设置为 Amazon DocumentDB 数据库使用的端口。
- 与 Amazon DocumentDB 数据库关联的安全组有一条入站规则，其源设置为与计算资源关联的安全组。

查看已连接到 Amazon DocumentDB 数据库的计算资源

1. 登录并打开 Amazon DocumentDB 控制台，网址为。 AWS 管理控制台https://console.aws.amazon.com/docdb
2. 在导航窗格中，选择数据库，然后选择 Amazon DocumentDB 数据库的名称。
3. 在连接和安全选项卡上，在连接的计算资源中查看计算资源。

手动连接 Amazon EC2

主题

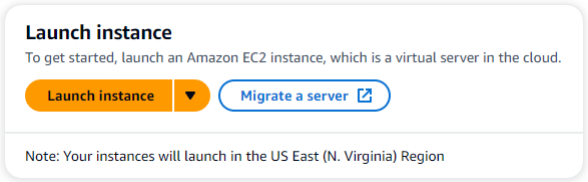
- 步骤 1：创建 Amazon EC2 实例
- 步骤 2：创建安全组
- 步骤 3：创建 Amazon DocumentDB 集群
- 第 4 步：配置 Amazon EC2 实例
- 步骤 5：安装 MongoDB Shell
- 步骤 6：管理 Amazon DocumentDB TLS
- 步骤 7：连接到 Amazon DocumentDB 集群
- 步骤 8：插入和查询数据
- 步骤 9：探索

下面的步骤假定您已经完成 先决条件 主题中的步骤。

步骤 1：创建 Amazon EC2 实例

在这个步骤，您将在同一区域和 Amazon VPC 中创建一个 Amazon EC2 实例，稍后您将使用这个实例配置您的 Amazon DocumentDB 集群。

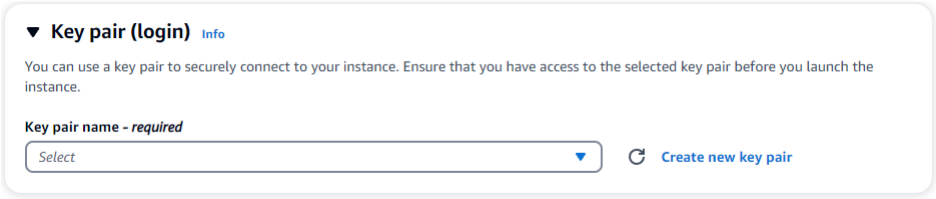
1. 从 Amazon EC2 控制台控制中，选择启动实例。



2. 在名称和标签部分的名称字段中，输入名称或标识符。
3. 在亚马逊机器映像 (AMI) 下拉列表中，找到并选中 Amazon Linux 2 AMI。

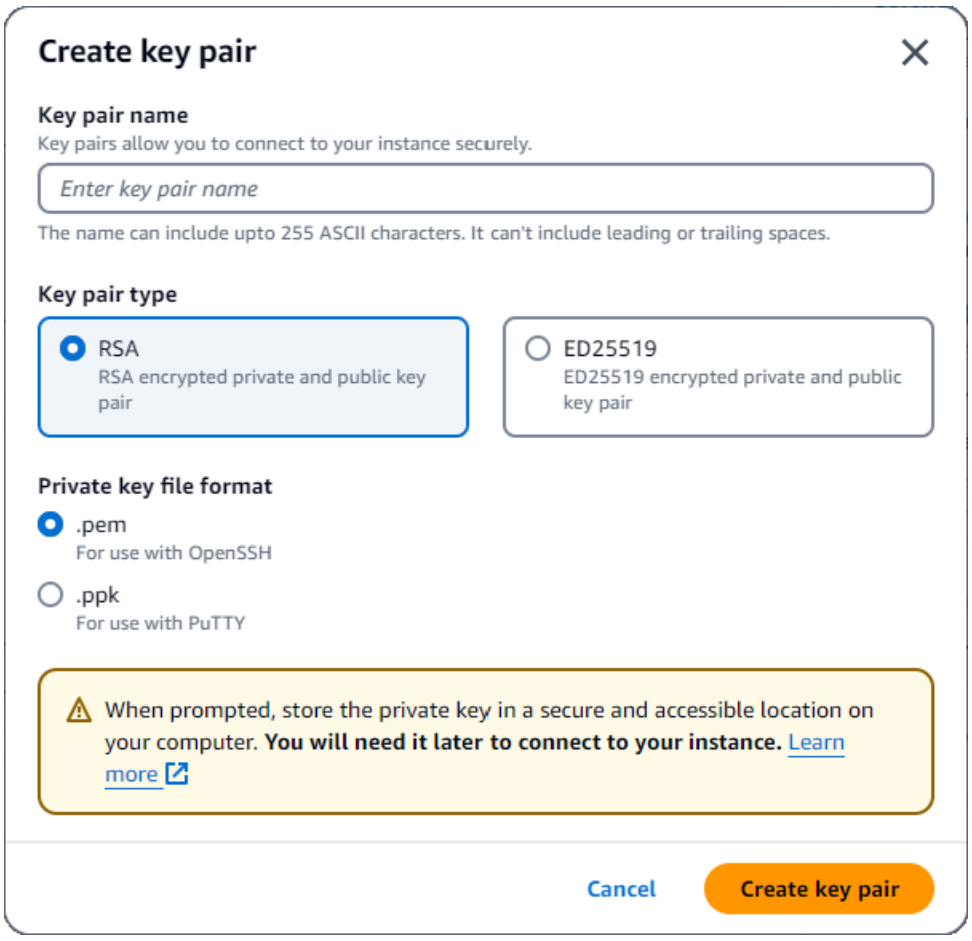


4. 在实例类型下拉列表中，找到并选中 t3.micro。
5. 在密钥对（登录）部分，输入现有密钥对的标识符，或选择新建密钥对。



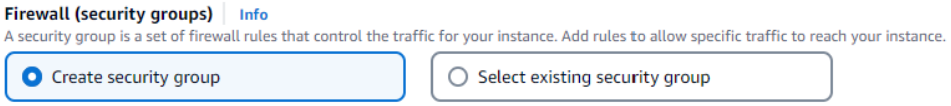
您必须提供 Amazon EC2 密钥对。

- 如果您的确有 Amazon EC2 密钥对：
  1. 选定一个密钥对，从列表中选择您的密钥对。
  2. 您必须有可用的私有密钥文件（.pem 或 .ppk 文件）才能登录 Amazon EC2 实例。
- 如果您没有 Amazon EC2 密钥对：
  1. 选择新建密钥对，随后出现创建密钥对对话框。
  2. 在密钥对名称字段中输入名称。
  3. 选择密钥对类型和私有密钥文件格式。
  4. 选择 Create key pair (创建密钥对)。



> **Note**
>
> 为安全起见，我们强烈建议使用密钥对进行 EC2 实例的 SSH 和互联网连接。

6. 在网络设置部分的防火墙（安全组）下，选择创建安全组或选择现有安全组。



如果选择现有安全组，请从通用安全组下拉列表中选择一个安全组。

如果选择创建新的安全组，请执行以下操作：

a. 查看所有适用于 EC2 连接的流量允许规则。

b. 在 IP 字段中，选择我的 IP 或选择自定义，以便从 CIDR 数据块、前缀列表或安全组列表中进行选择。除非 EC2 实例位于隔离网络上，否则我们不建议使用任何地方选项，因为它允许任何 IP 地址访问 EC2 实例。

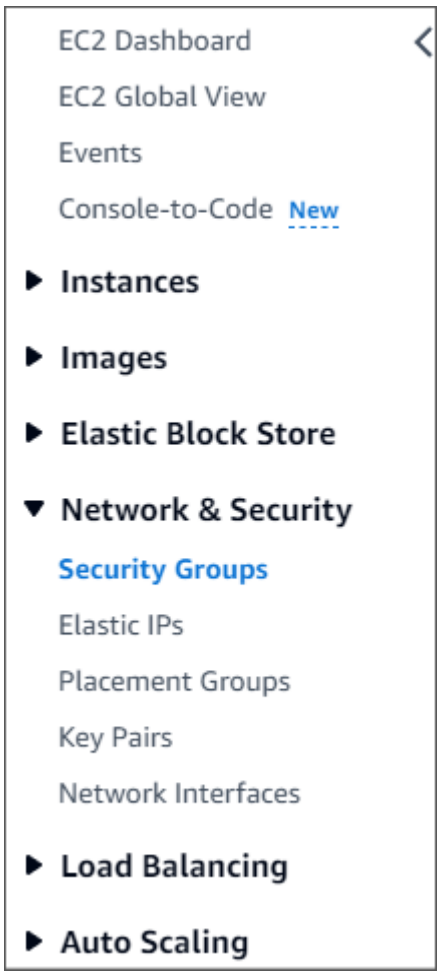


7. 在摘要部分中，查看 EC2 配置，如果配置正确，选择启动实例。

步骤 2：创建安全组

现在，您将在您的默认 Amazon VPC 中创建一个新安全组。该安全组demoDocDB使您能够在端口 27017（ Amazon DocumentDB 的默认端口）上从 Amazon EC2 实例连接到您的 Amazon DocumentDB 集群。

1. 在 Amazon EC2 管理控制台上，网络和安全下，选择安全组。



2. 选择创建安全组。



3. 在基本详细信息部分：

   a. 对于安全组名称，输入 demoDocDB。

   b. 对于说明，输入说明。

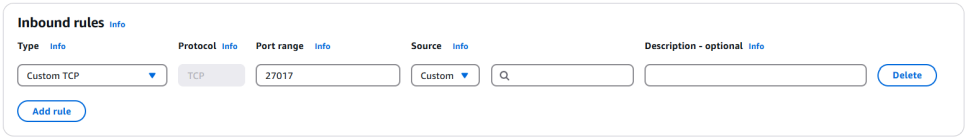
   c. 对于 VPC，请接受使用您的默认 VPC。

4. 在入站规则部分中，选择添加规则。

   a. 对于类型，请选择自定义 TCP 规则（默认值）。

   b. 对于端口范围，输入 27017。

   c. 对于 Source，选择 Custom。在与其紧邻的字段中，搜索您刚才在步骤 1 中创建的安全组。您可能需要刷新浏览器以便 Amazon EC2 控制台自动填充 来源名称。



5. 接受所有其他默认值并选择创建安全组。



步骤 3：创建 Amazon DocumentDB 集群

正在预配置 Amazon EC2 实例的同时，您将创建自己的 Amazon DocumentDB 集群。

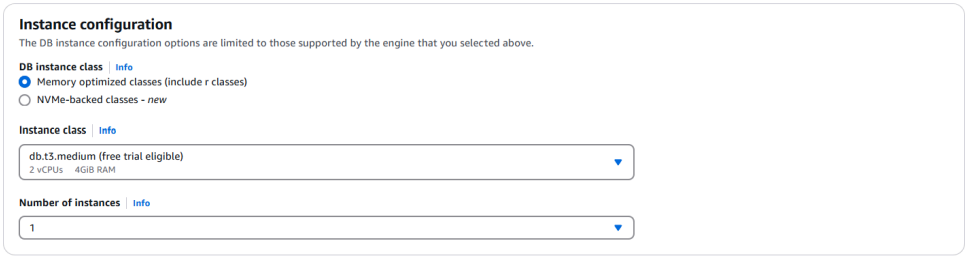
1. 导航至 Amazon DocumentDB 控制台并且从导航窗格中选择集群。
2. 选择创建。
3. 将集群类型设置保留为默认的基于实例的集群。
4. 在集群配置中，对于集群标识符，请输入唯一名称。请注意，无论如何输入，控制台都会将所有集群的名称更改为小写。

   将引擎版本保留为默认值 5.0.0。
5. 对于集群存储配置，请保留 Amazon DocumentDB 标准的默认设置。
6. 在实例配置中：

   - 对于数据库实例类，选择内存优化类（包括 r 类）（这是默认值）。

另一个实例选项是NVMe-backed 类。要了解更多信息，请参阅NVMe-backed 实例。

- 对于实例类，请选择符合您需求的实例类型。有关实例类的更详细说明，请参阅 实例类规格。
- 对于实例数量，请选择最能反映您的需求的数量。请记住，数字越低，成本越低，集群可以管理的 read/write 容量也越低。



7. 对于连接，保留默认设置：不连接到 EC2 计算资源。

> **Note**
>
> 连接到 EC2 计算资源时会自动为您的集群连接创建安全组。由于您已经在上一步中手动创建了这些安全组，因此应选择不连接到 EC2 计算资源，以免创建第二组安全组。

8. 在身份验证部分中，输入主要用户的用户名，然后选择自行管理。输入密码，然后确认密码。

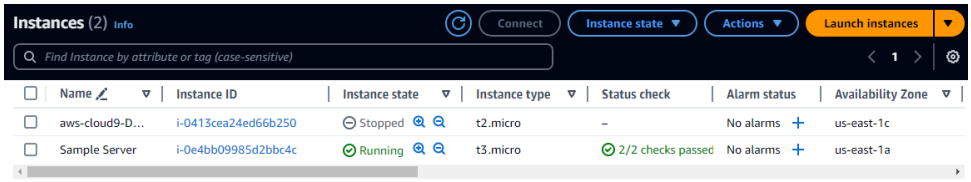
   如果您改为在中选择 “托管” AWS Secrets Manager，使用 Amazon DocumentDB 进行密码管理以及 AWS Secrets Manager请参阅，了解更多信息。

9. 选择创建集群。

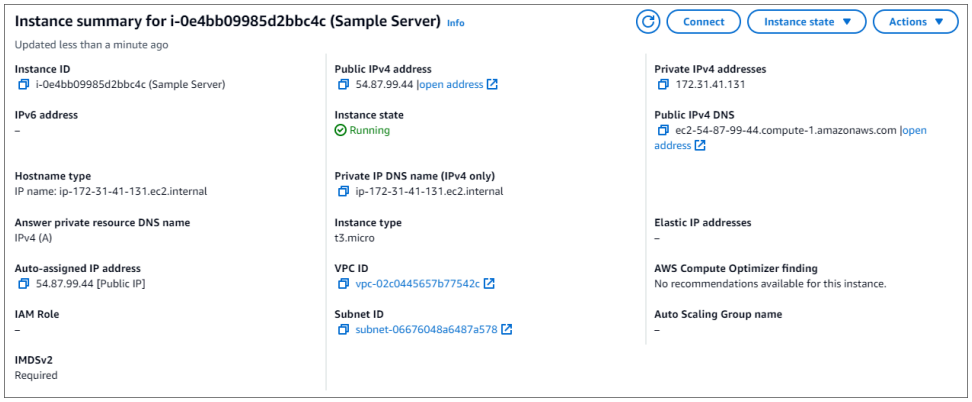
第 4 步：配置 Amazon EC2 实例

连接到您的 Amazon EC2 实例将允许您安装 MongoDB Shell。安装 Mongo Shell 使您能够连接到并查询您的 Amazon DocumentDB 集群。完成以下步骤：

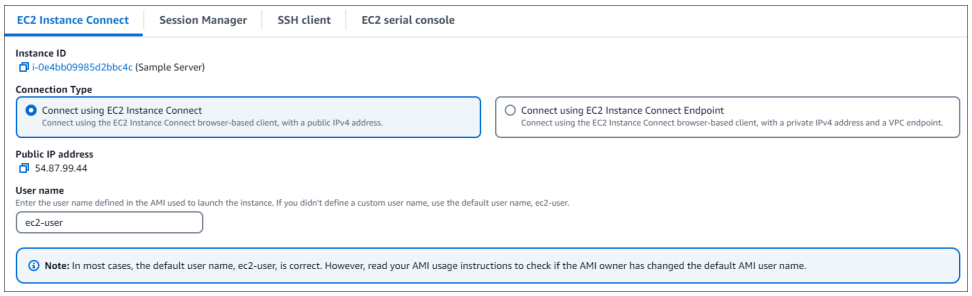
1. 在 Amazon EC2 控制台上，导航到您的实例并且查看您刚才创建的实例是否正在运行。如果是，请单击实例 ID 选择实例。



2. 选择连接。



3. 连接方法有四个选项卡选项：Amazon EC2 Instance Connect、会话管理器、SSH 客户端或 EC2 Serial Console。您必须选择一个选项并遵循其说明。完成后，选择连接。



> **Note**
>
> 如果您开始这次演练后自己的 IP 地址发生变化，或者您稍后正要返回您的环境，则必须更新您的 demoEC2 安全组入站规则，以启用来自新 API 地址的入站流量。

步骤 5：安装 MongoDB Shell

您现在可以安装 MongoDB Shell，它是一个命令行实用程序，用于连接和查询 Amazon DocumentDB 集群。目前有两个版本的 MongoDB Shell：最新版本 mongosh 和先前版本 mongo Shell。

> **Important**
>
> 版本早于 6.13.1 的 Node.js 驱动程序存在已知限制，Amazon DocumentDB 的 IAM 身份验证目前不支持这些限制。 Node.js 使用驱动程序（例如 mongosh）的 Node.js 驱动程序和工具必须升级为使用 Node.js 驱动程序版本 6.13.1 或更高版本。

遵循以下说明为您的操作系统安装 MongoDB Shell。

On Amazon Linux

在 Amazon Linux 上安装 MongoDB Shell

如果您未使用 IAM，并且想要使用最新的 MongoDB Shell（mongosh）连接到您的 Amazon DocumentDB 集群，请按照以下步骤操作：

1. 创建存储库文件。在您创建的 EC2 实例的命令行中，执行以下命令：

```
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. 完成后，在命令提示符处使用以下两个命令选项之一安装 mongosh：

   选项 1：如果您在 Amazon EC2 配置期间选择了默认的 Amazon Linux 2023，请输入以下命令：

```
sudo yum install -y mongodb-mongosh-shared-openssl3
```

   选项 2：如果您在 Amazon EC2 配置期间选择了 Amazon Linux 2，请输入以下命令：

```
sudo yum install -y mongodb-mongosh
```

如果您使用的是 IAM，则必须使用先前版本的 MongoDB Shell（5.0）连接到您的 Amazon DocumentDB 集群，请按照以下步骤操作：

1. 创建存储库文件。在您创建的 EC2 实例的命令行中，执行以下命令：

```
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. 完成后，在命令提示符处使用以下命令选项安装 mongodb 5.0 Shell：

```
sudo yum install -y mongodb-org-shell
```

On Ubuntu

在 Ubuntu 上安装 mongosh

1. 导入包管理系统将使用的公有密钥。

```
curl -fsSL https://pgp.mongodb.com/server-5.0.asc | sudo gpg --dearmor -o /usr/
share/keyrings/mongodb-server-5.0.gpg
```

2. 使用适合您的 Ubuntu 版本的命令创建用于 MongoDB 的列表文件 `mongodb-org-5.0.list`。

```
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. 使用以下命令导入并更新本地程序包数据库：

```
sudo apt-get update
```

4. 安装 mongosh。

```
sudo apt-get install -y mongodb-mongosh
```

有关在您的 Ubuntu 系统上安装早期版本的 MongoDB 的信息，请参阅在 Ubuntu 中安装 MongoDB Community Edition。

On other operating systems

要在其他操作系统上安装 mongo Shell，请参阅 MongoDB 文档中的安装 MongoDB Community Edition。

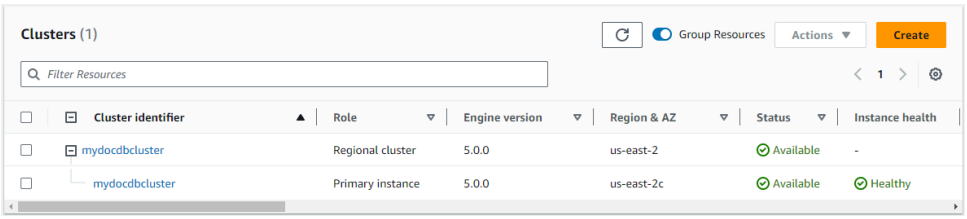
步骤 6：管理 Amazon DocumentDB TLS

用以下代码下载 Amazon DocumentDB 的 CA 证书：`wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem`

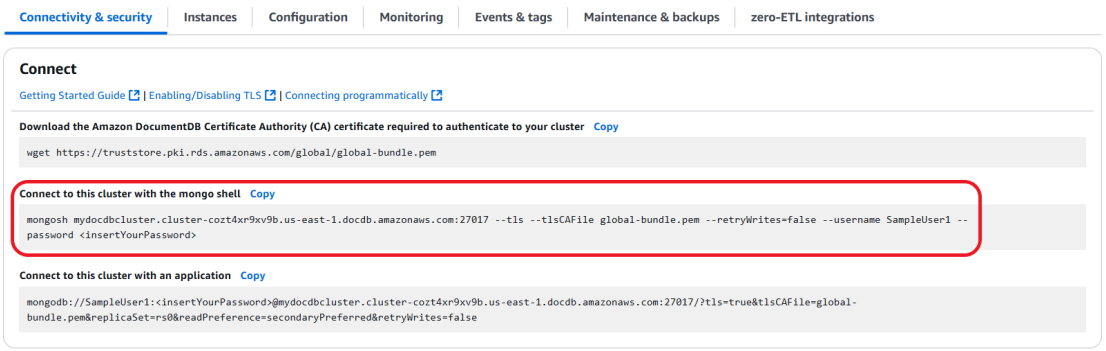
> **Note**
>
> 传输层安全性协议 (TLS)默认对所有新的 Amazon DocumentDB 集群启用。有关更多信息，请参阅管理 Amazon DocumentDB Cluster TLS 设置。

步骤 7：连接到 Amazon DocumentDB 集群

1. 在 Amazon DocumentDB 数据库控制台上的集群下，定位您的集群。通过单击该集群的集群标识符，选择您创建的集群。



2. 在连接和安全选项卡中，在连接方框中找到使用 mongo Shell 连接到此集群：



复制所提供的连接字符串，并将其粘贴到您的终端中。

对其进行以下更改：

a. 确保字符串中的用户名正确。

b. 省略 <insertYourPassword> 从而 mongo Shell 在您连接时提示您输入密码。

c. 可选：如果您使用的是 IAM 身份验证，或者使用的是先前版本的 MongoDB Shell，请按以下方式修改您的连接字符串：

```
mongo --ssl --host mydocdbcluster.cluster-cozt4xr9xv9b.us-
east-1.docdb.amazonaws.com:27017 --sslCAFile global-bundle.pem --
username SampleUser1 --password
```

将 mydocdbcluster.cluster-cozt4xr9xv9b.us-east-1 替换为您的集群中的相同信息。

3. 在您的终端中按回车。现在，系统将提示您输入密码。输入您的密码。
4. 当输入密码并可以看到 rs0 [direct: primary] <env-name>> 提示时，您已成功连接到您的 Amazon DocumentDB 集群。

连接时遇到问题？ 参见 Amazon DocumentDB 故障排除。

步骤 8：插入和查询数据

现在，您已连接到自己的集群，您可以运行几个查询来熟悉如何使用文档数据库。

1. 要插入单个文档，请输入以下内容：

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

您会得到以下输出：

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

2. 您可以读取您用 findOne() 命令编写过的文档（因为它只返回单个文档）。输入以下：

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

您会得到以下输出：

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

3. 要执行若干更多查询，请考虑游戏个人资料用例。首先，将几个条目插入标题为 profiles 的集合。输入以下：

```
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 }
```

```
])
```

您会得到以下输出：

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

4. 使用 find() 命令返回个人资料集合中的所有文档。输入以下：

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

您将获得与您在步骤 3 中键入的数据相匹配的输出。

5. 利用筛选器对单个文档使用查询。输入以下：

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

您会得到以下输出：

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

6. 现在，让我们尝试查找个人资料并使用 findAndModify 命令修改它。我们将用以下代码向用户 Matt 给予额外的 10 分：

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

你得到以下输出（请注意，他的分数尚未增加）：

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

7. 你可以借助以下查询验证他的分数是否已变化：

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

您会得到以下输出：

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

步骤 9：探索

恭喜您！您已成功完成 Amazon DocumentDB 快速入门指南。

接下来做什么？ 了解如何充分利用这款强大的数据库及其热门功能：

- 管理 Amazon DocumentDB
- 扩展
- 备份和还原

> **Note**
>
> 为了节省成本，您可以停用您的 Amazon DocumentDB 集群以降低成本，也可以删除该集群。默认情况下，在闲置 30 分钟后，您的 AWS Cloud9 环境将停止底层 Amazon EC2 实例。

使用 Amazon DocumentDB JDBC 驱动程序进行连接

适用于 Amazon DocumentDB 的 JDBC 驱动程序为开发人员提供了 SQL-relational 一个接口，并支持通过 Tableau 和等商业智能工具进行连接。 DbVisualizer

有关更多详细信息，请参阅上的 Amazon DocumentDB JDBC 驱动程序文档。 GitHub

主题

- 开始使用
- 从 Tableau Desktop 连接到 Amazon DocumentDB
- 从 Connect 到 Amazon DocumentDB DbVisualizer
- JDBC 自动生成架构
- SQL 支持和限制
- 问题排查

开始使用

步骤 1：创建 Amazon DocumentDB 集群

如果您尚未创建 Amazon DocumentDB 集群，请按照 Amazon DocumentDB 开发人员指南中入门部分的说明创建一个集群。

> **Note**
>
> Amazon DocumentDB 是一项仅针对虚拟私有云（VPC）的服务。如果您从集群 VPC 外部的本地计算机进行连接，则需要创建与 Amazon EC2 实例的 SSH 连接。在这种情况下，请按照使用 EC2 连接中的说明启动集群。有关 SSH 隧道以及何时可能需要隧道的更多信息，请参阅使用 SSH 隧道连接到 Amazon DocumentDB。

步骤 2：JRE 或 JDK 安装

根据您的 BI 应用程序，您可能需要确保计算机上安装了 64 位 JRE 或 JDK 安装版本 8 或更高版本。您可以在此处下载 Java SE 运行时环境 8。

步骤 3：下载 DocumentDB JDBC 驱动程序

在此下载 DocumentDB JDBC 驱动程序。该驱动程序被打包为单个 JAR 文件（例如 documentdb-jdbc-1.0.0-all.jar）。

步骤 4：使用 SSH 隧道连接到 Amazon DocumentDB

Amazon DocumentDB（与 MongoDB 兼容）集群部署在 Amazon Virtual Private Cloud (Amazon VPC) 中。部署在同一 Amazon VPC 中的 Amazon EC2 实例或其他 AWS 服务可以直接访问它们。此外，同一 AWS 区域或其他区域的不同 VPC 中的 ec2a 实例或其他 AWS 服务可以通过 VPC 对等互连访问 Amazon DocumentDB。

您可以使用 SSH 隧道（也称为端口转发）从集群的 VPC 外部访问您的 Amazon DocumentDB 资源。对于大多数不在与 DocumentDB 集群位于同一 VPC 中的虚拟机上运行应用程序的用户来说，情况就是如此。

要创建 SSH 隧道，您需要一个与您的 Amazon DocumentDB 集群在同一 Amazon VPC 中运行的 Amazon EC2 实例。您可以使用同一 VPC 中的现有 EC2 实例作为集群，或创建一个集群。您可以通过在本地计算机上运行以下命令设置到 Amazon DocumentDB 集群 `sample-cluster.node.us-east-1.docdb.amazonaws.com` 的 SSH 隧道。

```
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 标志用于转发本地端口。这是连接 VPC 外部客户端上运行的任何商业智能工具的前提条件。运行上述步骤后，您可以继续执行所选商业智能工具的后续步骤。

有关 SSH 隧道的更多信息，请参阅有关使用 SSH 隧道连接到 Amazon DocumentDB 的文档。

从 Tableau Desktop 连接到 Amazon DocumentDB

主题

- 添加 Amazon DocumentDB JDBC 驱动程序
- 使用 Tableau 连接到 Amazon DocumentDB：SSH 隧道

添加 Amazon DocumentDB JDBC 驱动程序

要从 Tableau Desktop 连接到 Amazon DocumentDB，您必须下载并安装 Amazon DocumentDB JDBC 驱动程序和 DocumentDB Tableau 连接器。

1. 从 Amazon DocumentDB JDBC 驱动程序存储库下载 Amazon DocumentDB JDBC 驱动程序 JAR 文件，然后根据您的操作系统将其复制到以下目录之一：
   - Windows：`C:\Program Files\Tableau\Drivers`
   - MacOS：`~/Library/Tableau/Drivers`
2. 从 Tableau E xchange 网站下载 DocumentDB Tableau 连接器（TACO 文件），然后将其复制到你的 “我的 Tableau” 目录中。 Repository/Connectors
   - Windows：`C:\Users\[user]\Documents\My Tableau Repository\Connectors`
   - MacOS：`/Users/[user]/Documents/My Tableau Repository/Connectors`

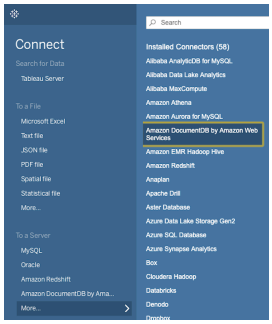
有关更多信息，请参阅 Tableau 文档。

> **Note**
>
> 如果您使用的是较新的 CA 证书，请务必将 JDBC 驱动程序升级到 v1.4.5（在此 AWS GitHub 存储库中可用。）。

使用 Tableau 连接到 Amazon DocumentDB：SSH 隧道

若要从 DocumentDB 集群的 VPC 外部的客户端计算机连接到 Tableau，您必须先设置一个 SSH 隧道，然后再执行以下步骤：

1. 启动 Tableau Desktop 应用程序。
2. 导航至连接 > 至服务器 > 更多。
3. 在已安装的连接器下选择 Amazon DocumentDB by Amazon Web Services。

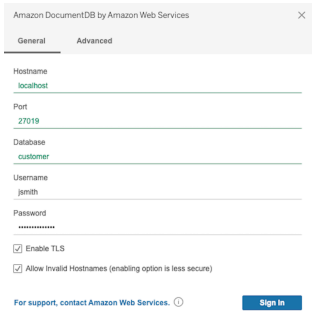


使用 Tableau 连接到 Amazon DocumentDB：外部 SSH 隧道

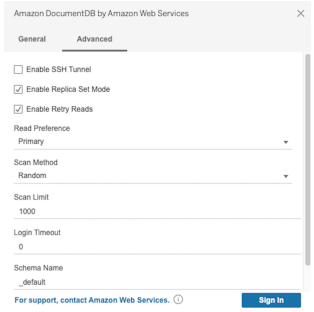
1. 输入所需的连接参数主机名、端口、数据库、用户名和密码。以下示例中的连接参数等同于 JDBC 连接字符串：

   ```
   jdbc:documentdb://localhost:27019/test?
   tls=true&tlsAllowInvalidHostnames=true&scanMethod=random&scanLimit=1000&logir
   ```

   在属性集合中分别传递用户名和密码参数。有关连接字符串参数的更多信息，请参阅Amazon DocumentDB JDBC 驱动程序 github 文档。



2. （可选）更多高级选项可在高级选项卡上找到。

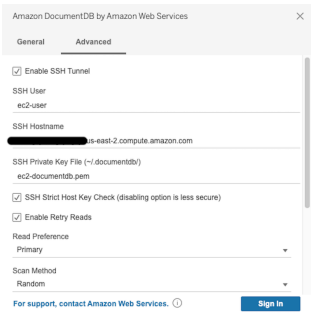


3. 选择登录。

使用 Tableau 连接到 Amazon DocumentDB：内部 SSH 隧道

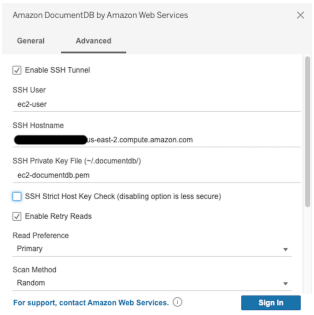
> **Note**
>
> 如果您不希望使用终端设置 SSH 隧道，则可以使用 Tableau GUI 来指定 EC2 实例详细信息，JDBC 驱动程序将使用这些详细信息自动创建一个 SSH 隧道。

1. 在高级选项卡上，选择启用 SSH 隧道选项以查看更多属性。



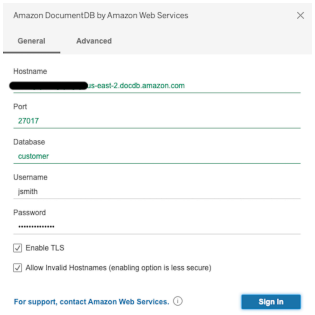
2. 输入 SSH 用户、SSH 主机名和 SSH 私钥文件。
3. （可选）您可以禁用 SSH 严格主机密钥检查选项，该选项会绕过针对已知主机文件的主机密钥检查。

> **Note**
>
> 禁用此选项安全性较低，因为它可能导致中间人攻击。



4. 输入所需的参数；主机名、端口、数据库、用户名和密码。

> **Note**
>
> 使用内部 SSH 隧道选项时，请确保使用 DocumentDB 集群端点而不是本地主机。



5. 选择登录。

从 Connect 到 Amazon DocumentDB DbVisualizer

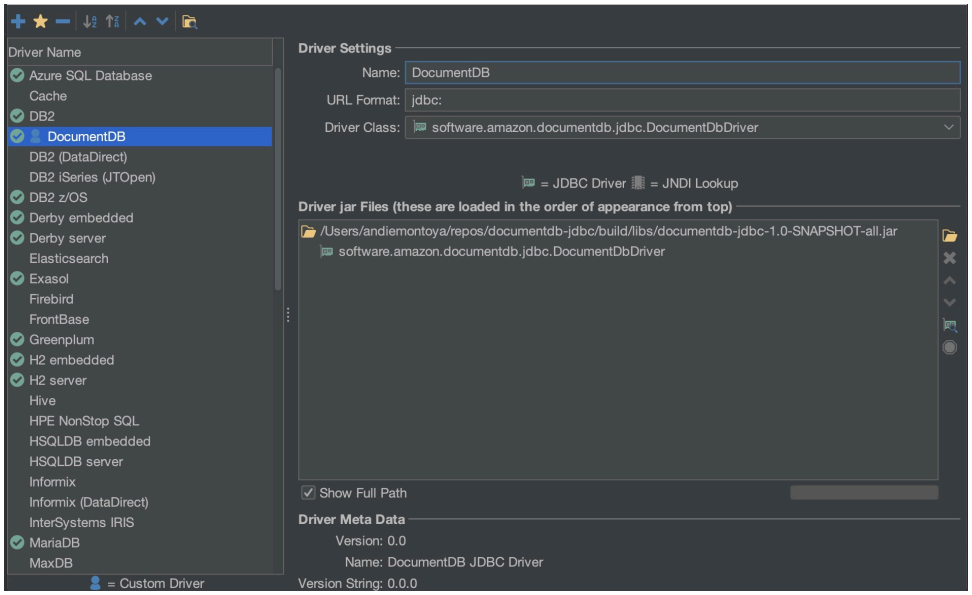
主题

- 添加 Amazon DocumentDB JDBC 驱动程序
- 使用连接亚马逊 DocumentDB DbVisualizer

添加 Amazon DocumentDB JDBC 驱动程序

要从中连接到亚马逊 DocumentDB，必须先导入 A DbVisualizer mazon DocumentDB JDBC 驱动程序

1. 启动 DbVisualizer 应用程序并导航到菜单路径：工具 > 驱动程序管理器...
2. 选择 + (或在菜单中选择驱动程序 > 创建驱动程序 ）。
3. 将名称设置为 DocumentDB。
4. 将 URL 格式设置为 `jdbc:documentdb://<host>[:port]/<database>[?option=value[&option=value[...]]]`
5. 选择文件夹按钮，然后选择 Amazon DocumentDB JDBC 驱动程序 JAR 文件并选择打开按钮。
6. 验证驱动程序类字段是否设置为 `software.amazon.documentdb.jdbc.DocumentDbDriver`。您的 DocumentDB 驱动程序管理器设置应如下所示。



7. 关闭对话框。Amazon DocumentDB JDBC 驱动程序将设置完毕并可供使用。

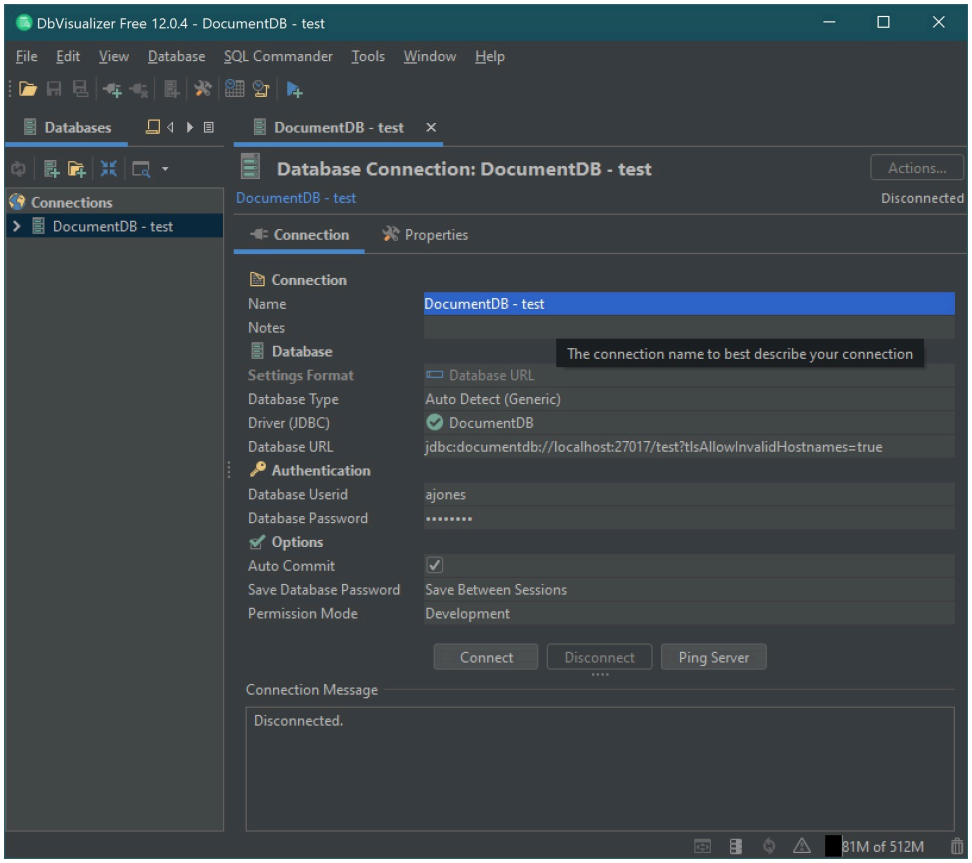
使用连接亚马逊 DocumentDB DbVisualizer

使用连接亚马逊 DocumentDB DbVisualizer

1. 如果您从 Amazon DocumentDB 集群的 VPC 外部进行连接，请确保您已设置 SSH 隧道。
2. 从顶级菜单中选择数据库 > 创建数据库连接。
3. 在名称字段中，输入一个描述性名称。
4. 将驱动程序 (JDBC) 设置为您在上一节中创建的 DocumentDB 驱动程序。
5. 将数据库 URL 设置为 JDBC 连接字符串。

   例如：`jdbc:documentdb://localhost:27017/database?tlsAllowInvalidHostnames=true`
6. 将数据库用户 ID 设置为您的 Amazon DocumentDB 用户 ID。
7. 将数据库密码设置为用户 ID 的相应密码。

   您的数据库连接对话框应类似于以下对话框：



8. 选择连接。

JDBC 自动生成架构

Amazon DocumentDB 是一个文档数据库，因此没有表和架构的概念。但是，像 Tableau 这样的商业智能工具会期望它所连接的数据库能够呈现架构。具体而言，当 JDBC 驱动程序连接需要获取数据库中集合的架构时，它将轮询数据库中的所有集合。驱动程序将确定该集合的架构的缓存版本是否已存在。如果缓存版本不存在，它将对文档集合进行采样，并基于以下行为创建架构。

主题

- 架构生成限制
- 扫描方法选项
- Amazon DocumentDB 数据类型
- 映射标量文档字段
- 对象和数组数据类型处理

架构生成限制

DocumentDB JDBC 驱动程序将标识符的长度限制为 128 个字符。架构生成器可能会截断生成的标识符（表名和列名）的长度，以确保它们符合该限制。

扫描方法选项

可以使用连接字符串或数据来源选项修改采样行为。

- scanMethod= <选项>
  - random -（默认）：按随机顺序返回示例文档。
  - idForward：按 id 顺序返回示例文档。
  - idReverse：按 id 的相反顺序返回示例文档。
  - all：对集合中的所有文档进行采样。
- scanLimit=<n>：要采样的文档数。该值必须为正整数。默认值是 1000。如果 scanMethod 设置为 all，则忽略此选项。

Amazon DocumentDB 数据类型

Amazon DocumentDB 服务器支持多种 MongoDB 数据类型。下面列出了支持的数据类型及其关联的 JDBC 数据类型。

MongoDB 数据类型	DocumentDB 支持	JDBC 数据类型
二进制数据	是	VARBINARY
布尔值	是	BOOLEAN
双精度	是	DOUBLE
32 位整数	是	INTEGER
64 位整数	是	BIGINT
字符串	是	VARCHAR
ObjectId	是	VARCHAR
日期	是	TIMESTAMP

MongoDB 数据类型	DocumentDB 支持	JDBC 数据类型
Null	是	VARCHAR
正则表达式	是	VARCHAR
Timestamp	是	VARCHAR
MinKey	是	VARCHAR
MaxKey	是	VARCHAR
对象	是	虚拟表
数组	是	虚拟表
Decimal128	否	DECIMAL
JavaScript	否	VARCHAR
JavaScript （带瞄准镜）	否	VARCHAR
未定义	否	VARCHAR
符号	否	VARCHAR
DBPointer (4.0+)	否	VARCHAR

映射标量文档字段

当扫描集合中的文档样本时，JDBC 驱动程序将创建一个或多个架构来表示集合中的样本。通常，文档中的标量字段会映射到表架构中的一列。例如，在名为 team 的集合和单个文档 `{ "_id" : "112233", "name" : "Alastair", "age": 25 }` 中，这将映射到架构：

表名称	列名称	数据类型	Key
team	team id	VARCHAR	PK
team	名称	VARCHAR	

表名称	列名称	数据类型	Key
team	age	INTEGER	

数据类型冲突提升

扫描样本文档时，文档之间的字段数据类型可能不一致。在这种情况下，JDBC 驱动程序会将 JDBC 数据类型提升为通用数据类型，该数据类型将适合采样文档中的所有数据类型。

例如：

```
{
"_id" : "112233",
"name" : "Alastair", "age" : 25
}

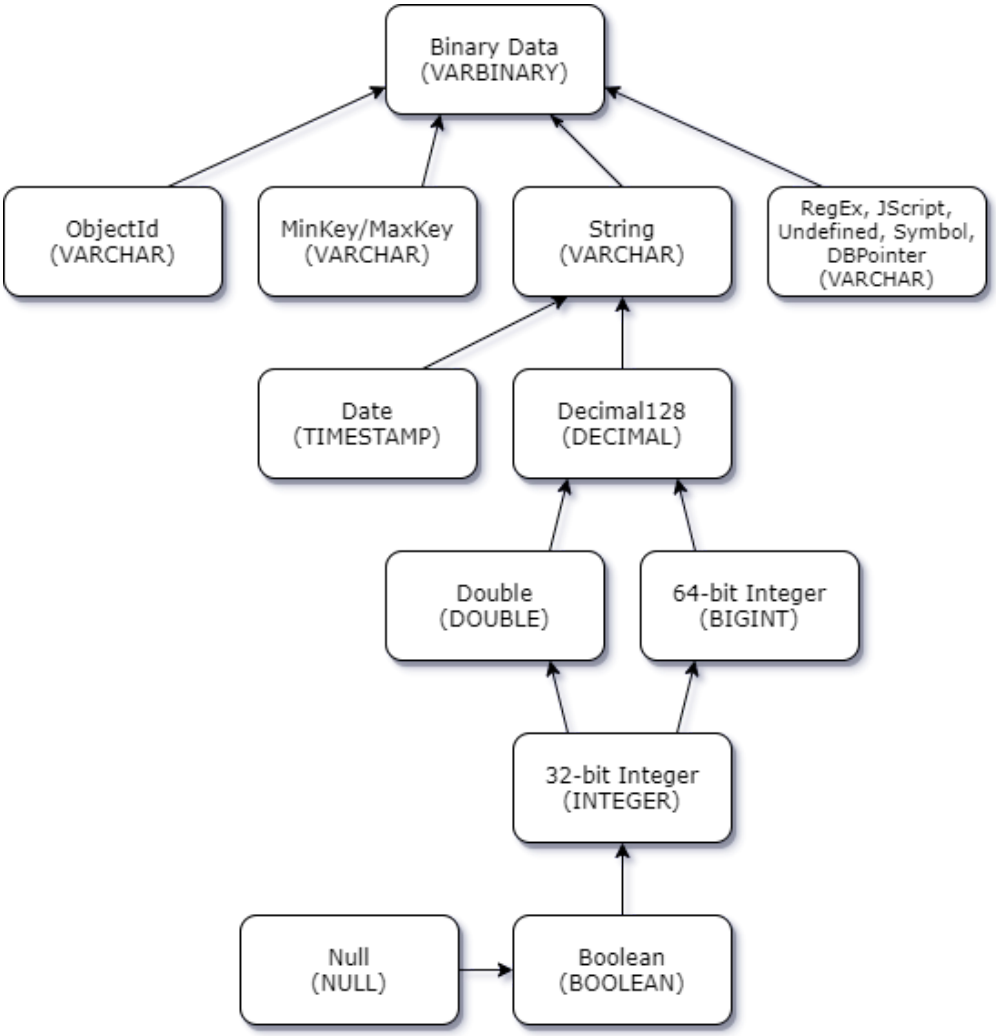
{
"_id" : "112244",
"name" : "Benjamin",
"age" : "32"
}
```

age 字段在第一个文档中为 32 位整数类型，但在第二个文档中为字符串。在这里，JDBC 驱动程序会将 JDBC 数据类型提升为 VARCHAR，以便在遇到任一数据类型时进行处理。

表名称	列名称	数据类型	Key
team	team id	VARCHAR	PK
team	名称	VARCHAR	
team	age	VARCHAR	

Scalar-scalar 冲突宣传

下图显示了标量-标量数据类型冲突的解决方法。



Scalar-complex 类型冲突推广

与标量-标量类型冲突一样，不同文档中的同一字段在复数（数组和对象）和标量（整数、布尔值等）之间可能存在冲突的数据类型。对于这些字段，所有这些冲突都已解决（提升）为 VARCHAR。在这种情况下，数组和对象数据以 JSON 表示形式返回。

嵌入式数组：字符串字段冲突示例：

```
{
   "_id":"112233",
   "name":"George Jackson",
   "subscriptions":[
      "Vogue",
      "People",
      "USA Today"
   ]
```

```
}
{
   "_id":"112244",
   "name":"Joan Starr",
   "subscriptions":1
}
```

前面的示例映射到 customer2 表的架构：

表名称	列名称	数据类型	Key
customer2	customer2 id	VARCHAR	PK
customer2	name	VARCHAR	
customer2	订阅	VARCHAR	

和 customer1_subscriptions 虚拟表：

表名称	列名称	数据类型	Key
customer1_subscriptions	customer1 id	VARCHAR	PK/FK
customer1_subscriptions	subscriptions_index_lvl0	BIGINT	PK
customer1_subscriptions	值	VARCHAR	
customer_address	city	VARCHAR	
customer_address	region	VARCHAR	
customer_address	country	VARCHAR	
customer_address	代码	VARCHAR	

对象和数组数据类型处理

到目前为止，我们只描述了标量数据类型的映射方式。对象和数组数据类型（当前）映射到虚拟表。JDBC 驱动程序将创建一个虚拟表来表示文档中的对象或数组字段。映射的虚拟表的名称将连接原始集合的名称，后跟由下划线字符（“_”）分隔的字段名称。

基表的主键（“_id”）在新虚拟表中采用新名称，并作为外键提供给关联的基表。

对于嵌入式数组类型字段，会生成索引列来表示数组每个级别的索引。

嵌入式对象字段示例

对于文档中的对象字段，JDBC 驱动程序会创建到虚拟表的映射。

```
{
   "Collection: customer",
   "_id":"112233",
   "name":"George Jackson",
   "address":{
      "address1":"123 Avenue Way",
      "address2":"Apt. 5",
      "city":"Hollywood",
      "region":"California",
      "country":"USA",
      "code":"90210"
   }
}
```

前面的示例映射到客户表的架构：

表名称	列名称	数据类型	Key
customer	customer id	VARCHAR	PK
customer	name	VARCHAR	

和 customer_address 虚拟表：

表名称	列名称	数据类型	Key
customer_address	customer id	VARCHAR	PK/FK

表名称	列名称	数据类型	Key
customer_address	address1	VARCHAR	
customer_address	address2	VARCHAR	
customer_address	city	VARCHAR	
customer_address	region	VARCHAR	
customer_address	country	VARCHAR	
customer_address	代码	VARCHAR	

嵌入式数组字段示例

对于文档中的数组字段，JDBC 驱动程序还会创建到虚拟表的映射。

```
{
   "Collection: customer1",
   "_id":"112233",
   "name":"George Jackson",
   "subscriptions":[
      "Vogue",
      "People",
      "USA Today"
   ]
}
```

前面的示例映射到 customer1 表的架构：

表名称	列名称	数据类型	Key
customer1	customer1 id	VARCHAR	PK
customer1	name	VARCHAR	

和 customer1_subscriptions 虚拟表：

表名称	列名称	数据类型	Key
customer1_subscriptions	customer1 id	VARCHAR	PK/FK
customer1_subscriptions	subscriptions_index_lvl0	BIGINT	PK
customer1_subscriptions	值	VARCHAR	
customer_address	city	VARCHAR	
customer_address	region	VARCHAR	
customer_address	country	VARCHAR	
customer_address	代码	VARCHAR	

SQL 支持和限制

Amazon DocumentDB JDBC 驱动程序是一个只读驱动程序，它支持部分 SQL-92 和一些常见的扩展。有关更多信息，请参阅 SQL 限制文档和 JDBC 限制文档。

问题排查

如果您在使用 Amazon DocumentDB JDBC 驱动程序时遇到问题，请参阅故障排除指南。

使用 Amazon DocumentDB ODBC 驱动程序进行连接

亚马逊 DocumentDB 的 ODBC 驱动程序为开发人员提供了 SQL-relational 一个接口，并支持通过 Power BI Desktop 和 Microsoft Excel 等商业智能工具进行连接。

有关更多详细信息，请参阅上的 Amazon DocumentDB ODBC 驱动程序文档。 GitHub

主题

- 开始使用
- 在 Windows 中设置 Amazon DocumentDB ODBC 驱动程序

- 从 Microsoft Excel 连接到 Amazon DocumentDB
- 从 Microsoft Power BI Desktop 连接到 Amazon DocumentDB
- 自动架构生成
- SQL 支持和限制
- 问题排查

开始使用

步骤 1：创建 Amazon DocumentDB 集群

如果您还没有 Amazon DocumentDB 集群，可以通过多种方式开始使用。

> **Note**
>
> Amazon DocumentDB 是一项仅针对虚拟私有云（VPC）的服务。如果您从集群 VPC 外部的本地计算机进行连接，则需要创建与 Amazon EC2 实例的 SSH 连接。在这种情况下，请按照使用 EC2 连接中的说明启动集群。有关 SSH 隧道以及何时可能需要隧道的更多信息，请参阅使用 SSH 隧道连接到 Amazon DocumentDB。

步骤 2：JRE 或 JDK 安装

根据您的 BI 应用程序，您可能需要确保计算机上安装了 64 位 JRE 或 JDK 安装版本 8 或更高版本。您可以在此处下载 Java SE 运行时系统环境 8。

第 3 步：下载 Amazon DocumentDB ODBC 驱动程序

在此处下载 Amazon DocumentDB ODBC 驱动程序。选择正确的安装程序（例如，documentdb-odbc-1.0.0.msi）。遵照安装指南。

步骤 4：使用 SSH 隧道连接到 Amazon DocumentDB

Amazon DocumentDB 集群部署在 Amazon Virtual Private Cloud (Amazon VPC) 中。部署在同一 Amazon VPC 中的 Amazon EC2 实例或其他 AWS 服务可以直接访问它们。此外，Amazon EC2 实例或其他 AWS 服务可以通过 VPC 对等互连在同一 AWS 地区或其他地区的不同 VPC 中访问亚马逊 DocumentDB。

但是，假设您的使用案例要求您（或您的应用程序）从集群的 VPC 外部访问您的 Amazon DocumentDB 资源。对于大多数不在与 Amazon DocumentDB 集群位于同一 VPC 中的虚拟机上运

行应用程序的用户来说，情况就是如此。从 VPC 外部连接时，您可以使用 SSH 隧道（也称为端口转发）来访问您的 Amazon DocumentDB 资源。

要创建 SSH 隧道，您需要一个与您的 Amazon DocumentDB 集群在同一 Amazon VPC 中运行的 Amazon EC2 实例。您可以使用同一 VPC 中的现有 EC2 实例作为集群，或创建一个集群。您可以通过在本地计算机上运行以下命令设置到 Amazon DocumentDB 集群 `sample-cluster.node.us-east-1.docdb.amazonaws.com` 的 SSH 隧道：

```
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
```

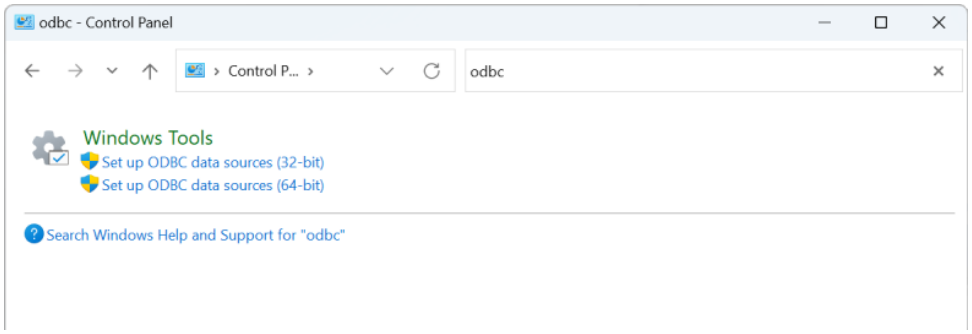
`-L` 标志用于转发本地端口。这是连接 VPC 外部客户端上运行的任何商业智能工具的前提条件。运行上述步骤后，您可以继续执行所选商业智能工具的后续步骤。

有关 SSH 隧道的更多信息，请参阅有关使用 SSH 隧道连接到 Amazon DocumentDB 的文档。

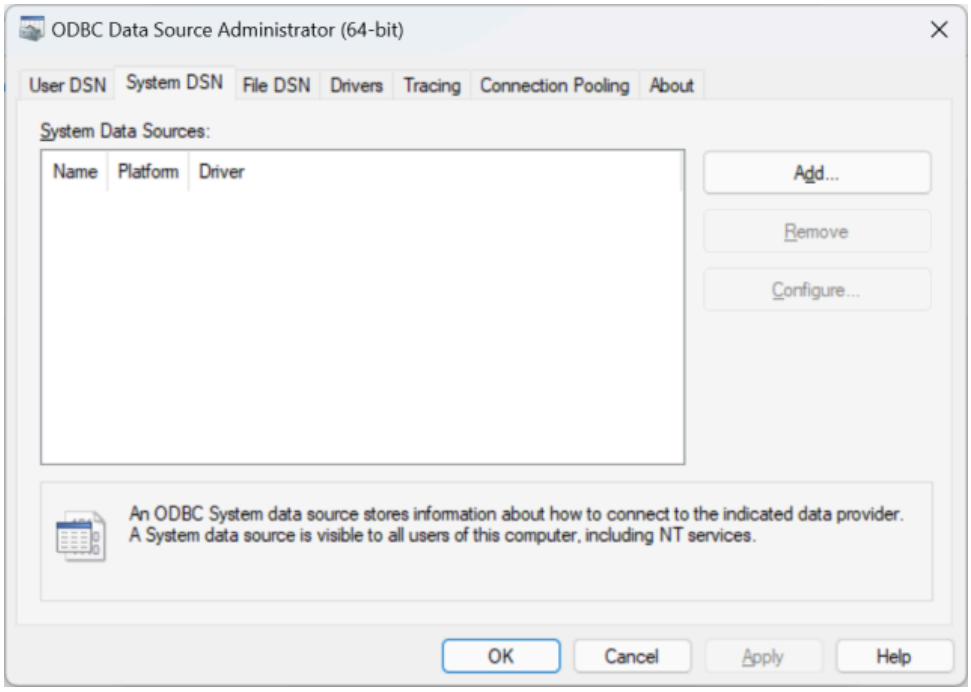
在 Windows 中设置 Amazon DocumentDB ODBC 驱动程序

使用以下步骤在 Windows 中设置 Amazon DocumentDB ODBC 驱动程序：

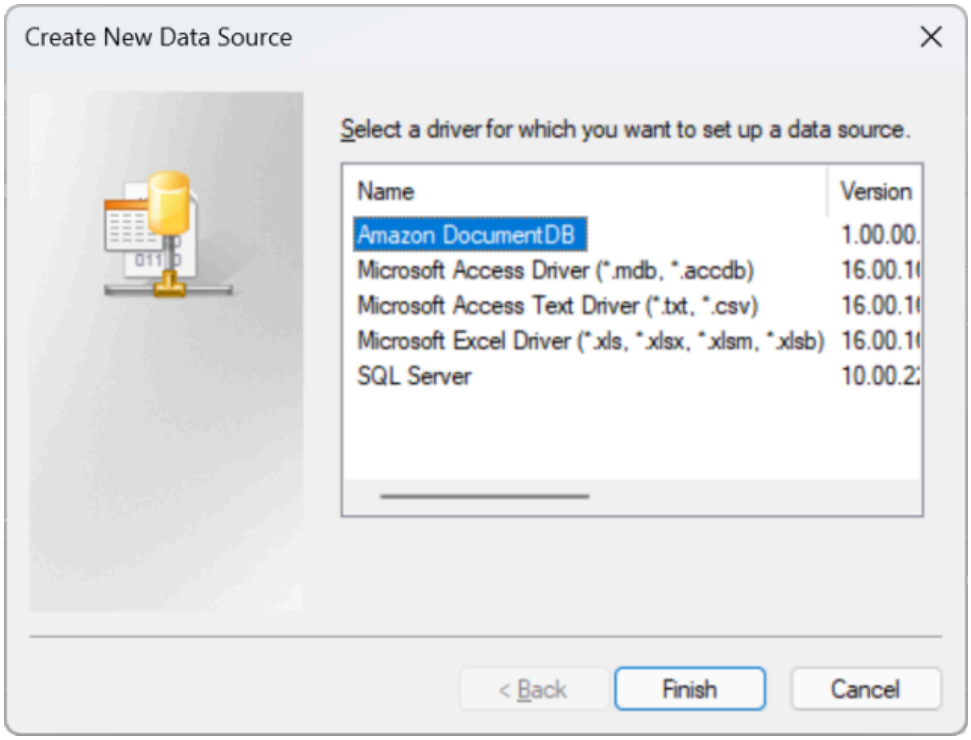
1. 在 Windows 中打开控制面板并搜索“ODBC”（或在菜单中选择Windows 工具>ODBC 数据来源（32 位）或ODBC 数据来源（64 位））：



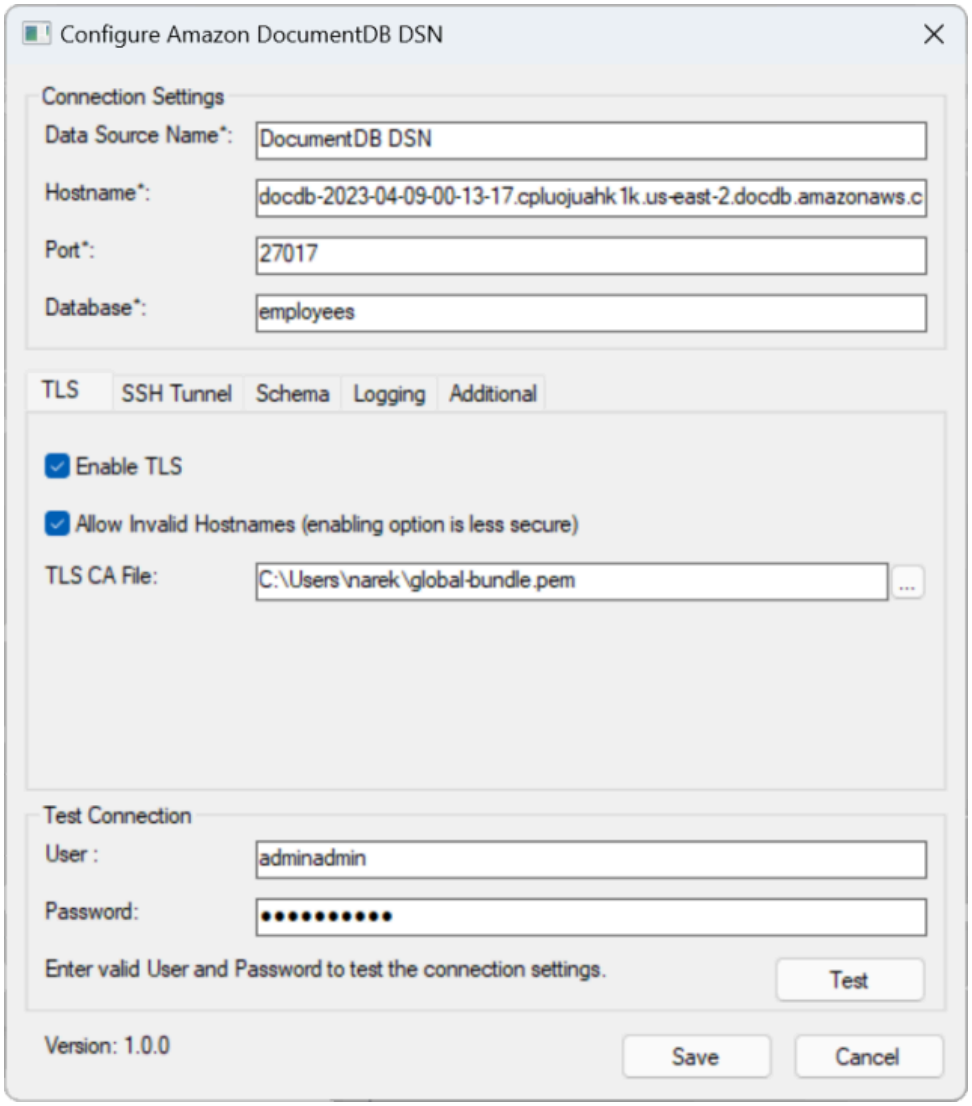
2. 选择适当的 ODBC 驱动程序数据来源管理员：如果已安装 32 位版本，请选择 32 位版本，否则，请选择 64 位版本。
3. 选择系统 DSN 选项卡，然后选择添加... 要添加新的 DSN，请执行以下操作：



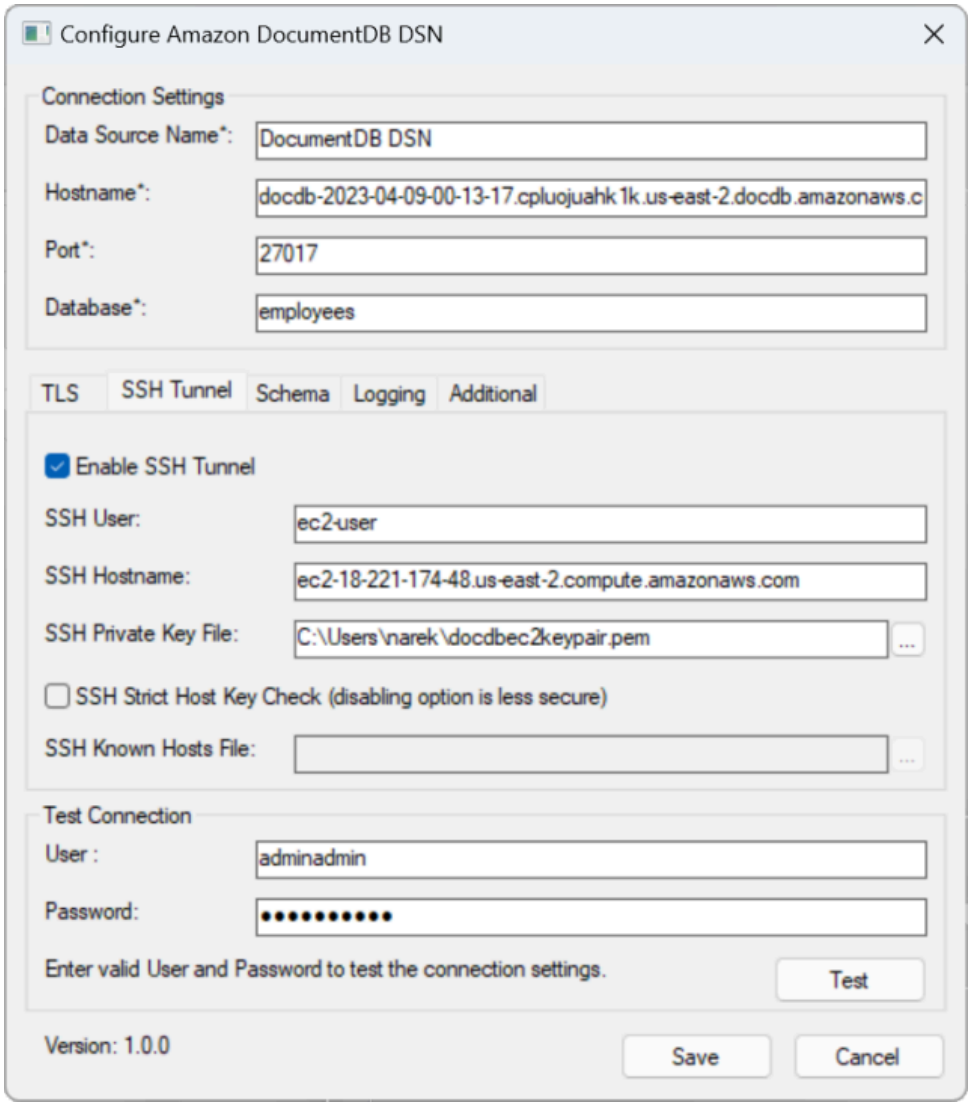
4. 从数据来源驱动程序列表中选择 Amazon DocumentDB：



5. 在 “配置 Amazon DocumentDB DSN” 对话框中，填写 “连接设置”、“TLS” 选项卡和 “测试连接” 字段，然后选择保存：



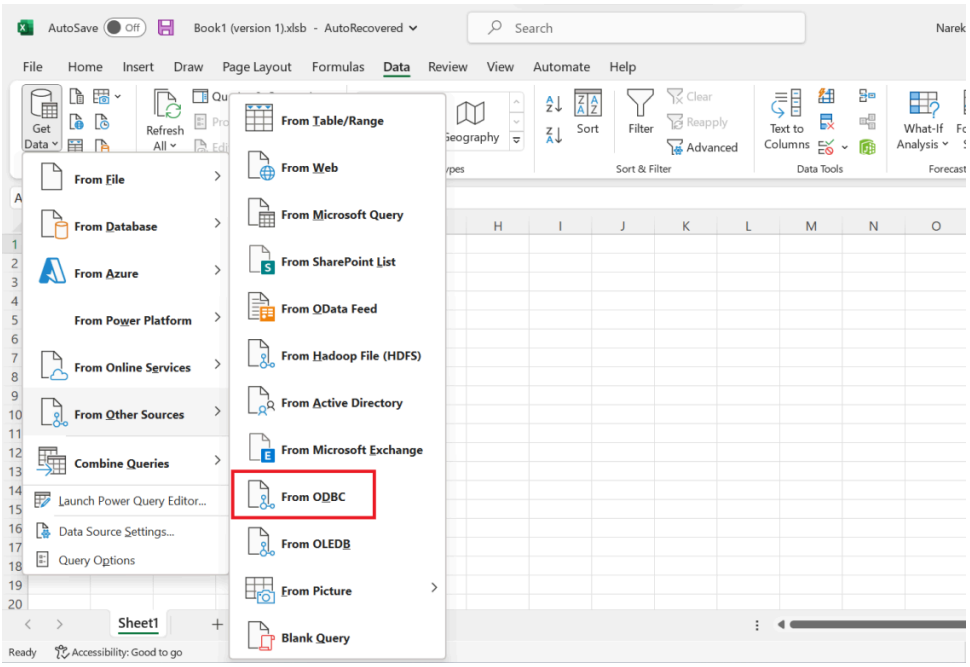
6. 确保您准确填写了 Windows 表单，因为连接详细信息会因您选择的连接 EC2 实例的 SSH 隧道方法而异。此处查看 SSH 隧道方法。有关每个属性的详细信息，请参阅连接字符串语法和选项。



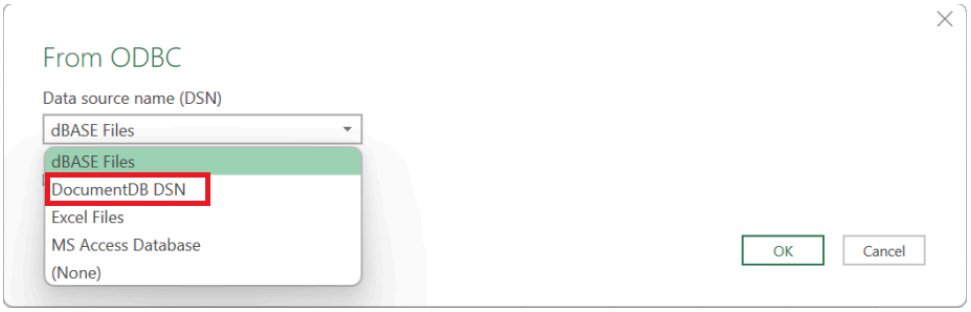
有关在 Windows 上配置 Amazon DocumentDB ODBC 驱动程序的更多信息，请参阅此处。

从 Microsoft Excel 连接到 Amazon DocumentDB

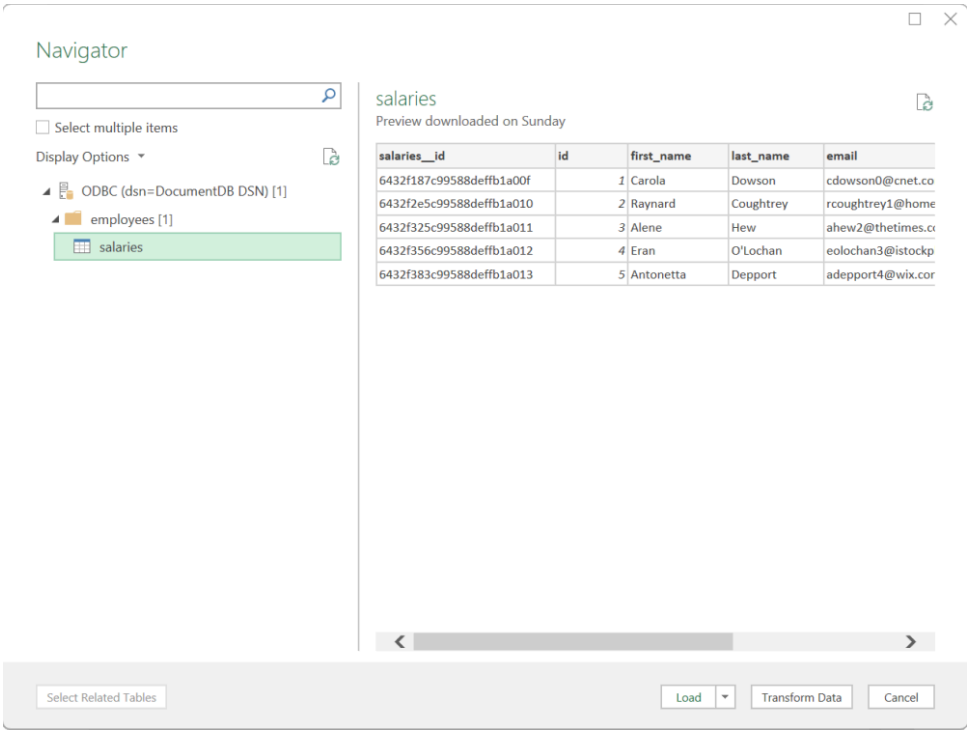
1. 确保已正确安装和配置 Amazon DocumentDB 驱动程序。有关其他信息，请参阅在 Windows 中设置 ODBC 驱动程序。
2. 启动 Microsoft Excel。
3. 导航至数据 > 获取数据 > 从其他来源。
4. 选择从 ODBC：



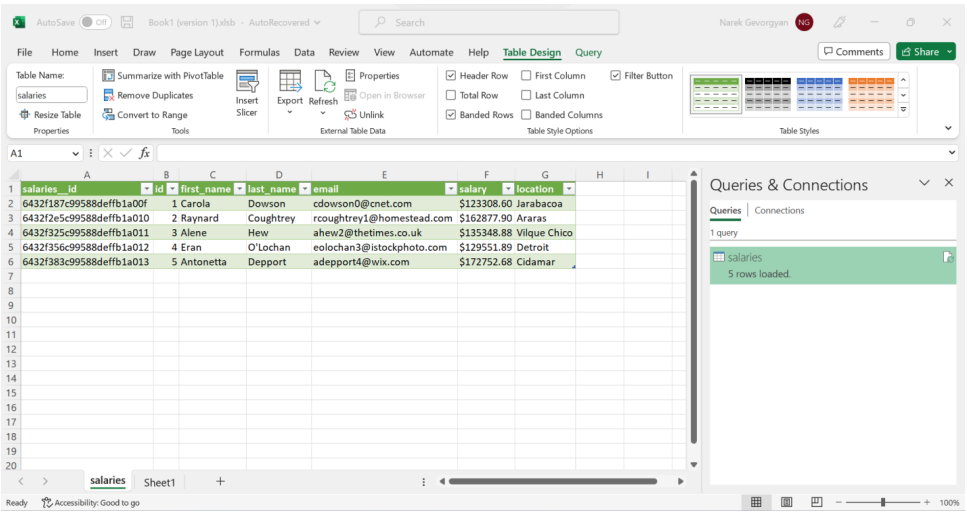
5. 从与 Amazon DocumentDB 关联的数据来源名称 (DSN)下拉菜单中选择数据来源：



6. 选择要从中将数据加载到 Excel 的集合：



7. 将数据加载到 Excel：



从 Microsoft Power BI Desktop 连接到 Amazon DocumentDB

主题

- 先决条件
- 添加 Microsoft Power BI Desktop 自定义连接器
- 使用 Amazon DocumentDB 自定义连接器进行连接
- 配置 Microsoft Power BI Gateway

先决条件

在开始之前，请确保已正确安装 Amazon DocumentDB ODBC 驱动程序。

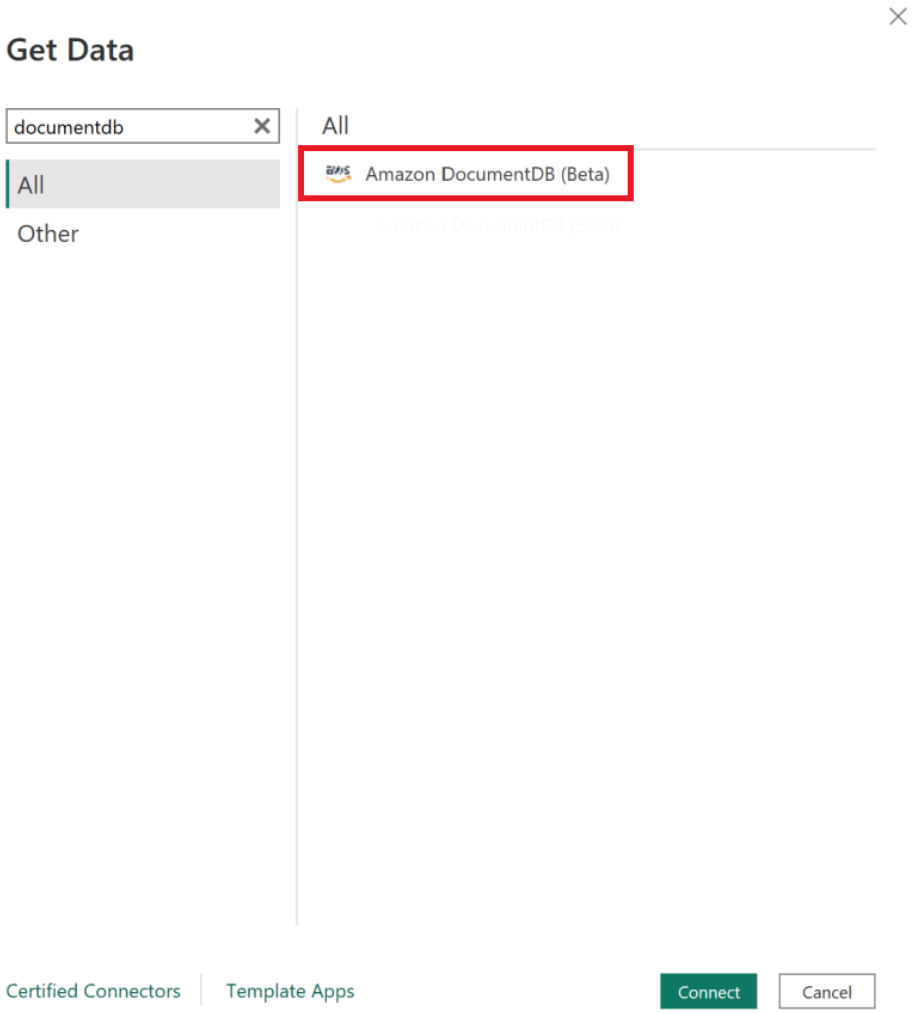
添加 Microsoft Power BI Desktop 自定义连接器

将AmazonDocumentDBConnector.mez文件复制到文件<User>\Documents\Power BI Desktop\Custom Connectors\夹（<User>\OneDrive\Documents\Power BI Desktop\Custom Connectors如果使用，则复制到 OneDrive）。这将允许 Power BI 访问自定义连接器。可以在此处获取 Power BI Desktop 的连接器。重启 Power BI Desktop 以确保连接器已加载。

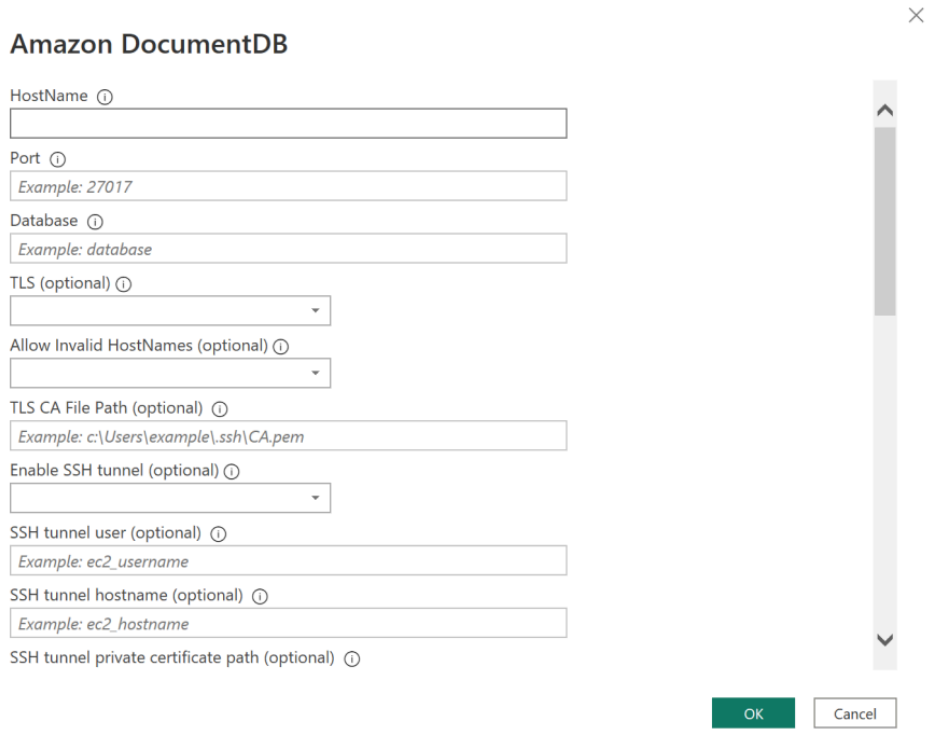
> **Note**
>
> 自定义连接器仅支持使用 Amazon DocumentDB 用户名和密码进行身份验证。

使用 Amazon DocumentDB 自定义连接器进行连接

1. 从 “获取数据” 中选择 Amazon DocumentDB（测试版），然后选择 Connect。如果您因使用第三方服务而收到警告，请选择 “继续”。



2. 输入连接您的 Amazon DocumentDB 集群所需的所有必要信息，然后选择确定：



> **Note**
>
> 根据 ODBC 驱动程序的数据来源名称 (DSN) 的配置，如果您已在 DSN 设置中提供了必要的信息，则可能不会显示 SSH 连接详细信息屏幕。

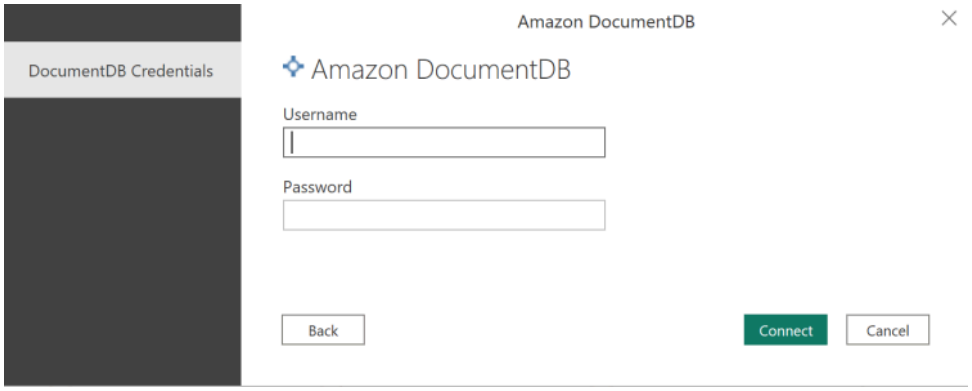
3. 选择数据连接模式：

   - 导入：加载所有数据并将信息存储在磁盘上。必须刷新并重新加载数据才能显示数据更新。
   - 直接查询：不加载数据，但对数据进行实时查询。这意味着无需刷新和重新加载数据即可显示数据更新。

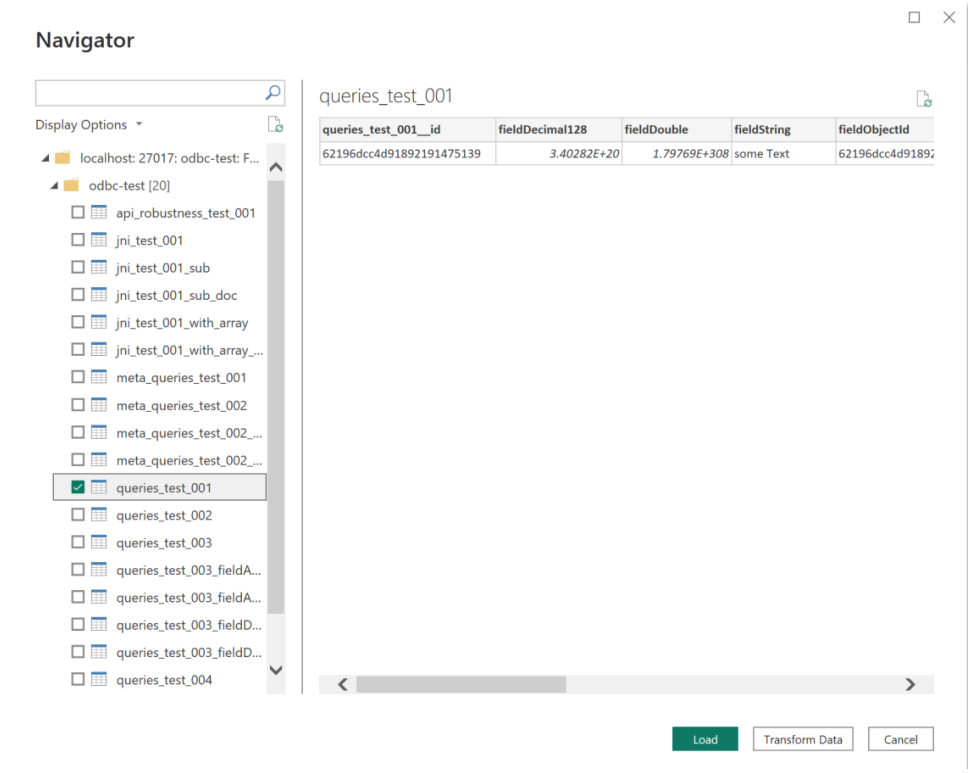


> **Note**
>
> 如果您使用的数据集非常大，则导入所有数据可能需要更长的时间。

4. 如果这是首次连接到此数据来源，请选择身份验证类型，并在出现提示时输入凭证。然后选择 Connect：



5. 在“导航”对话框中，选择所需的数据库表，然后选择“加载”加载数据，或者选择“转换数据”继续转换数据。



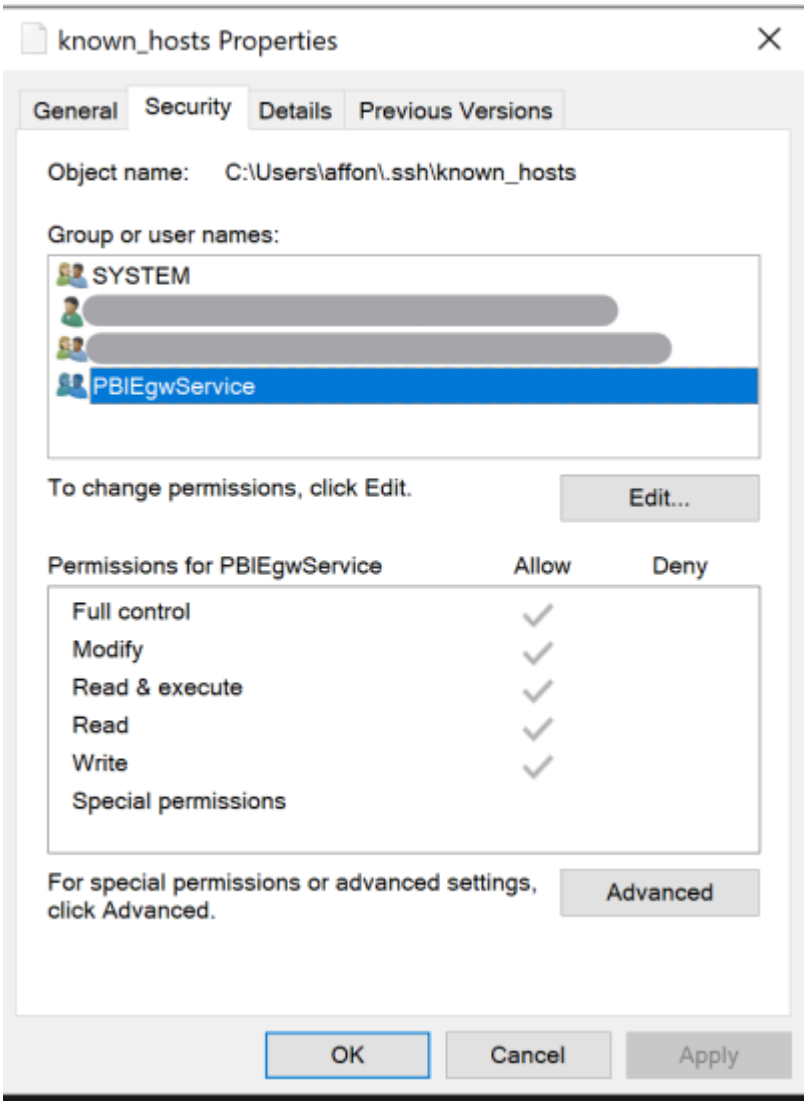
> **Note**
>
> 连接后，将保存数据来源设置。若要修改，请选择转换数据 > 数据来源设置。

配置 Microsoft Power BI Gateway

先决条件：

- 确保自定义连接器可与 Power BI Gateway 配合使用。
- 请确保 ODBC DS 创建在位于安装了 Power BI Gateway 的计算机的系统选项卡中的 ODBC 数据来源。

如果使用内部 SSH 隧道功能，则需要将文件 `known_hosts` 置于 Power BI 服务帐户有权访问的位置。



> **Note**
>
> 这也适用于您可能需要能够与 Amazon DocumentDB 集群建立连接的任何文件，例如证书颁发机构 (CA) 证书文件（pem 文件）。

自动架构生成

ODBC 驱动程序通过 JNI（Java 本机接口）使用 Amazon DocumentDB JDBC 驱动程序，使自动架构生成功能在 JDBC 驱动程序中类似地工作。有关自动架构生成的更多信息，请参阅 JDBC 自动架构生成。此外，要了解有关 ODBC 驱动程序架构的更多信息，请参阅此处。

SQL 支持和限制

Amazon DocumentDB ODBC 驱动程序是一个只读驱动程序，它支持部分 SQL-92 和一些常见的扩展。有关详细信息，请参阅 ODBC 支持和限制文档。

问题排查

如果您在使用 Amazon DocumentDB ODBC 驱动程序时遇到问题，请参阅故障排除指南。

使用 Amazon DocumentDB 进行编程

该服务支持 JSON 架构验证。

主题

- Amazon DocumentDB Java 编程指南
- 使用 JSON 架构验证

Amazon DocumentDB Java 编程指南

这个综合性指南详细介绍了如何使用 MongoDB 的 Java 驱动程序处理 Amazon DocumentDB，涵盖数据库操作与管理的基本方面。

主题

- 简介
- 先决条件
- 本指南的数据模型
- 使用 MongoDB Java 驱动程序连接到 Amazon DocumentDB
- 使用 Java 在 Amazon DocumentDB 中执行 CRUD 操作
- 使用 Java 在 Amazon DocumentDB 中进行索引管理
- Event-driven 使用亚马逊 DocumentDB 和 Java 编程

简介

本指南从连接开始，阐述如何使用 MongoDB Java 驱动程序建立到 DocumentDB 集群的安全连接。它详细介绍了连接字符串组件、 SSL/TLS 实现和各种连接选项，包括 IAM 身份验证和连接池，以及强大的错误处理策略。

在 CRUD（创建、读取、更新、删除）操作章节中，本指南全面介绍了文档操作，演示了如何使用单个和批量操作创建、读取、更新和删除文档。阐释了筛选条件、查询和各种操作选项的用法，同时强调了错误处理和实现重试逻辑以提高可靠性的最佳实践。本指南还广泛介绍了索引管理，详细说明了不同索引类型（包括单字段索引、复合索引、稀疏索引和文本索引）的创建和维护。阐释了如何通过正确选择索引和使用 `explain()` 函数分析查询执行计划，来优化查询性能。

最后一节重点介绍使用 Amazon DocumentDB 的变更流进行事件驱动型编程，演示如何在 Java 应用程序中实现实时数据变更监控。介绍了变更流光标的实现、用于持续操作的恢复令牌的处理以及用于历史数据处理的基于时间的操作。本指南中提供了实用的代码示例和最佳实践，可作为帮助您使用 Amazon DocumentDB 构建强大 Java 应用程序的宝贵资源。

先决条件

在开始之前，请确保您具有以下各项：

- 具有已配置文档数据库集群的 AWS 账户。请参阅此篇入门博客文章，了解 DocumentDB 集群设置。
- 已安装 Java 开发工具包（JDK）（在本指南中，我们将使用 Amazon Corretto 21）。
- 用于依赖项管理的 Maven。

本指南的数据模型

本指南中的所有示例代码都假设连接到具有 “RestaurantsProgGuideData” 集合的 “” 测试数据库。本指南中的所有示例代码都适用于餐厅列表系统，以下是该系统中文档的示例：

```
{
    "_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
    }
}
```

所有显示 CRUD、索引管理和事件驱动型编程的代码示例都假设您已具有一个 `MongoClient` 对象 `dbClient`、一个 `MongoDatabase` 对象 `connectionDB` 和一个 `MongoCollection` 对象 `collection`。

> **Note**
>
> 本指南中的所有代码示例均已使用 MongoDB Java 驱动程序版本 5.3.0 进行了测试。

使用 MongoDB Java 驱动程序连接到 Amazon DocumentDB

本节提供使用 Java 驱动程序连接到 Amazon DocumentDB 的分步指南。这将使您开始将 DocumentDB 集成到 Java 应用程序中。

主题

- 第 1 步：设置您的 项目

- 步骤 2：创建连接字符串
- 步骤 3：编写连接代码
- 步骤 4：处理连接异常
- 步骤 5：运行代码
- 连接最佳实践

第 1 步：设置您的 项目

1. 使用 Maven 创建 Java 项目：

```
mvn archetype:generate -DgroupId=com.docdb.guide -DartifactId=my-docdb-project -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

2. 将 MongoDB Java 驱动程序作为项目的依赖项添加到您的“pom.xml”文件中：

```
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-sync</artifactId>
    <version>5.3.0</version>
</dependency>
```

步骤 2：创建连接字符串

Amazon DocumentDB 连接字符串对于在您的应用程序与 DocumentDB 集群之间建立连接至关重要。此字符串封装了重要信息，例如集群端点、端口、身份验证详细信息以及各种连接选项。要构建 DocumentDB 连接字符串，通常需从基本格式开始：

```
"mongodb://username:password@cluster-endpoint:port/?[connection options]"
```

您需要将“用户名”和“密码”替换为您的实际凭证。你可以在中找到集群的终端节点和端口号 AWS 管理控制台 ，也可以通过 AWS CLI。要查找集群的集群端点，请参阅查找集群的端点。DocumentDB 的默认端口为 27017。

连接字符串示例

- 使用传输中加密建立到 DocumentDB 的连接，并确保将读取请求发送到只读副本，将写入请求发送到主项：

```
"mongodb://username:password@cluster-endpoint:27017/?tls=true&
   tlsCAFile=global-bundle.pem&
   readPreference=secondaryPreferred&
   retryWrites=false"
```

- 使用 IAM 身份验证建立到 DocumentDB 的连接：

```
"mongodb://cluster-endpoint:27017/?tls=true&
   tlsCAFile=global-bundle.pem&
   readPreference=secondaryPreferred&
   retryWrites=false&
   authSource=%24external&
   authMechanism=MONGODB-AWS"
```

可用于连接字符串的不同选项如下：

- TLS 证书
- 从只读副本读取
- 写入关注和日志
- RetryWrites
- IAM 身份验证
- 连接池
- 连接超时参数

TLS 证书

`tls=true|false`：此选项将启用或禁用传输层安全性协议（TLS）。默认情况下，在 Amazon DocumentDB 集群上启用传输中加密，因此，除非在集群级别禁用 TLS，否则此选项的值应为 `true`。

如果使用 TLS，在创建到 DocumentDB 集群的连接时，代码需要提供 SSL 证书。下载建立到集群的安全连接时所需要的证书：`global-bundle.pem`。可通过两种方式使用 `global-bundle.pem` 文件。

- 选项 1：从 `global-bundle.pem` 文件中提取所有证书，然后使用 Java 的 keytool 将其存储在 `.jks` 文件中，以便稍后在代码中使用。有关显示如何执行此操作的脚本，请参阅 启用了 TLS 的情况下的连接 中的 Java 选项卡。

- 选项 2：在代码中动态添加 `global-bundle.pem` 文件，构建内存密钥库，并在建立连接时使用 `SSLContext` 提供证书。

从只读副本读取

`replicaSet=rs0&readPreference=secondaryPreferred`：指定这两个选项会将所有读取请求路由到只读副本，将写入请求路由到主实例。在连接字符串中使用 `replicaSet=rs0` 可使 MongoDB 驱动程序维护自动更新的集群拓扑视图，从而允许应用程序在添加或移除实例时维护当前节点配置的可见性。未提供这些选项或指定 `readPreference=primary` 则会将所有读取和写入请求发送到主实例。有关 `readPreference` 的更多选项，请参阅 读取首选项选项。

写入关注和日志

写入关注决定了针对写入操作从数据库请求的确认级别。MongoDB 驱动程序提供了调整写入关注和日志文件的选项。Amazon DocumentDB 不希望您设置写入关注和日志，并且会忽略为 `w` 和 `j`（`writeConcern` 和 `journal`）发送的值。DocumentDB 始终使用 `writeConcern：majority` 和 `journal：true` 写入数据，因此在向客户端发送确认之前，写入会持久记录在大多数节点上。

RetryWrites

`retryWrites=false`：DocumentDB 不支持可重试写入，因此应始终将该属性设置为 `false`。

IAM 身份验证

`authSource=%24external`和 **`authMechanism=MONGODB-AWS`** — 这两个参数用于使用进行身份验证 AWS Identity and Access Management。IAM 身份验证目前仅在基于实例的集群版本 5.0 中可用。有关更多信息，请参阅 使用 IAM 身份进行身份验证。

连接池

以下选项可用于连接池：

- **`maxPoolSize`**：设置可以在池中创建的最大连接数。当所有连接都在使用中并且有新请求进来时，新请求需等待连接变为可用。MongoDB Java 驱动程序的默认值为 100。
- **`minPoolSize`**：表示应始终在池中维护的最小连接数。MongoDB Java 驱动程序的默认值为 0。
- **`maxIdleTimeMS`**：确定连接在被关闭和移除之前可以在池中保持空闲状态的时长。MongoDB Java 驱动程序的默认值为 100 毫秒。
- **`waitQueueTimeoutMS`**：配置当池达到其最大大小时，线程应等待连接变为可用的时长。如果连接未在此时间内变为可用，则抛出异常。MongoDB Java 驱动程序的默认值为 120000 毫秒（2 分钟）。

连接超时参数

超时是一种机制，用于限制操作或连接尝试在被认为失败之前所花费的时间量。以下超时参数可用于防止无限期等待及管理资源分配：

- **connectTimeoutMS**：配置驱动程序建立到集群的连接时的等待时长。默认值为 10,000 毫秒（10 秒）。
- **socketTimeoutMS**：指定驱动程序等待服务器对非写入操作的响应的时间长度。默认值为 0（无超时或无限期）。
- **serverSelectionTimeoutMS**：指定驱动程序在集群中查找可用服务器时的等待时长。此设置的默认值为 30 秒，足以在失效转移期间选择新的主实例。

步骤 3：编写连接代码

以下代码示例显示了如何建立到 Amazon DocumentDB 的 TLS 连接：

- 将创建 Java 的 KeyStore 和 SSLContext> 对象。
- 还将创建 MongoClientSettings 对象，方式是将其传递给 ConnectionString 对象。要建立 TLS 连接，必须使用 MongoClientSettings 对象绑定 connectionstring 和 sslcontext。
- 使用 MongoClients 获取 MongoClient 对象。

```
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);
    }
}
```

步骤 4：处理连接异常

在 Java 应用程序中使用 DocumentDB 时，处理连接异常对于保持稳健可靠的数据库操作而言至关重要。如果管理得当，这些异常不仅有助于快速诊断问题，还可以确保您的应用程序能够妥善处理临时网络中断或服务器不可用情况，从而提升稳定性和用户体验。与建立连接相关的一些关键异常包括：

- **MongoException**：一般异常，可能在各种未被更具体异常涵盖的场景中触发。请确保在处理所有其他具体异常后再处理此异常，因为这是一般的捕获所有 MongoDB 异常。
- **MongoTimeoutException**：在操作超时后触发。例如，查询不存在的集群端点。
- **MongoSocketException**：针对网络相关问题触发。例如，操作期间网络突然断开连接。
- **MongoSecurityException**：在身份验证失败时触发。例如，使用不正确的凭证进行连接。
- **MongoConfigurationException**：当客户端配置中存在错误时触发。例如，使用无效的连接字符串。

步骤 5：运行代码

以下代码示例将创建 Amazon DocumentDB 连接并打印所有数据库：

```
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);
    }
}
```

连接最佳实践

以下是使用 MongoDB Java 驱动程序连接到 Amazon DocumentDB 时应考虑的最佳实践：

- 当您不再需要客户端释放资源时，请务必关闭您的 `MongoClient`。
- 适当地处理异常并实现正确的错误日志记录。
- 使用环境变量或 AWS Secrets Manager 存储敏感信息，例如用户名和密码。

使用 Java 在 Amazon DocumentDB 中执行 CRUD 操作

本节讨论如何使用 MongoDB Java 驱动程序在 Amazon DocumentDB 中执行 CRUD（创建、读取、更新、删除）操作。

主题

- 在 DocumentDB 集合中创建和插入文档
- 从 DocumentDB 集合中读取和检索数据
- 更新 DocumentDB 集合中的现有文档
- 从 DocumentDB 集合中移除文档
- 使用重试逻辑进行错误处理

在 DocumentDB 集合中创建和插入文档

向 Amazon DocumentDB 插入文档可以让您向集合添加新数据。根据您的需求和正在处理的数据量，可采用多种方法执行插入。向集合插入单个文档的最基本方法是 `insertOne()`。要一次插入多个文档，可以使用 `insertMany()` 方法，该方法允许您在单个操作中添加文档数组。在 DocumentDB 集合中插入许多文档的另一个方法是 `bulkWrite()`。在本指南中，我们将讨论在 DocumentDB 集合中创建文档的所有方法。

insertOne()

让我们首先研究如何向 Amazon DocumentDB 集合插入单个文档。可使用 insertOne() 方法实现插入单个文档。此方法采用BsonDocument用于插入，并返回一个InsertOneResult可用于获取新插入文档的对象 ID 的对象 ID 的对象。以下示例代码显示了向集合插入一个餐厅文档：

```
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;
}
```

使用 insertOne() 时，请确保包含适当的错误处理。例如，在上面的代码中，“restaurantId”具有唯一索引，因此再次运行此代码将触发以下 MongoWriteException：

```
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()

用于向集合插入许多文档的主要方法是 insertMany() 和 bulkWrite()。

insertMany() 方法是在单个操作中插入多个文档的最简单方法。该方法接受文档列表并将其插入到集合中。当您要插入一批相互独立且无需任何特殊处理或混合操作的新文档时，此方法最为理想。以下代码显示了从文件中读取 JSON 文档并将其插入到集合中的过程。insertMany() 函数返回一个 InsertManyResultInsertManyResult 对象，该对象可用于获取所有已插入文档的 ID。

```
// 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() 方法允许在单个批处理中执行多个写入操作（插入、更新、删除）。当您需要在单个批处理中执行不同类型的操作时（例如在更新其他文档的同时插入一些文档），可以使用 bulkWrite()。bulkWrite() 支持两种类型的批处理写入，有序和无序：

- 有序操作：（默认）Amazon DocumentDB 按顺序处理写入操作，并在遇到首个错误时停止。当操作顺序至关重要时（例如后续操作依赖先前操作），有序操作很有用。但是，有序操作通常比无序操作慢。对于有序操作，您必须解决批处理在遇到首个错误时停止的情况，这可能会导致部分操作未被处理。
- 无序操作：允许 Amazon DocumentDB 将插入操作作为数据库中的单次执行来处理。如果一个文档出现错误，将继续对剩余文档执行操作。这在您要插入大量数据并且可以容忍某些失败时尤为有用，例如，在数据迁移或批量导入期间，某些文档可能会因为键重复而导致失败。对于无序操作，您必须处理部分成功场景，即部分操作成功而其他操作失败。

使用 bulkWrite() 方法时，需要一些基本类。首先，WriteModel 类充当所有写入操作的基类，并且具有特定的实现如

InsertOneModel、UpdateOneModel、UpdateManyModel、DeleteOneModel 和 DeleteManyModel，可用于处理不同类型的操作。

该BulkWriteOptions类是配置批量操作行为所必需的，例如设置 ordered/unordered 执行或绕过文档验证。BulkWriteResult 类提供有关执行结果的详细信息，包括已插入、已更新和已删除的文档的数量。

对于错误处理，MongoBulkWriteException 类至关重要，因为其包含有关批量操作期间所有失败的信息，而 BulkWriteError 类则提供有关单个操作失败的具体详细信息。以下代码显示了在执行单个 bulkWrite() 方法调用的过程中插入文档列表以及更新和删除单个文档的示例。该代码还显示了如何使用 BulkWriteOptions 和 BulkWriteResult，以及如何对 bulkWrite() 操作进行正确的错误处理。

```
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());
}
```

可重试写入

与 MongoDB 不同，Amazon DocumentDB 不支持可重试写入。因此，您必须在其应用程序中实现自定义的重试逻辑，尤其用于处理网络问题或临时服务不可用情况。通常，实现良好的重试策略包括增加重试之间的延迟和限制总重试次数。有关使用错误处理构建重试逻辑的代码示例，请参阅下面的 使用重试逻辑进行错误处理。

从 DocumentDB 集合中读取和检索数据

在 Amazon DocumentDB 中查询文档围绕几个关键组件展开，这些组件允许您精确检索和操作数据。`find()` 方法是 MongoDB Java 驱动程序中的基本查询 API。该方法允许复杂数据检索，并提供多种选项用于筛选、排序和投影结果。除了 `find()` 方法之外，`Filters` 和 `FindIterable` 是另外两个基本组件，为 MongoDB Java 驱动程序中的查询操作提供构建块。

`Filters` 类是 MongoDB Java 驱动程序中的一个实用程序类，提供流畅的 API 用于构造查询筛选条件。该类提供静态工厂方法，用于创建表示各种查询条件的 `Bson` 对象的实例。最常用的方法包括用于等值比较的 `eq()`，用于数值比较的 `gt()`、`lt()`、`gte()` 和 `lte()`，用于组合多个条件的 `and()` 和 `or()`，用于数组成员资格测试的 `in()` 和 `nin()`，以及用于用于模式匹配的 `regex()`。该类采用类型安全设计，与基于原始文档的查询相比，可提供更好的编译时检查，使其成为在 Java 应用程序中构造 DocumentDB 查询的首选方法。错误处理非常强大，对于无效的筛选条件构造，会抛出明确的异常。

`FindIterable` 是一个专用接口，旨在处理 `find()` 方法的结果。该接口提供一组丰富的方法来优化和控制查询执行，提供流畅的 API 以进行方法链接。该接口包含基本的查询修改方法，例如，`limit()` 用于限制返回的文档数量，`skip()` 用于分页，`sort()` 用于对结果排

序，projection() 用于选择特定字段，以及 hint() 用于选择索引。FindIterable 中的 batch、skip 和 limit 操作是基本的分页和数据管理工具，可帮助控制如何从数据库中检索和处理文档。

Batching（batchSize）控制单次网络往返中 DocumentDB 向客户端返回的文档数量。设置批处理大小时，DocumentDB 不会一次性返回所有匹配的文档，而是按指定的批处理大小分组返回。

Skip 允许您偏移结果的起始点，本质上是告知 DocumentDB 在开始返回匹配项之前跳过指定数量的文档。例如，skip(20) 将绕过前 20 个匹配文档。这通常用于要检索后续结果页面的分页场景。

Limit 限制可以从查询返回的文档总数。当您指定 limit(n) 时，即使数据库中存在更多匹配项，DocumentDB 也将在返回“n”个文档后停止返回文档。

从 Amazon DocumentDB 检索文档时，FindIterable 支持迭代器和光标模式。使用 FindIterable 作为迭代器的优势在于，允许文档延迟加载，并且仅在应用程序请求时才获取文档。使用迭代器的另一个优势在于，您无需负责维护到集群的连接，因此无需显式关闭连接。

FindIterable 还支持 MongoCursor，允许在处理 Amazon DocumentDB 查询时使用光标模式。MongoCursor 是特定于 MongoDB Java 驱动程序的实现，用于控制数据库操作和资源管理。其实现了 AutoCloseable 接口，允许通过 try-with-resources 块进行显式资源管理，这对于正确关闭数据库连接和释放服务器资源至关重要。默认情况下，光标会在 10 分钟后超时，并且 DocumentDB 不允许您选择更改此超时行为。使用批处理数据时，请确保在光标超时前检索下一批数据。使用 MongoCursor 时的一个关键考虑因素是，需要显式关闭以防止资源泄漏。

本节介绍了 find()、Filters 和 FindIterable 的几个示例。

以下代码示例显示了如何使用 find() 通过其“restaurantId”字段检索单个文档：

```
Document filter = new Document("restaurantId", "REST-21G145");
Document result = collection.find(filter).first();
```

尽管使用 Filters 可以更好地检查编译时错误，但 Java 驱动程序也允许您直接在 find() 方法中指定 Bson 筛选条件。以下示例代码可将 Bson 文档传递给 find()：

```
result = collection.find(new Document("$and", Arrays.asList(
    new Document("rating.totalReviews", new Document("$gt", 1000)),
    new Document("priceRange", "$$"))))
```

下一个示例代码显示了将 Filters 类与 find() 结合使用的几个示例：

```
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));
```

以下示例显示了如何在 FindIterable 对象上链接 sort()、skip()、limit()、和 batchSize() 操作。这些操作的提供顺序将影响查询性能。作为最佳实践，这些操作的顺序应为 sort()、projection()、skip()、limit() 和 batchSize()。

```
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);
```

以下示例代码显示了在 FindIterable 上创建迭代器。该代码使用 Java 的 forEach 构造遍历结果集。

```
collection.find(Filters.eq("cuisine", "American")).forEach(doc ->
 System.out.println(doc.toJson()));
```

在最后一个 find() 代码示例中，显示了如何使用 cursor() 进行文档检索。其在 try 块中创建光标，确保在代码退出 try 块时关闭光标。

```
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
```

更新 DocumentDB 集合中的现有文档

Amazon DocumentDB 提供了灵活而强大的机制，可修改现有文档，并在文档不存在时插入新文档。MongoDB Java 驱动程序提供了多种更新方法：updateOne() 用于单个文档更新，updateMany() 用于多个文档更新，以及 replaceOne() 用于完整文档替换。除了这三个方法之外，Updates、UpdateOptions 和 UpdateResult 是其他基本组件，为 MongoDB Java 驱动程序中的更新操作提供构建块。

MongoDB Java 驱动程序中的 Updates 类是一个实用程序类，提供用于创建更新运算符的静态工厂方法。其充当主要生成器，以类型安全且可读的方式构造更新操作。诸如 set()、unset()、和 inc() 之类的基本方法允许直接修改文档。当使用 Updates.combine() 方法（允许以原子方式执行多个更新操作，从而确保数据一致性）组合多个操作时，该类的功能优势尤为突出。

UpdateOptions 是 MongoDB 的 Java 驱动程序中的一个功能强大的配置类，为文档更新操作提供基本的自定义功能。该类的两个重要方面：为更新操作提供更新插入和数组筛选条件支持。通过 upsert(true) 启用的更新插入功能允许在更新操作期间未找到匹配文档时创建新文档。通过 arrayFilters()，更新操作可以精确更新满足特定条件的数组元素。

MongoDB 的 Java 驱动程序中的 UpdateResult 提供反馈机制，详细说明更新操作的结果。该类封装了三个关键指标：与更新条件匹配的文档数量（matchedCount）、实际修改的文档数量（modifiedCount）以及有关任何已更新插入文档的信息（upsertedId）。了解这些指标对于正确处理错误、验证更新操作和维护应用程序中的数据一致性而言至关重要。

更新并替换单个文档

在 DocumentDB 中，可以使用 updateOne() 方法来实现更新单个文档。该方法采用三个参数：通常由 Filters 类提供的用于标识要更新文档的 filter 参数、用于确定要更新字段的 Update 参数以及用于设置不同更新选项的可选 UpdateOptions 参数。使用 updateOne() 方法只会更新第一个符合选择条件的文档。以下示例代码将更新一个文档的单个字段：

```
collection.updateOne(Filters.eq("restaurantId", "REST-Y2E9H5"),
    Updates.set("name", "Amazing Japanese sushi"));
```

要更新一个文档中的多个字段，请结合使用 updateOne() 和 Update.combine()，如以下示例所示。此示例还显示了如何向文档中的数组添加项。

```
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);
```

以下代码示例演示了如何更新数据库中的文档。如果指定的文档不存在，则操作会自动将其作为新文档插入。此代码还显示了如何使用通过 UpdateResult 对象提供的指标。

```
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)");
}
```

以下代码示例演示了如何使用 replaceOne() 方法将现有文档完全替换为新文档，而非更新单个字段。replaceOne() 方法将覆盖整个文档，仅保留原始文档的 _id 字段。如果多个文档符合筛选条件，则仅替换首个匹配文档。

```
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());
```

更新多个文档

可通过两种方式同时更新集合中的多个文档。您可以使用 updateMany() 方法，或结合使用 UpdateManyModel 和 bulkWrite() 方法。updateMany() 方法使用 filter 参数来选择要更新的文档，使用 Update 参数来标识要更新的字段，使用可选的 UpdateOptions 参数来指定更新选项。

以下示例代码演示了 updateMany() 方法的使用：

```
Bson filter = Filters.and(
    Filters.in("features", Arrays.asList("Private Dining")),
    Filters.eq("cuisine", "Thai"));
UpdateResult result1 = collection.updateMany(filter, Updates.set("priceRange", "$$$"));
```

以下示例代码演示了使用相同更新的 bulkWrite() 方法：

```
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());
```

从 DocumentDB 集合中移除文档

MongoDB Java 驱动程序提供 deleteOne() 用于移除单个文档，并且提供 deleteMany() 用于移除符合特定条件的多个文档。与更新类似，删除操作也可以与 bulkWrite() 方法结合使用。deleteOne() 和 deleteMany() 都会返回 DeleteResult 对象，该对象提供有关操作结果的信息，包括已删除文档的数量。以下是使用 deleteMany() 移除多个文档的示例：

```
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());
```

使用重试逻辑进行错误处理

Amazon DocumentDB 强大的错误处理策略应将错误分为可重试错误（例如网络超时、连接问题）和不可重试错误（例如身份验证失败、无效查询）。对于因可重试的错误而导致的操作失败，可以实现

每次重试时间延迟以及最大重试次数。CRUD 操作应该位于捕获 MongoException 及其子类的 try-catch 块中。此外，还应包括错误的监控与日志记录，以实现操作可见性。以下是显示如何实现重试错误处理的示例代码：

```
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);
    }
}
```

使用 Java 在 Amazon DocumentDB 中进行索引管理

索引允许从 Amazon DocumentDB 集合中高效检索数据。如果没有索引，DocumentDB 必须扫描集合中的每个文档才能返回满足给定查询的结果。本主题提供有关如何使用 MongoDB Java 驱动程序创建、删除和列出索引的信息。还讨论了如何确定查询中是否使用了特定索引，以及如何向 Amazon DocumentDB 提供提示以使用特定索引。

主题

- 使用 Java 创建索引
- 删除索引
- 确定索引选择并提供索引提示

Amazon DocumentDB 支持多种类型的索引。有关所有受支持索引的全面概述，请参阅此博客文章。

使用 Java 创建索引

可通过两种机制使用 MongoDB Java 驱动程序在 Amazon DocumentDB 中创建索引：通过 runCommand()，以及通过 createIndex() 方法（创建单个索引）或 createIndexes() 方法（创建多个索引）。使用 createIndex() 和 createIndexes() 方法的原因之一是，通过捕获与创建索引相关的特定错误，您可以构建更好的错误处理。相较于 runCommand()，更推荐使用这些方法的另一个原因是，MongoDB Java 驱动程序为索引创建和操作提供了一组丰富的支持类。请注意，只有当您使用 createIndex() 或 createIndexes() 方法时才可以使用这些支持类。有三个支持类：

- **Indexes**：该类充当实用程序类，提供用于创建各种类型索引的静态工厂方法。该类简化了创建复杂索引定义的过程，通常和其他与索引相关的类一起使用。
- **IndexModel**：这是一个基本类，同时封装了索引键定义及其选项。其代表了一个完整的索引规范，将要索引的内容（键）与索引方式（选项）相结合。该类在同时创建多个索引时尤为有用，因为允许您定义可以传递给 createIndexes() 方法的索引规范集合。
- **IndexOptions**：这是一个全面的配置类，为自定义索引行为提供了一组丰富的方法。包含唯一索引、稀疏索引、过期时间（TTL）和部分筛选条件表达式的设置。通过方法链接，您可以配置后台索引构建和唯一约束等多个选项。

创建单个索引

此示例显示了如何在后台使用 createIndex() 方法创建单个索引。要了解背景和前台索引的创建，请参阅。索引构建类型以下代码示例用于IndexOptions创建后台名称为 “unique_restaurantId_idx” 的唯一索引。然后将此 IndexOptions 对象传递给 createIndex() 方法。

```
collection.createIndex(
    Indexes.ascending("restaurantId"),
    new IndexOptions()
        .unique(true)
        .name("unique_restaurantId_idx")
        .background(true));
```

创建多个索引

此示例使用 createIndexes() 方法创建多个索引。首先使用 IndexModel 对象为每个索引构建选项，然后将 IndexModel 对象列表传递给 createIndexes() 方法。以下代码示例显示了如何使用 Indexes 实用程序类创建复合索引。该类还用于指定要使用升序还是降序排序顺序创建索引。在创建多个索引后，该类会通过调用 listIndexes() 方法来验证索引创建情况。

```
// 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()));
```

创建稀疏索引和部分索引

此示例显示了通过为每种类型的索引创建 IndexModel，来创建稀疏索引和部分索引。

```
// 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))));
```

创建文本索引

此示例显示了如何创建文本索引。一个集合上只允许有一个文本索引，但该文本索引可以是覆盖多个字段的复合索引。在文本索引中使用多个字段时，也可以为索引中的每个字段分配权重。Amazon DocumentDB 不支持数组字段上的文本索引，尽管复合文本索引中最多可以使用 30 个字段，但只能为三个字段分配权重。

```
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());
```

使用 **runCommand()** 创建索引

Amazon DocumentDB 支持并行索引创建，以缩短创建索引所需要的时间。并行索引使用多个并发工作程序。用于创建索引的默认工作程序有两个。此篇博客文章对并行索引进行了深入讨论。当前，MongDB Java 驱动程序不支持您在使用 `createIndex()` 或 `createIndexes()` 时指定 worker 选项，因此指定工作程序的唯一方法是通过 `runCommand`。以下代码示例演示了如何使用 `runCommand` 创建可将工作程序增加到四个的索引：

```
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);
```

删除索引

MongoDB Java 驱动程序提供了多个可删除索引的方法，可满足不同的场景和您的首选项。您可以按名称、按键规范删除索引，也可以一次性删除所有索引。可以在集合对象上调用 dropIndex() 和 dropIndexes() 方法来删除索引。按名称删除索引时，应确保使用正确的索引名称，索引名称可能并不总是直观的，特别是对于复合索引或自动生成的索引而言。尝试删除不存在的索引将导致 MongoCommandException。不能删除 default _id 索引，因为该索引可以确保集合中文档的唯一性。

以下代码示例演示了如何通过提供创建索引的字段名称或通过删除所有索引来删除索引：

```
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();
```

使用多个键删除索引时，请确保存在一个包含所有指定键的复合索引，并且键的顺序正确。上面的索引创建示例代码显示了“cuisine”和 features 的复合键。如果您尝试删除该复合键，但顺序不是创建时使用的顺序，则会出现如下 MongoCommnadException 错误：

```
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);
}
```

将显示以下错误：

```
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
```

确定索引选择并提供索引提示

使用 Amazon DocumentDB 中的解释功能对于您了解查询性能和索引使用情况而言至关重要。执行查询时，您可以附加 explain() 方法以获取有关查询计划的详细信息，包括正在使用的索引（如果有）。explain() 输出提供了对查询执行阶段、检查的文档数量以及每个阶段所花费时间的洞察。这些信息对于确定特定索引是否得到有效使用或查询是否可以从不同的索引结构中获益而言非常有用。

explain() 方法可以与 find() 方法链接。explain() 方法可以采用可选的 ExplainVerbosity 枚举来确定 explain() 返回的详细程度级别。目前，DocumentDB 仅支持 EXECUTION_STATS 和 QUERY_PLANNER 枚举器。以下代码示例显示了如何获取特定查询的查询计划程序：

```
// 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());
```

针对查询计划程序详细程度级别返回以下 JSON 文档：

```
{
  "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
    }
  }
}
```

您可以通过多种方式来控制或强制 Amazon DocumentDB 使用特定索引。`hint()` 和 `hintString()` 方法允许您通过显式指定查询应使用的索引来覆盖查询优化程序的默认索引选择行为。虽然 DocumentDB 的查询优化程序通常能做出合理的索引选择，但在某些场景下，通过 `hint()` 或 `hintString()` 强制使用特定索引可能有益，例如在处理偏斜数据或测试索引性能时。

以下代码示例强制使用复合索引“cuisine_features_idx”来处理在上述代码中运行的相同查询：

```
// 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));
```

Event-driven 使用亚马逊 DocumentDB 和 Java 编程

Event-driven 在 Amazon DocumentDB 环境中进行编程代表了一种强大的架构模式，在这种模式中，数据库更改充当触发后续业务逻辑和流程的主要事件生成器。在 DocumentDB 集合中插入、更新或删除记录时，这些变更充当事件，用于自动启动各种下游进程、通知或数据同步任务。该模式在现代分布式系统中尤为重要，在这些系统中，多个应用程序或服务需要对数据变更做出实时反应。在 DocumentDB 中实现事件驱动型编程的主要机制是通过变更流。

> **Note**
>
> 本指南假设您已在使用的集合上启用了变更流。要了解如何在集合上启用变更流，请参阅 将变更流与 Amazon DocumentDB 结合使用。

通过 Java 应用程序处理变更流

MongoDB 的 Java 驱动程序中的 watch() 方法是监控 Amazon DocumentDB 中实时数据变更的主要机制。watch() 方法可以通过 MongoClient、MongoDatabase 和 MongoCollection 对象进行调用。

watch() 方法将返回支持各种配置选项的 ChangeStreamIterable 的实例，包括用于更新的完整文档查找、提供恢复令牌和时间戳以确保可靠性，以及用于筛选变更的管道聚合阶段。

ChangeStreamIterable 将实现核心 Java 接口 Iterable，可以与 forEach() 结合使用。要使用 forEach() 捕获事件，请将回调函数传递给处理变更事件的 forEach()。以下代码片段显示了如何在集合上打开变更流以启动变更事件监控：

```
ChangeStreamIterable < Document > iterator = collection.watch();
iterator.forEach(event - > {
    System.out.println("Received a change: " + event);
});
```

遍历所有变更事件的另一种方法是打开光标，该光标保持与集群的连接，并在发生新的变更事件时持续接收这些事件。要获取变更流光标，请使用 ChangeStreamIterable 对象的 cursor() 方法。以下代码示例显示了如何使用光标监控变更事件：

```
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch().cursor()) {
    System.out.println(cursor.tryNext());
}
```

作为最佳实践，请在 try-with-resource 语句中创建 MongoChangeStreamCursor，或手动关闭光标。在 ChangeStreamIterable 上调用 cursor() 方法将返回通过 ChangeStreamDocument 对象创建的 MongoChangeStreamCursor。

ChangeStreamDocument 类是表示流中各个变更事件的关键组件。包含有关每项修改的详细信息，包括操作类型（插入、更新、删除、替换）、文档键、命名空间信息以及完整的文档内容（如果有）。该类提供了访问变更事件各个方面的方法，例如 getOperationType() 用于确定变更类型，getFullDocument() 用于访问完整文档状态，以及 getDocumentKey() 用于识别被修改的文档。

ChangeStreamDocument 对象提供两条重要信息，即恢复令牌和变更事件发生时间。

DocumentDB 变更流中的恢复令牌和基于时间的操作为保持连续性和管理历史数据访问提供了关键机制。恢复令牌是为每个变更事件生成的唯一标识符，用作书签，允许应用程序在断开连接或发生故障后

从特定点重新启动变更流处理。创建变更流光标时，可以通过 resumeAfter() 选项使用先前存储的恢复令牌，使流能够从中断处继续，而非从头开始或丢失事件。

Time-based 变更流中的操作提供了不同的方法来管理变更事件监控的起点。startAtOperationTime() 选项允许您开始监视在特定时间戳或之后发生的变更。这些基于时间的功能在需要历史数据处理、时间点恢复或系统间同步的场景中尤为重要。

以下代码示例检索与插入文档关联的事件，捕获其恢复令牌，然后提供该令牌以开始监控插入事件之后的事件。该事件与更新事件关联，然后获取更新发生时的集群时间，并使用该时间戳作为进一步处理的起点。

```
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());
  }
```

默认情况下，更新变更事件不包含完整文档，仅包含所做变更。如果您需要访问已更新的完整文档，则可以在 ChangeStreamIterable 对象上调用 fullDocument() 方法。请记住，当您请求返回更新事件的完整文档时，会返回调用变更流时存在的文档。

此方法采用 `FullDocument` 枚举作为参数。目前，Amazon DocumentDB 仅支持 DEFAULT 和 UPDATE_LOOKUP 值。以下代码片段显示了在开始监视变更时如何请求提供更新事件的完整文档：

```
try (MongoChangeStreamCursor < ChangeStreamDocument < Document >> cursor =
 collection.watch().fullDocument(FullDocument.UPDATE_LOOKUP).cursor())
```

使用 JSON 架构验证

使用 `$jsonSchema` 评估查询运算符，您可以验证正在插入集合的文档。

主题

- 创建和使用 JSON 架构验证
- 受支持的关键词
- 旁路 DocumentValidation
- 限制

创建和使用 JSON 架构验证

用架构验证创建集合

您可以用 `createCollection` 操作和验证规则创建集合。这些验证规则在 Amazon DocumentDB 文档插入或更新期间适用。以下代码示例显示了员工集合的验证规则：

```
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"
} )
```

插入有效文档

以下示例插入符合上述架构验证规则的文档：

```
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})
```

插入无效文档

以下示例插入不符合上述架构验证规则的文档：在这个示例中，employeId 值不是字符串：

```
db.employees.insert({
    "name" : { "firstName" : "Carol" , "lastName" : "Smith"},
    "employeeId": 720 ,
    "salary": 1000.0
})
```

这个示例显示文档内部语法不正确。

修改集合。

collMod 命令用于添加或修改现有集合的验证规则。以下示例将薪金字段添加到必填字段列表中：

```
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
      }
   }
} )
```

处理在更改验证规则之前添加的文档

要处理在更改验证规则之前添加到您集合中的文档，请使用以下 validationLevel 修饰符：

- 严格：对所有插入和更新应用验证规则。
- 中等：向现有的有效文档应用验证规则。更新期间，不核查现有的无效文档。

在以下示例中，对名为“employees”的集合更新验证规则后，薪金字段为必填。更新以下文档将失败：

```
db.runCommand({
    update: "employees",
    updates: [{
        q: { "employeeId": "c721a" },
        u: { age: 25 , salary : 1000},
        upsert: true }]
})
```

Amazon DocumentDB 返回以下输出：

```
{
"n" : 0,
    "nModified" : 0,
    "writeErrors" : [
        {
"index" : 0,
            "code" : 121,
            "errmsg" : "Document failed validation"
        }
    ],
    "ok" : 1,
    "operationTime" : Timestamp(1234567890, 1)
}
```

将验证级别更新至 `moderate` 将允许成功更新上述文档：

```
db.runCommand({
    "collMod" : "employees",
    validationLevel : "moderate"
})

db.runCommand({
    update: "employees",
    updates: [{
        q: { "employeeId": "c721a" },
        u: { age: 25 , salary : 1000},
        upsert: true }]
})
```

Amazon DocumentDB 返回以下输出：

```
{
"n" : 1,
    "nModified" : 1,
    "ok" : 1,
    "operationTime" : Timestamp(1234567890, 1)
}
```

使用 $jsonSchema 检索文档

$jsonSchema 运算符可用作过滤器来查询与 JSON 架构匹配的文档。这是一个可以作为顶级字段出现在筛选器文档中或可以配合查询运算符（例如 $and、$or 和 $nor）一起使用的顶级运算符。以下示例显示了 $jsonSchema 如何作为单独过滤器使用以及配合其他过滤器运算符使用：

插入“员工”集合中的文档：

```
{ "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 }
```

仅用 $jsonSchema 运算符筛选出的集合：

```
db.employees.find({
       $jsonSchema: { required: ["age"] } })
```

Amazon DocumentDB 返回以下输出：

```
{ "_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 }
```

用 $jsonSchema 运算符和另一个运算符筛选出的集合：

```
db.employees.find({
       $or: [{ $jsonSchema: { required: ["age", "name"]}},
            { salary: { $lte:1000}}]});
```

Amazon DocumentDB 返回以下输出：

```
{ "_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 }
```

用 $jsonSchema 运算符及用聚合筛选器中 $match 筛选出的集合：

```
db.employees.aggregate(
    [{ $match: {
        $jsonSchema: {
            required: ["name", "employeeId"],
            properties: {"salary" :{"bsonType": "double"}}
        }
       }
    }]
)
```

Amazon DocumentDB 返回以下输出：

```
{
"_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
}
```

查看现有验证规则

要查看集合上的现有验证规则，请使用：

```
db.runCommand({
    listCollections: 1,
    filter: { name: 'employees' }
})
```

Amazon DocumentDB 返回以下输出：

```
{
    "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 DocumentDB 还在 $out 聚合阶段保留了验证规则。

受支持的关键词

create 和 collMod 命令中支持以下字段：

- **Validator**：支持 $jsonSchem 运算符。
- **ValidationLevel**：支持 off、strict 和 moderate 值。
- **ValidationAction**：支持 error 值。

$jsonSchema 运算符支持以下关键字：

- 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

旁路 DocumentValidation

Amazon DocumentDB 为以下命令和方法提供了 bypassDocumentValidation 支持：

- insert
- update
- findAndModify
- aggregate 命令和 db.collection.aggregate() 方法中的 $out 阶段

Amazon DocumentDB 未为以下命令提供 bypassDocumentValidation 支持：

- aggregate 命令和 db.collection.aggregate() 方法中的 $merge
- mapReduce 命令和 db.collection.mapReduce() 方法
- applyOps 命令

限制

以下限制适用于 $jsonSchema 验证：

- 当某操作未通过验证规则时，Amazon DocumentDB 返回“文档验证失败”错误。
- Amazon DocumentDB 弹性集群不支持 $jsonSchema。

Amazon DocumentDB 配额和限制

本主题介绍 Amazon DocumentDB（与 MongoDB 兼容）的资源配额、限制和命名约束。

对于某些管理功能，Amazon DocumentDB 使用与 Amazon Relational Database Service (Amazon RDS) 和 Amazon Neptune 共享的操作技术。

主题

- 支持的实例类型
- 支持的区域
- 区域配额
- 聚合限制
- 集群限制
- 实例限制
- 命名约束
- TTL 约束
- 弹性集群限制
- 弹性集群分片限制
- 每个分片的弹性集群 CPU、内存、连接和光标限制

支持的实例类型

Amazon DocumentDB 支持按需实例和以下实例类型：

- NVMe-backed:
  - R6GD 实例类型：
    db.r6gd.xlarge、db.r6gd.2xlarge、db.r6gd.4xlarge、db.r6gd.8xlarge、db.r6gd.12x
- 内存优化:
  - R8G 实例类
    型：db.r8g.large、、db.r8g.xlarge、db.r8g.2xlarge、、db.r8g.4xlarge、db.r8g.8xla
  - R6G 实例类
    型：db.r6g.large、、db.r6g.xlarge、、db.r6g.2xlarge、db.r6g.4xlargedb.r6g.8xlar

  - R5 实例类
    型：db.r5.large、、db.r5.xlarge、db.r5.2xlarge、db.r5.4xlarge、db.r5.8xlargedb.
  - R4 实例类
    型：db.r4.large、db.r4.2xlarge、db.r4.4xlarge、db.r4.8xlarge、db.r4.16xlarge。
- 可突增性能：
  - T4G 实例类型：db.t4g.medium。
  - T3 实例类型： db.t3.medium。

有关支持的实例类型及其规范的更多信息，请参阅实例类规格。

支持的区域

亚马逊 DocumentDB 可在以下 AWS 地区使用：

区域名称	Region	可用区（计算）
美国东部（俄亥俄州）	us-east-2	3
美国东部（弗吉尼亚州北部）	us-east-1	6
美国西部（俄勒冈州）	us-west-2	4
非洲（开普敦）	af-south-1	3
南美洲（圣保罗）	sa-east-1	3
亚太地区（香港）	ap-east-1	3
亚太地区（海得拉巴）	ap-south-2	3
亚太地区（马来西亚）	ap-southeast-5	3
亚太地区（孟买）	ap-south-1	3
亚太地区（大阪）	ap-northeast-3	3
亚太地区（首尔）	ap-northeast-2	4
亚太地区（新加坡）	ap-southeast-1	3
亚太地区（悉尼）	ap-southeast-2	3
亚太地区（雅加达）	ap-southeast-3	3
亚太地区（墨尔本）	ap-southeast-4	3

区域名称	Region	可用区（计算）
亚太地区（泰国）	ap-southeast-7	3
亚太地区（东京）	ap-northeast-1	3
加拿大（中部）	ca-central-1	3
加拿大西部（卡尔加里）	ca-west-1	3
中国（北京）区域	cn-north-1	3
中国（宁夏）	cn-northwest-1	3
欧洲地区（法兰克福）	eu-central-1	3
欧洲（苏黎世）	eu-central-2	3
欧洲地区（爱尔兰）	eu-west-1	3
欧洲地区（伦敦）	eu-west-2	3
欧洲地区（米兰）	eu-south-1	3
欧洲地区（巴黎）	eu-west-3	3
欧洲（西班牙）	eu-south-2	3
欧洲地区（斯德哥尔摩）	eu-north-1	3
墨西哥（中部）	mx-central-1	3
中东（阿联酋）：	me-central-1	3
以色列（特拉维夫）	il-central-1	3
AWS GovCloud (US-West)	us-gov-west-1	3

区域名称	Region	可用区（计算）
AWS GovCloud (US-East)	us-gov-east-1	3

区域配额

对于某些管理功能，Amazon DocumentDB 使用与 Amazon Relational Database Service (Amazon RDS) 共享的操作技术。下表包含在 Amazon DocumentDB 和 Amazon RDS 之间共享的区域限制。

> **Note**
>
> 上述 Amazon RDS 共享技术仅适用于 Amazon DocumentDB 基于实例的集群。Amazon DocumentDB 弹性集群不与 Amazon RDS 共享技术。

以下限制适用于基于 Amazon DocumentDB 实例的集群，并且每个区域的每个 AWS 账户都有以下限制。

资源	AWS 默认限制
集群	40
集群参数组	50
每个参数组的集群	100
每个参数组的实例数	300
事件订阅	20
实例	40
手动集群快照	100
每个集群的只读副本数	15
子网组	50

资源	AWS 默认限制
每个子网组的子网数	20
每个资源的标签	50
每个实例的 VPC 安全组数	5

以下限制适用于 Amazon DocumentDB 弹性集群，并且每个区域的每个 AWS 账户都有以下限制。

资源	AWS 默认限制
弹性集群	20
弹性集群 vCPU	1024
手动弹性集群快照	20

如果配额是可调整的，则可以使用服务配额来请求增加配额。有些请求会自动得到解决，而另一些则提交给 支持。您可以跟踪已提交给的增加配额请求的状态 支持。提高服务配额的请求没有得到优先支持。如果您有紧急请求，请联系支持。有关服务配额的更多信息，请参阅什么是服务配额？

要请求增加 Amazon DocumentDB 的限额，请执行以下操作：

1. 通过 https://console.aws.amazon.com/servicequotas 打开服务配额控制台，如有必要，请登录。
2. 在导航窗格中，选择 AWS 服务。
3. 从列表中选择 Amazon DocumentDB（兼容 MongoDB）或 Amazon DocumentDB 弹性集群，或者在搜索栏中键入任何一个条目。
4. 如果配额是可调整的，您可以选择其单选按钮或其名称，然后从页面右上角选择 Request quota increase (请求增加配额)。
5. 对于 Change quota value (更改配额值)，输入新值。新值必须大于当前值。
6. 选择请求。解决请求后，配额的 Applied quota value (应用的配额值) 设置为新值。
7. 要查看任何待处理或最近解决的请求，请从导航窗格选择 Dashboard (控制面板)。对于待处理的请求，请选择请求状态以打开收到的请求。请求的初始状态为 `Pending`。状态更改为后`Quota requested`，您将看到带有的案例编号 支持。选择案例编号以打开请求服务单。

聚合限制

下表介绍了 Amazon DocumentDB 中的聚合限制。

资源	限制
受支持的最大阶段数	500

集群限制

下表介绍了 Amazon DocumentDB 基于实例的集群的限制。

资源	限制
集群大小（所有集合的索引的总和）	128 TiB
集合大小（所有集合的总和不能超过集群限制）— 不包括索引大小	32 TiB
每集群的集合数	100000
每集群的数据库数	100000
数据库大小（所有数据库的总和不能超过集群限制）	128 TiB
文档嵌套深度	200 个级别
文档大小	16 MiB
索引键大小	2,048 字节
每个集合索引一次	64
复合索引中的键	32
单个批处理命令中的最大写入次数	100000
每个集群的用户数	1000

实例限制

下表介绍了每个实例中 Amazon DocumentDB 集群的限制。

Amazon DocumentDB 8.0

实例类型	实例内存 (GiB)	连接（全部）	光标限制	未完成事务	连接（活动）
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

实例类型	实例内存 (GiB)	连接（全部）	光标限制	未完成事务	连接（活动）
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

* 包括 R6GD

Amazon DocumentDB 5.0

实例类型	实例内存 (GiB)	连接（全部）	光标限制	未完成事务	连接（活动）
T3.medium	4	1000	30	50	102
T4G.medium	4	1000	30	50	102
R5.large	16	3400	450	200	1100

实例类型	实例内存 (GiB)	连接（全部）	光标限制	未完成事务	连接（活动）
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
R8G.2xlarge	64	14200	450	800	4500
R8G.4xlarge	128	28400	760	1600	4500

实例类型	实例内存 (GiB)	连接（全部）	光标限制	未完成事务	连接（活动）
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

* 包括 R6GD

Amazon DocumentDB 4.0

实例类型	实例内存 (GiB)	连接（全部）	光标限制	未完成事务	连接（活动）
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

实例类型	实例内存 (GiB)	连接（全部）	光标限制	未完成事务	连接（活动）
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

* 包括 R6GD

Amazon DocumentDB 3.6

实例类型	实例内存 (GiB)	连接（全部）	光标限制	未完成事务	连接（活动）
T3.medium	4	500	30	N/A	102
R4.large	15.25	1700	450	N/A	1100
R4.xlarge	30.5	3400	450	N/A	2700
R4.2xlarge	61	6800	450	N/A	4500
R4.4xlarge	122	13600	725	N/A	4500
R4.8xlarge	288	27200	1450	N/A	4500
R4.16xlarge	488	30000	2900	N/A	4500

实例类型	实例内存 (GiB)	连接（全部）	光标限制	未完成事务	连接（活动）
R5.large	16	1700	450	N/A	1100
R5.xlarge	32	3500	450	N/A	2700
R5.2xlarge	64	7100	450	N/A	4500
R5.4xlarge	128	14200	760	N/A	4500
R5.8xlarge	256	28400	1520	N/A	4500
R5.12xlarge	384	30000	2280	N/A	4500
R5.16xlarge	512	30000	3040	N/A	4500

您可以使用以下 CloudWatch 指标监控每个实例的限制并发出警报。有关亚马逊 DocumentDB CloudWatch 指标的更多信息，请参阅。使用以下方式监控亚马逊 DocumentDB CloudWatch

资源	CloudWatch 限制指标	CloudWatch 使用量指标（最大 1 分钟）	CloudWatch 使用量指标
实例内存	-	-	FreeableMemory
连接（全部）	DatabaseConnectionsLimit	DatabaseConnectionsMax	DatabaseConnections
游标	DatabaseCursorsLimit	DatabaseCursorsMax	DatabaseCursors
事务	TransactionsOpenLimit	TransactionsOpenMax	TransactionsOpen

命名约束

下表介绍 Amazon DocumentDB 中的命名约束。

资源	默认限制
集群标识符	• 长度为 [1-63] 个字母、数字或连字符。 • 第一个字符必须是字母。 • 不能以连字符结尾或包含两个连续的连字符。 • 每个区域的 AWS 每个账户的所有集群（跨亚马逊 RDS、Amazon Neptune 和 Amazon DocumentDB）必须是唯一的。
集合名称：<col>	长度为 [1-57] 个字符。
数据库名称：<db>	长度为 [1-63] 个字符。
完全限定集合名称：<db>.<col>	长度为 [3–120] 个字符。
完全限定索引名称：<db>.<col>$<index>	长度为 [6–377] 个字符。
索引名称：<col>$<index>	长度为 [3–255] 个字符。
实例标识符	• 长度为 [1-63] 个字母、数字或连字符 • 第一个字符必须是字母 • 不能以连字符结束或包含两个连续连字符 • AWS 每个账户、每个区域的所有实例（跨亚马逊 RDS、Amazon Neptune 和亚马逊 DocumentDB）必须是唯一的。

资源	默认限制
主密码	• 长度为 [8-100] 个可打印 ASCII 字符。 • 可以使用任何可打印 ASCII 字符，以下字符除外： • **/**（正斜杠） • **"**（双引号） • **@**（@ 符号）
主用户名称	• 长度为 [1-63] 个字母数字字符。 • 第一个字符必须是字母。 • 不能是数据库引擎的保留字。
参数组名称	• 长度为 [1-255] 个字母数字字符。 • 第一个字符必须是字母。 • 不能以连字符结尾或包含两个连续的连字符。

TTL 约束

在特定时间范围内无法保证从 TTL 索引中删除，只能尽力而为。实例资源利用率、文档大小和总体吞吐量等因素会影响 TTL 删除的时间。

弹性集群限制

下表介绍了 Amazon DocumentDB 弹性集群的最大限制。

资源	限制
每个区域的弹性集群	20
每个区域所有弹性集群的 vCPU 总和	1024

资源	限制
每个区域的手动集群快照	20
每个集群的分片数	32
每个集群的存储空间（当数据通过分片密钥均匀分布时）	4 PiB
集群的连接	300,000 或分片数量 x 与每个分片的 vCPU 相关的连接限制，取较低的值
UnSharded 收藏大小	32 TiB
分片集合大小（当数据通过分片键均匀分布时）	1PB
每集群的数据库数	10000
UnSharded 每个集群的集合	100000
每个集群的分片集群	1000
每个集群的用户	100
单个批处理命令中的写入次数	100000
每个集合索引一次	64
文档嵌套深度	100 个级别
文档大小	16 MB
索引键大小	2048 字节
复合索引中的键	32

弹性集群分片限制

下表介绍了 Amazon DocumentDB 弹性集群的最大分片限制。

资源	限制
每个分片实例的 vCPU	64
每个分片的实例	16
每个分片的存储	128 TiB
每个分片的每个集合的存储空间	32 TiB

每个分片的弹性集群 CPU、内存、连接和光标限制

下表描述了 Amazon DocumentDB 弹性集群分片中的最大 CPU、内存、连接和光标限制。

每个分片的 vCPU	实例内存 (GiB)	连接限制	光标限制
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 中的查询

本节介绍使用 Amazon DocumentDB 进行查询的所有方面。

主题

- 查询文档
- 查询计划
- 解释结果
- 查询计划器 v2
- 查询计划器 v3
- 通过 Amazon DocumentDB 查询地理空间数据
- 部分索引
- 执行 Amazon DocumentDB 文本搜索

查询文档

有时，您可能需要查看在线商店的库存，这样客户就能看到并购买您销售的物品。查询集合相对容易，无论您想要集合中的所有文档，还是仅需要那些满足特定标准的文档。

要查询文档，请使用 `find()` 操作。`find()` 命令具有单个文档参数，该参数定义了在选择要返回的文档时要使用的标准。`find()` 的输出是一个文档，其格式为一行文本，不含换行符。要格式化输出文档，从而更加轻松地读取，请使用 `find().pretty()`。本主题中的所有示例都使用 `.pretty()` 设置输出的格式。

下面的代码示例使用在前面两个练习中插入到 `example` 集合中的四个文档：`insertOne()` 和 `insertMany()`，它们位于使用文档的添加文档部分。

主题

- 检索集合中的所有文档
- 检索与字段值匹配的文档
- 检索与嵌入文档匹配的文档
- 检索与嵌入文档中的字段值匹配的文档
- 检索与数组匹配的文档
- 检索与数组中的值匹配的文档

- 使用运算符检索文档

检索集合中的所有文档

要检索集合中的所有文档，请将 `find()` 操作和空查询文档结合使用。

以下查询返回 `example` 集合中的所有文档。

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

检索与字段值匹配的文档

要检索与字段和值匹配的所有文档，请将 `find()` 操作和查询文档（标识要匹配的字段和值）结合使用。

通过使用前述文档，此查询将返回其中“Item”字段等于“Pen”的所有文档。

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

检索与嵌入文档匹配的文档

要查找与嵌入文档匹配的所有文档，请将 `find()` 操作和查询文档（指定嵌入文档名称和嵌入文档的所有字段和值）结合使用。

在与嵌入文档匹配时，该文档的嵌入文档的名称必须与查询中的名称相同。此外，嵌入文档中的字段和值必须与查询匹配。

以下查询仅返回“Poster Paint”文档。这是因为“Pen”具有不同的“OnHand”和“MinOnHand”值，并且“Spray Paint”比查询文档多一个字段 (`OrderQnty`)。

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

检索与嵌入文档中的字段值匹配的文档

要查找与嵌入文档匹配的所有文档，请将 `find()` 操作和查询文档（指定嵌入文档名称和嵌入文档的所有字段和值）结合使用。

考虑到上述文档，以下查询使用“点表示法”来指定嵌入文档和感兴趣的字段。将返回所有与这些内容匹配的文档，而不管嵌入文档中可能存在哪些其他字段。此查询将返回“Poster Paint”和“Spray Paint”，因为它们与指定的字段和值匹配。

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

检索与数组匹配的文档

要查找所有与数组匹配的文档，请将 `find()` 操作和您感兴趣的数组名称以及数组中的所有值结合使用。此查询将返回所有包含带该名称的数组（其中数组值和顺序与查询中的完全相同）的文档。

以下查询仅返回“Pen”，因为“Poster Paint”具有其他颜色 (White)，并且“Spray Paint”具有顺序不同的颜色。

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

检索与数组中的值匹配的文档

要查找所有具有特定数组值的文档，请将 `find()` 操作与您感兴趣的数组名称和值结合使用。

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

上述操作将返回所有三个文档，因为它们都有一个名为 `Colors` 的数组，并且此数组中的某个位置具有“`Red`”值。如果您指定值“`White`”，则查询将仅返回“Poster Paint”。

使用运算符检索文档

以下查询返回“`Inventory.OnHand`”值小于 50 的所有文档。

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

有关支持的查询运算符的列表，请参阅 查询和投影运算符。

查询计划

如何查看查询计划的 **Execut** ionStats？

在确定查询的执行速度低于预期速度的原因时，了解查询计划的 executionStats 会很有用。executionStats 提供从特定阶段返回的文档数量 (nReturned)、在每个阶段花费的执行时间 (executionTimeMillisEstimate) 以及生成查询计划所需的时间长度 (planningTimeMillis)。您可以确定查询中最耗时的阶段，以帮助将优化工作集中在的输出上executionStats，如以下查询示例所示。executionStats 参数当前不支持 update 和 delete 命令。

> **Note**
>
> Amazon DocumentDB 在利用分布式、容错、自修复的存储系统的专用数据库引擎上模拟 MongoDB 3.6 API。因此，查询计划和 explain() 的输出在 Amazon DocumentDB 和 MongoDB 之间可能有所不同。希望控制其查询计划的客户可以使用 $hint 运算符强制选择首选索引。

在 explain() 命令下运行要改进的查询，如下所示。

```
db.runCommand({explain: {query document}}).
explain("executionStats").executionStats;
```

以下是操作示例。

```
db.fish.find({}).limit(2).explain("executionStats");
```

此操作的输出将类似于下文。

```
{
    "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
}
```

如果您只想看到上面查询的 `executionStats`，您可以使用以下命令。对于较小的集合，如果性能增益微乎其微，Amazon DocumentDB 查询处理器可以选择不使用索引。

```
db.fish.find({}).limit(2).explain("executionStats").executionStats;
```

查询计划缓存

为了优化性能并缩短计划持续时间，Amazon DocumentDB 在内部缓存查询计划。这样，具有相同形状的查询就可以使用缓存计划直接执行。

但是，此缓存有时可能会导致同一查询的随机延迟;例如，通常需要 1 秒才能运行的查询有时可能需要 10 秒。这是因为随着时间的推移，读取器实例会缓存各种形状的查询，从而消耗内存。如果您遇到这种随机缓慢的情况，则无需执行任何操作即可释放内存：系统将为您管理内存使用量，一旦内存达到特定阈值，它将自动释放。

解释结果

如果要返回有关查询计划的信息，Amazon DocumentDB 支持详细程度模式 `queryPlanner`。`explain` 结果以类似于以下内容的格式返回优化程序选择的选定查询计划：

```
{
   "queryPlanner" : {
      "plannerVersion" : <int>,
      "namespace" : <string>,
      "winningPlan" : {
         "stage" : <STAGE1>,
         ...
         "inputStage" : {
            "stage" : <STAGE2>,
            ...
            "inputStage" : {
               ...
            }
         }
      }
   }
}
```

以下各节将定义常见的 `explain` 结果。

主题

- 扫描和过滤阶段
- 索引交集
- 索引并集
- 多索引 intersection/union
- 复合索引
- 排序阶段

- 小组阶段

扫描和过滤阶段

优化器可以选择以下扫描之一：

COLLSCAN

此阶段是顺序收集扫描。

```
{
    "stage" : "COLLSCAN"
}
```

IXSCAN

此阶段扫描索引键。优化程序可能会在此阶段内检索文档，这可能会导致稍后附加 FETCH 阶段。

```
db.foo.find({"a": 1})
{
    "stage" : "IXSCAN",
    "direction" : "forward",
    "indexName" : <idx_name>
}
```

FETCH

如果优化程序在 IXSCAN 以外的阶段检索文档，则结果将包括 FETCH 阶段。例如，上面的 IXSCAN 查询可能会导致 FETCH 和 IXSCAN 阶段的组合：

```
db.foo.find({"a": 1})
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXSCAN",
        "indexName" : <idx_name>
    }
}
```

IXONLYSCAN 仅扫描索引键。创建复合索引不会避免 FETCH。

索引交集

IXAND

如果 Amazon DocumentDB 可以利用索引交集，则可以包含具有 IXSCAN 的 inputStages 数组的 IXAND 阶段。例如，我们可能会看到如下输出：

```
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXAND",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                "indexName" : "a_1"
            },
            {
                "stage" : "IXSCAN",
                "indexName" : "b_1"
            }
        ]
    }
}
```

索引并集

IXOR

与索引交集类似，Amazon DocumentDB 可以包含 `IXOR` 阶段和 `$or` 运算符的 `inputStages` 数组。

```
db.foo.find({"$or": [{"a": {"$gt": 2}}, {"b": {"$lt": 2}}]})
```

对于上述查询，解释输出可能如下所示：

```
{
```

```
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXOR",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                "indexName" : "a_1"
            },
            {
                "stage" : "IXSCAN",
                "indexName" : "b_1"
            }
        ]
    }
}
```

多索引 intersection/union

Amazon DocumentDB 可以将多个索引交集或并集阶段组合在一起，然后获取结果。例如：

```
{
    "stage" : "FETCH",
    "inputStage" : {
        "stage" : "IXOR",
        "inputStages" : [
            {
                "stage" : "IXSCAN",
                ...
            },
            {
                "stage" : "IXAND",
                "inputStages" : [
                    {
                        "stage" : "IXSCAN",
                        ...
                    },
                    {
                        "stage" : "IXSCAN",
                        ...
                    }
                ]
            }
```

```
        ]
    }
}
```

索引交集或并集阶段的使用不受索引类型（稀疏、复合等）的影响。

复合索引

Amazon DocumentDB 复合索引的使用不受索引字段的起始子集的限制；它可以将索引与后缀部分一起使用，但可能不是很有效。

例如，`{ a: 1, b: -1 }` 的复合索引可以支持以下所有三个查询：

```
db.orders.find( { a: 1 } )

db.orders.find( { b: 1 } )

db.orders.find( { a: 1, b: 1 } )
```

排序阶段

如果请求的排序键上有索引，则 Amazon DocumentDB 可以使用该索引来获取顺序。在这种情况下，结果将不包括 `SORT` 阶段，而是包括 `IXSCAN` 阶段。如果优化程序偏向于普通排序，它将包括一个类似这样的阶段：

```
{
    "stage" : "SORT",
    "sortPattern" : {
        "a" : 1,
        "b" : -1
    }
}
```

小组阶段

Amazon DocumentDB 支持两种不同的组策略：

- `SORT_AGGREGATE`：在磁盘上对聚合进行排序。

- HASH_AGGREGATE: 在内存中对聚合进行哈希。

查询计划器 v2

Amazon DocumentDB 的新查询计划程序（计划程序版本 2.0）具有高级查询优化功能和提升的性能。将 `find` 和 `update` 运算符与索引结合使用后，适用于 Amazon DocumentDB 5.0 的计划程序版本 2.0 的性能较前一版本提升高达 10 倍。性能提升主要源于使用更优化的索引计划，以及为否定运算符（`$neq`、`$nin`）和嵌套的 `$elementMatch` 等运算符启用索引扫描支持。通过更优的成本估算技术、优化的算法和增强的稳定性，计划程序版本 2.0 查询可以更快速地运行。计划程序版本 2.0 还支持计划缓存筛选条件 API，可增强计划程序的稳定性。借助此功能，Amazon DocumentDB 5.0 现在可以从不同版本的查询计划程序中进行选择。

主题

- 先决条件
- 选择计划程序版本 2.0 作为默认查询计划程序
- 最佳实践
- 限制
- 对查找和更新运算符的改进
- 计划缓存筛选条件 API
- 计划程序版本 1.0、2.0 和 MongoDB 之间的潜在行为差异
- 计划程序版本 2.0 弥合了与 MongoDB 的行为差距

先决条件

以下先决条件适用于计划程序版本 2.0：

- 计划程序版本 2.0 已在所有提供引擎版本 5.0 的区域中推出。
- 要选择使用版本 2.0 作为默认查询计划程序，您的集群需要运行 Amazon DocumentDB 版本 5.0 的引擎补丁版本 3.0.15902 或更高版本。有关更新到最新引擎版本补丁的步骤，请参阅 对集群的引擎版本执行补丁更新。
- 要将计划程序版本 2.0 设置为默认查询计划程序，您需要具有更新集群参数组的 IAM 权限。
- 对于亚马逊 DocumentDB 8.0，计划器版本 3.0 是默认的查询计划器。有关详细信息，请参阅查询计划器 v3。

选择计划程序版本 2.0 作为默认查询计划程序

使用以下步骤从控制台或 CLI 中选择 2.0 作为默认查询计划程序：

- 按照 修改 Amazon DocumentDB 集群参数 中的步骤修改集群的参数组。
- 对于名为“plannerVersion”的参数，将值更改为 2.0，表示计划程序版本 2.0。
- 选择立即应用（若选择重启时应用，则在集群下次重启后才会生效）。

最佳实践

要获得预期结果，请在应用计划程序版本 2.0 时遵循以下最佳实践：

- 在全局集群中，在两个区域的集群参数组中选择相同的 `plannerVersion` 值（1.0 或 2.0）。请注意，在主要区域和辅助区域中选择不同的计划程序版本可能会导致查询行为和性能不一致。
- 在定期维护时段或流量减少期间更新到计划程序版本 2.0 可最大限度地减少中断，因为如果在工作负载活跃运行时更改计划程序版本，可能会导致错误率增加。
- 计划程序版本 2.0 与 MongoDB Shell 版本 5.0 配合使用效果最佳。

限制

以下限制适用于计划程序版本 2.0：

- 弹性集群不支持计划程序版本 2.0，将回退至计划程序版本 1.0。
- 聚合和不同命令不支持计划程序版本 2.0，将回退至计划程序版本 1.0。
- 计划程序版本 2.0 中的计划缓存筛选条件不支持包含以下内容的查询：正则表达式、文本搜索、地理空间、jsonschema 或筛选条件中的 `$expr`。

对##和##运算符的改进

计划程序版本 2.0 优化了基本操作，包括 `find`、`update`、`delete` 和 `find-and-modify` 命令。以下选项卡式部分显示了计划程序版本 2.0 中增强的索引功能以及查询性能提升：

Enhanced index support

- 计划程序版本 2.0 增加了对否定运算符的索引支持，包括 `$nin`、`$ne`、`$not {eq}` 和 `$not {in}` 以及 `$type` 和 `$elemMatch`。

```
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 }}}}})
```

- 即使查询表达式中没有 $exists，计划程序版本 2.0 版本也会使用稀疏索引或部分索引。

```
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" })
```

- 即使查询条件与部分索引筛选条件表达式不完全匹配，计划程序版本 2.0 也将使用部分索引。

```
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}})
```

- 计划程序版本 2.0 使用含 $elemMatch 查询的部分索引扫描。

```
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}}})
```

- 计划程序版本 2.0 包括针对 $regex 的索引扫描支持，无需在您的应用程序代码中提供 $hint。$regex 仅支持对前缀搜索进行索引。

```
Sample Document: { "x": [1, 2, 3], "y": "apple" }
db.foo.createIndex({ "x": 1, "y": 1 })
db.foo.find({"y":{ $regex: "^a" }})
```

- 计划程序版本 2.0 提升了涉及多键索引的查询性能，尤其优化了多键字段的等值条件查询。

```
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)
```

- 计划程序版本 2.0 提升了涉及多个筛选条件的查询性能，尤其针对文档大于 8KB 的集合。

```
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}}]})
```

- 计划程序版本 2.0 通过消除排序阶段，在将 $in 运算符与复合索引结合使用后提升了查询性能。

```
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})
```

它还提高了使用带$in元素的多键索引的查询的性能。

```
Sample Document: {"x": [1, 2, 3]}
db.foo.createIndex({"x": 1})
db.foo.find("x":{$in:[>100 elements]})
```

Query performance improvements

- 计划程序版本 2.0 提升了涉及多个筛选条件的查询性能，尤其针对文档大于 8KB 的集合。

```
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}}]})
```

- 计划程序版本 2.0 通过消除排序阶段，在将 $in 运算符与复合索引结合使用后提升了查询性能。

```
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})
```

还提升了使用包含 $in 元素的多键索引的查询性能。

```
Sample Document: {"x": [1, 2, 3]}
db.foo.createIndex({"x": 1})
db.foo.find("x":{$in:[>100 elements]})
```

计划缓存筛选条件 API

> **Note**
>
> 计划缓存筛选条件不支持文本索引。

- 计划程序版本 2.0 增加了对索引筛选条件功能的支持，该功能允许您指定特定查询形状可以使用的索引列表。此功能可通过 API 访问，可以从服务器端进行控制。如果您遇到查询回归，此功能可为您提供更快速、更灵活的选项，无需修改应用程序代码即可缓解问题。

```
db.runCommand({ planCacheSetFilter: <collection>, query: <query>,
sort: <sort>, // optional,
indexes: [ <index1>, <index2>, ...],
comment: <any> // optional})
```

要列出集合上的所有筛选条件，请使用以下命令：

```
db.runCommand(
{
planCacheListFilters: <collection>
}
)
```

此命令将显示集合上的所有索引筛选条件。输出示例：

```
{
"filters" : [
{
"query" : {a: "@", b: "@"},
"sort" : {a: 1},
"indexes" : [
<index1>,
...
]
},
...
],
"ok": 1
}
```

- 您可以使用 explain 命令输出中的两个新字段来分析计划程序版本 2.0 的索引筛选：indexFilterSet 和 indexFilterApplied。如果集合上设置了与查询形状相匹配的索引筛选条件，则将 indexFilterSet 设置为“true”。当且仅当查询应用了索引筛选条件并使用筛选条件列表中的索引选择了计划时，才会将 indexFilterApplied 设置为“true”。

  您可以使用以下命令清除索引筛选条件：

```
db.runCommand(
{
planCacheClearFilters: <collection>>
query: <query pattern>, // optional
sort: <sort specification>, // optional
comment: <any>. //optional
}
)
```

要清除集合“foo”上的所有筛选条件，请使用以下命令：

```
db.runCommand({planCacheClearFilters: "foo"})
```

要清除具有任意排序的特定查询形状，可以从 `planCacheListFilters` 的输出中复制并粘贴查询形状：

```
db.runCommand({planCacheClearFilters: "foo", query: {a: @}})
```

要清除排序依据为指定字段的特定查询形状，可以从 `planCacheListFilters` 的输出中复制并粘贴查询形状：

```
db.runCommand({planCacheClearFilters: "foo", query: {a: @},sort: {a: 1}})
```

计划程序版本 1.0、2.0 和 MongoDB 之间的潜在行为差异

在某些边缘情况下，计划程序版本 2.0 生成的结果可能与 MongoDB 中的结果略有不同。本节将介绍这些可能性的一些示例。

$(update) and $(projection)

- 在某些情况下，MongoDB 中 `$(update)` 和 `$(projection)` 运算符的行为可能与 Amazon DocumentDB 的计划程序版本 1.0 不同。下面是一些示例：

  ```
  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 } }
  ```

  - 计划程序版本 1.0：更新字段 2022
  - MongoDB：更新字段 2022
  - 计划程序版本 2.0：更新字段 2021

- ```
  db.col.insert({x:[1,2,3]})
  ```
  ```

```
db.col.update({$and:[{x:1},{x:3}]},{$set:{"x.$":500}})
```

- 计划程序版本 1.0：随机更新首个匹配元素
- MongoDB：随机更新首个匹配元素
- 计划程序版本 2.0：不进行更新

- ```
  db.col.insert({x:[1,2,3]})
  db.col.find()
  ```
  - 计划程序版本 1.0：随机选择匹配的元素
  - MongoDB：随机选择匹配的元素
  - 计划程序版本 2.0：不做选择
- ```
  db.col.insert({x:100})
  db.col.update({x:100},{x:100})
  ```
  - 计划程序版本 1.0：nModified 计数更改
  - MongoDB：nModified 计数更改
  - 计划程序版本 2.0：nModified 计数在更新为相同值时不会更改。
- 将 `$(update)` 运算符与 `$setOnInsert` 结合使用时，计划程序版本 1.0 和 MongoDB 会抛出错误，但计划程序版本 2.0 不会抛出错误。
- 在计划程序版本 2.0 中将不存在的字段重命名为 `$field` 会抛出错误，而在计划程序版本 1.0 和 MongoDB 中不会生成更新。

Index behavior

- 当使用不合适的索引应用 `$hint` 时，计划程序版本 2.0 会抛出错误，而计划程序版本 1.0 和 MongoDB 不会抛出错误。

```
// 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"
}
```

- $near 无法在计划程序版本 2.0 中使用 $hint({“$natural”:1})。

```
// 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" }
```

- 虽然 MongoDB 支持完整的正则表达式索引扫描，但计划程序版本 2.0 仅支持对前缀字段进行正则表达式索引扫描。

```
// 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)
```

- 在使用计划缓存筛选条件时，计划程序版本 2.0 与 MongoDB 存在一些固有差异。虽然计划程序版本 2.0 不支持使用计划缓存筛选条件指定“投影”和“排序规则”，但 MongoDB 支持。但 MongoDB 索引筛选条件仅在内存中，重新启动后会丢失。计划程序版本 2.0 通过重新启动和补丁可持久保留索引筛选条件。

Others

- 使用计划程序版本 2.0 时，DML 审计日志的格式与计划程序版本 1.0 的格式略有不同。

```
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}}
```

- 作为解释计划一部分的索引条件：

```
rs0:PRIMARY> db.col.createIndex({index1:1})
{
 "createdCollectionAutomatically" : false,
 "numIndexesBefore" : 1,
 "numIndexesAfter" : 2,
 "ok" : 1,
 "operationTime" : Timestamp(1761149251, 1)
}
```

计划程序版本 2.0 解释计划输出，显示了索引条件和筛选条件：

```
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)
}
```

计划程序版本 1.0 解释计划输出：

```
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)
}
```

计划程序版本 2.0 弥合了与 MongoDB 的行为差距

计划程序版本 2.0 在以下方面弥补了与 MongoDB 的行为差距：

- 计划程序版本 2.0 允许在 $elemMatch 的扁平化数组上进行数字索引查找：

```
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}}})
```

- 虽然计划程序版本 1.0 在投影中排除了字符串，但计划程序版本 2.0 的行为与 MongoDB 一致，将其视为文本值

```
// 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 }
```

- 计划程序版本 2.0 与 MongoDB 一样，不允许在相同的字段“x”和“x.a”上进行投影：

```
// 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 ] }
```

- 计划程序版本 2.0 与 MongoDB 一样，允许对子文档进行投影：

```
// 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"
}
```

- 计划程序版本 2.0 与 MongoDB 一样，投影不支持 $ 运算符之后的字段：

```
// 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 } ] }
```

- 计划程序版本 2.0 与 MongoDB 一样，允许使用 $hint：

```
// 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 }
```

查询计划器 v3

亚马逊 DocumentDB 8.0 中的 Planner 版本 3 支持 21 个聚合阶段，包括 6 个新阶段。Planner V3 内置了对不同命令的支持。与亚马逊 DocumentDB 5.0 中的 Planner v2 相比，它的总体性能提高了多达 2 倍。亚马逊 DocumentDB 8.0 中的所有新功能和运算符都与 Planner v3 兼容。Planner v3 支持的亚马逊 DocumentDB 8.0 中新的聚合阶段包括 $replaceWith、$VectorSearch、$merge、$set、$unset、$unset、$bucket。Planner v3 还支持 Amazon DocumentDB 8.0 中的新功能和运算符，包括排序规则、视图、$merge、$pow、$rand、$dateTrunc、$date、$date、$date、$date。ToParts FromParts

主题

- 先决条件
- 选择计划器版本 3.0 作为默认查询计划器
- 最佳实践

- 限制
- 对聚合和不同运算符的改进
- 计划器版本 1.0、3.0 和 MongoDB 之间的潜在行为差异

先决条件

以下先决条件适用于计划器版本 3.0：

- Planner 版本 3.0 适用于所有提供引擎版本 8.0 的区域。
- 选择引擎版本 8.0 时，Planner 版本 3.0 是默认的查询计划器。

选择计划器版本 3.0 作为默认查询计划器

如果您在 Amazon DocumentDB 8.0 中更改了默认查询计划器，并且需要恢复到计划器 v3，则可以通过控制台或 CLI 进行操作：

- 按照 修改 Amazon DocumentDB 集群参数 中的步骤修改集群的参数组。
- 对于名为 “PlannerVersion” 的参数，将值更改为 3.0，表示计划器版本 3.0。
- 选择立即应用（若选择重启时应用，则在集群下次重启后才会生效）。

最佳实践

要获得预期结果，请在应用 Planner 版本 3.0 时使用以下最佳实践：

- 在全局群集中，在两个区域的群集参数组中选择相同的`plannerVersion`值（1.0、2.0 或 3.0）。请注意，在主要区域和辅助区域中选择不同的计划程序版本可能会导致查询行为和性能不一致。
- 在计划维护时段或流量减少期间更新到计划器版本 3.0 的干扰最小，因为如果在工作负载处于活动运行状态时更改计划器版本，则错误率可能会增加。

限制

以下限制适用于计划器版本 3.0：

- 弹性集群不支持 Planner 版本 3.0，它将退回到计划器版本 1.0。

- 虽然 Planner v1 允许使用 “PlanHint” 来确保查询优化器选择特定的查询计划，但 Planner v3 不允许使用 “PlanHint”，而是依靠内部优化来为给定查询选择最佳计划。

对##和##运算符的改进

Planner 版本 3.0 引入了对 $聚合阶段和 $distinct 命令的改进。以下是一些最值得注意的改进。

- 如果可能，计划器会将$match阶段移到流程的更早阶段，从而减少后续阶段处理的文档数量。

```
//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 } }
])
```

- 当$lookup和$unwing阶段在同一字段上操作时，规划器会自动将它们组合在一起，从而减少中间数据处理并提高性能。

```
                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
```

- Planner 版本 3.0 引入了一种新的 Distinct Scan 执行策略，该策略可显著提高低基数索引上不同操作的性能。

```
                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"
                        }
                }
        }
}
```

计划器版本 1.0、3.0 和 MongoDB 之间的潜在行为差异

在某些边缘情况下，计划器版本 3.0 可能产生的结果与计划器版本 1.0 略有不同。本节将介绍这些可能性的一些示例。

Feature Differences

- 在空集合中，或者当之前的阶段（例如匹配筛选所有文档）时，Planner v1 会给出输出 “字段”: 0。Planner v3 和 MongoDB 不会给出任何输出。
- 在 Plannerv1 中，如果上一阶段返回的文档少于 n 个，{“$skip”: n} 不会跳过。无论返回的文档数量如何，Plannerv3 和 MongoDB 都能正确跳过。
- 当 $lookup 中引用的外来集合不存在时，plannerv1 会引发错误。Planner v3 和 MongoDB 将外来集合视为空集合并执行 $lookup。

```
db.coll.aggregate([
```

```
    {$lookup: {from: "does_not_exist", localField: "a", foreignField: "a", as:
 "c"}}
])
```

- 只有 Planner v1 允许在管道中使用多个 $search Planner v2/v3 会抛出错误，而 MongoDB 不支持它。

```
VectorSearch = { "$search": { "vectorSearch": {
   "vector": [0.2, 0.5, 0.8],
   "path": "vectorEmbedding",
   "similarity": "cosine",
   "k": 2,
   "efSearch": 1
   }}}
db.coll.aggregate([VectorSearch, VectorSearch])
```

- 只有当 VectorSearch 阶段不是第一个阶段时，Planner v3 才起作用。在这种情况下，MongoDB 会抛出错误，而 Planner v1 不支持 $VectorSearch 阶段。

```
VectorSearch = { {"$vectorSearch": {
    "queryVector": [0.2, 0.5, 0.8],
    "path": "vectorEmbedding",
    "similarity": "euclidean",
    "limit": 4,
    "numCandidates": 100} }

db.coll.aggregate([$match:{}, VectorSearch])
```

通过 Amazon DocumentDB 查询地理空间数据

本节将介绍可以怎样使用 Amazon DocumentDB 来查询地理空间数据。阅读本节后，您将能够回答如何在 Amazon DocumentDB 中存储、查询和索引化地理空间数据。

主题

- 概述
- 索引化和存储地理空间数据
- 查询地理空间数据
- 限制

概述

地理空间的常见用例涉及来自数据的邻近分析。例如，“查找距离西雅图 50 英里范围内的所有机场”，或“查找距离给定位置最近的餐厅”。Amazon DocumentDB 使用 GeoJSON 规范来表示地理空间数据。GeoJSON 是坐标空间中形状 JSON-formatting 的开源规范。GeoJSON 坐标捕获经度和纬度，表示在类地球球体上的位置。

索引化和存储地理空间数据

Amazon DocumentDB 使用“点” GeoJSON 类型来存储地理空间数据。每个 GeoJSON 文档（或子文档）通常由两个字段组成：

- 类型：要被表示的形状，它告知 Amazon DocumentDB 如何解释“坐标”字段。目前，Amazon DocumentDB 仅支持点
- 坐标：表示为数组中对象的经纬度对：[经度、纬度]

Amazon DocumentDB 还使用 2dsphere 索引来索引化地理空间数据。Amazon DocumentDB 支持索引化点。Amazon DocumentDB 支持用 2dsphere 索引化进行邻近查询。

让我们考虑一个场景，即您正在为送餐服务构建一个应用程序。您希望要在 Amazon DocumentDB 中存储各种餐厅的经纬度对。为此，我们首先建议您对地理空间字段创建一个包含经纬度对的索引。

```
use restaurantsdb
db.usarestaurants.createIndex({location:"2dsphere"})
```

该命令的输出内容类似如下所示：

```
{
 "createdCollectionAutomatically" : true,
 "numIndexesBefore" : 1,
 "numIndexesAfter" : 2,
 "ok" : 1
}
```

创建索引后，您可以开始将数据插入 Amazon DocumentDB 集合。

```
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
      ]
   }
});
```

查询地理空间数据

Amazon DocumentDB 支持对地理空间数据的邻近查询、包含查询和交叉点查询。一个良好的邻近查询示例是查找距另一个点（城市）小于某个距离且大于某个距离的所有点（所有机场）。包含查询的一

个很好的例子是查找位于指定（纽约州）的所有点 area/polygon （所有机场）。一个良好的交叉点查询示例是查找与某个点（城市）相交的多边形（州）。您可以使用以下地理空间运算符从数据中获取见解。

- **$nearSphere**：$nearSphere 是一个查找运算符，支持查找距 GeoJSON 点最近处到最远处的点。
- **$geoNear**：$geoNear 是一个聚合运算符，支持计算以米为单位的距 GeoJSON 点的距离。
- **$minDistance**：$minDistance 是一个查找运算符，结合 $nearSphere 或 $geoNear 一起用于筛选距中心点至少指定最小距离的文档。
- **$maxDistance**：$maxDistance 是一个查找运算符，结合 $nearSphere 或 $geoNear 一起用于筛选距中心点至多指定最大距离的文档。
- **$geoWithin**：$geoWithin 是一个查找运算符，支持查找包含具有以下地理空间数据的文档，这些数据完全存在于指定形状（如多边形）范围内。
- **$geoIntersects**：$geoIntersects 是一个查找运算符，支持查找其地理空间数据与指定 GeoJSON 对象相交的文档。

> **Note**
>
> $geoNear 和 $nearSphere 要求您在邻近查询中使用的 GeoJSON 字段上有 2dsphere 索引。

示例 1

在此示例中，您将学习如何查找按照距某地址（点）最近距离排序的所有餐厅（点）。

要执行这样的查询，您可以使用 $geoNear 计算一组点距另一点的距离。您也可以添加 distanceMultiplier 来测量以千米计的距离。

```
db.usarestaurants.aggregate([
   {
      "$geoNear":{
         "near":{
            "type":"Point",
            "coordinates":[
               -122.3516,
               47.6156
            ]
```

```
         },
         "spherical":true,
         "distanceField":"DistanceKilometers",
         "distanceMultiplier":0.001
      }
   }
])
```

上述命令将返回按照距指定点的距离（最近到最远）排序的餐厅。该命令的输出内容类似如下所示

```
{ "_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**
>
> 在 Amazon DocumentDB 8.0（引擎补丁版本 4.0.5748）`limit`及更高版本中，该`$geoNear`阶段的和`num`选项以及 100 个文档的默认限制已被弃用。以下示例仅使用`limit`并`num`适用于亚马逊 DocumentDB 8.0（引擎补丁版本 4.0.4448）和更早版本。

要限制查询中的结果数量，可使用 `limit` 或 `num` 选项。

`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
      }
   }
])
```

示例 2

在此示例中，您将学习如何查找距特定地址（点）2 千米范围内的所有餐厅（点）。要执行此类查询，您可以在距 GeoJSON 点的最小值 $minDistance 和最大值 $maxDistance 范围内使用 $nearSphere

```
db.usarestaurants.find({
   "location":{
      "$nearSphere":{
         "$geometry":{
            "type":"Point",
            "coordinates":[
               -122.3516,
               47.6156
            ]
```

```
         },
         "$minDistance":1,
         "$maxDistance":2000
      }
   }
},
{
   "name":1
})
```

上面的命令将返回距指定点最大距离 2 千米的餐厅。该命令的输出内容类似如下所示

```
{ "_id" : ObjectId("611f3da985009a81ad38e74b"), "name" : "Noodle House" }
```

限制

Amazon DocumentDB 不支持对多边形、、、 LineString MultiPoint MultiPolygon、 MultiLineString和进行查询或索引。 GeometryCollection

部分索引

部分索引对满足指定筛选条件的集合中的文档编制索引。Amazon DocumentDB 5.0 基于实例的集群支持部分索引功能。

主题

- 创建部分索引
- 支持的运算符
- 使用部分索引进行查询
- 部分索引功能
- 部分索引限制

创建部分索引

要创建部分索引，请使用带 `partialFilterExpression` 选项的 `createIndex()` 方法。例如，以下操作在订单集合中创建唯一的复合索引，该索引为具有 `OrderID` 且字段 `isDelivered` 为“true”的文档编制索引：

```
db.orders.createIndex(
```

```
  {"category": 1, "CustomerId": 1, "OrderId": 1},
  {"unique": true, "partialFilterExpression":
    {"$and": [
      {"OrderId": {"$exists": true}},
      {"isDelivered": {"$eq": false}}
    ]}
  }
)
```

支持的运算符

- $eq
- $exists
- $and（仅限顶层）
- $gt/$gte/$lt/$lte（只有查询中断言的筛选器与部分筛选表达式完全匹配时才使用索引扫描）（参见“限制”）

使用部分索引进行查询

使用部分索引可以实现以下查询模式：

- 查询谓词与部分索引筛选表达式完全匹配：

```
db.orders.find({"$and": [
    {"OrderId": {"$exists": true}},
    {"isDelivered": {"$eq": false}}
  ]}).explain()
```

- 查询筛选器的预期结果是部分筛选器的逻辑子集：

```
db.orders.find({"$and": [
    {"OrderId": {"$exists": true}},
    {"isDelivered": {"$eq": false}},
    {"OrderAmount": {"$eq": "5"}}
  ]}).explain()
```

- 查询子谓词可以与其他索引结合使用：

```
db.orders.createIndex({"anotherIndex":1})
```

```
db.orders.find({ "$or": [
      {"$and": [
        {"OrderId": {"$exists": true}},
        {"isDelivered": {"$eq": false}}
      ]},
      {"anotherIndex": {"$eq": 5}}
    ]
  }).explain()
```

> **Note**
>
> 如果有效的话，查询计划程序可以选择使用集合扫描而不是索引扫描。这通常出现在非常小的集合或会返回集合大部分内容的查询中。

部分索引功能

列出部分索引

FilterExpression 使用`getIndex`操作列出带有部分索引的部分索引。例如，发出的 `getIndex` 操作列出了带有键、名称和 partialFilterExpressions 字段的部分索引：

```
db.orders.getIndexes()
```

此示例返回以下输出：

```
[
  {
    "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}}
      ]
    }
  }
]
```

相同 key:order 上的多个部分筛选表达式

可以为相同的字段组合 (key:order) 创建不同的部分索引。这些索引必须使用不同的名称。

```
db.orders.createIndex(
  {"OrderId":1},
  {
    name:"firstPartialIndex",
    partialFilterExpression:{"OrderId":{"$exists": true}}
  }
)
```

```
db.orders.createIndex(
  {"OrderId":1},
  {
    name:"secondPartialIndex",
    partialFilterExpression:{"OrderId":{"$gt": 1000}}
  }
)
```

运行 `getIndexes` 操作列出集合中的所有索引：

```
db.orders.getIndexes()
```

这些示例返回以下输出：

```
[
```

```
  {
    "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**
>
> 索引名称必须不同，并且只能按名称删除。

具有部分和 TTL 属性的索引

您还可以通过在索引创建过程中同时指定 `partialFilterExpression` 和 `expireAfterSeconds` 选项来创建具有部分和 TTL 属性的索引。这使就可以更好地控制当前要从集合中删除的文档。

例如，您可能有一个可识别要在特定时间段后删除的文档的 TTL 索引。您现在可以使用部分索引选项，为何时删除文档提供额外的条件：

```
db.orders.createIndex(
```

```
    { "OrderTimestamp": 1 },
    {
        expireAfterSeconds: 3600 ,
        partialFilterExpression: { "isDelivered": { $eq: true } }
    }
)
```

此示例返回以下输出：

```
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1,
        "operationTime" : Timestamp(1234567890, 1)
}
```

运行 getIndexes 操作以列出集合中存在的索引：

```
db.orders.getIndexes()
[
    {
        "v" : 4,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.orders"
    }
```

此示例返回以下输出：

```
[
    {
        "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
    }
]
```

部分索引限制

以下限制适用于部分索引功能：

- Amazon DocumentDB 中的不等于查询只有在查询筛选器谓词与 `partialFilterExpression` 完全匹配且具有相同的数据类型时才会使用部分索引。

  > **Note**
  >
  > 对于上述情况，甚至是 `$hint` 也不能用于强制执行 IXSCAN。

  在以下示例中，`partialFilterExpression` 仅适用于 `field1`，但不适用 `field2`：

  ```
  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
  ```

- 不支持带数组运算符的 `partialFilterExpression`。以下操作会产生错误：

  ```
  db.orders.createIndex(
  ```

```
  {"CustomerId":1},
  {'partialFilterExpression': {'OrderId': {'$eq': [1000, 1001, 1002]}}}
)
```

- 部分FilterExpression 字段中不支持以下运算符：
  - $all（数组运算符）
  - $mod（数组运算符）
  - $or
  - $xor
  - $not
  - $nor
- 筛选表达式和筛选器应具有相同的数据类型。

执行 Amazon DocumentDB 文本搜索

Amazon DocumentDB 的原生全文搜索功能（文本索引 v1）允许您使用特殊用途的文本索引对大型文本数据集执行文本搜索。本节介绍文本索引功能的功能，并提供了有关如何在 Amazon DocumentDB 中创建和使用文本索引的步骤。还列出了文本搜索的限制。

主题

- 支持的功能
- 使用 Amazon DocumentDB 文本索引
- 与 MongoDB 的差异
- 最佳实践和准则
- 文本索引 V2
- 限制

支持的功能

Amazon DocumentDB 文本搜索支持以下 MongoDB API 兼容功能：

- 在单个字段上创建文本索引。
- 创建包含多个文本字段的复合文本索引。
- 执行单字或多字搜索。

- 使用权重控制搜索结果。
- 通过打分对搜索结果进行排序。
- 在聚合管道中使用文本索引。
- 搜索确切的短语。

使用 Amazon DocumentDB 文本索引

要在包含字符串数据的字段上创建文本索引，需指定以下所示的字符串“text”：

单个字段索引：

```
db.test.createIndex({"comments": "text"})
```

此索引支持在指定集合的“comments”字符串字段中进行文本搜索查询。

在多个字符串字段上创建复合文本索引：

```
db.test.createIndex({"comments": "text", "title":"text"})
```

此索引支持在指定集合中的“comments”和“title”字符串字段中进行文本搜索查询。在创建复合文本索引时，最多可指定 30 个字段。一旦创建后，文本搜索查询将对所有索引字段进行查询。

> **Note**
>
> 每个集合只能有一个文本索引。

列出 Amazon DocumentDB 集合上的文本索引

您可以在集合`getIndexes()`上使用来识别和描述索引，包括文本索引，如以下示例所示：

```
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
   }
]
```

创建索引后，开始将数据插入 Amazon DocumentDB 集合。

```
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"}])
```

运行文本搜索查询

运行单字文本搜索查询

需要使用 `$text` 和 `$search` 运算符来执行文本搜索。以下示例返回文本索引字段中包含字符串“apple”或“apples”等其他格式字符串的所有文档：

```
db.test.find({$text: {$search: "apple"}})
```

输出：

该命令的输出内容类似如下所示：

```
{ "_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" }
```

运行多字文本搜索

您还可以对 Amazon DocumentDB 数据执行多字文本搜索。以下命令返回文本索引字段中含有“apple”或“pie”的文档：

```
db.test.find({$text: {$search: "apple pie"}})
```

输出：

该命令的输出内容类似如下所示：

```
{ "_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" }
```

运行多字短语文本搜索

对于多字短语搜索，请使用以下示例：

```
db.test.find({$text: {$search: "\"apple pie\""}})
```

输出：

上述命令返回文本索引字段中含有确切短语“apple pie”的文档：该命令的输出内容类似如下所示：

```
{ "_id" : 4, "star_rating" : 2, "comments" : "bake the apple pie in the oven" }
```

使用筛选器进行文本搜索

您还可以将文本搜索与其他查询运算符结合起来使用，按照附加条件筛选结果：

```
db.test.find({$and: [{star_rating: 5}, {$text: {$search: "interest"}}]})
```

输出：

上述命令返回文本索引字段中包含任何形式的“interest”且“star_rating”等于 5 的文档。该命令的输出内容类似如下所示：

```
{ "_id" : 5, "star_rating" : 5, "comments" : "interesting couch" }
{ "_id" : 6, "star_rating" : 5, "comments" : "interested in couch for sale, year
 2022" }
```

限制文本搜索中返回的文档数量

可以选择使用 `limit` 限制返回的文档数量：

```
db.test.find({$and: [{star_rating: 5}, {$text: {$search: "couch"}}]}).limit(1)
```

输出：

上述命令返回一个满足筛选器的结果：

```
{ "_id" : 5, "star_rating" : 5, "comments" : "interesting couch" }
```

通过文本打分对结果进行排序

以下示例通过文本打分对文本搜索结果进行排序：

```
db.test.find({$text: {$search: "apple"}}, {score: {$meta: "textScore"}}).sort({score:
 {$meta: "textScore"}})
```

输出：

上述命令返回文本索引字段中包含“apple”或“apples”等其他格式的文档，并根据文档与搜索词的相关程度对结果进行排序。该命令的输出内容类似如下所示：

```
{ "_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 }
```

`aggregate`、`count`、`findAndModify`、`update` 和 `delete` 命令也支持 `$text` 和 `$search`。

聚合运算符

使用 **$match** 的聚合管道

```
db.test.aggregate(
   [{ $match: { $text: { $search: "apple pie" } } }]
)
```

输出：

上述命令返回以下结果：

```
{ "_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" }
```

其他聚合运算符的组合

```
db.test.aggregate(
   [
      { $match: { $text: { $search: "apple pie" } } },
      { $sort: { score: { $meta: "textScore" } } },
      { $project: { score: { $meta: "textScore" } } }
   ]
)
```

输出：

上述命令返回以下结果：

```
{ "_id" : 4, "score" : 0.6079270860936958 }
{ "_id" : 1, "score" : 0.3039635430468479 }
{ "_id" : 2, "score" : 0.3039635430468479 }
{ "_id" : 3, "score" : 0.3039635430468479 }
```

在创建文本索引时指定多个字段

最多可以为复合文本索引中的三个字段分配权重。分配给文本索引中的字段的默认权重为一 (1)。权重为可选参数，须介于 1 到 100000 之间。

```
db.test.createIndex(
   {
     "firstname": "text",
     "lastname": "text",
     ...
   },
   {
     weights: {
       "firstname": 5,
       "lastname":10,
       ...
     },
     name: "name_text_index"
   }
 )
```

与 MongoDB 的差异

Amazon DocumentDB 的文本索引功能使用反向索引和词频算法。文本索引默认为稀疏索引。由于解析逻辑、令牌化分隔符等方面的差异，对于相同的数据集或查询形状，可能无法返回与 MongoDB 相同的结果集。

Amazon DocumentDB 文本索引和 MongoDB 之间还存在以下差异：

- 不支持采用非文本索引的复合索引。
- Amazon DocumentDB 文本索引不区分大小写。
- 文本索引仅支持英语。
- 不支持数组（或多键）字段的文本索引。例如，使用文档对“a”创建文本索引 {“a”:[“apple”, “pie”]} 将会失败。
- 不支持通配符文本索引。
- 不支持唯一文本索引。
- 不支持排除某个词。

最佳实践和准则

- 为了优化通过文本打分排序进行文本搜索查询的性能，我们建议在加载数据之前创建文本索引。
- 文本索引需要额外的存储空间来实现索引数据内部副本的最优化。这将产生额外的费用。

文本索引 V2

Amazon DocumentDB 8.0 引入了新版本的文本索引 (V2)，该版本更改了底层文本搜索解析器，以提高与 MongoDB 的兼容性。

除了 V1 文本索引提供的功能外，V2 文本索引还提供以下支持：

- 如果可能，计划器会将$match阶段移到流程的更早阶段，从而减少后续阶段处理的文档数量。
- 改进了文本字段中特殊字符（例如电子邮件、URL 和文件路径）的标记化。V2 可以解析和匹配这些字符串中的单个标记，而 V1 不能。

```
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" }
```

限制

Amazon DocumentDB 中的文本搜索存在以下限制：

- 文本索引存储词素及其位置信息。单个文档中所有词素及其位置信息的组合大小限制为 1MB。

索引

主题

- 索引管理
- 索引类型
- 索引属性

索引管理

本节涵盖亚马逊文档数据库的索引管理主题。

主题

- 索引创建
- 维护索引
- 使用索引的最佳实践

索引创建

在 Amazon DocumentDB 中创建索引需要做出一些决定：

- 需要多久完成？
- 在构建过程中无法访问该集合吗？
- 实例的计算能力中有多少可以分配给构建？
- 应该创建什么类型的索引？

本节帮助您回答这些问题，并提供在基于实例的集群集合上创建 Amazon DocumentDB 索引的命令和监控示例。

指南要求

以下指南包括创建新索引时的基本限制和配置权衡：

- Amazon DocumentDB 版本支持：虽然所有版本的 Amazon DocumentDB 都支持单工作进程索引，但只有 Amazon DocumentDB 版本 4.0 和 5.0 支持多工作进程索引。

- 性能权衡：在索引创建过程中增加工作进程数量会增加 Amazon DocumentDB 数据库主实例上的 CPU 利用率和读取 IO。创建新索引所需的资源将无法用于正在运行的工作负载。
- 弹性集群：Amazon DocumentDB 弹性集群不支持并行索引。
- 最大工作进程：您可以配置的最大工作进程取决于数据库集群中主实例的大小。其等于数据库集群中主实例上 vCPU 总数的一半。例如，在有 64 个 vCPU 的 db.r6g.16xlarge 实例上，你最多可以运行 32 个工作进程。

> **Note**
>
> 2xlarge 及以上的实例支持并行索引构建。

- 最少工作进程：您可以配置的最小工作进程为 1。在基于实例的集群上创建索引的默认设置是两个工作进程。但是，您可以使用 “工作线程” 选项将工作线程数减少到一个。这将使用单工作进程运行该进程。
- 索引压缩：Amazon DocumentDB 不支持索引压缩。索引的数据大小可能比您使用其他选项时更大。
- 索引多个集合：数据库集群主实例上一半的 vCPU 可用于已配置的工作进程对多个集合执行索引创建。
- 索引类型-有关 Amazon DocumentDB 上支持的索引类型的列表，请参阅索引类型。

索引构建类型

索引构建的四种类型是：

- Foreground：在索引创建完成之前，前台索引构建会阻止对创建索引的集合进行的所有其他数据库操作。Amazon DocumentDB 前台构建由五个阶段组成。
- 前台（唯一）：单个文档（唯一）前台索引构建会阻止其他数据库操作，例如常规的前台构建。与基本前台构建不同，唯一构建使用额外的阶段（排序键 2）来查找重复的键。前台（唯一）构建由六个阶段组成。
- 背景：后台索引构建允许在创建索引时在前台运行其他数据库操作。Amazon DocumentDB 后台构建由八个阶段组成。
- 背景（唯一）：单个文档（唯一）后台索引构建允许在创建索引时在前台运行其他数据库操作。与基本背景构建不同，唯一构建使用额外的阶段（排序键 2）来查找重复的键。后台（唯一）构建由九个阶段组成。

默认索引构建类型为前台。

要在后台构建索引，请在创建索引时添加 “background”: true 参数。

开始使用

要开始在集合上创建索引，请使用 createIndexes 命令。默认情况下，该命令将运行两个并行工作进程，可将索引创建过程的速度提高两倍。

例如，以下命令过程演示了如何为文档中的 “user_name” 字段创建索引，并将索引处理速度提高到四个工作程序：

1. 在集群上使用两个 parallel worker 在后台创建索引：

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx", "background":true }]})
```

2. 要优化索引创建过程的速度，您可以使用 <number>db.runCommand createIndexes 命令中的 “工作线程” 选项 (“工作线程”:) 来指定工作线程数。

   将处理速度提至四个并行工作进程：

```
db.runCommand({"createIndexes":"test","indexes":[{"key": {"user_name":1},
 "name":"username_idx", "workers":4}]})
```

> **Note**
>
> 工作进程数越多，则索引创建速度越快。但是，工作进程数增幅越大，主实例的 vCPU 的加载及 IO 的读取增速越快。确保您的集群配置充足，足以应对增加的负担，不会使其他工作负载降级。

索引进度状态

索引创建过程的工作原理是初始化、扫描集合、排序键，最后通过索引生成器插入键。在前台运行时，该过程最多有六个阶段，在后台运行时，则最多有九个阶段。您可以逐阶段查看状态指标，例如完成百分比、扫描的存储块总数、已排序键和已插入键。

在 mongo shell 中使用 db.currenTop () 命令监控索引过程的进度。最后一个阶段的 100% 已完成表示所有索引都已成功创建：

```
db.currentOp({"command.createIndexes": { $exists : true } })
```

> **Note**
>
> 仅支持在 Amazon DocumentDB 5.0 上查看索引进度状态。

索引构建阶段

暂存区	前台	前台（唯一）	背景	背景（唯一）
INITIALIZING	1	1	1	1
构建索引：初始化	2	2	2	2
构建索引：扫描集合	3	3	3	3
构建索引：排序键 1	4	4	4	4
构建索引：排序键 2		5		5
构建索引：插入键	5	6	5	6

暂存区	前台	前台（唯一）	背景	背景（唯一）
验证：扫描索引			6	7
验证：排序元组			7	8
验证：扫描集合			8	9

- 正在初始化：createIndex 正在准备索引生成器。该阶段应该非常短暂。
- 构建索引：正在初始化：索引生成器正在准备创建索引。该阶段应该非常短暂。
- 构建索引：扫描集合：索引生成器正在执行集合扫描以收集索引秘钥。计量单位是 “块”。

> **Note**
>
> 如果为索引构建配置了多个工作进程，则在此阶段将显示该工作进程。“扫描集合” 阶段是在索引构建过程中使用多个工作线程的唯一阶段。所有其他阶段将显示单工作进程。

- 构建索引：排序键 1：索引生成器正在排序已收集的索引键。计量单位是 “钥匙”。
- 构建索引：排序键 2：索引生成器正在对收集的与死元组相对应的索引键进行排序。此阶段仅适用于唯一索引构建。计量单位是 “钥匙”。
- 构建索引：插入键：索引生成器正在将索引键插入到新索引中。计量单位是 “钥匙”。
- 验证：扫描索引：createIndex 正在扫描索引以查找需要验证的键。计量单位是 “块”。
- 验证：排序元组：createIndex 正在排序索引扫描阶段的输出。
- 验证：扫描集合：CreateIndex 正在扫描集合以验证在前两个阶段中找到的索引键。计量单位是 “块”。

索引构建输出示例

在以下输出示例（前台索引构建）中，显示了索引创建的状态。“msg” 字段通过指明构建的阶段和完成百分比来汇总构建进度。“workers” 字段表示在索引构建的该阶段使用的工作器数量。“进度” 字段显示用于计算完成百分比的实际数字。

> **Note**
>
> 亚马逊 DocumentDB 版本 4.0 不支持 IndexBuildName “当前”、“消息” 和 “进度” 字段。

```
{
    "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
}
```

资源问题排查

有关解决索引创建问题，请参阅以下资源：

- 索引构建失败
- 后台索引构建延迟问题和失败

维护索引

索引膨胀

Amazon DocumentDB 使用 Multi-Version 并发控制 (MVCC) 来管理并发交易。删除或更新文档后，其先前版本将作为“失效”版本保留在集合和索引中。垃圾回收过程会自动从这些失效版本中回收空间以用于未来的操作。

当集合的索引因失效或过时的索引条目累积或页面内的碎片累积而变大时，就会出现索引膨胀。报告的百分比表示未来索引条目可以使用的索引空间量。此膨胀会同时消耗缓冲区缓存和存储空间中的空间。如果要消除膨胀，则需要重建索引。

示例：运行以下命令来确定索引的未使用存储空间：

```
db.coll.aggregate({$indexStats:{}});
```

结果类似如下：

```
{
    "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
    }
}
```

使用 reIndex 命令无需停机即可重建索引，该命令需要扫描整个集合。参见使用 reIndex 维护索引。

使用 reIndex 维护索引

reIndex 是一个用于重建索引的命令。通常在索引损坏或效率低下时使用。随着时间的推移，索引会因为多次更新、插入或删除而积累未使用的空间，从而导致性能下降。重新编制索引有助于删除此类未使用的空间并恢复索引的效率。

重新索引指南

- 只有亚马逊 DocumentDB 5.0 及更高版本支持重新索引
- reIndex 总是在后台运行。
- Amazon DocumentDB 支持在后台重新索引单个索引，允许多个工作人员。reIndex 进程运行时，查询可以使用旧索引。
- 亚马逊 DocumentDB 支持通过 currentOP 为进度报告编制索引。您可以看到类似于索引创建期间所查看的索引构建阶段的索引构建阶段。唯一的区别是，无论reIndex是否独特，它总是有八个阶段。没有 “建筑索引：按键排序 2” 阶段。
- reIndex 可以与任何命令同时运行，但同一集合上的索引相关命令除外：createIndexes、dropIndexes、collMod 和 renameCollection。
- 文本、地理空间、矢量索引和部分索引目前不支持 reIndex。

重新索引构建

使用以下命令来重建索引：

```
db.runCommand({ reIndex: "collection-name", index: "index-name"})
```

您也可以选择控制分配给重建过程的工作人员的人数：

```
db.runCommand({ reIndex: "collection-name", index: "index-name", workers: number })
```

有关使用 Java 管理索引的特定信息，请参阅使用 Java 在 A mazon DocumentDB 中管理索引。

使用索引的最佳实践

建立索引

将数据导入 Amazon DocumentDB 时，最好在导入大型数据集之前先创建索引。您可以使用亚马逊 DocumentDB 索引工具从正在运行的 MongoDB 实例或 mongodump 目录中提取索引，然后在 Amazon DocumentDB 集群中创建这些索引。有关迁移的更多指导，请参阅迁移到 Amazon DocumentDB。

索引选择性

我们建议您将索引创建限制为其中的重复值数量低于集合中总文档数的 1% 的字段。例如，如果您的集合包含 100,000 个文档，则仅在相同值出现次数不超过 1,000 次的字段上创建索引。

选择具有大量唯一值（即高基数）的索引可确保筛选操作返回少量文档，从而在索引扫描期间获得良好的性能。高基数索引的一个例子是一个唯一索引，它保证相等谓词最多返回单个文档。低基数的示例包括布尔字段上的索引和一周中某天的索引。由于低基数索引的性能较差，数据库的查询优化器不太可能选择低基数索引。同时，低基数索引继续消耗资源，例如磁盘空间和。 I/Os作为经验法则，应将索引定位于典型值频率为总集合大小的 1% 或更小的字段。

此外，建议仅在通常用作筛选条件的字段上创建索引，并定期查找未使用的索引。有关更多信息，请参阅 the section called “分析索引使用情况并识别未使用的索引”。

索引对数据写入的影响

虽然索引可以通过避免扫描集合中的每个文档来提高查询性能，但这种提高是有代价的。对于集合的每个索引，每次插入、更新或删除文档时，数据库都必须更新集合并将字段写入集合的每个索引。例如，如果某个集合有九个索引，则数据库必须执行十次写入操作，才能将操作通知给客户端。因此，每增加一个索引都会导致额外的写入延迟 I/Os，并增加总利用的存储空间。

所以应该适当调整集群实例的大小，以确保将所有工作集容纳在内存中。这就避免了从存储卷中持续读取索引页的需要，因为这会对性能产生负面影响并产生更高 I/O 的成本。有关更多信息，请参阅调整实例规模。

为了获得最佳性能，请尽量减少集合中的索引数量，仅添加必要的索引来提高常用查询的性能。虽然工作负载会变化，但有一个很好的指导原则，就是将每个集合的索引数量保持在五个以内。

识别缺失的索引

我们建议您定期识别缺失的索引，这是一种很好的做法。有关更多信息，请参阅 the section called “识别缺失的索引”。

识别未使用的索引

我们建议您定期识别并删除未使用的索引，这是一种很好的做法。有关更多信息，请参阅 the section called “分析索引使用情况并识别未使用的索引”。

索引类型

主题

- 单字段索引
- 复合索引
- Multi-key 索引
- 文本索引
- 地理空间索引
- 向量索引

单字段索引

单字段索引存储来自文档集合中任何单个字段的信息。默认情况下，所有集合`_id`在该字段上都有索引。您可以添加其他索引以加快重要查询和操作的速度。

在以下情况下，单字段索引很有用：

- 您的应用程序经常根据特定字段的值查询或筛选文档。
- 您需要按特定字段高效地对文档进行排序。
- 您希望通过创建唯一索引来确保集合中文档中特定字段的唯一性。

支持的索引属性

Option	3.6	4.0	5.0	8.0	弹性集群
name	支持	是	是	是	是

Option	3.6	4.0	5.0	8.0	弹性集群
独特	支持	是	是	是	是
稀疏*	支持	是	是	是	是
部分 FilterExpression *	否	否	是	是	否
到期 AfterSeconds	支持	是	是	是	是

* sparse 和partialFilterExpression选项不能在同一个索引定义中一起使用。如果您尝试使用这些选项创建索引，它将失败并显示以下错误：

```
Error in specification: cannot mix partialFilterExpression and sparse options
```

创建单个字段索引

使用createIndex()方法创建单个字段索引。语法如下：db.collection.createIndex(<key>, <options>)。

key参数是一个 JSON 文档，用于指定字段和索引排序顺序：

```
{
  "<field>": <1 (ascending)|-1 (descending)>
}
```

options参数是一个 JSON 文档，用于指定索引的选项：

```
{
  "name": "<name>",
  "unique": <true | false>,
  "sparse": <true | false>,
  "partialFilterExpression": <filter expression>,
  "expireAfterSeconds": <seconds before expiry>
}
```

有关创建单字段索引的示例，请参阅索引属性。

复合索引

复合索引存储来自文档集合中两个或多个字段的信息，允许对第一个字段或任何前缀字段进行查询。这些索引可优化同时筛选多个字段或将筛选与排序操作相结合的查询的性能。它们对最左边的索引字段的单条件查询也很有效。数据库利用这些索引条目来高效地找到匹配的文档，而无需执行完整的馆藏扫描。

复合字段索引在以下情况下很有用：

- 您需要同时筛选多个字段。
- 您需要将筛选与排序操作相结合。

支持的索引属性

Option	3.6	4.0	5.0	8.0	弹性集群
name	支持	是	是	是	是
独特	支持	是	是	是	是
稀疏*	支持	是	是	是	是
部分 FilterExpression *	否	否	是	是	否

* `sparse` 和`partialFilterExpression`选项不能在同一个索引定义中一起使用。如果您尝试使用这些选项创建索引，它将失败并显示以下错误：

```
Error in specification: cannot mix partialFilterExpression and sparse options
```

创建复合索引

使用`createIndex()`方法创建复合索引。该方法的语法是：`db.collection.createIndex(<keys>, <options>)`

keys参数是一个 JSON 文档，用于指定字段和索引排序顺序：

```
{
  "<field 1>": <1 (ascending)|-1 (descending)>,
  "<field 2>": <1 (ascending)|-1 (descending)>,
  ...
}
```

请注意，复合索引中只有一个字段可以是数组。如果您尝试在两个或多个数组字段上创建复合索引，它将失败并显示以下错误：

```
multiple fields of compound index cannot be arrays
```

options参数是一个 JSON 文档，用于指定索引的选项：

```
{
  "name": "<name>",
  "unique": <true | false>,
  "sparse": <true | false>,
  "partialFilterExpression": <filter expression>
}
```

有关创建复合索引的示例，请参阅索引属性。

Multi-key 索引

对于具有数组值的字段，多键索引允许您为数组中的每个元素创建索引键。对数组进行索引会为数组的每个元素创建一个索引条目。

Multi-key 当您的应用程序经常根据数组中的值查询或筛选文档时，索引很有用。

支持的索引属性

Option	3.6	4.0	5.0	8.0	弹性集群
name	支持	是	是	是	是
独特	支持	是	是	是	是
稀疏*	支持	是	是	是	是

Option	3.6	4.0	5.0	8.0	弹性集群
部分 FilterExpression *	否	否	是	是	否
到期 AfterSeconds	支持	是	是	是	是

* sparse 和partialFilterExpression选项不能在同一个索引定义中一起使用。如果您尝试使用这些选项创建索引，它将失败并显示以下错误：

```
Error in specification: cannot mix partialFilterExpression and sparse options
```

创建多键索引

使用createIndex()方法创建多键索引。该方法的语法是：db.collection.createIndex(<key>, <options>)

key参数是一个 JSON 文档，用于指定字段和索引排序顺序：

```
{
  "<field>": <1 (ascending)|-1 (descending)>
}
```

options参数是一个 JSON 文档，用于指定索引的选项：

```
{
  "name": "<name>",
  "unique": <true | false>,
  "sparse": <true | false>,
  "partialFilterExpression": <filter expression>,
  "expireAfterSeconds": <seconds before expiry>
}
```

以下示例在categories字段上创建一个按升序排序的多键索引，名称为：book_categories

```
db.collection.createIndex(
  {
```

```
    "categories": 1
  },
  {
    "name": "book_categories"
  }
)
```

有关创建多键索引的示例，请参阅索引属性。

文本索引

文本索引可用于在文档的文本字段中搜索单词或短语，允许您执行全文搜索功能。您可以使用多个文本字段创建单字段文本索引或复合文本索引。但是，每个集合只能有一个文本索引。有关更多信息，请参阅使用实际用例解锁 Amazon DocumentDB 文本搜索的强大功能。

支持的索引属性

Option	3.6	4.0	5.0	8.0	弹性集群
name	否	否	是	是	否

创建文本索引

使用`createIndex()`方法创建文本索引。该方法的语法是：`db.collection.createIndex(<keys>, <options>)`

`keys`参数是一个 JSON 文档，用于指定字段和文本索引类型：

```
{
  "<field 1>": "text",
  "<field 2>": "text"
}
```

`options`参数是一个 JSON 文档，用于指定索引的名称：

```
{
  "name": "<name>"
}
```

有关创建文本索引的示例，请参阅索引属性。

地理空间索引

地理空间索引是一种特殊类型的索引，旨在有效地查询和管理存储在文档集合中的地理空间数据。Amazon DocumentDB 支持 2dsphere 索引，这些索引是专门为处理球体（如地球）上的地理空间数据而设计的。这允许基于球面几何形状进行精确的计算和查询。

当您的应用程序需要执行基于位置的查询时，地理空间索引非常有用，例如：

- 寻找附近的景点，
- 确定某个位置是否位于特定区域内
- 计算位置之间的距离

支持的索引属性

Option	3.6	4.0	5.0	8.0	弹性集群
name	支持	是	是	是	是

创建地理空间索引

使用`createIndex()`方法创建地理空间索引。该方法的语法是：`db.collection.createIndex(<key>, <options>)`

该`key`参数是一个 JSON 文档，用于指定字段和 2dsphere 索引类型：

```
{
  "<field>": "2dsphere"
}
```

`options`参数是一个 JSON 文档，用于指定索引的选项：

```
{
  "name": "<name>"
}
```

有关创建地理空间索引的示例，请参阅索引属性。

向量索引

矢量索引是一种特殊类型的索引，旨在有效地查询和管理存储在文档集合中的矢量数据。Amazon DocumentDB 支持分层可导航小世界 (HNSW) 和带平面压缩的倒置文件 (ivfFLat) 索引。

有关更多信息，请参阅 Amazon DocumentDB 向量搜索。

向量索引有利于机器学习和生成式 AI 用例，例如：

- 语义搜索
- 产品推荐
- 个性化
- 聊天机器人
- 欺诈检测
- 异常检测

支持的索引属性

Option	3.6	4.0	5.0	8.0	弹性集群
name	否	否	是	是	否

创建向量索引

使用带有`runCommand()`方法的 createIndex 命令来创建向量索引。语法如下：

```
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]
    }
  }]
})
```

关键参数是一个 JSON 文档，用于指定字段和向量索引类型：

```
{
  "<field>": "vector"
}
```

有关创建矢量索引的示例，请参阅索引属性。

索引属性

索引属性定义了 Amazon DocumentDB 中索引的行为和特征。了解这些属性有助于您设计满足应用程序特定要求的高效索引。

主题

- 索引属性：名称
- 索引属性：唯一
- 索引属性：稀疏
- 索引属性：部分 FilterExpression
- 索引属性：过期 AfterSeconds

索引属性：名称

支持的索引类型

Option	3.6	4.0	5.0	8.0	弹性集群
单字段	支持	是	是	是	是

Option	3.6	4.0	5.0	8.0	弹性集群
复合排序	支持	是	是	是	是
多键	支持	是	是	是	是
text	否	否	是	是	否
地理空间	支持	是	是	是	是
向量	否	否	是	是	否

使用 name 选项为索引提供可选名称。

所有示例都使用以下示例文档：

```
{
  "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')
}
```

单字段

```
db.collection.createIndex(
  {
    "productId": 1
  },
  {
    "name": "single_field_index"
  }
)
```

复合排序

```
db.collection.createIndex(
  {
    "productId": 1,
    "manufacturer": 1
  },
  {
    "name": "compound_index"
  }
)
```

Multi-key

```
db.collection.createIndex(
  {
    "tags": 1
  },
  {
    "name": "multikey_index"
  }
)
```

文本

```
db.collection.createIndex(
```

```
  {
    "name": "text"
  },
  {
    "name": "text_index"
  }
)
```

地理空间

```
db.collection.createIndex(
  {
    "supplier.location": "2dsphere"
  },
  {
    "name": "geospatial_index"
  }
)
```

向量

```
db.runCommand({
  "createIndexes": "collection",
  "indexes": [{
    "key": {
      "productEmbedding": "vector"
    },
    "name": "hnsw_index",
    "vectorOptions": {
      "type": "hnsw",
      "dimensions": 2,
      "similarity": "euclidean",
      "m": 16,
      "efConstruction": 64
    }
  }]
})
```

索引属性：唯一

支持的索引类型

Option	3.6	4.0	5.0	8.0	弹性集群
单字段	支持	是	是	是	是
复合排序	支持	是	是	是	是
多键	支持	是	是	是	是

使用 unique 选项可确保集合中各文档中各字段的唯一性。

示例

以下示例说明如何在以下示例文档中创建唯一索引：

```
{
  "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')
}
```

单字段

在 ProductID 上创建唯一的单字段索引，以确保同一个 productID 不存在于多个 1 个文档中：

```
db.collection.createIndex(
  {
    "productId": 1
  },
  {
    name: "productId_unique",
    unique: true
  }
)
```

化合物

在 SKU 和制造商上创建唯一的复合索引，以确保在多个 1 个文档中不存在相同的 SKU 和制造商组合：

```
db.collection.createIndex(
  {
    "sku": 1,
    "manufacturer": 1
  },
  {
    name: "sku_and_manufacturer_unique",
    unique: true
  }
)
```

多键

在条形码上创建唯一的多键索引，以确保条形码数组中的任何值不存在于超过 1 个文档中：

```
db.collection.createIndex(
  {
```

```
    "barcodes": 1
  },
  {
    name: "barcodes_unique",
    unique: true
  }
)
```

对数组进行索引会为数组的每个元素创建一个索引条目。例如，如果一个数组有 50 个项目，则它有 50 个索引条目。因此，唯一的多键索引强制所有单个项目的唯一性。例如，以下文档将违反 values 数组字段索引的唯一约束：

```
{ "values": [ 1, 2, 3] }
{ "values": [ 3, 2 ] }   --> 3 and 2 already exist
{ "values": [ 1 ] }      --> 1 already exists
```

请注意具有唯一索引的以下行为：

1. 如果您在现有数据上创建唯一索引，其中两个（或更多）文档的索引字段值相同，则索引构建将失败，并显示以下错误：`could not create unique index: <collection> index: <index name>`
2. 如果您插入的文档中索引字段的值与另一个文档中该字段的值相匹配，则插入操作将失败，并显示以下错误：`E11000 duplicate key error collection: <collection> index: <index name>`
3. 如果您更新现有文档，使索引字段的新值与其他文档中该字段的值相匹配，则更新将失败并显示以下错误：`E11000 duplicate key error collection: <collection> index: <index name>`
4. 如果文档中缺少索引字段，则该值将被视为空。如果两个（或更多）文档中缺少索引字段，则如上所述，索引构建、插入和更新将失败。

索引属性：稀疏

支持的索引类型

Option	3.6	4.0	5.0	8.0	弹性集群
单字段	支持	是	是	是	是

Option	3.6	4.0	5.0	8.0	弹性集群
复合排序	支持	是	是	是	是
多键	支持	是	是	是	是

使用 sparse 选项可以跳过对缺少索引字段的文档进行索引，从而减小索引大小并节省内存空间。由于索引大小较小，因此使用它的查询效率更高。要使查询使用稀疏索引，必须在索引字段上使用 $exists 子句。如果您省略 $exists 子句，Amazon DocumentDB 将不会使用稀疏索引。

示例

以下示例说明如何在以下示例文档中创建稀疏索引：

```
{
  "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')
```

```
}
```

请注意，并非所有文档中都存在评论、材料和颜色字段。

单字段

在材质字段上创建一个稀疏的单字段索引：

```
db.collection.createIndex(
  {
    "material": 1
  },
  {
    "name": "material_sparse",
    "sparse": true
  }
)
```

在查找所有具有材料价值的产品时，将使用此索引：

```
db.collection.find({
  "material": {
    $exists: true
  }
})
```

化合物

在材质和颜色字段上创建稀疏复合索引：

```
db.collection.createIndex(
  {
    "material": 1,
    "color": 1
  },
  {
    "name": "material_and_color_sparse",
    "sparse": true
  }
)
```

在查找所有具有材料和颜色值任意组合的产品时，将使用此索引：

```
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
      }
    }
  ]
})
```

多键

在 reviews 数组上创建一个稀疏的多键索引：

```
db.collection.createIndex(
  {
    "reviews": 1
  },
  {
    "name": "reviews_sparse",
    "sparse": true
  }
)
```

在查找所有具有评论数组的产品时，将使用此索引：

```
db.collection.find({
  "reviews": {
    $exists: true
  }
})
```

索引属性：部分 FilterExpression

支持的索引类型

Option	3.6	4.0	5.0	8.0	弹性集群
单字段	否	否	是	是	否
复合排序	否	否	是	是	否
多键	否	否	是	是	否
地理空间	否	否	否	否	否

使用 partial FilterExpression 选项创建仅包含满足指定筛选条件的文档的部分索引。这使您可以通过仅索引集合中的一部分文档来创建更高效的索引，而不是为所有文档编制索引，从而减少索引大小并节省内存空间。由于索引大小较小，因此使用它的查询效率更高。Amazon DocumentDB 将在以下情况下使用部分索引：

- 查询谓词与部分索引筛选表达式完全匹配。

- 查询筛选器的预期结果是部分过滤器的逻辑子集。
- 查询的子谓词可以与其他索引一起使用。

有关更多信息，请参阅 部分索引。

示例

以下示例说明如何在以下示例文档上创建部分索引：

```
{
  "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')
}
```

单字段

为深空灰色的商品创建制造商的部分单字段索引：

```
db.collection.createIndex(
  {
    "manufacturer": 1
  },
  {
    "name": "manufacturer_space_gray",
    "partialFilterExpression": {
      "color": {
        $eq: "Space Gray"
      }
    }
  }
)
```

在按制造商查找深空灰色产品时，将使用此索引：

```
db.collection.find({
  "manufacturer": "The Manufacturer",
  "color": {
    $eq: "Space Gray"
  }
})
```

化合物

为含有聚碳酸酯材料的商品创建制造商和颜色的部分化合物指数：

```
db.collection.createIndex(
  {
    "manufacturer": 1,
    "color": 1
  },
  {
    "name": "manufacturer_and_color_polycarbonate",
    "partialFilterExpression": {
      "material": {
        $eq: "Polycarbonate"
      }
    }
  }
)
```

在按制造商和颜色查找聚碳酸酯材料产品时，将使用此索引：

```
db.collection.find({
  "manufacturer": "The Manufacturer",
  "color": "Space Gray",
  "material": {
    $eq: "Polycarbonate"
  }
})
```

多键

为制造商为 “制造商” 的商品在标签上创建部分多键索引：

```
db.collection.createIndex(
  {
    "tags": 1
  },
  {
    "name": "tags_the_manufacturer",
    "partialFilterExpression": {
      "manufacturer": {
        $eq: "The Manufacturer"
      }
    }
  }
)
```

在查找带有打印机标签且制造商为 “制造商” 的产品时，将使用此索引：

```
db.collection.find({
  "tags": "printer",
  "manufacturer": {
    $eq: "The Manufacturer"
  }
})
```

索引属性：过期 AfterSeconds

支持的索引类型

索引类型	3.6	4.0	5.0	8.0	弹性集群
单字段	支持	是	是	是	是
多键	支持	是	是	是	是

使用过期AfterSeconds 选项创建存活时间 (TTL) 索引。使用 TTL 索引，您可以根据每个文档的超时条件根据文档的使用期限删除文档。当文档达到指定的 TTL 期限时，它会从收藏中删除。

有关 TTL 删除的最佳实践，请参阅TTL 和时间序列工作负载。

示例

以下示例说明如何在以下示例文档中创建 TTL 索引：

```
{
  "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')
}
```

单字段

在 LastUpdated 上创建 TTL 索引以删除在 90 天内未更新的文档：

```
db.collection.createIndex(
  {
    "lastUpdated": 1
  },
  {
    "name": "lastUpdated_ttl",
    "expireAfterSeconds": 7776000
  }
)
```

多键

在 reviews.review_date 上创建 TTL 索引，以删除在过去一年中没有评论的文档：

```
db.collection.createIndex(
  {
    "reviews.review_date": 1
  },
  {
    "name": "reviews_review_date_ttl",
    "expireAfterSeconds": 31536000
```

```
  }
)
```

请注意，如果 TTL 索引位于数组字段上，则会检查数组中的所有项目。如果数组中的任何时间戳满足超时条件，则该文档将被删除。

性能优化

本章提供有关优化 Amazon DocumentDB 性能和解决常见性能问题的指导。

主题

- 常见性能问题
- Anti-patterns
- 性能改进技巧
- 查询计划分析

常见性能问题

本节介绍您在使用 Amazon DocumentDB 时可能遇到的常见性能问题以及如何解决这些问题。

主题

- CPU 利用率高
- Amazon DocumentDB 的连接问题
- 查询运行缓慢

CPU 利用率高

CPU 利用率过高是您在使用 Amazon DocumentDB 时在某个时候可能会遇到的问题之一。本节提供解决问题的见解。

识别-发现问题

Amazon DocumentDB 实例上的 CPU 使用率过高，会增加所有数据库操作的延迟，从而导致应用程序运行缓慢。当 Amazon DocumentDB CPU 使用率超过最佳水平时，查询执行时间会增加，连接建立所需的时间更长，应用程序的响应速度也会降低。主实例的 CPU 峰值会减慢写入操作，而副本上的 CPU 峰值会导致读取操作的延迟更高。

Amazon DocumentDB 中 CPU 使用率过高的一些常见原因可能是：

- 集合扫描：没有适当索引的查询会强制执行完整的集合扫描，从而消耗过多的 CPU 来检查每个文档。

- 连接峰值：应用程序连接的突然激增使身份验证和会话管理所需的 CPU 资源不堪重负。
- 复杂聚合： Multi-stage 操作（排序、分组、联接）消耗大量的 CPU，尤其是在没有优化索引的大型数据集上。
- 查询效率低下：扫描大型结果集和正则表达式操作会导致 CPU 使用率过高。
- Write-Heavy 工作负载：频繁的更新和批量插入会给CPU带来文档处理、索引更新和事务日志的压力。

调查-收集指标

亚马逊 DocumentDB 通过亚马逊提供监控。 CloudWatchAmazon DocumentDB 指标可以大致分为集群级别和实例级指标。对于与 CPU、内存和连接相关的资源，请查看实例级别的指标，因为资源消耗是特定于实例的。确定显示 CPU 峰值的实例（主实例或副本），并记下峰值时间。

的 CPU 使用率 CloudWatch 指标`CPUUtilization`。

诊断-找出根本原因

在确定了 CPU 峰值的实例和时间表之后，接下来是诊断对 CPU 使用率有很大影响的原因。由于可能导致 CPU 峰值的原因有很多，因此让我们先确定是什么原因造成的：

连接：检查 CloudWatch 指标，`DatabaseConnections`包括`DatabaseConnectionsMax`，如果连接出现峰值并且与 CPU 峰值时间相吻合，则短时间内的连接激增很可能是 CPU 峰值的原因。

查询：如果不是连接，则可能是查询造成的。如果您已启用性能见解，请转到该实例的性能见解控制台并查看有关该实例的查询。要查看查询的另一件事来自探查器。

解决-修复问题

连接爆发：如果连接爆发是导致 CPU 峰值的原因，请考虑在应用程序中重复使用连接或实现连接池。有关优化高连接的更多信息，请参阅the section called “连接问题”。

查询分布：如果 CPU 峰值是由主实例上的读取查询引起的，如果您的查询结果最终可以保持一致，请考虑将读取操作重定向到副本实例。

查询优化：如果一个查询或一组查询导致 CPU 峰值，请考虑优化查询，例如在查询未使用索引的情况下添加索引。有关优化查询的更多信息，请参阅the section called “查询运行缓慢”。

实例扩展：如果经过优化后 CPU 利用率一直很高，请升级到具有更大 CPU 容量的更大实例类别，以有效处理您的工作负载需求。

无服务器：对于导致短期 CPU 峰值的不可预测的工作负载，可以考虑使用。使用 Amazon DocumentDB 无服务器无服务器实例可根据需求自动扩展资源，无需为此类工作负载预置固定实例大小。

Amazon DocumentDB 的连接问题

识别-发现问题

常见原因

连接问题通常源于三个主要方面：

当应用程序达到允许与 Amazon DocumentDB 的最大连接数时，连接池就会耗尽，无论是通过客户端连接池限制还是服务器端实例限制。这种情况会导致应用程序性能降低、超时以及可能出现故障，因为新的连接请求要么排队，要么被拒绝。

当 Amazon DocumentDB 遇到过多的并发身份验证请求时，就会出现身份验证过载，尤其是在短时间内处理超过 1,000 个新连接时。在身份验证维护期间，Amazon DocumentDB 在会话地图上保持独占锁，导致后续的身份验证尝试排队直到维护完成。

Amazon DocumentDB 中的配置问题通常源于网络、安全和客户端设置的错误配置。这包括安全组设置不正确、VPC 配置不正确或 SSL/TLS 证书问题等问题。了解正确的配置对于维护数据库访问的安全性和可靠性至关重要。

诊断-找出根本原因

连接池

创建 MongoClient 实例时，连接池会初始化。每个池都基于两个关键参数维护连接：

min PoolSize -保持的最小连接数

max PoolSize -允许的最大连接数

当请求需要连接时：

1. 池会检查是否有可用的空闲连接
2. 如果不存在且池大小小于 maxPoolSize，则会创建新连接
3. 如果达到最大值PoolSize，则请求进入等待队列
4. 如果队列已满或已达到超时，则会抛出 MongoWaitQueueFullException

等待队列的行为通过以下参数处理：

等待 QueueTimeout MS-连接的最大等待时间

等待 QueueSize -排队请求的最大数量

以下是连接到 Amazon DocumentDB 时出现问题的方法示例，在这种方法中，每次都会创建一个新池：

```
for(Request request : requests) {
    MongoClient client = MongoClients.create(settings);
    // Process request
    client.close();
}
```

要监控的关键 CloudWatch 指标有：

- DatabaseConnections-以 1 分钟为频率拍摄的实例上打开的连接数（活动和空闲）。
- DatabaseConnectionsMax-1 分钟内实例上打开的数据库连接（活动和空闲）的最大数量。
- DatabaseConnectionsLimit-在任何给定时间，实例上允许的最大并发数据库连接（活动和空闲）数。
- LowMemNumOperationsThrottled-1 分钟内由于可用内存不足而受到限制的请求数。

有关每个实例类别的限制，请参阅配额和限制。

应用程序级别连接池问题的常见警告信号包括：

- 增加连接获取时间
- 等待队列大小不断增加
- 超时异常数量不断增加

身份验证过载

连接亚马逊 DocumentDB 的流程如下所示：

连接请求 → SSL 握手 → 身份验证 → 会话创建 → 连接就绪

当处理超过 1,000 个新连接时，在完成 SSL 握手后，其他连接请求将进入身份验证队列。在这些过载事件期间，您的应用程序的平均连接时间将增加。

要监控的关键 CloudWatch 指标有：

- `DatabaseConnections`-以 1 分钟为频率拍摄的实例上打开的连接数（活动和空闲）。
- `DatabaseConnectionsMax`-1 分钟内实例上打开的数据库连接（活动和空闲）的最大数量。
- `DatabaseConnectionsLimit`-在任何给定时间，实例上允许的最大并发数据库连接（活动和空闲）数。

配置问题

最常见的配置问题是在尝试从无法访问私有网络环境的环境中连接私有 Amazon DocumentDB 集群终端节点时引起的。Amazon DocumentDB 仅面向虚拟私有云（VPC），并且当前不支持公共端点。您无法从笔记本电脑或 VPC 之外的本地开发环境直接连接到您的 Amazon DocumentDB 集群。

这将在错误中表现出来，如下所示：

```
Error: couldn't connect to server...
Failed to connect to...
exception: connect failed
connection attempt failed
```

不正确的安全组配置也可能导致连接失败。默认情况下，Amazon DocumentDB 集群监听 TCP 端口 27017 上的连接。如果尝试连接到与集群部署时使用的端口不同的端口，或者如果该应用程序未包含在集群的入口安全组配置中，则您的应用程序将失败。

不正确的证书管理也可能导致连接问题。默认情况下，为新创建的 Amazon DocumentDB 集群启用传输中加密。启用传输中加密后，需要使用 TLS 进行安全连接才能使用 global-bundle.pem 证书连接到集群。如果您尝试使用不正确的证书，则会收到错误消息，例如：

```
unable to get local issuer certificate
```

如果在未指定 TLS 参数的情况下尝试连接到启用了 TLS 的集群，则会收到如下错误：

```
Server selection timed out after 30000 ms
```

解决-修复问题

连接池：通过实施或调整池大小来审查连接池以满足工作负载要求。最佳池配置取决于您的工作负载和要求。你应该保持一分钟，PoolSize 使核心连接准备就绪并可用，并且最大值足够WaitTime 短，以便在池用尽时快速失败。

以下是如何在不每次都创建新池的情况下重复使用单个池的示例：

```
MongoClient client = MongoClients.create(settings);
    for(Request request : requests) {
    // Process request
}
```

身份验证过载：通过逐步增加连接并将新连接限制为 1,000 个来管理身份验证。使用连接池可以有效地重复使用经过身份验证的连接。为避免连接过载 Amazon DocumentDB 集群，请实施连接提升策略。

```
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;
        }
    }
}
```

您也可以配置连接池设置以限制允许的连接总数。

```
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();
```

配置问题：确保您的应用程序可以访问您的 Amazon DocumentDB 资源所在的私有 VPC 和子网。如果使用 VPC 对等连接，请查看开发者指南 VPC 对等连接疑难解答以了解更多信息。您也可以查看知识中心文章如何解决从互联网到我的 VPC 内的 Amazon EC2 实例的连接问题？ 。

要配置安全组，您必须在 Amazon DocumentDB 安全组中包含一条入口规则，以允许来自您的应用程序的连接。

```
{
  "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"
    }
  ]
}
```

如果集群配置了 TLS 加密，请下载名为 global-bundle.pem 的亚马逊 Amazon DocumentDB 的 TLS 证书，并在连接到集群时使用该证书。

```
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem
```

Long-term 解决方案

可能需要通过升级到更大的实例类或添加只读副本来分散连接负载来扩展实例。适当的负载平衡实施可确保整个群集的最佳资源利用率。

应用程序变更应侧重于实现强大的连接处理、全面监控和遵守连接池最佳实践。这包括适当的错误处理和连接生命周期管理。

架构改进可能包括为可变工作负载采用 Amazon DocumentDB Serverless、实现复杂的重试逻辑以及容错设计。考虑重组应用程序架构，以更好地处理连接管理。

最佳实践

连接池

通过适当的连接池管理和监控，应用程序可以保持稳定的数据库连接，同时防止可能影响系统可靠性和性能的耗尽情况。根据工作负载的特性配置适当的超时时间并调整池的大小。

连接池设置示例

```
MongoClientSettings settings = MongoClientSettings.builder()
    .applyToConnectionPoolSettings(builder ->
        builder.maxSize(10))
    .applyToConnectionPoolSettings(builder ->
        builder.maxWaitQueueSize(2))
    .applyToConnectionPoolSettings(builder ->
        builder.maxConnectionIdleTime(10, TimeUnit.MINUTES))
    .build();
```

欲了解更多信息，请参阅：https://aws.amazon.com/blogs/database/building-resilient-applications-with-amazon-documentdb-with-mongodb-compatibility-part-1-client-configuration/

身份验证过载

请务必根据您的工作负载使用适当的参数值实现连接池。尽可能使用渐进式连接建立技术并保持持久连接。实施适当的连接清理，确保不会浪费闲置资源。

配置问题

确保您已经配置了从应用程序到 Amazon DocumentDB 资源的适当路由。使用 TLS 对传输过程进行加密并实现最低权限访问。验证您的亚马逊 DocumentDB 凭证并验证连接字符串值。

查询运行缓慢

识别-发现问题

运行缓慢的查询是指超出业务需求预期执行时间的查询。构成慢速查询的定义因工作负载而异。您可以通过探查器日志或 Performance Insights 来识别慢速查询。

1. 如果尚未启用，请启用探查器。Profiler 日志在日志组 CloudWatch 下写入 Amazon: `/aws/docdb/<cluster-identifier>/profiler`。使用 CloudWatch Logs Insights 进行查询。

   获取 ecomerce.products 集合最慢的 10 个查询的 CloudWatch 日志分析查询@@ 示例：

   ```
   filter ns="ecommerce.products" | sort millis desc | limit 10
   ```

2. 使用 Performance Insights 近乎实时地识别昂贵的查询。启用 Performance Insights（如果尚未启用）。

   a. 打开 AWS 控制台，导航到亚马逊 Amazon DocumentDB，然后导航到 Performance Insights，然后选择您的集群实例。

   b. 查看数据库加载 (AAS) 时间表和热门查询（按数据库负载）。展开查询摘要以查看字面子语句。

   c. 捕获需要分析的查询。

   > **Note**
   >
   > 不是 Performance Insights 中的所有查询都可能是低效或缓慢的查询。

调查-收集信息

1. 分析器提供查询执行计划和与之相关的关键指标，包括 millis、nreturn 和 PlanSummary（索引使用情况）：

   ```
   {
       "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"
}
```

要查找 COLLSCAN 查询：

```
filter planSummary="COLLSCAN" | sort millis desc | limit 20
```

2. 使用 Performance Insights 实时分析查询执行趋势，例如等待、负载和资源影响（例如，每次查询的 IOPS 或 CPU）。

   由于 Performance Insights 不提供查询执行计划，因此请捕获查询并在 Amazon DocumentDB 集群上对查询运行 explain（“ExecutionStats”）：

   ```
   db.ecommerce.products.find().explain("executionStats")
   ```

3. 或者，将探查器指标与 Performance Insights 数据相关联（例如，将分析器中的高毫利查询与 Performance Insights 中的热门查询进行匹配）。

诊断 — 找出根本原因

在此步骤中，诊断查询计划以确定潜在的优化。使用以下流程-症状 → 可能的原因：

- 症状：计划摘要：“COLLSCAN”

  原因：索引缺失或不正确。

- 症状：$group/$sort 的聚合速度很慢

  原因：聚合管道可能处理的内存数据过多。

- 症状：尽管使用了索引，但 Performance Insights 显示按照 IO 等待分组的数据库负载（PlanSummary：“IXSCAN”），但延迟很高。

  原因： I/O 绑定查询；索引或工作集超出实例上可用的缓冲区缓存。

- 症状：PI 显示 CPU 等待状态，由于查询很少，AAS 过高

  原因： CPU-bound 查询（复杂聚合，$regex）。

- 症状：许多查询速度很慢，但没有一个查询显示昂贵的计划摘要

  原因：外部瓶颈（网络、限制、维护任务、查询量）。

解决-修复问题

实施修复时，请务必使用解释（“ExecutionStats”）进行验证，并监控 Performance Insights 数据库负载。

1. 计划摘要：“COLLSCAN”
   - 创建目标索引。

     示例：对于按 {类别，价格} 筛选并按价格降序排序的频繁查询：

     ```
     db.products.createIndex({ category: 1, price: -1 })
     ```

2. 使用 $group/$sort 聚合速度很慢
   - 尽早推送 $match 和 $project 以减少流入 $group/$sort 的文档。
   - 在管道初期限制字段的数量：

     ```
     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. 高延迟，而 Performance Insights 显示按 IO 等待分组的数据库负载
   - 验证实例上 BufferCacheHitRatio 的 readiOps 是否为低或为高。增加实例内存（升级实例类，例如 r6g.large → r6g.xlarge）或使用 NVMe 实例类。
   - 减少索引占用空间。
   - 添加只读副本以卸载读取流量（如果查询容忍且最终保持一致，则使用 readPreference 设置在副本上重定向流量）。
4. PI 显示 CPU 等待状态，由于查询很少，AAS 很高
   - 将昂贵的 $regex 替换为索引前缀搜索或 $text 索引。
   - Batch 写入（插入）以减少写入放大。

> **Note**
>
> 在将这些更改升级到生产环境之前，请务必在较低的环境（非生产环境）中测试这些更改。

Anti-patterns

本节介绍可能对 Amazon DocumentDB 性能产生负面影响的常见反模式。

主题

- 具有 3 个以上属性的复合索引
- 长时间运行的查询
- 低使用率和冗余索引
- Multi-Key 包含大型数组的索引

具有 3 个以上属性的复合索引

概述

复合索引或复合索引在单个索引结构中保留对多个字段的引用。这些索引可优化同时筛选多个字段或将筛选与排序操作相结合的查询的性能。它们对最左边的索引字段的单条件查询也很有效。数据库利用这些索引条目来高效地找到匹配的文档，而无需执行完整的馆藏扫描。

Amazon DocumentDB 可以使用复合索引来支持包含索引字段任何前导子集的查询，这个概念被称为索引前缀。例如，如果您开启了复合索引`{state: 1, city: 1, zipcode: 1}`，Amazon DocumentDB 可以高效地处理仅使用字段 “州”、同时使用 “州” 和 “城市” 字段，或者同时使用三个字段 “州”、“城市” 和 “邮政编码” 的查询。但是，查询必须使用从左到右的字段，不要跳过介于两者之间的任何字段。这意味着仅使用字段 “城市” 或 “邮政编码” 或 “州” 和 “邮政编码” 之类的组合（跳过 “城市”）的查询无法充分利用索引。

在大多数现实场景中，具有三个或更少属性的复合索引通常可以实现最佳性能和资源效率。虽然在综合指数中使用3个以上的属性是可行的，但它通常会导致资源消耗增加，超过收益。

对集群的影响

虽然创建涵盖查询中所有条件的索引以实现最佳理论性能很诱人，但大多数数据筛选都是通过复合索引中的前 1-3 个属性进行的。超过此阈值的其他字段主要影响索引大小，而不是有意义的查询优化。

存储和 I/O 开销：具有许多属性的复合索引比更简单的索引占用的存储空间要多得多。大小与索引属性的数量以及索引值本身的大小成正比。

内存占用：具有许多属性的复合索引的大量存储空间会在内存中造成相应的占用空间，从而使您的工作集更大，并取代缓冲池中其他经常访问的数据。

写入操作：每个影响多个索引字段的文档修改都需要更新整个复合索引条目，乘以完成写入操作所需的工作量。

如何识别

首先查看集合中的所有索引，找出具有三个以上属性的复合索引：

```
// 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 }
```

修复

如果使用的是具有三个以上属性的复合索引，请确定使用该索引的查询并寻找优化机会。考虑用包含三个或更少属性的更高效的索引替换这些索引。实现新索引后，删除包含三个以上属性的旧索引。

> **Note**
>
> 在删除任何索引之前，请务必与利益相关者协调并验证性能影响。

创建复合索引时，请遵循相等、排序、范围 (ESR) 规则。

相@@ 等排序范围 (ESR) 规则：这种排序通过首先使用相等条件筛选数据集，然后对简化的集合应用排序操作，最后对尽可能小的子集执行范围扫描，从而最大限度地提高索引效率

- 将字段排序为相等（完全匹配），
- 排序（排序），
- 范围 (>、<、$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
```

长时间运行的查询

概述

Long-running 查询可能会级联成群集范围的性能问题。这些查询可能会干扰 Amazon DocumentDB 的并 Multi-Version 发控制 (MVCC) 垃圾收集流程，从而导致旧版本文档积累并降低整个集群的性能。Amazon DocumentDB 实施了 2 小时的服务器端超时作为一种安全机制，以无限期限制失控的查询消耗资源。

什么是长时间运行的查询：

- 长时间执行的查询（通常大于 30 分钟）
- 打开保持活动状态数小时的游标（如果光标处于活动状态，则不适用 2 小时的超时时间）

对集群的影响

长时间运行的查询可能会干扰垃圾收集进程

- 旧版本累积：垃圾收集器无法回收旧文档版本
- 集合和索引膨胀：集合和索引条目会随着时间的推移而累积，膨胀会增加，这可能会导致更多的存储成本。
- CPU 和内存压力：由于处理越来越多的旧文档版本、索引条目和事务 ID 的效率低下，CPU 和内存压力增加。

长时间运行的查询 → 阻止 GC → 存储增长 → CPU 和内存压力 → 更多长查询

监控和检测

1。要查找长时间运行的查询，请使用 currentOp 命令。

```
// 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。查找活动时间超过 30 分钟的光标

```
// 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。通过监控垃圾收集器的进度 CloudWatch

LongestRunningGCProcess— 最长活动垃圾收集过程的持续时间（以秒为单位）。每分钟更新一次，仅跟踪活动操作，不包括在一分钟时段内完成的进程。

AvailableMVCCIds-一个计数器，显示在达到零之前剩余的可用写入操作数。当此计数器达到零时，您的集群将进入只读模式，直至 ID 被回收并循环利用。计数器会在每次写入操作后递减，并在垃圾回收利用旧的 MVCC ID 后递增。

> **Note**
>
> 较低的 MVCC ID 和较长的垃圾收集持续时间并不完全归因于长时间运行的查询。 Write-intensive 资源受限的实例上的工作负载还可能导致 MVCC ID 可用性降低，垃圾收集周期延长。

补救策略

- 在应用程序中实现查询超时
- 不要让光标存活时间更长
- 优化查询以提高性能。
- 更喜欢批量写入操作

低使用率和冗余索引

概述

索引可以提高工作负载的整体查询性能，但会带来成本。每个索引都消耗存储空间和 I/O 资源，写入操作需要更新集合上的所有索引。如果您的数据库包含未使用、未充分利用或冗余的索引，则会给集群带来开销，从而导致性能下降。多个不必要的索引会产生一种反模式，从而导致性能问题、存储成本增加和运营开销增加。

次优索引场景

- 未使用的索引：为一次性查询创建的索引，或者为当前查询模式无法再访问的早期产品迭代创建的索引。要识别集合上未使用的索引，你可以使用 IndexStats。有关更多信息，请参阅 the section called “分析索引使用情况并识别未使用的索引”。
- 冗余索引：涵盖重叠键模式或相同查询模式的多个索引。示例：当复合索引同时涵盖属性 A 和 B 时，属性 A 上的单键索引会变得多余，因为复合索引可以单独处理对属性 A 的查询。
- Over-Indexing：在不分析实际查询模式或性能要求的情况下，创建索引 “以防万一”。
- 低基数索引：对具有很少不同值的字段（例如布尔字段、状态标志）的索引，其查询优化优势微乎其微。

对集群性能的影响

- 存储和 IO：每个索引都会消耗存储空间和 IO 资源，从而增加总体成本。如果索引未使用或冗余，则会给您的集群带来不必要的成本。
- 写入性能降低：插入、更新和删除操作必须维护索引，这会产生消耗资源的开销。未使用或冗余的索引会给这些操作增加不必要的开销。
- 内存和 CPU 压力：索引争夺缓冲池内存，如果工作集不适合实例内存，则索引可能会删除经常访问的数据，从而导致次优 BufferCacheHitRatio化。系统还消耗维护索引结构的 CPU 资源。

可用于识别指数优化机会的工具

- 索引审查工具：
- 索引基数检测工具

正在收集工具包：

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

1。索引审阅工具

索引审查工具会分析所有集合和索引，以提供索引列表及其使用模式，并识别您的 Amazon DocumentDB 集群上的冗余索引。在所有集群实例上执行此工具以进行全面分析。有关更多信息，请参阅索引审阅工具。

用法：

```
cd performance/index-review/

python3 index-review.py --server-alias <server-alias> --uri <mongodb-uri>
```

输出文件

- `{server-alias}-collections.csv`: Collection-level 统计
- `{server-alias}-indexes.csv`: 详细的索引使用指标
- `{server-alias}-{timestamp}-index-review.json`: 用于进一步分析的原始数据

> **Note**
>
> 使用独立别名为集群中的每个实例运行以生成每个实例的统计信息，然后运行该工具以获得累积分析

例如：`python3 index-review.py --files "{server-alias}-{timestamp}-index-review.json, {server-alias}-{timestamp}-index-review.json" --server-alias full-review`

2。索引基数检测工具

此工具可识别基数较低、对查询性能不起作用的索引。

用法

```
cd performance/index-cardinality-detection/

python3 detect-cardinality.py --uri <mongodb-uri>
```

参数

- `--threshold`: 基数阈值百分比（默认值：1%）
- `--sample-count`: 要采样的文档数（默认值：100,000）
- `--max-collections`: 每个数据库要扫描的最大集合数（默认值：100）

有关更多信息，请参阅 https://aws.amazon.com/blogs/database/detect-and-fix-low-cardinality-indexes-in-amazon-documentdb/。

补救策略

1。移除未使用和冗余的索引

```
// Example: Drop unused index
db.collection.dropIndex("unused_index_name")
```

> **Note**
>
> 如果不与所有利益相关方讨论并测试性能，就不应删除索引。

2。低基数索引优化

- 将部分索引与筛选器一起使用
- 将单个低基数索引转换为复合索引

Multi-Key 包含大型数组的索引

概述

Multi-key 借助 Amazon DocumentDB 中的索引，您可以高效地查询数组字段。在数组数据中搜索时，这种索引策略可以显著降低查询延迟，使其成为具有复杂数据结构的应用程序的有吸引力的选择。

但是，在处理大型数组或集合中的多个数组字段时，重要的是要了解资源影响和性能特征，以便做出明智的架构决策。当您在数组字段上创建索引时，Amazon DocumentDB 会为数组中的每个元素生成单独的索引条目。

对集群的影响

- 存储和 IO 开销： Multi-key 索引有时会占用基表的多倍存储空间，大小与集合中带有索引数组属性的文档、数组中的元素数量和元素的大小成正比。
- 内存使用量：由于存储空间占用量相对较大，因此这些索引在内存中具有相应的占用空间，从而产生更大的工作集。
- 写入操作： Multi-key 索引在写入操作期间会产生开销。每个数组元素生成单独的索引条目，乘以完成写入操作所需的工作量。

补救策略

- 必要时创建多键索引，并限制数组中的字段数。
- 限制集合上多键索引的数量
- 考虑修改您的数据模型

性能改进技巧

本节为 Amazon DocumentDB 提供了五种性能优化策略，以提高应用程序效率和查询执行。

1. 使用 $match 作为聚合管道的第一阶段

始终将 $match 作为聚合管道中筛选的第一阶段，以最大限度地提高性能。当 $match 处于领先地位时，Amazon DocumentDB 将有效地利用索引，从而使数据库能够尽早筛选数据并减少处理开销。

```
// Optimized approach
db.orders.aggregate([
  { $match: { status: "active", category: "electronics" } }, // Index utilization
  { $group: { _id: "$category", total: { $sum: "$price" } } },
  { $sort: { total: -1 } }
])
```

影响：早期筛选减少了后续管道阶段处理的文档数量，从而加快了查询执行速度并降低了资源消耗。

2. 在聚合管道中使用 $project 来最小化管道数据大小

在聚合管道阶段仅传送基本字段，以最大限度地减少数据大小并提高性能。策略性地使用 $project 来仅包含您需要的数据。

```
// 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" } } }
])
```

影响：较小的文档可减少内存使用量并提高管道处理效率，从而增强整体查询性能。

3. 启用文档压缩以降低存储 I/O 成本、成本并提高查询性能

启用集群参数组中的文档压缩，以降低存储成本和 I/O 成本，并提高查询性能。Amazon DocumentDB 将压缩后的文档存储在磁盘和 RAM 中，从而减少了内存占用和 I/O 成本。

影响：

- 可用存储器中可容纳更多文档
- 通过减少磁盘读取次数，加快数据访问速度
- 降低存储成本和 I/O 成本，提高查询性能

> **Note**
>
> 默认情况下，亚马逊 DocumentDB 不为版本 5.0 启用压缩。您可以在 5.0 集群的集合或集群级别启用压缩。使用 Amazon DocumentDB 的压缩查看工具来分析您的馆藏的压缩率。
> 对于 Amazon DocumentDB 8.0，默认启用压缩。

4. 利用索引实现最佳查询性能

确保您的查询始终使用索引以获得最佳性能。亚马逊 DocumentDB 提供多种索引类型来匹配不同的用例。

索引原理：

- 每个查询都应利用适当的索引
- 亚马逊 DocumentDB 提供多种索引类型
- 复合索引通过单个索引支持各种查询形状，从而提供了最大的灵活性
- 设计索引以同时支持排序和筛选操作

了解索引前缀：复合索引通过索引前缀起作用——Amazon DocumentDB 可以使用索引字段中从左到右的任何子集。例如，该索引`{ category: 1, price: -1, inStock: 1 }`创建了以下可用的前缀：

- `{ category: 1 }`-仅支持按类别筛选查询
- `{ category: 1, price: -1 }`-支持按类别和 sorting/filtering 价格筛选查询
- `{ category: 1, price: -1, inStock: 1 }`-支持完整复合查询

仅针对价格、InStock 或 InStock 的查询不会使用此索引，因为它们不是从第一个字段（类别）开头。

如何识别不使用索引的查询：使用 explain () 方法来分析查询执行情况，并识别执行集合扫描而不是使用索引的查询。

影响：未使用索引的查询会导致集合扫描，从而增加实例的内存和 CPU 压力，并延长查询延迟。

5. 根据查询模式优化数据模型

使您的数据模型与应用程序查询和更新数据的方式保持一致。数据建模是高性能 Amazon DocumentDB 应用程序的基础。

优化策略：

为提高性能而嵌入

- 当经常作为一个单元访问相关数据时，将其存储在一起
- 嵌入始终一起检索的文档
- 适合一对几的关系

```
// 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" }
  ]
}
```

灵活性参考

- 对大型数据或不常访问的数据使用引用
- 建议用于具有大型数据集的一对多关系
- 防止文档膨胀并提高更新性能

馆藏拆分策略

当大型文档中只有几个字段经常更新，或者不经常访问的大型数据使文档膨胀时，可以考虑拆分集合：

- 将经常更新的字段保存在一个单独的、较小的集合中
- 将静态或不常访问的数据存储在另一个集合中
- 需要时将它们与参考文献联系起来

```
// 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 */]
}
```

性能提升：较小的文档意味着更快的更新、更少的内存使用和更高的缓存效率。

影响：数据建模效率低下会导致查询次优化、文档大小增加和内存使用量增加，从而导致应用程序性能下降和运营成本增加。

查询计划分析

通过解释计划进行查询计划分析可提供对 Amazon DocumentDB 查询性能的基本见解。使用 ExecutionStats 的查询计划揭示了关键指标，包括：

- 每个阶段返回的文档（n已返回）
- Stage-specific 执行时间（执行TimeMillisEstimate）
- 计划生成持续时间（计划TimeMillis）

通过检查查询计划输出，开发人员可以分析执行模式，评估索引利用率，并识别整个查询管道阶段的潜在优化机会。

要分析查询计划，您可以使用以下格式的 explain () 命令。

```
db.runCommand({explain: {query document}, verbosity: "executionStats"})
db.collection.find().explain("executionStats");
```

以下是一个操作示例：

```
db.collection.find({ companyname: { '$eq': 'ANYCOMPANY' }, isDeleted: { '$eq':
 false } }).sort({"createdAt":1}).limit(2).explain("executionStats");
```

此操作的输出将类似于以下内容：

```
{
  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 })
}
```

以下是对 Amazon DocumentDB 查询执行计划的详细分析，详细分析了每个组件及其性能特征。

总体时机

执行TimeMillis 表示查询所花费的总时间，包括计划时间。

planning TimeMillis 表示查询所花费的总计划时间。

执行阶段

它描述了 Amazon DocumentDB 用于执行查询的分步过程，显示了数据如何通过不同的操作流动。

```
"executionStages": {
    "stage": "[STAGE_NAME]",
    "nReturned": "[NUMBER_OF_DOCS]",
    "executionTimeMillisEstimate": "[TIME]",
    "inputStage": {
        // Nested stages
    }
}
```

查询计划中的常见阶段

以下是查询计划中常见的执行阶段。每个阶段都返回执行TimeMillisEstimate （执行时间）和 nReturn（文档数）指标，以帮助评估每个阶段的查询性能。

COLLSCAN（馆藏扫描）

- 逐个文档扫描整个馆藏文档
- nReturen：返回集合中所有匹配的文档

- 可能是一项昂贵的操作，在没有可用的索引时可以看到

IXSCAN（索引扫描）

- 使用索引查找匹配的文档
- nReturen：仅根据索引匹配文档
- 高效运行，比 COLLSCAN 更受青睐

SORT

- 根据指定字段对文档进行排序
- 如果查询中的排序属性不是索引的一部分，则此阶段将显式显示
- n返回：与输入文档的编号相同
- Memory-intensive 用于大型结果集。要进行优化，请将排序字段名称添加到索引中

极限_跳过

- 控制返回和跳过的文档数量
- n返回：受限值限制
- 用于分页或限制操作

子扫描

- 执行嵌套查询操作
- nReturen：因子查询结果而异
- 用于具有多个阶段的复杂查询

执行力高的阶段TimeMillisEstimate 是优化的理想选择。

> **Note**
>
> ExecutionStats 参数目前不支持更新和删除命令。

在 “解释计划执行统计信息” 中了解已检查的文档

当您使用运行查询时`explain("executionStats")`，Amazon DocumentDB 会提供检查指标，帮助您了解为生成查询结果而扫描了多少文档。这些指标对于识别低效的查询计划和优化性能非常有用。

> **Note**
>
> 仅在亚马逊 DocumentDB 8.0.0+ 版本中可用。

字段

字段	说明	级别
总计 DocsExamined	在所有执行阶段检查的文档总数。	Top-level 执行统计
已检查文档	特定执行阶段检查的文件数量。	Stage-level

该`docsExamined`字段出现在以下阶段：

暂存区	说明
COLLSCAN	馆藏扫描。馆藏中的所有文件都经过检查。
IXSCAN	索引扫描。仅检查符合索引条件的文档。
FETCH	当优化器在与 IXSCAN 不同的阶段检索文档时，FETCH 阶段会报告 docSexamined。在索引扫描计划中，子级 IXSCAN 阶段不报告 docSexamined。在 $lookup 计划中，FETCH 阶段及其子阶段都可能报告 docSexamined。

> **Note**
>
> `docsExamined`不会出现在 IXONLYSCAN 阶段上，因为无需访问文档即可完全从索引中满足查询。

在 “解释计划” 执行统计输出中显示已检查的文档

以下示例使用包含 500,000 个文档的集合，并演示了不同查询执行阶段的docsExamined变化情况。

创建示例数据：

```
for (let i = 0; i < 500000; i++) {
    db.coll.insertOne({ number: i, arr: [i, [i+1]], value: "test", bool: i % 2 === 
 0 });
}
```

馆藏扫描 (COLLSCAN)

如果没有索引，Amazon DocumentDB 会执行完整的馆藏扫描。

```
db.coll.find({ "number": { "$lt": 500 } }).explain("executionStats").executionStats
```

输出：

```
{
    "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
            }
        }
    }
}
```

对所有 500,000 份文件进行了检查，得出了 500 份结果。在数字字段上创建索引可以改善这种情况。

索引扫描 (IXSCAN)

```
db.coll.createIndex({ number: 1 });
db.coll.find({ "number": { "$lt": 5000 } }).explain("executionStats").executionStats
```

输出：

```
{
    "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
                    }
                }
            ]
        }
    }
}
```

使用该索引，在50万份文档中仅检查和提取了5,000份，与返回的数字相符。索引条件筛选出了 49.5万份与查询不匹配的文档。

使用残留过滤器进行索引扫描

```
db.coll.find({ "number": { "$lt": 5000 }, "arr": { "$gt":
 4000 } }).explain("executionStats").executionStats
```

输出：

```
{
    "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
            }
        }
    }
}
```

索引条件匹配了 500,000 个文档中的 5,000 个，然后启用剩余过滤器将结果arr减少到 999 个。5,000 的docsExamined值反映了在索引条件之后但在应用残差过滤器之前检查的所有文档。

使用 IXSCAN 获取

```
db.coll.find({ "$or": [{ "number": { "$lt": 100000 } }, { "number": { "$gt":
 400000 } }] }).explain("executionStats").executionStats
```

输出：

```
{
    "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
                                }
                            }
                        ]
                    }
                }
            ]
        }
    }
}
```

当 Amazon DocumentDB 优化器使用提取阶段检索文档时，FETCH 阶段会报告。docsExamined子级 IXSCAN 阶段不报告，docsExamined因为它们只扫描索引键，而不直接访问文档。

使用聚合查询获取

```
db.coll.explain("executionStats").aggregate([
    { $match: { "number": { "$lt": 5 } } },
    { $lookup: { from: "coll", pipeline: [{ $match: { "number": { "$lt": 3 } } }], as:
 "sub" } }
]).executionStats
```

输出：

```
{
    "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
```

```
                                    }
                                }
                            ]
                        }
                    }
                }
            }
        ]
    }
}
```

外部 IXSCAN 检查了 5 份与数字 < 5 相匹配的文档。内部 IXSCAN 检查了 3 个与数字 < 3 匹配的文档，FETCH 阶段检查了 1 个汇总结果。9 `totalDocsExamined` 中的是所有阶段的总和（5 + 3 + 1）。

Amazon DocumentDB 故障排除

以下部分提供了有关如何排查您在使用 Amazon DocumentDB（与 MongoDB 兼容）时可能遇到的问题的信息。

主题

- 排除连接问题
- 索引故障排除
- 性能和资源利用率故障排除
- Amazon DocumentDB 中的垃圾回收

排除连接问题

连接遇到问题？ 以下是一些常见场景及如何解决。

主题

- 无法连接到 Amazon DocumentDB 端点
- 测试与 Amazon DocumentDB 实例的连接
- 连接到无效终端节点
- 影响连接数的驱动程序配置

无法连接到 Amazon DocumentDB 端点

当您尝试连接到 Amazon DocumentDB 时，以下错误消息是您可能收到的最常见的错误消息之一。

```
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
```

此错误消息通常意味着您的客户端（此示例中为 mongo Shell）无法访问 Amazon DocumentDB 端点。以下几个原因可能会导致出现此情况：

主题

- 从公有端点连接
- 跨区域连接
- 从不同的 Amazon VPC 连接
- 安全组阻止入站连接
- Java Mongo 驱动程序读取首选项问题

从公有端点连接

您正在尝试直接通过笔记本电脑或本地开发计算机连接到 Amazon DocumentDB 集群。

从公有端点（例如，您的笔记本电脑或本地开发机）直接连接到 Amazon DocumentDB 集群的尝试会失败。Amazon DocumentDB 仅面向虚拟私有云（VPC），并且当前不支持公共端点。因此，您无法从笔记本电脑或 VPC 外部的本地开发环境直接连接到 Amazon DocumentDB 集群。

要从 Amazon VPC 外部连接到 Amazon DocumentDB 集群，您可使用 SSH 隧道。有关更多信息，请参阅 从 Amazon VPC 外部连接到 Amazon DocumentDB 集群。此外，如果您的开发环境位于不同的 Amazon VPC 中，您还可以使用 VPC 对等，并从同一区域或不同区域中的另一个 Amazon VPC 连接到您的 Amazon DocumentDB 集群。

跨区域连接

您正在尝试连接到另一个区域中的 Amazon DocumentDB 集群。

如果您尝试从集群区域以外某区域的 Amazon EC2 实例连接到 Amazon DocumentDB 集群，例如，尝试从美国西部（俄勒冈州）区域 (us-west-2) 连接到美国东部（弗吉尼亚州北部）区域 (us-east-1) 中的集群，连接将失败。

要验证您的 Amazon DocumentDB 集群的区域，请运行以下命令。区域位于终端节点中。

```
aws docdb describe-db-clusters \
   --db-cluster-identifier sample-cluster \
```

```
   --query 'DBClusters[*].Endpoint'
```

此操作的输出将类似于下文。

```
[
    "sample-cluster.node.us-east-1.docdb.amazonaws.com"
]
```

要验证您的 EC2 实例的区域，请运行以下命令。

```
 aws ec2 describe-instances \
     --query 'Reservations[*].Instances[*].Placement.AvailabilityZone'
```

此操作的输出将类似于下文。

```
[
    [
        "us-east-1a"
    ]
]
```

从不同的 Amazon VPC 连接

您正在尝试通过一个 VPC（不同于您的集群部署到的 Amazon VPC）连接到 Amazon DocumentDB 集群。

如果您的亚马逊 DocumentDB 集群和 Amazon EC2 实例位于相同 AWS 区域但不在同一个亚马逊 VPC 中，则除非在两个 Amazon VPC 之间启用 VPC 对等，否则您无法直接连接到您的亚马逊文档数据库集群。

要验证 Amazon DocumentDB 实例的 Amazon VPC，请运行以下命令。

```
aws docdb describe-db-instances \
   --db-instance-identifier sample-instance \
   --query 'DBInstances[*].DBSubnetGroup.VpcId'
```

要验证您的 Amazon EC2 实例的 Amazon VPC，请运行以下命令。

```
aws ec2 describe-instances \
   --query 'Reservations[*].Instances[*].VpcId'
```

安全组阻止入站连接

您正在尝试连接到 Amazon DocumentDB 集群，并且该集群的安全组不允许集群端口（默认端口：27017）上的入站连接。

假设您的 Amazon DocumentDB 集群和 Amazon EC2 实例位于同一区域和 Amazon VPC 中，并且使用相同的 Amazon VPC 安全组。如果您无法连接到您的 Amazon DocumentDB 集群，这可能是因为您的集群的安全组（即防火墙）不允许您为 Amazon DocumentDB 集群选择的端口（默认端口为 27017）上的入站连接。

要验证您的 Amazon DocumentDB 集群的端口，请运行以下命令。

```
aws docdb describe-db-clusters \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[DBClusterIdentifier,Port]'
```

要获取您的集群的 Amazon DocumentDB 安全组，请运行以下命令。

```
aws docdb describe-db-clusters \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[VpcSecurityGroups[*],VpcSecurityGroupId]'
```

要检查您的安全组的入站规则，请参阅 Amazon EC2 文档中的以下主题：

- 为您的 Linux 实例授权入站流量
- 为您的 Windows 实例授权入站流量

Java Mongo 驱动程序读取首选项问题

客户端读取首选项未被接受，并且除非重启，某些客户端无法在失效转移后写入 Amazon DocumentDB。

这个问题最初在 Java Mongo Driver 3.7.x 中发现，当客户端使用 `MongoClientSettings` 与 Amazon DocumentDB 建立连接时，尤其链接 `applyToClusterSettings` 方法时，会发生。可以使用几种不同的方法来定义 MongoClient 集群设置，例如`hosts()`、`requiredReplicaSetName()`和`mode()`。

当客户端在 `hosts()` 方法中仅指定一台主机时，模式将设置为 `ClusterConnectionMode.SINGLE` 而非 `ClusterConnectionMode.MULTIPLE`。这导致客户端忽略读取首选项而仅连接到按 `hosts()`

配置的服务器。因此，即使客户端设置如下所示初始化，所有读取仍将转至主服务器而不是辅助服务器。

```
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);
```

失效转移案例

使用上述的客户端连接设置，如果集群写入器端点存在失效转移和延迟的 DNS 记录更新，则客户端仍尝试向旧写入器（失效转移后现在是读取器）发出写入请求。这导致 Java 驱动程序未恰当处理的服务器端错误（不是主错误）（这仍在调查中）。因此，可能听任客户端处于不良状态，例如直到应用程序服务器重新启动为止。

对此有两种应变方法：

- 通过连接字符串连接到 Amazon DocumentDB 的客户端将不会出现这个问题，因为设置读取首选项时`ClusterConnectionMode` 会设置成 `MULTIPLE`。

  ```
  MongoClientURI mongoClientURI = new MongoClientURI("mongodb://usr:pass:cluster-
  endpoint:27317/test?ssl=false&replicaSet=rs0&readpreference=secondaryPreferred");
  MongoClient mongoClient = MongoClients.create(mongoClientURI.getURI());
  ```

  或者配合 `applyConnectionString` 方法使用 `MongoClientSettings` 生成器。

  ```
  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);
```

- 将 ClusterConnectionMode 明确设置成 MULTIPLE。只有使用 applyToClusterSettings 和 hosts().size() == 1 时才需要这样做。

```
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);
```

测试与 Amazon DocumentDB 实例的连接

您可以使用常见的 Linux 或 Windows 工具测试与集群的连接。

从 Linux 或 Unix 终端测试此连接，方法为输入以下命令（将 cluster-endpoint 替换为终端节点并将 port 替换为实例的端口）。

```
nc -zv cluster-endpoint port
```

以下是示例操作和返回值的示例：

```
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!
```

连接到无效终端节点

在连接到 Amazon DocumentDB 集群并且使用的是无效的集群端点时，将出现与以下内容类似的错误。

```
mongo --ssl \
   --host sample-cluster.node.us-east-1.docdb.amazonaws.com:27017 \
   --sslCAFile global-bundle.pem \
   --username <user-name> \
   --password <password>
```

该输出类似于以下示例：

```
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
```

要获取集群的有效终端节点，请运行以下命令：

```
aws docdb describe-db-clusters \
   --db-cluster-identifier sample-cluster \
   --query 'DBClusters[*].[Endpoint,Port]'
```

要获取实例的有效终端节点，请运行以下命令：

```
aws docdb describe-db-instances \
   --db-instance-identifier sample-instance \
   --query 'DBInstances[*].[Endpoint.Address,Endpoint.Port]'
```

有关更多信息，请参阅 了解 Amazon DocumentDB 端点。

影响连接数的驱动程序配置

使用客户端驱动程序连接到 Amazon DocumentDB 集群时，请务必考虑 `maxPoolSize` 配置参数。`maxPoolSize` 设置决定了客户端驱动程序可以在其连接池中保持的最大连接数。

索引故障排除

以下主题说明了索引或后台索引构建失败时的操作。

主题

- 索引构建失败
- 后台索引构建延迟问题和失败
- 数据库索引膨胀

索引构建失败

在索引创建过程中，Amazon DocumentDB 将使用实例上的本地存储。您可以使用FreeLocalStorage CloudWatch 指标 (`CloudWatch -> Metrics -> DocDB -> Instance Metrics`) 监控此磁盘使用情况。如果索引构建用尽所有本地磁盘并且失败，您将收到错误。将数据迁移到 Amazon DocumentDB 时，我们建议您先创建索引，然后插入数据。有关迁移策略和创建索引的更多信息，请参阅 Amazon DocumentDB 文档中的 迁移到 Amazon DocumentDB 和博客文章：使用离线方法从 MongoDB 迁移到 Amazon DocumentDB。

在现有集群上创建索引时，如果索引构建花费的时间超过预期或构建失败，我们建议您扩展实例来创建索引，创建好索引后，再缩减回来。Amazon DocumentDB 使您能够使用 AWS 管理控制台 或在几分钟内快速扩展实例大小。 AWS CLI有关更多信息，请参阅 管理实例类。使用每秒实例定价时，您只需为使用 1 秒以上的资源付费。

后台索引构建延迟问题和失败

Amazon DocumentDB 中后台索引构建直到索引构建发起之前启动的主实例上所有查询都完成执行后才启动。如果有长时间运行的查询，则后台索引构建将阻塞，直到查询完成为止，因此可能耗时比预期完成时间更长。即使集合为空也如此。

前台索引构建并不展现出相同的阻塞行为。取而代之，前台索引构建对集合进行独占锁定，直至完成索引构建为止。因此，要在空集合上创建索引并避免在任何长时间运行的查询上阻塞，我们建议使用前台索引构建。

> **Note**
>
> 在任何给定时间，Amazon DocumentDB 只允许在一个集合中构建一个后台索引。如果后台索引构建期间在相同集合上出现 DDL（数据定义语言）操作（如 `createIndex()` 或 `dropIndex()`），后台索引构建将失败。

数据库索引膨胀

Amazon DocumentDB 使用 Multi-Version 并发控制 (MVCC) 来管理并发交易。删除或更新文档后，其先前版本将作为“失效”版本保留在集合和索引中。垃圾回收过程会自动从这些失效版本中回收空间以用于未来的操作。

当集合的索引因失效或过时的索引条目累积或页面内的碎片累积而变大时，就会出现索引膨胀。报告的百分比表示未来索引条目可以使用的索引空间量。此膨胀会同时消耗缓冲区缓存和存储空间中的空间。如果要消除膨胀，则需要重建索引。

Example示例

运行以下命令以确定索引未使用的存储空间：

```
db.coll.aggregate({$indexStats:{}});
```

结果类似如下：

```
{
    "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
    }
}
```

使用 reIndex 命令无需停机即可重建索引，该命令需要扫描整个集合。请参阅使用 re Index 维护索引。

性能和资源利用率故障排除

本部分提供了 Amazon DocumentDB 部署中的常见诊断问题和解决方案。提供的示例使用 mongo Shell，并限制在单个实例范围内。要查找实例终端节点，请参阅了解 Amazon DocumentDB 端点。

主题

- 如何确定通过 Mongo API 对我的集合执行的插入、更新和删除操作的次数？
- 如何分析缓存性能？
- 如何查找和终止长时间运行或受阻的查询？
- 如何查看查询计划和优化查询？
- 如何在弹性集群中查看查询计划？
- 如何列出实例上所有正在运行的操作？
- 如何知道查询何时取得进展？
- 如何确定系统突然运行缓慢的原因？
- 如何确定一个或多个集群实例上 CPU 使用率过高的原因？
- 如何确定实例上打开的光标？
- 如何确定当前的 Amazon DocumentDB 引擎版本？
- 如何分析索引使用情况并识别未使用的索引？
- 如何识别缺失的索引？
- 如何确定数据库集合膨胀？
- 有用查询的摘要

如何确定通过 Mongo API 对我的集合执行的插入、更新和删除操作的次数？

要查看对某个集合执行的插入、更新和删除操作的数量，请对该集合运行以下命令：

```
db.collection.stats()
```

此命令的输出在其 opCounters 字段下描述了以下内容：

- num DocsIns-插入到此集合中的文档数量。这包括使用 insert 和 insertMany 命令插入的文档，以及通过更新插入的文档。
- num DocsUpd-此集合中更新的文档数量。这包括使用 update 和 findAndModify 命令更新的文档。
- num DocsDel-从该集合中删除的文档数。这包括使用deleteOne、deleteMany、remove、和 findAndModify 命令删除的文档。
- lastReset：上次重置这些计数器的时间。当集群或扩展 up/down 实例时 starting/stopping ，此命令提供的统计数据将被重置。

运行 db.collection.stats() 的输出示例如下。

```
{
    "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)
}
```

通过 Mongo API 查看用于插入、更新和删除操作的集合专用计数器时，应使用此 stats 命令。另一种查看特定于集合的操作计数器的方法是启用 DML 审核。可以在 使用以下方式监控亚马逊 DocumentDB CloudWatch 中查看在一分钟时间间隔内对所有集合执行的插入、更新和删除操作的次数。

如何分析缓存性能？

分析缓存性能可以深入了解数据检索的效率和系统性能，并且基于从磁盘读取的数据量与从缓存中读取的数据量。我们提供有关缓存命中次数（从缓存中读取的数据）和缓存未命中（在缓存中找不到并从磁盘读取的数据）的缓存统计信息，以便深入了解缓存性能。通过对该集合运行以下命令可以找到特定集合的缓存统计信息：

```
db.collection.stats()
```

此命令输出中 cacheStats 字段中的值提供集合的缓存统计信息，以及在集合上创建的索引的缓存总统计信息。以下列出了这些统计数据：

- **collBlksHit**：操作此集合期间从缓存中读取的块数。
- **collBlksRead**：操作此集合期间从磁盘读取的块数（缓存未命中）。
- **collHitRatio**：此集合的缓存命中率 (`100 * [collBlksHit / (collBlksHit + collBlksRead)]`)。
- **idxBlksHit**：从缓存中读取在此集合上创建的任何索引的块数。
- **idxBlksRead**：在此集合上创建的任何索引从磁盘读取的块数（缓存未命中）。
- **idxHitRatio**：在此集合上创建的索引的缓存命中率 (`100 * [idxBlksHit / (idxBlksHit + idxBlksRead)]`)。
- **lastReset**：上次重置这些统计数据的时间。集群或扩展 up/down 实例时 starting/stopping ，提供的统计数据会`db.collection.stats()`被重置。

也可以使用 indexStats 命令查看每个索引的 idxBlksHit 和 idxBlksRead 字段的细分。运行以下命令可以找到特定于索引的缓存统计信息：

```
db.collection.aggregate([{$indexStats:{}}]).pretty()
```

对于每个索引，可以在 cacheStats 字段下找到以下缓存统计信息：

- **blksHit**：从缓存中读取此索引的数据块的数量。
- **blksRead**：从磁盘中读取此索引的数据块的数量。
- **blksHitRatio**：缓存命中率四舍五入到小数点后四位，计算公式为 100 * [blksHit / (blksHit + blksRead)]。

如何查找和终止长时间运行或受阻的查询？

用户查询可能因查询计划不够理想而运行缓慢，或者由于资源争用而受阻。

要查找因查询计划不够理想而速度缓慢的长时间运行的查询，或者由于资源争用而受阻的查询，请使用 currentOp 命令。可以筛选该命令以帮助缩小要终止的相关查询的列表。长时间运行的查询必须拥有关联的 opid，才能够终止查询。

以下查询使用 currentOp 命令列出受阻或运行时间超过 10 秒的所有查询。

```
db.adminCommand({
    aggregate: 1,
    pipeline: [
        {$currentOp: {}},
        {$match:
            {$or: [
                {secs_running: {$gt: 10}},
                {WaitState: {$exists: true}}]}},
        {$project: {_id:0, opid: 1, secs_running: 1}}],
    cursor: {}
});
```

接下来，您可以缩小查询，以查找运行时间超过 10 秒的查询的 opid 并终止它。

查找并终止运行时间超过 10 秒的查询

1. 查找查询的 opid。

```
db.adminCommand({
    aggregate: 1,
    pipeline: [
        {$currentOp: {}},
        {$match:
            {$or:
                [{secs_running: {$gt: 10}},
                 {WaitState: {$exists: true}}]}}],
    cursor: {}
});
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "waitedMS" : NumberLong(0),
    "cursor" : {
        "firstBatch" : [
            {
                "opid" : 24646,
                "secs_running" : 12
            }
        ],
        "id" : NumberLong(0),
        "ns" : "admin.$cmd"
    },
    "ok" : 1
}
```

2. 使用 killOp 操作终止查询。

```
db.adminCommand({killOp: 1, op: 24646});
```

如何查看查询计划和优化查询？

如果查询运行缓慢，可能是因为查询执行需要对集合进行完全扫描以选择相关的文档。有时，可通过创建合适的索引提高查询的运行速度。要检测此情形并确定要在其中创建索引的字段，请使用 explain 命令。

> **Note**
>
> Amazon DocumentDB 在利用分布式、容错、自修复的存储系统的专用数据库引擎上模拟 MongoDB 3.6 API。因此，查询计划和 `explain()` 的输出在 Amazon DocumentDB 和 MongoDB 之间可能有所不同。希望控制其查询计划的客户可以使用 `$hint` 运算符强制选择首选索引。

在 `explain` 命令下运行要改进的查询，如下所示。

```
db.runCommand({explain: {<query document>}})
```

以下是操作示例。

```
db.runCommand({explain:{
    aggregate: "sample-document",
    pipeline: [{$match: {x: {$eq: 1}}}],
    cursor: {batchSize: 1}}
});
```

此操作的输出将类似于下文（JSON 格式）。

```
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "db.test",
        "winningPlan" : {
            "stage" : "COLLSCAN"
        }
    },
    "serverInfo" : {
        "host" : "...",
        "port" : ...,
        "version" : "..."
    },
    "ok" : 1
}
```

上述输出表明，$match 阶段要求扫描整个集合并检查每个文档中的字段 "x" 是否等于 1。如果集合中有很多文档，集合扫描将非常慢，因此整体查询性能非常低。因此，explain 命令输出中的 "COLLSCAN" 的存在表明，可以通过创建合适的索引来提高查询性能。

在本例中，查询检查所有文档中的字段 "x" 是否等于 1。因此，在字段 "x" 上创建索引，可使查询避免对集合进行完全扫描，并可使用索引更快地返回相关文档。

在字段 "x" 上创建索引后，explain 输出如下所示。

```
{
    "queryPlanner" : {
         "plannerVersion" : 1,
         "namespace" : "db.test",
         "winningPlan" : {
             "stage" : "IXSCAN",
             "indexName" : "x_1",
             "direction" : "forward"
         }
    },
    "serverInfo" : {
        "host" : "...",
        "port" : ...,
        "version" : "..."
    },
    "ok" : 1
}
```

因此，在 "x" 字段上创建索引后，$match 阶段即可使用索引扫描来减少必须对其评估 "x = 1" 谓词的文档的数量。

对于较小的集合，如果性能增益微乎其微，Amazon DocumentDB 查询处理器可以选择不使用索引。

如何在弹性集群中查看查询计划？

要检查弹性集群中的查询计划，请使用 explain 命令。以下是针对分片集合的查找查询的 explain 操作示例：

```
db.runCommand(
   {
     explain: { find: "cities", filter: {"name": "Seoul"}}
   }
```

```
)
```

> **Note**
>
> Amazon DocumentDB 在专用数据库引擎上模拟 MongoDB。因此，查询计划和 `explain()` 的输出在 Amazon DocumentDB 和 MongoDB 之间可能有所不同。您可以使用 `$hint` 运算符强制选择首选索引来控制查询计划。

此操作的输出将可能类似于下文（JSON 格式）。

```
{
  "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)
}
```

前面的输出显示了三分片集群上 find 查询的查询计划。每个分片都有多个数据分区，这些分区可以有不同的输入阶段。在此示例中，在每个分片的“PARTITION_MERGE”阶段合并结果之前，将在所有分区上运行“COLLSCAN”（集合扫描）。然后，分片上的结果在“SHARD_MERGE”阶段合并在一起，再发送回客户端。

如何列出实例上所有正在运行的操作？

作为用户或主用户，您经常需要列出实例上当前正在运行的所有操作，以进行诊断和故障排除。（有关管理用户的信息，请参阅管理 Amazon DocumentDB 用户）。

借助 mongo Shell，您可以使用以下查询列出 Amazon DocumentDB 实例上正在运行的所有操作。

```
db.adminCommand({currentOp: 1, $all: 1});
```

该查询返回当前在实例上运行的所有用户查询和内部系统任务的完整列表。

此操作的输出将类似于下文（JSON 格式）。

```
{
    "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" 字段的有效值如下所示：

- **INTERNAL**：内部系统任务，如游标清理或过时用户清理任务。
- **TTLMonitor**：生存时间 (TTL) 监视器线程。其运行状态在 "active" 字段中反映。
- **GARBAGE_COLLECTION**：内部垃圾收集器线程。
- **CONN**：用户查询。
- **CURSOR**：该操作是一个空闲的光标，等待用户调用“getMore”命令以获取下一批结果。在此状态下，光标正在消耗内存，但不消耗任何计算。

上述输出还列出了在系统中运行的所有用户查询。每个用户查询都在数据库和集合的上下文中运行，而这二者的并集称为命名空间。每个用户查询的命名空间都可在 "ns" 字段中获得。

有时，您需要列出在特定命名空间中运行的所有用户查询。因此，必须在 "ns" 字段上过滤之前的输出。下面是一个实现要筛选的输出的示例查询。该查询列出当前在数据库 "db" 和集合 "test"（即 "db.test" 命名空间）中运行的所有用户查询。

```
db.adminCommand({aggregate: 1,
    pipeline: [{$currentOp: {allUsers: true, idleConnections: true}},
               {$match: {ns: {$eq: "db.test"}}}],
    cursor: {}
});
```

作为系统主用户，您可以查看所有用户的查询以及所有内部系统任务。所有其他用户只能查看其各自的查询。

如果查询和内部系统任务的总数超过默认批处理游标大小，则 mongo Shell 将自动生成迭代器对象 'it' 以查看其余结果。可继续执行 'it' 命令，直到查看完所有结果。

如何知道查询何时取得进展？

用户查询可能由于不够理想的查询计划而运行缓慢，或可能由于资源争用而受阻。调试此类查询是一个多步骤过程，可能需要多次执行相同的步骤。

调试的第一步是列出长时间运行或受阻的所有查询。以下查询列出了运行时间超过 10 秒或正在等待资源的所有用户查询。

```
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: {}
                });
```

定期重复上述查询以确定查询列表更改并识别长时间运行或受阻的查询。

如果相关的查询的输出文档具有 WaitState 字段，则表示资源争用是查询运行缓慢或受阻的原因。资源争用可能是由于 I/O内部系统任务或其他用户查询造成的。

此操作的输出将类似于下文（JSON 格式）。

```
{
    "waitedMS" : NumberLong(0),
    "cursor" : {
        "firstBatch" : [
            {
                "opid" : 201,
                "command" : {
                    "aggregate" : ...
                },
                "secs_running" : 208,
                "WaitState" : "IO"
            }
        ],
        "id" : NumberLong(0),
        "ns" : "admin.$cmd"
    },
    "ok" : 1
}
```

I/O 如果跨不同集合的许多查询在同一个实例上同时运行，或者实例大小对于正在运行查询的数据集来说太小，则可能成为瓶颈。如果查询是只读查询，您可以通过在不同副本中分隔每个集合的查询来缓解上述情况。对于跨不同集合的并发更新，或者当实例大小对数据集而言过小时，您可以通过向上扩展实例来进行缓解。

如果资源争用是由其他用户查询引起的，则输出文档中的 "blockedOn" 字段将具有影响此查询的查询的 "opid"。利用 "opid"，可沿着所有查询的 "WaitState" 和 "blockedOn" 字段的链查找位于链头的查询。

如果链头的任务是一个内部任务，那么在这种情况下的唯一缓解措施将是，终止查询之后重新运行它。

下面是一个示例输出，其中的查找查询在由另一个任务拥有的集合锁上被阻止。

```
{
    "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
}
```

如果 "WaitState" 具有值 "Latch"、"SystemLock"、"BufferLock"、"BackgroundActivity" 或 "Other"，则资源争

用的来源为内部系统任务。如果这种情况持续了很长时间，则唯一的缓解措施将是，终止查询并在之后重新运行。

如何确定系统突然运行缓慢的原因？

以下是系统速度减慢的一些常见原因：

- 并发查询之间的资源争用过多
- 活动并发查询的数量会随着时间的推移而增加
- 内部系统任务，例如 "GARBAGE_COLLECTION"

要监控系统的长期使用情况，请定期运行以下 "currentOp" 查询并将结果输出到外部存储。此查询会计算系统中每个命名空间中的查询和操作数。您可以分析系统使用情况的结果以了解系统上的负载并做出适当的决策。

```
db.adminCommand({aggregate: 1,
                 pipeline: [{$currentOp: {allUsers: true, idleConnections: true}},
                            {$group: {_id: {desc: "$desc", ns: "$ns", WaitState:
 "$WaitState"}, count: {$sum: 1}}}],
                 cursor: {}
                });
```

此查询将返回在每个命名空间中运行的所有查询和所有内部系统任务的聚合，以及每个命名空间的等待状态（如果有）的数量。

此操作的输出将类似于下文（JSON 格式）。

```
{
    "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
}
```

在上述输出中，命名空间 "db.test" 中有两个用户查询在集合锁上被阻止，命名空间 "admin.$cmd" 中有一个查询，另外还有一个内部 "TTLMonitor" 任务。

如果输出表明有许多查询处于受阻等待状态，请参阅如何查找和终止长时间运行或受阻的查询？

如何确定一个或多个集群实例上 CPU 使用率过高的原因？

以下各节可以帮助您确定实例 CPU 使用率过高的原因。根据工作负载，您的结果可能会有所不同。

- 要确定实例突然运行缓慢的原因，请参阅如何确定系统突然运行缓慢的原因？
- 要确定并终止特定实例上长时间运行的查询，请参阅如何查找和终止长时间运行或受阻的查询？
- 要了解查询是否取得进展，请参阅如何知道查询何时取得进展？
- 要确定查询长时间运行的原因，请参阅如何查看查询计划和优化查询？
- 要跟踪一段时间内长时间运行的查询，请参阅 分析 Amazon DocumentDB 操作。

根据实例 CPU 使用率高的原因，执行以下一项或多项操作可能会有所帮助。

- 如果主实例的 CPU 利用率较高，但副本实例没有，请考虑通过客户端读取首选项设置（例如 secondaryPreferred）在副本之间分配读取流量。有关更多信息，请参阅 作为副本集连接到 Amazon DocumentDB。

使用副本进行读取可以让主实例处理更多的写入流量，从而更好地利用集群的资源。从副本中读取的数据最终是一致的。

- 如果写入工作负载导致高 CPU 使用率，则将集群实例的大小更改为更大的实例类型会增加可用于为该工作负载提供服务的 CPU 内核数量。有关更多信息，请参阅管理 Amazon DocumentDB 实例和实例类规格。
- 如果所有集群实例都表现出较高的 CPU 利用率，并且工作负载使用副本进行读取，则向集群添加更多副本会增加可用于读取流量的资源。有关更多信息，请参阅 向集群添加 Amazon DocumentDB 实例。

如何确定实例上打开的光标？

连接到 Amazon DocumentDB 实例后，您可以使用命令 `db.runCommand("listCursors")` 列出该实例上打开的光标。在给定的 Amazon DocumentDB 实例上，在任何给定时间最多只能打开 4,560 个活动光标，具体取决于实例类型。通常建议关闭不再使用的游标，因为游标占用实例上的资源并具有上限。有关具体限制，请参阅 Amazon DocumentDB 配额和限制。

```
db.runCommand("listCursors")
```

如何确定当前的 Amazon DocumentDB 引擎版本？

要确定当前的 Amazon DocumentDB 引擎版本，请运行以下命令。

```
db.runCommand({getEngineVersion: 1})
```

此操作的输出将类似于下文（JSON 格式）。

```
{ "engineVersion" : "2.x.x", "ok" : 1 }
```

> **Note**
>
> 亚马逊 DocumentDB 3.6 的引擎版本为 1.x.x，亚马逊 DocumentDB 4.0 的引擎版本为 2.x.x，亚马逊 DocumentDB 5.0 的引擎版本为 3.x.x，亚马逊 DocumentDB 8.0 的引擎版本为 4.x.x。

如何分析索引使用情况并识别未使用的索引？

要识别给定集合的索引，请运行以下命令：

```
db.collection.getIndexes()
```

要分析在对集合执行操作期间使用了多少索引，可以使用 `collStats` 和 `indexStats` 命令。要查看使用索引（索引扫描）执行的扫描总数与不使用索引执行的扫描数（集合扫描）的对比，请运行以下命令：

```
db.collection.stats()
```

此命令的输出包括以下值：

- **`idxScans`**：使用索引对该集合执行的扫描次数。
- **`collScans`**：不使用索引对此集合执行的扫描次数。这些扫描将涉及逐一查看馆藏中的文档。
- **`lastReset`**：上次重置这些计数器的时间。当集群或扩展 up/down 实例时 starting/stopping ，此命令提供的统计数据将被重置。

在以下命令的输出中可以找到每个索引的使用量明细。为了提高性能和降低成本，最佳做法是定期识别和删除未使用的索引，因为这样可以省去不必要的计算、存储，并 I/Os 用于维护索引。

```
db.collection.aggregate([{$indexStats:{}}]).pretty()
```

此命令的输出给出了在集合上创建的每个索引的以下值：

- **`ops`**：使用索引的操作数。如果您的工作负载已经运行了足够长的时间，并且您确信自己的工作负载处于稳定状态，则 `ops` 值为零表示根本没有使用索引。
- **`numDocsRead`**：使用此索引进行操作期间读取的文档数。
- **`since`**：自 Amazon DocumentDB 开始收集索引使用统计数据以来的时间，该值通常是自上次数据库重启或维护操作以来的值。
- **`size`**：索引的大小以字节为单位。

下面的示例是从以上命令中的样本输出。

```
{
    "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" : ...
    }
}
```

要确定集合的整体索引大小，请运行以下命令：

```
db.collection.stats()
```

要删除未使用的索引，请运行以下命令：

```
db.collection.dropIndex("indexName")
```

如何识别缺失的索引？

您可以使用 Amazon DocumentDB 分析器来记录慢速查询。在慢速查询日志中反复出现的查询可能表示需要额外的索引才能提高该查询的性能。

您可以通过寻找具有一个或多个阶段至少执行一个 COLLSCAN 阶段的长时间运行的查询来发现有用索引的机会，这意味着查询阶段必须读取集合中的每个文档才能对查询提供响应。

以下示例显示了对在大型集合上运行的出租车乘车集合的查询。

```
db.rides.count({"fare.totalAmount":{$gt:10.0}}))
```

为了执行此示例，查询必须执行集合扫描（即读取集合中的每个文档），因为 `fare.totalAmount` 字段上没有索引。此查询的 Amazon DocumentDB 分析器输出如下所示：

```
{
    ...
    "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"
}
```

为了加快本示例中的查询速度，您需要在 `fare.totalAmount` 上创建索引，如下所示。

```
db.rides.createIndex( {"fare.totalAmount": 1}, {background: true} )
```

> **Note**
>
> 在前台创建的索引（即如果在创建索引时未提供 {background:true} 选项）采用独占写入锁，这可以防止应用程序在索引构建完成之前将数据写入集合。在生产集群上创建索引时，请注意这种潜在影响。创建索引时，我们建议设置 {background:true}。

通常，您希望在具有高基数的字段（例如，大量唯一值）上创建索引。在基数较低的字段上创建索引可能会导致不使用的索引过大。Amazon DocumentDB 查询优化器在创建查询计划时会考虑集合的总体大小和索引的选择性。有时候，即使存在索引，您也会看到查询处理器选择一个 COLLSCAN。当查询处理器估计使用索引不会比扫描整个集合更具性能优势时，就会发生这种情况。如果要强制查询处理器使用特定的索引，则可以使用如下所示的 hint() 运算符。

```
db.collection.find().hint("indexName")
```

如何确定数据库集合膨胀？

当集合的大小因失效或过时的文档累积或数据库页面内的碎片累积而变大时，就会出现集合膨胀。报告的百分比表示未来文档可以使用的文档空间量。此膨胀会同时消耗缓冲区缓存和存储空间中的空间。要消除膨胀，必须在维护时段内通过 dump/restore 或使用迁移环回和切换来重新加载集合。

Example示例

运行以下命令以确定集合未使用的存储空间：

```
db.runCommand({collStats:'coll'})
```

结果类似如下：

```
{
        "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)
}
```

有用查询的摘要

以下查询可用于监控 Amazon DocumentDB 中的性能和资源利用率。

- 使用以下命令查看有关特定集合的统计信息，包括操作计数器、缓存统计信息、访问统计信息和大小统计信息：

```
db.collection.stats()
```

- 使用以下命令查看有关在集合上创建的每个索引的统计信息，包括索引的大小、索引特定的缓存统计信息和索引使用情况统计信息：

```
db.collection.aggregate([{$indexStats:{}}]).pretty()
```

- 使用以下查询列出所有活动。

```
db.adminCommand({currentOp: 1, $all: 1});
```

- 以下代码列出了所有长时间运行或已阻止的查询。

```
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: {}
                });
```

- 以下代码终止查询。

```
db.adminCommand({killOp: 1, op: <opid of running or blocked query>});
```

- 使用以下代码获取系统状态的汇总视图。

```
db.adminCommand({aggregate: 1,
                 pipeline: [{$currentOp: {allUsers: true, idleConnections: true}},
                            {$group: {_id: {desc: "$desc", ns: "$ns", WaitState:
 "$WaitState"}, count: {$sum: 1}}}],
                 cursor: {}
                });
```

Amazon DocumentDB 中的垃圾回收

Amazon DocumentDB 实现了多版本并发控制（MVCC）数据库架构，可为每次更新操作创建新版本的文档和索引条目。该架构提供了事务隔离，防止一个事务的更改出现在另一个事务中。

主题

- 了解 Amazon DocumentDB 中的垃圾收集
- 垃圾回收过程

- 存储架构和扩展存储
- 监控垃圾回收
- collStats 输出示例
- 常见问题

了解 Amazon DocumentDB 中的垃圾收集

垃圾回收（GC）是一个自动化后台过程，可在 Amazon DocumentDB 中维持最佳的系统性能和可用性。与许多现代数据库一样，Amazon DocumentDB 的 MVCC 架构每次更新都会创建新的文档和索引版本。每个写入操作都会使用有限计数器中唯一的 MVCC ID。这些 ID 标识文档版本属于哪个事务，以及该交易是已提交还是已回退。随着时间的推移，这些旧版本及其 MVCC ID 会不断累积，需要进行清理以防止性能下降。

垃圾回收的功能

垃圾回收器有三个基本功能：

- 回收存储空间：将移除活动查询不再需要的过时文档和索引版本，从而为将来的写入操作释放空间。
- 防止 MVCC ID 溢出：通过管理 MVCC ID 的有限计数器来防止 MVCC ID 溢出。如果没有这种管理，计数器最终将达到其限制，从而迫使数据库进入临时只读模式，直至 ID 被回收利用。
- 维持查询性能：通过消除失效的文档版本来维持最佳的查询性能，否则这些文档版本会累积并拖慢查询处理速度。

垃圾回收过程

垃圾回收过程对每个集合进行操作，并且可以在不同的集合上并发运行多个过程。此过程包括四个连续阶段：

1. 识别：系统识别活动事务或查询不再引用的文档和索引版本。
2. 内存加载：如果旧文档和索引条目尚未存在于内存中，则会将其加载到内存中。
3. 删除：永久删除过时版本以回收存储空间。
4. MVCC ID 回收：系统将从已删除的版本中回收 MVCC ID 以用于新操作。

当垃圾回收完成对旧文档版本的处理后，会从系统中移除最旧的 MVCC ID。这种清理对于通过回收 MVCC ID 来防止 MVCC ID 溢出至关重要，使 MVCC ID 可用于整个集群中的新写入操作。若无此回收过程，系统最终将耗尽其有限的 MVCC ID 计数器并进入只读状态。

垃圾回收调度

垃圾回收会定期在后台自动运行。时间和频率会根据系统负载、可用资源、写入量和 MVCC ID 消耗水平进行动态调整。在高写入活动期间，垃圾回收过程将更频繁地执行以管理数量激增的文档版本。

存储架构和扩展存储

Amazon DocumentDB 使用复杂的存储架构，将文档存储分为两个不同的部分：

基础存储段

基本存储段包含主要文档数据和元数据。此区段存储：

- 符合标准页面大小 (8 KB) 的文档内容。
- 文档元数据和结构信息。
- 主索引及其条目。
- Collection-level 统计和配置。

扩展存储段

扩展存储段使用专门的大型文档对象存储，用于处理超过标准存储页面大小的文档。本部分提供：

- 高效的大型文档处理 — 大于基本存储阈值的文档会自动移至扩展存储段。
- 优化的存储布局 — 该分段使用针对大型对象进行了优化的不同存储格式，从而减少了碎片并改善了访问模式。
- 独立垃圾收集 — 扩展存储段有自己的垃圾收集过程，该过程可以独立于基本存储清理运行。
- 透明访问 — 应用程序可以无缝访问大型文档，而无需知道哪个存储段包含数据。

扩展存储区段对以下方面特别有利：

- 包含大型嵌入式数组的文档的集合。
- 具有大量嵌套结构的文档。
- 存储二进制数据或大型文本字段的集合。
- 具有混合文档大小的应用程序，其中一些文档大大超过平均大小。

监控垃圾回收

集群级别指标

AvailableMVCCIds

- 地点-亚马逊 CloudWatch
- 描述-一个计数器，显示从上限 18 亿个起可用的剩余写入操作数。当此计数器达到零时，您的集群将进入只读模式，直至 ID 被回收并循环利用。计数器会在每次写入操作后递减，并在垃圾回收利用旧的 MVCC ID 后递增。
- 建议：当值低于 13 亿时设置警报。此预警可让您采取稍后讨论的建议步骤。

LongestActiveGCRuntime

- 地点-亚马逊 CloudWatch
- 描述：最长活动垃圾回收过程的持续时间（以秒为单位）。每分钟更新一次，仅跟踪活动操作，不包括在一分钟时段内完成的进程。
- 建议-与`gcRuntimeStats`历史数据进行比较，以识别异常的垃圾收集行为，例如批量删除期间的运行时间延长。

回收级别指标

MVCCIDStats: MVCCIdScale

- 位置：数据库 collStats 命令
- 描述：以 0 到 1 的等级衡量 MVCC ID 使用期限，其中 1 表示集群进入只读状态前的最长使用期限。将此指标与 `AvailableMVCCIds` 结合使用，以识别包含使集群老化的最旧 MVCC ID 的集合。
- 建议：将每个集合的值维持在 0.3 以下。

gcRuntimeStats

- 位置：数据库 collStats 命令
- 描述：提供两个月的垃圾回收指标历史记录，包括总运行次数、平均持续时间和最长持续时间。仅包括持续时间超过五分钟的垃圾回收操作，以确保有意义的统计数据。

storageSizeStats

- 位置：数据库 collStats 命令
- 描述 — 提供不同存储段的存储利用率的详细明细：
  - `storageSegmentBase`— 基础存储段用于存储标准文档的存储空间
  - `storageSegmentExtended`— 扩展存储段用于存储大型文档的存储空间
- 用法-帮助识别具有大量文件存储空间的馆藏，并了解存储分配模式。

`unusedStorageSize`（集合级别）

- 位置：数据库 collStats 命令
- 描述：根据抽样统计数据估算集合中未使用的存储空间。包括已删除文档和空分段所占用的空间。该指标同时提供合并总数和每个区段的细分：
  - 合并`unusedBytes`并`unusedPercent`跨所有存储段
  - `storageSegmentBase`— 未使用的空间，特别是基础存储段中的未使用空间
  - `storageSegmentExtended`— 未使用的空间，特别是在扩展存储段中

`documentFragmentStats`

- 位置：数据库 collStats 命令
- 描述-提供有关文档片段和集合中无效数据的详细信息。文档片段代表数据库引擎使用的内部存储单元，而失效的碎片表示无法再访问但尚未被回收的数据。该指标包括：
  - `totalDocFragmentsCount`— 馆藏中的文档片段总数
  - `deadDocFragmentsCount`— 包含失效（无法访问）数据的片段数
  - `deadDocFragmentsPercent`— 包含失效数据的片段的百分比
  - `deadDocFragmentBytes`— 失效文档片段消耗的估计字节数
  - Per-segment `storageSegmentBase`和的细分 `storageSegmentExtended`
- 使用情况-监控此指标以了解垃圾收集的有效性，并确定可能从维护操作中受益的收集。高百分比的死碎片表明垃圾收集可能落后，或者垃圾收集将从优化中受益。

索引级别指标

`unusedStorageSize`（索引级别）

- 位置：数据库 indexStats 命令

- 描述：根据抽样统计数据估算索引中未使用的存储空间。包括过时的索引条目和空分段所占用的空间。
- 建议：使用 reIndex 命令在不停机的情况下重建索引并回收未使用的空间。有关更多详细信息，请参阅“管理索引”。

collStats 输出示例

以下示例显示了包含垃圾收集和存储指标的典型collStats输出：

```
{
    "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)
}
```

常见问题

如何识别垃圾回收效率低下？

监控以下表明垃圾回收效率低下的警告信号：

- 集合膨胀过多 — 在大量写入或批量删除期间，unusedStorageSize指标会稳步增加，尤其是在索引较大的情况下。
- 高死亡碎片百分比 — documentFragmentStats 显示的deadDocFragmentsPercent值一直很高（高于 10-15%）。
- 查询延迟降低-由于累积的失效文档导致查询延迟增加。
- GC 持续时间延长 — 垃圾收集操作花费的时间比历史平均值长gcRuntimeStats。
- GC 处理能力提升 — 高LongestActiveGCRuntime表示垃圾收集器无法满足系统需求。

垃圾回收会影响我的数据库性能吗？

正常情况下，垃圾回收对性能的影响微乎其微。但是，当垃圾回收滞后时，您可能会遇到：

- 积累的失效文档导致存储成本增加。
- 由于索引条目过时，查询性能降低。
- 如果 MVCC ID 已用完，则为临时只读模式。
- 在密集收集运行期间，资源使用量更高，尤其是在较小的实例上。
- 降低了大型文档的扩展存储段操作效率。

我是否可以手动触发垃圾回收？

不可以，无法手动触发 Amazon DocumentDB 中的垃圾回收。作为内部维护操作的一部分，系统会自动管理垃圾回收。

作为操作最佳实践，我应该设置哪些警报？

我们建议在集群级别和集合级别设置监控，以确保您的 Amazon DocumentDB 系统实现最佳性能。

要进行集群级别的监控，首先要为阈值为 13 亿的AvailableMVCCIds指标创建 Amazon CloudWatch 警报。这使您有足够的时间在指标达到零（届时您的集群将进入只读模式）之前采取措施。请记住，该指标可能会根据您的具体使用模式而波动——一些客户看到该指标降至13亿以下，然后在垃圾收集完成工作后恢复到15亿以上。

通过Amazon监控LongestActiveGCRuntime指标也很重要 CloudWatch。此指标与 gcRuntimeStats 一起可帮助您了解整个系统中垃圾回收的执行效率。

对于集合级别的监控，请重点关注以下关键指标：

- MVCCIdScale— 注意值是否不断增加，这些值表明 MVCC ID 正在老化，可能需要注意。
- gcRuntimeStats— 确定垃圾收集过程耗时异常长或持续多天。
- documentFragmentStats— 监控deadDocFragmentsPercent值——持续较高的百分比（高于 10-15%）可能表明垃圾收集落后。
- storageSizeStats以及 unusedStorageSize — 跟踪存储利用模式，识别两个存储段中存在大量未使用空间的馆藏。

需要特别注意具有频繁写入操作的集合，因为其会增加垃圾收集器的工作。对于具有大量写入活动的集合，我们建议更频繁地检查这些指标，以确保垃圾回收能跟上您的工作负载。

请注意，这些监控建议仅作为起点。随着您对系统行为越来越熟悉，您可能需要调整这些阈值以更好地匹配您的具体使用模式和要求。

如果我的**##MVCCID** 降至13亿以下，我该怎么办？

如果您的 AvailableMVCCIds指标降至 13 亿以下，我们建议您立即采取措施以防止集群进入只读模式。我们建议先扩展实例大小，为垃圾回收器提供更多计算资源。这是我们的主要建议，因为它允许您的应用程序继续正常运行，同时为垃圾收集器提供 catch 所需的额外功能。

如果仅靠扩展并不能改善状况，我们建议您考虑减少写入操作。使用 MVCCIdScale 指标识别哪些特定集合包含需要注意的较旧 MVCC ID。此外，还documentFragmentStats要进行监控以识别可能导致垃圾收集效率低下的死碎片百分比高的集合。

识别出这些集合后，可能需要暂时减少针对其的写入操作，以便垃圾回收能够跟上节奏。在恢复期间，我们建议您密切监控 AvailableMVCCIds 指标，以确保措施达到预期效果。AvailableMVCCIds 值恢复到 15 亿或更高后，您的集群就会被视为运行状况良好。

请记住，这些步骤属于预防性措施，旨在帮助系统在达到紧急状态前恢复。在看到指标降至 13 亿以下后，您越早采取措施，就越有可能避免对写入操作造成任何影响。

发布说明

这些发行说明描述了按主要引擎版本组织的 Amazon DocumentDB 功能、改进和错误修复。在每个版本部分中，条目按日期以表格格式列出。

有关引擎版本和补丁版本的编号方式（包括引擎补丁版本前缀与 Amazon DocumentDB 引擎版本之间的映射）的详细信息，请参阅。引擎版本号

您可以通过运行以下命令来确定当前的 Amazon DocumentDB 引擎修补版本：

```
db.runCommand({getEngineVersion: 1})
```

如果您的集群未使用最新版本的引擎，则很可能有待进行的维护可用来升级您的引擎。有关更多信息，请参阅开发人员指南中的 维护 Amazon DocumentDB。

主题

- 亚马逊 DocumentDB 引擎版本发行说明
- 亚马逊 DocumentDB 通用发行说明
- 亚马逊 DocumentDB 弹性集群发行说明

亚马逊 DocumentDB 引擎版本发行说明

按主要版本整理的 Amazon DocumentDB 引擎版本的发行说明。

Amazon DocumentDB 8.0

亚马逊 DocumentDB 引擎 8.0 版（兼容 MongoDB 8.0）的发行说明。

日期	发行说明
2026 年 5 月 13 日	新特征 亚马逊 DocumentDB 无服务器现已适用于亚马逊 DocumentDB 8.0。有关更多信息，请参阅 使用 Amazon DocumentDB 无服务器。
2026 年 5 月 6 日	错误修复和其他更改 亚马逊 DocumentDB 8.0（引擎版本：4.0.6940）

日期	发行说明
	此更新提供了一个可选的引擎补丁，其中包含以下改进： 1. 修复了垃圾收集中的一个错误，在极端情况下，该错误可能会导致读者崩溃。
2026 年 4 月 20 日	新特征 In-place 从亚马逊 DocumentDB 5.0 到亚马逊 DocumentDB 8.0 的主要版本升级 (MVU) 现已推出。有关更多信息，请参阅 Amazon DocumentDB 主版本就地升级。

日期	发行说明
2026 年 4 月 20 日	新功能和改进 亚马逊 DocumentDB 8.0（引擎版本：4.0.5995、4.0.6473） 这是必需的引擎补丁，包括以下新功能、改进和修复： 1. $sortArray 操作员在$project: • 支持在 Planner 版本 3 的聚合管道中$project处于阶段的$sortArray 运算符。该$sortArray 运算符根据数组的元素（对于标量值）或指定的字段（对于文档数组）对数组进行排序，并按升序（ ）或降序（1-1）返回排序后的数组。 2. 支持中的 parallel 无序批量更新。bulkWrite() 3. $geoNear使用 Planner v3 支持$out$collStats $indexStats 、、、。借助 Planner V3，该$out阶段还支持跨不同数据库写入集合。 4. 现在，即使$unwind$unwind阶段包含参数，聚合管道也会合并$lookup和分段。includeArrayIndex 以前，此优化在存在时会includeArrayIndex 被跳过。 5. 性能增强可提高 db.r8g.8xlarge 和在配置上运行的大型实例的写入吞吐量。 I/O-Optimized 6. 公开要监控Connections（active）的实例指标。这些限制可以在我们的 Amazon DocumentDB 配额和限制页面上找到。 7. 通过使用全局生存只读功能消除辅助集群重启，从而提高了全球集群的跨区域弹性。 8. 对聚合运算符的支持$dateToParts 和$dateFromParts 聚合运算符。 9. Command-level 现在，可在 CloudWatch CRUD 操作（find、、、、、、insert、findAndModify 、、update、delete、aggregate 、count、、distinct、commitTransaction ）getMoreabortTransaction 、公开和MaxConcurrent 维度中提供性能指标 AvgDuration P100Duration ，以改善操作监控。

日期	发行说明
	10.修复了输出文档中缺少包含布尔值`true/false`或数值1为 `0` / 的`newRoot`字段的`$replaceRoot` 行为错误。 11.`$indexOfArray` 修复了有效查询返回错误以表明查询无效的错误。 12.解释计划现在在`executionStats` 输出中包含`docsExamined` 指标。 13.提高了 SASL 身份验证性能。 14.当索引值保持不变且文档适合现有页面时，优化了文档更新，以减少存储开销。 15.Support fo `bypassDocumentValidation` r in `$merge`。 所有 Amazon DocumentDB 8.0 集群将在下一个维护时段内自动升级到此补丁版本。
2026 年 3 月 26 日	新特征 Amazon DocumentDB 现在支持最新一代内存优化实例 db.r8g，该实例由 G Arm-based AWS raviton4 处理器提供支持，性能比 R6G 实例提高多达 30%。有关更多信息，请参阅本指南管理实例类中的

日期	发行说明
2025 年 11 月 11 日	新特征 亚马逊 DocumentDB 现在提供与 MongoDB 8.0 的完全有线协议兼容性。Amazon DocumentDB 8.0 提高了查询性能，使您能够以更低的成本构建高性能应用程序。 亚马逊 DocumentDB 8.0（引擎补丁版本 4.0.4448） • 增加了对 MongoDB 8.0 API 驱动程序的支持。 • 增加了对 Planner Version3 的支持，该版本将性能改进扩展到聚合阶段运算符，并支持聚合管道优化和不同的命令。 • 通过 Zstandard 压缩算法增加了对基于字典的压缩的支持，压缩率提高了 5 倍。 • 增加了新功能：Amazon DocumentDB 8.0 支持排序规则和视图。 • 增加了对新聚合阶段和运算符的支持：$replaceWith、$VectorSearch、$merge、$set、$unset、$bucket 以及 3 个新的聚合运算符 $pow、$rand、$dateTrunc。 • 增加了对新版本文本索引的支持：Amazon DocumentDB 8.0 中的文本索引 v2 引入了额外的标记，增强了文本搜索功能。 • 包括通过并行向量索引构建改进向量搜索，最多可将索引构建时间缩短 30 倍。 • 移除了对 TLS 1.0 和 TLS 1.1 的支持。现在，最低的 TLS 版本为 TLS 1.2。

Amazon DocumentDB 5.0

亚马逊 DocumentDB 引擎版本 5.0（兼容 MongoDB 5.0）的发行说明。

日期	次要版本	发行说明
2026年6月8日	5.0.1	新的次要版本 亚马逊 DocumentDB 5.0.1（引擎版本 3.0.21138）

日期	次要版本	发行说明
		亚马逊 DocumentDB 5.0.1 是亚马逊 DocumentDB 5.0 主版本的第一个次要版本。次要版本包括新功能、安全修复和错误修复等改进。您可以使用次要版本升级从 5.0.0 升级到 5.0.1。有关更多信息，请参阅 亚马逊 DocumentDB 次要版本升级。 此次要版本包括以下新功能和改进： 1. 在垃圾收集过程的整体运行状况中添加了 gcRuntimeStats documentFragmentStats 、storageSegmentBase 和分解了收集级别的指标，collstats 以便storageSegmentExtended 在垃圾收集过程的整体运行状况中显示更多信息。有关详细信息，请参阅Amazon DocumentDB 中的垃圾回收。 2. 对聚合运算符的支持$rand和$pow聚合运算符。 3. 公开要监控Connections （active） 的实例指标。这些限制可以在我们的 Amazon DocumentDB 配额和限制页面上找到。 4. 对聚合运算符的支持$dateToParts 和$dateFromParts 聚合运算符。 5. Command-level 现在，可在 CloudWatch CRUD 操作（ find、、、、、、insert、findAndModify 、、update、delete、aggregate 、count、、distinct、commitTransaction ）getMoreabortTransaction 、公开和MaxConcurrent 维度中提供性能指标 AvgDuration P100Duration ，以改善操作监控。
2026 年 5 月 6 日	5.0.0	错误修复和其他更改 亚马逊 DocumentDB 5.0（引擎版本：3.0.20421） 此版本适用于必需的安全补丁。

日期	次要版本	发行说明
2026 年 4 月 20 日	5.0.0	新特征 In-place 从亚马逊 DocumentDB 5.0 到亚马逊 DocumentDB 8.0 的主要版本升级 (MVU) 现已推出。有关更多信息，请参阅 Amazon DocumentDB 主版本就地升级。
2026年4月13日	5.0.0	错误修复和其他更改 亚马逊 DocumentDB 5.0（引擎版本：3.0.19917） 这是必需的引擎补丁，包括以下新功能、改进和修复： 1. 添加了 DatabaseConnectionsActiveMax 指标，以提供对实例的“连接（活动）”的见解。这些限制可以在我们的 Amazon DocumentDB 配额和限制页面上找到。 2. 提高了 SASL 身份验证性能。 所有 Amazon DocumentDB 5.0 集群将在下一个维护时段内自动升级到此补丁版本。

日期	次要版本	发行说明
2025 年 10 月 22 日	5.0.0	新特征 亚马逊 DocumentDB 5.0（引擎补丁版本 3.0.17983） Amazon DocumentDB 现在支持最新一代内存优化实例 db.r8g，该实例由 G Arm-based AWS raviton4 处理器提供支持，性能比 R6G 实例提高多达 30%。有关更多信息，请参阅本指南中的 管理实例类 以及此篇博客文章。 新特征 亚马逊 DocumentDB 5.0（引擎补丁版本 3.0.15902） • 增加了对更长集合名称（命名空间长度最长可达 255 个字符）的支持。 • 引入了新的查询计划程序（计划程序版本 2.0）。有关更多信息，请参阅本指南中的查询计划器 v2。 • 在 `collstats` “mvccidStats” 和 “gcRuntimeStats” 中添加了新字段，这些字段提供了有关垃圾收集和老化收集的集合整体运行状况的信息。
2025 年 10 月 16 日	5.0.0	新特征 亚马逊 DocumentDB 5.0（引擎补丁版本 3.0.15902） 当您的集群位于虚拟私有云 (VPCIPv4/IPv6) 中时，Amazon DocumentDB 现在支持双堆栈模式 ()。 有关更多信息，请参阅此篇博客文章以及《Amazon DocumentDB 开发人员指南》中的 Amazon VPC 和 Amazon DocumentDB。
2025 年 9 月 15 日	5.0.0	错误修复和其他更改 亚马逊 DocumentDB 5.0（引擎补丁版本 3.0.17983） 修复了当查询包含后续 `$replaceRoot` 和 `$lookup` 阶段时引擎崩溃的问题。

日期	次要版本	发行说明
2025 年 7 月 29 日	5.0.0	新特征 亚马逊 DocumentDB 5.0（引擎补丁版本 3.0.15902） • 增加了对 `$regexFindAll` 聚合运算符的支持。 • 推出了索引膨胀和基于集合的膨胀指标 • 添加了 CloudWatch 指标`LongestActiveGCRuntime`，这些指标提供了有关垃圾收集`AvailableMVCCIds` 和过时收集的集群整体运行状况的信息。 • 该`serverStatus` 命令现在包含一个 “nvme_cache” 字段，用于显示实例上写入和未写入 NVMe 缓存的页数。 NVMe-backed • 移除了对 `$group` 和 `$project` 的字段数量的限制 • 支持以下字符串运算符：`$trim`、`$rtrim`、`$ltrim`、`$regexFindAll` 、`$replaceOne` 和 `$replaceAll` • 支持以下数组运算符：`$first` 和 `$last` • 支持以下阶段运算符：`$collStats` 错误修复和其他更改 亚马逊 DocumentDB 5.0（引擎补丁版本 3.0.15902） 修复了索引创建过程中影响大型集合的问题。
2025 年 7 月 28 日	5.0.0	新特征 亚马逊 DocumentDB 5.0（引擎补丁版本 3.0.14228） 基于 Amazon DocumentDB 实例的集群现已支持 Amazon DocumentDB 无服务器，这是一种按需自动扩缩配置。 有关更多信息，请参阅 使用 Amazon DocumentDB 无服务器 以及此篇新增功能博客文章。

日期	次要版本	发行说明
2025 年 3 月 24 日	5.0.0	新特征 亚马逊 DocumentDB 5.0（引擎补丁版本 3.0.11800） 添加了对变更流中 `postBatchResumeToken` 的支持。 有关更多信息，请参阅 ######### BatchResumeToken 。
2025 年 1 月 15 日	5.0.0	新特征 亚马逊 DocumentDB 5.0（引擎补丁版本 3.0.11800） Amazon DocumentDB 现在可通过重启写入器实例来维持读取可用性。现在，在写入器实例重新启动期间，读取器实例将继续处理读取请求。 错误修复和其他更改 亚马逊 DocumentDB 5.0（引擎补丁版本 3.0.11800） • 修复了 `killOp` 命令以处理批量插入的特殊情况。 • 改善了亚马逊文档数据库实例的网络 I/O-related 内存使用情况。 • 修复了 `$text` 筛选条件查询的 `count` 命令。
2024 年 12 月 18 日	5.0.0	新特征 Amazon DocumentDB 5.0（引擎补丁版本 3.0.5721） 基于亚马逊文档数据库实例的集群现在支持 NVMe-backed R6GD 实例。 有关更多信息，请参阅 NVMe-backed 实例 以及此篇新增功能博客文章。

日期	次要版本	发行说明
2024 年 11 月 6 日	5.0.0	新特征 亚马逊 DocumentDB 5.0（引擎补丁版本 3.0.11051） • 向 `tls` 集群参数增加了对最低 TLS 版本 `tls1.2+` 和 `tls1.3+` 的支持。 • 启用了对字段名称中美元符号（$）和句点（.）字符的支持。有关功能差异，请参阅 字段名称中的美元符号（$）和句点（.）。
2024 年 9 月 17 日	5.0.0	新特征 Amazon DocumentDB 5.0（引擎补丁版本 3.0.10696） Amazon DocumentDB 现在支持在 `runCommand` 中使用 `reIndex` 重建索引。有关更多信息，请参阅 使用 reIndex 维护索引。 Note `reIndex` 仅在 Amazon DocumentDB 5.0（引擎补丁版本 3.0.10696 及更高版本）受支持。 错误修复和其他更改 Amazon DocumentDB 5.0（引擎补丁版本 3.0.10696） • `$setOnInsert` 现在支持在更新操作造成的插入期间使用 `_id` 字段。 • 修复了在将主要版本升级到 Amazon DocumentDB 5.0 后无法回收存储空间的问题。

日期	次要版本	发行说明
2024 年 8 月 22 日	5.0.0	新特征 亚马逊 DocumentDB 5.0（所有引擎补丁版本） Amazon DocumentDB 全局集群现在支持集群切换和托管式集群失效转移。有关更多信息，请参阅对 Amazon DocumentDB 全局集群执行切换和对 Amazon DocumentDB 全局集群执行托管式失效转移。 **Note** 只有 Amazon DocumentDB 4.0 和 5.0 支持全局集群切换和失效转移。

<table>
<tr><th>日期</th><th>次要版本</th><th>发行说明</th></tr>
<tr><td>2024 年 7 月 23 日</td><td>5.0.0</td><td>新特征

Amazon DocumentDB 5.0（引擎补丁版本 3.0.8126）

• 增加了对长索引名称（最多 255 个字符）的支持。有关更多信息，请参阅 命名约束。
• 将最大连接限制提高了两倍：

<table>
<tr><th>实例类型</th><th>原始限制</th><th>新限制</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>日期</th><th>次要版本</th><th>发行说明</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>实例类型</th><th>原始限制</th><th>新限制</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>
有关更多信息，请参阅 实例限制。

错误修复和其他更改

Amazon DocumentDB 5.0（引擎补丁版本 3.0.8126）

增强了为读取器上的变更流同步 <code>CurrentTime</code> 和 <code>ResumeToken</code> 的逻辑。
</td></tr>
<tr><td>2024 年 7 月 22 日</td><td>5.0.0</td><td>
新特征

Amazon DocumentDB 5.0（引擎补丁版本 3.0.6742）

• 增加了对 DML 审核进行筛选的支持。您现在可以设置筛选条件，根据特定要求筛选掉 DML 审核日志，而不是记录每个 DML 查询。有关更多信息，请参阅 筛选 DML 审核事件。
• 增加了以下操作的文档压缩支持：
  • 设置最小压缩阈值
  • 为现有集合启用压缩（适用于新文档）
  • 允许在集群级别采用默认压缩设置

  有关更多信息，请参阅 管理集合级文档压缩。
• 增加了对在读取器实例上使用变更流的支持。有关更多信息，请参阅 在辅助实例上使用变更流。
</td></tr>
</table>

日期	次要版本	发行说明
2024 年 7 月 9 日	5.0.0	新特征 对于亚马逊 DocumentDB 5.0（最低引擎补丁版本 3.0.4780），TLS CA 证书更新不再需要重启系统。有关更多信息，请参阅 更新您的 Amazon DocumentDB TLS 证书。
2024 年 5 月 29 日	5.0.0	新特征 Amazon DocumentDB 5.0（引擎补丁版本 3.0.6742） • 增加了对 `$regexMatch` 和 `$regexFind` 运算符的支持。 • 增加了在处理大整数时确保审核日志完全准确的支持。现在，审核日志保留了所有数字的精确数值表示，从而防止精度损失。
2024 年 4 月 3 日	5.0.0	新特征 Amazon DocumentDB 5.0（引擎补丁版本 3.0.5721） • 增加了对 `bypassDocumentValidation` 的支持并使 `$jsonSchema` 采用精细的错误消息。有关 `bypassDocumentValidation` 的更多信息，请参阅旁路 DocumentValidation。 • 增加加了对 `$expr` 的支持。 • 增加了对 `$lookup` 中使用不相关联接的支持。 • 增加了在 `$out` 聚合阶段保留验证规则的支持。

日期	次要版本	发行说明
2024 年 1 月 10 日	5.0.0	新特征 Amazon DocumentDB 5.0（引擎补丁版本 3.0.4574、3.0.4780、3.0.4960） • 增加了对 HNSW 向量索引的支持。有关更多信息，请参阅 Amazon DocumentDB 向量搜索。 • 添加了对部分索引的支持。有关更多信息，请参阅 部分索引。 • 增加了在 `currentOp` 命令中对集合上 GC 运行时的支持。 • 增加了对 Amazon DocumentDB 原生文本搜索的文本索引支持。有关更多信息，请参阅 执行 Amazon DocumentDB 文本搜索。 • 增加了对 `$jsonSchema` 架构关键字`type`、`allOf`、`oneOf`、`anyOf`、`not`、`maxItems`、`minItems`、`minProperties`、`pattern`、`patternProperties`、`multipleOf`、`dependencies` 和 `uniqueItems` 的支持。 有关更多信息，请参阅 使用 JSON 架构验证。 • 增加了对算术运算符 `$ceil`、`$floor`、`$ln`、`$log`、`$log10`、`$sqrt` 和 `$exp` 的支持。 有关更多信息，请参阅 算术运算符。 • 添加了对条件表达式运算符 `$switch` 的支持。 • 添加了对并行 `IVFFLAT` 向量索引构建的支持。更新了文档，从开发人员指南中删除了并行 `IVFFLAT` 向量索引构建的限制。 错误修复和其他更改

日期	次要版本	发行说明
		• 重新增加了调用 `db.runCommand("dbstats")` 不区分大小写的功能。使用 3.0.4960 或 2.0.10221 之前的引擎补丁版本的亚马逊 DocumentDB 5.0 客户应应用这些最新的引擎补丁。 • 修复了在 mongo Shell 1.7 及更高版本中调用 `db.coll.stats()` 时出现的错误。更新了文档，从开发人员指南中删除了 mongo Shell `db.coll.stats()` 故障排除技巧。
2023 年 11 月 29 日	5.0.0	新特征 Amazon DocumentDB 5.0（引擎补丁版本 3.0.3727） 增加了对向量搜索的支持。有关更多信息，请参阅这篇博客文章并访问《Amazon DocumentDB 开发人员指南》中的 Amazon DocumentDB 向量搜索 。
2023 年 11 月 21 日	5.0.0	新特征 Amazon DocumentDB 5.0（引擎补丁版本 3.0.3727） 增加了对 I/O-optimized 存储的支持。有关更多信息，请参阅《Amazon DocumentDB 开发人员指南》中的 Amazon DocumentDB 集群存储配置。 添加了与 C SageMaker anvas 的无代码机器学习的集成。有关更多信息，请参阅《Amazon DocumentDB 开发人员指南》中的 No-code 使用 Amazon A SageMaker I Canvas 进行机器学习。

日期	次要版本	发行说明
2023 年 11 月 6 日	5.0.0	亚马逊 DocumentDB 5.0（引擎补丁版本 3.0.3727） 新特征 • 增加了对 $jsonSchema 架构关键字maxLength 、minLength 、maximum、minimum、exclusiveMaximum 、exclusiveMinimum 、items、和 additionalItems 的支持。 请注意，仅基于实例的集群支持 JSON 架构验证。 • 增加了对 $convert 聚合管道运算符及其速记派生运算符$toBool、$toInt、$toLong、$toDouble 、$toString 、$toDecimal 、$toObjectId 、和 $toDate 的支持。 • 增加了对集合表达式运算符$setDifference 、$anyElementTrue 、和 $allElementTrue 的支持。 错误修复和其他更改 修复了变更流从 -NaN 到 NaN 的更新未显示的问题。
2023 年 9 月 15 日	5.0.0	新特征 亚马逊 DocumentDB 5.0（引擎补丁版本 3.0.3140） 增加了对仅在基于实例的集群中使用 $jsonSchema 架构验证器的支持。有关更多信息，请参阅 使用 JSON 架构验证。

日期	次要版本	发行说明
2023 年 7 月 13 日	5.0.0	新特征 Amazon DocumentDB 5.0（引擎补丁版本 3.0.1948） • 增加了对文档压缩的支持。 • 增加了对并行索引构建的支持。 • 添加了对索引构建状态的支持。 错误修复和其他更改 Amazon DocumentDB 5.0（引擎补丁版本 3.0.1948） • 修复了当用户无权访问系统集合时，Amazon DocumentDB 弹性集群 `createCollection` 的身份验证问题。 • 修复了次要区域实例无法使用相同的主区域实例名称的问题。
2023 年 6 月 7 日	5.0.0	错误修复和其他更改 Amazon DocumentDB 5.0 • Amazon DocumentDB 5.0 现在支持 r5 和 t3.medium 实例。 • `engineVersion` 选项`5.0.0`的默认值为 AWS SDK AWS CLI、和 CloudFormation。
2023 年 5 月 10 日	5.0.0	错误修复和其他更改 Amazon DocumentDB 5.0（引擎补丁版本 3.0.1361） • 在createIndex 命令中增加了对 `ignoreunknownindexoptions` 的支持。 • 已停止向审核日志添加内部监控查询。 • 用户定义的局部变量名称现在支持 `$let` 和 `$filter` 等投影运算符使用“_”（下划线）。

日期	次要版本	发行说明

2023 年 3 月 1 日	5.0.0	新特征
		
		Amazon DocumentDB 5.0（引擎补丁版本 3.0.775）
		
		• 推出了 Amazon DocumentDB 5.0
		  • 兼容 MongoDB 5.0（支持 MongoDB 5.0 API 驱动程序）
		  • 支持 Client-side 字段级加密 (FLE)。现在，在写入数据至 Amazon DocumentDB 集群之前，您可以在客户端对字段加密。有关更多信息，请参阅Client-side 字段级加密
		  • 新聚合操作符：`$dateAdd`、`$dateSubtract`
		• 对于所有基于实例的 Amazon DocumentDB 集群和基于分片的弹性集群，将存储限值升高到 128TiB。
		• Amazon DocumentDB 5.0 现在支持使用第一个嵌套级别的 `$elemMatch` 运算符进行索引扫描。如果查询只有一级 `$elemMatch` 过滤器，且嵌套的 `$elemMatch` 查询不支持索引扫描，则支持索引扫描。
		
		支持索引扫描的查询形状：
		
		`db.foo.find( { "a": {$elemMatch: { "b": "xyz", "c": "abc"} } })`
		
		不支持索引扫描的查询形状：
		
		`db.foo.find( { "a": {$elemMatch: { "b": {$elemMatch: { "d": "xyz", "e": "abc"} }} } })`

Amazon DocumentDB 4.0

亚马逊 DocumentDB 引擎版本 4.0（兼容 MongoDB 4.0）的发行说明。

日期	发行说明
2026 年 4 月 27 日	错误修复和其他更改 亚马逊 DocumentDB 4.0（引擎版本：2.0.14278） 这是解决以下问题的必需引擎补丁： 1. 修复了在极少数情况下可能导致时间点恢复 (PITR) 和主要版本升级 (MVU) 无响应的问题。 所有 Amazon DocumentDB 4.0 集群将在下一个维护时段内自动升级到此补丁版本。
2026年3月9日	错误修复和其他更改 亚马逊 DocumentDB 4.0（引擎版本：2.0.14165） 此更新提供了一个可选的引擎补丁，其中包含以下改进： 1. 添加了 DatabaseConnectionsActiveMax 指标，以提供对实例的 “连接（活动）” 的见解。这些限制可以在我们的 Amazon DocumentDB 配额和限制页面上找到。 2. 提高了 SASL 身份验证性能。
2025 年 10 月 22 日	新特征 亚马逊 DocumentDB 4.0（引擎补丁版本 2.0.11747） 在 `collstats` “mvccidStats” 和 “gcRuntimeStats” 中添加了新字段，这些字段提供了有关垃圾收集和老化收集的集合整体运行状况的信息。
2025 年 10 月 16 日	新特征 亚马逊 DocumentDB 4.0（引擎补丁版本 2.0.11747） 当您的集群位于虚拟私有云 (VPCIPv4/IPv6) 中时，Amazon DocumentDB 现在支持双堆栈模式 ()。

日期	发行说明
	有关更多信息，请参阅此篇博客文章以及《Amazon DocumentDB 开发人员指南》中的 Amazon VPC 和 Amazon DocumentDB。
2025 年 7 月 29 日	新特征 亚马逊 DocumentDB 4.0（引擎补丁版本 2.0.11747） • 推出了索引膨胀和基于集合的膨胀指标 • 添加了 CloudWatch 指标`LongestActiveGCRuntime` ，这些指标提供了有关垃圾收集`AvailableMVCCIds` 和过时收集的集群整体运行状况的信息。 • 支持以下字符串运算符：`$trim`、`$rtrim` 和 `$ltrim` • 支持以下阶段运算符：`$collStats` 错误修复和其他更改 亚马逊 DocumentDB 4.0（引擎补丁版本 2.0.11747） 修复了索引创建过程中影响大型集合的问题。
2025 年 1 月 15 日	错误修复和其他更改 亚马逊 DocumentDB 4.0（引擎补丁版本 2.0.11096） • 修复了 `killOp` 命令以处理批量插入的特殊情况。 • 改善了亚马逊文档数据库实例的网络 I/O-related 内存使用情况。
2024 年 11 月 6 日	新特征 亚马逊 DocumentDB 4.0（引擎补丁版本 2.0.10980） • 向 `tls` 集群参数增加了对最低 TLS 版本 `tls1.2+` 和 `tls1.3+` 的支持。

日期	发行说明
2024 年 9 月 17 日	错误修复和其他更改 亚马逊 DocumentDB 4.0（引擎补丁版本 2.0.10898） • `$setOnInsert` 现在支持在更新操作造成的插入期间使用 `_id` 字段。 • 修复了在将主要版本升级到 Amazon DocumentDB 5.0 后无法回收存储空间的问题。
2024 年 8 月 22 日	新特征 亚马逊 DocumentDB 4.0（引擎补丁版本 2.0.5704） Amazon DocumentDB 全局集群现在支持集群切换和托管式集群失效转移。有关更多信息，请参阅对 Amazon DocumentDB 全局集群执行切换和对 Amazon DocumentDB 全局集群执行托管式失效转移。 **Note** 只有 Amazon DocumentDB 4.0 和 5.0 支持全局集群切换和失效转移。

<table>
<tr><th>日期</th><th>发行说明</th></tr>
<tr><td>2024 年 7 月 23 日</td><td>新特征

亚马逊 DocumentDB 4.0（引擎补丁版本 2.0.10709）

• 增加了对长索引名称（最多 255 个字符）的支持。有关更多信息，请参阅 命名约束。
• 将最大连接限制提高了两倍：

<table>
<tr><th>实例类型</th><th>原始限制</th><th>新限制</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>日期</th><th>发行说明</th></tr>
<tr><td></td><td>
<table>
<tr><th>实例类型</th><th>原始限制</th><th>新限制</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>
有关更多信息，请参阅 实例限制。
</td></tr>
<tr><td>2024 年 7 月 22 日</td><td>新特征

Amazon DocumentDB 4.0（引擎补丁版本 2.0.10593）

• 增加了对 DML 审核进行筛选的支持。您现在可以设置筛选条件，根据特定要求筛选掉 DML 审核日志，而不是记录每个 DML 查询。有关更多信息，请参阅 筛选 DML 审核事件。</td></tr>
<tr><td>2024 年 7 月 9 日</td><td>新特征

对于亚马逊 DocumentDB 4.0（最低引擎补丁版本 2.0.10179），TLS CA 证书更新不再需要重启系统。有关更多信息，请参阅 更新您的 Amazon DocumentDB TLS 证书。</td></tr>
<tr><td>2024 年 5 月 29 日</td><td>新特征

Amazon DocumentDB 4.0（引擎补丁版本 2.0.10593）

• 增加了在处理大整数时确保审核日志完全准确的支持。现在，审核日志保留了所有数字的精确数值表示，从而防止精度损失。</td></tr>
</table>

日期	发行说明
2024 年 4 月 3 日	新特征 Amazon DocumentDB 4.0（引擎补丁版本 2.0.10392） • 增加了对 $jsonSchema 的 bypassDocumentValidation 的支持。有关 bypassDocumentValidation 的更多信息，请参阅旁路 DocumentValidation。 • 增加加了对 $expr 的支持。 • 增加了对 $lookup 中使用不相关联接的支持。 • 增加了在 $out 聚合阶段保留验证规则的支持。
2024 年 1 月 10 日	新特征 Amazon DocumentDB 4.0（引擎补丁版本 2.0.10124、2.0.10179、2.0.10221） • 增加了在 currentOp 命令中对集合上 GC 运行时的支持。 • 增加了对 $jsonSchema 架构关键字type、allOf、oneOf、anyOf、not、maxItems、minItems、maxProperties 、minProperties 、pattern、patternProperties 、multipleOf 、dependencies 和 uniqueItems 的支持。 有关更多信息，请参阅 使用 JSON 架构验证。 • 增加了对算术运算符 $ceil、$floor、$ln、$log、$log10、$sqrt 和 $exp 的支持。 有关更多信息，请参阅 算术运算符。 • 添加了对条件表达式运算符 $switch 的支持。
2023 年 12 月 20 日	其他更改 在 Amazon DocumentDB 4.0 中启用了对主版本就地升级的支持。有关更多信息，请参阅 Amazon DocumentDB 主版本就地升级。

日期	发行说明
2023 年 11 月 6 日	亚马逊 DocumentDB 4.0（引擎补丁版本 2.0.9876） 新特征 • 增加了对 $jsonSchema 架构关键字maxLength 、minLength 、maximum、minimum、exclusiveMaximum 、exclusive Minimum 、items、和 additionalItems 的支持。 请注意，仅基于实例的集群支持 JSON 架构验证。 • 增加了对 $convert 聚合管道运算符及其速记派生运算符$toBool、$toInt、$toLong、$toDouble 、$toString 、$toDecimal 、$toObjectId 、和 $toDate 的支持。 • 增加了对集合表达式运算符$setDifference 、$anyElement True 、和 $allElementTrue 的支持。 错误修复和其他更改 修复了变更流从 -NaN 到 NaN 的更新未显示的问题。
2023 年 9 月 20 日	新特征 在 Amazon DocumentDB 4.0 中增加了对主版本就地升级的支持。有关更多信息，请参阅 Amazon DocumentDB 主版本就地升级。
2023 年 9 月 15 日	新特征 亚马逊 DocumentDB 4.0（引擎补丁版本 2.0.9686） 增加了对仅在基于实例的集群中使用 $jsonSchema 架构验证器的支持。有关更多信息，请参阅 使用 JSON 架构验证。

日期	发行说明
2023 年 7 月 13 日	新特征 Amazon DocumentDB 4.0（引擎补丁版本 2.0.9259） • 增加了对并行索引构建的支持。 错误修复和其他更改 Amazon DocumentDB 4.0（引擎补丁版本 2.0.9259） • 已停止向审核日志添加内部监控查询。
2023 年 4 月 4 日	错误修复和其他更改 Amazon DocumentDB 4.0（引擎补丁版本 2.0.8934） • 修复了在持续工作负载期间启用 DML 审核时出现的问题。 • 修复了当传递带有提示的聚合命令传递字符串值时，DML 审核的问题。 • 修复了当具有 readwriteanydatabase 角色的用户同时将 authorizedCollections 和 nameOnly 选项设为 true 时，`listCollections` 命令无法工作的问题。 • 修复了正确解析字段名称中的数字字符串的问题。 • 当长时间运行的光标影响垃圾回收时，将其取消。 • 用户定义的局部变量名称现在支持 `$let` 和 `$filter` 等投影运算符使用“_”（下划线）。
2023 年 2 月 27 日	错误修复和其他更改 Amazon DocumentDB 4.0 增加了对的支持 AWS Lambda。有关更多信息，请参阅AWS Lambda 与更改流一起使用。

日期	发行说明
2022 年 8 月 9 日	新特征 4.0 • 添加了对 Decimal128 数据类型的支持。Decimal128 是一种 BSON 数据类型，在所有提供 DocumentDB 的地区都受支持。 有关更多信息，请参阅数据类型。 • 增加了对使用 Amazon Lo CloudWatch gs 进行 DML 查询审计的支持。现在，Amazon DocumentDB 可以将数据操纵语言 (DML) 事件和数据定义语言 (DDL) 事件记录到亚马逊日志。 CloudWatch 有关更多信息，请参阅此博客文章。 错误修复和其他更改 4.0 • 现在，您可以使用自己的 `changeOwnPassword` 特权密码更改自己的密码。
2022 年 7 月 25 日	新特征 Amazon DocumentDB 4.0 现在，您可以创建使用相同 DocumentDB 集群卷并具有与原始集群相同数据的克隆，从而更快地创建集群。有关详细信息，请参阅管理 Amazon DocumentDB 集群。
2022 年 6 月 27 日	新特征 Amazon DocumentDB 4.0（引擎补丁版本 2.0.7509） Amazon DocumentDB 根据使用模式动态调整数据库的大小。添加更多数据会将空间增加到 64 TiB，而删除数据会减少分配的空间。

日期	发行说明
2022 年 4 月 7 日	新特征 4.0（引擎补丁版本 2.0.6142 和 2.0.6948） Amazon DocumentDB Performance Insights 现已推出预览版。现在，您可以在滚动窗口中存储七天的性能历史记录，无需支付额外费用。有关更多信息，请参阅通过性能洞察进行监控。

2022 年 1 月 24 日	新特征
	
	4.0（引擎补丁版本 2.0.5170）
	
	• Amazon DocumentDB 现在提供免费试用。有关详情，请参阅 Amazon DocumentDB 免费试用页面。
	• 现在，您可以在地理空间查询中使用增强功能，包括以下 API：
	  • `$geoWithin`
	  • `$geoIntersects`
	• 增加了对以下 MongoDB 运算符的支持：
	  • `$mergeObjects`
	  • `$reduce`
	
	有关更多信息，请参阅使用 Amazon DocumentDB 查询地理空间数据。

日期	发行说明

2022 年 1 月 21 日	新特征
	
	Amazon DocumentDB 4.0（引擎补丁版本 2.0.5706）
	
	• 现在支持 Amazon DocumentDB Graviton2（r6g.large、r6g.2xlarge、r6g.4xlarge、r6g.8xlarge、r6g.12xlarge、r6g.16xlarge 和 t4g.medium）实例。
	
	新特征
	
	4.0（引擎补丁版本 2.0.5706）
	
	• 增加了对以下 MongoDB API 内容的支持：
	  • `$reduce`
	  • `$mergeObjects`
	  • `$geoWithin`
	  • `$geoIntersects`

日期	发行说明

2021 年 10 月 25 日	新特征
	
	4.0（引擎补丁版本 2.0.5704）
	
	• 增加了对以下 MongoDB API 内容的支持
	  • `$literal`
	  • `$map`
	  • `$$ROOT`
	• Support 对 GeoSpatial 查询功能的支持。有关更多详细信息，请参阅此博客文章
	• 支持使用用户定义的角色进行访问控制。有关更多详细信息，请参阅此博客文章
	• Amazon DocumentDB JDBC 驱动程序支持通过 BI 工具（例如 Tableau）和 SQL Workbench 等查询工具进行连接
	
	错误修复和其他更改
	
	4.0（引擎补丁版本 2.0.5704）
	
	• 错误修复：当显式 `.sort()` 与 `$natural` 同时存在时，`$natural` 可正确排序
	• 变更流与 `$redact` 协同工作的错误修复
	• 修复了 `$ifNull` 在处理空数组时的错误
	• 修复了当前登录的用户被删除或该用户对正在进行的活动的权限被撤销时资源过度 consumption/server 崩溃的错误
	• 错误修复 `listDatabase` 和 `listCollection` 权限检查
	• 错误修复了多键元素的重复数据删除逻辑

日期	发行说明
2021 年 6 月 24 日	新特征 4.0（引擎补丁版本 2.0.3371） • 现在支持 r5.8xlarge 和 r5.16xlarge 实例。在博客文章中了解更多信息 Amazon DocumentDB 现在支持 r5.8xlarge 和 r5.16xlarge 实例。 • 现在支持全局集群，以便从区域范围的中断中提供灾难恢复，并允许从最近的 Amazon DocumentDB 集群进行读取，从而实现低延迟的全局读取。请注意，以下区域当前不支持全局集群：南美洲（圣保罗）、欧洲地区（米兰）、中国（北京）和中国（宁夏）。
2021 年 5 月 4 日	新特征 4.0（引擎补丁版本 2.0.3371） • `renameCollection` • `$zip` • `$indexOfArray` • `$reverseArray` • `$natural` • `$hint` 支持更新 • 索引扫描 `distinct` 错误修复和其他更改 4.0（引擎补丁版本 2.0.3371） • 减少 `$in` 查询的内存使用量 • 修复了多键索引中的内存泄漏问题 • 修复了 `$out` 的解释计划和探查器输出 • 增加了内部监控系统操作的超时时间，以提高可靠性 • 修复了影响传递给多键索引的查询谓词的缺陷

日期	发行说明
2021 年 1 月 15 日	新特征 Amazon DocumentDB 4.0（引擎补丁版本 2.0.722） • 无

日期	发行说明
2020 年 11 月 9 日	新特征 Amazon DocumentDB 4.0（引擎补丁版本 2.0.722） • 与 MongoDB 4.0 的兼容性 • ACID 事务 • 支持 cluster(client.watch() 或 mongo.watch()) 和数据库级 (db.watch()) 变更流 • 能够使用 startAtOperationTime 启动或恢复变更流 • 将变更流保留期延长至 7 天（之前为 24 小时） • AWS DMS 亚马逊 DocumentDB 4.0 的目标 • CloudWatch 指标：TransactionsOpen TransactionsOpenMax 、TransactionsAborted 、TransactionsStarted 、和 TransactionsCommitted • currentOp 、ServerStatus 和 profiler 中事务的新字段。 • 能够在 $lookup 聚合阶段使用索引 • 可以从索引方向提供带有投影的 find() 查询（涵盖的查询） • 能够使用 hint() 和 findAndModify • 为 $addToSet 运算符提供性能优化 • 缩小整体索引大小的改进。 • 新的聚合运算符：$ifNull、$replaceRoot 、$setIsSubset 、$setInstersection 、$setUnion 、和 $setEquals • 使用 ListCollection 和 ListDatabase 命令，您现在可以选择使用 authorizedCollections 和 authorizedDatabases 参数来允许用户列出他们有权访问的集合和数据库，而无需分别使用 listCollections 和 listDatabase 角色 • 用户也可以在不需要 KillCursor 角色的情况下结束自己的光标 • 现在，比较子文档的数值类型与比较第一级文档的数值类型是一致的。Amazon DocumentDB 4.0 中的行为现在与 MongoDB 兼容。 错误修复和其他更改

日期	发行说明
	Amazon DocumentDB 4.0（引擎补丁版本 2.0.722） • 在使用位置运算符 $ 时，$setOnInsert 不再允许更新。Amazon DocumentDB 4.0 中的行为现在与 MongoDB 兼容。 • 借助 $createCollection 修复了问题并设置 autoIndexId • 嵌套文档投影 • 已将工作内存的默认设置更改为随实例内存大小进行扩展 • 垃圾回收改进 • 在路径中使用空键查找，与 mongo 的行为差异 • dateToString 修复了时区行为中的错误 • 修复 $push（聚合）以遵守排序顺序 • 修复了 $currentOp 中的聚合错误 • 修复了 readPreference 在辅助设备上的问题 • 修复了验证 $createIndex 是与发出命令时相同的数据库的问题 • 修复了 minKey、maxKey 查找失败时的不一致行为 • 修复了 $size 运算符无法使用复合数组的问题 • 修复了使用正则表达式否定 $in 的问题 • 修复了 $distinct 命令在视图上运行的问题 • 修复了聚合和查找命令对缺失字段进行不同排序的问题 • 修复了 $eq 到正则表达式不检查类型的问题 • 修复了时间戳顺序位置行为中的 $currentDate 错误 • 修复了 $currentDate 的毫秒粒度

Amazon DocumentDB 3.6

亚马逊 DocumentDB 引擎版本 3.6（兼容 MongoDB 3.6）的发行说明。

日期	发行说明
2026 年 4 月 27 日	新特征 亚马逊 DocumentDB 3.6（引擎版本：1.0.208692）[进行中]

日期	发行说明
	此更新提供了一个可选的引擎补丁，其中包含以下改进： 1. 添加了对变更流中 `startAtOperationTime` 的支持。
2026 年 2 月 12 日	错误修复和其他更改 亚马逊 DocumentDB 3.6（引擎版本：1.0.208675） 1. 添加了 DatabaseConnectionsActiveMax 指标，以提供对实例的 “连接（活动）” 的见解。这些限制可以在我们的 Amazon DocumentDB 配额和限制页面上找到。 2. 提高了 SASL 身份验证性能。 3. 修复了 $indexStats 命令的错误。
2024 年 8 月 20 日	新特征 对于 Amazon DocumentDB 3.6（最低引擎补丁版本 1.0.208662），TLS CA 证书更新不再需要重新启动系统。有关更多信息，请参阅更新您的 Amazon DocumentDB TLS 证书。
2024 年 7 月 22 日	新特征 Amazon DocumentDB 3.6（引擎补丁版本 1.0.208662） 移除了 Amazon DocumentDB 主版本就地升级中对 db.r5.* 和 db.r6.* 实例的索引限制。有关更多信息，请参阅 先决条件。 错误修复和其他更改 Amazon DocumentDB 3.6（引擎补丁版本 1.0.208662） Amazon DocumentDB 现在将 -NaN 识别为有效的 JSON 令牌。
2023 年 12 月 20 日	其他更改 在 Amazon DocumentDB 3.6 中启用了对主版本就地升级的支持。有关更多信息，请参阅 Amazon DocumentDB 主版本就地升级。

日期	发行说明
2023 年 11 月 17 日	错误修复和其他更改 Amazon DocumentDB 3.6（引擎补丁版本 1.0.208570） 用户定义的局部变量名称现在支持 `$let` 和 `$filter` 等投影运算符使用“_”（下划线）。
2023 年 9 月 20 日	新特征 在 Amazon DocumentDB 3.6 中增加了对主版本就地升级的支持。有关更多信息，请参阅 Amazon DocumentDB 主版本就地升级。
2023 年 2 月 2 日	错误修复和其他更改 Amazon DocumentDB 3.6（引擎补丁版本 1.0.208432） • 修复了在持续工作负载期间启用 DML 审核时出现的问题。 • 修复了当传递带有提示的聚合命令传递字符串值时，DML 审核的问题。 • 修复了当具有 readwriteanydatabase 角色的用户同时将 authorizedCollections 和 nameOnly 选项设为 true 时，`listCollections` 命令无法工作的问题。 • 修复了正确解析字段名称中的数字字符串的问题。 • 当长时间运行的光标影响垃圾回收时，将其取消。

日期	发行说明
2022 年 8 月 9 日	新特征 亚马逊 DocumentDB 3.6（引擎补丁版本 1.0.208152） • 添加了对 Decimal128 数据类型的支持。Decimal128 是一种 BSON 数据类型，在所有提供 DocumentDB 的地区都受支持。 有关更多信息，请参阅数据类型。 • 增加了对使用 Amazon Lo CloudWatch gs 进行 DML 查询审计的支持。现在，Amazon DocumentDB 可以将数据操纵语言 (DML) 事件和数据定义语言 (DDL) 事件记录到亚马逊日志。 CloudWatch 有关更多信息，请参阅此博客文章。 错误修复和其他更改 亚马逊 DocumentDB 3.6（引擎补丁版本 1.0.208152） • 现在，您可以使用自己的 `changeOwnPassword` 特权密码更改自己的密码。
2022 年 4 月 7 日	新特征 亚马逊 DocumentDB 3.6（引擎补丁版本 1.0.207836 和 1.0.208015） Amazon DocumentDB Performance Insights 现已推出预览版。现在，您可以在滚动窗口中存储七天的性能历史记录，无需支付额外费用。有关更多信息，请参阅通过性能洞察进行监控。

日期	发行说明

2022 年 1 月 24 日	新特征
	
	亚马逊 DocumentDB 3.6（引擎补丁版本 1.0.207684）
	
	• Amazon DocumentDB 现在提供免费试用。有关详情，请参阅 Amazon DocumentDB 免费试用页面。
	• 现在，您可以在地理空间查询中使用增强功能，包括以下 API：
	  • `$geoWithin`
	  • `$geoIntersects`
	• 增加了对以下 MongoDB 运算符的支持：
	  • `$mergeObjects`
	  • `$reduce`
	
	有关更多信息，请参阅使用 Amazon DocumentDB 查询地理空间数据。

2022 年 1 月 21 日	新特征
	
	亚马逊 DocumentDB 3.6（引擎补丁版本 1.0.207781）
	
	• 增加了对以下 MongoDB API 内容的支持：
	  • `$reduce`
	  • `$mergeObjects`
	  • `$geoWithin`
	  • `$geoIntersects`

日期	发行说明

2021 年 10 月 25 日	新特征
	
	亚马逊 DocumentDB 3.6（引擎补丁版本 1.0.207780）
	
	• 增加了对以下 MongoDB API 内容的支持
	  • `$literal`
	  • `$map`
	  • `$$ROOT`
	• Support 对 GeoSpatial 查询功能的支持。有关更多详细信息，请参阅此博客文章
	• 支持使用用户定义的角色进行访问控制。有关更多详细信息，请参阅此博客文章
	• Amazon DocumentDB JDBC 驱动程序支持通过 BI 工具（例如 Tableau）和 SQL Workbench 等查询工具进行连接
	
	错误修复和其他更改
	
	亚马逊 DocumentDB 3.6（引擎补丁版本 1.0.207780）
	
	• 错误修复：当显式 `.sort()` 与 `$natural` 同时存在时，`$natural` 可正确排序
	• 变更流与 `$redact` 协同工作的错误修复
	• 修复了 `$ifNull` 在处理空数组时的错误
	• 修复了当前登录的用户被删除或该用户对正在进行的活动的权限被撤销时资源过度 consumption/server 崩溃的错误
	• 错误修复 `listDatabase` 和 `listCollection` 权限检查
	• 错误修复了多键元素的重复数据删除逻辑

日期	发行说明
2021 年 6 月 24 日	新特征 亚马逊 DocumentDB 3.6（引擎补丁版本 1.0.207117） • 现在支持 r5.8xlarge 和 r5.16xlarge 实例。在博客文章中了解更多信息 Amazon DocumentDB 现在支持 r5.8xlarge 和 r5.16xlarge 实例。 • 现在支持全局集群，以便从区域范围的中断中提供灾难恢复，并允许从最近的 Amazon DocumentDB 集群进行读取，从而实现低延迟的全局读取。请注意，以下区域当前不支持全局集群：南美洲（圣保罗）、欧洲地区（米兰）、中国（北京）和中国（宁夏）。
2021 年 5 月 4 日	新特征 亚马逊 DocumentDB 3.6（引擎补丁版本 1.0.207117） • `renameCollection` • `$zip` • `$indexOfArray` • `$reverseArray` • `$natural` • `$hint` 支持更新 • 索引扫描 `distinct` 错误修复和其他更改 亚马逊 DocumentDB 3.6（引擎补丁版本 1.0.207117） • 减少 `$in` 查询的内存使用量 • 修复了多键索引中的内存泄漏问题 • 修复了 `$out` 的解释计划和探查器输出 • 增加了内部监控系统操作的超时时间，以提高可靠性 • 修复了影响传递给多键索引的查询谓词的缺陷

日期	发行说明
2021 年 1 月 15 日	新特征 Amazon DocumentDB 3.6（引擎补丁版本 1.0.206295） • 能够在 $lookup 聚合阶段使用索引 • 可以从索引方向提供带有投影的 find() 查询（涵盖的查询） • 能够使用 hint() 和 findAndModify • 为 $addToSet 运算符提供性能优化 • 缩小整体索引大小的改进 • 新的聚合运算符：$ifNull、$replaceRoot 、$setIsSubset 、$setInstersection 、$setUnion 、和 $setEquals • 用户也可以在不需要 KillCursor 角色的情况下结束自己的光标
2020 年 11 月 9 日	新特征 Amazon DocumentDB 3.6（引擎补丁版本 1.0.206295） • 无 错误修复和其他更改 Amazon DocumentDB 3.6（引擎补丁版本 1.0.206295） • 无

日期	发行说明
2020 年 10 月 30 日	新特征 Amazon DocumentDB 3.6（引擎补丁版本 1.0.206295） • 添加了在集群级别（client.watch() 或 mongo.watch()）和数据库（db.watch()）中打开变更流光标的功能 • 能够将变更流保留期延长至 7 天（之前为 24 小时） 错误修复和其他更改 Amazon DocumentDB 3.6（引擎补丁版本 1.0.206295） • 各种常规手机壳性能改进 • 有针对性的安全改进 • 修复了在复合索引的第二个字段上跳过排序的问题 • 在多键索引（非复合索引）的单个字段上启用常规索引以实现相等 • 修复了身份验证竞赛条件 • 修复了导致垃圾回收不频繁崩溃的问题 • RBAC 安全改进 • 已添加 databaseConnectionsMax 指标 • r5.24xlarge 实例上某些工作负载的性能改进
2020 年 9 月 22 日	新特征 Amazon DocumentDB 3.6（引擎补丁版本 1.0.206295） • $out 聚合阶段 • 将每个实例的最大连接数和光标数增加了 10 倍 错误修复和其他更改 Amazon DocumentDB 3.6（引擎补丁版本 1.0.206295） • 无

日期	发行说明
2020 年 7 月 10 日	新特征 Amazon DocumentDB 3.6（引擎补丁版本 1.0.206295） • 跨区域快照复制 错误修复和其他更改 Amazon DocumentDB 3.6（引擎补丁版本 1.0.206295） • 无

日期	发行说明
2020 年 6 月 30 日	新特征 Amazon DocumentDB 3.6（引擎补丁版本 1.0.206295） • T3 medium 实例 错误修复和其他更改 Amazon DocumentDB 3.6（引擎补丁版本 1.0.206295） • t3 实例的空闲内存回收 • 身份验证改进 • 改进 SASL 身份验证性能 • 修复了超过最大可能操作数的 `currentOp` 问题 • 修复了批量更新和删除的 `killOps` 问题 • 使用 `$match` 改进 `$sample` 的性能 • 在编辑阶段修复了对 cond case 中 $$ 的支持 • 修复了各种反复出现的崩溃根本原因 • 对 TTL 扫描进行了改进，以减少 IOs 和延迟 • 优化了 `$unwind` 的内存利用率 • 修复了带有掉落指数的收集统计数据竞赛条件 • 修复了并发索引构建期间的争用条件 • 修复了 `hash_search` 索引中很少出现的崩溃问题

亚马逊 DocumentDB 通用发行说明

常规服务更新，包括区域可用性、工具集成和跨版本功能。

日期	发行说明
2026 年 4 月 29 日	新特征

日期	发行说明
	Amazon DocumentDB 现已在加拿大西部（卡尔加里）地区推出。有关更多信息，请参阅此博客文章。
2026 年 4 月 17 日	新特征 现在可以在 DocumentDB 无服务器上实现更快的扩展。
2026 年 2 月 11 日	新特征 Amazon DocumentDB 现已在亚太地区（墨尔本）地区推出。有关更多信息，请参阅此博客文章。 Amazon DocumentDB 现已在欧洲（苏黎世）地区推出。有关更多信息，请参阅此博客文章。
2026年1月8日	新特征 Amazon DocumentDB 现已在亚太地区（雅加达）地区推出。有关更多信息，请参阅此博客文章。
2025 年 10 月 13 日	新特征 Amazon DocumentDB 现已在亚太地区（海得拉巴）区域支持 R6G 和 T4G 实例。 有关更多信息，请参阅 不同区域支持的实例类。
2025 年 10 月 7 日	新特征 Amazon DocumentDB 现在已在以下区域推出： • 亚太地区（马来西亚） • 亚太地区（大阪） • 亚太地区（泰国） • 墨西哥（中部） 有关更多信息，请参阅此博客文章。

日期	发行说明
2025 年 9 月 26 日	新特征 Amazon DocumentDB 现已支持跨区域和跨账户的快照复制和快照共享。 有关更多信息，请参阅跨复制快照 AWS 区域和Amazon DocumentDB 集群快照共享。
2025 年 6 月 18 日	新特征 Amazon DocumentDB 现在已在以色列（特拉维夫）区域推出。
2025 年 5 月 8 日	新特征 Amazon DocumentDB 现已在欧洲地区（斯德哥尔摩）区域推出。有关更多信息，请参阅此博客文章。
2025 年 4 月 2 日	错误修复和其他更改 修复了使用后台索引创建向量索引时出现的错误。
2025 年 2 月 6 日	新特征 亚马逊 DocumentDB 现已与集成。 AWS Toolkit for Visual Studio Code有关更多信息，请参阅此篇新增功能博客文章，并参阅《AWS Toolkit for Visual Studio Code 用户指南》中的在 Toolkit 中使用 Amazon DocumentDB。
2024 年 10 月 22 日	新特征 Amazon DocumentDB 现在支持存储网络吞吐量指标 `StorageNetworkReceiveThroughput` 、`StorageNetworkTransmitThroughput` 和 `StorageNetworkThroughput` 。有关更多信息，请参阅 使用指标评估亚马逊 DocumentDB 实例使用情况 CloudWatch。

日期	发行说明
2024 年 9 月 18 日	新特征 Amazon DocumentDB 现已在非洲（开普敦）区域中推出。有关更多信息，请参阅此博客文章。 Amazon DocumentDB 现已在欧洲（西班牙）区域推出。有关更多信息，请参阅此博客文章。
2024 年 6 月 25 日	新特征 使用 AWS IAM ARN 进行身份验证可在所有受支持区域的 Amazon DocumentDB 实例的 5.0 集群中使用。有关更多信息，请参阅 使用 IAM 身份进行身份验证。
2024 年 4 月 3 日	新特征 Amazon DocumentDB 现已在中东（阿联酋）区域中推出。有关更多信息，请参阅此博客文章。 错误修复和其他更改 • 修复了在 mongo Shell 1.7 及更高版本中调用 `db.coll.stats()` 时出现的错误。 • 修复了变更流查询的内存泄漏问题，该查询中含有 `$regex` 作为相同聚合管道中的一部分。
2024 年 1 月 30 日	新特征 Amazon DocumentDB 全局集群 全球集群现已在两个 AWS GovCloud (US) 区域推出： AWS GovCloud (US-East) 和 AWS GovCloud (US-West)。
2023 年 12 月 13 日	新特征 增加了对一键连接 EC2 的支持。有关更多信息，请参阅 使用 Amazon EC2 进行连接。

日期	发行说明
2023 年 11 月 17 日	新特征 Amazon DocumentDB 现已在 AWS GovCloud (US-East) 区域上线。有关更多信息，请参阅此博客文章。
2023 年 9 月 25 日	新特征 Amazon DocumentDB 现已在亚太地区（香港）区域中提供。有关更多信息，请参阅此博客文章。
2023 年 9 月 11 日	新特征 Amazon DocumentDB 现已在亚太地区（海得拉巴）区域中推出。有关更多信息，请参阅此博客文章。
2022 年 4 月 29 日	新特征 Amazon DocumentDB 现已在中国（北京）区域推出。有关更多信息，请参阅此博客文章。
2022 年 3 月 16 日	新特征 Amazon DocumentDB 现已在欧洲（米兰）区域推出。有关更多信息，请参阅此博客文章。
2022 年 2 月 8 日	新特征 Amazon DocumentDB R6G 和 T4G 实例现已在亚太地区、南美和欧洲上市。有关更多信息，请参阅此博客文章。
2021 年 5 月 4 日	新特征 在这篇博客文章中查看所有新功能。
2020 年 11 月 9 日	新特征 在这篇博客文章中查看所有新功能。

日期	发行说明
2020 年 10 月 30 日	新特征 在这篇博客文章中查看所有新功能。
2020 年 9 月 22 日	新特征 在这篇博客文章中查看所有新功能。
2020 年 7 月 10 日	新特征 在这篇博客文章中查看所有新功能。
2020 年 6 月 30 日	新特征 在这篇博客文章中查看所有新功能。

亚马逊 DocumentDB 弹性集群发行说明

亚马逊 DocumentDB 弹性集群的发行说明。

日期	发行说明
2025 年 1 月 28 日	新特征 为了实现一键连接，Amazon DocumentDB 现已 AWS CloudShell 与基于实例的集群和弹性集群集集成。 有关更多信息，请参阅以下任何或全部文章： • 开始使用 Amazon DocumentDB • 开始使用 Amazon DocumentDB 弹性集群 • 最新功能博客文章 • 技术操作指南博客文章
2024 年 11 月 12 日	新特征 Amazon DocumentDB 弹性集群现在支持后台索引。

日期	发行说明
2024 年 11 月 1 日	新特征 Amazon DocumentDB 现在支持弹性集群维护操作。有关更多信息，请参阅 维护 Amazon DocumentDB 弹性集群。
2024 年 8 月 8 日	新特征 Amazon DocumentDB 弹性集群目前已在亚太地区（香港）、加拿大（中部）和欧洲地区（巴黎）推出。有关更多信息，请参阅 弹性集群的区域可用性。
2024 年 7 月 8 日	新特征 Amazon DocumentDB 弹性集群现已在欧洲地区（米兰）推出。有关更多信息，请参阅 弹性集群的区域可用性。
2024 年 2 月 22 日	新特征 Amazon DocumentDB 弹性集群 Amazon DocumentDB 弹性集群现在支持以下功能： • 可读的辅助分片实例副本：有关更多信息，请参阅 第 1 步：创建弹性集群 的步骤 5b。 • Start/stop 集群-有关更多信息，请参阅停止和启动 Amazon DocumentDB 弹性集群。 • 可配置的分片实例：有关更多信息，请参阅 第 1 步：创建弹性集群 的步骤 5b。 • 自动快照备份：有关更多信息，请参阅 管理弹性集群快照的自动备份。 • 复制快照：有关更多信息，请参阅 复制弹性集群快照。

日期	发行说明
2024 年 1 月 30 日	新特征 Amazon DocumentDB 弹性集群 Amazon DocumentDB 弹性集群现已在以下区域推出。 • 亚太地区（孟买） • 亚太地区（首尔） • 南美洲（圣保罗） • 欧洲地区（伦敦） 有关更多信息，请参阅 弹性集群区域和版本可用性。

2023 年 8 月 3 日	新特征
	
	Amazon DocumentDB 弹性集群
	
	• Amazon DocumentDB 弹性集群现在支持以下操作：
	  • `top`
	  • `collStats`
	  • `hint`
	  • `dataSize`
	
	  有关支持的命令和操作的完整列表，请参阅 Amazon DocumentDB 中支持的 MongoDB API、操作和数据类型。
	• 现在支持生存时间 (TTTL) 索引。
	• 索引 `hints` 表达式现在支持索引。

2023 年 3 月 22 日	新特征 Amazon DocumentDB 弹性集群现已在亚太地区（新加坡）、亚太地区（悉尼）和亚太地区（东京）推出。有关更多信息，请参阅 弹性集群区域和版本可用性。

日期	发行说明
2022 年 11 月 30 日	新特征 Amazon DocumentDB 弹性集群 Amazon DocumentDB 弹性集群是一种新型的 Amazon DocumentDB 集群，它使用户能够利用 MongoDB 分片 API 来横向扩展其集群。弹性集群通过将数据和计算分布在多个底层计算实例和卷上，以 PB 级的存储容量处理几乎任意数量的读取和写入。要了解更多信息，请参阅使用 Amazon DocumentDB 弹性集群。

Amazon DocumentDB 开发人员指南的文档历史记录

- API 版本：2014-10-31
- 最新文档更新：2026 年 5 月 27 日

下表介绍了该版本的 Amazon DocumentDB 开发人员指南文档。

变更	说明	日期
全新 Amazon DocumentDB 开发者指南	亚马逊 DocumentDB 开发者指南现已以新 URL 发布。API 参考可在亚马逊 DocumentDB API 参考中单独获得。	2026年5月27日
AWS 托管策略更新-策略变更	Amazon DocumentDB 更新了弹性集群的完全访问策略。	2025 年 2 月 11 日
AWS 托管策略更新-策略变更	Amazon DocumentDB 更新了弹性集群的完全访问策略。	2024 年 2 月 21 日
AWS 托管策略更新-策略变更	Amazon DocumentDB 更新了弹性集群的只读和完全访问策略。	2023 年 6 月 21 日
AWS 托管策略更新-新策略	Amazon DocumentDB 为弹性集群引入了新的只读策略。	2023 年 6 月 8 日
AWS 托管策略更新-新策略	Amazon DocumentDB 为弹性集群引入了新的完全访问策略。	2023 年 6 月 5 日
MongoDB 5.0 兼容性	Amazon DocumentDB 现在与 MongoDB 的 5.0 版本兼容。	2023 年 3 月 1 日
策略更新	为了支持 Amazon DocumentDB 弹性集群功能，我们更新了 AmazonDocDBConsoleFullAccess 政策，并引	2022 年 11 月 30 日

	入 AmazonDocDB-ElasticServiceRolePolicy 了。	
弹性集群	增加了新的弹性集群功能，支持在 Amazon DocumentDB 的分布式存储系统中对数据进行基于哈希的分区（分片）。	2022 年 11 月 30 日
全局集群	增加了有关如何使用全局集群的文档。	2021 年 6 月 2 日
事件订阅	增加了活动订阅文档。	2021 年 3 月 26 日
版本 3.6 升级	记录了对版本 3.6 在基于角色的访问控制、聚合运算符和性能方面的改进。	2021 年 1 月 15 日
MongoDB 4.0 兼容性	现在，Amazon DocumentDB 与 MongoDB 的 4.0 版本兼容。	2020 年 11 月 9 日
入门指南	使用亚马逊 EC2 AWS Cloud9、Robo3T 或 Studio3T 开始使用亚马逊 DocumentDB 的新入门指南。	2023 年 8 月 15 日
支持其他可用区	Amazon DocumentDB 增加了对亚太地区（首尔）(ap-northeast-2) 可用区的支持。	2023 年 7 月 14 日
增加对跨区域复制快照的支持。	Amazon DocumentDB 增加了对跨 AWS 区域集群快照复制的支持。有关更多信息，请参阅 跨区域复制快照。	2020 年 7 月 10 日

增加了对 T3 实例类的支持。	在支持 Amazon DocumentDB 的所有区域中增加了对 T3 实例类型的支持。有关更多信息，请参阅 按区域支持的实例类和实例类规格。	2020 年 6 月 30 日
添加了对的支持 AWS GovCloud (US)。	亚马逊 DocumentDB 现已 AWS GovCloud (US) 在该地区 (us-gov-west-1) 上市。	2020 年 6 月 29 日
添加了 16 个新 CloudWatch 指标。	亚马逊 DocumentDB 增加了对 16 个新的亚马逊 CloudWatch 指标的支持。有关更多信息，请参阅使用监控 Amazon DocumentDB。 CloudWatch	2020 年 6 月 23 日
添加了对 null 字符和 $regex 运算符的支持。	Amazon DocumentDB 添加了对字符串中的 null 字符的支持，并能够为 $regex 使用索引。要查看支持的 MongoDB API 和 Amazon DocumentDB 的聚合管道功能，请参阅 与 MongoDB 的功能差异。	2020 年 6 月 22 日
增加了对改进的多键索引功能的支持。	Amazon DocumentDB 增加了对改进的多键索引功能的支持，这些功能包括对大于 2048 个字节的数组编制索引的功能和使用相同数组中的多个键创建复合多键索引的能力。有关更多信息，请参阅与 MongoDB 之间的功能差异。	2020 年 4 月 23 日
增加了对亚马逊 DocumentDB CloudFormation 堆栈的删除保护的支持。	亚马逊 DocumentDB 增加了对在创建亚马 CloudFormation 逊 DocumentDB 堆栈时启用删除保护的支持。	2020 年 4 月 20 日

添加了对基于角色的访问控制的支持。	Amazon DocumentDB 使用内置角色添加了对基于角色的访问控制的支持。	2020 年 3 月 26 日
在加拿大（中部）(ca-central-1) 中添加了对其他可用区的支持。	现在，Amazon DocumentDB 在加拿大（中部）区域 (ca-central-1) 推出，提供 R5 类实例和 3 个可用区。	2020 年 3 月 26 日
增加了对两个额外的 MongoDB API 的支持。	Amazon DocumentDB 增加了对 `$dateFromString` 和 `executionStats` MongoDB API 的支持。	2020 年 3 月 23 日
增加了对五个额外的 MongoDB API 的支持。	Amazon DocumentDB 增加了对 `$objectToArray` 、`$arrayTo0bject` 、`$slice`、`$mod` 和 `$range` MongoDB API 的支持。	2020 年 2 月 6 日
增加了对加拿大（中部）的支持。	Amazon DocumentDB 现已在在加拿大（中部）区域 (ca-central-1) 推出，提供 R5 类实例。	2019 年 12 月 11 日
添加了对的支持 ChangeStreamLogSize。	Amazon DocumentDB 为 `ChangeStreamLogSize` 添加了对 Cloudwatch 指标的支持。	2019 年 11 月 22 日
增加了对欧洲地区（巴黎）的支持	现在，Amazon DocumentDB 在欧洲地区（巴黎）区域 (eu-west-3) 推出，提供 R5 类实例。	2019 年 10 月 30 日

添加了对亚太地区（孟买）区域的支持	Amazon DocumentDB 现已在亚太地区（孟买）区域 (ap-south-1) 推出，提供 R5 类实例。	2019 年 10 月 17 日
增加了对三个额外的 MongoDB API 的支持	Amazon DocumentDB 增加了对 `$addFields` 、`$concatArrays` 和 `$lookup` MongoDB API 的支持。	2019 年 10 月 16 日
添加了对亚太地区（新加坡）区域的支持	Amazon DocumentDB 现已在亚太地区（新加坡）区域 (ap-southeast-1) 推出，提供 R5 类实例。	2019 年 10 月 14 日
增加了有关更新 TLS 证书的新文档	增加了有关更新 CA 证书以使用新的 CA 证书来创建 TLS 连接的说明。	2019 年 10 月 2 日
增加了对证书的 API 支持	Amazon DocumentDB 增加了用于实例的新证书数据类型。有关更多信息，请参阅 DBInstance。	2019 年 10 月 1 日
支持查询分析	Amazon DocumentDB 增加了对集群的实例和数据库上支持的操作进行分析的功能。	2019 年 8 月 19 日
已在亚太地区（东京）增加第三个 AZ	Amazon DocumentDB 在亚太地区（东京）为您的计算实例增加了第三个可用区 (AZ)。	2019 年 8 月 9 日

支持其他 Mongo API	增加了对其他聚合管道功能的支持，包括 `$in`、`$isoWeek`、`$isoWeekYear` 、`$isoDayOfWeek` 、和 `$dateToString` 聚合运算符以及 `$addToSet` 聚合阶段。Amazon DocumentDB 还增加了对集合级别诊断 `top()` 命令的支持，以及使用 `collMod()` 命令修改 TTL 索引 `expireAfterSeconds` 参数的功能。	2019 年 7 月 31 日
增加了对欧洲地区（伦敦）的支持	Amazon DocumentDB 现已在欧洲（伦敦）(eu-west-2) 推出，提供 R5 类实例。	2019 年 7 月 18 日
添加了代码示例	在 R 和 Ruby 中添加了代码示例，用于以编程方式连接到 Amazon DocumentDB。	2023 年 7 月 17 日
添加了最佳实践	添加了最佳实践，以帮助您管理 Amazon DocumentDB 成本。	2023 年 7 月 17 日
支持停止和启动集群	Amazon DocumentDB 增加了对于停止和启动集群以控制开发和测试环境成本的支持。	2019 年 7 月 1 日

支持集群删除保护	为了防止您的集群遭到意外删除，Amazon DocumentDB 添加了删除保护功能。有关更多信息，请参阅以下主题：创建 Amazon DocumentDB 集群、修改 Amazon DocumentDB 集群、删除 Amazon DocumentDB 集群以及 AIP 主题 DBCluster 中的 `DeletionProtection` 。	2019 年 7 月 1 日
功能差异更新	向功能差异添加了隐式事务。	2019 年 6 月 26 日
功能差异添加	添加了有关 Amazon DocumentDB 中的存储和索引压缩的注释。	2019 年 6 月 13 日
增加了支持区域	Amazon DocumentDB 现已在亚太地区（悉尼）(ap-southeast-2) 推出，提供 R5 类实例。	2019 年 6 月 5 日
更多区域支持 R5 实例类	为以下 4 个区域增加了 R5 实例类支持：美国东部（俄亥俄）、美国东部（弗吉尼亚北部）、美国西部（俄勒冈）和欧洲（爱尔兰）。通过此次更改，支持 Amazon DocumentDB 的所有区域现都支持 R5 实例。	2019 年 5 月 17 日

支持的其他区域	添加了对 2 个其他区域的支持：亚太地区（东京) (ap-northeast-1) 和亚太地区（首尔）(ap-northeast-2)，提供 R5 实例类。有关更多信息，请参阅按区域支持的实例类和 实例类规格。	2019 年 5 月 8 日
添加了更多连接代码示例	添加了 Java 和 C# 的代码示例以连接到 Amazon DocumentDB。	2019 年 4 月 24 日
其他 Mongo API 支持	添加了对七个聚合字符串运算符（$indexOfBytes 、$indexOfCP 、$strLenBytes 、$strLenCP 、$toLower、$toUpper 和 $split）、九个日期时间运算符（$dayOfYear 、$dayOfMonth 、$dayOfWeek 、$year、$month、$hour、$minute、$second 和 $millisecond ）和 $sample 聚合管道分段的支持。	2019 年 4 月 4 日
添加了连接代码示例	在 Python、 Node.js、PHP 和 Go 中添加了用于连接亚马逊 DocumentDB 的代码示例。	2019 年 3 月 21 日
对法兰克福区域和 R5 实例的支持	添加了对欧洲地区（法兰克福）区域 (eu-central-1) 的支持，提供 R5 实例类。有关更多信息，请参阅 按区域支持的实例类和 实例类规格。	2019 年 3 月 13 日

聚合管道运算符支持	添加了对新的聚合字符串运算符（$concat、$substr、$substrBytes 、$substrCP 、$strcasecmp ）、数组聚合运算符 ($size)、聚合组累计器运算符 ($push) 和聚合阶段（$redact 和 $indexStats ）的支持。我们还添加了对位置数组运算符（$[] 和 $[<identifier>] ）和 hint() 的支持。	2019 年 2 月 28 日
引擎升级	添加了确定待处理的集群修改和升级集群引擎版本的文档。	2019 年 2 月 15 日
审核事件	增加了对使用 Amazon CloudWatch 日志审计数据库事件的支持。	2019 年 2 月 12 日
快速入门	添加了一个快速入门主题，以帮助您轻松开始使用 Amazon DocumentDB。 CloudFormation	2019 年 1 月 11 日
公开发布版	这是 Amazon DocumentDB（与 MongoDB 兼容）的首次公开发布。本版本包括开发人员指南和集成的资源管理 API 参考。	2019 年 1 月 9 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。