

---

# Amazon Keyspaces (for Apache Cassandra) Developer Guide



## **Amazon Keyspaces (for Apache Cassandra): Developer Guide**

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

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

## Table of Contents

What is Amazon Keyspaces? .....	1
How it works .....	1
High-level architecture .....	1
Cassandra data model .....	3
Accessing Amazon Keyspaces .....	4
Use cases .....	4
What is CQL? .....	5
Compare Amazon Keyspaces with Cassandra .....	6
Functional differences with Apache Cassandra .....	6
Apache Cassandra APIs, operations, and data types .....	7
Asynchronous creation and deletion of keyspaces and tables .....	7
Authentication and authorization .....	7
Batch .....	7
Cluster configuration .....	7
CQL query throughput tuning .....	7
Empty strings .....	7
Lightweight transactions .....	8
Load balancing .....	8
Pagination .....	8
Prepared statements .....	8
Range delete .....	8
System tables .....	8
Supported Cassandra APIs, operations, and data types .....	8
Cassandra API support .....	9
Cassandra control plane API support .....	10
Cassandra data plane API support .....	10
Cassandra data type support .....	10
Supported Cassandra consistency levels .....	11
Write consistency levels .....	12
Read consistency levels .....	12
Unsupported consistency levels .....	12
Accessing Amazon Keyspaces .....	14
Signing up for AWS .....	14
Setting up AWS Identity and Access Management .....	14
Using the console .....	15
Connecting programmatically .....	15
Creating credentials .....	16
Service endpoints .....	17
Using <code>cqlsh</code> .....	19
Using a Cassandra client driver .....	21
Using a Cassandra Java client driver .....	22
Using a Cassandra Python client driver .....	29
Using a Cassandra Node.js client driver .....	31
Using a Cassandra .NET Core client driver .....	34
Using a Cassandra Go client driver .....	35
Using a Cassandra Perl client driver .....	38
Getting started .....	40
Prerequisites .....	40
Step 1: Create a keyspace and a table .....	41
Creating a keyspace .....	41
Creating a table .....	42
Step 2: CRUD operations .....	45
Create .....	45
Read .....	46

Update .....	48
Delete .....	49
Step 3: Clean up (optional) .....	50
Deleting a table .....	50
Deleting a keyspace .....	51
Migrating to Amazon Keyspaces .....	53
Loading data using cqlsh .....	53
Prerequisites .....	54
Step 1: Create source and target .....	54
Step 2: Prepare the data .....	55
Step 3: Set throughput capacity for the table .....	56
Step 4: Configure cqlsh COPY FROM settings .....	57
Step 5: Run the cqlsh COPY FROM command .....	59
Troubleshooting .....	60
Loading data using DSBulk .....	61
Prerequisites .....	61
Step 1: Create source and target .....	63
Step 2: Prepare the data .....	64
Step 3: Set throughput capacity for the table .....	65
Step 4: Configure DSBulk settings .....	66
Step 5: Run the DSBulk load command .....	67
Code examples and tools .....	69
Libraries and examples .....	69
Amazon Keyspaces (for Apache Cassandra) developer toolkit .....	69
Amazon Keyspaces (for Apache Cassandra) examples .....	69
AWS Signature Version 4 (SigV4) authentication plugins .....	69
Highlighted sample and developer tool repos .....	70
AWS CloudFormation template to create Amazon CloudWatch dashboard for Amazon Keyspaces (for Apache Cassandra) metrics .....	70
Using Amazon Keyspaces (for Apache Cassandra) with AWS Lambda .....	70
Using Amazon Keyspaces (for Apache Cassandra) with Spring .....	70
Using Amazon Keyspaces (for Apache Cassandra) with Scala .....	70
Amazon Keyspaces (for Apache Cassandra) Cassandra query language (CQL) to AWS CloudFormation converter .....	71
Amazon Keyspaces (for Apache Cassandra) helpers for Apache Cassandra driver for Java .....	71
Amazon Keyspaces (for Apache Cassandra) snappy compression demo .....	71
Amazon Keyspaces (for Apache Cassandra) and Amazon S3 codec demo .....	71
Serverless resource management .....	72
Storage .....	72
Read/write capacity modes .....	72
On-demand capacity mode .....	73
Provisioned throughput capacity mode .....	74
Managing and viewing capacity modes .....	75
Considerations when changing capacity modes .....	76
Managing throughput capacity with Application Auto Scaling .....	76
How Amazon Keyspaces automatic scaling works .....	77
Usage notes .....	78
Using the console .....	78
Managing programmatically .....	81
Working with Amazon Keyspaces .....	87
Working with keyspaces .....	87
Creating keyspaces .....	87
Working with tables .....	88
Creating tables .....	88
Static columns .....	88
Working with rows .....	91
Calculating row size .....	91

Working with queries .....	93
Ordering results .....	93
Paginating results .....	93
Data modeling .....	95
Partition key design .....	95
Write sharding .....	95
NoSQL Workbench .....	97
Download .....	97
Getting started .....	98
Data modeler .....	99
Creating a data model .....	99
Editing a data model .....	101
Data visualizer .....	102
Visualizing a Data Model .....	102
Aggregate View .....	104
Committing a data model .....	105
Before you begin .....	106
Connecting with service-specific credentials .....	107
Connecting with IAM credentials .....	108
Using a saved connection .....	110
Apache Cassandra .....	111
Sample data models .....	112
Employee data model .....	113
Credit card transactions data model .....	113
Airline operations data model .....	113
Release history .....	113
Point-in-time recovery .....	115
How it works .....	115
Enabling PITR .....	115
Restore Permissions .....	117
Continuous backups .....	118
Restore settings .....	119
PITR and encrypted tables .....	119
Table restore time .....	120
Integration with AWS services .....	120
Restoring a table to a point in time .....	121
Before you begin .....	121
Restoring a table to a point in time (console) .....	121
Restoring a table to a point in time with CQL .....	122
Restoring a deleted table with CQL .....	123
AWS CloudFormation resources .....	125
Amazon Keyspaces and AWS CloudFormation templates .....	125
Learn more about AWS CloudFormation .....	125
Tagging resources .....	126
Tagging restrictions .....	126
Tagging operations .....	127
Adding tags to new or existing keyspace and tables using the console .....	127
Adding tags to new or existing keyspace and tables using CQL .....	128
Cost allocation reports for Amazon Keyspaces .....	129
Security .....	131
Data protection .....	131
Encryption at rest .....	132
Encryption in transit .....	145
Internetwork traffic privacy .....	146
AWS Identity and Access Management .....	146
Audience .....	147
Authenticating with identities .....	147

Managing access using policies .....	149
How Amazon Keyspaces works with IAM .....	151
Identity-based policy examples .....	154
AWS managed policies .....	160
Troubleshooting .....	163
Using service-linked roles .....	166
Logging and monitoring .....	167
Monitoring tools .....	168
Monitoring with CloudWatch .....	169
Logging Amazon Keyspaces API calls with AWS CloudTrail .....	180
Compliance validation .....	184
Resilience .....	185
Infrastructure security .....	185
Using interface VPC endpoints .....	186
Configuration and vulnerability analysis for Amazon Keyspaces .....	189
Security best practices .....	190
Preventative security best practices .....	190
Detective security best practices .....	191
CQL language reference .....	193
Language elements .....	193
Identifiers .....	193
Constants .....	193
Terms .....	194
Data types .....	194
JSON encoding of Amazon Keyspaces data types .....	196
DDL statements .....	197
Keyspaces .....	198
Tables .....	199
DML statements .....	204
SELECT .....	204
INSERT .....	205
UPDATE .....	206
DELETE .....	207
Built-in functions .....	207
Scalar functions .....	207
Quotas .....	209
Amazon Keyspaces service quotas .....	209
Increasing or decreasing throughput (for provisioned tables) .....	211
Increasing provisioned throughput .....	211
Decreasing provisioned throughput .....	211
Amazon Keyspaces encryption at rest .....	211
Document history .....	212

# What is Amazon Keyspaces (for Apache Cassandra)?

Amazon Keyspaces (for Apache Cassandra) is a scalable, highly available, and managed Apache Cassandra-compatible database service. With Amazon Keyspaces, you don't have to provision, patch, or manage servers, and you don't have to install, maintain, or operate software.

Amazon Keyspaces is serverless, so you pay for only the resources that you use, and the service automatically scales tables up and down in response to application traffic. You can build applications that serve thousands of requests per second with virtually unlimited throughput and storage.

## Note

Apache Cassandra is an open-source, wide-column datastore that is designed to handle large amounts of data. For more information, see [Apache Cassandra](#).

Amazon Keyspaces makes it easy to migrate, run, and scale Cassandra workloads in the AWS Cloud. With just a few clicks on the AWS Management Console or a few lines of code, you can create keyspace and tables in Amazon Keyspaces, without deploying any infrastructure or installing software.

With Amazon Keyspaces, you can run your existing Cassandra workloads on AWS using the same Cassandra application code and developer tools that you use today.

For a list of available AWS Regions and endpoints, see [Service endpoints for Amazon Keyspaces](#).

We recommend that you start by reading the following sections:

## Topics

- [Amazon Keyspaces: How it works \(p. 1\)](#)
- [Amazon Keyspaces use cases \(p. 4\)](#)
- [What is Cassandra Query Language \(CQL\)? \(p. 5\)](#)

## Amazon Keyspaces: How it works

Amazon Keyspaces removes the administrative overhead of managing Cassandra. To understand why, it's helpful to begin with Cassandra architecture and then compare it to Amazon Keyspaces.

## Topics

- [High-level architecture: Apache Cassandra vs. Amazon Keyspaces \(p. 1\)](#)
- [Cassandra data model \(p. 3\)](#)
- [Accessing Amazon Keyspaces from an application \(p. 4\)](#)

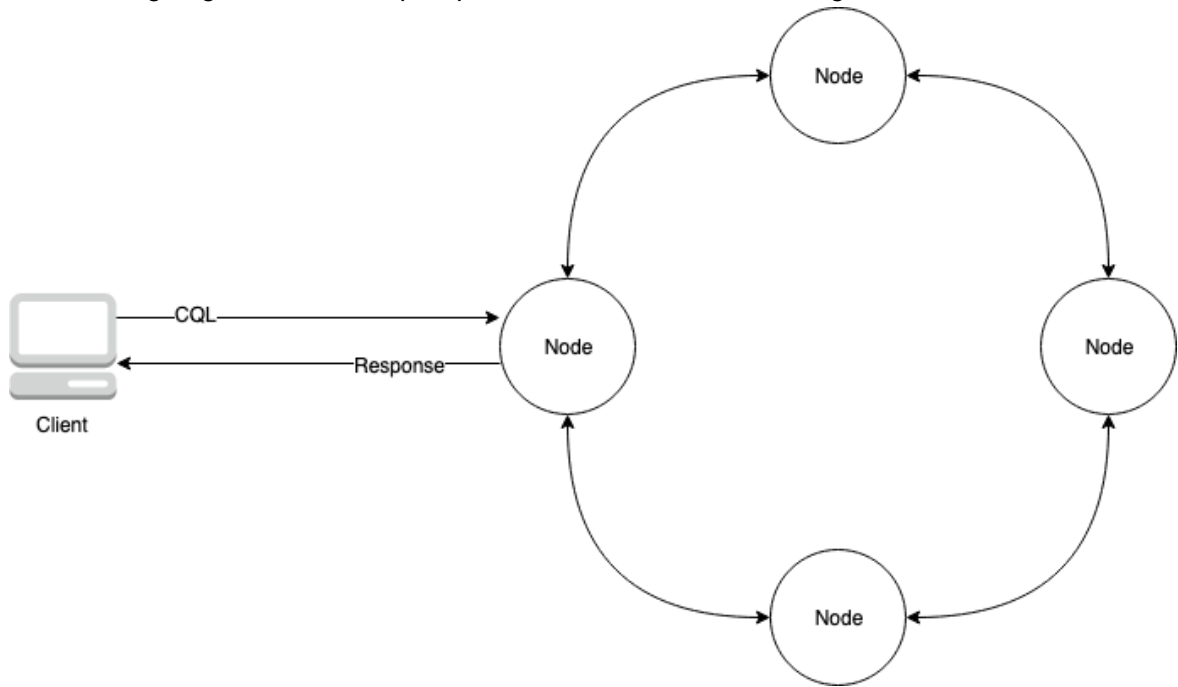
## High-level architecture: Apache Cassandra vs. Amazon Keyspaces

Traditional Apache Cassandra is deployed in a cluster made up of one or more nodes. You are responsible for managing each node and adding and removing nodes as your cluster scales.

A client program accesses Cassandra by connecting to one of the nodes and issuing Cassandra Query Language (CQL) statements. CQL is similar to SQL, the popular language used in relational databases.

Even though Cassandra is not a relational database, CQL provides a familiar interface for querying and manipulating data in Cassandra.

The following diagram shows a simple Apache Cassandra cluster, consisting of four nodes.



A production Cassandra deployment might consist of hundreds of nodes, running on hundreds of physical computers across one or more physical data centers. This can cause an operational burden for application developers who need to provision, patch, and manage servers in addition to installing, maintaining, and operating software.

With Amazon Keyspaces (for Apache Cassandra), you don't need to provision, patch, or manage servers, so you can focus on building better applications. Amazon Keyspaces offers two throughput capacity modes for reads and writes: on-demand and provisioned. You can choose your table's throughput capacity mode to optimize the price of reads and writes based on the predictability and variability of your workload.

With on-demand mode, you pay for only the reads and writes that your application actually performs. You do not need to specify your table's throughput capacity in advance. Amazon Keyspaces accommodates your application traffic almost instantly as it ramps up or down, making it a good option for applications with unpredictable traffic.

Provisioned capacity mode helps you optimize the price of throughput if you have predictable application traffic and can forecast your table's capacity requirements in advance. With provisioned capacity mode, you specify the number of reads and writes per second that you expect your application to perform. You can increase and decrease the provisioned capacity for your table automatically by enabling [automatic scaling](#).

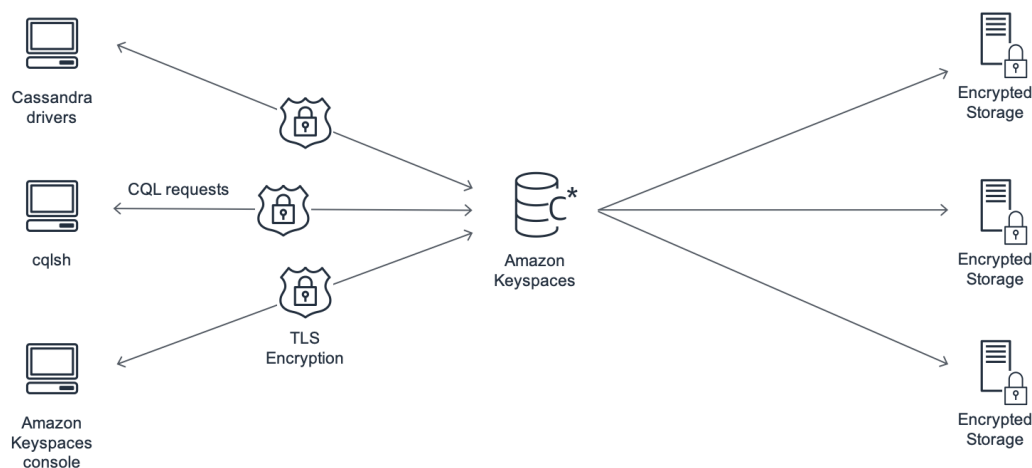
You can change the capacity mode of your table once per day as you learn more about your workload's traffic patterns, or if you expect to have a large burst in traffic, such as from a major event that you anticipate will drive a lot of table traffic. For more information about read and write capacity provisioning, see [the section called "Read/write capacity modes" \(p. 72\)](#).

Amazon Keyspaces (for Apache Cassandra) stores three copies of your data in multiple [Availability Zones](#) for durability and high availability. In addition, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations. Encryption at rest is



automatically enabled when you create a new Amazon Keyspaces table and all client connections require Transport Layer Security (TLS). Additional AWS security features include [monitoring](#), [AWS Identity and Access Management](#), and [virtual private cloud \(VPC\) endpoints](#). For an overview of all available security features, see [Security](#) (p. 131).

The following diagram shows the architecture of Amazon Keyspaces.



A client program accesses Amazon Keyspaces by connecting to a predetermined endpoint (hostname and port number) and issuing CQL statements. For a list of available endpoints, see [the section called "Service endpoints"](#) (p. 17).

## Cassandra data model

How you model your data for your business case is critical to achieving optimal performance from Amazon Keyspaces. A poor data model can significantly degrade performance.

Even though CQL looks similar to SQL, the backends of Cassandra and relational databases are very different and must be approached differently. The following are some of the more significant issues to consider:

### Storage

You can visualize your Cassandra data in tables, with each row representing a record and each column a field within that record.

#### Table design: Query first

There are no `JOINS` in CQL. Therefore, you should design your tables with the shape of your data and how you need to access it for your business use cases. This might result in de-normalization with duplicated data. You should design each of your tables specifically for a particular access pattern.

### Partitions

Your data is stored in partitions on disk. The number of partitions your data is stored in and how it is distributed across the partitions is determined by your *partition key*. How you define your partition key can have a significant impact upon the performance of your queries.

### Primary key

In Cassandra, data is stored as a key-value pair. To that end, every Cassandra table must have a primary key, which is the key to each row in the table. The primary key is the composite of a required partition key and optional clustering columns. The data that comprises the primary key must be unique across all records in a table.

- **Partition key** – The partition key portion of the primary key is required and determines which partition of your cluster the data is stored in. The partition key can be a single column, or it can be a compound value composed of two or more columns. You would use a compound partition key if a single column partition key would result in a single partition or a very few partitions having most of the data and thus bearing the majority of the disk I/O operations.
- **Clustering column** – The optional clustering column portion of your primary key determines how the data is clustered and sorted within each partition. If you include a clustering column in your primary key, the clustering column can have one or more columns. If there are multiple columns in the clustering column, the sorting order is determined by the order that the columns are listed in the clustering column, from left to right.

## Accessing Amazon Keyspaces from an application

Amazon Keyspaces (for Apache Cassandra) implements the Apache Cassandra Query Language (CQL) API, so you can use CQL and Cassandra drivers that you already use. Updating your application is as easy as updating your Cassandra driver or `cqlsh` configuration to point to the Amazon Keyspaces service endpoint.

### Note

To help you get started, you can find end-to-end code samples of connecting to Amazon Keyspaces by using various Cassandra client drivers in the Amazon Keyspaces code example repository on [GitHub](#).

Consider the following Python program, which connects to a Cassandra cluster and queries a table.

```
from cassandra.cluster import Cluster
#TLS/SSL configuration goes here

ksp = 'MyKeyspace'
tbl = 'WeatherData'

cluster = Cluster(['NNN.NNN.NNN.NNN'], port=NNNN)
session = cluster.connect(ksp)

session.execute('USE ' + ksp)

rows = session.execute('SELECT * FROM ' + tbl)
for row in rows:
    print(row)
```

To run the same program against Amazon Keyspaces, you need to:

- **Add the cluster endpoint and port:** For example, the host can be replaced with a service endpoint, such as `cassandra.us-east-2.amazonaws.com` and the port number with: `9142`.
- **Add the TLS/SSL configuration:** For more information on adding the TLS/SSL configuration to connect to Amazon Keyspaces by using a Cassandra client Python driver, see [Using a Cassandra Python client driver to access Amazon Keyspaces programmatically](#) (p. 29).

## Amazon Keyspaces use cases

The following are just some of the ways in which you can use Amazon Keyspaces:

- **Build applications that require low latency** – Process data at high speeds for applications that require single-digit-millisecond latency, such as industrial equipment maintenance, trade monitoring, fleet management, and route optimization.

- **Build applications using open-source technologies** – Build applications on AWS using open-source Cassandra APIs and drivers that are available for a wide range of programming languages, such as Java, Python, Ruby, Microsoft .NET, Node.js, PHP, C++, Perl, and Go. For code examples, see [Code examples and tools](#) (p. 69).
- **Move your Cassandra workloads to the cloud** – Managing Cassandra tables yourself is time-consuming and expensive. With Amazon Keyspaces, you can set up, secure, and scale Cassandra tables in the AWS Cloud without managing infrastructure. For more information, see [Serverless resource management](#) (p. 72).

## What is Cassandra Query Language (CQL)?

*Cassandra Query Language* (CQL) is the primary language for communicating with Apache Cassandra. Amazon Keyspaces (for Apache Cassandra) is compatible with the CQL 3.x API (backward-compatible with version 2.x).

To run CQL queries, you can do one of the following:

- Use the CQL editor on the AWS Management Console.
- Run them on the `cqlsh` client.
- Run them programmatically using an Apache 2.0 licensed Cassandra client driver.

For more information about using these methods to access Amazon Keyspaces, see [Accessing Amazon Keyspaces \(for Apache Cassandra\)](#) (p. 14).

For more information about CQL, see [CQL language reference for Amazon Keyspaces \(for Apache Cassandra\)](#) (p. 193).

# How does Amazon Keyspaces (for Apache Cassandra) compare to Apache Cassandra?

Amazon Keyspaces (for Apache Cassandra) appears as a nine-node, Apache Cassandra 3.11.2 cluster to clients and supports drivers and clients that are compatible with Apache Cassandra 3.11.2. Amazon Keyspaces supports the 3.x Cassandra Query Language (CQL) API and is backward-compatible with version 2.x. With Amazon Keyspaces, you can run your Cassandra workloads on AWS using the same Cassandra application code, Apache 2.0–licensed drivers, and tools that you use today.

Amazon Keyspaces supports all commonly used Cassandra data-plane operations, such as creating keyspaces and tables, reading data, and writing data. Amazon Keyspaces is serverless, so you don't have to provision, patch, or manage servers. You also don't have to install, maintain, or operate software. As a result, the Cassandra control plane API operations to manage cluster and node settings are not required to use Amazon Keyspaces.

Settings such as replication factor and consistency level are configured automatically to provide you with high availability, durability, and single-digit-millisecond performance.

## Topics

- [Functional differences: Amazon Keyspaces vs. Apache Cassandra \(p. 6\)](#)
- [Supported Cassandra APIs, operations, and data types in Amazon Keyspaces \(p. 8\)](#)
- [Supported Apache Cassandra consistency levels in Amazon Keyspaces \(p. 11\)](#)

## Functional differences: Amazon Keyspaces vs. Apache Cassandra

The following are the functional differences between Amazon Keyspaces and Apache Cassandra.

## Topics

- [Apache Cassandra APIs, operations, and data types \(p. 7\)](#)
- [Asynchronous creation and deletion of keyspaces and tables \(p. 7\)](#)
- [Authentication and authorization \(p. 7\)](#)
- [Batch \(p. 7\)](#)
- [Cluster configuration \(p. 7\)](#)
- [CQL query throughput tuning \(p. 7\)](#)
- [Empty strings \(p. 7\)](#)
- [Lightweight transactions \(p. 8\)](#)
- [Load balancing \(p. 8\)](#)
- [Pagination \(p. 8\)](#)
- [Prepared statements \(p. 8\)](#)
- [Range delete \(p. 8\)](#)
- [System tables \(p. 8\)](#)

## Apache Cassandra APIs, operations, and data types

Amazon Keyspaces supports all commonly used Cassandra data-plane operations, such as creating keyspaces and tables, reading data, and writing data. To see what is currently supported, see [Supported Cassandra APIs, operations, and data types in Amazon Keyspaces \(p. 8\)](#).

## Asynchronous creation and deletion of keyspaces and tables

Amazon Keyspaces performs data definition language (DDL) operations, such as creating and deleting keyspaces and tables, asynchronously. To learn how to monitor the creation status of resources, see [the section called "Creating keyspaces" \(p. 87\)](#) and [the section called "Creating tables" \(p. 88\)](#). For a list of DDL statements in the CQL language reference, see [the section called "DDL statements" \(p. 197\)](#).

## Authentication and authorization

Amazon Keyspaces (for Apache Cassandra) uses AWS Identity and Access Management (IAM) for user authentication and authorization, and supports the equivalent authorization policies as Apache Cassandra. As such, Amazon Keyspaces does not support Apache Cassandra's security configuration commands.

## Batch

Amazon Keyspaces supports unlogged batch commands with up to 30 commands in the batch. Only unconditional **INSERT**, **UPDATE**, or **DELETE** commands are permitted in a batch. Logged batches are not supported.

## Cluster configuration

Amazon Keyspaces is serverless, so there are no clusters, hosts, or Java virtual machines (JVMs) to configure. Cassandra's settings for compaction, compression, caching, garbage collection, and bloom filtering are not applicable to Amazon Keyspaces and are ignored if specified.

## CQL query throughput tuning

Amazon Keyspaces supports up to 3,000 CQL queries per TCP connection per second, but there is no limit on the number of connections a driver can establish.

Most open-source Cassandra drivers establish a connection pool to Cassandra and load balance queries over that pool of connections. Amazon Keyspaces exposes 9 peer IP addresses to drivers, and the default behavior of most drivers is to establish a single connection to each peer IP address. Therefore, the maximum CQL query throughput of a driver using the default settings will be 27,000 CQL queries per second.

To increase this number, we recommend increasing the number of connections per IP address your driver is maintaining in its connection pool. For example, setting the maximum connections per IP address to 2 will double the maximum throughput of your driver to 54,000 CQL queries per second.

## Empty strings

Amazon Keyspaces supports empty strings and blob values. However, empty strings and blobs are not supported as clustering column values.

## Lightweight transactions

Amazon Keyspaces (for Apache Cassandra) fully supports compare and set functionality on **INSERT** and **UPDATE** commands, which are known as *lightweight transactions* (LWTs) in Apache Cassandra. As a serverless offering, Amazon Keyspaces (for Apache Cassandra) provides consistent performance at any scale, including for lightweight transactions. With Amazon Keyspaces, there is no performance penalty for using lightweight transactions.

## Load balancing

The `system.peers` table entries correspond to Amazon Keyspaces load balancers. For best results, we recommend using a round robin load-balancing policy and tuning the number of connections per IP to suit your applications needs.

## Pagination

Amazon Keyspaces paginates results based on the number of rows that it reads to process a request, not the number of rows returned in the result set. As a result, some pages might contain fewer rows than you specify in `PAGE SIZE` for filtered queries. In addition, Amazon Keyspaces paginates results automatically after reading 1 MBof data to provide customers with consistent, single-digit millisecond read performance. For more information, see [the section called "Paginating results" \(p. 93\)](#).

## Prepared statements

Amazon Keyspaces supports the use of prepared statements for data manipulation language (DML) operations, such as reading and writing data. Amazon Keyspaces does not currently support the use of prepared statements for data definition language (DDL) operations, such as creating tables and keyspaces. DDL operations must be run outside of prepared statements.

## Range delete

Amazon Keyspaces supports deleting rows in range. A range is a contiguous set of rows within a partition. You specify a range in a `DELETE` operation by using a `WHERE` clause. You can specify the range to be an entire partition.

Furthermore, you can specify a range to be a subset of contiguous rows within a partition by using relational operators (for example, `'>'`, `'<'`), or by including the partition key and omitting one or more clustering columns. With Amazon Keyspaces, you can delete up to 1,000 rows within a range in a single operation. Additionally, range deletes are atomic, but not isolated.

## System tables

Amazon Keyspaces populates the system tables required by Apache 2.0 open source Cassandra drivers. The system tables visible to a client contain information unique to the authenticated user. The system tables are fully controlled by Amazon Keyspaces and are read-only.

# Supported Cassandra APIs, operations, and data types in Amazon Keyspaces

Amazon Keyspaces (for Apache Cassandra) is compatible with Cassandra Query Language (CQL) 3.11 API (backward-compatible with version 2.x).

Amazon Keyspaces supports all commonly used Cassandra data-plane operations, such as creating keyspaces and tables, reading data, and writing data.

The following sections list the supported functionality.

### Topics

- [Cassandra API support \(p. 9\)](#)
- [Cassandra control plane API support \(p. 10\)](#)
- [Cassandra data plane API support \(p. 10\)](#)
- [Cassandra data type support \(p. 10\)](#)

## Cassandra API support

API operation	Supported
CREATE KEYSPACE	Yes
ALTER KEYSPACE	Yes
DROP KEYSPACE	Yes
CREATE TABLE	Yes
ALTER TABLE	Yes
DROP TABLE	Yes
CREATE INDEX	No
DROP INDEX	No
UNLOGGED BATCH	Yes
LOGGED BATCH	No
SELECT	Yes
INSERT	Yes
DELETE	Yes
UPDATE	Yes
USE	Yes
CREATE TYPE	No
ALTER TYPE	No
DROP TYPE	No
CREATE TRIGGER	No
DROP TRIGGER	No
CREATE FUNCTION	No
DROP FUNCTION	No

API operation	Supported
CREATE AGGREGATE	No
DROP AGGREGATE	No
CREATE MATERIALIZED VIEW	No
ALTER MATERIALIZED VIEW	No
DROP MATERIALIZED VIEW	No
TRUNCATE	No

## Cassandra control plane API support

Because Amazon Keyspaces is managed, the Cassandra control plane API operations to manage cluster and node settings are not required. As a result, the following Cassandra features are not applicable.

Feature	Reason
Durable writes toggle	All writes are durable
Read repair settings	Not applicable
GC grace seconds	Not applicable
Bloom filter settings	Not applicable
Compaction settings	Not applicable
Compression settings	Not applicable
Caching settings	Not applicable
Security settings	Replaced by IAM

## Cassandra data plane API support

Feature	Supported
Static columns	Yes
Time to Live (TTL)	No
JSON support for SELECT and INSERT statements	Yes

## Cassandra data type support

Data type	Supported
ascii	Yes



Data type	Supported
bigint	Yes
blob	Yes
boolean	Yes
counter	Yes
date	Yes
decimal	Yes
double	Yes
float	Yes
frozen	No
inet	Yes
int	Yes
list	Yes
map	Yes
set	Yes
smallint	Yes
text	Yes
time	Yes
timestamp	Yes
timeuuid	Yes
tinyint	Yes
tuple	Yes
user-defined types (UDT)	No
uuid	Yes
varchar	Yes
varint	Yes

## Supported Apache Cassandra consistency levels in Amazon Keyspaces

The topics in this section describe which Apache Cassandra consistency levels are supported for read and write operations in Amazon Keyspaces (for Apache Cassandra).

### Topics

- [Write consistency levels \(p. 12\)](#)
- [Read consistency levels \(p. 12\)](#)
- [Unsupported consistency levels \(p. 12\)](#)

## Write consistency levels

Amazon Keyspaces replicates all write operations three times across multiple Availability Zones for durability and high availability. Writes are durably stored before they are acknowledged using the `LOCAL_QUORUM` consistency level. For each 1 KB write, you are billed 1 write capacity unit (WCU) for tables using provisioned capacity mode or 1 write request unit (WRU) for tables using on-demand mode.

## Read consistency levels

Amazon Keyspaces supports three read consistency levels: `ONE`, `LOCAL_ONE`, and `LOCAL_QUORUM`. During a `LOCAL_QUORUM` read, Amazon Keyspaces returns a response reflecting the most recent updates from all prior successful write operations. Using the consistency level `ONE` or `LOCAL_ONE` can improve the performance and availability of your read requests, but the response might not reflect the results of a recently completed write.

For each 4 KB read using `ONE` or `LOCAL_ONE` consistency, you are billed 0.5 read capacity units (RCUs) for tables using provisioned capacity mode or 0.5 read request units (RRUs) for tables using on-demand mode. For each 4 KB read using `LOCAL_QUORUM` consistency, you are billed 1 read capacity unit (RCU) for tables using provisioned capacity mode or 1 read request units (RRU) for tables using on-demand mode.

### Billing based on read consistency and read capacity throughput mode per table for each 4 KB of reads

Consistency level	Provisioned	On-demand
<code>ONE</code>	0.5 RCUs	0.5 RRUs
<code>LOCAL_ONE</code>	0.5 RCUs	0.5 RRUs
<code>LOCAL_QUORUM</code>	1 RCU	1 RRU

## Unsupported consistency levels

The following consistency levels are not supported by Amazon Keyspaces and will result in exceptions.

### Unsupported consistency levels

Apache Cassandra	Amazon Keyspaces
<code>EACH_QUORUM</code>	Not supported
<code>QUORUM</code>	Not supported
<code>ALL</code>	Not supported
<code>TWO</code>	Not supported
<code>THREE</code>	Not supported
<code>ANY</code>	Not supported

Apache Cassandra	Amazon Keyspaces
SERIAL	Not supported
LOCAL_SERIAL	Not supported

# Accessing Amazon Keyspaces (for Apache Cassandra)

You can access Amazon Keyspaces using the console, or programmatically by running a `cqlsh` client, or using an Apache 2.0 licensed Cassandra driver. Amazon Keyspaces supports drivers and clients that are compatible with Apache Cassandra 3.11.2. Before accessing Amazon Keyspaces, you must complete the following two steps:

1. [Signing up for AWS \(p. 14\)](#)
2. [Setting up AWS Identity and Access Management \(p. 14\)](#)

## Signing up for AWS

To use the Amazon Keyspaces service, you must have an AWS account. If you don't already have an account, you are prompted to create one when you sign up. You're not charged for any AWS services that you sign up for unless you use them.

### To sign up for AWS

1. Open <https://portal.aws.amazon.com/billing/signup>.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

## Setting up AWS Identity and Access Management

Access to Amazon Keyspaces resources is managed using [AWS Identity and Access Management \(IAM\)](#). Using IAM, you can attach policies to IAM users, roles, and federated identities that grant read and write permissions to specific resources. For example, you can grant an IAM user read-only access to only a subset of keyspace and tables.

The following example IAM policy grants full read and write access to your Amazon Keyspaces resources, which is only recommended for trials. For sample policies following security guidelines, see [the section called "Accessing Amazon Keyspaces tables" \(p. 155\)](#).

1. Create an AWS Identity and Access Management user.
2. Create and attach the following policy to the user you just created.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cassandra:*"
      ],
    }
  ]
}
```

```
    "Resource": [
      "*"
    ]
  }
]
}
```

To access Amazon Keyspaces after you have created the AWS account and IAM policies, see the following sections:

- [Using the console \(p. 15\)](#)
- [Connecting programmatically \(p. 15\)](#)

## Accessing Amazon Keyspaces using the console

You can access the console for Amazon Keyspaces at <https://console.aws.amazon.com/keyspaces/home>.

You can use the console to do the following in Amazon Keyspaces:

- Create, delete, describe, and list keyspaces and tables.
- Insert, update, and delete data.
- Run queries using the CQL editor.

To learn how to create an Amazon Keyspaces keyspace and table and set it up with sample application data, see [Getting started with Amazon Keyspaces \(for Apache Cassandra\) \(p. 40\)](#).

## Connecting programmatically to Amazon Keyspaces

This topic outlines the steps required to connect to Amazon Keyspaces programmatically. It guides you through creating service-specific credentials and lists the available AWS service endpoints. The last section shows how to connect to Amazon Keyspaces using `cqlsh`. For step-by-step tutorials to connect to Amazon Keyspaces using different Apache Cassandra drivers, see [the section called "Using a Cassandra client driver" \(p. 21\)](#).

### Note

To help you get started, you can find end-to-end code samples of connecting to Amazon Keyspaces by using various Cassandra client drivers in the Amazon Keyspaces code example repository on [GitHub](#).

Amazon Keyspaces supports drivers and clients that are compatible with Apache Cassandra 3.11.2. It assumes that you have already completed the AWS setup instructions in [Accessing Amazon Keyspaces \(p. 14\)](#).

If you already have an AWS account, see the following topics to learn how to access Amazon Keyspaces using `cqlsh` programmatically:

### Topics

- [Creating credentials to access Amazon Keyspaces programmatically \(p. 16\)](#)
- [Service endpoints for Amazon Keyspaces \(p. 17\)](#)
- [Using `cqlsh` to connect to Amazon Keyspaces \(p. 19\)](#)

## Creating credentials to access Amazon Keyspaces programmatically

To provide users and applications with credentials for programmatic access to Amazon Keyspaces resources, you can do either of the following:

- Create service-specific credentials that are associated with a specific AWS Identity and Access Management (IAM) user.
- Use an authentication plugin for temporary credentials. This plugin enables [IAM users, roles, and federated identities](#) to add authentication information to Amazon Keyspaces API requests using the [AWS Signature Version 4 process \(SigV4\)](#).

For code samples that show how to connect using the SigV4 authentication plugin, see [the section called "Using a Cassandra client driver"](#) (p. 21).

### Topics

- [Generate service-specific credentials](#) (p. 16)

## Generate service-specific credentials

Service-specific credentials enable IAM users to access a specific AWS service. The credentials cannot be used to access other AWS services. They are associated with a specific IAM user and cannot be used by other IAM users.

### Important

Service-specific credentials can only be used by IAM users. To give IAM roles or federated identities permissions to access your resources, you should use the authentication plugin to create temporary credentials.

Use one of the following procedures to generate a service-specific credential.

### [Generate service-specific credentials using the console](#)

#### To generate service-specific credentials using the console

1. Sign in to the AWS Management Console and open the AWS Identity and Access Management console at <https://console.aws.amazon.com/iam/home>.
2. In the navigation pane, choose **Users**, and then choose the user that you created earlier that has Amazon Keyspaces permissions (policy attached).
3. Choose **Security Credentials**. Under **Credentials for Amazon Keyspaces**, choose **Generate credentials** to generate the service-specific credentials.

Your service-specific credentials are now available. This is the only time you can download or view the password. You cannot recover it later. However, you can reset your password at any time. Save the user and password in a secure location, because you'll need them later.

### [Generate service-specific credentials using the AWS CLI](#)

#### To generate service-specific credentials using the AWS CLI

Before generating service-specific credentials, you need to download, install, and configure the AWS Command Line Interface (AWS CLI):

1. Download the AWS CLI at <http://aws.amazon.com/cli>.

**Note**

The AWS CLI runs on Windows, macOS, or Linux.

2. Follow the instructions for [Installing the AWS CLI](#) and [Configuring the AWS CLI](#) in the *AWS Command Line Interface User Guide*.
3. Using the AWS CLI, run the following command to generate service-specific credentials for the user *alice*, so that she can access Amazon Keyspaces.

```
aws iam create-service-specific-credential \  
  --user-name alice \  
  --service-name cassandra.amazonaws.com
```

The output looks like the following.

```
{  
  "ServiceSpecificCredential": {  
    "CreateDate": "2019-10-09T16:12:04Z",  
    "ServiceName": "cassandra.amazonaws.com",  
    "ServiceUserName": "alice-at-111122223333",  
    "ServicePassword": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",  
    "ServiceSpecificCredentialId": "ACCAYFI33SINPGJEBYESF",  
    "UserName": "alice",  
    "Status": "Active"  
  }  
}
```

In the output, note the values for *ServiceUserName* and *ServicePassword*. Save these values in a secure location, because you'll need them later.

**Important**

This is the only time that the *ServicePassword* will be available to you.

## Service endpoints for Amazon Keyspaces

Amazon Keyspaces is available in the following AWS Regions. The table shows the service endpoint for each Region.

Region Name	Region	Endpoint	Protocol	
US East (Ohio)	us-east-2	cassandra.us-east-2.amazonaws.com	TLS	
US East (N. Virginia)	us-east-1	cassandra.us-east-1.amazonaws.com	TLS	
US West (N. California)	us-west-1	cassandra.us-west-1.amazonaws.com	TLS	
US West (Oregon)	us-west-2	cassandra.us-west-2.amazonaws.com	TLS	
Asia Pacific (Hong Kong)	ap-east-1	cassandra.ap-east-1.amazonaws.com	TLS	

Amazon Keyspaces (for Apache  
Cassandra) Developer Guide  
Service endpoints

Region Name	Region	Endpoint	Protocol	
Asia Pacific (Mumbai)	ap-south-1	cassandra.ap-south-1.amazonaws.com	TLS	
Asia Pacific (Seoul)	ap-northeast-2	cassandra.ap-northeast-2.amazonaws.com	TLS	
Asia Pacific (Singapore)	ap-southeast-1	cassandra.ap-southeast-1.amazonaws.com	TLS	
Asia Pacific (Sydney)	ap-southeast-2	cassandra.ap-southeast-2.amazonaws.com	TLS	
Asia Pacific (Tokyo)	ap-northeast-1	cassandra.ap-northeast-1.amazonaws.com	TLS	
Canada (Central)	ca-central-1	cassandra.ca-central-1.amazonaws.com	TLS	
Europe (Frankfurt)	eu-central-1	cassandra.eu-central-1.amazonaws.com	TLS	
Europe (Ireland)	eu-west-1	cassandra.eu-west-1.amazonaws.com	TLS	
Europe (London)	eu-west-2	cassandra.eu-west-2.amazonaws.com	TLS	
Europe (Paris)	eu-west-3	cassandra.eu-west-3.amazonaws.com	TLS	
Europe (Stockholm)	eu-north-1	cassandra.eu-north-1.amazonaws.com	TLS	
Middle East (Bahrain)	me-south-1	cassandra.me-south-1.amazonaws.com	TLS	
South America (São Paulo)	sa-east-1	cassandra.sa-east-1.amazonaws.com	TLS	

For the following AWS Regions, FIPS endpoints are available. For more information about FIPS endpoints, see [AWS General Reference FIPS endpoints](#).

Region name	Region	FIPS endpoint	Protocol
US East (N. Virginia)	us-east-1	cassandra-fips.us-east-1.amazonaws.com	TLS



Region name	Region	FIPS endpoint	Protocol
US West (Oregon)	us-west-2	cassandra-fips.us-west-2.amazonaws.com	TLS

The following table shows the authentication mechanism to use for the port.

Port	Authentication mechanism
9142	TLS client authentication, with certificates

## Using `cqlsh` to connect to Amazon Keyspaces

The following section describes how to use `cqlsh` to connect to Amazon Keyspaces (for Apache Cassandra).

For information about `cqlsh`, see [cqlsh: the CQL shell](#).

### Topics

- [Installing and using `cqlsh` to connect to Amazon Keyspaces \(for Apache Cassandra\)](#) (p. 19)

## Installing and using `cqlsh` to connect to Amazon Keyspaces (for Apache Cassandra)

To install and use `cqlsh`, you must do the following:

1. [Install Python 2.7](#) (p. 19).
2. [Install the `cqlsh` client](#) (p. 20).
3. [Encrypt `cqlsh` connections using TLS](#) (p. 20).

### Note

To make `cqlsh` connections to Amazon Keyspaces for functional testing, light operations, and migrations you can use a preconfigured Docker container that includes all prerequisites and configuration settings optimized for Amazon Keyspaces, and is available from <https://github.com/aws-samples/amazon-keyspaces-toolkit>.

### Install Python 2.7

To determine whether you have Python installed on your computer and which version, run the following operation.

```
python --version
```

If you have Python 2.7 installed, you should see something like the following for output.

```
Python 2.7.16
```

If you need to install Python 2.7, follow the instructions at [Python downloads](#).

## Install and configure the CQL client

cqlsh is bundled with Apache Cassandra. To get it, install Apache Cassandra by following the instructions in [Downloading and installing Apache Cassandra](#). Amazon Keyspaces supports drivers and clients that are compatible with Apache Cassandra 3.11.2. The currently recommended version of cqlsh can be downloaded from [Apache](#).

After installing Cassandra, verify that cqlsh is installed by running the following command.

```
cqlsh --version
```

You should see something like the following for output.

```
cqlsh 5.0.1
```

If you are using Windows, replace all instances of cqlsh with cqlsh.bat. For example, to check the version of cqlsh in Windows, run the following command.

```
cqlsh.bat --version
```

Download the configuration file cqlshrc optimized for Amazon Keyspaces from [Github](#). Save the downloaded cqlshrc file to the Cassandra directory.

```
${HOME}/.cassandra/cqlshrc
```

## Encrypting cqlsh connections using TLS

Amazon Keyspaces only accepts secure connections using Transport Layer Security (TLS).

Before you can connect using SSL/TLS, you must do the following:

1. Download the Starfield digital certificate using the following command and save sf-class2-root.crt locally or in your home directory.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

### Note

You can also use the Amazon digital certificate to connect to Amazon Keyspaces and can continue to do so if your client is connecting to Amazon Keyspaces successfully. The Starfield certificate provides additional backwards compatibility for clients using older certificate authorities.

2. Connect to Amazon Keyspaces with the following command.

### Important

The ServiceUserName and ServicePassword should match the ones obtained when you generated the service-specific credentials by following the steps in [Generate service-specific credentials](#) (p. 16).

You can also manage Amazon Keyspaces cqlsh access through AWS IAM users and roles by using the AWS authentication plugin expansion for cqlsh. To learn more, see [Amazon Keyspaces \(for Apache Cassandra\) developer toolkit on Github](#).

```
cqlsh host 9142 -u ServiceUserName -p ServicePassword --ssl
```

Note that 9142 is the secure port.

The following is an example.

```
cqlsh cassandra.us-east-2.amazonaws.com 9142 -u "alice-at-111122223333" -  
p "wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY" --ssl
```

After connecting, you see something like the following for output. The currently supported version of Apache Cassandra is 3.11.2.

```
Connected to Amazon Keyspaces at cassandra.us-east-2.amazonaws.com:9142.  
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]  
Use HELP for help.  
alice-at-111122223333@cqlsh>
```

### Updating an existing configuration file for cqlsh connections

If you want to edit an existing configuration file to support TLS connections, open the configuration file in the Cassandra home directory, for example `${HOME}/.cassandra/cqlshrc` and add the following lines.

```
[connection]  
port = 9142  
factory = cqlshlib.ssl.ssl_transport_factory  
  
[ssl]  
validate = true  
certfile = path_to_file/sf-class2-root.crt
```

You can configure cqlsh COPY settings to ensure cqlsh stays within the Amazon Keyspaces [the section called "CQL query throughput tuning" \(p. 7\)](#) guidelines.

Modify the default for the COPY FROM option in the configuration file `${HOME}/.cassandra/cqlshrc` and add the following lines.

```
[copy-from]  
CHUNKSIZE=50
```

This setting for `CHUNKSIZE` works well to get started with a newly created table and should be changed to support larger workloads. For more information on how to optimize cqlsh COPY configuration settings for Amazon Keyspaces, see [the section called "Step 4: Configure cqlsh COPY FROM settings" \(p. 57\)](#) in the data migration tutorial.

## Using a Cassandra client driver to access Amazon Keyspaces programmatically

You can use many third-party, open-source Cassandra drivers to connect to Amazon Keyspaces. Amazon Keyspaces is compatible with Cassandra drivers that support Apache Cassandra version 3.11.2. For more information about Cassandra drivers, see [Apache Cassandra Client drivers](#).

### Note

To help you get started, you can find end-to-end code samples of connecting to Amazon Keyspaces by using various Cassandra client drivers in the Amazon Keyspaces code example repository on [GitHub](#).

The tutorials in this chapter include a simple CQL query to confirm that the connection to Amazon Keyspaces has been successfully established. To learn how to work with keyspaces and tables after you connect to an Amazon Keyspaces endpoint, see [CQL language reference \(p. 193\)](#).

### Topics

- [Using a Cassandra Java client driver to access Amazon Keyspaces programmatically \(p. 22\)](#)
- [Using a Cassandra Python client driver to access Amazon Keyspaces programmatically \(p. 29\)](#)
- [Using a Cassandra Node.js client driver to access Amazon Keyspaces programmatically \(p. 31\)](#)
- [Using a Cassandra .NET Core client driver to access Amazon Keyspaces programmatically \(p. 34\)](#)
- [Using a Cassandra Go client driver to access Amazon Keyspaces programmatically \(p. 35\)](#)
- [Using a Cassandra Perl client driver to access Amazon Keyspaces programmatically \(p. 38\)](#)

## Using a Cassandra Java client driver to access Amazon Keyspaces programmatically

This section shows you how to connect to Amazon Keyspaces by using a Java client driver. To provide users and applications with credentials for programmatic access to Amazon Keyspaces resources, you can do either of the following:

- Create service-specific credentials that are associated with a specific AWS Identity and Access Management (IAM) user.
- Use an authentication plugin for temporary credentials. This plugin enables [IAM users, roles, and federated identities](#) to add authentication information to Amazon Keyspaces (for Apache Cassandra) API requests using the [AWS Signature Version 4 Process \(SigV4\)](#).

### Note

For an example how to use Amazon Keyspaces with Spring Boot, see <https://github.com/aws-samples/amazon-keyspaces-spring-app-example>.

### Topics

- [Before you begin \(p. 22\)](#)
- [Step-by-step tutorial to connect to Amazon Keyspaces using the DataStax Java driver for Apache Cassandra using service-specific credentials \(p. 24\)](#)
- [Step-by-step tutorial to connect to Amazon Keyspaces using the 4.x DataStax Java driver for Apache Cassandra and the SigV4 authentication plugin \(p. 26\)](#)
- [Connect to Amazon Keyspaces using the 3.x DataStax Java driver for Apache Cassandra and the SigV4 authentication plugin \(p. 28\)](#)

## Before you begin

To connect to Amazon Keyspaces, you need to complete the following tasks before you can start.

1. Amazon Keyspaces requires the use of Transport Layer Security (TLS) to help secure connections with clients.
  - a. Download the Starfield digital certificate using the following command and save `sf-class2-root.crt` locally or in your home directory.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

### Note

You can also use the Amazon digital certificate to connect to Amazon Keyspaces and can continue to do so if your client is connecting to Amazon Keyspaces successfully. The

Starfield certificate provides additional backwards compatibility for clients using older certificate authorities.

- b. Convert the Starfield digital certificate into a trustStore file.

```
openssl x509 -outform der -in sf-class2-root.crt -out temp_file.der
keytool -import -alias cassandra -keystore cassandra_truststore.jks -file
temp_file.der
```

In this step, you need to create a password for the keystore and trust this certificate. The interactive command looks like this.

```
Enter keystore password:
Re-enter new password:
Owner: OU=Starfield Class 2 Certification Authority, O="Starfield Technologies,
Inc.", C=US
Issuer: OU=Starfield Class 2 Certification Authority, O="Starfield Technologies,
Inc.", C=US
Serial number: 0
Valid from: Tue Jun 29 17:39:16 UTC 2004 until: Thu Jun 29 17:39:16 UTC 2034
Certificate fingerprints:
  MD5: 32:4A:4B:BB:C8:63:69:9B:BE:74:9A:C6:DD:1D:46:24
  SHA1: AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
  SHA256:
14:65:FA:20:53:97:B8:76:FA:A6:F0:A9:95:8E:55:90:E4:0F:CC:7F:AA:4F:B7:C2:C8:67:75:21:FB:5F:B6:5
Signature algorithm name: SHA1withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
Extensions:
#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: BF 5F B7 D1 CE DD 1F 86 F4 5B 55 AC DC D7 10 C2 ._.....[U.....
0010: 0E A9 88 E7 ....
]
[OU=Starfield Class 2 Certification Authority, O="Starfield Technologies, Inc.",
C=US]
SerialNumber: [ 00]
]
#2: ObjectId: 2.5.29.19 Criticality=false
BasicConstraints:[
CA:true
PathLen:2147483647
]
#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: BF 5F B7 D1 CE DD 1F 86 F4 5B 55 AC DC D7 10 C2 ._.....[U.....
0010: 0E A9 88 E7 ....
]
]
Trust this certificate? [no]: y
```

2. Attach the trustStore file in the JVM arguments:

```
-Djavax.net.ssl.trustStore=path_to_file/cassandra_truststore.jks
-Djavax.net.ssl.trustStorePassword=my_password
```

## Step-by-step tutorial to connect to Amazon Keyspaces using the DataStax Java driver for Apache Cassandra using service-specific credentials

The following step-by-step tutorial walks you through connecting to Amazon Keyspaces using a Java driver for Cassandra using service-specific credentials. Specifically, you'll use the 4.0 version of the DataStax Java driver for Apache Cassandra.

### Topics

- [Step 1: Prerequisites](#) (p. 24)
- [Step 2: Configure the driver](#) (p. 24)
- [Step 3: Run the sample application](#) (p. 25)

### Step 1: Prerequisites

To follow this tutorial, you need to generate service-specific credentials and add the DataStax Java driver for Apache Cassandra to your Java project.

- Generate service-specific credentials for your Amazon Keyspaces IAM user by completing the steps in [the section called "Generate service-specific credentials" \(p. 16\)](#). If you prefer to use temporary credentials, follow the steps at [the section called "Authentication plugin for Java 4.x" \(p. 26\)](#).
- Add the DataStax Java driver for Apache Cassandra to your Java project. Ensure that you're using a version of the driver that supports Apache Cassandra 3.11.2. For more information, see the [DataStax Java driver for Apache Cassandra documentation](#).

### Step 2: Configure the driver

You can specify settings for the DataStax Java Cassandra driver by creating a configuration file for your application. This configuration file overrides the default settings and tells the driver to connect to the Amazon Keyspaces service endpoint using port 9142. For a list of available service endpoints, see [the section called "Service endpoints" \(p. 17\)](#).

Create a configuration file and save the file in the application's resources folder—for example, `src/main/resources/application.conf`. Open `application.conf` and add the following configuration settings.

1. **Authentication provider** – Create the authentication provider with the `PlainTextAuthProvider` class. `ServiceUserName` and `ServicePassword` should match the user name and password you obtained when you generated the service-specific credentials by following the steps in [Generate service-specific credentials \(p. 16\)](#).

#### Note

You can use short-term credentials by using the authentication plugin for the DataStax Java driver for Apache Cassandra instead of hardcoding credentials in your driver configuration file. To learn more, follow the instructions for the [the section called "Authentication plugin for Java 4.x" \(p. 26\)](#).

2. **Local data center** – Set the value for `local-datacenter` to the Region you're connecting to. For example, if the application is connecting to `cassandra.us-east-2.amazonaws.com`, then set the local data center to `us-east-2`. For all available AWS Regions, see [??? \(p. 17\)](#).
3. **SSL/TLS** – Initialize the `SSLFactory` by adding a section in the configuration file with a single line that specifies the class with `class = DefaultSslEngineFactory`. Provide the path to the `trustStore` file and the password that you created previously.

```
datastax-java-driver {  
  
    basic.contact-points = [ "cassandra.us-east-2.amazonaws.com:9142"  
    advanced.auth-provider{  
        class = PlainTextAuthProvider  
        username = "ServiceUserName"  
        password = "ServicePassword"  
    }  
    basic.load-balancing-policy {  
        local-datacenter = "us-east-2"  
    }  
  
    advanced.ssl-engine-factory {  
        class = DefaultSslEngineFactory  
        truststore-path = "./src/main/resources/cassandra_truststore.jks"  
        truststore-password = "my_password"  
    }  
}
```

#### Note

Instead of adding the path to the trustStore in the configuration file, you can also add the trustStore path directly in the application code or you can add the path to the trustStore to your JVM arguments.

### Step 3: Run the sample application

This code example shows a simple command line application that creates a connection pool to Amazon Keyspaces by using the configuration file we created earlier. It confirms that the connection is established by running a simple query.

```
package <your package>;  
// add the following imports to your project  
import com.datastax.oss.driver.api.core.CqlSession;  
import com.datastax.oss.driver.api.core.config.DriverConfigLoader;  
import com.datastax.oss.driver.api.core.cql.ResultSet;  
import com.datastax.oss.driver.api.core.cql.Row;  
  
public class App  
{  
  
    public static void main( String[] args )  
    {  
        //Use DriverConfigLoader to load your configuration file  
        DriverConfigLoader loader = DriverConfigLoader.fromClasspath("application.conf");  
        try (CqlSession session = CqlSession.builder()  
            .withConfigLoader(loader)  
            .build()) {  
  
            ResultSet rs = session.execute("select * from system_schema.keyspaces");  
            Row row = rs.one();  
            System.out.println(row.getString("keyspace_name"));  
        }  
    }  
}
```

#### Note

Use a try block to establish the connection to ensure that it's always closed. If you don't use a try block, remember to close your connection to avoid leaking resources.

## Step-by-step tutorial to connect to Amazon Keyspaces using the 4.x DataStax Java driver for Apache Cassandra and the SigV4 authentication plugin

The following sections describe how to use the SigV4 authentication plugin for the open-source 4.x DataStax Java driver for Apache Cassandra to access Amazon Keyspaces (for Apache Cassandra). The plugin is available from the [GitHub repository](#).

The SigV4 authentication plugin allows you to use temporary credentials when connecting to Amazon Keyspaces. Instead of requiring a user name and password, this plugin signs API requests for you using the access key that you specify when you configure the plugin. For more information, see [AWS Signature Version 4 Process \(SigV4\)](#).

### Step 1: Prerequisites

To follow this tutorial, you need to complete the following tasks.

- Add the DataStax Java driver for Apache Cassandra to your Java project. Ensure that you're using a version of the driver that supports Apache Cassandra 3.11.2. For more information, see the [DataStax Java Driver for Apache Cassandra documentation](#).
- Add the authentication plugin to your application. The authentication plugin supports version 4.x of the DataStax Java driver for Apache Cassandra. If you're using Apache Maven, or a build system that can use Maven dependencies, add the following dependencies to your `pom.xml` file. Replace the version of the plugin with the latest version as shown at [GitHub repository](#).

```
<dependency>
  <groupId>software.aws.mcs</groupId>
  <artifactId>aws-sigv4-auth-cassandra-java-driver-plugin</artifactId>
  <version>4.0.4</version>
</dependency>
```

### Step 2: Configure the driver

You can specify settings for the DataStax Java Cassandra driver by creating a configuration file for your application. This configuration file overrides the default settings and tells the driver to connect to the Amazon Keyspaces service endpoint using port 9142. For a list of available service endpoints, see [the section called "Service endpoints" \(p. 17\)](#).

Create a configuration file and save the file in the application's resources folder—for example, `src/main/resources/application.conf`. Open `application.conf` and add the following configuration settings.

1. **Authentication provider** – Set the `advanced.auth-provider.class` to a new instance of `software.aws.mcs.auth.SigV4AuthProvider`. The `SigV4AuthProvider` is the authentication handler provided by the plugin for performing SigV4 authentication.
2. **Local data center** – Set the value for `local-datacenter` to the Region you're connecting to. For example, if the application is connecting to `cassandra.us-east-2.amazonaws.com`, then set the local data center to `us-east-2`. For all available AWS Regions, see [??? \(p. 17\)](#).
3. **SSL/TLS** – Initialize the `SSLEngineFactory` by adding a section in the configuration file with a single line that specifies the class with `class = DefaultSslEngineFactory`. Provide the path to the `trustStore` file and the password that you created previously.

```
datastax-java-driver {
```



```
basic.contact-points = ["cassandra.us-east-2.amazonaws.com:9142"]
basic.load-balancing-policy {
    class = DefaultLoadBalancingPolicy
    local-datacenter = us-east-2
}
advanced {
    auth-provider = {
        class = software.aws.mcs.auth.SigV4AuthProvider
        aws-region = us-east-2
    }
    ssl-engine-factory {
        class = DefaultSslEngineFactory
        truststore-path = "./src/main/resources/cassandra_truststore.jks"
        truststore-password = "my_password"
    }
}
}
```

#### Note

Instead of adding the path to the trustStore in the configuration file, you can also add the trustStore path directly in the application code or you can add the path to the trustStore to your JVM arguments.

### Step 3: Run the application

This code example shows a simple command line application that creates a connection pool to Amazon Keyspaces by using the configuration file we created earlier. It confirms that the connection is established by running a simple query.

```
package <your package>;
// add the following imports to your project
import com.datastax.oss.driver.api.core.CqlSession;
import com.datastax.oss.driver.api.core.config.DriverConfigLoader;
import com.datastax.oss.driver.api.core.cql.ResultSet;
import com.datastax.oss.driver.api.core.cql.Row;

public class App
{
    public static void main( String[] args )
    {
        //Use DriverConfigLoader to load your configuration file
        DriverConfigLoader loader = DriverConfigLoader.fromClasspath("application.conf");
        try (CqlSession session = CqlSession.builder()
            .withConfigLoader(loader)
            .build()) {

            ResultSet rs = session.execute("select * from system_schema.keyspaces");
            Row row = rs.one();
            System.out.println(row.getString("keyspace_name"));

        }
    }
}
```

#### Note

Use a try block to establish the connection to ensure that it's always closed. If you don't use a try block, remember to close your connection to avoid leaking resources.

## Connect to Amazon Keyspaces using the 3.x DataStax Java driver for Apache Cassandra and the SigV4 authentication plugin

The following section describes how to use the SigV4 authentication plugin for the 3.x open-source DataStax Java driver for Apache Cassandra to access Amazon Keyspaces. The plugin is available from the [GitHub repository](#).

The SigV4 authentication plugin allows you to use temporary credentials when connecting to Amazon Keyspaces. Instead of requiring a user name and password, this plugin signs API requests for you using the access key that you specify when you configure the plugin. For more information, see [AWS Signature Version 4 Process \(SigV4\)](#).

### Step 1: Prerequisites

To run this code sample, you first need to complete the following tasks.

- Follow the steps at [the section called "Before you begin" \(p. 22\)](#) to download the Starfield digital certificate, convert it to a trustStore file, and attach the trustStore file in the JVM arguments to your application.
- Add the DataStax Java driver for Apache Cassandra to your Java project. Ensure that you're using a version of the driver that supports Apache Cassandra 3.11.2. For more information, see the [DataStax Java Driver for Apache Cassandra documentation](#).
- Add the authentication plugin to your application. The authentication plugin supports version 3.x of the DataStax Java driver for Apache Cassandra. If you're using Apache Maven, or a build system that can use Maven dependencies, add the following dependencies to your `pom.xml` file. Replace the version of the plugin with the latest version as shown at [GitHub repository](#).

```
<dependency>
  <groupId>software.aws.mcs</groupId>
  <artifactId>aws-sigv4-auth-cassandra-java-driver-plugin_3</artifactId>
  <version>3.0.3</version>
</dependency>
```

### Step 2: Run the application

This code example shows a simple command line application that creates a connection pool to Amazon Keyspaces. It confirms that the connection is established by running a simple query.

```
package <your package>;
// add the following imports to your project

import software.aws.mcs.auth.SigV4AuthProvider;
import com.datastax.driver.core.Cluster;
import com.datastax.driver.core.ResultSet;
import com.datastax.driver.core.Row;
import com.datastax.driver.core.Session;

public class App
{
    public static void main( String[] args )
    {
        String endPoint = "cassandra.us-east-2.amazonaws.com";
        int portNumber = 9142;
        Session session = Cluster.builder()
                                .addContactPoint(endPoint)
```

```
        .withPort(portNumber)
        .withAuthProvider(new SigV4AuthProvider("us-east-2"))
        .withSSL()
        .build()
        .connect();

ResultSet rs = session.execute("select * from system_schema.keyspaces");
Row row = rs.one();
System.out.println(row.getString("keyspace_name"));
    }
}
```

Usage notes:

For a list of available endpoints, see [the section called “Service endpoints” \(p. 17\)](#).

## Using a Cassandra Python client driver to access Amazon Keyspaces programmatically

In this section, we show you how to connect to Amazon Keyspaces using a Python client driver. To provide users and applications with credentials for programmatic access to Amazon Keyspaces resources, you can do either of the following:

- Create service-specific credentials that are associated with a specific AWS Identity and Access Management (IAM) user.
- Use an authentication plugin for temporary credentials. This plugin enables [IAM users, roles, and federated identities](#) to add authentication information to Amazon Keyspaces (for Apache Cassandra) API requests using the [AWS Signature Version 4 Process \(SigV4\)](#).

### Topics

- [Before you begin \(p. 29\)](#)
- [Connect to Amazon Keyspaces using the Python driver for Apache Cassandra and service-specific credentials \(p. 30\)](#)
- [Connect to Amazon Keyspaces using the DataStax Python driver for Apache Cassandra and the SigV4 authentication plugin \(p. 30\)](#)

## Before you begin

You need to complete the following task before you can start.

Amazon Keyspaces requires the use of Transport Layer Security (TLS) to help secure connections with clients. To connect to Amazon Keyspaces using TLS, you need to download an Amazon digital certificate and configure the Python driver to use TLS.

Download the Starfield digital certificate using the following command and save `sf-class2-root.crt` locally or in your home directory.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

### Note

You can also use the Amazon digital certificate to connect to Amazon Keyspaces and can continue to do so if your client is connecting to Amazon Keyspaces successfully. The Starfield certificate provides additional backwards compatibility for clients using older certificate authorities.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

## Connect to Amazon Keyspaces using the Python driver for Apache Cassandra and service-specific credentials

The following code example shows you how to connect to Amazon Keyspaces with a Python client driver and service-specific credentials.

```
from cassandra.cluster import Cluster
from ssl import SSLContext, PROTOCOL_TLSv1_2, CERT_REQUIRED
from cassandra.auth import PlainTextAuthProvider

ssl_context = SSLContext(PROTOCOL_TLSv1_2)
ssl_context.load_verify_locations('path_to_file/sf-class2-root.crt')
ssl_context.verify_mode = CERT_REQUIRED
auth_provider = PlainTextAuthProvider(username='ServiceUserName',
    password='ServicePassword')
cluster = Cluster(['cassandra.us-east-2.amazonaws.com'], ssl_context=ssl_context,
    auth_provider=auth_provider, port=9142)
session = cluster.connect()
r = session.execute('select * from system_schema.keyspaces')
print(r.current_rows)
```

Usage notes:

1. Replace "`path_to_file/sf-class2-root.crt`" with the path to the certificate saved in the first step.
2. Ensure that the `ServiceUserName` and `ServicePassword` match the user name and password you obtained when you generated the service-specific credentials by following the steps to [Generate service-specific credentials](#) (p. 16).
3. For a list of available endpoints, see [the section called "Service endpoints"](#) (p. 17).

## Connect to Amazon Keyspaces using the DataStax Python driver for Apache Cassandra and the SigV4 authentication plugin

The following section shows how to use the SigV4 authentication plugin for the open-source DataStax Python driver for Apache Cassandra to access Amazon Keyspaces (for Apache Cassandra). The plugin is available from the [GitHub repository](#).

Add the Python SigV4 authentication plugin to your environment.

```
pip install cassandra-sigv4
```

The following code example shows how to connect to Amazon Keyspaces by using the open-source DataStax Python driver for Cassandra and the SigV4 authentication plugin. The plugin depends on the AWS SDK for Python (Boto3). It uses `boto3.session` to obtain credentials.

```
from cassandra.cluster import Cluster
from ssl import SSLContext, PROTOCOL_TLSv1_2, CERT_REQUIRED
from cassandra.auth import PlainTextAuthProvider
import boto3
from cassandra_sigv4.auth import SigV4AuthProvider

ssl_context = SSLContext(PROTOCOL_TLSv1_2)
```

```
ssl_context.load_verify_locations('path_to_file/sf-class2-root.crt')
ssl_context.verify_mode = CERT_REQUIRED

# use this if you want to use Boto to set the session parameters.
boto_session = boto3.Session(aws_access_key_id="AKIAIOSFODNN7EXAMPLE",
                             aws_secret_access_key="wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY",
                             aws_session_token="AQoDYXdzEJr...<remainder of token>",
                             region_name="us-east-2")
auth_provider = SigV4AuthProvider(boto_session)

# Use this instead of the above line if you want to use the Default Credentials and not
# bother with a session.
# auth_provider = SigV4AuthProvider()

cluster = Cluster(['cassandra.us-east-2.amazonaws.com'], ssl_context=ssl_context,
                  auth_provider=auth_provider,
                  port=9142)
session = cluster.connect()
r = session.execute('select * from system_schema.keyspaces')
print(r.current_rows)
```

Usage notes:

1. Replace "`path_to_file/sf-class2-root.crt`" with the path to the certificate saved in the first step.
2. Ensure that the `aws_access_key_id`, `aws_secret_access_key`, and the `aws_session_token` match the Access Key, Secret Access Key, and Session Token you obtained using `boto3.session`. For more information, see [Credentials](#) in the *AWS SDK for Python (Boto3)*.
3. For a list of available endpoints, see [the section called "Service endpoints" \(p. 17\)](#).

## Using a Cassandra Node.js client driver to access Amazon Keyspaces programmatically

This section shows you how to connect to Amazon Keyspaces by using a Node.js client driver. To provide users and applications with credentials for programmatic access to Amazon Keyspaces resources, you can do either of the following:

- Create service-specific credentials that are associated with a specific AWS Identity and Access Management (IAM) user.
- Use an authentication plugin for temporary credentials. This plugin enables [IAM users, roles, and federated identities](#) to add authentication information to Amazon Keyspaces (for Apache Cassandra) API requests using the [AWS Signature Version 4 Process \(SigV4\)](#).

### Topics

- [Before you begin \(p. 31\)](#)
- [Connect to Amazon Keyspaces using the NodeJS DataStax driver for Apache Cassandra and service-specific credentials \(p. 32\)](#)
- [Connect to Amazon Keyspaces using the DataStax NodeJS driver for Apache Cassandra and the SigV4 authentication plugin \(p. 33\)](#)

## Before you begin

You need to complete the following task before you can start.

Amazon Keyspaces requires the use of Transport Layer Security (TLS) to help secure connections with clients. To connect to Amazon Keyspaces using TLS, you need to download an Amazon digital certificate and configure the Python driver to use TLS.

Download the Starfield digital certificate using the following command and save `sf-class2-root.crt` locally or in your home directory.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

#### Note

You can also use the Amazon digital certificate to connect to Amazon Keyspaces and can continue to do so if your client is connecting to Amazon Keyspaces successfully. The Starfield certificate provides additional backwards compatibility for clients using older certificate authorities.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

## Connect to Amazon Keyspaces using the NodeJS DataStax driver for Apache Cassandra and service-specific credentials

Configure your driver to use the Starfield digital certificate for TLS and authenticate using service-specific credentials. For example:

```
const cassandra = require('cassandra-driver');
const fs = require('fs');
const auth = new cassandra.auth.PlainTextAuthProvider('ServiceUserName',
  'ServicePassword');
const sslOptions1 = {
  ca: [
    fs.readFileSync('path_to_file/sf-class2-root.crt', 'utf-8')],
  host: 'cassandra.us-west-2.amazonaws.com',
  rejectUnauthorized: true
};
const client = new cassandra.Client({
  contactPoints: ['cassandra.us-west-2.amazonaws.com'],
  localDataCenter: 'us-west-2',
  authProvider: auth,
  sslOptions: sslOptions1,
  protocolOptions: { port: 9142 }
});
const query = 'SELECT * FROM system_schema.keyspaces';

client.execute(query)
  .then( result => console.log('Row from Keyspaces %s', result.rows[0]))
  .catch( e=> console.log(` ${e}`));
```

Usage notes:

1. Replace `"path_to_file/sf-class2-root.crt"` with the path to the certificate saved in the first step.
2. Ensure that the `ServiceUserName` and `ServicePassword` match the user name and password you obtained when you generated the service-specific credentials by following the steps to [Generate service-specific credentials](#) (p. 16).
3. For a list of available endpoints, see [the section called "Service endpoints"](#) (p. 17).

## Connect to Amazon Keyspaces using the DataStax NodeJS driver for Apache Cassandra and the SigV4 authentication plugin

The following section shows how to use the SigV4 authentication plugin for the open-source DataStax NodeJS driver for Apache Cassandra to access Amazon Keyspaces (for Apache Cassandra). The plugin is available from the [GitHub repository](#).

Add the NodeJS SigV4 authentication plugin to your application. The plugin supports version 4.x of the DataStax NodeJS driver for Cassandra. The plugin depends on the AWS SDK for NodeJS. It uses `AWSCredentialsProvider` to obtain credentials.

```
$ npm install aws-sigv4-auth-cassandra-plugin --save
```

This code example shows how to set a Region-specific instance of `SigV4AuthProvider` as the authentication provider.

```
const cassandra = require('cassandra-driver');
const fs = require('fs');
const sigV4 = require('aws-sigv4-auth-cassandra-plugin');

const auth = new sigV4.SigV4AuthProvider({
  region: 'us-west-2',
  accessKeyId: 'AKIAIOSFODNN7EXAMPLE',
  secretAccessKey: 'wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY' });

const sslOptions1 = {
  ca: [
    fs.readFileSync('path_to_filecassandra/sf-class2-root.crt', 'utf-8')],
  host: 'cassandra.us-west-2.amazonaws.com',
  rejectUnauthorized: true
};

const client = new cassandra.Client({
  contactPoints: ['cassandra.us-west-2.amazonaws.com'],
  localDataCenter: 'us-west-2',
  authProvider: auth,
  sslOptions: sslOptions1,
  protocolOptions: { port: 9142 }
});

const query = 'SELECT * FROM system_schema.keyspaces';

client.execute(query).then(
  result => console.log('Row from Keyspaces %s', result.rows[0]))
  .catch( e=> console.log(`${e}`));
```

Usage notes:

1. Replace "`path_to_file/sf-class2-root.crt`" with the path to the certificate saved in the first step.
2. Ensure that the `accessKeyId` and `secretAccessKey` match the Access Key and Secret Access Key you obtained using `AWSCredentialsProvider`. For more information, see [Setting Credentials in Node.js](#) in the *AWS SDK for JavaScript in Node.js*.
3. For a list of available endpoints, see [the section called "Service endpoints"](#) (p. 17).

## Using a Cassandra .NET Core client driver to access Amazon Keyspaces programmatically

This section shows you how to connect to Amazon Keyspaces by using a .NET Core client driver. The setup steps will vary depending on your environment and operating system, you might have to modify them accordingly. Amazon Keyspaces requires the use of Transport Layer Security (TLS) to help secure connections with clients. To connect to Amazon Keyspaces using TLS, you need to download a Starfield digital certificate and configure your driver to use TLS.

1. Download the Starfield certificate and save it to a local directory, taking note of the path. Following is an example using PowerShell.

```
$client = new-object System.Net.WebClient
$client.DownloadFile("https://certs.secmaster.net/repository/sf-class2-root.crt", "path_to_file\sf-class2-root.crt")
```

2. Install the CassandraCSharpDriver through nuget, using the nuget console.

```
PM> Install-Package CassandraCSharpDriver
```

3. The following example uses a .NET Core C# console project to connect to &MCS; and run a query.

```
using Cassandra;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Security;
using System.Runtime.ConstrainedExecution;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;

namespace CSharpKeyspacesExample
{
    class Program
    {
        public Program(){}

        static void Main(string[] args)
        {
            X509Certificate2Collection certCollection = new X509Certificate2Collection();
            X509Certificate2 amazoncert = new X509Certificate2(@"path_to_file\sf-class2-root.crt");
            var userName = "ServiceUserName";
            var pwd = "ServicePassword";
            certCollection.Add(amazoncert);

            var awsEndpoint = "cassandra.us-east-2.amazonaws.com" ;

            var cluster = Cluster.Builder()
                .AddContactPoints(awsEndpoint)
                .WithPort(9142)
                .WithAuthProvider(new PlainTextAuthProvider(userName, pwd))
                .WithSSL(new SSLOptions().SetCertificateCollection(certCollection))
                .Build();

            var session = cluster.Connect();
            var rs = session.Execute("SELECT * FROM system_schema.tables;");
            foreach (var row in rs)
            {

```



```
var name = row.GetValue<String>("keyspace_name");  
Console.WriteLine(name);  
    }  
    }  
}
```

Usage notes:

- Replace "[path\\_to\\_file/sf-class2-root.crt](#)" with the path to the certificate saved in the first step.
- Ensure that the [ServiceUserName](#) and [ServicePassword](#) match the user name and password you obtained when you generated the service-specific credentials by following the steps to [Generate service-specific credentials](#) (p. 16).
- For a list of available endpoints, see [the section called "Service endpoints"](#) (p. 17).

## Using a Cassandra Go client driver to access Amazon Keyspaces programmatically

This section shows you how to connect to Amazon Keyspaces by using a Go client driver. To provide users and applications with credentials for programmatic access to Amazon Keyspaces resources, you can do either of the following:

- Create service-specific credentials that are associated with a specific AWS Identity and Access Management (IAM) user.
- Use an authentication plugin for temporary credentials. This plugin enables [IAM users, roles, and federated identities](#) to add authentication information to Amazon Keyspaces (for Apache Cassandra) API requests using the [AWS Signature Version 4 Process \(SigV4\)](#).

### Topics

- [Before you begin](#) (p. 35)
- [Connect to Amazon Keyspaces using the Gocql driver for Apache Cassandra and service-specific credentials](#) (p. 36)
- [Connect to Amazon Keyspaces using the Go driver for Apache Cassandra and the SigV4 authentication plugin](#) (p. 37)

## Before you begin

You need to complete the following task before you can start.

Amazon Keyspaces requires the use of Transport Layer Security (TLS) to help secure connections with clients. To connect to Amazon Keyspaces using TLS, you need to download an Amazon digital certificate and configure the Python driver to use TLS.

Download the Starfield digital certificate using the following command and save `sf-class2-root.crt` locally or in your home directory.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

### Note

You can also use the Amazon digital certificate to connect to Amazon Keyspaces and can continue to do so if your client is connecting to Amazon Keyspaces successfully. The Starfield

certificate provides additional backwards compatibility for clients using older certificate authorities.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

## Connect to Amazon Keyspaces using the Gocql driver for Apache Cassandra and service-specific credentials

1. Create a directory for your application.

```
mkdir ./gocqlexample
```

2. Navigate to the new directory.

```
cd gocqlexample
```

3. Create a file for your application.

```
touch cqlapp.go
```

4. Download the Go driver.

```
go get github.com/gocql/gocql
```

5. Add the following sample code to the cqlapp.go file.

```
package main

import (
    "fmt"
    "github.com/gocql/gocql"
    "log"
)

func main() {

    // add the Amazon Keyspaces service endpoint
    cluster := gocql.NewCluster("cassandra.us-east-2.amazonaws.com:9142")
    // add your service specific credentials
    cluster.Authenticator = gocql.PasswordAuthenticator{
        Username: "ServiceUserName",
        Password: "ServicePassword"}
    // provide the path to the sf-class2-root.crt
    cluster.SslOpts = &gocql.SslOptions{
        CaPath: "path_to_file/sf-class2-root.crt",
    }

    // Override default Consistency to LocalQuorum
    cluster.Consistency = gocql.LocalQuorum

    // Disable initial host lookup
    cluster.DisableInitialHostLookup = true
    session, err := cluster.CreateSession()
    if err != nil {
        fmt.Println("err>", err)
    }
    defer session.Close()

    // run a sample query from the system keyspace
    var text string
```

```
iter := session.Query("SELECT keyspace_name FROM system_schema.tables;").Iter()
for iter.Scan(&text) {
    fmt.Println("keyspace_name:", text)
}
if err := iter.Close(); err != nil {
    log.Fatal(err)
}
session.Close()
}
```

Usage notes:

- a. Replace "[path\\_to\\_file/sf-class2-root.crt](#)" with the path to the certificate saved in the first step.
  - b. Ensure that the [ServiceUserName](#) and [ServicePassword](#) match the user name and password you obtained when you generated the service-specific credentials by following the steps to [Generate service-specific credentials](#) (p. 16).
  - c. For a list of available endpoints, see [the section called "Service endpoints"](#) (p. 17).
6. Build the program.

```
go build cqlapp.go
```

7. Run the program.

```
./cqlapp
```

## Connect to Amazon Keyspaces using the Go driver for Apache Cassandra and the SigV4 authentication plugin

The following code sample shows how to use the SigV4 authentication plugin for the open-source Go driver to access Amazon Keyspaces (for Apache Cassandra). The plugin is available from the [GitHub repository](#).

Add the Go SigV4 authentication plugin to your application. The plugin supports version 4.x of the open-source Go driver for Cassandra and depends on the AWS SDK for Go.

```
$ go mod init
$ go get github.com/aws/aws-sigv4-auth-cassandra-gocql-driver-plugin
```

In this code sample, the Amazon Keyspaces endpoint is represented by the `Cluster` class. It uses the `AwsAuthenticator` for the authenticator property of the cluster to obtain credentials.

```
package main

import (
    "fmt"
    "github.com/aws/aws-sigv4-auth-cassandra-gocql-driver-plugin/sigv4"
    "github.com/gocql/gocql"
    "log"
)

func main() {
    // configuring the cluster options
    cluster := gocql.NewCluster("cassandra.us-west-2.amazonaws.com:9142")
    var auth sigv4.AwsAuthenticator = sigv4.NewAwsAuthenticator()
    auth.Region = "us-west-2"
    auth.AccessKeyId = "AKIAIOSFODNN7EXAMPLE"
}
```

```
auth.SecretAccessKey = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"

cluster.Authenticator = auth

cluster.SslOpts = &gocql.SslOptions{
    CaPath: "path_to_file/sf-class2-root.crt",
}
cluster.Consistency = gocql.LocalQuorum
cluster.DisableInitialHostLookup = true

session, err := cluster.CreateSession()
if err != nil {
    fmt.Println("err>", err)
    return
}
defer session.Close()

// doing the query
var text string
iter := session.Query("SELECT keyspace_name FROM system_schema.tables;").Iter()
for iter.Scan(&text) {
    fmt.Println("keyspace_name:", text)
}
if err := iter.Close(); err != nil {
    log.Fatal(err)
}
}
```

Usage notes:

1. Replace "[path\\_to\\_file/sf-class2-root.crt](#)" with the path to the certificate saved in the first step.
2. Ensure that the [AccessKeyId](#) and [SecretAccessKey](#) match the access key and secret access key you obtained using [AwsAuthenticator](#). For more information, see [Configuring the AWS SDK for Go](#) in the *AWS SDK for Go*.
3. For a list of available endpoints, see [the section called "Service endpoints"](#) (p. 17).

## Using a Cassandra Perl client driver to access Amazon Keyspaces programmatically

This section shows you how to connect to Amazon Keyspaces by using a Perl client driver. For this code sample, we used Perl 5. Amazon Keyspaces requires the use of Transport Layer Security (TLS) to help secure connections with clients.

### Important

To create a secure connection, our code samples use the Starfield digital certificate to authenticate the server before establishing the TLS connection. The Perl driver doesn't validate the server's Amazon SSL certificate, which means that you can't confirm that you are connecting to Amazon Keyspaces. The second step, to configure the driver to use TLS when connecting to Amazon Keyspaces is still required, and ensures that data transferred between the client and server is encrypted.

1. Download the Cassandra DBI driver from <https://metacpan.org/pod/DBD::Cassandra> and install the driver to your Perl environment. The exact steps depend on the environment. The following is a common example.

```
cpanm DBD::Cassandra
```

2. Create a file for your application.

```
touch cqlapp.pl
```

3. Add the following sample code to the cqlapp.pl file.

```
use DBI;
my $user = "ServiceUserName";
my $password = "ServicePassword";
my $db = DBI->connect("dbi:Cassandra:host=cassandra.us-east-2.amazonaws.com;port=9142;tls=1;",
    $user, $password);

my $rows = $db->selectall_arrayref("select * from system_schema.keyspaces");
print "Found the following Keyspaces...\n";
for my $row (@$rows) {
    print join(" ", @$row['keyspace_name']), "\n";
}

$db->disconnect;
```

**Important**

Ensure that the *ServiceUserName* and *ServicePassword* match the user name and password you obtained when you generated the service-specific credentials by following the steps to [Generate service-specific credentials](#) (p. 16).

**Note**

For a list of available endpoints, see [the section called "Service endpoints"](#) (p. 17).

4. Run the application.

```
perl cqlapp.pl
```

# Getting started with Amazon Keyspaces (for Apache Cassandra)

This section is for you if you're new to Apache Cassandra and Amazon Keyspaces (for Apache Cassandra). In this section, you install all the programs and drivers that you need to successfully use Amazon Keyspaces.

This section covers the following topics:

## Topics

- [Tutorial prerequisites and considerations \(p. 40\)](#)
- [Tutorial Step 1: Create a keyspace and a table in Amazon Keyspaces \(p. 41\)](#)
- [Tutorial Step 2: Create, read, update, and delete data \(CRUD\) \(p. 45\)](#)
- [Tutorial Step 3: Delete a table and keyspace in Amazon Keyspaces \(p. 50\)](#)

## Tutorial prerequisites and considerations

Before you start this tutorial, follow the AWS setup instructions in [Accessing Amazon Keyspaces \(for Apache Cassandra\) \(p. 14\)](#). These steps include signing up for AWS and creating an AWS Identity and Access Management (IAM) user with access to Amazon Keyspaces.

Additionally, if you're completing the tutorial using `cqlsh` or an Apache 2.0 licensed Cassandra client driver, complete the setup instructions in [Using `cqlsh` to connect to Amazon Keyspaces \(p. 19\)](#).

After completing the prerequisite steps, proceed to [Tutorial Step 1: Create a keyspace and a table in Amazon Keyspaces \(p. 41\)](#).

# Tutorial Step 1: Create a keyspace and a table in Amazon Keyspaces

In this section, you create a keyspace and add a table to it using the console.

## Note

Before you begin, make sure that you have all the [tutorial prerequisites](#).

## Topics

- [Creating a keyspace \(p. 41\)](#)
- [Creating a table \(p. 42\)](#)

## Creating a keyspace

A *keyspace* groups related tables that are relevant for one or more applications. A keyspace contains one or more tables and defines the replication strategy for all the tables it contains. For more information about keyspace, see the following topics:

- Working with keyspaces: [the section called "Creating keyspaces" \(p. 87\)](#)
- Data definition language (DDL) statements: [Keyspaces \(p. 198\)](#)
- [Quotas for Amazon Keyspaces \(for Apache Cassandra\) \(p. 209\)](#)

When you create a keyspace, you must specify the keyspace name.

## Note

The replication strategy of the keyspace must be `SingleRegionStrategy`. `SingleRegionStrategy` replicates data across three Availability Zones in its AWS Region.

## Using the console

### To create a keyspace using the console

1. Sign in to the AWS Management Console, and open the Amazon Keyspaces console at <https://console.aws.amazon.com/keyspaces/home>.
2. In the navigation pane, choose **Keyspaces**.
3. Choose **Create keyspace**.
4. In the **Keyspace name** box, enter **myGSGKeyspace** as the name for your keyspace.

### Name constraints:

- Cannot be empty.
  - Allowed characters: alphanumeric characters and underscore ( \_ ).
  - Maximum length is 48 characters.
5. To create the keyspace, choose **Create keyspace**.
  6. Verify that the keyspace `myGSGKeyspace` was created by doing the following:
    - a. In the navigation pane, choose **Keyspaces**.
    - b. Locate your keyspace `myGSGKeyspace` in the list of keyspaces.

## Using CQL

The following procedure creates a keyspace using CQL.

### To create a keyspace using CQL

1. Open a command shell, and enter the following:

```
cqlsh
```

2. Create your keyspace using the following CQL command.

```
CREATE KEYSPACE IF NOT EXISTS "myGSGKeyspace"  
  WITH REPLICATION = {'class': 'SingleRegionStrategy'};
```

SingleRegionStrategy uses a replication factor of three and replicates data across three AWS Availability Zones in its Region.

#### Note

Amazon Keyspaces defaults all input to lowercase unless you enclose it in quotation marks. In this case, note "myGSGKeyspace".

3. Verify that your keyspace was created.

```
SELECT * from system_schema.keyspaces ;
```

Your keyspace should be listed.

## Creating a table

A table is where your data is organized and stored. The primary key of your table determines how data will be partitioned in your table. Primary key is composed of a required partition key and one or more optional clustering columns. The combined values that compose the primary key must be unique across all the table's data. For more information about tables, see the following topics:

- Working with tables: [the section called "Creating tables" \(p. 88\)](#)
- DDL statements: [Tables \(p. 199\)](#)
- Table resource management: [Serverless resource management \(p. 72\)](#)
- Monitoring table resource utilization: [the section called "Monitoring with CloudWatch" \(p. 169\)](#)
- [Quotas for Amazon Keyspaces \(for Apache Cassandra\) \(p. 209\)](#)

When you create a table, you specify the following:

- The name of the table.
- The name and data type of each column in the table.
- The primary key for the table.
  - **Partition key** – Required
  - **Clustering columns** – Optional

Use the following procedure to create a table with the specified columns, data types, partition key, and clustering column.



## Using the console

The following procedure creates the table `employees_tbl` with these columns and data types.

```
ID          text
name        text
region      text
division    text
project     text
role        text
pay_scale   int
vacation_hrs float
manager_id  text
```

### To create a table using the console

1. Sign in to the AWS Management Console, and open the Amazon Keyspaces console at <https://console.aws.amazon.com/keyspaces/home>.
2. In the navigation pane, choose **Keyspaces**.
3. Choose `myGSGKeyspace` as the keyspace you want to create this table in.
4. Choose **Create table**.
5. In the **Table name** box, enter `employees_tbl` as a name for your table.

#### Name constraints:

- Cannot be empty.
  - Allowed characters: alphanumeric characters and underscore ( `_` ).
  - Maximum length is 48 characters.
6. In the **Columns** section, repeat the following steps for each column that you want to add to this table.

Add the following columns and data types.

```
id          text
name        text
region      text
division    text
project     text
role        text
pay_scale   int
vacation_hrs float
manager_id  text
```

- a. **Name** – Enter a name for the column.

#### Name constraints:

- Cannot be empty.
- Allowed characters: alphanumeric characters and underscore ( `_` ).
- Maximum length is 48 characters.

- b. **Type** – In the list of data types, choose the data type for this column.
  - c. If you want to add another column, choose **Add column**.
7. Choose `id` as a partition key under **Partition Key**. A partition key is required for each table. A partition key can be made of one or more columns.

8. Add `division` as a clustering column. Clustering columns are optional and determine the sort order within each partition.
  - a. To add a clustering column, choose **Add clustering column**.
  - b. In the **Column** list, choose **division**. In the **Order** list, choose **ASC** to sort in ascending order on the values in this column. (Choose **DESC** for descending order.)
9. In the **Table settings** section, choose **Default settings**.
10. Choose **Create table**.
11. Verify that your table was created.
  - a. In the navigation pane, choose **Tables**.
  - b. Confirm that your table is in the list of tables.
  - c. Choose the name of your table.
  - d. Confirm that all your columns and data types are correct.

**Note**

The columns might not be listed in the same order that you added them to the table.

- e. In the clustering column, confirm that **division** is identified with **true**. All other table columns should be **false**.

## Using CQL

The following procedure creates a table with the following columns and data types using CQL. The `id` column is to be the partition key.

```
id          text
name        text
region      text
division    text
project     text
role        text
pay_scale   int
vacation_hrs float
manager_id  text
```

### To create a table using CQL

1. Open a command shell and enter the following:  
**cqlsh**
2. At the `cqlsh` prompt (`cqlsh>`), specify a keyspace to create your table in.

```
USE "myGSGKeyspace" ;
```

3. At the keyspace prompt (`cqlsh:keyspace_name>`), create your table by entering the following code into your command window.

```
CREATE TABLE IF NOT EXISTS "myGSGKeyspace".employees_tbl (
  id text,
  name text,
  region text,
  division text,
  project text,
  role text,
  pay_scale int,
  vacation_hrs float,
  manager_id text,
```

```
PRIMARY KEY (id,division))  
WITH CLUSTERING ORDER BY (division ASC) ;
```

**Note**

ASC is the default clustering order. You can also specify DESC for descending order.

Note that the `id` column is to be the partition key. Then, `division` is the clustering column ordered by ascending order (ASC).

4. Verify that your table was created.

```
SELECT * from system_schema.tables WHERE keyspace_name='myGSGKeyspace' ;
```

Your table should be listed.

5. Verify your table's structure.

```
SELECT * FROM system_schema.columns WHERE keyspace_name = 'myGSGKeyspace' AND  
table_name = 'employees_tbl' ;
```

Confirm that all the columns and data types are as you expected. The order of the columns might be different than in the CREATE statement.

To perform CRUD (create, read, update, and delete) operations on the data in your table, proceed to [the section called "Step 2: CRUD operations" \(p. 45\)](#).

## Tutorial Step 2: Create, read, update, and delete data (CRUD)

In this section, you use the CQL editor on the console to perform CRUD (create, read, update, and delete) operations on the data in your table. You can also run the commands using `cqlsh`.

**Topics**

- [Tutorial: Inserting and loading data into an Amazon Keyspaces table \(p. 45\)](#)
- [Tutorial: Read from an Amazon Keyspaces table \(p. 46\)](#)
- [Tutorial: Update data in an Amazon Keyspaces table \(p. 48\)](#)
- [Tutorial: Delete data in an Amazon Keyspaces table \(p. 49\)](#)

## Tutorial: Inserting and loading data into an Amazon Keyspaces table

To create data in your `employees_tbl` table, use the INSERT statement to add a single row.

1. To insert a single record, run the following command in the CQL editor.

```
INSERT INTO "myGSGKeyspace".employees_tbl (id, name, project, region, division, role,  
pay_scale, vacation_hrs, manager_id)  
VALUES ('012-34-5678', 'Russ', 'NightFlight', 'US', 'Engineering', 'IC', 3, 12.5,  
'234-56-7890') ;
```

2. Verify that the data was correctly added to your table by running the following command.

```
SELECT * FROM "myGSGKeyspace".employees_tbl ;
```

### To insert multiple records from a file using cqlsh

1. Download the sample data file (employees.csv) contained in the following archive file [sampledata.zip](#). This CSV (comma-separated values) file contains the following data. Remember the path that you save the file to.

ID	name	project	region	division	role	pay_scale	vacation_hrs	manager_id
123-45-6789	Bob	NightFlight	US	Engineering	Intern	1	0	234-56-7890
234-56-7890	Bob	NightFlight	US	Engineering	Manager	6	72	789-01-2345
345-67-8901	Sarah	Storm	US	Engineering	IC	4	108	234-56-7890
456-78-9012	Beth	NightFlight	US	Engineering	IC	7	100.5	234-56-7890
567-89-0123	Ahmed	NightFlight	US	Marketing	IC	4	88	678-90-1234
678-90-1234	Alan	Storm	US	Marketing	Manager	3	18.4	789-01-2345
789-01-2345	Roberta	All	US	Executive	CEO	15	184	None

2. Open a command shell and enter the following:

```
cqlsh
```

3. At the cqlsh prompt (cqlsh>), specify a keyspace.

```
USE "myGSGKeyspace" ;
```

4. Set write consistency to LOCAL\_QUORUM. For more information about supported consistency levels see [the section called "Write consistency levels" \(p. 12\)](#).

```
CONSISTENCY LOCAL_QUORUM;
```

5. At the keyspace prompt (cqlsh:keyspace\_name>), run the following query.

```
COPY employees_tbl  
(id,name,project,region,division,role,pay_scale,vacation_hrs,manager_id)  
FROM 'path-to-the-csv-file/employees.csv' WITH delimiter=',' AND header=TRUE ;
```

6. Verify that the data was correctly added to your table by running the following query.

```
SELECT * FROM employees_tbl ;
```

## Tutorial: Read from an Amazon Keyspaces table

In the [Tutorial: Inserting and loading data into an Amazon Keyspaces table \(p. 45\)](#) section, you used the SELECT statement to verify that you had successfully added data to your table. In this section, you refine your use of SELECT to display specific columns, and only rows that meet specific criteria.

The general form of the SELECT statement is as follows.

```
SELECT column_list FROM table_name [WHERE condition [ALLOW FILTERING]] ;
```

### Topics

- [Selecting all the data in your table \(p. 47\)](#)

- [Selecting a subset of columns \(p. 47\)](#)
- [Selecting a subset of rows \(p. 47\)](#)

## Selecting all the data in your table

The simplest form of the `SELECT` statement returns all the data in your table.

### Important

In a production environment, it is typically not a best practice to run this command, which returns all the data in your table.

### To select all your table's data

- Run the following query.

```
SELECT * FROM "myGSGKeyspace".employees_tbl ;
```

Using the wild-card character ( `*` ) for the `column_list` selects all columns.

## Selecting a subset of columns

### To query for a subset of columns

- To retrieve only the `id`, `name`, and `manager_id` columns, run the following query.

```
SELECT name, id, manager_id FROM "myGSGKeyspace".employees_tbl ;
```

The output will contain only the specified columns in the order listed in the `SELECT` statement.

## Selecting a subset of rows

When querying a large dataset, you might only want records that meet certain criteria. To do this, you can append a `WHERE` clause to the end of our `SELECT` statement.

### To query for a subset of rows

- To retrieve only the record for the employee with the `id` '234-56-7890', run the following query.

```
SELECT * FROM "myGSGKeyspace".employees_tbl WHERE id='234-56-7890' ;
```

The preceding `SELECT` statement returns only the rows where the `id` is 234-56-7890.

## Understanding the `WHERE` clause

The `WHERE` clause is used to filter the data and return only the data that meets the specified criteria. The specified criteria can be a simple condition or a compound condition.

### How to use conditions in a `WHERE` clause

- A simple condition – A single column.

```
WHERE column_name=value
```

You can use a simple condition in a `WHERE` clause if any of the following conditions are met:

- The column is the only column in the table's partition key.
- You add `ALLOW FILTERING` after the condition in the `WHERE` clause.

Be aware that using `ALLOW FILTERING` can result in inconsistent performance, especially with large, and multi-partitioned tables.

- A compound condition – Multiple simple conditions connected by `AND`.

```
WHERE column_name1=value1 AND column_name2=value2 AND column_name3=value3...
```

You can use compound conditions in a `WHERE` clause if any of the following conditions are met:

- The columns in the `WHERE` clause exactly match the columns in the table's partition key, no more and no fewer.
- You add `ALLOW FILTERING` after the compound condition in the `WHERE` clause, as in the following example.

```
SELECT * FROM my_table WHERE col1=5 AND col2='Bob' ALLOW FILTERING ;
```

Be aware that using `ALLOW FILTERING` can result in inconsistent performance, especially with large, and multi-partitioned tables.

## Try it

Create your own CQL queries to find the following from your `employees_tbl` table:

- Find the name, project, and id of all employees.
- Find what project Bob the intern is working on (include at least his name, project, and role in the output).
- **Advanced:** Create an application to find all the employees who have the same manager as Bob the intern. HINT: This might take more than one query.
- **Advanced:** Create an application to find selected columns of all employees working on the project `NightFlight`. HINT: Solving this might require multiple statements.

## Tutorial: Update data in an Amazon Keyspaces table

To update the data in your `employees_tbl` table, use the `UPDATE` statement.

The general form of the `UPDATE` statement is as follows.

```
UPDATE table_name SET column_name=new_value WHERE primary_key=value ;
```

### Tip

- You can update multiple columns by using a comma-separated list of `column_names` and values, as in the following example.

```
UPDATE my_table SET col1='new_value_1', col2='new_value2' WHERE id='12345' ;
```

- If the primary key is composed of multiple columns, all primary key columns and their values must be included in the `WHERE` clause.

- You cannot update any column in the primary key because that would change the primary key for the record.

### To update a single cell

Using your `employees_tbl` table, give the employee with id 567-89-0123 a raise.

```
UPDATE "myGSGKeyspace".employees_tbl SET pay_scale=5 WHERE id='567-89-0123' AND  
division='Marketing' ;
```

Verify that employee's pay scale is now 5.

```
SELECT * FROM "myGSGKeyspace".employees_tbl WHERE id='567-89-0123' ;
```

## Try it

**Advanced:** Your company has hired Bob the intern. Change his record so that his role is 'IC' and his pay scale is 2.

## Tutorial: Delete data in an Amazon Keyspaces table

To delete data in your `employees_tbl` table, use the `DELETE` statement.

You can delete a column from a specific row, individual rows from a table, all the rows from a table, an entire table, or a keyspace. Be careful when deleting data because each of these actions is permanently irreversible.

Deleting one or all rows from a table does not delete the table. Thus you can repopulate it with data. Deleting a table deletes the table and all data in it. To use the table again, you must re-create it and add data to it. Deleting a keyspace deletes the keyspace and all tables within it. To use the keyspace and tables, you must re-create them, and then populate them with data.

## Deleting cells

Deleting a column from a row removes the data from the specified cell. When you display that column using a `SELECT` statement, the data is displayed as `null`, though a null value is not stored in that location.

The general syntax to delete one or more specific columns is as follows.

```
DELETE column_name1[, column_name2...] FROM table_name WHERE condition ;
```

In your `employees_tbl` table, you can see that the CEO has "None" for a manager. First, delete that cell so that you're not carrying any data in it.

### To delete a specific cell

1. Run the following `DELETE` query.

```
DELETE manager_id FROM "myGSGKeyspace".employees_tbl WHERE id='789-01-2345' AND  
division='Executive' ;
```

2. Verify that the delete was made as expected.

```
SELECT * FROM "myGSGKeyspace".employees_tbl WHERE id='789-01-2345' AND  
division='Executive';
```

## Deleting rows

There might be a time when you need to delete an entire row, such as when an employee retires. The general syntax for deleting a row is as follows.

```
DELETE FROM table_name WHERE condition ;
```

### To delete a row

1. Run the following `DELETE` query.

```
DELETE FROM "myGSGKeyspace".employees_tbl WHERE id='456-78-9012' AND  
division='Engineering';
```

2. Verify that the delete was made as expected.

```
SELECT * FROM "myGSGKeyspace".employees_tbl WHERE id='456-78-9012' AND  
division='Engineering';
```

# Tutorial Step 3: Delete a table and keyspace in Amazon Keyspaces

To avoid being charged for tables and data that you don't need, delete all the tables and keyspace that you're not using. When you delete a table, the table and its data are deleted and you stop accruing charges for them. However, the keyspace remains. When you delete a keyspace, the keyspace and all its tables are deleted and you stop accruing charges for them.

## Deleting a table

You can delete a table using the console or CQL. When you delete a table, the table and all its data are deleted.

### Using the console

The following procedure deletes a table and all its data using the AWS Management Console.

#### To delete a table using the console

1. Sign in to the AWS Management Console, and open the Amazon Keyspaces console at <https://console.aws.amazon.com/keyspaces/home>.
2. In the navigation pane, choose **Tables**.
3. Choose the box to the left of the name of each table that you want to delete.
4. Choose **Delete**.
5. On the **Delete table** screen, enter **Delete** in the box. Then, choose **Delete table**.
6. To verify that the table was deleted, choose **Tables** in the navigation pane, and confirm that the `employees_tbl` table is no longer listed.



## Using CQL

The following procedure deletes a table and all its data using CQL.

### To delete a table using CQL

1. Open a command shell and enter the following:

**cqlsh**

2. Delete your table by entering the following command at the keyspace prompt (cqlsh:*keyspace\_name*>).

```
DROP TABLE IF EXISTS "myGSGKeyspace".employees_tbl ;
```

3. Verify that your table was deleted.

```
SELECT * FROM system_schema.tables WHERE keyspace_name = 'myGSGKeyspace' ;
```

Your table should not be listed.

## Deleting a keyspace

You can delete a keyspace using either the AWS Management Console or CQL. When you delete a keyspace, the keyspace and all its tables and data are deleted.

### Using the AWS Management Console

The following procedure deletes a keyspace and all its tables and data using the AWS Management Console.

#### To delete a keyspace using the console

1. Sign in to the AWS Management Console, and open the Amazon Keyspaces console at <https://console.aws.amazon.com/keyspaces/home>.
2. In the navigation pane, choose **Keyspaces**.
3. Choose the box to the left of the name of each keyspace that you want to delete.
4. Choose **Delete**.
5. On the **Delete keyspace** screen, enter **Delete** in the box. Then, choose **Delete keyspace**.
6. To verify that the keyspace myGSGKeyspace was deleted, choose **Keyspaces** in the navigation pane and confirm that it is no longer listed. Because you deleted its keyspace, the `employees_tbl` table under **Tables** should also not be listed.

## Using CQL

The following procedure deletes a keyspace and all its tables and data using CQL.

### To delete a keyspace using CQL

1. Open a command shell and enter the following:

**cqlsh**

2. Delete your keyspace by entering the following command at the keyspace prompt (cqlsh:*keyspace\_name*>).

```
DROP KEYSPACE IF EXISTS "myGSGKeyspace" ;
```

3. Verify that your keyspace was deleted.

```
SELECT * from system_schema.keyspaces ;
```

Your keyspace should not be listed.

# Migrating to Amazon Keyspaces

Amazon Keyspaces (for Apache Cassandra) is a scalable, highly available, and managed Apache Cassandra-compatible database service. You can migrate your data to Amazon Keyspaces from Cassandra databases running on premises or on Amazon Elastic Compute Cloud (Amazon EC2) by using the steps in this section.

We recommend that you follow these best practices to ensure that your migration is successful:

- **Break the migration down into smaller components.**

Consider the following units of migration and their potential footprint in terms of raw data size. Migrating smaller amounts of data in one or more phases may help simplify your migration.

**By cluster** – Migrate all of your Cassandra data at once. This approach may be fine for smaller clusters.

**By keyspace or table** – Break up your migration into groups of keyspaces or tables. This approach can help you migrate data in phases based on your requirements for each workload.

**By data** – Consider migrating data for a specific group of users or products, to bring the size of data down even more.

- **Prioritize what data to migrate first based on simplicity.**

Consider if you have data that could be migrated first more easily—for example, data that does not change during specific times, data from nightly batch jobs, data not used during offline hours, or data from internal apps.

- **Use specific tooling.**

- Get started quickly with loading data into Amazon Keyspaces by using the cqlsh `COPY FROM` command. cqlsh is included with Apache Cassandra and is best suited for loading small datasets or test data. For step-by-step instructions, see [the section called “Loading data using cqlsh” \(p. 53\)](#).
- For production workloads with large datasets, you can use the DataStax Bulk Loader for Apache Cassandra to load data into Amazon Keyspaces using the `dsbulk` command. DSBulk provides more robust import capabilities and is available from the [GitHub repository](#). For step-by-step instructions, see [the section called “Loading data using DSBulk” \(p. 61\)](#).
- For complex migrations, consider using an extract, transform, and load (ETL) tool. You can use Amazon EMR to quickly and effectively perform data transformation workloads. For more information, see the [Amazon EMR Management Guide](#).

## Topics

- [Tutorial: Loading data into Amazon Keyspaces using cqlsh \(p. 53\)](#)
- [Tutorial: Loading data into Amazon Keyspaces using DSBulk \(p. 61\)](#)

## Tutorial: Loading data into Amazon Keyspaces using cqlsh

This step-by-step tutorial guides you through migrating data from Apache Cassandra to Amazon Keyspaces using the `cqlsh COPY` command. In this tutorial, you do the following:

## Topics

- [Prerequisites \(p. 54\)](#)

- [Step 1: Create the source CSV file and target table \(p. 54\)](#)
- [Step 2: Prepare the data \(p. 55\)](#)
- [Step 3: Set throughput capacity for the table \(p. 56\)](#)
- [Step 4: Configure cqlsh COPY FROM settings \(p. 57\)](#)
- [Step 5: Run the cqlsh COPY FROM command \(p. 59\)](#)
- [Troubleshooting \(p. 60\)](#)

## Prerequisites

You must complete the following tasks before you can start this tutorial.

1. If you have not already done so, sign up for an AWS account by following the steps at [the section called “Signing up for AWS” \(p. 14\)](#).
2. Create service-specific credentials by following the steps at [the section called “Generate service-specific credentials using the console” \(p. 16\)](#).
3. Set up the Cassandra Query Language shell (cqlsh) connection and confirm that you can connect to Amazon Keyspaces by following the steps at [the section called “Using cqlsh” \(p. 19\)](#).

## Step 1: Create the source CSV file and target table

For this tutorial, we use a comma-separated values (CSV) file with the name `keyspaces_sample_table.csv` as the source file for the data migration. The provided sample file contains a few rows of data for a table with the name `book_awards`.

1. Create the source file. You can choose one of the following options:
  - Download the sample CSV file (`keyspaces_sample_table.csv`) contained in the following archive file [samplemigration.zip](#). Unzip the archive and take note of the path to `keyspaces_sample_table.csv`.
  - To populate a CSV file with your own data stored in an Apache Cassandra database, you can populate the source CSV file by using the `cqlsh COPY TO` statement as shown in the following example.

```
cqlsh localhost 9042 -u "username" -p "password" --execute "COPY mykeyspace.mytable  
TO 'keyspaces_sample_table.csv' WITH HEADER=true"
```

Make sure the CSV file you create meets the following requirements:

- The first row contains the column names.
  - The column names in the source CSV file match the column names in the target table.
  - The data is delimited with a comma.
  - All data values are valid Amazon Keyspaces data types. See [the section called “Data types” \(p. 194\)](#).
2. Create the target keyspace and table in Amazon Keyspaces.
    - a. Connect to Amazon Keyspaces using `cqlsh`, replacing the service endpoint, user name, and password in the following example with your own values.

```
cqlsh cassandra.us-east-2.amazonaws.com 9142 -u "111122223333" -p "wJalrXUtnFEMI/  
K7MDENG/bPxrFiCYEXAMPLEKEY" --ssl
```

- b. Create a new keyspace with the name `catalog` as shown in the following example.

```
CREATE KEYSPACE catalog WITH REPLICATION = {'class': 'SingleRegionStrategy'};
```

- c. When the new keyspace is available, use the following code to create the target table `book_awards`.

```
CREATE TABLE "catalog.book_awards" (  
    year int,  
    award text,  
    rank int,  
    category text,  
    book_title text,  
    author text,  
    publisher text,  
    PRIMARY KEY ((year, award), category, rank)  
);
```

If Apache Cassandra is your original data source, a simple way to create the Amazon Keyspaces target table with matching headers is to generate the `CREATE TABLE` statement from the source table, as shown in the following statement.

```
cqlsh localhost 9042 -u "username" -p "password" --execute "DESCRIBE  
TABLE mykeyspace.mytable;"
```

Then create the target table in Amazon Keyspaces with the column names and data types matching the description from the Cassandra source table.

## Step 2: Prepare the data

Preparing the source data for an efficient transfer is a two-step process. First, you randomize the data. In the second step, you analyze the data to determine the appropriate `cqlsh` parameter values and required table settings.

### Randomize the data

The `cqlsh COPY FROM` command reads and writes data in the same order that it appears in the CSV file. If you use the `cqlsh COPY TO` command to create the source file, the data is written in key-sorted order in the CSV. Internally, Amazon Keyspaces partitions data using partition keys. Although Amazon Keyspaces has built-in logic to help load balance requests for the same partition key, loading the data is faster and more efficient if you randomize the order. This is because you can take advantage of the built-in load balancing that occurs when Amazon Keyspaces is writing to different partitions.

To spread the writes across the partitions evenly, you must randomize the data in the source file. You can write an application to do this or use an open-source tool, such as [Shuf](#). [Shuf](#) is freely available on Linux distributions, on macOS (by installing `coreutils` in [homebrew](#)), and on Windows (by using Windows Subsystem for Linux (WSL)). One extra step is required to prevent the header row with the column names to get shuffled in this step.

To randomize the source file while preserving the header, enter the following code.

```
tail +2 keyspaces_sample_table.csv | shuf -o keyspace.table.csv  
(head -1 keyspaces_sample_table.csv && cat keyspace.table.csv ) > keyspace.table.csv1 &&  
mv keyspace.table.csv1 keyspace.table.csv
```

`Shuf` rewrites the data to a new CSV file called `keyspace.table.csv`. You can now delete the `keyspaces_sample_table.csv` file—you no longer need it.

### Analyze the data

Determine the average and maximum row size by analyzing the data.

You do this for the following reasons:

- The average row size helps to estimate the total amount of data to be transferred.
- You need the average row size to provision the write capacity needed for the data upload.
- You can make sure that each row is less than 1 MB in size, which is the maximum row size in Amazon Keyspaces.

#### Note

This quota refers to row size, not partition size. Unlike Apache Cassandra partitions, Amazon Keyspaces partitions can be virtually unbound in size. Partition keys and clustering columns require additional storage for metadata, which you must add to the raw size of rows. For more information, see [the section called “Calculating row size” \(p. 91\)](#).

The following code uses [AWK](#) to analyze a CSV file and print the average and maximum row size.

```
awk -F, 'BEGIN {samp=10000;max=-1;}{if(NR>1){len=length($0);t+=len;avg=t/NR;max=(len>max ? len : max)}}NR==samp{exit}END{printf("{lines: %d, average: %d bytes, max: %d bytes}\n",NR,avg,max);}' keyspace.table.csv
```

Running this code results in the following output.

```
using 10,000 samples:
{lines: 10000, avg: 123 bytes, max: 225 bytes}
```

You use the average row size in the next step of this tutorial to provision the write capacity for the table.

## Step 3: Set throughput capacity for the table

This tutorial shows you how to tune cqlsh to load data within a set time range. Because you know how many reads and writes you perform in advance, use provisioned capacity mode. After you finish the data transfer, you should set the capacity mode of the table to match your application's traffic patterns. To learn more about capacity management, see [Serverless resource management \(p. 72\)](#).

With provisioned capacity mode, you specify how much read and write capacity you want to provision to your table in advance. Write capacity is billed hourly and metered in write capacity units (WCUs). Each WCU is enough write capacity to support writing 1 KB of data per second. When you load the data, the write rate must be under the max WCUs (parameter: `write_capacity_units`) that are set on the target table.

By default, you can provision up to 40,000 WCUs to a table and 80,000 WCUs across all the tables in your account. If you need additional capacity, you can request a quota increase in the [Service Quotas](#) console. For more information about quotas, see [Quotas \(p. 209\)](#).

### Calculate the average number of WCUs required for an insert

Inserting 1 KB of data per second requires 1 WCU. If your CSV file has 360,000 rows and you want to load all the data in 1 hour, you must write 100 rows per second (360,000 rows / 60 minutes / 60 seconds = 100 rows per second). If each row has up to 1 KB of data, to insert 100 rows per second, you must provision 100 WCUs to your table. If each row has 1.5 KB of data, you need two WCUs to insert one row per second. Therefore, to insert 100 rows per second, you must provision 200 WCUs.

To determine how many WCUs you need to insert one row per second, divide the average row size in bytes by 1024 and round up to the nearest whole number.

For example, if the average row size is 3000 bytes, you need three WCUs to insert one row per second.

```
ROUNDUP(3000 / 1024) = ROUNDUP(2.93) = 3 WCUs
```

### Calculate data load time and capacity

Now that you know the average size and number of rows in your CSV file, you can calculate how many WCUs you need to load the data in a given amount of time, and the approximate time it takes to load all the data in your CSV file using different WCU settings.

For example, if each row in your file is 1 KB and you have 1,000,000 rows in your CSV file, to load the data in 1 hour, you need to provision at least 278 WCUs to your table for that hour.

```
1,000,000 rows * 1 KBs = 1,000,000 KBs  
1,000,000 KBs / 3600 seconds = 277.8 KBs / second = 278 WCUs
```

### Configure provisioned capacity settings

You can set a table's write capacity settings when you create the table or by using the `ALTER TABLE CQL` command. The following is the syntax for altering a table's provisioned capacity settings with the `ALTER TABLE CQL` statement.

```
ALTER TABLE mykeyspace.mytable WITH custom_properties={'capacity_mode':{'throughput_mode':  
'PROVISIONED', 'read_capacity_units': 100, 'write_capacity_units': 278}} ;
```

For the complete language reference, see [the section called "ALTER TABLE" \(p. 202\)](#).

## Step 4: Configure `cqlsh COPY FROM` settings

This section outlines how to determine the parameter values for `cqlsh COPY FROM`. The `cqlsh COPY FROM` command reads the CSV file that you prepared earlier and inserts the data into Amazon Keyspaces using CQL. The command divides up the rows and distributes the `INSERT` operations among a set of workers. Each worker establishes a connection with Amazon Keyspaces and sends `INSERT` requests along this channel.

The `cqlsh COPY` command doesn't have internal logic to distribute work evenly among its workers. However, you can configure it manually to make sure that the work is distributed evenly. Start by reviewing these key `cqlsh` parameters:

- **DELIMITER** – If you used a delimiter other than a comma, you can set this parameter, which defaults to comma.
- **INGESTRATE** – The target number of rows that `cqlsh COPY FROM` attempts to process per second. If unset, it defaults to 100,000.
- **NUMPROCESSES** – The number of child worker processes that `cqlsh` creates for `COPY FROM` tasks. The maximum for this setting is 16, the default is `num_cores - 1`, where `num_cores` is the number of processing cores on the host running `cqlsh`.
- **MAXBATCHSIZE** – The batch size determines the maximal number of rows inserted into the destination table in a single batch. If unset, `cqlsh` uses batches of 20 inserted rows.
- **CHUNKSIZE** – The size of the work unit that passes to the child worker. By default, it is set to 5,000.
- **MAXATTEMPTS** – The maximum number of times to retry a failed worker chunk. After the maximum attempt is reached, the failed records are written to a new CSV file that you can run again later after investigating the failure.

Set `INGESTRATE` based on the number of WCUs that you provisioned to the target destination table. The `INGESTRATE` of the `cqlsh COPY FROM` command isn't a limit—it's a target average. This means

it can (and often does) burst above the number you set. To allow for bursts and make sure that enough capacity is in place to handle the data load requests, set `INGESTRATE` to 90% of the table's write capacity.

```
INGESTRATE = WCUs * .90
```

Next, set the `NUMPROCESSES` parameter to equal to one less than the number of cores on your system. To find out what the number of cores of your system is, you can run the following code.

```
python -c "import multiprocessing; print(multiprocessing.cpu_count())"
```

For this tutorial, we use the following value.

```
NUMPROCESSES = 4
```

Each process creates a worker, and each worker establishes a connection to Amazon Keyspaces. Amazon Keyspaces can support up to 3,000 CQL requests per second on every connection. This means that you have to make sure that each worker is processing fewer than 3,000 requests per second.

As with `INGESTRATE`, the workers often burst above the number you set and aren't limited by clock seconds. Therefore, to account for bursts, set your `cqlsh` parameters to target each worker to process 2,500 requests per second. To calculate the amount of work distributed to a worker, use the following guideline.

- Divide `INGESTRATE` by `NUMPROCESSES`.
- If `INGESTRATE / NUMPROCESSES > 2,500`, lower the `INGESTRATE` to make this formula true.

```
INGESTRATE / NUMPROCESSES <= 2,500
```

Before you configure the settings to optimize the upload of our sample data, let's review the `cqlsh` default settings and see how using them impacts the data upload process. Because `cqlsh COPY FROM` uses the `CHUNKSIZE` to create chunks of work (`INSERT` statements) to distribute to workers, the work is not automatically distributed evenly. Some workers might sit idle, depending on the `INGESTRATE` setting.

To distribute work evenly among the workers and keep each worker at the optimal 2,500 requests per second rate, you must set `CHUNKSIZE`, `MAXBATCHSIZE`, and `INGESTRATE` by changing the input parameters. To optimize network traffic utilization during the data load, choose a value for `MAXBATCHSIZE` that is close to the maximum value of 30. By changing `CHUNKSIZE` to 100 and `MAXBATCHSIZE` to 25, the 10,000 rows are spread evenly among the four workers ( $10,000 / 2500 = 4$ ).

The following code example illustrates this.

```
INGESTRATE = 10,000
NUMPROCESSES = 4
CHUNKSIZE = 100
MAXBATCHSIZE. = 25
Work Distribution:
Connection 1 / Worker 1 : 2,500 Requests per second
Connection 2 / Worker 2 : 2,500 Requests per second
Connection 3 / Worker 3 : 2,500 Requests per second
Connection 4 / Worker 4 : 2,500 Requests per second
```

To summarize, use the following formulas when setting `cqlsh COPY FROM` parameters:

- **`INGESTRATE`** = `write_capacity_units` \* .90



- **NUMPROCESSES** = `num_cores - 1` (default)
- **INGESTRATE / NUMPROCESSES** = 2,500 (This must be a true statement.)
- **MAXBATCHSIZE** = 30 (Defaults to 20. Amazon Keyspaces accepts batches up to 30.)
- **CHUNKSIZE** = `(INGESTRATE / NUMPROCESSES) / MAXBATCHSIZE`

Now that you have calculated `NUMPROCESSES`, `INGESTRATE`, and `CHUNKSIZE`, you're ready to load your data.

## Step 5: Run the `cqlsh COPY FROM` command

To run the `cqlsh COPY FROM` command, complete the following steps.

1. Connect to Amazon Keyspaces using `cqlsh`.
2. Choose your keyspace with the following code.

```
use mykeyspace;
```

3. Set write consistency to `LOCAL_QUORUM`. To ensure data durability, Amazon Keyspaces doesn't allow other write consistency settings. See the following code.

```
CONSISTENCY LOCAL_QUORUM;
```

4. Prepare your `cqlsh COPY FROM` syntax using the following code example.

```
COPY mytable FROM './keyspace.table.csv' WITH HEADER=true  
AND INGESTRATE=calculated ingestrate  
AND NUMPROCESSES=calculated numprocess  
AND MAXBATCHSIZE=20  
AND CHUNKSIZE=calculated chunksize;
```

5. Run the statement prepared in the previous step. `cqlsh` echoes back all the settings that you've configured.
  - a. Make sure that the settings match your input. See the following example.

```
Reading options from the command line: {'chunksize': '120', 'header': 'true',  
'ingestrate': '36000', 'numprocesses': '15', 'maxbatchsize': '20'}  
Using 15 child processes
```

- b. Review the number of rows transferred and the current average rate, as shown in the following example.

```
Processed: 57834 rows; Rate: 6561 rows/s; Avg. rate: 31751 rows/s
```

- c. When `cqlsh` has finished uploading the data, review the summary of the data load statistics (the number of files read, runtime, and skipped rows) as shown in the following example.

```
15556824 rows imported from 1 files in 8 minutes and 8.321 seconds (0 skipped).
```

In this final step of the tutorial, you have uploaded the data to Amazon Keyspaces.

### Important

Now that you have transferred your data, adjust the capacity mode settings of your target table to match your application's regular traffic patterns. You incur charges at the hourly rate for your provisioned capacity until you change it.

## Troubleshooting

After the data upload has completed, check to see if rows were skipped. To do so, navigate to the source directory of the source CSV file and search for a file with the following name.

```
import_yourcsvfilename.err.timestamp.csv
```

cqlsh writes any skipped rows of data into a file with that name. If the file exists in your source directory and has data in it, these rows didn't upload to Amazon Keyspaces. To retry these rows, first check for any errors that were encountered during the upload and adjust the data accordingly. To retry these rows, you can rerun the process.

### Common errors

The most common reasons why rows aren't loaded are capacity errors and parsing errors.

#### Parser errors when uploading data to Amazon Keyspaces

The following example shows a skipped row due to a `ParseError`.

```
Failed to import 1 rows: ParseError - Invalid ... -
```

To resolve this error, you need to make sure that the data to be imported matches the table schema in Amazon Keyspaces. Review the import file for parsing errors. You can try using a single row of data using an `INSERT` statement to isolate the error.

#### Capacity errors when uploading data to Amazon Keyspaces

```
Failed to import 1 rows: WriteTimeout - Error from server: code=1100 [Coordinator node  
timed out waiting for replica nodes' responses]  
message="Operation timed out - received only 0 responses." info={'received_responses': 0,  
'required_responses': 2, 'write_type': 'SIMPLE', 'consistency':  
'LOCAL_QUORUM'}, will retry later, attempt 1 of 100
```

Amazon Keyspaces uses the `ReadTimeout` and `WriteTimeout` exceptions to indicate when a write request fails due to insufficient throughput capacity. To help diagnose insufficient capacity exceptions, Amazon Keyspaces publishes `WriteThrottleEvents` and `ReadThrottledEvents` metrics in Amazon CloudWatch. For more information, see [the section called "Monitoring with CloudWatch" \(p. 169\)](#).

#### cqlsh errors when uploading data to Amazon Keyspaces

To help troubleshoot cqlsh errors, rerun the failing command with the `--debug` flag.

When using an incompatible version of cqlsh, you see the following error.

```
AttributeError: 'NoneType' object has no attribute 'is_up'  
Failed to import 3 rows: AttributeError - 'NoneType' object has no attribute 'is_up',  
given up after 1 attempts
```

Confirm that the correct version of cqlsh is installed by running the following command.

```
cqlsh --version
```

You should see something like the following for output.

```
cqlsh 5.0.1
```

If you're using Windows, replace all instances of `cqlsh` with `cqlsh.bat`. For example, to check the version of `cqlsh` in Windows, run the following command.

```
cqlsh.bat --version
```

The connection to Amazon Keyspaces fails after the `cqlsh` client receives three consecutive errors of any type from the server. The `cqlsh` client fails with the following message.

```
Failed to import 1 rows: NoHostAvailable - , will retry later, attempt 3 of 100
```

To resolve this error, you need to make sure that the data to be imported matches the table schema in Amazon Keyspaces. Review the import file for parsing errors. You can try using a single row of data by using an `INSERT` statement to isolate the error.

The client automatically attempts to reestablish the connection.

## Tutorial: Loading data into Amazon Keyspaces using DSBulk

This step-by-step tutorial guides you through migrating data from Apache Cassandra to Amazon Keyspaces using the DataStax Bulk Loader (DSBulk) available on [GitHub](#). In this tutorial, you complete the following steps:

### Topics

- [Prerequisites \(p. 61\)](#)
- [Step 1: Create the source CSV file and target table \(p. 63\)](#)
- [Step 2: Prepare the data \(p. 64\)](#)
- [Step 3: Set throughput capacity for the table \(p. 65\)](#)
- [Step 4: Configure DSBulk settings \(p. 66\)](#)
- [Step 5: Run the DSBulk load command \(p. 67\)](#)

## Prerequisites

You must complete the following tasks before you can start this tutorial.

1. If you have not already done so, sign up for an AWS account by following the steps at [the section called "Signing up for AWS" \(p. 14\)](#).
2. Create credentials by following the steps at [the section called "Creating credentials" \(p. 16\)](#).

### Note

DSBulk currently only supports service-specific credentials. If you prefer to manage access to Amazon Keyspaces by using AWS Identity and Access Management users and roles, you can use the SigV4 authentication plugin and disable the service-specific credentials after you complete data upload with DSBulk.

3. Create a JKS trust store file.
  - a. Download the Starfield digital certificate using the following command and save `sf-class2-root.crt` locally or in your home directory.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

### Note

You can also use the Amazon digital certificate to connect to Amazon Keyspaces and can continue to do so if your client is connecting to Amazon Keyspaces successfully. The Starfield certificate provides additional backwards compatibility for clients using older certificate authorities.

- b. Convert the Starfield digital certificate into a trustStore file.

```
openssl x509 -outform der -in sf-class2-root.crt -out temp_file.der  
keytool -import -alias cassandra -keystore cassandra_truststore.jks -file  
temp_file.der
```

In this step, you need to create a password for the keystore and trust this certificate. The interactive command looks like this.

```
Enter keystore password:  
Re-enter new password:  
Owner: OU=Starfield Class 2 Certification Authority, O="Starfield Technologies,  
Inc.", C=US  
Issuer: OU=Starfield Class 2 Certification Authority, O="Starfield Technologies,  
Inc.", C=US  
Serial number: 0  
Valid from: Tue Jun 29 17:39:16 UTC 2004 until: Thu Jun 29 17:39:16 UTC 2034  
Certificate fingerprints:  
MD5: 32:4A:4B:BB:C8:63:69:9B:BE:74:9A:C6:DD:1D:46:24  
SHA1: AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A  
SHA256:  
14:65:FA:20:53:97:B8:76:FA:A6:F0:A9:95:8E:55:90:E4:0F:CC:7F:AA:4F:B7:C2:C8:67:75:21:FB:5F:B6:5  
Signature algorithm name: SHA1withRSA  
Subject Public Key Algorithm: 2048-bit RSA key  
Version: 3  
Extensions:  
#1: ObjectId: 2.5.29.35 Criticality=false  
AuthorityKeyIdentifier [  
KeyIdentifier [  
0000: BF 5F B7 D1 CE DD 1F 86 F4 5B 55 AC DC D7 10 C2 ._.....[U.....  
0010: 0E A9 88 E7 ....  
]  
[OU=Starfield Class 2 Certification Authority, O="Starfield Technologies, Inc.",  
C=US]  
SerialNumber: [ 00]  
]  
#2: ObjectId: 2.5.29.19 Criticality=false  
BasicConstraints:[  
CA:true  
PathLen:2147483647  
]  
#3: ObjectId: 2.5.29.14 Criticality=false  
SubjectKeyIdentifier [  
KeyIdentifier [  
0000: BF 5F B7 D1 CE DD 1F 86 F4 5B 55 AC DC D7 10 C2 ._.....[U.....  
0010: 0E A9 88 E7 ....  
]  
]  
Trust this certificate? [no]: y
```

4. Set up the Cassandra Query Language shell (cqlsh) connection and confirm that you can connect to Amazon Keyspaces by following the steps at [the section called "Using cqlsh" \(p. 19\)](#).
5. Download and install DSBulk.

- a. To download DSBulk, you can use the following code.

```
curl -OL https://downloads.datastax.com/dsbulk/dsbulk-1.8.0.tar.gz
```

- b. Then unpack the tar file and add DSBulk to your `PATH` as shown in the following example.

```
tar -zxvf dsbulk-1.8.0.tar.gz
# add the DSBulk directory to the path
export PATH=$PATH:./dsbulk-1.8.0/bin
```

- c. Create an `application.conf` file to store settings to be used by DSBulk. You can save the following example as `./dsbulk_keyspaces.conf`. Replace `localhost` with the contact point of your local Cassandra cluster if you are not on the local node, for example the DNS name or IP address. Replace `username` and `password` with your server credentials. Take note of the file name and path, as you're going to need to specify this later in the `dsbulk load` command.

```
datastax-java-driver {
  basic.contact-points = [ "localhost" ]
  advanced.auth-provider {
    class = PlainTextAuthProvider
    username = "username"
    password = "password"
  }
}
```

## Step 1: Create the source CSV file and target table

For this tutorial, we use a comma-separated values (CSV) file with the name `keyspaces_sample_table.csv` as the source file for the data migration. The provided sample file contains a few rows of data for a table with the name `book_awards`.

1. Create the source file. You can choose one of the following options:
  - Download the sample CSV file (`keyspaces_sample_table.csv`) contained in the following archive file [samplemigration.zip](#). Unzip the archive and take note of the path to `keyspaces_sample_table.csv`.
  - To populate a CSV file with your own data stored in an Apache Cassandra database, you can populate the source CSV file by using `dsbulk unload` as shown in the following example.

```
dsbulk unload -k mykeyspace -t mytable -f ./my_application.conf
> keyspace_sample_table.csv
```

Make sure the CSV file you create meets the following requirements:

- The first row contains the column names.
  - The column names in the source CSV file match the column names in the target table.
  - The data is delimited with a comma.
  - All data values are valid Amazon Keyspaces data types. See [the section called “Data types” \(p. 194\)](#).
2. Create the target keyspace and table in Amazon Keyspaces.
    - a. Connect to Amazon Keyspaces using `cqlsh`, replacing the service endpoint, user name, and password in the following example with your own values.

```
cqlsh cassandra.us-east-2.amazonaws.com 9142 -u "111122223333" -p "wJalrXUtnFEMI/  
K7MDENG/bPxRfiCYEXAMPLEKEY" --ssl
```

- b. Create a new keyspace with the name `catalog` as shown in the following example.

```
CREATE KEYSPACE catalog WITH REPLICATION = {'class': 'SingleRegionStrategy'};
```

- c. After the new keyspace has a status of available, use the following code to create the target table `book_awards`. To learn more about asynchronous resource creation and how to check if a resource is available, see [the section called "Creating keyspaces" \(p. 87\)](#).

```
CREATE TABLE catalog.book_awards (  
    year int,  
    award text,  
    rank int,  
    category text,  
    book_title text,  
    author text,  
    publisher text,  
    PRIMARY KEY ((year, award), category, rank)  
);
```

If Apache Cassandra is your original data source, a simple way to create the Amazon Keyspaces target table with matching headers is to generate the `CREATE TABLE` statement from the source table as shown in the following statement.

```
cqlsh localhost 9042 -u "username" -p "password" --execute "DESCRIBE  
TABLE mykeyspace.mytable;"
```

Then create the target table in Amazon Keyspaces with the column names and data types matching the description from the Cassandra source table.

## Step 2: Prepare the data

Preparing the source data for an efficient transfer is a two-step process. First, you randomize the data. In the second step, you analyze the data to determine the appropriate `dsbulk` parameter values and required table settings.

### Randomize the data

The `dsbulk` command reads and writes data in the same order that it appears in the CSV file. If you use the `dsbulk` command to create the source file, the data is written in key-sorted order in the CSV. Internally, Amazon Keyspaces partitions data using partition keys. Although Amazon Keyspaces has built-in logic to help load balance requests for the same partition key, loading the data is faster and more efficient if you randomize the order. This is because you can take advantage of the built-in load balancing that occurs when Amazon Keyspaces is writing to different partitions.

To spread the writes across the partitions evenly, you must randomize the data in the source file. You can write an application to do this or use an open-source tool, such as [Shuf](#). `Shuf` is freely available on Linux distributions, on macOS (by installing `coreutils` in [homebrew](#)), and on Windows (by using Windows Subsystem for Linux (WSL)). One extra step is required to prevent the header row with the column names to get shuffled in this step.

To randomize the source file while preserving the header, enter the following code.

```
tail +2 keyspaces_sample_table.csv | shuf -o keyspace.table.csv  
(head -1 keyspaces_sample_table.csv && cat keyspace.table.csv ) > keyspace.table.csv1 &&  
mv keyspace.table.csv1 keyspace.table.csv
```

Shuf rewrites the data to a new CSV file called `keyspace.table.csv`. You can now delete the `keyspaces_sample_table.csv` file—you no longer need it.

### Analyze the data

Determine the average and maximum row size by analyzing the data.

You do this for the following reasons:

- The average row size helps to estimate the total amount of data to be transferred.
- You need the average row size to provision the write capacity needed for the data upload.
- You can make sure that each row is less than 1 MB in size, which is the maximum row size in Amazon Keyspaces.

#### Note

This quota refers to row size, not partition size. Unlike Apache Cassandra partitions, Amazon Keyspaces partitions can be virtually unbound in size. Partition keys and clustering columns require additional storage for metadata, which you must add to the raw size of rows. For more information, see [the section called “Calculating row size” \(p. 91\)](#).

The following code uses [AWK](#) to analyze a CSV file and print the average and maximum row size.

```
awk -F, 'BEGIN {samp=10000;max=-1;}{if(NR>1){len=length($0);t+=len;avg=t/NR;max=(len>max ?  
len : max)}}NR==samp{exit}END{printf("{lines: %d, average: %d bytes, max: %d  
bytes}\n",NR,avg,max);}' keyspace.table.csv
```

Running this code results in the following output.

```
using 10,000 samples:  
{lines: 10000, avg: 123 bytes, max: 225 bytes}
```

Make sure that your maximum row size doesn't exceed 1 MB. If it does, you have to break up the row or compress the data to bring the row size below 1 MB. In the next step of this tutorial, you use the average row size to provision the write capacity for the table.

## Step 3: Set throughput capacity for the table

This tutorial shows you how to tune DSBulk to load data within a set time range. Because you know how many reads and writes you perform in advance, use provisioned capacity mode. After you finish the data transfer, you should set the capacity mode of the table to match your application's traffic patterns. To learn more about capacity management, see [Serverless resource management \(p. 72\)](#).

With provisioned capacity mode, you specify how much read and write capacity you want to provision to your table in advance. Write capacity is billed hourly and metered in write capacity units (WCUs). Each WCU is enough write capacity to support writing 1 KB of data per second. When you load the data, the write rate must be under the max WCUs (parameter: `write_capacity_units`) that are set on the target table.

By default, you can provision up to 40,000 WCUs to a table and 80,000 WCUs across all the tables in your account. If you need additional capacity, you can request a quota increase in the [Service Quotas](#) console. For more information about quotas, see [Quotas \(p. 209\)](#).

### Calculate the average number of WCUs required for an insert

Inserting 1 KB of data per second requires 1 WCU. If your CSV file has 360,000 rows and you want to load all the data in 1 hour, you must write 100 rows per second (360,000 rows / 60 minutes / 60 seconds = 100 rows per second). If each row has up to 1 KB of data, to insert 100 rows per second, you must provision 100 WCUs to your table. If each row has 1.5 KB of data, you need two WCUs to insert one row per second. Therefore, to insert 100 rows per second, you must provision 200 WCUs.

To determine how many WCUs you need to insert one row per second, divide the average row size in bytes by 1024 and round up to the nearest whole number.

For example, if the average row size is 3000 bytes, you need three WCUs to insert one row per second.

```
ROUNDUP(3000 / 1024) = ROUNDUP(2.93) = 3 WCUs
```

### Calculate data load time and capacity

Now that you know the average size and number of rows in your CSV file, you can calculate how many WCUs you need to load the data in a given amount of time, and the approximate time it takes to load all the data in your CSV file using different WCU settings.

For example, if each row in your file is 1 KB and you have 1,000,000 rows in your CSV file, to load the data in 1 hour, you need to provision at least 278 WCUs to your table for that hour.

```
1,000,000 rows * 1 KBs = 1,000,000 KBs  
1,000,000 KBs / 3600 seconds = 277.8 KBs / second = 278 WCUs
```

### Configure provisioned capacity settings

You can set a table's write capacity settings when you create the table or by using the `ALTER TABLE` command. The following is the syntax for altering a table's provisioned capacity settings with the `ALTER TABLE` command.

```
ALTER TABLE catalog.book_awards WITH custom_properties={'capacity_mode':{'throughput_mode':  
'PROVISIONED', 'read_capacity_units': 100, 'write_capacity_units': 278}} ;
```

For the complete language reference, see [the section called "CREATE TABLE" \(p. 200\)](#) and [the section called "ALTER TABLE" \(p. 202\)](#).

## Step 4: Configure DSBulk settings

This section outlines the steps required to configure DSBulk for data upload to Amazon Keyspaces. You configure DSBulk by using a configuration file. You specify the configuration file directly from the command line.

- Create a DSBulk configuration file for the migration to Amazon Keyspaces, in this example we use the file name `dsbulk_keyspaces.conf`. Specify the following settings in the DSBulk configuration file.
  - a. Authentication provider – Create the authentication provider with the `PlainTextAuthProvider` class. `ServiceUserName` and `ServicePassword` should match the user name and password you obtained when you generated the service-specific credentials by following the steps at [the section called "Creating credentials" \(p. 16\)](#).
  - b. Local data center – Set the value for `local-datacenter` to the AWS Region that you're connecting to. For example, if the application is connecting to `cassandra.us-east-2.amazonaws.com`, then set the local data center to `us-east-2`. For all available AWS Regions, see [the section called "Service endpoints" \(p. 17\)](#).



- c. SSL/TLS – Initialize the `SSLConnectionFactory` by adding a section in the configuration file with a single line that specifies the class with `class = DefaultSslConnectionFactory`. Provide the path to `cassandra_truststore.jks` and the password that you created previously.
- d. Set the consistency level to `LOCAL_QUORUM` and turn off the `token_metadata` setting. Other write consistency levels are not supported, for more information see [the section called "Supported Cassandra consistency levels" \(p. 11\)](#).

The following is the complete sample configuration file.

```
datastax-java-driver {  
  basic.contact-points = [ "cassandra.us-east-2.amazonaws.com:9142"  
  advanced.auth-provider {  
    class = PlainTextAuthProvider  
    username = "ServiceUserName"  
    password = "ServicePassword"  
  }  
  basic.load-balancing-policy {  
    local-datacenter = "us-east-2"  
  }  
  
  basic.request {  
    consistency = LOCAL_QUORUM  
    default-idempotence = true  
  }  
  advanced.ssl-engine-factory {  
    class = DefaultSslConnectionFactory  
    truststore-path = "./cassandra_truststore.jks"  
    truststore-password = "my_password"  
    hostname-validation = false  
  }  
  advanced.metadata {  
    schema {  
      token-map.enabled = false  
    }  
  }  
}
```

## Step 5: Run the DSBulk load command

In the final step of this tutorial, you upload the data into Amazon Keyspaces.

To run the DSBulk load command, complete the following steps.

1. Run the following code to upload the data from your csv file to your Amazon Keyspaces table. Make sure to update the path to the application configuration file you created earlier.

```
dsbulk load -f ./dsbulk_keyspaces.conf --connector.csv.url keyspace.table.csv -  
header true --batch.mode <DISABLED> --executor.maxPerSecond <INGESTRATE NUMBER>  
--driver.basic.request.timeout "5 minutes" --driver.advanced.retry-policy.max-  
retries <retry number> --dsbulk.executor.maxInFlight <PROCESS number> -k catalog -t  
book_awards
```

2. The output includes the location of a log file that details successful and unsuccessful operations. The file is stored in the following directory.

```
Operation directory: /home/user_name/logs/UNLOAD_20210308-202317-801911
```

- The log file entries will include metrics, as in the following example. Check to make sure that the number of rows is consistent with the number of rows in your csv file.

total	failed	rows/s	p50ms	p99ms	p999ms
200	0	200	21.63	21.89	21.89

**Important**

Now that you have transferred your data, adjust the capacity mode settings of your target table to match your application's regular traffic patterns. You incur charges at the hourly rate for your provisioned capacity until you change it.

# Amazon Keyspaces (for Apache Cassandra) code examples and tools

This section provides information about Amazon Keyspaces (for Apache Cassandra) code examples and tools.

## Topics

- [Libraries and examples \(p. 69\)](#)
- [Highlighted sample and developer tool repos \(p. 70\)](#)

## Libraries and examples

You can find Amazon Keyspaces open-source libraries and developer tools on GitHub in the [AWS](#) and [AWS samples](#) repos.

## Amazon Keyspaces (for Apache Cassandra) developer toolkit

This repository provides a docker image with helpful developer tools for Amazon Keyspaces. For example, it includes a CQLSHRC file with best practices, an optional AWS authentication expansion for cqlsh, and helper tools to perform common tasks. The toolkit is optimized for Amazon Keyspaces, but also works with Apache Cassandra clusters.

<https://github.com/aws-samples/amazon-keyspaces-toolkit>.

## Amazon Keyspaces (for Apache Cassandra) examples

This repo is our official list of Amazon Keyspaces example code. The repo is subdivided into sections by language (see [Examples](#)). Each language has its own subsection of examples. These examples demonstrate common Amazon Keyspaces service implementations and patterns that you can use when building applications.

<https://github.com/aws-samples/amazon-keyspaces-examples/>.

## AWS Signature Version 4 (SigV4) authentication plugins

The plugins enable you to manage access to Amazon Keyspaces by using AWS Identity and Access Management (IAM) users and roles.

Java: <https://github.com/aws/aws-sigv4-auth-cassandra-java-driver-plugin>.

Node.js: <https://github.com/aws/aws-sigv4-auth-cassandra-nodejs-driver-plugin>.

Python: <https://github.com/aws/aws-sigv4-auth-cassandra-python-driver-plugin>.

Go: <https://github.com/aws/aws-sigv4-auth-cassandra-gocql-driver-plugin>.

## Highlighted sample and developer tool repos

Below are a selection of helpful community tools for Amazon Keyspaces (for Apache Cassandra).

### AWS CloudFormation template to create Amazon CloudWatch dashboard for Amazon Keyspaces (for Apache Cassandra) metrics

This repository provides AWS CloudFormation templates to quickly set up CloudWatch metrics for Amazon Keyspaces. Using this template will allow you to get started more easily by providing deployable prebuilt CloudWatch dashboards with commonly used metrics.

<https://github.com/aws-samples/amazon-keyspaces-cloudwatch-cloudformation-templates>.

### Using Amazon Keyspaces (for Apache Cassandra) with AWS Lambda

The repository contains examples that show how to connect to Amazon Keyspaces from Lambda. Below are some examples.

C#/.NET: <https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/dotnet/datastax-v3/connection-lambda>.

Java: <https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/java/datastax-v4/connection-lambda>.

Another Lambda example that shows how to deploy and use Amazon Keyspaces from a Python Lambda is available from the following repo.

<https://github.com/aws-samples/aws-keyspaces-lambda-python>

### Using Amazon Keyspaces (for Apache Cassandra) with Spring

This is an example that shows you how to use Amazon Keyspaces with Spring Boot.

<https://github.com/aws-samples/amazon-keyspaces-spring-app-example>

### Using Amazon Keyspaces (for Apache Cassandra) with Scala

This is an example that shows how to connect to Amazon Keyspaces using the SigV4 authentication plugin with Scala.

<https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/scala/datastax-v4/connection-sigv4>

## Amazon Keyspaces (for Apache Cassandra) Cassandra query language (CQL) to AWS CloudFormation converter

This package implements a command-line tool for converting Apache Cassandra Query Language (CQL) scripts to AWS CloudFormation (CloudFormation) templates, which allows Amazon Keyspaces schemas to be easily managed in CloudFormation stacks.

<https://github.com/aws/amazon-keyspaces-cql-to-cfn-converter>.

## Amazon Keyspaces (for Apache Cassandra) helpers for Apache Cassandra driver for Java

This repository contains driver policies, examples, and best practices when using the DataStax Java Driver with Amazon Keyspaces (for Apache Cassandra).

<https://github.com/aws-samples/amazon-keyspaces-java-driver-helpers>.

## Amazon Keyspaces (for Apache Cassandra) snappy compression demo

This repository demonstrates how to compress, store, and read/write large objects for faster performance and lower throughput and storage costs.

<https://github.com/aws-samples/amazon-keyspaces-compression-example>.

## Amazon Keyspaces (for Apache Cassandra) and Amazon S3 codec demo

Custom Amazon S3 Codec supports transparent, user-configurable mapping of UUID pointers to Amazon S3 objects.

<https://github.com/aws-samples/amazon-keyspaces-large-object-s3-demo>.

# Serverless resource management in Amazon Keyspaces (for Apache Cassandra)

Amazon Keyspaces (for Apache Cassandra) is serverless. Instead of deploying, managing, and maintaining storage and compute resources for your workload through nodes in a cluster, Amazon Keyspaces allocates storage and read/write throughput resources directly to tables.

This chapter provides details about serverless resource management in Amazon Keyspaces. To learn how to monitor serverless resources with Amazon CloudWatch, see [the section called “Monitoring with CloudWatch”](#) (p. 169).

## Topics

- [Storage in Amazon Keyspaces](#) (p. 72)
- [Read/write capacity modes in Amazon Keyspaces](#) (p. 72)
- [Managing Amazon Keyspaces throughput capacity with Application Auto Scaling](#) (p. 76)

## Storage in Amazon Keyspaces

Amazon Keyspaces (for Apache Cassandra) provisions storage to tables automatically based on the actual data stored in your table. You do not need to provision storage to tables upfront. Amazon Keyspaces scales your table storage up and down automatically as your application writes, updates, and deletes data. Unlike traditional Apache Cassandra clusters, Amazon Keyspaces does not require additional storage to support low-level system operations such as compaction. You pay only for the storage you use.

Amazon Keyspaces configures keyspaces with a replication factor of three by default. You cannot modify the replication factor. Amazon Keyspaces replicates table data three times automatically in multiple AWS Availability Zones for high availability. The per-GB price of Amazon Keyspaces storage already includes replication. See [Amazon Keyspaces \(for Apache Cassandra\) Pricing](#) for more information.

Amazon Keyspaces monitors the size of your tables continuously to determine your storage charges. For more information about how Amazon Keyspaces calculates the billable size of the data, see [the section called “Calculating row size”](#) (p. 91).

## Read/write capacity modes in Amazon Keyspaces

Amazon Keyspaces has two read/write capacity modes for processing reads and writes on your tables:

- On-demand (default)
- Provisioned

The read/write capacity mode that you choose controls how you are charged for read and write throughput and how table throughput capacity is managed.

### Topics

- [On-demand capacity mode \(p. 73\)](#)
- [Provisioned throughput capacity mode \(p. 74\)](#)
- [Managing and viewing capacity modes \(p. 75\)](#)
- [Considerations when changing capacity modes \(p. 76\)](#)

## On-demand capacity mode

Amazon Keyspaces (for Apache Cassandra) *on-demand* capacity mode is a flexible billing option capable of serving thousands of requests per second without capacity planning. This option offers pay-per-request pricing for read and write requests so that you pay only for what you use.

When you choose on-demand mode, Amazon Keyspaces can scale the throughput capacity for your table up to any previously reached traffic level instantly, and then back down when application traffic decreases. If a workload's traffic level hits a new peak, the service adapts rapidly to increase throughput capacity for your table. You can enable on-demand capacity mode for both new and existing tables.

On-demand mode is a good option if any of the following is true:

- You create new tables with unknown workloads.
- You have unpredictable application traffic.
- You prefer the ease of paying for only what you use.

To get started with on-demand mode, you can create a new table or update an existing table to use on-demand capacity mode using the console or with a few lines of Cassandra Query Language (CQL) code. For more information, see [the section called "Tables" \(p. 199\)](#).

### Topics

- [Read request units and write request units \(p. 73\)](#)
- [Peak traffic and scaling properties \(p. 74\)](#)
- [Initial throughput for on-demand capacity mode \(p. 74\)](#)

## Read request units and write request units

With on-demand capacity mode tables, you don't need to specify how much read and write throughput you expect your application to use in advance. Amazon Keyspaces charges you for the reads and writes that you perform on your tables in terms of read request units (RRUs) and write request units (WRUs).

- One *RRU* represents one `LOCAL_QUORUM` read request, or two `LOCAL_ONE` read requests, for a row up to 4 KB in size. If you need to read a row that is larger than 4 KB, the read operation uses additional RRUs. The total number of RRUs required depends on the row size, and whether you want to use `LOCAL_QUORUM` or `LOCAL_ONE` read consistency. For example, reading an 8 KB row requires 2 RRUs using `LOCAL_QUORUM` read consistency, and 1 RRU if you choose `LOCAL_ONE` read consistency.
- One *WRU* represents one write for a row up to 1 KB in size. All writes are using `LOCAL_QUORUM` consistency, and there is no additional charge for using lightweight transactions (LWTs). If you need to write a row that is larger than 1 KB, the write operation uses additional WRUs. The total number of WRUs required depends on the row size. For example, if your row size is 2 KB, you require 2 WRUs to perform one write request.

For information about supported consistency levels, see [the section called "Supported Cassandra consistency levels" \(p. 11\)](#).

## Peak traffic and scaling properties

Amazon Keyspaces tables that use on-demand capacity mode automatically adapt to your application's traffic volume. On-demand capacity mode instantly accommodates up to double the previous peak traffic on a table. For example, your application's traffic pattern might vary between 5,000 and 10,000 `LOCAL_QUORUM` reads per second, where 10,000 reads per second is the previous traffic peak.

With this pattern, on-demand capacity mode instantly accommodates sustained traffic of up to 20,000 reads per second. If your application sustains traffic of 20,000 reads per second, that peak becomes your new previous peak, enabling subsequent traffic to reach up to 40,000 reads per second.

If you need more than double your previous peak on a table, Amazon Keyspaces automatically allocates more capacity as your traffic volume increases. This helps ensure that your table has enough throughput capacity to process the additional requests. However, you might observe insufficient throughput capacity errors if you exceed double your previous peak within 30 minutes.

For example, suppose that your application's traffic pattern varies between 5,000 and 10,000 strongly consistent reads per second, where 20,000 reads per second is the previously reached traffic peak. In this case, the service recommends that you space your traffic growth over at least 30 minutes before driving up to 40,000 reads per second.

To learn more about default quotas for your account and how to increase them, see [Quotas \(p. 209\)](#).

## Initial throughput for on-demand capacity mode

If you create a new table with on-demand capacity mode enabled or switch an existing table to on-demand capacity mode for the first time, the table has the following previous peak settings, even though it hasn't served traffic previously using on-demand capacity mode:

- **Newly created table with on-demand capacity mode:** The previous peak is 2,000 WRUs or 6,000 RRUs. You can drive up to double the previous peak immediately. Doing this enables newly created on-demand tables to serve up to 4,000 WRUs or 12,000 RRUs, or any linear combination of the two.
- **Existing table switched to on-demand capacity mode:** The previous peak is half the previous WCUs and RCUs provisioned for the table or the settings for a newly created table with on-demand capacity mode, whichever is higher.

## Provisioned throughput capacity mode

If you choose *provisioned throughput* capacity mode, you specify the number of reads and writes per second that are required for your application. This helps you manage your Amazon Keyspaces usage to stay at or below a defined request rate to optimize price and maintain predictability. To learn more about automatic scaling for provisioned throughput see [the section called "Managing throughput capacity with Application Auto Scaling" \(p. 76\)](#).

Provisioned throughput capacity mode is a good option if any of the following is true:

- You have predictable application traffic.
- You run applications whose traffic is consistent or ramps up gradually.
- You can forecast capacity requirements to optimize price.

## Read capacity units and write capacity units

For provisioned throughput capacity mode tables, you specify throughput capacity in terms of read capacity units (RCUs) and write capacity units (WCUs):



- One *RCU* represents one `LOCAL_QUORUM` read per second, or two `LOCAL_ONE` reads per second, for a row up to 4 KB in size. If you need to read a row that is larger than 4 KB, the read operation uses additional RCUs.

The total number of RCUs required depends on the row size, and whether you want `LOCAL_QUORUM` or `LOCAL_ONE` reads. For example, if your row size is 8 KB, you require 2 RCUs to sustain one `LOCAL_QUORUM` read per second, and 1 RCU if you choose `LOCAL_ONE` reads.

- One *WCU* represents one write per second for a row up to 1 KB in size. All writes are using `LOCAL_QUORUM` consistency, and there is no additional charge for using lightweight transactions (LWTs). If you need to write a row that is larger than 1 KB, the write operation uses additional WCUs.

The total number of WCUs required depends on the row size. For example, if your row size is 2 KB, you require 2 WCUs to sustain one write request per second.

If your application reads or writes larger rows (up to the Amazon Keyspaces maximum row size of 1 MB), it consumes more capacity units. To learn more about how to estimate the row size, see [the section called “Calculating row size” \(p. 91\)](#). For example, suppose that you create a provisioned table with 6 RCUs and 6 WCUs. With these settings, your application could do the following:

- Perform `LOCAL_QUORUM` reads of up to 24 KB per second (4 KB × 6 RCUs).
- Perform `LOCAL_ONE` reads of up to 48 KB per second (twice as much read throughput).
- Write up to 6 KB per second (1 KB × 6 WCUs).

*Provisioned throughput* is the maximum amount of throughput capacity an application can consume from a table. If your application exceeds your provisioned throughput capacity, you might observe insufficient capacity errors.

For example, a read request that doesn't have enough throughput capacity fails with a `Read_Timeout` exception and is posted to the `ReadThrottleEvents` metric. A write request that doesn't have enough throughput capacity fails with a `Write_Timeout` exception and is posted to the `WriteThrottleEvents` metric.

You can use Amazon CloudWatch to monitor your provisioned and actual throughput metrics and insufficient capacity events. For more information about these metrics, see [the section called “Metrics and dimensions” \(p. 170\)](#).

#### Note

Repeated errors due to insufficient capacity can lead to client-side driver specific exceptions, for example the DataStax Java driver fails with a `NoHostAvailableException`.

To change the throughput capacity settings for tables, you can use the AWS Management Console or the `ALTER TABLE` statement using CQL, for more information see [the section called “ALTER TABLE” \(p. 202\)](#).

To learn more about default quotas for your account and how to increase them, see [Quotas \(p. 209\)](#).

## Managing and viewing capacity modes

You can query the system table in the Amazon Keyspaces system keyspace to review capacity mode information about a table. You can also see whether a table is using on-demand or provisioned throughput capacity mode. If the table is configured with provisioned throughput capacity mode, you can see the throughput capacity provisioned for the table.

#### Example

```
SELECT * from system_schema_mcs.tables where keyspace_name = 'mykeyspace' and table_name = 'mytable';
```

A table configured with on-demand capacity mode returns the following.

```
{
  'capacity_mode': {
    'last_update_to_pay_per_request_timestamp': '1579551547603',
    'throughput_mode': 'PAY_PER_REQUEST'
  }
}
```

A table configured with provisioned throughput capacity mode returns the following.

```
{
  'capacity_mode': {
    'last_update_to_pay_per_request_timestamp': '1579048006000',
    'read_capacity_units': '5000',
    'throughput_mode': 'PROVISIONED',
    'write_capacity_units': '6000'
  }
}
```

The `last_update_to_pay_per_request_timestamp` value is measured in milliseconds.

To change the provisioned throughput capacity for a table, use [the section called “ALTER TABLE” \(p. 202\)](#).

## Considerations when changing capacity modes

When you switch a table from provisioned capacity mode to on-demand capacity mode, Amazon Keyspaces makes several changes to the structure of your table and partitions. This process can take several minutes. During the switching period, your table delivers throughput that is consistent with the previously provisioned WCU and RCU amounts.

When you switch from on-demand capacity mode back to provisioned capacity mode, your table delivers throughput that is consistent with the previous peak reached when the table was set to on-demand capacity mode.

### Note

You can switch between read/write capacity modes once every 24 hours.

## Managing Amazon Keyspaces throughput capacity with Application Auto Scaling

Many database workloads are cyclical in nature or are difficult to predict in advance. For example, consider a social networking app where most of the users are active during daytime hours. The database must be able to handle the daytime activity, but there's no need for the same levels of throughput at night.

Another example might be a new mobile gaming app that is experiencing rapid adoption. If the game becomes very popular, it could exceed the available database resources, which would result in slow performance and unhappy customers. These kinds of workloads often require manual intervention to scale database resources up or down in response to varying usage levels.

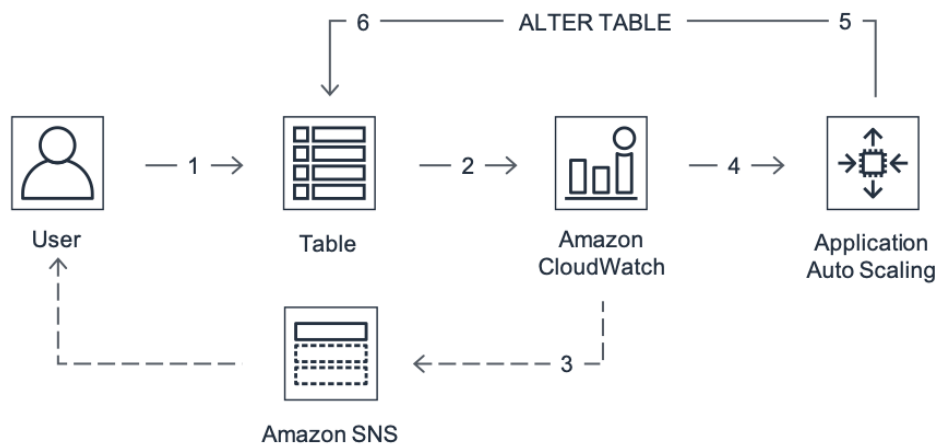
Amazon Keyspaces (for Apache Cassandra) helps you provision throughput capacity efficiently for variable workloads by adjusting throughput capacity automatically in response to actual application traffic. Amazon Keyspaces uses the Application Auto Scaling service to increase and decrease a table's read and write capacity on your behalf. For more information about Application Auto Scaling, see the [Application Auto Scaling User Guide](#).

#### Note

To get started with Amazon Keyspaces automatic scaling quickly, see [the section called "Using the console" \(p. 78\)](#). You cannot manage Amazon Keyspaces scaling policies with Cassandra Query Language (CQL). To learn how to manage Amazon Keyspaces scaling policies programmatically, see [the section called "Managing programmatically" \(p. 81\)](#).

## How Amazon Keyspaces automatic scaling works

The following diagram provides a high-level overview of how Amazon Keyspaces automatic scaling manages throughput capacity for a table.



To enable automatic scaling for a table, you create a *scaling policy*. The scaling policy specifies whether you want to scale read capacity or write capacity (or both), and the minimum and maximum provisioned capacity unit settings for the table.

The scaling policy also defines a *target utilization*. Target utilization is the ratio of consumed capacity units to provisioned capacity units at a point in time, expressed as a percentage. Automatic scaling uses a *target tracking* algorithm to adjust the provisioned throughput of the table upward or downward in response to actual workloads. It does this so that the actual capacity utilization remains at or near your target utilization.

You can set the automatic scaling target utilization values between 20 and 90 percent for your read and write capacity. The default target utilization rate is 70 percent. You can set the target utilization to be a lower percentage if your traffic changes quickly and you want capacity to begin scaling up sooner. You can also set the target utilization rate to a higher rate if your application traffic changes more slowly and you want to reduce the cost of throughput. For more information about scaling policies, see [Target tracking scaling policies for Application Auto Scaling](#).

When you create a scaling policy, Application Auto Scaling creates two pairs of Amazon CloudWatch alarms on your behalf. Each pair represents your upper and lower boundaries for provisioned and consumed throughput settings. These CloudWatch alarms are triggered when the table's actual utilization deviates from your target utilization for a sustained period of time. To learn more about Amazon CloudWatch, see the [Amazon CloudWatch User Guide](#).

When one of the CloudWatch alarms is triggered, Amazon Simple Notification Service (Amazon SNS) sends you a notification (if you have enabled it). The CloudWatch alarm then invokes Application Auto Scaling to evaluate your scaling policy. This in turn issues an Alter Table request to Amazon Keyspaces to adjust the table's provisioned capacity upward or downward as appropriate. To learn more about Amazon SNS notifications, see [Setting up Amazon SNS notifications](#).

Amazon Keyspaces processes the Alter Table request by increasing (or decreasing) the table's provisioned throughput capacity so that it approaches your target utilization.

**Note**

Amazon Keyspaces automatic scaling modifies provisioned throughput settings only when the actual workload stays elevated (or depressed) for a sustained period of several minutes. The Application Auto Scaling target tracking algorithm seeks to keep the target utilization at or near your chosen value over the long term. Sudden, short-duration spikes of activity are accommodated by the table's built-in burst capacity.

## Usage notes

Before you begin using Amazon Keyspaces automatic scaling, you should be aware of the following:

- Amazon Keyspaces automatic scaling can increase read capacity or write capacity as often as necessary, in accordance with your scaling policy. All Amazon Keyspaces quotas remain in effect, as described in [Quotas](#) (p. 209).
- Amazon Keyspaces automatic scaling doesn't prevent you from manually modifying provisioned throughput settings. These manual adjustments don't affect any existing CloudWatch alarms that are attached to the scaling policy.
- If you use the console to create a table with provisioned throughput capacity, Amazon Keyspaces automatic scaling is enabled by default. You can modify your automatic scaling settings at any time. For more information, see [the section called "Using the console"](#) (p. 78).
- If you are using AWS CloudFormation to create scaling policies, you should manage the scaling policies from AWS CloudFormation so that the stack is in sync with the stack template. If you change scaling policies from Amazon Keyspaces or Application Auto Scaling, they will get overwritten with the original values from the AWS CloudFormation stack template when the stack is reset.
- If you use CloudTrail to monitor Amazon Keyspaces automatic scaling, you might see alerts for calls made by Application Auto Scaling as part of its configuration validation process. You can filter out these alerts by using the `invokedBy` field, which contains `application-autoscaling.amazonaws.com` for these validation checks.

## Managing Amazon Keyspaces automatic scaling policies with the console

You can use the console to enable Amazon Keyspaces automatic scaling for new and existing tables. You can also use the console to modify automatic scaling settings or disable automatic scaling.

**Note**

For more advanced features like setting scale-in and scale-out cooldown times, use the AWS Command Line Interface (AWS CLI) to manage Amazon Keyspaces scaling policies programmatically. For more information, see [Managing Amazon Keyspaces scaling policies programmatically](#) (p. 81).

**Topics**

- [Before you begin: Granting user permissions for Amazon Keyspaces automatic scaling](#) (p. 79)
- [Creating a new table with Amazon Keyspaces automatic scaling enabled](#) (p. 79)
- [Enabling Amazon Keyspaces automatic scaling on existing tables](#) (p. 80)

- [Modifying or disabling Amazon Keyspaces automatic scaling settings \(p. 80\)](#)
- [Viewing Amazon Keyspaces automatic scaling activities on the console \(p. 81\)](#)

## Before you begin: Granting user permissions for Amazon Keyspaces automatic scaling

To get started, confirm that the user has the appropriate permissions to create and manage automatic scaling settings. In AWS Identity and Access Management (IAM), the AWS managed policy `AmazonKeyspacesFullAccess` is required to manage Amazon Keyspaces scaling policies.

### Important

`application-autoscaling:*` permissions are required to disable automatic scaling on a table. Automatic scaling must be disabled using [the section called “Disable automatic scaling on an existing table: Deregister a scalable target” \(p. 86\)](#) before you delete a table.

To set up an IAM user for Amazon Keyspaces console access and Amazon Keyspaces automatic scaling, add the following policy.

### To attach the `AmazonKeyspacesFullAccess` policy

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. On the IAM console dashboard, choose **Users**, and then choose your IAM user from the list.
3. On the **Summary** page, choose **Add permissions**.
4. Choose **Attach existing policies directly**.
5. From the list of policies, choose **AmazonKeyspacesFullAccess**, and then choose **Next: Review**.
6. Choose **Add permissions**.

## Creating a new table with Amazon Keyspaces automatic scaling enabled

### Note

Amazon Keyspaces automatic scaling requires the presence of a service-linked role (`AWSServiceRoleForApplicationAutoScaling_CassandraTable`) that performs automatic scaling actions on your behalf. This role is created automatically for you. For more information, see [the section called “Using service-linked roles” \(p. 166\)](#).

### To create a new table with automatic scaling enabled

1. Sign in to the AWS Management Console, and open the Amazon Keyspaces console at <https://console.aws.amazon.com/keyspaces/home>.
2. In the navigation pane, choose **Tables**, and then choose **Create table**.
3. On the **Create table** page in the **Table details** section, select a keyspace and provide a name for the new table.
4. In the **Schema** section, create the schema for your table.
5. In the **Table settings** section, choose **Customize settings**.
6. Continue to **Read/write capacity settings**.
7. For **Capacity mode**, choose **Provisioned**.
8. In the **Read capacity** section, confirm that **Scale automatically** is selected.

In this step, you select the minimum and maximum read capacity units for the table, as well as the target utilization.

- **Minimum capacity units** – Enter the value for the minimum level of throughput that the table should always be ready to support. The value must be between 1 and the maximum throughput per second quota for your account (40,000 by default).
- **Maximum capacity units** – Enter the maximum amount of throughput you want to provision for the table. The value must be between 1 and the maximum throughput per second quota for your account (40,000 by default).
- **Target utilization** – Enter a target utilization rate between 20% and 90%. When traffic exceeds the defined target utilization rate, capacity is automatically scaled up. When traffic falls below the defined target, it is automatically scaled down again.

**Note**

To learn more about default quotas for your account and how to increase them, see [Quotas \(p. 209\)](#).

9. In the **Write capacity** section, choose the same settings as defined in the previous step for read capacity, or enter write capacity values manually.
10. Choose **Create table**. Your table is created with the specified automatic scaling parameters.

**Note**

Enabling Amazon Keyspaces automatic scaling is not included in the Cassandra Query Language (CQL) command shown in the console. It is completed with a separate API call that registers the table as a scalable target with Application Auto Scaling.

## Enabling Amazon Keyspaces automatic scaling on existing tables

**Note**

Amazon Keyspaces automatic scaling requires the presence of a service-linked role (AWSServiceRoleForApplicationAutoScaling\_CassandraTable) that performs automatic scaling actions on your behalf. This role is created automatically for you. For more information, see [the section called “Using service-linked roles” \(p. 166\)](#).

### To enable Amazon Keyspaces automatic scaling for an existing table

1. Sign in to the AWS Management Console, and open the Amazon Keyspaces console at <https://console.aws.amazon.com/keyspaces/home>.
2. Choose the table that you want to work with, and then choose **Change capacity settings**.
3. Select **Scale automatically** and see step 6 in [Creating a new table with Amazon Keyspaces automatic scaling enabled \(p. 79\)](#) to edit read and write capacity.
4. When the automatic scaling settings are defined, choose **Save**.

**Note**

Enabling Amazon Keyspaces automatic scaling is not included in the CQL command shown in the console. It is completed with a separate API call that registers the table as a scalable target with Application Auto Scaling.

## Modifying or disabling Amazon Keyspaces automatic scaling settings

You can use the AWS Management Console to modify your Amazon Keyspaces automatic scaling settings. To do this, go to the table you want to edit and then choose **Change capacity settings**. You can

modify the settings in the **Read capacity** or **Write capacity** sections. For more information about these settings, see [Creating a new table with Amazon Keyspaces automatic scaling enabled](#) (p. 79).

To disable Amazon Keyspaces automatic scaling, deselect **Scale automatically**. Disabling automatic scaling deregisters the table as a scalable target. To delete the service-linked role used by Application Auto Scaling to access the table, follow the steps in [??? \(p. 167\)](#).

**Note**

To delete the service-linked role used by Application Auto Scaling, you must disable automatic scaling on all tables in the account across all AWS Regions.

## Viewing Amazon Keyspaces automatic scaling activities on the console

You can monitor how Amazon Keyspaces automatic scaling uses resources by using Amazon CloudWatch, which generates metrics about your usage and performance. Follow the steps in the [Application Auto Scaling User Guide](#) to create a CloudWatch dashboard.

## Managing Amazon Keyspaces scaling policies programmatically

To update and manage Amazon Keyspaces automatic scaling settings programmatically, you can use the AWS Command Line Interface (AWS CLI) or the AWS API. Amazon Keyspaces scaling policies cannot be managed using Cassandra Query Language (CQL). This topic gives an overview of the automatic scaling tasks that you can manage programmatically using the AWS CLI.

For more information about the Application Auto Scaling AWS CLI commands described in this topic, see [application-autoscaling](#) in the *AWS CLI Command Reference*. For more information about using the AWS API, see the [Application Auto Scaling API Reference](#).

**Topics**

- [Before you begin](#) (p. 81)
- [Enable automatic scaling on an existing table: Register a scalable target](#) (p. 82)
- [View scalable targets registered with Application Auto Scaling](#) (p. 82)

## Before you begin

You need to complete the following tasks before you can start.

### Configure permissions

If you haven't already done so, you must configure the appropriate permissions for the user to create and manage automatic scaling settings. In AWS Identity and Access Management (IAM), the AWS managed policy `AmazonKeyspacesFullAccess` is required to manage Amazon Keyspaces scaling policies. For detailed steps see [the section called "Before you begin: Granting user permissions for Amazon Keyspaces automatic scaling"](#) (p. 79).

### Install the AWS CLI

If you haven't already done so, you must install and configure the AWS CLI. To do this, go to the [AWS Command Line Interface User Guide](#) and follow these instructions:

- [Installing the AWS CLI](#)
- [Configuring the AWS CLI](#)

## Enable automatic scaling on an existing table: Register a scalable target

For an existing Amazon Keyspaces table, you can register the table's write or read capacity as a scalable target with Application Auto Scaling. This allows Application Auto Scaling to adjust the provisioned write or read capacity for the table that you specify. In the following example, we register *mytable* as a scalable target with write capacity within the range of 5–10 capacity units.

### Note

Amazon Keyspaces automatic scaling requires the presence of a service-linked role (AWSServiceRoleForApplicationAutoScaling\_CassandraTable) that performs automatic scaling actions on your behalf. This role is created automatically for you. For more information, see [the section called “Using service-linked roles”](#) (p. 166).

Enter the following command to register a table as a scalable target.

```
aws application-autoscaling register-scalable-target \
  --service-namespace cassandra \
  --resource-id "keyspace/mykeyspace/table/mytable" \
  --scalable-dimension "cassandra:table:WriteCapacityUnits" \
  --min-capacity 5 \
  --max-capacity 10
```

## View scalable targets registered with Application Auto Scaling

To view details of a registration, use the following command.

```
aws application-autoscaling describe-scalable-targets
  --service-namespace cassandra
  --resource-ids "keyspace/mykeyspace/table/mytable"
```

The output for this command looks like this.

```
{
  "ScalableTargets": [
    {
      "ServiceNamespace": "cassandra",
      "ResourceId": "keyspace/mykeyspace/table/mytable",
      "ScalableDimension": "cassandra:table:WriteCapacityUnits",
      "MinCapacity": 5,
      "MaxCapacity": 10,
      "RoleARN": "arn:aws:iam::012345678910:role/aws-
service-role/cassandra.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_CassandraTable",
      "CreationTime": 1587495787.412,
      "SuspendedState": {
        "DynamicScalingInSuspended": false,
        "DynamicScalingOutSuspended": false,
        "ScheduledScalingSuspended": false
      }
    }
  ]
}
```

## View scalable targets registered with Application Auto Scaling

To view details of a registration, use the following command.

```
aws application-autoscaling describe-scalable-targets \
```



```
--service-namespace cassandra \  
--resource-id "keyspace\mytable"
```

## Create a scaling policy

To manage automatic scaling settings programmatically, you create a scaling policy for the table. The policy defines the conditions that direct Application Auto Scaling to adjust your table's provisioned throughput, and what actions to take when it does so. In this example, you associate this policy with the scalable target that you have previously defined (write capacity units for the *mytable* table).

The policy contains the following elements:

- **PredefinedMetricSpecification** – The metric that Application Auto Scaling is allowed to adjust. For Amazon Keyspaces, the following values are valid values for **PredefinedMetricType**:
  - **CassandraReadCapacityUtilization**
  - **CassandraWriteCapacityUtilization**
- **ScaleOutCooldown** – A scale-out activity increases the provisioned throughput of your table. To add a cooldown period for scale-out activities, specify a value, in seconds, for **ScaleOutCooldown**. The default value is 0. For more information, see [Target Tracking Scaling Policies in the Application Auto Scaling User Guide](#).
- **ScaleInCooldown** – A scale-in activity decreases the provisioned throughput of your table. To add a cooldown period for scale-in activities, specify a value, in seconds, for **ScaleInCooldown**. The default value is 0. For more information, see [Target Tracking Scaling Policies in the Application Auto Scaling User Guide](#).
- **TargetValue** – Application Auto Scaling ensures that the ratio of consumed capacity to provisioned capacity stays at or near this value. You define **TargetValue** as a percentage.

### Note

To further understand how **TargetValue** works, suppose that you have a table with a provisioned throughput setting of 200 write capacity units. You decide to create a scaling policy for this table, with a **TargetValue** of 70 percent. Now suppose that you begin driving write traffic to the table so that the actual write throughput is 150 capacity units. The consumed-to-provisioned ratio is now (150 / 200), or 75 percent. This ratio exceeds your target, so Application Auto Scaling increases the provisioned write capacity to 215 so that the ratio is (150 / 215), or 69.77 percent—as close to your **TargetValue** as possible, but not exceeding it.

For *mytable*, you set **TargetValue** to 50 percent. Application Auto Scaling adjusts the table's provisioned throughput within the range of 5–10 capacity units (see [Enable automatic scaling on an existing table: Register a scalable target \(p. 82\)](#)) so that the consumed-to-provisioned ratio remains at or near 50 percent. You set the values for **ScaleOutCooldown** and **ScaleInCooldown** to 60 seconds.

1. Create a file that contains the policy you want to apply to the table as in the following example. Then save the file, for this example, with the name `scaling-policy.json`.

```
{  
  "PredefinedMetricSpecification": {  
    "PredefinedMetricType": "CassandraWriteCapacityUtilization"  
  },  
  "ScaleOutCooldown": 60,  
  "ScaleInCooldown": 60,  
  "TargetValue": 50.0  
}
```

2. Use the following AWS CLI command to create the policy.

```
aws application-autoscaling put-scaling-policy \  

```

```
--service-namespace cassandra \  
--resource-id "keyspace/mykeyspace/table/mytable" \  
--scalable-dimension "cassandra:table:WriteCapacityUnits" \  
--policy-name "MyScalingPolicy" \  
--policy-type "TargetTrackingScaling" \  
--target-tracking-scaling-policy-configuration file://scaling-policy.json
```

In the output of this command, you can see the CloudWatch alarms that Application Auto Scaling creates—one each for the consumed upper and lower capacity, and one each for the upper and lower boundary of the scaling target range.

```
{  
  "PolicyARN": "arn:aws:autoscaling:us-  
west-2:012345678910:scalingPolicy:8d606c33-2078-4f37-8305-36e89c56a779:resource/cassandra/  
keyspace/mykeyspace/table/mytable:policyName/MyScalingPolicy",  
  "Alarms": [  
    {  
      "AlarmName": "TargetTracking-keyspace/mykeyspace/table/mytable-AlarmHigh-  
d421fec6-fa82-44b4-aab6-6a9bfb6f0ced",  
      "AlarmARN": "arn:aws:cloudwatch:us-west-2:012345678910:alarm:TargetTracking-  
keyspace/mykeyspace/table/mytable-AlarmHigh-d421fec6-fa82-44b4-aab6-6a9bfb6f0ced"  
    },  
    {  
      "AlarmName": "TargetTracking-keyspace/mykeyspace/table/mytable-  
AlarmLow-04479372-e50b-4652-a06d-b3055744ae23",  
      "AlarmARN": "arn:aws:cloudwatch:us-west-2:012345678910:alarm:TargetTracking-  
keyspace/mykeyspace/table/mytable-AlarmLow-04479372-e50b-4652-a06d-b3055744ae23"  
    },  
    {  
      "AlarmName": "TargetTracking-keyspace/mykeyspace/table/mytable-  
MCSProvisionedCapacityHigh-c6a26783-837e-4f70-919e-1fbe4362b6ab",  
      "AlarmARN": "arn:aws:cloudwatch:us-west-2:012345678910:alarm:TargetTracking-  
keyspace/mykeyspace/table/mytable-MCSProvisionedCapacityHigh-  
c6a26783-837e-4f70-919e-1fbe4362b6ab"  
    },  
    {  
      "AlarmName": "TargetTracking-keyspace/mykeyspace/table/mytable-  
MCSProvisionedCapacityLow-e1ce121a-48ea-4148-ace2-25c5f854c215",  
      "AlarmARN": "arn:aws:cloudwatch:us-west-2:012345678910:alarm:TargetTracking-  
keyspace/mykeyspace/table/mytable-MCSProvisionedCapacityLow-e1ce121a-48ea-4148-  
ace2-25c5f854c215"  
    }  
  ]  
}
```

## View a scaling policy

You can use the following AWS CLI command to view the details of existing scaling policies:

```
aws application-autoscaling describe-scaling-policies \  
--service-namespace cassandra \  
--resource-ids "keyspace/mykeyspace/table/mytable" \  
--policy-name "MyScalingPolicy"
```

In the output of this command, you can see the details of the scaling policy, as well as the CloudWatch alarms that Application Auto Scaling creates—one each for the consumed upper and lower capacity, and one each for the upper and lower boundary of the scaling target range.

```
{
```

```

    "ScalingPolicies": [
      {
        "PolicyARN": "arn:aws:autoscaling:us-
west-2:012345678910:scalingPolicy:8d606c33-2078-4f37-8305-36e89c56a779:resource/cassandra/
keyspace/mykeyspace/table/mytable:policyName/MyScalingPolicy",
        "PolicyName": "MyScalingPolicy",
        "ServiceNamespace": "cassandra",
        "ResourceId": "keyspace/mykeyspace/table/mytable",
        "ScalableDimension": "cassandra:table:WriteCapacityUnits",
        "PolicyType": "TargetTrackingScaling",
        "TargetTrackingScalingPolicyConfiguration": {
          "TargetValue": 50.0,
          "PredefinedMetricSpecification": {
            "PredefinedMetricType": "CassandraWriteCapacityUtilization"
          },
          "ScaleOutCooldown": 60,
          "ScaleInCooldown": 60
        },
        "Alarms": [
          {
            "AlarmName": "TargetTracking-keyspace/mykeyspace/table/mytable-
AlarmHigh-d421fec6-fa82-44b4-aab6-6a9bfb6f0ced",
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:012345678910:alarm:TargetTracking-keyspace/mykeyspace/table/mytable-AlarmHigh-
d421fec6-fa82-44b4-aab6-6a9bfb6f0ced"
          },
          {
            "AlarmName": "TargetTracking-keyspace/mykeyspace/table/mytable-
AlarmLow-04479372-e50b-4652-a06d-b3055744ae23",
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:012345678910:alarm:TargetTracking-keyspace/mykeyspace/table/mytable-
AlarmLow-04479372-e50b-4652-a06d-b3055744ae23"
          },
          {
            "AlarmName": "TargetTracking-keyspace/mykeyspace/table/mytable-
MCSProvisionedCapacityHigh-c6a26783-837e-4f70-919e-1fbe4362b6ab",
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:012345678910:alarm:TargetTracking-keyspace/mykeyspace/table/mytable-
MCSProvisionedCapacityHigh-c6a26783-837e-4f70-919e-1fbe4362b6ab"
          },
          {
            "AlarmName": "TargetTracking-keyspace/mykeyspace/table/mytable-
MCSProvisionedCapacityLow-e1ce121a-48ea-4148-ace2-25c5f854c215",
            "AlarmARN": "arn:aws:cloudwatch:us-
west-2:012345678910:alarm:TargetTracking-keyspace/mykeyspace/table/mytable-
MCSProvisionedCapacityLow-e1ce121a-48ea-4148-ace2-25c5f854c215"
          }
        ],
        "CreationTime": 1587497009.591
      }
    ]
  }
}

```

## View Application Auto Scaling actions

You can view the Application Auto Scaling actions that are initiated on your behalf with the following AWS CLI command.

```
aws application-autoscaling describe-scaling-activities \
--service-namespace cassandra
```

If you run this command on a table where automatic scaling is changing the write capacity, you would see output similar to this:

```
...
{
  "ScalableDimension": "cassandra:table:WriteCapacityUnits",
  "Description": "Setting write capacity units to 10.",
  "ResourceId": "keyspace/mykeyspace/table/mytable",
  "ActivityId": "0cc6fb03-2a7c-4b51-b67f-217224c6b656",
  "StartTime": 1489088210.175,
  "ServiceNamespace": "cassandra",
  "EndTime": 1489088246.85,
  "Cause": "monitor alarm AutoScaling-keyspace/mykeyspace/table/mytable-
AlarmHigh-1bb3c8db-1b97-4353-baf1-4def76f4e1b9 in state ALARM triggered policy
MyScalingPolicy",
  "StatusMessage": "Successfully set write capacity units to 10. Change successfully
fulfilled by cassandra.",
  "StatusCode": "Successful"
},
...
```

This sample output indicates that Application Auto Scaling has issued an `Alter Table` request to Amazon Keyspaces to change the write capacity.

## Delete a scaling policy

Delete the scaling policy for *mytable*. If you no longer need to scale your table's write capacity, you should consider deleting your scaling policy so that Amazon Keyspaces does not continue modifying your table's write capacity settings. You can either delete your scaling infrastructure by deregistering the scalable target, or delete just your scaling policy and keep your scalable target registered to use later.

The following command deletes the specified target tracking scaling policy. It also deletes the CloudWatch alarms that Application Auto Scaling created on your behalf.

```
aws application-autoscaling delete-scaling-policy \
  --service-namespace cassandra \
  --resource-id "keyspace/mykeyspace/table/mytable" \
  --scalable-dimension "cassandra:table:WriteCapacityUnits" \
  --policy-name "MyScalingPolicy"
```

## Disable automatic scaling on an existing table: Deregister a scalable target

Deregister the scalable target with the following AWS CLI command. This also deletes the scaling policy attached to this target.

```
aws application-autoscaling deregister-scalable-target \
  --service-namespace cassandra \
  --resource-id "keyspace/mykeyspace/table/mytable" \
  --scalable-dimension "cassandra:table:WriteCapacityUnits"
```

# Working with keyspaces, tables, and rows in Amazon Keyspaces (for Apache Cassandra)

This chapter provides details about working with keyspaces, tables, rows, and more in Amazon Keyspaces (for Apache Cassandra). To learn how to monitor keyspaces and tables with Amazon CloudWatch, see [the section called “Monitoring with CloudWatch” \(p. 169\)](#).

## Topics

- [Working with keyspaces in Amazon Keyspaces \(p. 87\)](#)
- [Working with tables in Amazon Keyspaces \(p. 88\)](#)
- [Working with rows in Amazon Keyspaces \(p. 91\)](#)
- [Working with queries in Amazon Keyspaces \(p. 93\)](#)

## Working with keyspaces in Amazon Keyspaces

This section provides details about working with keyspaces in Amazon Keyspaces (for Apache Cassandra).

## Topics

- [Creating keyspaces in Amazon Keyspaces \(p. 87\)](#)

## Creating keyspaces in Amazon Keyspaces

Amazon Keyspaces performs data definition language (DDL) operations, such as creating and deleting keyspaces, asynchronously. You can monitor the creation status of new keyspaces in the AWS Management Console, which will indicate when a keyspace is pending or active. You can also monitor the creation status of a new keyspace programmatically by using the system schema table. A keyspace will become visible in the system schema once it is ready for use. The recommended design pattern to check when a new keyspace is ready for use is to poll the Amazon Keyspaces system schema tables (system\_schema\_mcs.\*). For a list of DDL statements for keyspaces, see the [the section called “Keyspaces” \(p. 198\)](#) section in the CQL language reference.

The following query shows whether a keyspace has been successfully created.

```
SELECT * FROM system_schema_mcs.keyspaces WHERE keyspace_name = 'mykeyspace';
```

For a keyspace that has been successfully created the output of the query looks like the following.

```
keyspace_name | durable_writes | replication
-----+-----+-----
mykeyspace | true | {...} 1 item
```

## Working with tables in Amazon Keyspaces

This section provides details about working with tables in Amazon Keyspaces (for Apache Cassandra).

### Topics

- [Creating tables in Amazon Keyspaces \(p. 88\)](#)
- [Static columns in Amazon Keyspaces \(p. 88\)](#)

## Creating tables in Amazon Keyspaces

Amazon Keyspaces performs data definition language (DDL) operations, such as creating and deleting tables, asynchronously. You can monitor the creation status of new tables in the AWS Management Console, which indicates when a table is pending or active. You can also monitor the creation status of a new table programmatically by using the system schema table.

A table shows as active in the system schema when it's ready for use. The recommended design pattern to check when a new table is ready for use is to poll the Amazon Keyspaces system schema tables (`system_schema_mcs.*`). For a list of DDL statements for tables, see the [the section called "Tables" \(p. 199\)](#) section in the CQL language reference.

The following query shows the status of a table.

```
SELECT keyspace_name, table_name, status FROM system_schema_mcs.tables WHERE keyspace_name  
= 'mykeyspace' AND table_name = 'mytable';
```

For a table that is still being created and is pending, the output of the query looks like this.

```
keyspace_name | table_name | status  
-----+-----+-----  
mykeyspace | mytable | CREATING
```

For a table that has been successfully created and is active, the output of the query looks like the following.

```
keyspace_name | table_name | status  
-----+-----+-----  
mykeyspace | mytable | ACTIVE
```

## Static columns in Amazon Keyspaces

When you declare a column in an Amazon Keyspaces table as static, the value stored in this column is shared between all rows in a logical partition. When you update the value of this column, Amazon Keyspaces applies the change automatically to all rows in the partition.

This section describes how to calculate the encoded size of data when you're writing to static columns. This process is handled separately from the process that writes data to the nonstatic columns of a row. In

addition to size quotas for static data, read and write operations on static columns also affect metering and throughput capacity for tables independently.

## Calculating static column size per logical partition in Amazon Keyspaces

This section provides details about how to estimate the encoded size of static columns in Amazon Keyspaces. The encoded size is used when you're calculating your bill and quota use. You should also use the encoded size when you calculate provisioned throughput capacity requirements for tables. To calculate the encoded size of static columns in Amazon Keyspaces, you can use the following guidelines.

- Partition keys can contain up to 2048 bytes of data. Each key column in the partition key requires up to 3 bytes of metadata. These metadata bytes count towards your static data size quota of 1 MB per partition. When calculating the size of your static data, you should assume that each partition key column uses the full 3 bytes of metadata.
- Use the raw size of the static column data values based on the data type. For more information about data types, see [the section called "Data types" \(p. 194\)](#).
- Add 104 bytes to the size of the static data for metadata.
- Clustering columns and regular, nonprimary key columns do not count towards the size of static data. To learn how to estimate the size of nonstatic data within rows, see [the section called "Calculating row size" \(p. 91\)](#).

The total encoded size of a static column is based on the following formula:

```
partition key columns + static columns + metadata = total encoded size of static data
```

Consider the following example of a table where all columns are of type integer. The table has two partition key columns, two clustering columns, one regular column, and one static column.

```
CREATE TABLE mykeyspace.mytable(pk_col1 int, pk_col2 int, ck_col1 int, ck_col2 int,  
    reg_col1 int, static_col1 int static, primary key((pk_col1, pk_col2),ck_col1, ck_col2));
```

In this example, we calculate the size of static data of the following statement:

```
INSERT INTO mykeyspace.mytable (pk_col1, pk_col2, static_col1) values(1,2,6);
```

To estimate the total bytes required by this write operation, you can use the following steps.

1. Calculate the size of a partition key column by adding the bytes for the data type stored in the column and the metadata bytes. Repeat this for all partition key columns.
  - a. Calculate the size of the first column of the partition key (pk\_col1):

```
4 bytes for the integer data type + 3 bytes for partition key metadata = 7 bytes
```

- b. Calculate the size of the second column of the partition key (pk\_col2):

```
4 bytes for the integer data type + 3 bytes for partition key metadata = 7 bytes
```

- c. Add both columns to get the total estimated size of the partition key columns:

```
7 bytes + 7 bytes = 14 bytes for the partition key columns
```

2. Add the size of the static columns. In this example, we only have one static column that stores an integer (which requires 4 bytes).
3. Finally, to get the total encoded size of the static column data, add up the bytes for the primary key columns and static columns, and add the additional 104 bytes for metadata:

```
14 bytes for the partition key columns + 4 bytes for the static column + 104 bytes for  
metadata = 122 bytes.
```

You can also update static and nonstatic data with the same statement. To estimate the total size of the write operation, you must first calculate the size of the nonstatic data update. Then calculate the size of the row update as shown in the example at [the section called “Calculating row size” \(p. 91\)](#), and add the results.

In this case, you can write a total of 2 MB—1 MB is the maximum row size quota, and 1 MB is the quota for the maximum static data size per logical partition.

To calculate the total size of an update of static and nonstatic data in the same statement, you can use the following formula:

```
(partition key columns + static columns + metadata = total encoded size of static data)  
+ (partition key columns + clustering columns + regular columns + row metadata = total  
encoded size of row)  
= total encoded size of data written
```

Consider the following example of a table where all columns are of type integer. The table has two partition key columns, two clustering columns, one regular column, and one static column.

```
CREATE TABLE mykeyspace.mytable(pk_col1 int, pk_col2 int, ck_col1 int, ck_col2 int,  
    reg_col1 int, static_col1 int static, primary key((pk_col1, pk_col2),ck_col1, ck_col2));
```

In this example, we calculate the size of data when we write a row to the table, as shown in the following statement:

```
INSERT INTO mykeyspace.mytable (pk_col1, pk_col2, ck_col1, ck_col2, reg_col1, static_col1)  
values(2,3,4,5,6,7);
```

To estimate the total bytes required by this write operation, you can use the following steps.

1. Calculate the total encoded size of static data as shown earlier. In this example, it's 122 bytes.
2. Add the size of the total encoded size of the row based on the update of nonstatic data, following the steps at [the section called “Calculating row size” \(p. 91\)](#). In this example, the total size of the row update is 134 bytes.

```
122 bytes for static data + 134 bytes for nonstatic data = 256 bytes.
```

## Metering read/write operations of static data in Amazon Keyspaces

Static data is associated with logical partitions in Cassandra, not individual rows. Logical partitions in Amazon Keyspaces can be virtually unbound in size by spanning across multiple physical storage partitions. As a result, Amazon Keyspaces meters write operations on static and nonstatic data separately. Furthermore, writes that include both static and nonstatic data require additional underlying operations to provide data consistency.



If you perform a mixed write operation of both static and nonstatic data, this results in two separate write operations—one for nonstatic and one for static data. This applies to both on-demand and provisioned read/write capacity modes.

The following example provides details about how to estimate the required read capacity units (RCUs) and write capacity units (WCUs) when you're calculating provisioned throughput capacity requirements for tables in Amazon Keyspaces that have static columns. You can estimate how much capacity your table needs to process writes that include both static and nonstatic data by using the following formula:

$$2 \times \text{WCUs required for nonstatic data} + 2 \times \text{WCUs required for static data}$$

For example, if your application writes 27 KBs of data per second and each write includes 25.5 KBs of nonstatic data and 1.5 KBs of static data, then your table requires 56 WCUs ( $2 \times 26 \text{ WCUs} + 2 \times 2 \text{ WCUs}$ ).

Amazon Keyspaces meters the reads of static and nonstatic data the same as reads of multiple rows. As a result, the price of reading static and nonstatic data in the same operation is based on the aggregate size of the data processed to perform the read.

To learn how to monitor serverless resources with Amazon CloudWatch, see [the section called "Monitoring with CloudWatch" \(p. 169\)](#).

## Working with rows in Amazon Keyspaces

This section provides details about working with rows in Amazon Keyspaces (for Apache Cassandra). Tables are the primary data structures in Amazon Keyspaces and data in tables is organized into columns and rows.

### Topics

- [Calculating row size in Amazon Keyspaces \(p. 91\)](#)

## Calculating row size in Amazon Keyspaces

Amazon Keyspaces (for Apache Cassandra) provides fully managed storage that offers single-digit millisecond read and write performance and stores data durably across multiple AWS Availability Zones. Amazon Keyspaces attaches metadata to all rows and primary key columns to support efficient data access and high availability.

This section provides details about how to estimate the encoded size of rows in Amazon Keyspaces. The encoded row size is used when calculating your bill and quota use. You should also use the encoded row size when calculating provisioned throughput capacity requirements for tables. To calculate the encoded size of rows in Amazon Keyspaces, you can use the following guidelines.

- Partition keys can contain up to 2048 bytes of data. Each key column in the partition key requires up to 3 bytes of metadata. These metadata bytes count towards your 1 MB row size quota. When calculating the size of your row, you should assume each partition key column uses the full 3 bytes of metadata.
- Each row can have up to 850 bytes of clustering column data and each clustering column requires up to 4 bytes for metadata. These metadata bytes count towards your 1 MB row size quota. When calculating the size of your row, you should assume each clustering column uses the full 4 bytes of metadata.
- For regular, nonstatic, nonprimary key columns, use the raw size of the cell data based on the data type. For more information about data types, see [the section called "Data types" \(p. 194\)](#).
- Static column data does not count towards the maximum row size of 1 MB. To calculate the data size of static columns, see [the section called "Calculating static column size per logical partition" \(p. 89\)](#).

- Add 100 bytes to the size of each row for row metadata.

The total size of an encoded row of data is based on the following formula:

```
partition key columns + clustering columns + regular columns + row metadata = total encoded  
size of row
```

Consider the following example of a table where all columns are of type integer. The table has two partition key columns, two clustering columns, and one regular column.

```
CREATE TABLE mykeyspace.mytable(pk_col1 int, pk_col2 int, ck_col1 int, ck_col2 int,  
reg_col1 int, primary key((pk_col1, pk_col2),ck_col1, ck_col2));
```

In this example, we calculate the size of data when we write a row to the table as shown in the following statement:

```
INSERT INTO mykeyspace.mytable (pk_col1, pk_col2, ck_col1, ck_col2, reg_col1)  
values(1,2,3,4,5);
```

To estimate the total bytes required by this write operation, you can use the following steps.

1. Calculate the size of a partition key column by adding the bytes for the data type stored in the column and the metadata bytes. Repeat this for all partition key columns.

- a. Calculate the size of the first column of the partition key (pk\_col1):

```
4 bytes for the integer data type + 3 bytes for partition key metadata = 7 bytes
```

- b. Calculate the size of the second column of the partition key (pk\_col2):

```
4 bytes for the integer data type + 3 bytes for partition key metadata = 7 bytes
```

- c. Add both columns to get the total estimated size of the partition key columns:

```
7 bytes + 7 bytes = 14 bytes for the partition key columns
```

2. Calculate the size of the clustering column by adding the bytes for the data type stored in the column and the metadata bytes. Repeat this for all clustering columns.

- a. Calculate the size of the first column of the clustering column (ck\_col1):

```
4 bytes for the integer data type + 4 bytes for clustering column metadata = 8  
bytes
```

- b. Calculate the size of the second column of the clustering column (ck\_col2):

```
4 bytes for the integer data type + 4 bytes for clustering column metadata = 8  
bytes
```

- c. Add both columns to get the total estimated size of the clustering columns:

```
8 bytes + 8 bytes = 16 bytes for the clustering columns
```

3. Add the size of the regular columns. In this example we only have one column that stores an integer, which requires 4 bytes.

4. Finally, to get the total encoded row size, add up the bytes for all columns and add the additional 100 bytes for row metadata:

```
14 bytes for the partition key columns + 16 bytes for clustering columns + 4 bytes for  
the regular column + 100 bytes for row metadata = 134 bytes.
```

To learn how to monitor serverless resources with Amazon CloudWatch, see [the section called “Monitoring with CloudWatch”](#) (p. 169).

## Working with queries in Amazon Keyspaces

This section gives an introduction into working with queries in Amazon Keyspaces (for Apache Cassandra). The CQL statements available to query, transform, and manage data are `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. The following topics outline some of the more complex options available when working with queries. For the complete language syntax with examples, see [the section called “DML statements”](#) (p. 204).

### Topics

- [Ordering results in Amazon Keyspaces](#) (p. 93)
- [Paginating results in Amazon Keyspaces](#) (p. 93)

## Ordering results in Amazon Keyspaces

The `ORDER BY` clause specifies the sort order of the results returned in a `SELECT` statement. The statement takes a list of column names as arguments and for each column you can specify the sort order for the data. You can only specify clustering columns in ordering clauses, non-clustering columns are not allowed.

The two available sort order options for the returned results are `ASC` for ascending and `DESC` for descending sort order. If you don't specify the sort order in the query statement, the default ordering of the clustering column is used.

The possible sort orders you can use in an ordering clause depend on the sort order assigned to each clustering column at table creation. Query results can only be sorted in the order defined for all clustering columns at table creation or the inverse of the defined sort order. Other possible combinations are not allowed.

For example, if the table's `CLUSTERING ORDER` is (col1 `ASC`, col2 `DESC`, col3 `ASC`), then the valid parameters for `ORDER BY` are either (col1 `ASC`, col2 `DESC`, col3 `ASC`) or (col1 `DESC`, col2 `ASC`, col3 `DESC`). For more information on `CLUSTERING ORDER`, see `table_options` under [the section called “CREATE TABLE”](#) (p. 200).

## Paginating results in Amazon Keyspaces

Amazon Keyspaces automatically *paginates* the results from `SELECT` statements when the data read to process the `SELECT` statement exceeds 1 MB. With pagination, the `SELECT` statement results are divided into “pages” of data that are 1 MB in size (or less). An application can process the first page of results, then the second page, and so on. Clients should always check for pagination tokens when executing `SELECT` queries that return multiple rows.

If a client supplies a `PAGE SIZE` that requires reading more than 1 MB of data, Amazon Keyspaces breaks up the results automatically into multiple pages based on the 1 MB data-read increments.

For example, if the average size of a row is 100 KB and you specify a PAGE SIZE of 20, Amazon Keyspaces paginates data automatically after it reads 10 rows (1000 KB of data read).

Because Amazon Keyspaces paginates results based on the number of rows read to process a request and not the number of rows returned in the result set, some pages may not contain any rows if you are running filtered queries.

For example, if you set PAGE SIZE to 10 and Keyspaces evaluates 30 rows to process your `SELECT` query, Amazon Keyspaces will return three pages. If only a subset of the rows matched your query, some pages may have less than 10 rows.

# Data modeling in Amazon Keyspaces (for Apache Cassandra)

This topic introduces data modeling concepts in Amazon Keyspaces (for Apache Cassandra). Use this section to find recommendations for designing data models that align with your application's data access patterns. Implementing data modeling best practices improves performance and minimizes throughput costs when working with Amazon Keyspaces.

To visualize and design data models more easily, you can use the [NoSQL Workbench \(p. 97\)](#).

## Topics

- [How to use partition keys effectively in Amazon Keyspaces \(p. 95\)](#)

## How to use partition keys effectively in Amazon Keyspaces

The primary key that uniquely identifies each row in an Amazon Keyspaces table can consist of one or multiple partition key columns, which determine which partitions the data is stored in, and one or more optional clustering column, which define how data is clustered and sorted within a partition.

Because the partition key establishes the number of partitions your data is stored in and how the data is distributed across these partitions, how you chose your partition key can have a significant impact upon the performance of your queries. In general, you should design your application for uniform activity across all partitions on disk.

Distributing read and write activity of your application evenly across all partitions helps to minimize throughput costs and this applies to on-demand as well as provisioned read/write capacity modes. For example, if you are using provisioned capacity mode, you can determine the access patterns that your application needs, and estimate the total read capacity units (RCU) and write capacity units (WCU) that each table requires. Amazon Keyspaces supports your access patterns using the throughput that you provisioned as long as the traffic against a given partition does not exceed 3,000 RCUs or 1,000 WCUs.

## Topics

- [Using write sharding to distribute workloads evenly in Amazon Keyspaces \(p. 95\)](#)

## Using write sharding to distribute workloads evenly in Amazon Keyspaces

One way to better distribute writes across a partition in Amazon Keyspaces is to expand the space. You can do this in several different ways. You can add an additional partition key column to which you write random numbers to distribute the rows among partitions. Or you can use a number that is calculated based on something that you're querying on.

## Sharding using compound partition keys and random values

One strategy for distributing loads more evenly across a partition is to add an additional partition key column to which you write random numbers. Then you randomize the writes across the larger space.

For example, consider the following table which has a single partition key representing a date.

```
CREATE TABLE IF NOT EXISTS tracker.blogs (  
    publish_date date,  
    title string,  
    description int,  
    PRIMARY KEY (publish_date));
```

To more evenly distribute this table across partitions, you could include an additional partition key column `shard` that stores random numbers. For example:

```
CREATE TABLE IF NOT EXISTS tracker.blogs (  
    publish_date date,  
    shard int,  
    title string,  
    description int,  
    PRIMARY KEY (publish_date, shard));
```

When inserting data you might choose a random number between 1 and 200 for the `shard` column. This yields compound partition key values like `(2020-07-09, 1)`, `(2020-07-09, 2)`, and so on, through `(2020-07-09, 200)`. Because you are randomizing the partition key, the writes to the table on each day are spread evenly across multiple partitions. This results in better parallelism and higher overall throughput.

However, to read all the rows for a given day, you would have to query the rows for all the shards and then merge the results. For example, you would first issue a `SELECT` statement for the partition key value `(2020-07-09, 1)`. Then issue another `SELECT` statement for `(2020-07-09, 2)`, and so on, through `(2020-07-09, 200)`. Finally, your application would have to merge the results from all those `SELECT` statements.

## Sharding using compound partition keys and calculated values

A randomizing strategy can greatly improve write throughput. But it's difficult to read a specific row because you don't know which value was written to the `shard` column when the row was written. To make it easier to read individual rows, you can use a different strategy. Instead of using a random number to distribute the rows among partitions, use a number that you can calculate based upon something that you want to query on.

Consider the previous example, in which a table uses today's date in the partition key. Now suppose that each row has an accessible `title` column, and that you most often need to find rows by title in addition to date. Before your application writes the row to the table, it could calculate a hash value based on the title and use it to populate the `shard` column. The calculation might generate a number between 1 and 200 that is fairly evenly distributed, similar to what the random strategy produces.

A simple calculation would likely suffice, such as the product of the UTF-8 code point values for the characters in the title, modulo 200, + 1. The compound partition key value would then be the combination of the date and calculation result.

With this strategy, the writes are spread evenly across the partition key values, and thus across the physical partitions. You can easily perform a `SELECT` statement for a particular row and date because you can calculate the partition key value for a specific `title` value.

To read all the rows for a given day, you still must `SELECT` each of the `(2020-07-09, N)` keys (where `N` is 1–200), and your application then has to merge all the results. The benefit is that you avoid having a single "hot" partition key value taking all of the workload.

# Using NoSQL Workbench with Amazon Keyspaces (for Apache Cassandra)

NoSQL Workbench is a client-side application that helps you design and visualize nonrelational data models for Amazon Keyspaces more easily. NoSQL Workbench clients are available for Windows, macOS, and Linux.

## Designing data models and creating resources automatically

NoSQL Workbench provides you a point-and-click interface to design and create Amazon Keyspaces data models. You can easily create new data models from scratch by defining keyspaces, tables, and columns. You can also import existing data models and make modifications (such as adding, editing, or removing columns) to adapt the data models for new applications. NoSQL Workbench then enables you to commit the data models to Amazon Keyspaces or Apache Cassandra, and create the keyspaces and tables automatically. To learn how to build data models, see [the section called “Data modeler” \(p. 99\)](#).

## Visualizing data models

Using NoSQL Workbench, you can visualize your data models to help ensure that the data models can support your application's queries and access patterns. You can also save and export your data models in a variety of formats for collaboration, documentation, and presentations. For more information, see [the section called “Data visualizer” \(p. 102\)](#).

## Topics

- [Download NoSQL Workbench \(p. 97\)](#)
- [Getting started with NoSQL Workbench \(p. 98\)](#)
- [How to build data models \(p. 99\)](#)
- [How to visualize data models \(p. 102\)](#)
- [How to commit data models to Amazon Keyspaces and Apache Cassandra \(p. 105\)](#)
- [Sample data models in NoSQL Workbench \(p. 112\)](#)
- [Release history for NoSQL Workbench \(p. 113\)](#)

## Download NoSQL Workbench

Follow these instructions to download and install NoSQL Workbench.

### To download and install NoSQL Workbench

1. Use one of the following links to download NoSQL Workbench for free.

Operating System	Download Link	Checksum Link
macOS	<a href="#">Download for macOS</a>	<a href="#">Checksum</a>

## Amazon Keyspaces (for Apache Cassandra) Developer Guide

### Getting started

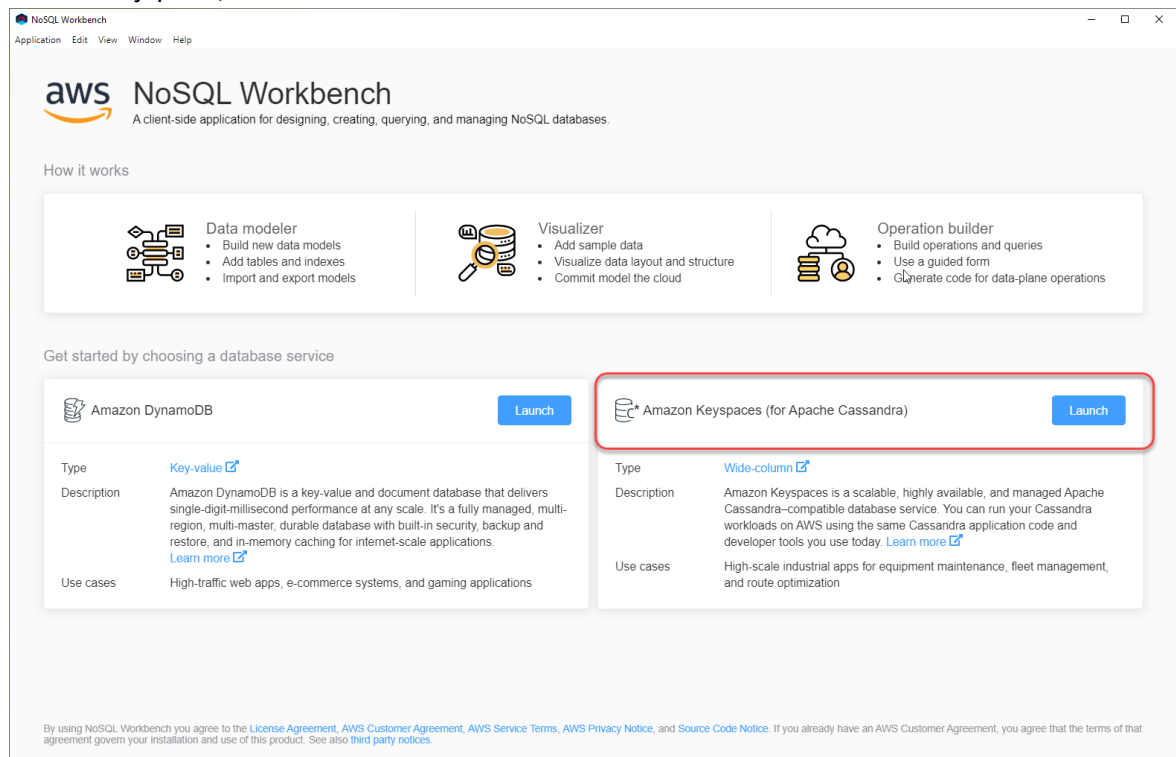
Operating System	Download Link	Checksum Link
Linux*	<a href="#">Download for Linux</a>	<a href="#">Checksum</a>
Windows	<a href="#">Download for Windows</a>	<a href="#">Checksum</a>

\* NoSQL Workbench supports Ubuntu 12.04, Fedora 21, and Debian 8 or any newer versions of these Linux distributions.

2. After the download completes, start the application and follow the onscreen instructions to complete the installation.

## Getting started with NoSQL Workbench

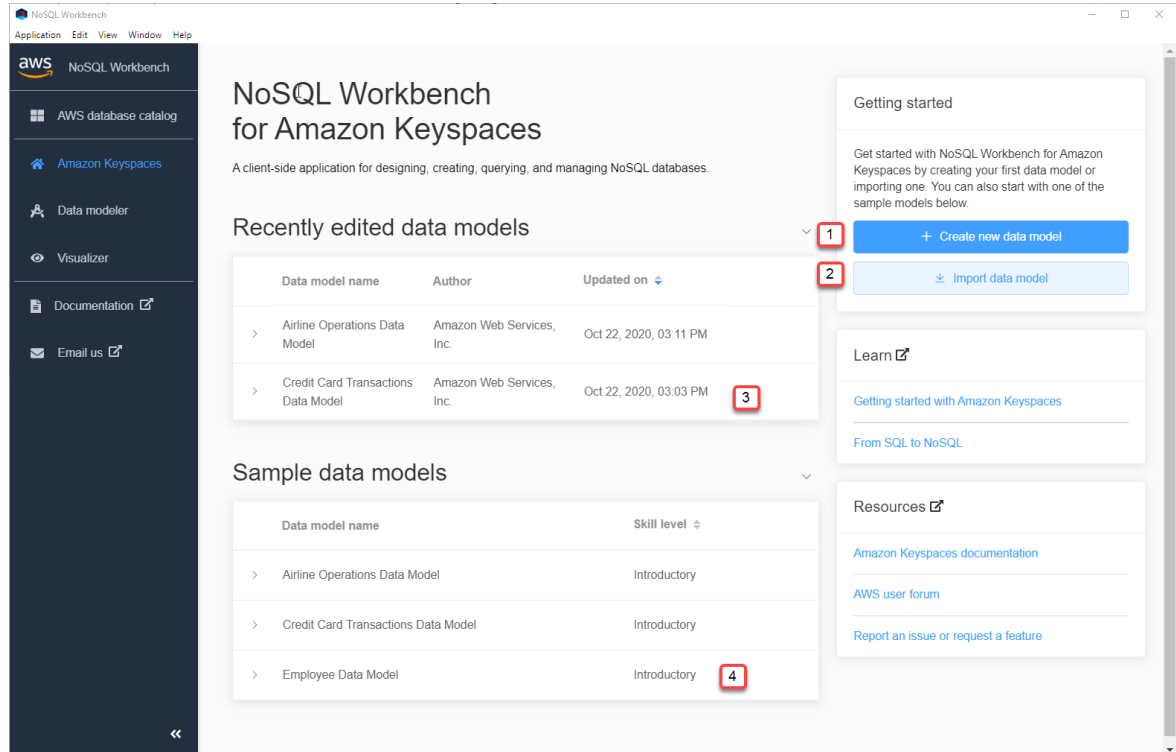
To get started with NoSQL Workbench, on the Database Catalog page in NoSQL Workbench, choose Amazon Keyspaces, and then choose **Launch**.



This opens the NoSQL Workbench home page for Amazon Keyspaces where you have the following options to get started:

1. Create a new data model.
2. Import an existing data model in JSON format.
3. Open a recently edited data model.
4. Open one of the available sample models.





Each of the options opens the NoSQL Workbench data modeler. To continue creating a new data model, see [the section called “Creating a data model” \(p. 99\)](#). To edit an existing data model, see [the section called “Editing a data model” \(p. 101\)](#).

## How to build data models

You can use the NoSQL Workbench data modeler to design new data models based on your application's data access patterns. You can use the data modeler to design new data models or import and modify existing data models created using NoSQL Workbench. The data modeler also includes a few sample data models to help you get started with data modeling.

### Topics

- [Building new data models with NoSQL Workbench \(p. 99\)](#)
- [Editing existing data models with NoSQL Workbench \(p. 101\)](#)

## Building new data models with NoSQL Workbench

To create a new data model for Amazon Keyspaces, you can use the NoSQL Workbench data modeler to create keyspaces, tables, and columns. Follow these steps to create a new data model.

1. To create a new keyspace, choose the plus sign under **Keyspace**.

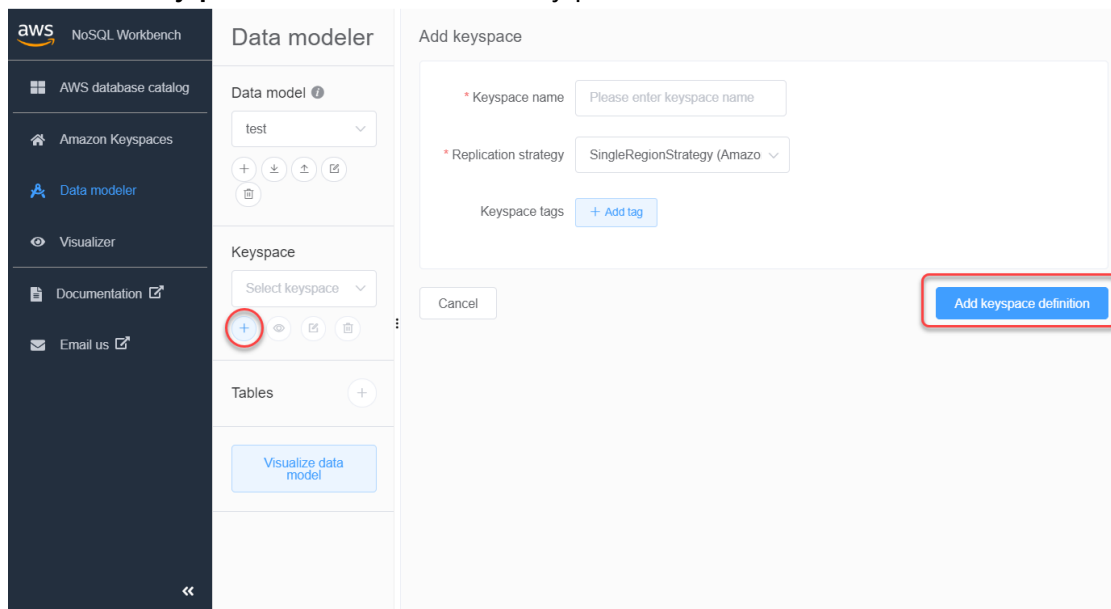
In this step, choose the following properties and settings.

- **Keyspace name** – Enter the name of the new keyspace.
- **Replication strategy** – Choose the replication strategy for the keyspace. Amazon Keyspaces uses the **SingleRegionStrategy** to replicate data three times automatically in multiple AWS Availability

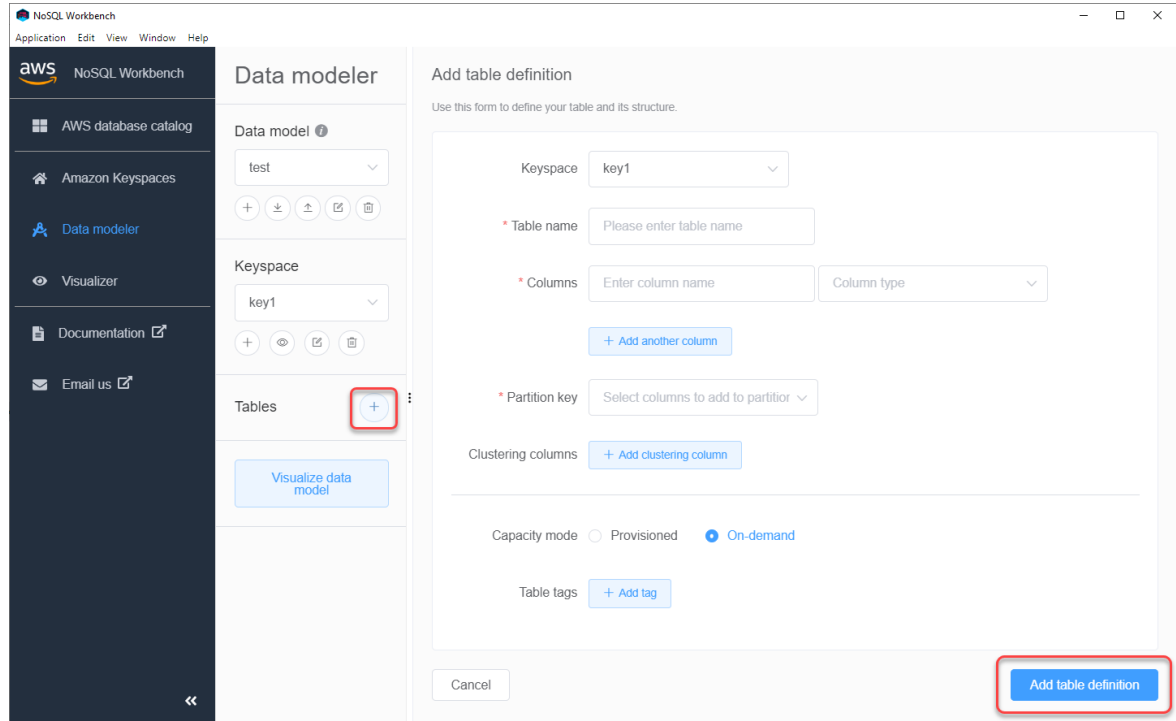
Zones. If you're planning to commit the data model to an Apache Cassandra cluster, you can choose **SimpleStrategy** or **NetworkTopologyStrategy**.

- **Keyspaces tags** – Resource tags are optional and let you categorize your resources in different ways—for example, by purpose, owner, environment, or other criteria. To learn more about tags for Amazon Keyspaces resources, see [Tagging resources \(p. 126\)](#).

2. Choose **Add keyspace definition** to create the keyspace.



3. To create a new table, choose the plus sign next to **Tables**. In this step, you define the following properties and settings.
  - **Table name** – The name of the new table.
  - **Columns** – Add a column name and choose the data type. Repeat these steps for every column in your schema.
  - **Partition key** – Choose columns for the partition key.
  - **Clustering columns** – Choose clustering columns (optional).
  - **Capacity mode** – Choose the read/write capacity mode for the table. You can choose provisioned or on-demand capacity. To learn more about capacity modes, see [the section called “Read/write capacity modes” \(p. 72\)](#).
  - **Table tags** – Resource tags are optional and let you categorize your resources in different ways—for example, by purpose, owner, environment, or other criteria. To learn more about tags for Amazon Keyspaces resources, see [Tagging resources \(p. 126\)](#).
4. Choose **Add table definition** to create the new table.
5. Repeat these steps to create additional tables.
6. Continue to [the section called “Visualizing a Data Model” \(p. 102\)](#) to visualize the data model that you created.



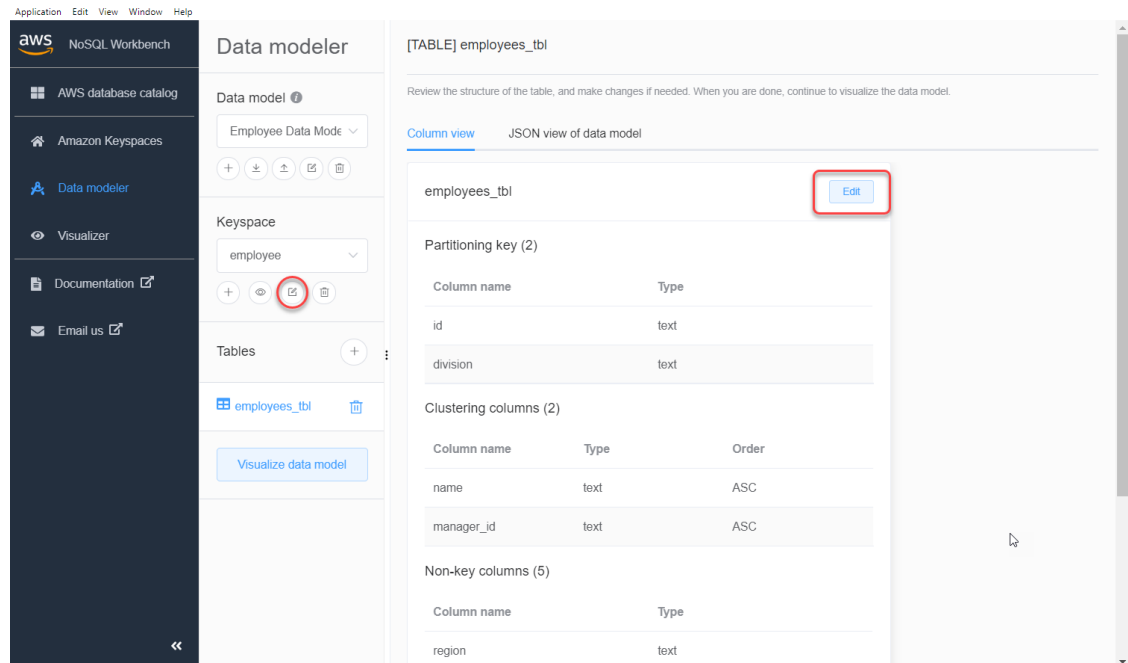
## Editing existing data models with NoSQL Workbench

With the NoSQL Workbench data modeler, you can edit existing data models in Amazon Keyspaces. These can be data models that are imported from a file, the provided sample data models, or data models that you created previously.

1. To edit a keyspace, choose the edit symbol under **Keyspace**.

In this step, you can edit the following properties and settings.

- **Keyspace name** – Enter the name of the new keyspace.
  - **Replication strategy** – Choose the replication strategy for the keyspace. Amazon Keyspaces uses the **SingleRegionStrategy** to replicate data three times automatically in multiple AWS Availability Zones. If you're planning to commit the data model to an Apache Cassandra cluster, you can choose **SimpleStrategy** or **NetworkTopologyStrategy**.
  - **Keyspaces tags** – Resource tags are optional and let you categorize your resources in different ways—for example, by purpose, owner, environment, or other criteria. To learn more about tags for Amazon Keyspaces resources, see [Tagging resources \(p. 126\)](#).
2. Choose **Save edits** to update the keyspace.



3. To edit a table, choose **Edit** next to the table name. In this step, you can update the following properties and settings.
  - **Table name** – The name of the new table.
  - **Columns** – Add a column name and choose the data type. Repeat these steps for every column in your schema.
  - **Partition key** – Choose columns for the partition key.
  - **Clustering columns** – Choose clustering columns (optional).
  - **Capacity mode** – Choose the read/write capacity mode for the table. You can choose provisioned or on-demand capacity. To learn more about capacity modes, see [the section called “Read/write capacity modes”](#) (p. 72).
  - **Table tags** – Resource tags are optional and let you categorize your resources in different ways—for example, by purpose, owner, environment, or other criteria. To learn more about tags for Amazon Keyspaces resources, see [Tagging resources](#) (p. 126).
4. Choose **Save edits** to update the table.
5. Continue to [the section called “Visualizing a Data Model”](#) (p. 102) to visualize the data model that you updated.

## How to visualize data models

Using NoSQL Workbench, you can visualize your data models to help ensure that the data models can support your application’s queries and access patterns. You also can save and export your data models in a variety of formats for collaboration, documentation, and presentations.

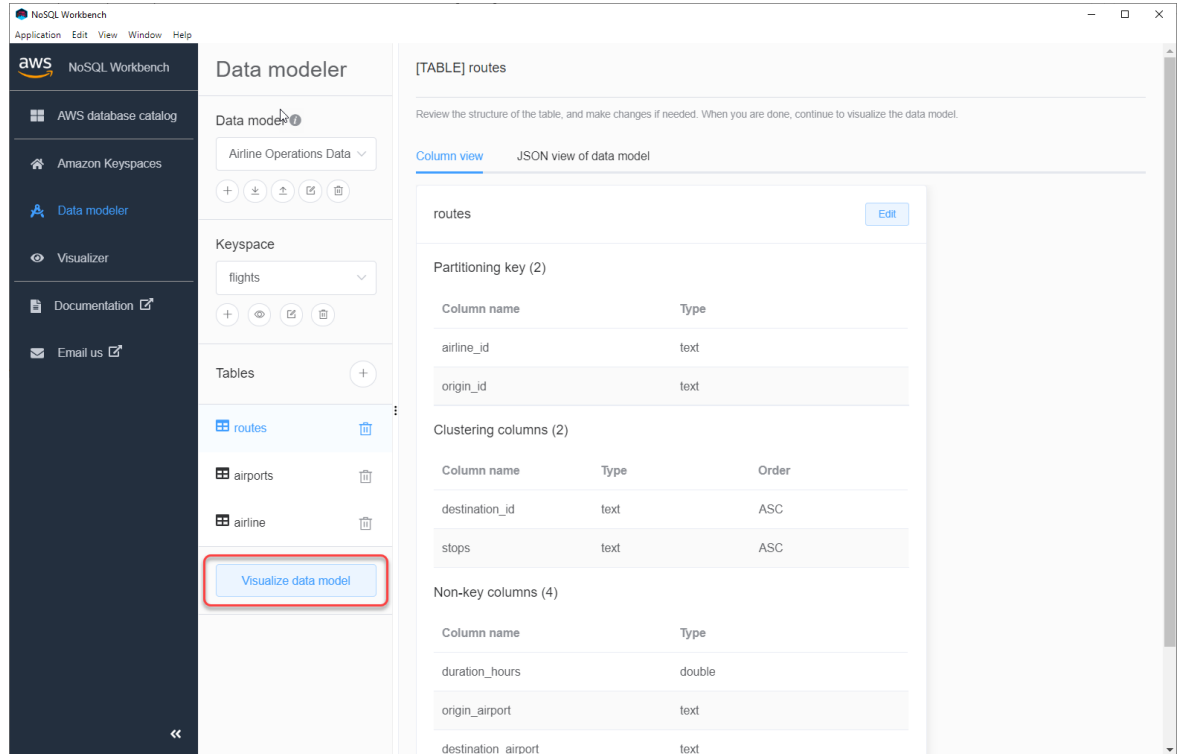
After you have created a new data model or edited an existing data model, you can visualize the model.

## Visualizing data models with NoSQL Workbench

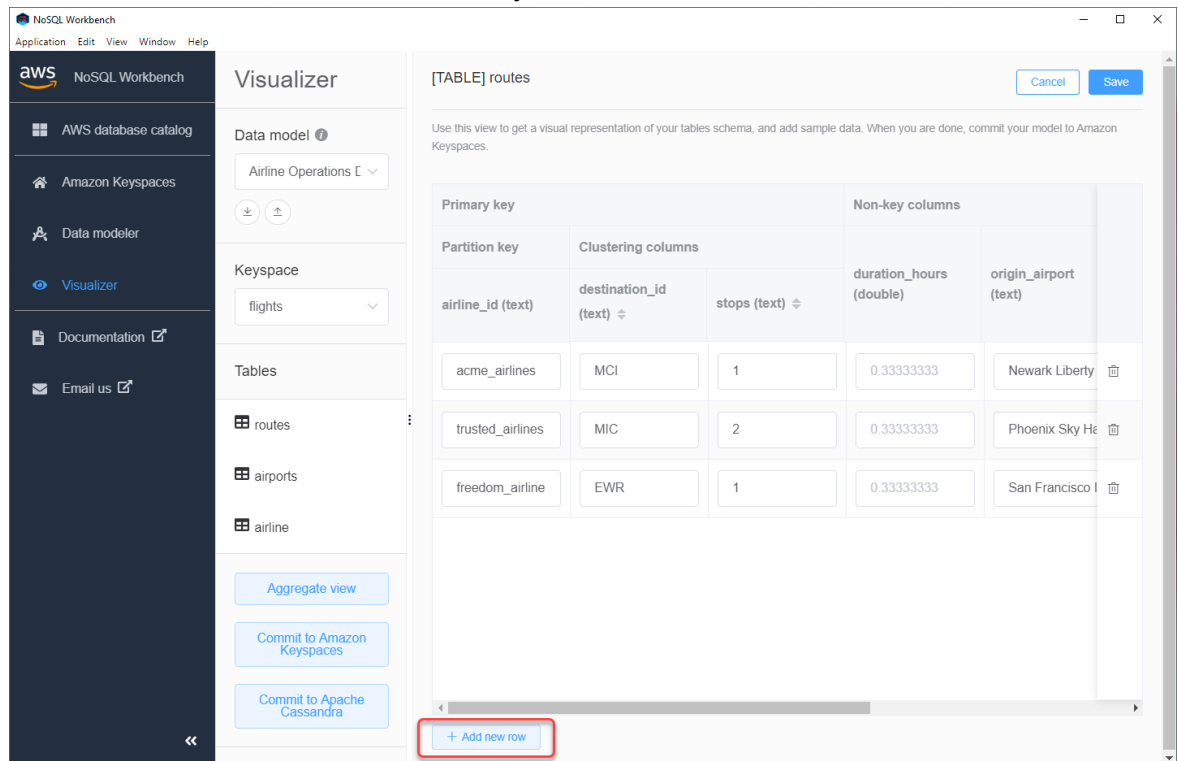
When you have completed the data model in the data modeler, choose **Visualize data model**.

## Amazon Keyspaces (for Apache Cassandra) Developer Guide

### Visualizing a Data Model



This takes you to the data visualizer in NoSQL Workbench. The data visualizer provides a visual representation of the table's schema and lets you add sample data. To add sample data to a table, choose a table from the model, and then choose **Edit**. To add a new row of data, choose **Add new row** at the bottom of the screen. Choose **Save** when you're done.



## Aggregate view

After you have confirmed the table's schema, you can aggregate data model visualizations.

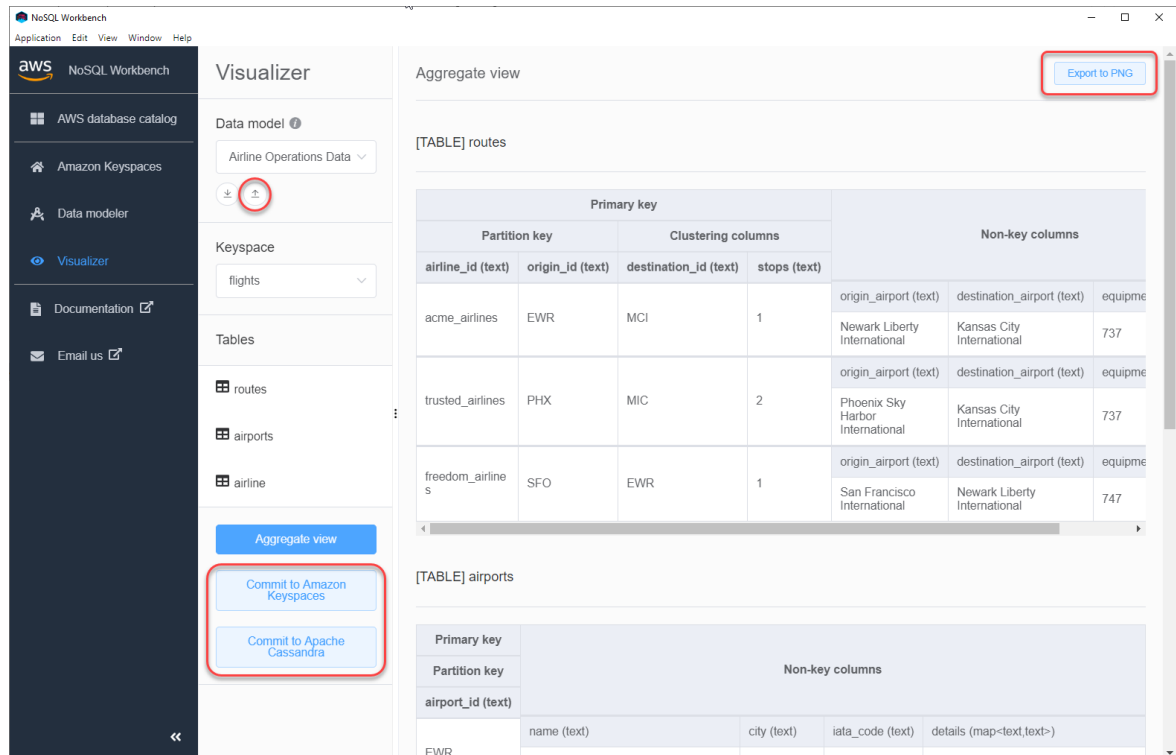
The screenshot shows the AWS NoSQL Workbench Visualizer interface. On the left is a dark sidebar with navigation links: AWS database catalog, Amazon Keyspaces, Data modeler, Visualizer (selected), Documentation, and Email us. The main panel is titled 'Visualizer' and contains a 'Data model' section with a dropdown menu set to 'Airline Operations Data'. Below this is a 'Keyspace' dropdown set to 'flights'. A 'Tables' list on the right includes 'routes', 'airports', and 'airline'. The 'Aggregate view' button is highlighted with a red rectangle. Below the table list are buttons for 'Commit to Amazon Keyspaces' and 'Commit to Apache Cassandra'. The right pane shows a table visualization for the 'routes' table, with a title '[TABLE] routes' and an 'Edit' button. Below the title is a descriptive text: 'Use this view to get a visual representation of your tables schema, and add sample data. When you are done, commit your model to Amazon Keyspaces.' The table has a 'Primary key' section with 'Partition key' (airline\_id (text), origin\_id (text)) and 'Clustering columns' (destination\_id (text), stops (text)). The 'Non-key columns' section includes origin\_airport (text), destination\_airport (text), and equipment. The table contains three rows of data: 'acme\_airlines' (EWR to MCI, 1 stop, equipment 737), 'trusted\_airlines' (PHX to MIC, 2 stops, equipment 737), and 'freedom\_airlines' (SFO to EWR, 1 stop, equipment 747).

Primary key				Non-key columns		
Partition key		Clustering columns				
airline_id (text)	origin_id (text)	destination_id (text)	stops (text)	origin_airport (text)	destination_airport (text)	equipment
acme_airlines	EWR	MCI	1	Newark Liberty International	Kansas City International	737
trusted_airlines	PHX	MIC	2	Phoenix Sky Harbor International	Kansas City International	737
freedom_airlines	SFO	EWR	1	San Francisco International	Newark Liberty International	747

After you have aggregated the view of the data model, you can export the view to a PNG file. To export the data model to a JSON file, choose the upload sign under the data model name.

### Note

You can export the data model in JSON format at any time in the design process.



You have the following options to commit the changes:

- Commit to Amazon Keyspaces
- Commit to an Apache Cassandra cluster

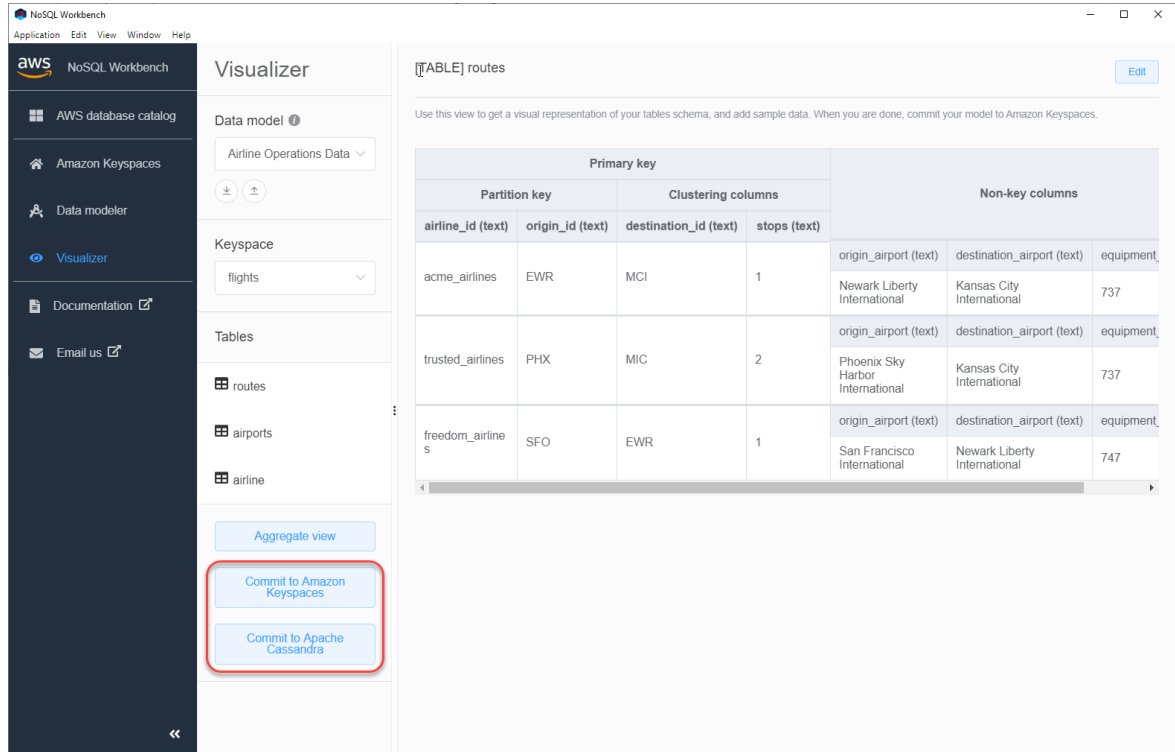
To learn more about how to commit changes, see [the section called “Committing a data model”](#) (p. 105).

## How to commit data models to Amazon Keyspaces and Apache Cassandra

This section shows you how to commit completed data models to Amazon Keyspaces and Apache Cassandra clusters. This process automatically creates the server-side resources for keyspaces and tables based on the settings that you defined in the data model.

## Amazon Keyspaces (for Apache Cassandra) Developer Guide

### Before you begin



### Topics

- [Before you begin \(p. 106\)](#)
- [Connecting to Amazon Keyspaces with service-specific credentials \(p. 107\)](#)
- [Connecting to Amazon Keyspaces with AWS Identity and Access Management \(IAM\) credentials \(p. 108\)](#)
- [Using a saved connection \(p. 110\)](#)
- [Committing to Apache Cassandra \(p. 111\)](#)

## Before you begin

Amazon Keyspaces requires the use of Transport Layer Security (TLS) to help secure connections with clients. To connect to Amazon Keyspaces using TLS, you need to complete the following task before you can start.

- Download the Starfield digital certificate using the following command and save `sf-class2-root.crt` locally or in your home directory.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

### Note

You can also use the Amazon digital certificate to connect to Amazon Keyspaces and can continue to do so if your client is connecting to Amazon Keyspaces successfully. The Starfield certificate provides additional backwards compatibility for clients using older certificate authorities.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```



After you have saved the certificate file, you can connect to Amazon Keyspaces. One option is to connect by using service-specific credentials. Service-specific credentials are a user name and password that are associated with a specific IAM user and can only be used with the specified service. The second option is to connect with IAM credentials that are using the [AWS Signature Version 4 process \(SigV4\)](#). To learn more about these two options, see [the section called “Creating credentials” \(p. 16\)](#).

To connect with service-specific credentials, see [the section called “Connecting with service-specific credentials” \(p. 107\)](#).

To connect with IAM credentials, see [the section called “Connecting with IAM credentials” \(p. 108\)](#).

## Connecting to Amazon Keyspaces with service-specific credentials

This section shows how to use service-specific credentials to commit the data model you created or edited with NoSQL Workbench.

1. To create a new connection using service-specific credentials, choose the **Connect by using user name and password** tab.
  - Before you begin, you must create service-specific credentials using the process documented at [the section called “Generate service-specific credentials” \(p. 16\)](#).


After you have obtained the service-specific credentials, you can continue to set up the connection. Continue with one of the following:

- **User name** – Enter the user name.
- **Password** – Enter the password.
- **AWS Region** – For available Regions, see [the section called “Service endpoints” \(p. 17\)](#).
- **Port** – Amazon Keyspaces uses port 9142.


Alternatively, you can import saved credentials from a file.

2. Choose **Commit** to update Amazon Keyspaces with the data model.

## Commit to Amazon Keyspaces

 On this page, you can create server-side resources such as keyspace and tables for the chosen data model.

< Use saved connections    Connect by using IAM credentials    Connect by using user name >

 You can generate service-specific credentials to allow your users to access Amazon Keyspaces using AWS Management Console or AWS CLI.

[How to generate Amazon Keyspaces credentials](#)

\* User Name

anika

\* Password

.....



\* AWS Region


us-east-1



\* Port

9142

OR

 Import from credential file

Cancel

Reset

Commit

## Connecting to Amazon Keyspaces with AWS Identity and Access Management (IAM) credentials

This section shows how to use IAM credentials to commit the data model created or edited with NoSQL Workbench.

1. To create a new connection using IAM credentials, choose the **Connect by using IAM credentials** tab.
  - Before you begin, you must create IAM credentials using one of the following methods.

- For console access, use your IAM user name and password to sign in to the [AWS Management Console](#) from the [IAM sign-in page](#). IAM lets you securely control access to AWS services and resources in your AWS account. For details about console and programmatic credentials, see [Understanding and getting your security credentials](#) in the *AWS General Reference*.
- For CLI access, you need an access key ID and secret access key. Use IAM user access keys instead of AWS account root user access keys. IAM lets you securely control access to AWS services and resources in your AWS account. For more information about creating access keys, see [Understanding and getting your security credentials](#) in the *AWS General Reference*.
- For API access, you need an access key ID and secret access key. Use IAM user access keys instead of AWS account root user access keys. For more information about creating access keys, see [Managing access keys for IAM users](#) in the *IAM User Guide*.

For more information, see [Managing access keys for IAM users](#).

After you have obtained the IAM credentials, you can continue to set up the connection.

- **Connection name** – The name of the connection.
  - **AWS Region** – For available Regions, see [the section called “Service endpoints” \(p. 17\)](#).
  - **Access key ID** – Enter the access key ID.
  - **Secret access key** – Enter the secret access key.
  - **Port** – Amazon Keyspaces uses port 9142.
  - **AWS public certificate** – Point to the AWS certificate that was downloaded in the first step.
  - **Persist connection** – Select this check box if you want to save the AWS connection secrets locally.
2. Choose **Commit** to update Amazon Keyspaces with the data model.


 On this page, you can create server-side resources such as keyspaces and tables for the chosen data model.

< Use saved connections **Connect by using IAM credentials** Connect by using user name >


\* Connection name

\* AWS Region

\* Access key ID

\* Secret access key  

\* Port

\* AWS public certificate  AmazonRootCA1.pem  
Choose an AWS public certificate for a Signature Version 4–based connection to Amazon Keyspaces. 

Persist connection ☒

If you select this check box, AWS connection secrets will be persisted in  
C:\Users\la...\.aws\credentials

Cancel

Reset

Commit

## Using a saved connection

If you have previously set up a connection to Amazon Keyspaces, you can use that as the default connection to commit data model changes. Choose the **Use saved connections** tab and continue to commit the updates.

## Commit to Amazon Keyspaces



 On this page, you can create server-side resources such as keyspaces and tables for the chosen data model.

< [Use saved connections](#)    Connect by using IAM credentials    Connect by using user name >

\* Saved connections

default




\* Port

9142

\* AWS public certificate

 [Choose AWS public certificate](#)

AmazonRootCA1.pem

Choose an AWS public certificate for a Signature Version 4–based connection to Amazon Keyspaces. 

Cancel

Reset

Commit

## Committing to Apache Cassandra

This section walks you through making the connections to an Apache Cassandra cluster to commit the data model created or edited with NoSQL Workbench.

### Note

Only data models that have been created with `SimpleStrategy` or `NetworkTopologyStrategy` can be committed to Apache Cassandra clusters. To change the replication strategy, edit the keyspace in the data modeler.

- **User name** – Enter the user name if authentication is enabled on the cluster.
  - **Password** – Enter the password if authentication is enabled on the cluster.
  - **Contact points** – Enter the contact points.
  - **Local data center** – Enter the name of the local data center.
  - **Port** – The connection uses port 9042.
2. Choose **Commit** to update the Apache Cassandra cluster with the data model.

### Commit to Apache Cassandra

**i** Configure the connection to the Apache Cassandra cluster so that you can commit your data model to the database, including keyspaces, tables, and sample data. The user name and password are not required, and are necessary only if authentication is enabled on the cluster

User name

User name

Password

Password

\* Contact points

Contact point

+ Add another contact point

\* Local data center

Data center

\* Port

9042

Cancel

Reset

Commit

## Sample data models in NoSQL Workbench

The home page for the modeler and visualizer displays a number of sample models that ship with NoSQL Workbench. This section describes these models and their potential uses.

### Topics

- [Employee data model \(p. 113\)](#)
- [Credit card transactions data model \(p. 113\)](#)
- [Airline operations data model \(p. 113\)](#)

## Employee data model

This data model represents an Amazon Keyspaces schema for an employee database application.

Applications that access employee information for a given company can use this data model.

The access patterns supported by this data model are:

- Retrieval of an employee record with a given ID.
- Retrieval of an employee record with a given ID and division.
- Retrieval of an employee record with a given ID and name.

## Credit card transactions data model

This data model represents an Amazon Keyspaces schema for credit card transactions at retail stores.

The storage of credit card transactions not only helps stores with bookkeeping, but also helps store managers analyze purchase trends, which can help them with forecasting and planning.

The access patterns supported by this data model are:

- Retrieval of transactions by credit card number, month and year, and date.
- Retrieval of transactions by credit card number, category, and date.
- Retrieval of transactions by category, location, and credit card number.
- Retrieval of transactions by credit card number and dispute status.

## Airline operations data model

This data model shows data about plane flights, including airports, airlines, and flight routes.

Key components of Amazon Keyspaces modeling that are demonstrated are key-value pairs, wide-column data stores, composite keys, and complex data types such as maps to demonstrate common NoSQL data-access patterns.

The access patterns supported by this data model are:

- Retrieval of routes originating from a given airline at a given airport.
- Retrieval of routes with a given destination airport.
- Retrieval of airports with direct flights.
- Retrieval of airport details and airline details.

## Release history for NoSQL Workbench

The following table describes the important changes in each release of the *NoSQL Workbench* client-side application.

Change	Description	Date
NoSQL Workbench for Amazon Keyspaces – GA.	NoSQL Workbench for Amazon Keyspaces is generally available.	October 28, 2020

Amazon Keyspaces (for Apache  
Cassandra) Developer Guide  
Release history

---

Change	Description	Date
NoSQL Workbench preview released.	NoSQL Workbench is a client-side application that helps you design and visualize nonrelational data models for Amazon Keyspaces more easily. NoSQL Workbench clients are available for Windows, macOS, and Linux. For more information, see <a href="#">NoSQL Workbench for Amazon Keyspaces</a> .	October 5, 2020



# Point-in-time recovery for Amazon Keyspaces (for Apache Cassandra)

Point-in-time recovery (PITR) helps protect your Amazon Keyspaces tables from accidental write or delete operations by providing you continuous backups of your table data.

For example, suppose that a test script writes accidentally to a production Amazon Keyspaces table. With point-in-time recovery, you can restore that table's data to any second in time since PITR was enabled within the last 35 days. If you delete a table with point-in-time recovery enabled, you can query for the deleted table's data for 35 days (at no additional cost), and restore it to the state it was in just before the point of deletion.

You can restore an Amazon Keyspaces table to a point in time by using the console or Cassandra Query Language (CQL). For more information, see [Restoring an Amazon Keyspaces table to a point in time \(p. 121\)](#).

Point-in-time operations have no performance or availability impact on the base table, and restoring a table doesn't consume additional throughput.

For information about PITR quotas, see [Quotas \(p. 209\)](#).

For information about pricing, see [Amazon Keyspaces \(for Apache Cassandra\) pricing](#).

## Topics

- [How point-in-time recovery works in Amazon Keyspaces \(p. 115\)](#)
- [Restoring an Amazon Keyspaces table to a point in time \(p. 121\)](#)

## How point-in-time recovery works in Amazon Keyspaces

This section provides an overview of how Amazon Keyspaces point-in-time recovery (PITR) works. For more information about pricing, see [Amazon Keyspaces \(for Apache Cassandra\) pricing](#).

## Topics

- [Enabling point-in-time recovery \(PITR\) \(p. 115\)](#)
- [Permissions required to restore a table \(p. 117\)](#)
- [Time window for PITR continuous backups \(p. 118\)](#)
- [PITR restore settings \(p. 119\)](#)
- [PITR restore of encrypted tables \(p. 119\)](#)
- [Table restore time with PITR \(p. 120\)](#)
- [Amazon Keyspaces PITR and integration with AWS services \(p. 120\)](#)

## Enabling point-in-time recovery (PITR)

You can enable PITR by using the console, or you can enable it programmatically.

## Enabling PITR with the console

PITR settings for new tables can be managed under the **Customized settings** option. By default, PITR is enabled on new tables created through the console.

To enable PITR for an existing table, complete the following steps.

1. Sign in to the AWS Management Console, and open the Amazon Keyspaces console at <https://console.aws.amazon.com/keyspaces/home>.
2. In the navigation pane, choose **Tables** and select the table you want to edit.
3. On the **Backups** tab, choose **Edit**.
4. In the **Edit point-in-time recovery settings** section, select **Enable Point-in-time recovery**.

You can disable PITR on a table at any time with the following steps.

1. Sign in to the AWS Management Console, and open the Amazon Keyspaces console at <https://console.aws.amazon.com/keyspaces/home>.
2. In the navigation pane, choose **Tables** and select the table you want to edit.
3. On the **Backups** tab, choose **Edit**.
4. In the **Edit point-in-time recovery settings** section, clear the **Enable Point-in-time recovery** check box.

### Important

Disabling PITR deletes your backup history immediately, even if you reenables PITR on the table within 35 days.

To learn how to restore a table using the console, see [the section called "Restoring a table to a point in time \(console\)" \(p. 121\)](#).

## Enabling PITR programmatically

You can manage PITR settings for tables by using the `point_in_time_recovery` custom property.

When creating a new table using CQL, you must explicitly enable PITR when you create the new table.

To enable PITR when you're creating a new table, you can use the following CQL command as an example.

```
CREATE TABLE "my_keyspace1"."my_table1"(  
  "id" int,  
  "name" ascii,  
  "date" timestamp,  
  PRIMARY KEY("id"))  
WITH CUSTOM_PROPERTIES = {  
  'capacity_mode':{'throughput_mode':'PAY_PER_REQUEST'},  
  'point_in_time_recovery':{'status':'enabled'}  
}
```

### Note

If no point-in-time recovery custom property is specified, point-in-time recovery is disabled by default.

To enable PITR for an existing table using CQL, run the following CQL command.

```
ALTER TABLE mykeyspace.mytable
```

```
WITH custom_properties = {'point_in_time_recovery': {'status': 'enabled'}}
```

To disable PITR on an existing table, run the following CQL command.

```
ALTER TABLE mykeyspace.mytable  
WITH custom_properties = {'point_in_time_recovery': {'status': 'disabled'}}
```

### Important

Disabling PITR deletes your backup history immediately, even if you reenables PITR on the table within 35 days.

For more information in the CQL Language Reference, see [the section called “CREATE TABLE” \(p. 200\)](#) and [the section called “ALTER TABLE” \(p. 202\)](#). To learn how to restore a table using CQL, see [the section called “Restoring a table to a point in time with CQL” \(p. 122\)](#).

## Permissions required to restore a table

To successfully restore a table, the IAM user or role needs the following minimum permissions:

- `cassandra:Restore` – The restore action is required for the target table to be restored.
- `cassandra:Select` – The select action is required to read from the source table.
- `cassandra:TagResource` – The tag action is optional, and only required if the restore operation adds tags.

The following is an example of a policy that grants minimum required permissions to a user to restore tables in keyspace `mykeyspace`.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "cassandra:Restore",  
        "cassandra:Select"  
      ],  
      "Resource": [  
        "arn:aws:cassandra:us-east-1:111122223333:keyspace/mykeyspace/*",  
        "arn:aws:cassandra:us-east-1:111122223333:keyspace/system*"  
      ]  
    }  
  ]  
}
```

Additional permissions to restore a table might be required based on other selected features. For example, if the source table is encrypted at rest with a customer managed key, Amazon Keyspaces must have permissions to access the customer managed key of the source table to successfully restore the table. For more information, see [the section called “PITR and encrypted tables” \(p. 119\)](#).

If you are using IAM policies with [condition keys](#) to restrict incoming traffic to specific sources, you must ensure that Amazon Keyspaces has permission to perform a restore operation on your principal's behalf. You must add an `aws:ViaAWSService` condition key to your IAM policy if your policy restricts incoming traffic to any of the following:

- VPC endpoints with `aws:SourceVpce`
- IP ranges with `aws:SourceIp`
- VPCs with `aws:SourceVpc`

The `aws:ViaAWSService` condition key allows access when any AWS service makes a request using the principal's credentials. For more information, see [IAM JSON policy elements: Condition key](#) in the *IAM User Guide*.

The following is an example of a policy that restricts source traffic to a specific IP address and allows Amazon Keyspaces to restore a table on the principal's behalf.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CassandraAccessForCustomIp",
      "Effect": "Allow",
      "Action": "cassandra:*",
      "Resource": "*",
      "Condition": {
        "Bool": {
          "aws:ViaAWSService": "false"
        },
        "ForAnyValue:IpAddress": {
          "aws:SourceIp": [
            "123.45.167.89"
          ]
        }
      }
    },
    {
      "Sid": "CassandraAccessForAwsService",
      "Effect": "Allow",
      "Action": "cassandra:*",
      "Resource": "*",
      "Condition": {
        "Bool": {
          "aws:ViaAWSService": "true"
        }
      }
    }
  ]
}
```

For an example policy using the `aws:ViaAWSService` global condition key, see [the section called “VPC Endpoint policies and Amazon Keyspaces point-in-time recovery \(PITR\)”](#) (p. 189).

## Time window for PITR continuous backups

Amazon Keyspaces PITR uses two timestamps to maintain the time frame for which restorable backups are available for a table.

- **Earliest restorable time** – Marks the time of the earliest restorable backup. The earliest restorable backup goes back up to 35 days or when PITR was enabled, whichever is more recent. The maximum backup window of 35 days can't be modified.
- **Current time** – The timestamp for the latest restorable backup is the current time. If no timestamp is provided during a restore, current time is used.

When PITR is enabled, you can restore to any point in time between `EarliestRestorableDateTime` and `CurrentTime`. You can only restore table data to a time when PITR was enabled.

If you disable PITR and later reenabling it again, you reset the start time for the first available backup to when PITR was reenabled. This means that disabling PITR erases your backup history.

**Note**

Data definition language (DDL) operations on tables, such as schema changes, are performed asynchronously. You can only see completed operations in your restored table data, but you might see additional actions on your source table if they were in progress at the time of the restore. For a list of DDL statements, see [the section called “DDL statements” \(p. 197\)](#).

A table doesn't have to be active to be restored. You can also restore deleted tables if PITR was enabled on the deleted table and the deletion occurred within the backup window (or within the last 35 days).

**Note**

If a new table is created with the same qualified name (for example, `mykeyspace.mytable`) as a previously deleted table, the deleted table will no longer be restorable. If you attempt to do this from the console, a warning is displayed.

## PITR restore settings

When you restore a table using PITR, Amazon Keyspaces restores your source table's schema and data to the state based on the selected timestamp (`day:hour:minute:second`) to a new table. PITR doesn't overwrite existing tables.

In addition to the table's schema and data, PITR restores the `custom_properties` from the source table. Unlike the table's data, which is restored based on the selected timestamp between earliest restore time and current time, custom properties are always restored based on the table's settings as of the current time.

The settings of the restored table match the settings of the source table with the timestamp of when the restore was initiated. If you want to overwrite these settings during restore, you can do so using `WITH custom_properties`. Custom properties include the following settings.

- Read/write capacity mode
- Provisioned throughput capacity settings
- PITR settings

When you do a full table restore, all table settings for the restored table come from the current settings of the source table at the time of the restore.

For example, suppose that a table's provisioned throughput was recently lowered to 50 read capacity units and 50 write capacity units. You then restore the table's state to three weeks ago. At this time, its provisioned throughput was set to 100 read capacity units and 100 write capacity units. In this case, Amazon Keyspaces restores your table data to that point in time, but uses the current provisioned throughput settings (50 read capacity units and 50 write capacity units).

The following settings are not restored, and you must configure them manually for the new table.

- Automatic scaling policies (for tables that use provisioned capacity mode)
- AWS Identity and Access Management (IAM) policies
- Amazon CloudWatch metrics and alarms
- Tags (can be added to the CQL `RESTORE` statement using `WITH TAGS`)

## PITR restore of encrypted tables

When you restore a table using PITR, Amazon Keyspaces restores your source table's encryption settings. If the table was encrypted with an AWS owned key (default), the table is restored with the same setting

automatically. If the table you want to restore was encrypted using a customer managed key, the same customer managed key needs to be accessible to Amazon Keyspaces to restore the table data.

You can change the encryption settings of the table at the time of restore. To change from an AWS owned key to a customer managed key, you need to supply a valid and accessible customer managed key at the time of restore.

If you want to change from a customer managed key to an AWS owned key, confirm that Amazon Keyspaces has access to the customer managed key of the source table to restore the table with an AWS owned key. For more information about encryption at rest settings for tables, see [the section called "How it works" \(p. 133\)](#).

**Note**

If the table was deleted because Amazon Keyspaces lost access to your customer managed key, you need to ensure the customer managed key is accessible to Amazon Keyspaces before trying to restore the table. A table that was encrypted with a customer managed key can't be restored if Amazon Keyspaces doesn't have access to that key. For more information, see [Troubleshooting key access](#) in the AWS Key Management Service Developer Guide.

## Table restore time with PITR

The time it takes you to restore a table is based on multiple factors and isn't always correlated directly to the size of the table.

The following are some considerations for restore times.

- You restore backups to a new table. It can take up to 20 minutes (even if the table is empty) to perform all the actions to create the new table and initiate the restore process.
- Restore times for large tables with well-distributed data models can be several hours or longer.
- If your source table contains data that is significantly skewed, the time to restore might increase. For example, if your table's primary key is using the month of the year as a partition key, and all your data is from the month of December, you have skewed data.

A best practice when planning for disaster recovery is to regularly document average restore completion times and establish how these times affect your overall Recovery Time Objective.

## Amazon Keyspaces PITR and integration with AWS services

The following PITR operations are logged using AWS CloudTrail to enable continuous monitoring and auditing.

- Create a new table with PITR enabled or disabled.
- Enable or disable PITR on an existing table.
- Restore an active or a deleted table.

For more information, see [Logging Amazon Keyspaces API calls with AWS CloudTrail \(p. 180\)](#).

You can perform the following PITR actions using AWS CloudFormation.

- Create a new table with PITR enabled or disabled.
- Enable or disable PITR on an existing table.

For more information, see the [Cassandra Resource Type Reference](#) in the [AWS CloudFormation User Guide](#).

## Restoring an Amazon Keyspaces table to a point in time

Amazon Keyspaces (for Apache Cassandra) point-in-time recovery (PITR) provides continuous backups of your Amazon Keyspaces table data. This tutorial shows you how to restore a table to a point in time by using the Amazon Keyspaces console or Cassandra Query Language (CQL), and how to restore a deleted table using CQL.

### Topics

- [Before you begin](#) (p. 121)
- [Restoring a table to a point in time \(console\)](#) (p. 121)
- [Restoring a table to a point in time with CQL](#) (p. 122)
- [Restoring a deleted table with CQL](#) (p. 123)

## Before you begin

If you haven't already done so, you must configure the appropriate permissions for the user to restore Amazon Keyspaces tables. In AWS Identity and Access Management (IAM), the AWS managed policy `AmazonKeyspacesFullAccess` includes the permissions to restore Amazon Keyspaces tables. For detailed steps to implement a policy with minimum required permissions, see [the section called "Restore Permissions"](#) (p. 117).

## Restoring a table to a point in time (console)

The following example demonstrates how to use the Amazon Keyspaces console to restore an existing table named `mytable` to a point in time.

### Note

This procedure assumes that you have enabled point-in-time recovery. To enable PITR for the `mytable` table, follow the steps in [the section called "Using the console"](#) (p. 116).

1. Sign in to the AWS Management Console, and open the Amazon Keyspaces console at <https://console.aws.amazon.com/keyspaces/home>.
2. In the navigation pane on the left side of the console, choose **Tables**.
3. In the list of tables, choose the `mytable` table.
4. On the **Backups** tab of the `mytable` table, in the **Point-in-time recovery** section, choose **Restore**.
5. For the new table name, enter `mytable_restored`.
6. To define the point in time for the restore operation, you can choose between two options:
  - Select the preconfigured **Earliest** time.
  - Select **Specify date and time** and enter the date and time you want to restore the new table to.

### Note

You can restore to any point in time within **Earliest** time and the current time. Amazon Keyspaces restores your table data to the state based on the selected date and time (day:hour:minute:second).

7. Choose **Restore** to start the restore process.

The table that is being restored is shown with the status **Restoring**. After the restore process is finished, the status of the `mytable_restored` table changes to **Active**.

**Important**

While a restore is in progress, don't modify or delete the AWS Identity and Access Management (IAM) policies that grant the IAM entity (for example, user, group, or role) permission to perform the restore. Otherwise, unexpected behavior can result. For example, suppose that you removed write permissions for a table while that table was being restored. In this case, the underlying `RestoreTableToPointInTime` operation can't write any of the restored data to the table.

You can modify or delete permissions only after the restore operation is completed.

## Restoring a table to a point in time with CQL

The following procedure shows how to use CQL to restore an existing table named `mytable` to a point in time.

**Note**

This procedure assumes that you have enabled point-in-time recovery. To enable PITR on the table, follow the steps in [the section called "Programmatically" \(p. 116\)](#).

1. You can restore an active table to a point-in-time between `earliest_restorable_timestamp` and the current time. Current time is the default.

To confirm that point-in-time recovery is enabled for the `mytable` table, query the `system_schema_mcs.tables` as follows.

```
SELECT custom_properties
FROM system_schema_mcs.tables
WHERE keyspace_name = 'mykeyspace' AND table_name = 'mytable';
```

Point-in-time recovery is enabled as shown in the following sample output.

```
custom_properties
-----
{
  ...,
  "point_in_time_recovery": {
    "earliest_restorable_timestamp": "2020-06-30T19:19:21.175Z"
    "status": "enabled"
  }
}
```

2. Restore the table to a point in time, specified by a `restore_timestamp` in ISO 8601 format. In this case, the `mytable` table is restored to the current time. You can omit the `WITH restore_timestamp = ...` clause. Without the clause, the current timestamp is used.

```
RESTORE TABLE mykeyspace.mytable_restored
FROM TABLE mykeyspace.mytable;
```

You can also restore to a specific point in time. You can specify any point in time during the last 35 days. For example, the following command restores the table to the `EarliestRestorableDateTime`.

```
RESTORE TABLE mykeyspace.mytable_restored
FROM TABLE mykeyspace.mytable
```



```
WITH restore_timestamp = '2020-06-30T19:19:21.175Z';
```

For a full syntax description, see [the section called “RESTORE TABLE” \(p. 203\)](#) in the language reference.

To verify that the restore of the table was successful, query the `system_schema_mcs.tables` to confirm the status of the table.

```
SELECT status
FROM system_schema_mcs.tables
WHERE keyspace_name = 'mykeyspace' AND table_name = 'mytable_restored'
```

The query shows the following output.

```
status
-----
RESTORING
```

The table that is being restored is shown with the status **Restoring**. After the restore process is finished, the status of the `mytable_restored` table changes to **Active**.

### Important

While a restore is in progress, don't modify or delete the AWS Identity and Access Management (IAM) policies that grant the IAM entity (for example, user, group, or role) permission to perform the restore. Otherwise, unexpected behavior can result. For example, suppose that you removed write permissions for a table while that table was being restored. In this case, the underlying `RestoreTableToPointInTime` operation can't write any of the restored data to the table. You can modify or delete permissions only after the restore operation is completed.

## Restoring a deleted table with CQL

The following procedure shows how to use CQL to restore a deleted table named `mytable` to the time of deletion.

### Note

This procedure assumes that PITR was enabled on the deleted table.

1. To confirm that point-in-time recovery is enabled for a deleted table, query the system table. Only tables with point-in-time recovery enabled are shown.

```
SELECT custom_properties
FROM system_schema_mcs.tables_history
WHERE keyspace_name = 'mykeyspace' AND table_name = 'my_table';
```

The query shows the following output.

```
custom_properties
-----
{
  ...,
  "point_in_time_recovery":{
    "restorable_until_time":"2020-08-04T00:48:58.381Z",
    "status":"enabled"
  }
}
```

2. Restore the table to the time of deletion with the following sample statement.

```
RESTORE TABLE mykeyspace.mytable_restored  
FROM TABLE mykeyspace.mytable;
```

# Creating Amazon Keyspaces resources with AWS CloudFormation

Amazon Keyspaces (for Apache Cassandra) is integrated with AWS CloudFormation, a service that helps you model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want (like keyspaces and tables), and AWS CloudFormation takes care of provisioning and configuring those resources for you.

When you use AWS CloudFormation, you can reuse your template to set up your Amazon Keyspaces resources consistently and repeatedly. Just describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

## Amazon Keyspaces and AWS CloudFormation templates

To provision and configure resources for Amazon Keyspaces, you must understand [AWS CloudFormation templates](#). Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see [What is AWS CloudFormation designer?](#) in the *AWS CloudFormation User Guide*.

Amazon Keyspaces supports creating keyspaces and tables in AWS CloudFormation. For the tables you create using AWS CloudFormation templates, you can specify the schema, read/write mode, and provisioned throughput settings. For more information, including examples of JSON and YAML templates for keyspaces and tables, see [Cassandra resource type reference](#) in the *AWS CloudFormation User Guide*.

## Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- [AWS CloudFormation](#)
- [AWS CloudFormation User Guide](#)
- [AWS CloudFormation command line interface User Guide](#)

# Adding tags and labels to Amazon Keyspaces resources

You can label Amazon Keyspaces (for Apache Cassandra) resources using *tags*. Tags let you categorize your resources in different ways—for example, by purpose, owner, environment, or other criteria. Tags can help you do the following:

- Quickly identify a resource based on the tags that you assigned to it.
- See AWS bills broken down by tags.

Tagging is supported by AWS services like Amazon Elastic Compute Cloud (Amazon EC2), Amazon Simple Storage Service (Amazon S3), Amazon Keyspaces, and more. Efficient tagging can provide cost insights by enabling you to create reports across services that carry a specific tag.

To get started with tagging, do the following:

1. Understand [Tagging restrictions for Amazon Keyspaces \(p. 126\)](#).
2. Create tags by using [Tagging operations for Amazon Keyspaces \(p. 127\)](#).
3. Use [Cost allocation reports for Amazon Keyspaces \(p. 129\)](#) to track your AWS costs per active tag.

Finally, it is good practice to follow optimal tagging strategies. For information, see [AWS tagging strategies](#).

## Tagging restrictions for Amazon Keyspaces

Each tag consists of a key and a value, both of which you define. The following restrictions apply:

- Each Amazon Keyspaces keyspace or table can have only one tag with the same key. If you try to add an existing tag (same key), the existing tag value is updated to the new value.
- Tags applied to a keyspace do not automatically apply to tables within that keyspace. To apply the same tag to a keyspace and all its tables, each resource must be individually tagged.
- A value acts as a descriptor within a tag category (key). In Amazon Keyspaces the value cannot be empty or null.
- Tag keys and values are case sensitive.
- The maximum key length is 128 Unicode characters.
- The maximum value length is 256 Unicode characters.
- The allowed characters are letters, white space, and numbers, plus the following special characters: + - = . \_ : /
- The maximum number of tags per resource is 50.
- AWS-assigned tag names and values are automatically assigned the `aws :` prefix, which you can't assign. AWS-assigned tag names don't count toward the tag limit of 50. User-assigned tag names have the prefix `user :` in the cost allocation report.
- You can't backdate the application of a tag.

## Tagging operations for Amazon Keyspaces

You can add, list, edit, or delete tags for keyspace and tables using the Amazon Keyspaces (for Apache Cassandra) console or Cassandra Query Language (CQL). You can then activate these user-defined tags so that they appear on the AWS Billing and Cost Management console for cost allocation tracking. For more information, see [Cost allocation reports for Amazon Keyspaces \(p. 129\)](#).

For bulk editing, you can also use Tag Editor on the console. For more information, see [Working with Tag Editor](#) in the *AWS Resource Groups User Guide*.

### Topics

- [Adding tags to new or existing keyspace and tables using the console \(p. 127\)](#)
- [Adding tags to new or existing keyspace and tables using CQL \(p. 128\)](#)

## Adding tags to new or existing keyspace and tables using the console

You can use the Amazon Keyspaces console to add tags to new keyspace and tables when you create them. You can also add, edit, or delete tags for existing tables.

### To tag keyspace when creating them (console)

1. Sign in to the AWS Management Console, and open the Amazon Keyspaces console at <https://console.aws.amazon.com/keyspaces/home>.
2. In the navigation pane, choose **Keyspace**, and then choose **Create keyspace**.
3. On the **Create keyspace** page, provide a name for the keyspace. Enter a key and value for the tag, and then choose **Add new tag**.
4. Choose **Create keyspace**.

### To tag tables when creating them (console)

1. Sign in to the AWS Management Console, and open the Amazon Keyspaces console at <https://console.aws.amazon.com/keyspaces/home>.
2. In the navigation pane, choose **Tables**, and then choose **Create table**.
3. On the **Create table** page in the **Table details** section, select a keyspace and provide a name for the table.
4. In the **Schema** section, create the schema for your table.
5. In the **Table settings** section, choose **Customize settings**.
6. Continue to the **Table tags – optional** section, and choose **Add new tag** to create new tags.
7. Choose **Create table**.

### To tag existing resources (console)

1. Sign in to the AWS Management Console, and open the Amazon Keyspaces console at <https://console.aws.amazon.com/keyspaces/home>.
2. In the navigation pane, choose **Keyspace** or **Tables**.
3. Choose a keyspace or table in the list. Then choose **Manage tags** to add, edit, or delete your tags.

For information about tag structure, see [Tagging restrictions for Amazon Keyspaces \(p. 126\)](#).

## Adding tags to new or existing keyspaces and tables using CQL

The following examples show how to use CQL to specify tags when you create keyspaces and tables, how to tag existing resources, and how to read tags.

The following example creates a new keyspace with tags.

```
CREATE KEYSPACE mykeyspace WITH TAGS = {'key1':'val1', 'key2':'val2'} ;
```

The following example creates a new table with tags.

```
CREATE TABLE mytable(...) WITH TAGS = {'key1':'val1', 'key2':'val2'};
```

To tag resources in a statement with other commands.

```
CREATE KEYSPACE mykeyspace WITH REPLICATION = {'class': 'Simple Strategy'} AND TAGS  
= {'key1':'val1', 'key2':'val2'};
```

The following example shows how to add or remove tags on existing keyspaces and tables.

```
ALTER KEYSPACE mykeyspace ADD TAGS {'key1':'val1', 'key2':'val2'};
```

```
ALTER TABLE mytable DROP TAGS {'key1':'val1', 'key2':'val2'};
```

To read the tags attached to a resource, use the following CQL statement.

```
SELECT * FROM system_schema_mcs.tags WHERE valid_where_clause;
```

The WHERE clause is required, and must be one of the following formats:

- `keyspace_name = 'mykeyspace' AND resource_type = 'keyspace'`
- `keyspace_name = 'mykeyspace' AND resource_name = 'mytable'`
- `resource_id = arn`

### Examples:

The following query shows whether a keyspace has tags.

```
SELECT * FROM system_schema_mcs.tags WHERE keyspace_name = 'mykeyspace' AND resource_type =  
'keyspace';
```

The output of the query looks like the following.

```
resource_id | keyspace_name |  
resource_name | resource_type | tags
```

```
-----+-----  
+-----+-----+-----  
arn:aws:cassandra:us-east-1:123456789:/keyspace/mykeyspace/ | mykeyspace |  
mykeyspace | keyspace | {'key1': 'val1', 'key2': 'val2'}
```

The following query is showing the tags for a table.

```
SELECT * FROM system_schema_mcs.tags WHERE keyspace_name = 'mykeyspace' AND resource_name  
= 'mytable';
```

The output of that query looks like the following.

```
resource_id | keyspace_name  
| resource_name | resource_type | tags  
-----  
+-----+-----+-----+-----  
arn:aws:cassandra:us-east-1:123456789:/keyspace/mykeyspace/table/mytable | mykeyspace  
| mytable | table | {'key1': 'val1', 'key2': 'val2'}
```

## Cost allocation reports for Amazon Keyspaces

AWS uses tags to organize resource costs on your cost allocation report. AWS provides two types of cost allocation tags:

- An AWS-generated tag. AWS defines, creates, and applies this tag for you.
- User-defined tags. You define, create, and apply these tags.

You must activate both types of tags separately before they can appear in Cost Explorer or on a cost allocation report.

To activate AWS-generated tags:

1. Sign in to the AWS Management Console and open the Billing and Cost Management console at <https://console.aws.amazon.com/billing/home#/>.
2. In the navigation pane, choose **Cost Allocation Tags**.
3. Under **AWS-Generated Cost Allocation Tags**, choose **Activate**.

To activate user-defined tags:

1. Sign in to the AWS Management Console and open the Billing and Cost Management console at <https://console.aws.amazon.com/billing/home#/>.
2. In the navigation pane, choose **Cost Allocation Tags**.
3. Under **User-Defined Cost Allocation Tags**, choose **Activate**.

After you create and activate tags, AWS generates a cost allocation report with your usage and costs grouped by your active tags. The cost allocation report includes all of your AWS costs for each billing

period. The report includes both tagged and untagged resources, so that you can clearly organize the charges for resources.

**Note**

Currently, any data transferred out from Amazon Keyspaces won't be broken down by tags on cost allocation reports.

For more information, see [Using cost allocation tags](#).



# Security in Amazon Keyspaces (for Apache Cassandra)

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. The effectiveness of our security is regularly tested and verified by third-party auditors as part of the [AWS compliance programs](#). To learn about the compliance programs that apply to Amazon Keyspaces, see [AWS Services in scope by compliance program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

This documentation will help you understand how to apply the shared responsibility model when using Amazon Keyspaces. The following topics show you how to configure Amazon Keyspaces to meet your security and compliance objectives. You'll also learn how to use other AWS services that can help you to monitor and secure your Amazon Keyspaces resources.

## Topics

- [Data protection in Amazon Keyspaces \(p. 131\)](#)
- [AWS Identity and Access Management for Amazon Keyspaces \(p. 146\)](#)
- [Logging and monitoring in Amazon Keyspaces \(for Apache Cassandra\) \(p. 167\)](#)
- [Compliance validation for Amazon Keyspaces \(for Apache Cassandra\) \(p. 184\)](#)
- [Resilience and disaster recovery in Amazon Keyspaces \(p. 185\)](#)
- [Infrastructure security in Amazon Keyspaces \(p. 185\)](#)
- [Configuration and vulnerability analysis for Amazon Keyspaces \(p. 189\)](#)
- [Security best practices for Amazon Keyspaces \(p. 190\)](#)

## Data protection in Amazon Keyspaces

The AWS [shared responsibility model](#) applies to data protection in Amazon Keyspaces (for Apache Cassandra). As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the [AWS Security Blog](#).

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given

only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a **Name** field. This includes when you work with Amazon Keyspaces or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

#### Topics

- [Encryption at rest in Amazon Keyspaces \(p. 132\)](#)
- [Encryption in transit in Amazon Keyspaces \(p. 145\)](#)
- [Internetwork traffic privacy in Amazon Keyspaces \(p. 146\)](#)

## Encryption at rest in Amazon Keyspaces

Amazon Keyspaces (for Apache Cassandra) *encryption at rest* provides enhanced security by encrypting all your data at rest using encryption keys stored in [AWS Key Management Service \(AWS KMS\)](#). This functionality helps reduce the operational burden and complexity involved in protecting sensitive data. With encryption at rest, you can build security-sensitive applications that meet strict compliance and regulatory requirements for data protection.

Amazon Keyspaces encryption at rest encrypts your data using 256-bit Advanced Encryption Standard (AES-256). This helps secure your data from unauthorized access to the underlying storage.

Amazon Keyspaces encrypts and decrypts the table data transparently. Amazon Keyspaces uses envelope encryption and a key hierarchy to protect data encryption keys. It integrates with AWS KMS for storing and managing the root encryption key. For more information about the encryption key hierarchy, see [the section called “How it works” \(p. 133\)](#). For more information about AWS KMS concepts like envelope encryption, see [AWS KMS management service concepts](#) in the *AWS Key Management Service Developer Guide*.

When creating a new table, you can choose one of the following *AWS KMS keys (KMS keys)*:

- **AWS owned key** – This is the default encryption type. The key is owned by Amazon Keyspaces (no additional charge).
- **Customer managed key** – This key is stored in your account and is created, owned, and managed by you. You have full control over the customer managed key (AWS KMS charges apply).

You can switch between the AWS owned key and the customer managed key at any given time. You can specify a customer managed key when you create a new table or change the KMS key of an existing table

by using the console or programmatically using CQL statements. To learn how, see [Encryption at rest: How to use customer managed keys to encrypt tables in Amazon Keyspaces \(p. 136\)](#).

Encryption at rest using the default option of AWS owned keys is offered at no additional charge. However, AWS KMS charges apply for customer managed keys. For more information about pricing, see [AWS KMS pricing](#).

Amazon Keyspaces encryption at rest is available in all AWS Regions, including the AWS China (Beijing) and AWS China (Ningxia) Regions. For more information, see [Encryption at rest: How it works in Amazon Keyspaces \(p. 133\)](#).

#### Topics

- [Encryption at rest: How it works in Amazon Keyspaces \(p. 133\)](#)
- [Encryption at rest: How to use customer managed keys to encrypt tables in Amazon Keyspaces \(p. 136\)](#)

## Encryption at rest: How it works in Amazon Keyspaces

Amazon Keyspaces (for Apache Cassandra) *encryption at rest* encrypts your data using the 256-bit Advanced Encryption Standard (AES-256). This helps secure your data from unauthorized access to the underlying storage. All customer data in Amazon Keyspaces tables is encrypted at rest by default, and server-side encryption is transparent, which means that changes to applications aren't required.

Encryption at rest integrates with AWS Key Management Service (AWS KMS) for managing the encryption key that is used to encrypt your tables. When creating a new table or updating an existing table, you can choose one of the following *AWS KMS key* options:

- **AWS owned key** – This is the default encryption type. The key is owned by Amazon Keyspaces (no additional charge).
- **Customer managed key** – This key is stored in your account and is created, owned, and managed by you. You have full control over the customer managed key (AWS KMS charges apply).

### AWS KMS key (KMS key)

Encryption at rest protects all your Amazon Keyspaces data with a AWS KMS key. By default, Amazon Keyspaces uses an [AWS owned key](#), a multi-tenant encryption key that is created and managed in an Amazon Keyspaces service account.

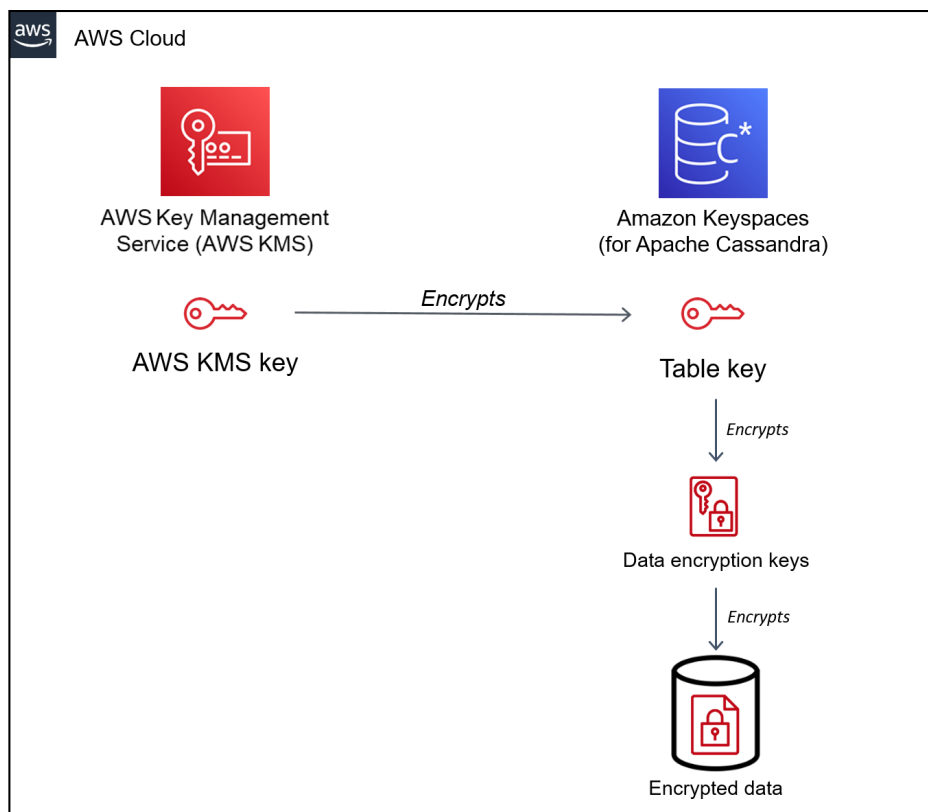
However, you can encrypt your Amazon Keyspaces tables using a [customer managed key](#) in your AWS account. You can select a different KMS key for each table in a keyspace. The KMS key you select for a table is also used to encrypt all its metadata and restorable backups.

You select the KMS key for a table when you create or update the table. You can change the KMS key for a table at any time, either in the Amazon Keyspaces console or by using the [ALTER TABLE \(p. 199\)](#) statement. The process of switching KMS keys is seamless, and doesn't require downtime or cause service degradation.

### Key hierarchy

Amazon Keyspaces uses a key hierarchy to encrypt data. In this key hierarchy, the KMS key is the root key. It's used to encrypt and decrypt the Amazon Keyspaces table encryption key. The table encryption key is used to encrypt the encryption keys used internally by Amazon Keyspaces to encrypt and decrypt data when performing read and write operations.

With the encryption key hierarchy, you can make changes to the KMS key without having to reencrypt data or impacting applications and ongoing data operations.



### Table key

The Amazon Keyspaces table key is used as a key encryption key. Amazon Keyspaces uses the table key to protect the internal data encryption keys that are used to encrypt the data stored in tables, log files, and restorable backups. Amazon Keyspaces generates a unique data encryption key for each underlying structure in a table. However, multiple table rows might be protected by the same data encryption key.

When you first set the KMS key to a customer managed key, AWS KMS generates a *data key*. The AWS KMS data key refers to the table key in Amazon Keyspaces.

When you access an encrypted table, Amazon Keyspaces sends a request to AWS KMS to use the KMS key to decrypt the table key. Then, it uses the plaintext table key to decrypt the Amazon Keyspaces data encryption keys, and it uses the plaintext data encryption keys to decrypt table data.

Amazon Keyspaces uses and stores the table key and data encryption keys outside of AWS KMS. It protects all keys with [Advanced Encryption Standard](#) (AES) encryption and 256-bit encryption keys. Then, it stores the encrypted keys with the encrypted data so that they're available to decrypt the table data on demand.

### Table key caching

To avoid calling AWS KMS for every Amazon Keyspaces operation, Amazon Keyspaces caches the plaintext table keys for each connection in memory. If Amazon Keyspaces gets a request for the cached table key after five minutes of inactivity, it sends a new request to AWS KMS to decrypt the table key. This call captures any changes made to the access policies of the KMS key in AWS KMS or AWS Identity and Access Management (IAM) since the last request to decrypt the table key.

### Envelope encryption

If you change the customer managed key for your table, Amazon Keyspaces generates a new table key. Then, it uses the new table key to reencrypt the data encryption keys. It also uses the new table

key to encrypt previous table keys that are used to protect restorable backups. This process is called envelope encryption. This ensures that you can access restorable backups even if you rotate the customer managed key. For more information about envelope encryption, see [Envelope Encryption](#) in the *AWS Key Management Service Developer Guide*.

### Topics

- [AWS owned keys](#) (p. 135)
- [Customer managed keys](#) (p. 135)
- [Encryption at rest usage notes](#) (p. 136)

## AWS owned keys

AWS owned keys aren't stored in your AWS account. They are part of a collection of KMS keys that AWS owns and manages for use in multiple AWS accounts. AWS services can use AWS owned keys to protect your data.

You can't view, manage, or use AWS owned keys, or audit their use. However, you don't need to do any work or change any programs to protect the keys that encrypt your data.

You aren't charged a monthly fee or a usage fee for use of AWS owned keys, and they don't count against AWS KMS quotas for your account.

## Customer managed keys

Customer managed keys are keys in your AWS account that you create, own, and manage. You have full control over these KMS keys.

Use a customer managed key to get the following features:

- You create and manage the customer managed key, including setting and maintaining the [key policies](#), [IAM policies](#), and [grants](#) to control access to the customer managed key. You can [enable and disable](#) the customer managed key, enable and disable [automatic key rotation](#), and [schedule the customer managed key](#) for deletion when it is no longer in use. You can create tags and aliases for the customer managed keys you manage.
- You can use a customer managed key with [imported key material](#) or a customer managed key in a [custom key store](#) that you own and manage.
- You can use AWS CloudTrail and Amazon CloudWatch Logs to track the requests that Amazon Keyspaces sends to AWS KMS on your behalf. For more information, see [the section called "Step 6: Configure monitoring with AWS CloudTrail"](#) (p. 141).

Customer managed keys [incur a charge](#) for each API call, and AWS KMS quotas apply to these KMS keys. For more information, see [AWS KMS resource or request quotas](#).

When you specify a customer managed key as the root encryption key for a table, restorable backups are encrypted with the same encryption key that is specified for the table at the time the backup is created. If the KMS key for the table is rotated, key enveloping ensures that the latest KMS key has access to all restorable backups.

Amazon Keyspaces must have access to your customer managed key to provide you access to your table data. If the state of the encryption key is set to disabled or it's scheduled for deletion, Amazon Keyspaces is unable to encrypt or decrypt data. As a result, you are not able to perform read and write operations on the table. As soon as the service detects that your encryption key is inaccessible, Amazon Keyspaces sends an email notification to alert you.

You must restore access to your encryption key within seven days or Amazon Keyspaces deletes your table automatically. As a precaution, Amazon Keyspaces creates a restorable backup of your table data

before deleting the table. Amazon Keyspaces maintains the restorable backup for 35 days. After 35 days, you can no longer restore your table data. You aren't billed for the restorable backup, but standard [restore charges apply](#).

You can use this restorable backup to restore your data to a new table. To initiate the restore, the last customer managed key used for the table must be enabled, and Amazon Keyspaces must have access to it.

**Note**

When you're creating a table that's encrypted using a customer managed key that's inaccessible or scheduled for deletion before the creation process completes, an error occurs. The create table operation fails, and you're sent an email notification.

## Encryption at rest usage notes

Consider the following when you're using encryption at rest in Amazon Keyspaces.

- Server-side encryption at rest is enabled on all Amazon Keyspaces tables and can't be disabled. The entire table is encrypted at rest, you can't select specific columns or rows for encryption.
- By default, Amazon Keyspaces uses a single-service default key (AWS owned key) for encrypting all of your tables. If this key doesn't exist, it's created for you. Service default keys can't be disabled.
- Encryption at rest only encrypts data while it's static (at rest) on a persistent storage media. If data security is a concern for data in transit or data in use, you must take additional measures:
  - Data in transit: All your data in Amazon Keyspaces is encrypted in transit. By default, communications to and from Amazon Keyspaces are protected by using Secure Sockets Layer (SSL)/Transport Layer Security (TLS) encryption.
  - Data in use: Protect your data before sending it to Amazon Keyspaces by using client-side encryption.
  - Customer managed keys: Data at rest in your tables is always encrypted using your customer managed keys. However operations that perform atomic updates of multiple rows encrypt data temporarily using AWS owned keys during processing. This includes range delete operations and operations that simultaneously access static and non-static data.
- A single customer managed key can have up to 50,000 [grants](#). Every Amazon Keyspaces table associated with a customer managed key consumes 2 grants. One grant is released when the table is deleted. The second grant is used to create an automatic snapshot of the table to protect from data loss in case Amazon Keyspaces lost access to the customer managed key unintentionally. This grant is released 42 days after deletion of the table.

## Encryption at rest: How to use customer managed keys to encrypt tables in Amazon Keyspaces

You can use the console or CQL statements to specify the AWS KMS key for new tables and update the encryption keys of existing tables in Amazon Keyspaces. The following topic outlines how to implement customer managed keys for new and existing tables.

### Topics

- [Prerequisites: Create a customer managed key using AWS KMS and grant permissions to Amazon Keyspaces \(p. 137\)](#)
- [Step 3: Specify a customer managed key for a new table \(p. 139\)](#)
- [Step 4: Update the encryption key of an existing table \(p. 140\)](#)
- [Step 5: Use the Amazon Keyspaces encryption context in logs \(p. 141\)](#)
- [Step 6: Configure monitoring with AWS CloudTrail \(p. 141\)](#)

## Prerequisites: Create a customer managed key using AWS KMS and grant permissions to Amazon Keyspaces

Before you can protect an Amazon Keyspaces table with a [customer managed key \(p. 135\)](#), you must first create the key in AWS Key Management Service (AWS KMS) and then authorize Amazon Keyspaces to use that key.

### Step 1: Create a customer managed key using AWS KMS

To create a customer managed key to be used to protect an Amazon Keyspaces table, you can follow the steps in [Creating symmetric KMS keys](#) using the console or the AWS API.

### Step 2: Authorize the use of your customer managed key

Before you can choose a [customer managed key \(p. 135\)](#) to protect an Amazon Keyspaces table, the policies on that customer managed key must give Amazon Keyspaces permission to use it on your behalf. You have full control over the policies and grants on the customer managed key. You can provide these permissions in a [key policy](#), an [IAM policy](#), or a [grant](#).

Amazon Keyspaces doesn't need additional authorization to use the default [AWS owned key \(p. 135\)](#) to protect the Amazon Keyspaces tables in your AWS account.

The following topics show how to configure the required permissions using IAM policies and grants that allow Amazon Keyspaces tables to use a customer managed key.

#### Topics

- [Key policy for customer managed keys \(p. 137\)](#)
- [Example key policy \(p. 137\)](#)
- [Using grants to authorize Amazon Keyspaces \(p. 138\)](#)

### Key policy for customer managed keys

When you select a [customer managed key \(p. 135\)](#) to protect an Amazon Keyspaces table, Amazon Keyspaces gets permission to use the customer managed key on behalf of the principal who makes the selection. That principal, a user or role, must have the permissions on the customer managed key that Amazon Keyspaces requires.

At a minimum, Amazon Keyspaces requires the following permissions on a customer managed key:

- [kms:Encrypt](#)
- [kms:Decrypt](#)
- [kms:ReEncrypt\\*](#) (for [kms:ReEncryptFrom](#) and [kms:ReEncryptTo](#))
- [kms:GenerateDataKey\\*](#) (for [kms:GenerateDataKey](#) and [kms:GenerateDataKeyWithoutPlaintext](#))
- [kms:DescribeKey](#)
- [kms:CreateGrant](#)

### Example key policy

For example, the following example key policy provides only the required permissions. The policy has the following effects:

- Allows Amazon Keyspaces to use the customer managed key in cryptographic operations and create grants—but only when it's acting on behalf of principals in the account who have permission to use Amazon Keyspaces. If the principals specified in the policy statement don't have permission to use Amazon Keyspaces, the call fails, even when it comes from the Amazon Keyspaces service.

- The [kms:ViaService](#) condition key allows the permissions only when the request comes from Amazon Keyspaces on behalf of the principals listed in the policy statement. These principals can't call these operations directly. Note that the `kms:ViaService` value, `cassandra.*.amazonaws.com`, has an asterisk (\*) in the Region position. Amazon Keyspaces requires the permission to be independent of any particular AWS Region.
- Gives the customer managed key administrators (users who can assume the `db-team` role) read-only access to the customer managed key and permission to revoke grants, including the [grants that Amazon Keyspaces requires \(p. 138\)](#) to protect the table.
- Gives Amazon Keyspaces read-only access to the customer managed key. In this case, Amazon Keyspaces can call these operations directly. It doesn't have to act on behalf of an account principal.

Before using an example key policy, replace the example principals with actual principals from your AWS account.

```
{
  "Id": "key-policy-cassandra",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow access through Amazon Keyspaces for all principals in the account that
are authorized to use Amazon Keyspaces",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:user/db-lead"},
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey",
        "kms:CreateGrant"
      ],
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "kms:ViaService": "cassandra.*.amazonaws.com"
        }
      }
    },
    {
      "Sid": "Allow administrators to view the customer managed key and revoke grants",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/db-team"
      },
      "Action": [
        "kms:Describe*",
        "kms:Get*",
        "kms:List*",
        "kms:RevokeGrant"
      ],
      "Resource": "*"
    }
  ]
}
```

### Using grants to authorize Amazon Keyspaces

In addition to key policies, Amazon Keyspaces uses grants to set permissions on a customer managed key. To view the grants on a customer managed key in your account, use the [ListGrants](#) operation. Amazon Keyspaces doesn't need grants, or any additional permissions, to use the [AWS owned key \(p. 135\)](#) to protect your table.



Amazon Keyspaces uses the grant permissions when it performs background system maintenance and continuous data protection tasks. It also uses grants to generate table keys.

Each grant is specific to a table. If the account includes multiple tables encrypted under the same customer managed key, there is a grant of each type for each table. The grant is constrained by the [Amazon Keyspaces encryption context](#), which includes the table name and the AWS account ID. The grant includes permission to [retire the grant](#) if it's no longer needed.

To create the grants, Amazon Keyspaces must have permission to call `CreateGrant` on behalf of the user who created the encrypted table.

The key policy can also allow the account to [revoke the grant](#) on the customer managed key. However, if you revoke the grant on an active encrypted table, Amazon Keyspaces will not be able to protect and maintain the table.

### Step 3: Specify a customer managed key for a new table

Follow these steps to specify the customer managed key on a new table using the Amazon Keyspaces console or CQL.

#### Create an encrypted table using a customer managed key (console)

1. Sign in to the AWS Management Console, and open the Amazon Keyspaces console at <https://console.aws.amazon.com/keyspaces/home>.
2. In the navigation pane, choose **Tables**, and then choose **Create table**.
3. On the **Create table** page in the **Table details** section, select a keyspace and provide a name for the new table.
4. In the **Schema** section, create the schema for your table.
5. In the **Table settings** section, choose **Customize settings**.
6. Continue to **Encryption settings**.

In this step, you select the encryption settings for the table.

In the **Encryption at rest** section under **Choose an AWS KMS key**, choose the option **Choose a different KMS key (advanced)**, and in the search field, choose an AWS KMS key or enter an Amazon Resource Name (ARN).

#### Note

If the key you selected is not accessible or is missing the required permissions, see [Troubleshooting key access](#) in the AWS Key Management Service Developer Guide.

7. Choose **Create** to create the encrypted table.

#### Create a new table using a customer managed key for encryption at rest (CQL)

To create a new table that uses a customer managed key for encryption at rest, you can use the `CREATE TABLE` statement as shown in the following example. Make sure to replace the key ARN with an ARN for a valid key with permissions granted to Amazon Keyspaces.

```
CREATE TABLE my_keyspace.my_table(id bigint, name text, place text STATIC, PRIMARY KEY(id,
name)) WITH CUSTOM_PROPERTIES = {
    'encryption_specification':{
        'encryption_type': 'CUSTOMER_MANAGED_KMS_KEY',
        'kms_key_identifier':'arn:aws:kms:eu-
west-1:555555555555:key/11111111-1111-111-1111-111111111111'
    }
};
```

If you receive an `Invalid Request Exception`, you need to confirm that the customer managed key is valid and Amazon Keyspaces has the required permissions. To confirm that the key has been configured correctly, see [Troubleshooting key access](#) in the AWS Key Management Service Developer Guide.

## Step 4: Update the encryption key of an existing table

You can also use the Amazon Keyspaces console or CQL to change the encryption keys of an existing table between an AWS owned key and a customer managed KMS key at any time.

### Update an existing table with the new customer managed key (console)

1. Sign in to the AWS Management Console, and open the Amazon Keyspaces console at <https://console.aws.amazon.com/keyspaces/home>.
2. In the navigation pane, choose **Tables**.
3. Choose the table that you want to update, and then choose the **Additional settings** tab.
4. In the **Encryption at rest** section, choose **Manage Encryption** to edit the encryption settings for the table.

Under **Choose an AWS KMS key**, choose the option **Choose a different KMS key (advanced)**, and in the search field, choose an AWS KMS key or enter an Amazon Resource Name (ARN).

#### Note

If the key you selected is not valid, see [Troubleshooting key access](#) in the AWS Key Management Service Developer Guide.

Alternatively, you can choose an AWS owned key for a table that is encrypted with a customer managed key.

5. Choose **Save changes** to save your changes to the table.

### Update the encryption key used for an existing table

To change the encryption key of an existing table, you use the `ALTER TABLE` statement to specify a customer managed key for encryption at rest. Make sure to replace the key ARN with an ARN for a valid key with permissions granted to Amazon Keyspaces.

```
ALTER TABLE my_keyspace.my_table WITH CUSTOM_PROPERTIES = {
    'encryption_specification':{
        'encryption_type': 'CUSTOMER_MANAGED_KMS_KEY',
        'kms_key_identifier': 'arn:aws:kms:eu-
west-1:555555555555:key/11111111-1111-111-1111-111111111111'
    }
};
```

If you receive an `Invalid Request Exception`, you need to confirm that the customer managed key is valid and Amazon Keyspaces has the required permissions. To confirm that the key has been configured correctly, see [Troubleshooting key access](#) in the AWS Key Management Service Developer Guide.

To change the encryption key back to the default encryption at rest option with AWS owned keys, you can use the `ALTER TABLE` statement as shown in the following example.

```
ALTER TABLE my_keyspace.my_table WITH CUSTOM_PROPERTIES = {
    'encryption_specification':{
        'encryption_type' : 'AWS_OWNED_KMS_KEY'
    }
};
```

## Step 5: Use the Amazon Keyspaces encryption context in logs

An [encryption context](#) is a set of key–value pairs that contain arbitrary nonsecret data. When you include an encryption context in a request to encrypt data, AWS KMS cryptographically binds the encryption context to the encrypted data. To decrypt the data, you must pass in the same encryption context.

Amazon Keyspaces uses the same encryption context in all AWS KMS cryptographic operations. If you use a [customer managed key](#) (p. 135) to protect your Amazon Keyspaces table, you can use the encryption context to identify the use of the customer managed key in audit records and logs. It also appears in plaintext in logs, such as in logs for [AWS CloudTrail](#) and [Amazon CloudWatch Logs](#).

In its requests to AWS KMS, Amazon Keyspaces uses an encryption context with three key–value pairs.

```
"encryptionContextSubset": {
  "aws:cassandra:keyspaceName": "my_keyspace",
  "aws:cassandra:tableName": "mytable"
  "aws:cassandra:subscriberId": "111122223333"
}
```

- **Keyspace** – The first key–value pair identifies the keyspace that includes the table that Amazon Keyspaces is encrypting. The key is `aws:cassandra:keyspaceName`. The value is the name of the keyspace.

```
"aws:cassandra:keyspaceName": "<keyspace-name>"
```

For example:

```
"aws:cassandra:keyspaceName": "my_keyspace"
```

- **Table** – The second key–value pair identifies the table that Amazon Keyspaces is encrypting. The key is `aws:cassandra:tableName`. The value is the name of the table.

```
"aws:cassandra:tableName": "<table-name>"
```

For example:

```
"aws:cassandra:tableName": "my_table"
```

- **Account** – The third key–value pair identifies the AWS account. The key is `aws:cassandra:subscriberId`. The value is the account ID.

```
"aws:cassandra:subscriberId": "<account-id>"
```

For example:

```
"aws:cassandra:subscriberId": "111122223333"
```

## Step 6: Configure monitoring with AWS CloudTrail

If you use a [customer managed key](#) (p. 135) to protect your Amazon Keyspaces tables, you can use AWS CloudTrail logs to track the requests that Amazon Keyspaces sends to AWS KMS on your behalf.

The `GenerateDataKey`, `DescribeKey`, `Decrypt`, and `CreateGrant` requests are discussed in this section. In addition, Amazon Keyspaces uses a [RetireGrant](#) operation to remove a grant when you delete a table.

## GenerateDataKey

Amazon Keyspaces creates a unique table key to encrypt data at rest. It sends a [GenerateDataKey](#) request to AWS KMS that specifies the KMS key for the table.

The event that records the `GenerateDataKey` operation is similar to the following example event. The user is the Amazon Keyspaces service account. The parameters include the Amazon Resource Name (ARN) of the customer managed key, a key specifier that requires a 256-bit key, and the [encryption context](#) (p. 141) that identifies the keyspace, the table, and the AWS account.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AWSService",
    "invokedBy": "AWS Internal"
  },
  "eventTime": "2021-04-16T04:56:05Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "GenerateDataKey",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "AWS Internal",
  "userAgent": "AWS Internal",
  "requestParameters": {
    "keySpec": "AES_256",
    "encryptionContext": {
      "aws:cassandra:keyspaceName": "my_keyspace",
      "aws:cassandra:tableName": "my_table",
      "aws:cassandra:subscriberId": "123SAMPLE012"
    },
    "keyId": "arn:aws:kms:eu-west-1:555555555555:key/11111111-1111-111-1111-111111111111"
  },
  "responseElements": null,
  "requestID": "5e8e9cb5-9194-4334-aacc-9dd7d50fe246",
  "eventID": "49fccab9-2448-4b97-a89d-7d5c39318d6f",
  "readOnly": true,
  "resources": [
    {
      "accountId": "123SAMPLE012",
      "type": "AWS::KMS::Key",
      "ARN": "arn:aws:kms:eu-west-1:555555555555:key/11111111-1111-111-1111-111111111111"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "eventCategory": "Management",
  "recipientAccountId": "123SAMPLE012",
  "sharedEventID": "84fbaaf0-9641-4e32-9147-57d2cb08792e"
}
```

## DescribeKey

Amazon Keyspaces uses a [DescribeKey](#) operation to determine whether the KMS key you selected exists in the account and Region.

The event that records the `DescribeKey` operation is similar to the following example event. The user is the Amazon Keyspaces service account. The parameters include the ARN of the customer managed key and a key specifier that requires a 256-bit key.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
```

```
    "type": "IAMUser",
    "principalId": "AIDAZ3FNIIVIZZ6H7CFQG",
    "arn": "arn:aws:iam::123SAMPLE012:user/admin",
    "accountId": "123SAMPLE012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "admin",
    "sessionContext": {
      "sessionIssuer": {},
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2021-04-16T04:55:42Z"
      }
    },
    "invokedBy": "AWS Internal"
  },
  "eventTime": "2021-04-16T04:55:58Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "DescribeKey",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "AWS Internal",
  "userAgent": "AWS Internal",
  "requestParameters": {
    "keyId": "arn:aws:kms:eu-west-1:555555555555:key/11111111-1111-111-1111-111111111111"
  },
  "responseElements": null,
  "requestID": "c25a8105-050b-4f52-8358-6e872fb03a6c",
  "eventID": "0d96420e-707e-41b9-9118-56585a669658",
  "readOnly": true,
  "resources": [
    {
      "accountId": "123SAMPLE012",
      "type": "AWS::KMS::Key",
      "ARN": "arn:aws:kms:eu-west-1:555555555555:key/11111111-1111-111-1111-111111111111"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "eventCategory": "Management",
  "recipientAccountId": "123SAMPLE012"
}
```

## Decrypt

When you access an Amazon Keyspaces table, Amazon Keyspaces needs to decrypt the table key so that it can decrypt the keys below it in the hierarchy. It then decrypts the data in the table. To decrypt the table key, Amazon Keyspaces sends a [Decrypt](#) request to AWS KMS that specifies the KMS key for the table.

The event that records the `Decrypt` operation is similar to the following example event. The user is the principal in your AWS account who is accessing the table. The parameters include the encrypted table key (as a ciphertext blob) and the [encryption context](#) (p. 141) that identifies the table and the AWS account. AWS KMS derives the ID of the customer managed key from the ciphertext.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AWSService",
    "invokedBy": "AWS Internal"
  },
  "eventTime": "2021-04-16T05:29:44Z",
  "eventSource": "kms.amazonaws.com",
```

```
"eventName": "Decrypt",
"awsRegion": "us-east-1",
"sourceIPAddress": "AWS Internal",
"userAgent": "AWS Internal",
"requestParameters": {
  "encryptionContext": {
    "aws:cassandra:keyspaceName": "my_keyspace",
    "aws:cassandra:tableName": "my_table",
    "aws:cassandra:subscriberId": "123SAMPLE012"
  },
  "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
},
"responseElements": null,
"requestID": "50e80373-83c9-4034-8226-5439e1c9b259",
"eventID": "8db9788f-04a5-4ae2-90c9-15c79c411b6b",
"readOnly": true,
"resources": [
  {
    "accountId": "123SAMPLE012",
    "type": "AWS::KMS::Key",
    "ARN": "arn:aws:kms:eu-west-1:555555555555:key/11111111-1111-1111-1111-111111111111"
  }
],
"eventType": "AwsApiCall",
"managementEvent": true,
"eventCategory": "Management",
"recipientAccountId": "123SAMPLE012",
"sharedEventID": "7ed99e2d-910a-4708-a4e3-0180d8dbb68e"
}
```

## CreateGrant

When you use a [customer managed key \(p. 135\)](#) to protect your Amazon Keyspaces table, Amazon Keyspaces uses [grants \(p. 138\)](#) to allow the service to perform continuous data protection and maintenance and durability tasks. These grants aren't required on [AWS owned keys \(p. 135\)](#).

The grants that Amazon Keyspaces creates are specific to a table. The principal in the [CreateGrant](#) request is the user who created the table.

The event that records the `CreateGrant` operation is similar to the following example event. The parameters include the ARN of the customer managed key for the table, the grantee principal and retiring principal (the Amazon Keyspaces service), and the operations that the grant covers. It also includes a constraint that requires all encryption operations use the specified [encryption context \(p. 141\)](#).

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDAZ3FNIIVIZZ6H7CFQG",
    "arn": "arn:aws:iam::arn:aws:kms:eu-west-1:555555555555:key/11111111-1111-1111-1111-111111111111:user/admin",
    "accountId": "arn:aws:kms:eu-west-1:555555555555:key/11111111-1111-1111-1111-111111111111",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "userName": "admin",
    "sessionContext": {
      "sessionIssuer": {},
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2021-04-16T04:55:42Z"
      }
    }
  }
}
```

```

    },
    "invokedBy": "AWS Internal"
  },
  "eventTime": "2021-04-16T05:11:10Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "CreateGrant",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "AWS Internal",
  "userAgent": "AWS Internal",
  "requestParameters": {
    "keyId": "a7d328af-215e-4661-9a69-88c858909f20",
    "operations": [
      "DescribeKey",
      "GenerateDataKey",
      "Decrypt",
      "Encrypt",
      "ReEncryptFrom",
      "ReEncryptTo",
      "RetireGrant"
    ],
    "constraints": {
      "encryptionContextSubset": {
        "aws:cassandra:keyspaceName": "my_keyspace",
        "aws:cassandra:tableName": "my_table",
        "aws:cassandra:subscriberId": "123SAMPLE012"
      }
    }
  },
  "retiringPrincipal": "cassandratest.us-east-1.amazonaws.com",
  "granteePrincipal": "cassandratest.us-east-1.amazonaws.com"
},
"responseElements": {
  "grantId": "18e4235f1b07f289762a31a1886cb5efd225f069280d4f76cd83b9b9b5501013"
},
"requestID": "b379a767-1f9b-48c3-b731-fb23e865e7f7",
"eventID": "29ee1fd4-28f2-416f-a419-551910d20291",
"readOnly": false,
"resources": [
  {
    "accountId": "123SAMPLE012",
    "type": "AWS::KMS::Key",
    "ARN": "arn:aws:kms:eu-west-1:555555555555:key/11111111-1111-1111-1111-111111111111"
  }
],
"eventType": "AwsApiCall",
"managementEvent": true,
"eventCategory": "Management",
"recipientAccountId": "123SAMPLE012"
}

```

## Encryption in transit in Amazon Keyspaces

Amazon Keyspaces only accepts secure connections using Transport Layer Security (TLS). Encryption in transit provides an additional layer of data protection by encrypting your data as it travels to and from Amazon Keyspaces. Organizational policies, industry or government regulations, and compliance requirements often require the use of encryption in transit to increase the data security of your applications when they transmit data over the network.

To learn how to encrypt `cqlsh` connections to Amazon Keyspaces using TLS, see [the section called “Encrypting `cqlsh` connections using TLS”](#) (p. 20). To learn how to use TLS encryption with client drivers, see [the section called “Using a Cassandra client driver”](#) (p. 21).

## Internetwork traffic privacy in Amazon Keyspaces

This topic describes how Amazon Keyspaces (for Apache Cassandra) secures connections from on-premises applications to Amazon Keyspaces and between Amazon Keyspaces and other AWS resources within the same AWS Region.

### Traffic between service and on-premises clients and applications

You have two connectivity options between your private network and AWS:

- An AWS Site-to-Site VPN connection. For more information, see [What is AWS Site-to-Site VPN?](#) in the *AWS Site-to-Site VPN User Guide*.
- An AWS Direct Connect connection. For more information, see [What is AWS Direct Connect?](#) in the *AWS Direct Connect User Guide*.

Access to Amazon Keyspaces via the network is through AWS published APIs. Clients must support Transport Layer Security (TLS) 1.0. We recommend TLS 1.2 or above. Clients must also support cipher suites with Perfect Forward Secrecy (PFS), such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). Most modern systems such as Java 7 and later support these modes.

Amazon Keyspaces supports two methods of authenticating client requests. The first method uses service-specific credentials, which are password based credentials generated for a specific AWS Identity and Access Management (IAM) user. You can create and manage the password using the IAM console, the AWS CLI, or the AWS API. For more information, see [Using IAM with Amazon Keyspaces](#).

The second method uses an authentication plugin for the open-source DataStax Java Driver for Cassandra. This plugin enables [IAM users, roles, and federated identities](#) to add authentication information to Amazon Keyspaces (for Apache Cassandra) API requests using the [AWS Signature Version 4 process \(SigV4\)](#). For more information, see [??? \(p. 15\)](#).

### Traffic between AWS resources in the same Region

Interface VPC endpoints enable private communication between your virtual private cloud (VPC) running in Amazon VPC and Amazon Keyspaces. Interface VPC endpoints are powered by AWS PrivateLink, which is an AWS service that enables private communication between VPCs and AWS services. AWS PrivateLink enables this by using an elastic network interface with private IPs in your VPC so that network traffic does not leave the Amazon network. Interface VPC endpoints don't require an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. For more information, see [Amazon Virtual Private Cloud](#) and [Interface VPC endpoints \(AWS PrivateLink\)](#). For example policies, see [the section called "Using interface VPC endpoints for Amazon Keyspaces" \(p. 186\)](#).

## AWS Identity and Access Management for Amazon Keyspaces

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon Keyspaces resources. IAM is an AWS service that you can use with no additional charge.

#### Topics

- [Audience \(p. 147\)](#)
- [Authenticating with identities \(p. 147\)](#)



- [Managing access using policies \(p. 149\)](#)
- [How Amazon Keyspaces works with IAM \(p. 151\)](#)
- [Amazon Keyspaces identity-based policy examples \(p. 154\)](#)
- [AWS managed policies for Amazon Keyspaces \(p. 160\)](#)
- [Troubleshooting Amazon Keyspaces identity and access \(p. 163\)](#)
- [Using service-linked roles for Amazon Keyspaces \(p. 166\)](#)

## Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon Keyspaces.

**Service user** – If you use the Amazon Keyspaces service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon Keyspaces features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon Keyspaces, see [Troubleshooting Amazon Keyspaces identity and access \(p. 163\)](#).

**Service administrator** – If you're in charge of Amazon Keyspaces resources at your company, you probably have full access to Amazon Keyspaces. It's your job to determine which Amazon Keyspaces features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon Keyspaces, see [How Amazon Keyspaces works with IAM \(p. 151\)](#).

**IAM administrator** – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon Keyspaces. To view example Amazon Keyspaces identity-based policies that you can use in IAM, see [Amazon Keyspaces identity-based policy examples \(p. 154\)](#).

## Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see [Signing in to the AWS Management Console as an IAM user or root user](#) in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the [AWS Management Console](#), use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see [Signature Version 4 signing process](#) in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Using multi-factor authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

## AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and

is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the [best practice of using the root user only to create your first IAM user](#). Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

## IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see [Managing access keys for IAM users](#) in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [When to create an IAM user \(instead of a role\)](#) in the *IAM User Guide*.

## IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by [switching roles](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Using IAM roles](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an [identity provider](#). For more information about federated users, see [Federated users and roles](#) in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
- **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional

dependent actions in a policy, see [Actions, Resources, and Condition Keys for Amazon Keyspaces \(for Apache Cassandra\)](#) in the *Service Authorization Reference*.

- **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see [When to create an IAM role \(instead of a user\)](#) in the *IAM User Guide*.

## Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

## Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choosing between managed policies and inline policies](#) in the *IAM User Guide*.

## Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

## Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access control list \(ACL\) overview](#) in the *Amazon Simple Storage Service Developer Guide*.

## Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see [How SCPs work](#) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see [Session policies](#) in the *IAM User Guide*.

## Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

## How Amazon Keyspaces works with IAM

Before you use IAM to manage access to Amazon Keyspaces, you should understand what IAM features are available to use with Amazon Keyspaces. To get a high-level view of how Amazon Keyspaces and other AWS services work with IAM, see [AWS services that work with IAM](#) in the *IAM User Guide*.

### Topics

- [Amazon Keyspaces identity-based policies](#) (p. 151)
- [Amazon Keyspaces resource-based policies](#) (p. 153)
- [Authorization based on Amazon Keyspaces tags](#) (p. 153)
- [Amazon Keyspaces IAM roles](#) (p. 153)

## Amazon Keyspaces identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon Keyspaces supports specific actions and resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

To see the Amazon Keyspaces service-specific resources and actions, and condition context keys that can be used for IAM permissions policies, see the [Actions, Resources, and Condition Keys for Amazon Keyspaces \(for Apache Cassandra\)](#) in the *Service Authorization Reference*.

### Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The **Action** element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon Keyspaces use the following prefix before the action: `cassandra:`. For example, to grant someone permission to create an Amazon Keyspaces keyspace with the Amazon Keyspaces `CREATE` CQL statement, you include the `cassandra:Create` action in their policy. Policy statements must include either an **Action** or **NotAction** element. Amazon Keyspaces defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [
    "cassandra:CREATE",
    "cassandra:MODIFY"
]
```

To see a list of Amazon Keyspaces actions, see [Actions Defined by Amazon Keyspaces \(for Apache Cassandra\)](#) in the *Service Authorization Reference*.

### Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (\*) to indicate that the statement applies to all resources.

```
"Resource": "*" 
```

In Amazon Keyspaces keyspaces and tables can be used in the `Resource` element of IAM permissions.

The Amazon Keyspaces keyspace resource has the following ARN:

```
arn:${Partition}:cassandra:${Region}:${Account}:/keyspace/${KeyspaceName}
```

The Amazon Keyspaces table resource has the following ARN:

```
arn:${Partition}:cassandra:${Region}:${Account}:/keyspace/${KeyspaceName}/table/  
${tableName}
```

For more information about the format of ARNs, see [Amazon Resource Names \(ARNs\) and AWS service namespaces](#).

For example, to specify the `mykeyspace` keyspace in your statement, use the following ARN:

```
"Resource": "arn:aws:cassandra:us-east-1:123456789012:/keyspace/mykeyspace" 
```

To specify all keyspaces that belong to a specific account, use the wildcard (\*):

```
"Resource": "arn:aws:cassandra:us-east-1:123456789012:/keyspace/*" 
```

Some Amazon Keyspaces actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (\*).

```
"Resource": "*" 
```

To connect to Amazon Keyspaces programmatically with a standard driver, a user must have `SELECT` access to the system tables, because most drivers read the system keyspaces/tables on connection. For example, to grant `SELECT` permissions to a user for `mytable` in `mykeyspace`, the IAM user must have permissions to read both, `mytable` and the `system` keyspace. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/mytable",  
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*" 
```

To see a list of Amazon Keyspaces resource types and their ARNs, see [Resources Defined by Amazon Keyspaces \(for Apache Cassandra\)](#) in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see [Actions Defined by Amazon Keyspaces \(for Apache Cassandra\)](#).

## Condition keys

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.



The `Condition` element (or *Condition block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM policy elements: variables and tags](#) in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

Amazon Keyspaces defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

All Amazon Keyspaces actions support the `aws:RequestTag/${TagKey}`, the `aws:ResourceTag/${TagKey}`, and the `aws:TagKeys` condition keys. For more information, see [the section called "Amazon Keyspaces resource access based on tags" \(p. 159\)](#).

To see a list of Amazon Keyspaces condition keys, see [Condition Keys for Amazon Keyspaces \(for Apache Cassandra\)](#) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions Defined by Amazon Keyspaces \(for Apache Cassandra\)](#).

## Examples

To view examples of Amazon Keyspaces identity-based policies, see [Amazon Keyspaces identity-based policy examples \(p. 154\)](#).

## Amazon Keyspaces resource-based policies

Amazon Keyspaces does not support resource-based policies. To view an example of a detailed resource-based policy page, see <https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html>.

## Authorization based on Amazon Keyspaces tags

You can manage access to your Amazon Keyspaces resources by using tags. To manage resource access based on tags, you provide tag information in the [condition element](#) of a policy using the `cassandra:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. For more information about tagging Amazon Keyspaces resources, see [Tagging resources \(p. 126\)](#).

To view example identity-based policies for limiting access to a resource based on the tags on that resource, see [Amazon Keyspaces resource access based on tags \(p. 159\)](#).

## Amazon Keyspaces IAM roles

An [IAM role](#) is an entity within your AWS account that has specific permissions.

## Using temporary credentials with Amazon Keyspaces

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as [AssumeRole](#) or [GetFederationToken](#).

Amazon Keyspaces supports using temporary credentials with the [Amazon Keyspaces authentication plugin \(p. 26\)](#). To view an example of how to use the authentication plugin to access a table

programmatically, see [the section called “Accessing Amazon Keyspaces using the authentication plugin” \(p. 157\)](#).

## Service-linked roles

[Service-linked roles](#) allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

For details about creating or managing Amazon Keyspaces service-linked roles, see [the section called “Using service-linked roles” \(p. 166\)](#).

## Service roles

Amazon Keyspaces does not support service roles.

# Amazon Keyspaces identity-based policy examples

By default, IAM users and roles don't have permission to create or modify Amazon Keyspaces resources. They also can't perform tasks using the console, CQLSH, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see [Creating policies on the JSON tab](#) in the *IAM User Guide*.

## Topics

- [Policy best practices \(p. 154\)](#)
- [Using the Amazon Keyspaces console \(p. 155\)](#)
- [Allow users to view their own permissions \(p. 155\)](#)
- [Accessing Amazon Keyspaces tables \(p. 155\)](#)
- [Accessing Amazon Keyspaces using the authentication plugin \(p. 157\)](#)
- [Amazon Keyspaces resource access based on tags \(p. 159\)](#)

## Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon Keyspaces resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started using AWS managed policies** – To start using Amazon Keyspaces quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see [Get started using permissions with AWS managed policies](#) in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see [Grant least privilege](#) in the *IAM User Guide*.
- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see [Using multi-factor authentication \(MFA\) in AWS](#) in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to



specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.

## Using the Amazon Keyspaces console

Amazon Keyspaces does not require specific permissions to access the Amazon Keyspaces (for Apache Cassandra) console. You need at least read-only permissions to list and view details about the Amazon Keyspaces resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

## Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
  "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": "*"
    }
  ]
}
```

## Accessing Amazon Keyspaces tables

The following is a sample policy that grants read-only (`SELECT`) access to the system tables. For all samples, replace the Region and account ID in the Amazon Resource Name (ARN) with the one from your service.

### Note

To connect with a standard driver, a user must have at least `SELECT` access to the system tables, because most drivers read the system keyspaces/tables on connection.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cassandra:Select"
      ],
      "Resource": [
        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
      ]
    }
  ]
}
```

The following sample policy adds read-only access to a user table.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cassandra:Select"
      ],
      "Resource": [
        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/mytable",
        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
      ]
    }
  ]
}
```

The following sample policy assigns read/write access to a user table and read access to the system tables.

**Note**

System tables are read-only.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cassandra:Select",
        "cassandra:Modify"
      ],
      "Resource": [
        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/mytable",
        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
      ]
    }
  ]
}
```

The following sample policy allows a user to create tables in keyspace mykeyspace.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cassandra:Create",
        "cassandra:Select"
      ],
      "Resource": [
        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/*",
        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
      ]
    }
  ]
}
```

## Accessing Amazon Keyspaces using the authentication plugin

In the following example, you run an application that uses the [Amazon Keyspaces authentication plugin \(p. 26\)](#) to access Amazon Keyspaces from an EC2 instance associated with an IAM role to query your Amazon Keyspaces table. The application returns all orders from a table called `orders` for a given customer ID.

The example includes the following steps:

1. Create an Amazon Keyspaces keyspace and table, and insert records.
2. Launch an EC2 instance and associate it with an IAM role.

Define the IAM policy that gives the EC2 instance permissions to your Amazon Keyspaces table.

3. Deploy and run the sample code on the EC2 instance to query your Amazon Keyspaces table.

### Create a sample table

To start, create a keyspace and a table, and insert some records. To create the `orders` table, you can use the CQL Editor on the Amazon Keyspaces console, or you can use `cqlsh`. For more information, see [the section called "Installing and using `cqlsh`" \(p. 19\)](#).

First, create a keyspace and a table with the following table schema.

```
create keyspace acme with replication = {
  'class': 'SimpleStrategy', 'replication_factor' : 1
};
create table acme.orders (
  customer_id text,
  order_timestamp timestamp,
  order_id uuid,
  primary key (customer_id, order_timestamp)
) with clustering order by (order_timestamp desc);
```

Next, insert a few order records into the table.

```
insert into acme.orders (customer_id, order_timestamp, order_id)
values ('1234', toTimestamp(now()), uuid());
insert into acme.orders (customer_id, order_timestamp, order_id)
values ('1234', toTimestamp(now()), uuid());
insert into acme.orders (customer_id, order_timestamp, order_id)
values ('1234', toTimestamp(now()), uuid());
insert into acme.orders (customer_id, order_timestamp, order_id)
values ('1234', toTimestamp(now()), uuid());
```

## Launch an EC2 instance associated with the IAM role

Create the IAM role that will be associated with the EC2 instance that will run the application.

Save the following document to a file named `iam-keyspaces-ec2-role.json`.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"Service": "ec2.amazonaws.com"},
    "Action": "sts:AssumeRole"
  }
}
```

To give the EC2 instance read-only access to your `orders` table, create the following IAM policy. Replace the Region and account ID in the Amazon Resource Name (ARN) with that from your Amazon Keyspaces account.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "cassandra:Select",
      "Resource": [
        "arn:aws:cassandra:us-east-2:111122223333:/keyspace/acme/table/
orders",
        "arn:aws:cassandra:us-east-2:111122223333:/keyspace/system/*"
      ]
    }
  ]
}
```

Save the following document to a file named `iam-keyspaces-ec2-policy.json`.

### Note

You must include permissions for the `system` keyspace in the policy because the driver has to access this keyspace to retrieve cluster metadata.

To launch an EC2 instance and associate the created IAM role with the instance, follow the steps in [Launching an instance with an IAM role](#). To associate the role with an existing EC2 instance, see [Attaching an IAM role](#).

### Note

Choose an Amazon Linux AMI with Java pre-installed for this example. If you are using an existing Amazon Linux instance, confirm that you have Java installed, or complete this step using the command `sudo yum install java-11-amazon-corretto-headless`.

## Deploying and running the code

To deploy and run the code to query your Amazon Keyspaces table, use the following steps:

1. Download the sample code from GitHub.
2. Compile the sample code using Apache Maven.
3. Copy the JAR file to the EC2 instance, and run the application.

This example uses a sample application that implements the [Amazon Keyspaces authentication plugin \(p. 26\)](#) to access the table using the IAM role. In your own code, you must first integrate the plugin following the steps in [the section called "Authentication plugin for Java 4.x" \(p. 26\)](#).

After downloading the sample from the [GitHub repository](#), compile the code using Apache Maven version 3.6.3 or higher using the following command.

```
mvn package
```

This creates the following JAR file with all dependencies included in the `target` directory.

```
aws-sigv4-auth-cassandra-java-driver-examples-1.0.0.jar
```

Copy this JAR file along with the `cassandra_truststore.jks` located in the sample code directory to the EC2 instance. Then run the application using the following command. Replace the AWS Region `us-east-2` with the Region that you are using.

```
java -Djavax.net.ssl.trustStore=./cassandra_truststore.jks \  
-Djavax.net.ssl.trustStorePassword=amazon -jar \  
aws-sigv4-auth-cassandra-java-driver-examples-1.0.0.jar \  
us-east-2 cassandra.us-east-2.amazonaws.com 1234
```

This command results in the records that you inserted into the table earlier. The application accessed the table using temporary credentials assigned to the IAM role session and with the read-only permissions defined in the IAM policy.

## Amazon Keyspaces resource access based on tags

You can use conditions in your identity-based policy to control access to Amazon Keyspaces resources based on tags. This section provides some examples.

The following example shows how you can create a policy that grants permissions to a user to view a table if the table's Owner contains the value of that user's user name.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "ReadOnlyAccessTaggedTables",  
      "Effect": "Allow",  
      "Action": "cassandra:Select",  
      "Resource": "arn:aws:cassandra:us-east-2:111122223333:/keyspace/mykeyspace/  
table/*",  
      "Condition": {  
        "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}      }  
    }  
  ]  
}
```

```
}  
]  
}
```

You can attach this policy to the IAM users in your account. If a user named `richard-roe` attempts to view an Amazon Keyspaces table, the table must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise, he is denied access. The condition tag key `Owner` matches both `Owner` and `owner` because condition key names are not case-sensitive. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.

The following policy grants permissions to a user to create tables with tags if the table's `Owner` contains the value of that user's user name.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "CreateTagTableUser",  
      "Effect": "Allow",  
      "Action": ["cassandra:Create", "cassandra:TagResource"],  
      "Resource": "arn:aws:cassandra:us-east-2:111122223333:/keyspace/mykeyspace/  
table/*",  
      "Condition": {  
        "ForAnyValue:StringEquals" : {"aws:RequestTag/Owner": "${aws:username}" }  
      }  
    }  
  ]  
}
```

## AWS managed policies for Amazon Keyspaces

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to [create IAM customer managed policies](#) that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see [AWS managed policies](#) in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ViewOnlyAccess** AWS managed policy provides read-only access to many AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see [AWS managed policies for job functions](#) in the *IAM User Guide*.

## AWS managed policy: AmazonKeyspacesReadOnlyAccess

You can attach the `AmazonKeyspacesReadOnlyAccess` policy to your IAM identities.

This policy grants read-only access to Amazon Keyspaces.

#### Permissions details

This policy includes the following permissions.

- **Amazon Keyspaces** – Provides read-only access to Amazon Keyspaces.
- **Application Auto Scaling** – Allows principals to view configurations from Application Auto Scaling. This is required so that users can view automatic scaling policies that are attached to a table.
- **CloudWatch** – Allows principals to view alarms configured in CloudWatch. This is required so users can view CloudWatch alarms that have been configured for a table.
- **AWS KMS** – Allows principals to view keys configured in AWS KMS. This is required so users can view AWS KMS keys that they create and manage in their account to confirm that the key assigned to Amazon Keyspaces is a symmetric key that is enabled.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cassandra:Select"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:DescribeScalableTargets",
        "application-autoscaling:DescribeScalingActivities",
        "application-autoscaling:DescribeScalingPolicies",
        "application-autoscaling:DescribeScheduledActions",
        "cloudwatch:DescribeAlarms",
        "kms:DescribeKey",
        "kms:ListAliases"
      ],
      "Resource": "*"
    }
  ]
}
```

## AWS managed policy: AmazonKeyspacesFullAccess

You can attach the AmazonKeyspacesFullAccess policy to your IAM identities.

This policy grants administrative permissions that allow your administrators unrestricted access to Amazon Keyspaces.

#### Permissions details

This policy includes the following permissions.

- **Amazon Keyspaces** – Allows principals to access any Amazon Keyspaces resource and perform all actions.
- **Application Auto Scaling** – Allows principals to create, view, and delete automatic scaling policies for Amazon Keyspaces tables. This is required so that administrators can manage automatic scaling policies for Amazon Keyspaces tables.
- **CloudWatch** – Allows principals to create, view, and delete CloudWatch alarms for Amazon Keyspaces automatic scaling policies. This is required so that administrators can create a CloudWatch dashboard.
- **IAM** – Allows Amazon Keyspaces to create a service-linked role with IAM automatically when an administrator enables Application Auto Scaling for a table. This is required so that Amazon Keyspaces can perform automatic scaling actions on your behalf.
- **AWS KMS** – Allows principals to view keys configured in AWS KMS. This is required so that users can view AWS KMS keys that they create and manage in their account to confirm that the key assigned to Amazon Keyspaces is a symmetric key that is enabled.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cassandra:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "application-autoscaling:DeleteScalingPolicy",
        "application-autoscaling:DeleteScheduledAction",
        "application-autoscaling:DeregisterScalableTarget",
        "application-autoscaling:DescribeScalableTargets",
        "application-autoscaling:DescribeScalingActivities",
        "application-autoscaling:DescribeScalingPolicies",
        "application-autoscaling:DescribeScheduledActions",
        "application-autoscaling:PutScheduledAction",
        "application-autoscaling:PutScalingPolicy",
        "application-autoscaling:RegisterScalableTarget",
        "kms:DescribeKey",
        "kms:ListAliases"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:DeleteAlarms",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:PutMetricAlarm"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/cassandra.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_CassandraTable",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "cassandra.application-autoscaling.amazonaws.com"
        }
      }
    }
  ]
}
```



```
}  
}  
]  
}
```

## Amazon Keyspaces updates to AWS managed policies

View details about updates to AWS managed policies for Amazon Keyspaces since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the [Document history \(p. 212\)](#) page.

Change	Description	Date
<a href="#">AmazonKeyspacesReadOnlyAccess (p. 167)</a> – Update to an existing policy	Amazon Keyspaces added new permissions to allow users to view AWS KMS keys that have been configured for Amazon Keyspaces encryption at rest.  Amazon Keyspaces encryption at rest integrates with AWS KMS for protecting and managing the encryption keys used to encrypt data at rest. To view the AWS KMS key configured for Amazon Keyspaces, read-only permissions have been added.	06/01/2021
<a href="#">AmazonKeyspacesFullAccess (p. 167)</a> – Update to an existing policy	Amazon Keyspaces added new permissions to allow users to view AWS KMS keys that have been configured for Amazon Keyspaces encryption at rest.  Amazon Keyspaces encryption at rest integrates with AWS KMS for protecting and managing the encryption keys used to encrypt data at rest. To view the AWS KMS key configured for Amazon Keyspaces, read-only permissions have been added.	06/01/2021
Amazon Keyspaces started tracking changes	Amazon Keyspaces started tracking changes for its AWS managed policies.	June 1, 2021

## Troubleshooting Amazon Keyspaces identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon Keyspaces and IAM.

## Topics

- [I'm not authorized to perform an action in Amazon Keyspaces \(p. 164\)](#)
- [I modified an IAM user or role and the changes did not take effect immediately \(p. 164\)](#)
- [I can't restore a table using Amazon Keyspaces point-in-time recovery \(PITR\) \(p. 164\)](#)
- [I'm not authorized to perform iam:PassRole \(p. 165\)](#)
- [I want to view my access keys \(p. 165\)](#)
- [I'm an administrator and want to allow others to access Amazon Keyspaces \(p. 165\)](#)
- [I want to allow people outside of my AWS account to access my Amazon Keyspaces resources \(p. 165\)](#)

## I'm not authorized to perform an action in Amazon Keyspaces

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a `table` but does not have `cassandra:Select` permissions for the table.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
cassandra:Select on resource: mytable
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `mytable` resource using the `cassandra:Select` action.

## I modified an IAM user or role and the changes did not take effect immediately

IAM policy changes may take up to 10 minutes to take effect for applications with existing, established connections to Amazon Keyspaces. IAM policy changes take effect immediately when applications establish a new connection. If you have made modifications to an existing IAM user or role, and it has not taken immediate effect, either wait for 10 minutes or disconnect and reconnect to Amazon Keyspaces.

## I can't restore a table using Amazon Keyspaces point-in-time recovery (PITR)

If you are trying to restore an Amazon Keyspaces table with point-in-time recovery (PITR), and you see the restore process begin, but not complete successfully, you might not have configured all of the required permissions that are needed by the restore process. You must contact your administrator for assistance and ask that person to update your policies to allow you to restore a table in Amazon Keyspaces.

In addition to user permissions, Amazon Keyspaces may require permissions to perform actions during the restore process on your principal's behalf. This is the case if the table is encrypted with a customer-managed key, or if you are using IAM policies that restrict incoming traffic. For example, if you are using condition keys in your IAM policy to restrict source traffic to specific endpoints or IP ranges, the restore operation fails. To allow Amazon Keyspaces to perform the table restore operation on your principal's behalf, you must add an `aws:ViaAWSService` global condition key in the IAM policy.

For more information about permissions to restore tables, see [the section called "Restore Permissions" \(p. 117\)](#).

## I'm not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amazon Keyspaces.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon Keyspaces. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

## I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

### Important

Do not provide your access keys to a third party, even to help [find your canonical user ID](#). By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see [Managing access keys](#) in the *IAM User Guide*.

## I'm an administrator and want to allow others to access Amazon Keyspaces

To allow others to access Amazon Keyspaces, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon Keyspaces.

To get started right away, see [Creating your first IAM delegated user and group](#) in the *IAM User Guide*.

## I want to allow people outside of my AWS account to access my Amazon Keyspaces resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon Keyspaces supports these features, see [How Amazon Keyspaces works with IAM](#) (p. 151).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.

## Using service-linked roles for Amazon Keyspaces

Amazon Keyspaces (for Apache Cassandra) uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to Amazon Keyspaces. Service-linked roles are predefined by Amazon Keyspaces and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon Keyspaces automatic scaling easier because you don't have to manually add the necessary permissions for Application Auto Scaling. Amazon Keyspaces defines the permissions of its service-linked roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

For information about other services that support service-linked roles, see [AWS Services That Work with IAM](#) and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

### Service-linked role permissions for Amazon Keyspaces

Amazon Keyspaces automatic scaling uses the service-linked role named **AWSServiceRoleForApplicationAutoScaling\_CassandraTable** to allow Application Auto Scaling to call Amazon Keyspaces and Amazon CloudWatch on your behalf.

The AWSServiceRoleForApplicationAutoScaling\_CassandraTable service-linked role trusts the following service to assume the role:

- `cassandra.application-autoscaling.amazonaws.com`

The role permissions policy allows Application Auto Scaling to complete the following actions on the specified Amazon Keyspaces resources:

- Action: `cassandra:Select` on the resource `arn:*:cassandra:*:*:/keyspace/system/table/*`
- Action: `cassandra:Select` on the resource `arn:*:cassandra:*:*:/keyspace/system_schema/table/*`
- Action: `cassandra:Select` on the resource `arn:*:cassandra:*:*:/keyspace/system_schema_mcs/table/*`
- Action: `cassandra:Alter` on the resource `arn:*:cassandra:*:*:""`

### Creating a service-linked role for Amazon Keyspaces

You don't need to manually create a service-linked role for Amazon Keyspaces automatic scaling. When you enable Amazon Keyspaces automatic scaling on a table with the AWS Management Console, the AWS CLI, or the AWS API, Application Auto Scaling creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you enable Amazon Keyspaces automatic scaling for a table, Application Auto Scaling creates the service-linked role for you again.

## Editing a service-linked role for Amazon Keyspaces

Amazon Keyspaces does not allow you to edit the `AWSServiceRoleForApplicationAutoScaling_CassandraTable` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a Service-Linked Role](#) in the *IAM User Guide*.

## Deleting a service-linked role for Amazon Keyspaces

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that isn't actively monitored or maintained. However, you must first disable automatic scaling on all tables in the account across all Regions before you can delete the service-linked role manually. To disable automatic scaling on Amazon Keyspaces tables, see [??? \(p. 80\)](#).

### Note

If Amazon Keyspaces automatic scaling is using the role when you try to modify the resources, then the deregistration might fail. If that happens, wait for a few minutes and try the operation again.

### To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the `AWSServiceRoleForApplicationAutoScaling_CassandraTable` service-linked role. For more information, see [Deleting a Service-Linked Role](#) in the *IAM User Guide*.

### Note

To delete the service-linked role used by Amazon Keyspaces automatic scaling, you must first disable automatic scaling on all tables in the account.

## Supported AWS Regions for Amazon Keyspaces service-linked roles

Amazon Keyspaces automatic scaling and the Application Auto Scaling service-linked role are available in all AWS Regions where Amazon Keyspaces is available. For more information, see [Service endpoints for Amazon Keyspaces](#).

# Logging and monitoring in Amazon Keyspaces (for Apache Cassandra)

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Keyspaces and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. However, before you start monitoring Amazon Keyspaces, you should create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?

- How often will you monitor these resources?
- What monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

To provide an overview, the AWS Management Console for Amazon Keyspaces offers a preconfigured dashboard showing the latency and errors aggregated across all tables in an account. The next step is to set up a CloudWatch dashboard for individual Amazon Keyspaces resources. Start by establishing a baseline for normal Amazon Keyspaces performance in your environment by measuring performance at various times and under different load conditions. As you monitor Amazon Keyspaces, store historical monitoring data so that you can compare it with current performance data, identify normal performance patterns and performance anomalies, and devise methods to address issues.

To establish a baseline, you should, at a minimum, monitor the following items:

- System errors, so that you can determine whether any requests resulted in an error.

#### Topics

- [Monitoring tools for Amazon Keyspaces \(p. 168\)](#)
- [Monitoring Amazon Keyspaces with Amazon CloudWatch \(p. 169\)](#)
- [Logging Amazon Keyspaces API calls with AWS CloudTrail \(p. 180\)](#)

## Monitoring tools for Amazon Keyspaces

AWS provides various tools that you can use to monitor Amazon Keyspaces. You can configure some of these tools to do the monitoring for you, while some of the tools require manual intervention. We recommend that you automate monitoring tasks as much as possible.

#### Topics

- [Automated monitoring tools for Amazon Keyspaces \(p. 168\)](#)
- [Manual monitoring tools for Amazon Keyspaces \(p. 168\)](#)

## Automated monitoring tools for Amazon Keyspaces

You can use the following automated monitoring tools to watch Amazon Keyspaces and report when something is wrong:

- **Amazon CloudWatch alarms** – Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Amazon EC2 Auto Scaling policy.

CloudWatch alarms do not invoke actions simply because they are in a particular state. The state must have changed and been maintained for a specified number of periods. For more information, see [Monitoring Amazon Keyspaces with Amazon CloudWatch \(p. 169\)](#).

## Manual monitoring tools for Amazon Keyspaces

Another important part of monitoring Amazon Keyspaces involves manually monitoring those items that the CloudWatch alarms don't cover. The Amazon Keyspaces, CloudWatch, Trusted Advisor, and other AWS Management Console dashboards provide an at-a-glance view of the state of your AWS environment.

- The CloudWatch home page shows the following:
  - Current alarms and status
  - Graphs of alarms and resources
  - Service health status

In addition, you can use CloudWatch to do the following:

- Create [customized dashboards](#) to monitor the services you care about
- Graph metric data to troubleshoot issues and discover trends
- Search and browse all your AWS resource metrics
- Create and edit alarms to be notified of problems

## Monitoring Amazon Keyspaces with Amazon CloudWatch

You can monitor Amazon Keyspaces using CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing.

You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the [Amazon CloudWatch User Guide](#).

### Note

To get started quickly with a preconfigured CloudWatch dashboard showing common metrics for Amazon Keyspaces, you can use an AWS CloudFormation template available from <https://github.com/aws-samples/amazon-keyspaces-cloudwatch-cloudformation-templates>.

### Topics

- [How do I use Amazon Keyspaces metrics? \(p. 169\)](#)
- [Amazon Keyspaces metrics and dimensions \(p. 170\)](#)
- [Creating CloudWatch alarms to monitor Amazon Keyspaces \(p. 180\)](#)

## How do I use Amazon Keyspaces metrics?

The metrics reported by Amazon Keyspaces provide information that you can analyze in different ways. The following list shows some common uses for the metrics. These are suggestions to get you started, not a comprehensive list. For more information about metrics and retention, see [Metrics](#).

How can I?	Relevant metrics
How can I determine if any system errors occurred?	You can monitor <code>SystemErrors</code> to determine whether any requests resulted in a server error code. Typically, this metric should be equal to zero. If it isn't, you might want to investigate.
How can I compare average provisioned read to consumed read capacity?	<p>To monitor average provisioned read capacity and consumed read capacity</p> <ol style="list-style-type: none"><li>1. Set the <b>Period</b> for <code>ConsumedReadCapacityUnits</code> and <code>ProvisionedReadCapacityUnits</code> to the interval you want to monitor.</li><li>2. Change the <b>Statistic</b> for <code>ConsumedReadCapacityUnits</code> from <code>Average</code> to <code>Sum</code>.</li></ol>

How can I?	Relevant metrics
	<ol style="list-style-type: none"> <li>3. Create a new empty <b>Math expression</b>.</li> <li>4. In the <b>Details</b> section of the new math expression, enter the <b>Id</b> of ConsumedReadCapacityUnits and divide the metric by the CloudWatch <b>PERIOD</b> function of the metric (metric_id/(PERIOD(metric_id))).</li> <li>5. Unselect ConsumedReadCapacityUnits.</li> </ol> <p>You can now compare your average consumed read capacity to your provisioned capacity. For more information on basic arithmetic functions and how to create a time series see <a href="#">Using metric math</a>.</p>
How can I compare average provisioned write to consumed write capacity?	<p>To monitor average provisioned write capacity and consumed write capacity</p> <ol style="list-style-type: none"> <li>1. Set the <b>Period</b> for ConsumedWriteCapacityUnits and ProvisionedWriteCapacityUnits to the interval you want to monitor.</li> <li>2. Change the <b>Statistic</b> for ConsumedWriteCapacityUnits from Average to Sum.</li> <li>3. Create a new empty <b>Math expression</b>.</li> <li>4. In the <b>Details</b> section of the new math expression, enter the <b>Id</b> of ConsumedWriteCapacityUnits and divide the metric by the CloudWatch <b>PERIOD</b> function of the metric (metric_id/(PERIOD(metric_id))).</li> <li>5. Unselect ConsumedWriteCapacityUnits.</li> </ol> <p>You can now compare your average consumed write capacity to your provisioned capacity. For more information on basic arithmetic functions and how to create a time series see <a href="#">Using metric math</a>.</p>

## Amazon Keyspaces metrics and dimensions

When you interact with Amazon Keyspaces, it sends the following metrics and dimensions to Amazon CloudWatch. All metrics are aggregated and reported every minute. You can use the following procedures to view the metrics for Amazon Keyspaces.

### To view metrics using the CloudWatch console

Metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace.

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, change the Region. On the navigation bar, choose the Region where your AWS resources reside. For more information, see [AWS service endpoints](#).
3. In the navigation pane, choose **Metrics**.
4. Under the **All metrics** tab, choose AWS/Cassandra.

### To view metrics using the AWS CLI

- At a command prompt, use the following command.



```
aws cloudwatch list-metrics --namespace "AWS/Cassandra"
```

## Amazon Keyspaces metrics and dimensions

The metrics and dimensions that Amazon Keyspaces sends to Amazon CloudWatch are listed here.

### Amazon Keyspaces metrics

Amazon CloudWatch aggregates Amazon Keyspaces metrics at one-minute intervals.

Not all statistics, such as Average or Sum, are applicable for every metric. However, all of these values are available through the Amazon Keyspaces console, or by using the CloudWatch console, AWS CLI, or AWS SDKs for all metrics. In the following table, each metric has a list of valid statistics that are applicable to that metric.

Metric	Description
AccountMaxTableLevelReads	<p>The maximum number of read capacity units that can be used by a table of an account. For on-demand tables this limit caps the maximum read request units a table can use.</p> <p>Units: Count</p> <p>Valid Statistics:</p> <ul style="list-style-type: none"><li>• <b>Maximum</b> – The maximum number of read capacity units that can be used by a table of the account.</li></ul>
AccountMaxTableLevelWrites	<p>The maximum number of write capacity units that can be used by a table of an account. For on-demand tables this limit caps the maximum write request units a table can use.</p> <p>Units: Count</p> <p>Valid Statistics:</p> <ul style="list-style-type: none"><li>• <b>Maximum</b> – The maximum number of write capacity units that can be used by a table of the account.</li></ul>
AccountProvisionedReadCapacityUtilization	<p>The percentage of provisioned read capacity units utilized by an account.</p> <p>Units: Percent</p> <p>Valid Statistics:</p> <ul style="list-style-type: none"><li>• <b>Maximum</b> – The maximum percentage of provisioned read capacity units utilized by the account.</li><li>• <b>Minimum</b> – The minimum percentage of provisioned read capacity units utilized by the account.</li><li>• <b>Average</b> – The average percentage of provisioned read capacity units utilized by the account. The metric is published for five-minute intervals. Therefore, if you rapidly adjust the provisioned read capacity units, this statistic might not reflect the true average.</li></ul>

Metric	Description
<p><code>AccountProvisionedWriteCapacityUnits</code></p>	<p>The percentage of provisioned write capacity units utilized by an account.</p> <p>Units: Percent</p> <p>Valid Statistics:</p> <ul style="list-style-type: none"> <li>• <b>Maximum</b> – The maximum percentage of provisioned write capacity units utilized by the account.</li> <li>• <b>Minimum</b> – The minimum percentage of provisioned write capacity units utilized by the account.</li> <li>• <b>Average</b> – The average percentage of provisioned write capacity units utilized by the account. The metric is published for five-minute intervals. Therefore, if you rapidly adjust the provisioned write capacity units, this statistic might not reflect the true average.</li> </ul>
<p><code>ConditionalCheckFailedRequests</code></p>	<p>The number of failed lightweight transaction (LWT) write requests. The <code>INSERT</code>, <code>UPDATE</code>, and <code>DELETE</code> operations let you provide a logical condition that must evaluate to true before the operation can proceed. If this condition evaluates to false, <code>ConditionalCheckFailedRequests</code> is incremented by one.</p> <p>Units: Count</p> <p>Dimensions: <code>Keyspace</code>, <code>TableName</code></p> <p>Valid Statistics:</p> <ul style="list-style-type: none"> <li>• <b>Minimum</b></li> <li>• <b>Maximum</b></li> <li>• <b>Average</b></li> <li>• <b>SampleCount</b></li> <li>• <b>Sum</b></li> </ul>

Metric	Description
ConsumedReadCapacityUnits	<p>The number of read capacity units consumed over the specified time period. For more information, see <a href="#">Read/Write capacity mode</a>.</p> <p><b>Note</b> To understand your average throughput utilization per second, use the <code>Sum</code> statistic to calculate the consumed throughput for the one minute period. Then divide the sum by the number of seconds in a minute (60) to calculate the average <code>ConsumedReadCapacityUnits</code> per second (recognizing that this average does not highlight any large but brief spikes in read activity that occurred during that minute). For more information on comparing average consumed read capacity to provisioned read capacity, see <a href="#">the section called "Using metrics"</a> (p. 169)</p> <p>Units: Count</p> <p>Dimensions: <code>Keyspace</code>, <code>TableName</code></p> <p>Valid Statistics:</p> <ul style="list-style-type: none"><li>• <b>Minimum</b> – The minimum number of read capacity units consumed by any individual request to the table.</li><li>• <b>Maximum</b> – The maximum number of read capacity units consumed by any individual request to the table.</li><li>• <b>Average</b> – The average per-request read capacity consumed.</li></ul> <p><b>Note</b> The <b>Average</b> value is influenced by periods of inactivity where the sample value will be zero.</p> <ul style="list-style-type: none"><li>• <b>Sum</b> – The total read capacity units consumed. This is the most useful statistic for the <code>ConsumedReadCapacityUnits</code> metric.</li><li>• <b>SampleCount</b> – The number of requests to Amazon Keyspaces, even if no read capacity was consumed.</li></ul> <p><b>Note</b> The <b>SampleCount</b> value is influenced by periods of inactivity where the sample value will be zero.</p>

Metric	Description
ConsumedWriteCapacityUnits	<p>The number of write capacity units consumed over the specified time period. You can retrieve the total consumed write capacity for a table. For more information, see <a href="#">Read/Write capacity mode</a>.</p> <p><b>Note</b> To understand your average throughput utilization per second, use the <code>Sum</code> statistic to calculate the consumed throughput for the one minute period. Then divide the sum by the number of seconds in a minute (60) to calculate the average <code>ConsumedWriteCapacityUnits</code> per second (recognizing that this average does not highlight any large but brief spikes in write activity that occurred during that minute). For more information on comparing average consumed write capacity to provisioned write capacity, see <a href="#">the section called "Using metrics"</a> (p. 169)</p> <p>Units: Count</p> <p>Dimensions: <code>Keyspace</code>, <code>TableName</code></p> <p>Valid Statistics:</p> <ul style="list-style-type: none"> <li>• <code>Minimum</code> – The minimum number of write capacity units consumed by any individual request to the table.</li> <li>• <code>Maximum</code> – The maximum number of write capacity units consumed by any individual request to the table.</li> <li>• <code>Average</code> – The average per-request write capacity consumed.</li> </ul> <p><b>Note</b> The <code>Average</code> value is influenced by periods of inactivity where the sample value will be zero.</p> <ul style="list-style-type: none"> <li>• <code>Sum</code> – The total write capacity units consumed. This is the most useful statistic for the <code>ConsumedWriteCapacityUnits</code> metric.</li> <li>• <code>SampleCount</code> – The number of requests to Amazon Keyspaces, even if no write capacity was consumed.</li> </ul> <p><b>Note</b> The <code>SampleCount</code> value is influenced by periods of inactivity where the sample value will be zero.</p>

Metric	Description
<code>MaxProvisionedTableReadCapacityUnits</code>	<p>The percentage of provisioned read capacity units utilized by the highest provisioned read table of an account.</p> <p>Units: Percent</p> <p>Valid Statistics:</p> <ul style="list-style-type: none"> <li>• <b>Maximum</b> – The maximum percentage of provisioned read capacity units utilized by the highest provisioned read table of the account.</li> <li>• <b>Minimum</b> – The minimum percentage of provisioned read capacity units utilized by the highest provisioned read table of the account.</li> <li>• <b>Average</b> – The average percentage of provisioned read capacity units utilized by the highest provisioned read table of the account. The metric is published for five-minute intervals. Therefore, if you rapidly adjust the provisioned read capacity units, this statistic might not reflect the true average.</li> </ul>
<code>MaxProvisionedTableWriteCapacityUnits</code>	<p>The percentage of provisioned write capacity utilized by the highest provisioned write table of an account.</p> <p>Units: Percent</p> <p>Valid Statistics:</p> <ul style="list-style-type: none"> <li>• <b>Maximum</b> – The maximum percentage of provisioned write capacity units utilized by the highest provisioned write table of the account.</li> <li>• <b>Minimum</b> – The minimum percentage of provisioned write capacity units utilized by the highest provisioned write table of the account.</li> <li>• <b>Average</b> – The average percentage of provisioned write capacity units utilized by the highest provisioned write table of the account. The metric is published for five-minute intervals. Therefore, if you rapidly adjust the provisioned write capacity units, this statistic might not reflect the true average.</li> </ul>
<code>PerConnectionRequestRateExceeded</code>	<p>Requests to Amazon Keyspaces that exceed the per-connection request rate quota. Each client connection to Amazon Keyspaces can support up to 3000 CQL requests per second. Clients can create multiple connections to increase throughput.</p> <p>Units: Count</p> <p>Dimensions: Keyspace, TableName, Operation</p> <p>Valid Statistics:</p> <ul style="list-style-type: none"> <li>• <b>SampleCount</b></li> <li>• <b>Sum</b></li> </ul>

Metric	Description
ProvisionedReadCapacityUnits	<p>The number of provisioned read capacity units for a table.</p> <p>The <code>TableName</code> dimension returns the <code>ProvisionedReadCapacityUnits</code> for the table.</p> <p>Units: Count</p> <p>Dimensions: <code>Keyspace</code>, <code>TableName</code></p> <p>Valid Statistics:</p> <ul style="list-style-type: none"><li>• <b>Minimum</b> – The lowest setting for provisioned read capacity. If you use <code>ALTER TABLE</code> to increase read capacity, this metric shows the lowest value of provisioned <code>ReadCapacityUnits</code> during this time period.</li><li>• <b>Maximum</b> – The highest setting for provisioned read capacity. If you use <code>ALTER TABLE</code> to decrease read capacity, this metric shows the highest value of provisioned <code>ReadCapacityUnits</code> during this time period.</li><li>• <b>Average</b> – The average provisioned read capacity. The <code>ProvisionedReadCapacityUnits</code> metric is published at five-minute intervals. Therefore, if you rapidly adjust the provisioned read capacity units, this statistic might not reflect the true average.</li></ul>
ProvisionedWriteCapacityUnits	<p>The number of provisioned write capacity units for a table.</p> <p>The <code>TableName</code> dimension returns the <code>ProvisionedWriteCapacityUnits</code> for the table.</p> <p>Units: Count</p> <p>Dimensions: <code>Keyspace</code>, <code>TableName</code></p> <p>Valid Statistics:</p> <ul style="list-style-type: none"><li>• <b>Minimum</b> – The lowest setting for provisioned write capacity. If you use <code>ALTER TABLE</code> to increase write capacity, this metric shows the lowest value of provisioned <code>WriteCapacityUnits</code> during this time period.</li><li>• <b>Maximum</b> – The highest setting for provisioned write capacity. If you use <code>ALTER TABLE</code> to decrease write capacity, this metric shows the highest value of provisioned <code>WriteCapacityUnits</code> during this time period.</li><li>• <b>Average</b> – The average provisioned write capacity. The <code>ProvisionedWriteCapacityUnits</code> metric is published at five-minute intervals. Therefore, if you rapidly adjust the provisioned write capacity units, this statistic might not reflect the true average.</li></ul>

Metric	Description
ReadThrottleEvents	<p>Requests to Amazon Keyspaces that exceed the provisioned read capacity for a table.</p> <p>Units: Count</p> <p>Dimensions: Keyspace, TableName, Operation</p> <p>Valid Statistics:</p> <ul style="list-style-type: none"><li>• SampleCount</li><li>• Sum</li></ul>
ReturnedItemCount	<p>The number of rows returned by multi-row <code>SELECT</code> queries during the specified time period. Multi-row <code>SELECT</code> queries are queries which do not contain the fully qualified primary key, such a full table scans and range queries.</p> <p>The number of rows <i>returned</i> is not necessarily the same as the number of rows that were evaluated. For example, suppose that you requested a <code>SELECT *</code> with <code>ALLOW FILTERING</code> on a table that had 100 rows, but specified a <code>WHERE</code> clause that narrowed the results so that only 15 rows were returned. In this case, the response from <code>SELECT</code> would contain a <code>ScanCount</code> of 100 and a <code>Count</code> of 15 returned rows.</p> <p>Units: Count</p> <p>Dimensions: Keyspace, TableName, Operation</p> <p>Valid Statistics:</p> <ul style="list-style-type: none"><li>• Minimum</li><li>• Maximum</li><li>• Average</li><li>• SampleCount</li><li>• Sum</li></ul>

Metric	Description
StoragePartitionThroughputCapacityExceeded	<p>Requests to an Amazon Keyspaces storage partition that exceed the throughput capacity of the partition. Amazon Keyspaces storage partitions can support up to 1000 WCU/WRU per second or 3000 RCU/RRU per second, or a linear combination of the two. We recommend reviewing your data model to distribute read/write traffic across more partitions to mitigate these exceptions.</p> <p><b>Note</b> Logical Amazon Keyspaces partitions can span multiple storage partitions and are virtually unbounded in size.</p> <p>Units: Count</p> <p>Dimensions: Keyspace, TableName, Operation</p> <p>Valid Statistics:</p> <ul style="list-style-type: none"> <li>• SampleCount</li> <li>• Sum</li> </ul>
SuccessfulRequestLatency	<p>The successful requests to Amazon Keyspaces during the specified time period. SuccessfulRequestLatency can provide two different kinds of information:</p> <ul style="list-style-type: none"> <li>• The elapsed time for successful requests (Minimum, Maximum, Sum, or Average).</li> <li>• The number of successful requests (SampleCount).</li> </ul> <p>SuccessfulRequestLatency reflects activity only within Amazon Keyspaces and does not take into account network latency or client-side activity.</p> <p>Units: Milliseconds</p> <p>Dimensions: Keyspace, TableName, Operation</p> <p>Valid Statistics:</p> <ul style="list-style-type: none"> <li>• Minimum</li> <li>• Maximum</li> <li>• Average</li> <li>• SampleCount</li> </ul>



Metric	Description
<code>SystemErrors</code>	<p>The requests to Amazon Keyspaces that generate a <code>ServerError</code> during the specified time period. A <code>ServerError</code> usually indicates an internal service error.</p> <p>Units: Count</p> <p>Dimensions: <code>Keyspace</code>, <code>TableName</code>, <code>Operation</code></p> <p>Valid Statistics:</p> <ul style="list-style-type: none"> <li>Sum</li> <li>SampleCount</li> </ul>
<code>UserErrors</code>	<p>Requests to Amazon Keyspaces that generate an <code>InvalidRequest</code> error during the specified time period. An <code>InvalidRequest</code> usually indicates a client-side error, such as an invalid combination of parameters, an attempt to update a nonexistent table, or an incorrect request signature.</p> <p><code>UserErrors</code> represents the aggregate of invalid requests for the current AWS Region and the current AWS account.</p> <p>Units: Count</p> <p>Dimensions: <code>Keyspace</code>, <code>TableName</code>, <code>Operation</code></p> <p>Valid Statistics:</p> <ul style="list-style-type: none"> <li>Sum</li> <li>SampleCount</li> </ul>
<code>WriteThrottleEvents</code>	<p>Requests to Amazon Keyspaces that exceed the provisioned write capacity for a table.</p> <p>Units: Count</p> <p>Dimensions: <code>Keyspace</code>, <code>TableName</code>, <code>Operation</code></p> <p>Valid Statistics:</p> <ul style="list-style-type: none"> <li>SampleCount</li> <li>Sum</li> </ul>

### Dimensions for Amazon Keyspaces metrics

The metrics for Amazon Keyspaces are qualified by the values for the account, table name, or operation. You can use the CloudWatch console to retrieve Amazon Keyspaces data along any of the dimensions in the following table.

Dimension	Description
<code>Keyspace</code>	This dimension limits the data to a specific keyspace. This value can be any keyspace in the current Region and the current AWS account.

Dimension	Description
Operation	This dimension limits the data to one of the Amazon Keyspaces CQL operations, such as <code>INSERT</code> or <code>SELECT</code> operations.
TableName	This dimension limits the data to a specific table. This value can be any table name in the current Region and the current AWS account. If the table name is not unique within the account, you must also specify Keyspace.

## Creating CloudWatch alarms to monitor Amazon Keyspaces

You can create an Amazon CloudWatch alarm for Amazon Keyspaces that sends an Amazon Simple Notification Service (Amazon SNS) message when the alarm changes state. An alarm watches a single metric over a time period that you specify. It performs one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon SNS topic or Auto Scaling policy.

Alarms invoke actions for sustained state changes only. CloudWatch alarms do not invoke actions simply because they are in a particular state. The state must have changed and been maintained for a specified number of periods.

For more information about creating CloudWatch alarms, see [Using Amazon CloudWatch alarms](#) in the *Amazon CloudWatch User Guide*.

## Logging Amazon Keyspaces API calls with AWS CloudTrail

Amazon Keyspaces is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon Keyspaces. CloudTrail captures Data Definition Language (DDL) API calls for Amazon Keyspaces as events. The calls that are captured include calls from the Amazon Keyspaces console and code calls to the Amazon Keyspaces API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon Simple Storage Service (Amazon S3) bucket, including events for Amazon Keyspaces.

If you don't configure a trail, you can still view the most recent events on the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Keyspaces, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

## Amazon Keyspaces information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon Keyspaces, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing events with CloudTrail event history](#).

For an ongoing record of events in your AWS account, including events for Amazon Keyspaces, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs.

For more information, see the following topics in the *AWS CloudTrail User Guide*:

- [Overview for creating a trail](#)
- [CloudTrail supported services and integrations](#)
- [Configuring Amazon SNS notifications for CloudTrail](#)
- [Receiving CloudTrail log files from multiple Regions](#)
- [Receiving CloudTrail log files from multiple accounts](#)

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials
- Whether the request was made with temporary security credentials for a role or federated user
- Whether the request was made by another AWS service

For more information, see the [CloudTrail userIdentity element](#).

## Understanding Amazon Keyspaces log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateKeyspace`, `DropKeyspace`, `CreateTable`, and `DropTable` actions:

```
{
  "Records": [
    {
      "eventVersion": "1.05",
      "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAIOSFODNN7EXAMPLE:alice",
        "arn": "arn:aws:sts::111122223333:assumed-role/users/alice",
        "accountId": "111122223333",
        "sessionContext": {
          "sessionIssuer": {
            "type": "Role",
            "principalId": "AKIAIOSFODNN7EXAMPLE",
            "arn": "arn:aws:iam::111122223333:role/Admin",
            "accountId": "111122223333",
            "userName": "Admin"
          },
          "webIdFederationData": {},
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2020-01-15T18:47:56Z"
          }
        }
      },
      "eventTime": "2020-01-15T18:53:04Z",
      "eventSource": "cassandra.amazonaws.com",
      "eventName": "CreateKeyspace",
```

```
"awsRegion": "us-east-2",
"sourceIPAddress": "52.95.24.72",
"userAgent": "Cassandra Client/ProtocolV4",
"requestParameters": {
  "rawQuery": "\n\tCREATE KEYSPACE \\"mykeyspace\\" \n\tWITH \n\t\tREPLICATION =
{'class': 'SingleRegionStrategy'} \n\t\t",
  "keyspaceName": "mykeyspace"
},
"responseElements": null,
"requestID": "bfa3e75d-bf4d-4fc0-be5e-89d15850eb41",
"eventID": "d25beae8-f611-4229-877a-921557a07bb9",
"readOnly": false,
"resources": [
  {
    "accountId": "111122223333",
    "type": "AWS::Cassandra::Keyspace",
    "ARN": "arn:aws:cassandra:us-east-2:111122223333:/keyspace/mykeyspace/"
  }
],
"eventType": "AwsApiCall",
"apiVersion": "3.4.4",
"recipientAccountId": "111122223333"
},
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AKIAIOSFODNN7EXAMPLE:alice",
    "arn": "arn:aws:sts:111122223333:assumed-role/users/alice",
    "accountId": "111122223333",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:role/Admin",
        "accountId": "111122223333",
        "userName": "Admin"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-01-15T18:47:56Z"
      }
    }
  },
  "eventTime": "2020-01-15T19:28:39Z",
  "eventSource": "cassandra.amazonaws.com",
  "eventName": "DropKeyspace",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "52.95.24.73",
  "userAgent": "Cassandra Client/ProtocolV4",
  "requestParameters": {
    "rawQuery": "DROP KEYSPACE \\"mykeyspace\\" ",
    "keyspaceName": "mykeyspace"
  },
  "responseElements": null,
  "requestID": "66f3d86a-56ae-4c29-b46f-abcd489ed86b",
  "eventID": "e5aebac-e1dd-41e3-a515-84fe6aaabd7b",
  "readOnly": false,
  "resources": [
    {
      "accountId": "111122223333",
      "type": "AWS::Cassandra::Keyspace",
      "ARN": "arn:aws:cassandra:us-east-2:111122223333:/keyspace/mykeyspace/"
    }
  ]
},
```

[illegible]

```
    "arn": "arn:aws:iam::111122223333:role/Admin",
    "accountId": "111122223333",
    "userName": "Admin"
  },
  "webIdFederationData": {},
  "attributes": {
    "mfaAuthenticated": "false",
    "creationDate": "2020-01-15T18:47:56Z"
  }
},
"eventTime": "2020-01-15T19:27:59Z",
"eventSource": "cassandra.amazonaws.com",
"eventName": "DropTable",
"awsRegion": "us-east-2",
"sourceIPAddress": "52.95.24.66",
"userAgent": "Cassandra Client/ProtocolV4",
"requestParameters": {
  "rawQuery": "DROP TABLE \"mykeyspace\".\"mytable\"",
  "keyspaceName": "mykeyspace",
  "tableName": "mytable"
},
"responseElements": null,
"requestID": "025501b0-3582-437e-9d18-8939e9ef262f",
"eventID": "1a5cbcdc-4e38-4889-8475-3eab98de0ffd",
"readOnly": false,
"resources": [
  {
    "accountId": "111122223333",
    "type": "AWS::Cassandra::Table",
    "ARN": "arn:aws:cassandra:us-east-2:111122223333:/keyspace/mykeyspace/table/
mytable"
  }
],
"eventType": "AwsApiCall",
"apiVersion": "3.4.4",
"recipientAccountId": "111122223333"
}
]
```

## Compliance validation for Amazon Keyspaces (for Apache Cassandra)

Third-party auditors assess the security and compliance of Amazon Keyspaces (for Apache Cassandra) as part of multiple AWS compliance programs. These include:

- ISO/IEC 27001:2013, 27017:2015, 27018:2019, and ISO/IEC 9001:2015. For more information, see [AWS ISO and CSA STAR certifications and services](#).
- System and Organization Controls (SOC)
- Payment Card Industry (PCI)
- Health Insurance Portability and Accountability Act (HIPAA)

To learn whether Amazon Keyspaces or other AWS services are in scope of specific compliance programs, see [AWS Services in Scope by Compliance Program](#). For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security and Compliance Quick Start Guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.

**Note**

Not all services are compliant with HIPAA.

- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [Evaluating Resources with Rules](#) in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.
- [AWS Audit Manager](#) – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

## Resilience and disaster recovery in Amazon Keyspaces

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

Amazon Keyspaces replicates data automatically three times in multiple AWS Availability Zones within the same AWS Region for durability and high availability.

For more information about AWS Regions and Availability Zones, see [AWS global infrastructure](#).

## Infrastructure security in Amazon Keyspaces

As a managed service, Amazon Keyspaces (for Apache Cassandra) is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of security processes](#) whitepaper.

You use AWS published API calls to access Amazon Keyspaces through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Amazon Keyspaces supports two methods of authenticating client requests. The first method uses service-specific credentials, which are password based credentials generated for a specific AWS Identity

and Access Management (IAM) user. You can create and manage the password using the IAM console, the AWS CLI, or the AWS API. For more information, see [Using IAM with Amazon Keyspaces](#).

The second method uses an authentication plugin for the open-source DataStax Java Driver for Cassandra. This plugin enables [IAM users, roles, and federated identities](#) to add authentication information to Amazon Keyspaces (for Apache Cassandra) API requests using the [AWS Signature Version 4 process \(SigV4\)](#). For more information, see [??? \(p. 15\)](#).

You can use an interface VPC endpoint to keep traffic between your Amazon VPC and Amazon Keyspaces from leaving the Amazon network. Interface VPC endpoints are powered by AWS PrivateLink, an AWS technology that enables private communication between AWS services using an elastic network interface with private IPs in your Amazon VPC. For more information, see [the section called “Using interface VPC endpoints” \(p. 186\)](#).

## Using Amazon Keyspaces with interface VPC endpoints

Interface VPC endpoints enable private communication between your virtual private cloud (VPC) running in Amazon VPC and Amazon Keyspaces. Interface VPC endpoints are powered by AWS PrivateLink, which is an AWS service that enables private communication between VPCs and AWS services.

AWS PrivateLink enables this by using an elastic network interface with private IPs in your VPC so that network traffic does not leave the Amazon network. Interface VPC endpoints don't require an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. For more information, see [Amazon Virtual Private Cloud](#) and [Interface VPC endpoints \(AWS PrivateLink\)](#).

### Using interface VPC endpoints for Amazon Keyspaces

You can create an interface VPC endpoint so that traffic between Amazon Keyspaces and your Amazon VPC resources starts flowing through the interface VPC endpoint. To get started, follow the steps to [create an interface endpoint](#). Next, edit the security group associated with the endpoint that you created in the previous step, and configure an inbound rule for port 9142. For more information, see [Adding, removing, and updating rules](#).

### Populating `system.peers` table entries with interface VPC endpoint information

Apache Cassandra drivers use the `system.peers` table to query for node information about the cluster. Cassandra drivers use the node information to load balance connections and retry operations. Amazon Keyspaces populates nine entries in the `system.peers` table automatically for clients connecting through the public endpoint.

To provide clients connecting through interface VPC endpoints with similar functionality, Amazon Keyspaces populates the `system.peers` table in your account with an entry for each availability zone where a VPC endpoint is available. To look up and store available interface VPC endpoints in the `system.peers` table, Amazon Keyspaces requires that you grant the IAM entity used to connect to Amazon Keyspaces access permissions to query your VPC for the endpoint and network interface information.

#### **Important**

Populating the `system.peers` table with your available interface VPC endpoints improves load balancing and increases read/write throughput. It is recommended for all clients accessing Amazon Keyspaces using interface VPC endpoints.

To grant the IAM entity used to connect to Amazon Keyspaces permissions to look up the necessary interface VPC endpoint information, you can update your existing IAM role or user policy, or create a new IAM policy as shown in the following example.



```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ListVPCEndpoints",
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcEndpoints"
      ],
      "Resource": "*"
    }
  ]
}
```

To confirm that the policy has been setup correctly, query the `system.peers` table to see networking information. If the `system.peers` table is empty, it could indicate that the policy hasn't been configured successfully or that you have exceeded the request rate quota for `DescribeNetworkInterfaces` and `DescribeVPCEndpoints` API actions. `DescribeVPCEndpoints` falls into the `Describe*` category and is considered a *non-mutating action*. `DescribeNetworkInterfaces` falls into the subset of *unfiltered and unpaginated non-mutating actions*, and different quotas apply. For more information, see [Request token bucket sizes and refill rates](#) in the Amazon EC2 API Reference.

If you do see an empty table, try again a few minutes later to rule out request rate quota issues. If your query returns results from the table, your policy has been configured correctly.

## Controlling access to interface VPC endpoints for Amazon Keyspaces

VPC endpoint policies enable you to control access to resources in two ways:

- **IAM policy** – You can control the requests, users, or groups that are allowed to access Amazon Keyspaces through a specific VPC endpoint. You can do this by using a [condition key](#) in the policy that is attached to an IAM user, group, or role.
- **VPC policy** – You can control which VPC endpoints have access to your Amazon Keyspaces resources by attaching policies to them. To restrict access to a specific keyspace or table to only allow traffic coming through a specific VPC endpoint, edit the existing IAM policy that restricts resource access and add that VPC endpoint.

The following are example endpoint policies for accessing Amazon Keyspaces resources.

- **IAM policy example: Restrict all access to a specific Amazon Keyspaces table unless traffic comes from the specified VPC endpoint** – This sample policy can be attached to an IAM user, role, or group. It restricts access to a specified Amazon Keyspaces table unless incoming traffic originates from a specified VPC endpoint.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "UserOrRolePolicyToDenyAccess",
      "Action": "cassandra:*",
      "Effect": "Deny",
      "Resource": [
        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/
        table/mytable",
```

```
        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
      ],
      "Condition": { "StringNotEquals" : { "aws:sourceVpce": "vpce-abc123" } }
    }
  ]
}
```

#### Note

To restrict access to a specific table, you must also include access to the system tables. System tables are read-only.

- **VPC policy example: Read-only access** – This sample policy can be attached to a VPC endpoint. (For more information, see [Controlling access to Amazon VPC resources](#)). It restricts actions to read-only access to Amazon Keyspaces resources through the VPC endpoint it's attached to.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "cassandra:Select"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- **VPC policy example: Restrict access to a specific Amazon Keyspaces table** – This sample policy can be attached to a VPC endpoint. It restricts access to a specific table through the VPC endpoint that it's attached to.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RestrictAccessToTable",
      "Principal": "*",
      "Action": "cassandra:*",
      "Effect": "Allow",
      "Resource": [
        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/
table/mytable",
        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
      ]
    }
  ]
}
```

#### Note

To restrict access to a specific table, you must also include access to the system tables. System tables are read-only.

## Availability

Amazon Keyspaces supports using interface VPC endpoints in all of the AWS Regions where the service is available. For more information, see [??? \(p. 17\)](#).

## VPC Endpoint policies and Amazon Keyspaces point-in-time recovery (PITR)

If you are using IAM policies with [condition keys](#) to restrict incoming traffic, the table restore operation may fail. For example, if you restrict source traffic to specific VPC endpoints using `aws:SourceVpce` condition keys, the table restore operation fails. To allow Amazon Keyspaces to perform a restore operation on your principal's behalf, you must add an `aws:ViaAWSService` condition key to your IAM policy. The `aws:ViaAWSService` condition key allows access when any AWS service makes a request using the principal's credentials. For more information, see [IAM JSON policy elements: Condition key](#) in the *IAM User Guide*. The following policy is an example of this.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CassandraAccessForVPCE",
      "Effect": "Allow",
      "Action": "cassandra:*",
      "Resource": "*",
      "Condition": {
        "Bool": {
          "aws:ViaAWSService": "false"
        },
        "StringEquals": {
          "aws:SourceVpce": [
            "vpce-12345678901234567"
          ]
        }
      }
    },
    {
      "Sid": "CassandraAccessForAwsService",
      "Effect": "Allow",
      "Action": "cassandra:*",
      "Resource": "*",
      "Condition": {
        "Bool": {
          "aws:ViaAWSService": "true"
        }
      }
    }
  ]
}
```

## Configuration and vulnerability analysis for Amazon Keyspaces

AWS handles basic security tasks like guest operating system (OS) and database patching, firewall configuration, and disaster recovery. These procedures have been reviewed and certified by the appropriate third parties. For more details, see the following resources:

- [Shared responsibility model](#)
- [Amazon Web Services: Overview of security processes](#)(whitepaper)

# Security best practices for Amazon Keyspaces

Amazon Keyspaces (for Apache Cassandra) provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

## Topics

- [Preventative security best practices for Amazon Keyspaces \(p. 190\)](#)
- [Detective security best practices for Amazon Keyspaces \(p. 191\)](#)

## Preventative security best practices for Amazon Keyspaces

The following security best practices are considered preventative because they can help you anticipate and prevent security incidents in Amazon Keyspaces.

### Use encryption at rest

Amazon Keyspaces encrypts at rest all user data that's stored in tables by using encryption keys stored in [AWS Key Management Service \(AWS KMS\)](#). This provides an additional layer of data protection by securing your data from unauthorized access to the underlying storage.

By default, Amazon Keyspaces uses an AWS owned key for encrypting all of your tables. If this key doesn't exist, it's created for you. Service default keys can't be disabled.

Alternatively, you can use a [customer managed key](#) for encryption at rest. For more information, see [Amazon Keyspaces Encryption at Rest](#).

### Use IAM roles to authenticate access to Amazon Keyspaces

For users, applications, and other AWS services to access Amazon Keyspaces, they must include valid AWS credentials in their AWS API requests. You should not store AWS credentials directly in the application or EC2 instance. These are long-term credentials that are not automatically rotated, and therefore could have significant business impact if they are compromised. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources.

For more information, see [IAM Roles](#).

### Use IAM policies for Amazon Keyspaces base authorization

When granting permissions, you decide who is getting them, which Amazon Keyspaces APIs they are getting permissions for, and the specific actions you want to allow on those resources. Implementing least privilege is key in reducing security risks and the impact that can result from errors or malicious intent.

Attach permissions policies to IAM identities (that is, users, groups, and roles) and thereby grant permissions to perform operations on Amazon Keyspaces resources.

You can do this by using the following:

- [AWS managed \(predefined\) policies](#)
- [Customer managed policies](#)

### Use IAM policy conditions for fine-grained access control

When you grant permissions in Amazon Keyspaces, you can specify conditions that determine how a permissions policy takes effect. Implementing least privilege is key in reducing security risks and the impact that can result from errors or malicious intent.

You can specify conditions when granting permissions using an IAM policy. For example, you can do the following:

- Grant permissions to allow users read-only access to specific keyspaces or tables.
- Grant permissions to allow a user write access to a certain table, based upon the identity of that user.

For more information, see [Identity-Based Policy Examples](#).

### **Consider client-side encryption**

If you store sensitive or confidential data in Amazon Keyspaces, you might want to encrypt that data as close as possible to its origin so that your data is protected throughout its lifecycle. Encrypting your sensitive data in transit and at rest helps ensure that your plaintext data isn't available to any third party.

## Detective security best practices for Amazon Keyspaces

The following security best practices are considered detective because they can help you detect potential security weaknesses and incidents.

### **Use AWS CloudTrail to monitor AWS Key Management Service (AWS KMS) AWS KMS key usage**

If you're using a [customer managed AWS KMS key](#) for encryption at rest, usage of this key is logged into AWS CloudTrail. CloudTrail provides visibility into user activity by recording actions taken on your account. CloudTrail records important information about each action, including who made the request, the services used, the actions performed, parameters for the actions, and the response elements returned by the AWS service. This information helps you track changes made to your AWS resources and troubleshoot operational issues. CloudTrail makes it easier to ensure compliance with internal policies and regulatory standards.

You can use CloudTrail to audit key usage. CloudTrail creates log files that contain a history of AWS API calls and related events for your account. These log files include all AWS KMS API requests that were made using the console, AWS SDKs, and command line tools, in addition to those made through integrated AWS services. You can use these log files to get information about when the AWS KMS key was used, the operation that was requested, the identity of the requester, the IP address that the request came from, and so on. For more information, see [Logging AWS Key Management Service API Calls with AWS CloudTrail](#) and the [AWS CloudTrail User Guide](#).

### **Use CloudTrail to monitor Amazon Keyspaces data definition language (DDL) operations**

CloudTrail provides visibility into user activity by recording actions taken on your account. CloudTrail records important information about each action, including who made the request, the services used, the actions performed, parameters for the actions, and the response elements returned by the AWS service. This information helps you to track changes made to your AWS resources and to troubleshoot operational issues. CloudTrail makes it easier to ensure compliance with internal policies and regulatory standards.

All Amazon Keyspaces [DDL operations \(p. 197\)](#) are logged in CloudTrail automatically. DDL operations let you create and manage Amazon Keyspaces keyspaces and tables.

When activity occurs in Amazon Keyspaces, that activity is recorded in a CloudTrail event along with other AWS service events in the event history. For more information, see [Logging Amazon Keyspaces operations by using AWS CloudTrail](#). You can view, search, and download recent events in your AWS account. For more information, see [Viewing events with CloudTrail event history](#) in the *AWS CloudTrail User Guide*.

For an ongoing record of events in your AWS account, including events for Amazon Keyspaces, create a [trail](#). A trail enables CloudTrail to deliver log files to an Amazon Simple Storage Service (Amazon S3) bucket. By default, when you create a trail on the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs.

#### **Tag your Amazon Keyspaces resources for identification and automation**

You can assign metadata to your AWS resources in the form of tags. Each tag is a simple label that consists of a customer-defined key and an optional value that can make it easier to manage, search for, and filter resources.

Tagging allows for grouped controls to be implemented. Although there are no inherent types of tags, they enable you to categorize resources by purpose, owner, environment, or other criteria. The following are some examples:

- Access – Used to control access to Amazon Keyspaces resources based on tags. For more information, see [the section called “Authorization based on Amazon Keyspaces tags” \(p. 153\)](#).
- Security – Used to determine requirements such as data protection settings.
- Confidentiality – An identifier for the specific data-confidentiality level that a resource supports.
- Environment – Used to distinguish between development, test, and production infrastructure.

For more information, see [AWS tagging strategies](#) and [Adding tags and labels to resources](#).

# CQL language reference for Amazon Keyspaces (for Apache Cassandra)

After you connect to an Amazon Keyspaces (for Apache Cassandra) endpoint, you use Cassandra Query Language (CQL) to work with your database. CQL is similar in many ways to Structured Query Language (SQL).

## Topics

- [Cassandra Query Language \(CQL\) elements in Amazon Keyspaces \(p. 193\)](#)
- [DDL statements \(data definition language\) in Amazon Keyspaces \(p. 197\)](#)
- [DML statements \(data manipulation language\) in Amazon Keyspaces \(p. 204\)](#)
- [Built-in functions in Amazon Keyspaces \(p. 207\)](#)

## Cassandra Query Language (CQL) elements in Amazon Keyspaces

Learn about the Cassandra Query Language (CQL) elements that are supported by Amazon Keyspaces, including identifiers, constants, terms, and data types.

## Topics

- [Identifiers \(p. 193\)](#)
- [Constants \(p. 193\)](#)
- [Terms \(p. 194\)](#)
- [Data types \(p. 194\)](#)
- [JSON encoding of Amazon Keyspaces data types \(p. 196\)](#)

## Identifiers

Identifiers (or names) are used to identify tables, columns, and other objects. An identifier can be quoted or not quoted. The following applies.

```
identifier      ::= unquoted_identifier | quoted_identifier
unquoted_identifier ::= re('[a-zA-Z][a-zA-Z0-9_]*')
quoted_identifier ::= '"' (any character where " can appear if doubled)+ '"'
```

## Constants

The following constants are defined.

```
constant ::= string | integer | float | boolean | uuid | blob | NULL
string    ::= '\'' (any character where ' can appear if doubled)+ '\''
```

```
integer ::= '$$' (any character other than '$$') '$$'
float   ::= re('-?[0-9]+' | re('-?[0-9](\.[0-9]*)?([eE][+-]?[0-9+])?') | NAN | INFINITY
boolean ::= TRUE | FALSE
uuid    ::= hex{8}-hex{4}-hex{4}-hex{4}-hex{12}
hex     ::= re("[0-9a-fA-F]")
blob    ::= '0' ('x' | 'X') hex+
```

## Terms

A term denotes the kind of values that are supported. Terms are defined by the following.

```
term      ::= constant | literal | function_call | arithmetic_operation |
             type_hint | bind_marker
literal   ::= collection_literal | tuple_literal
function_call ::= identifier '(' [ term (',' term)* ] ') '
arithmetic_operation ::= '-' term | term ('+' | '-' | '*' | '/' | '%') term
```

## Data types

Amazon Keyspaces supports the following data types:

### String types

Data type	Description
ascii	Represents an ASCII character string.
text	Represents a UTF-8 encoded string.
varchar	Represents a UTF-8 encoded string (varchar is an alias for text).

### Numeric types

Data type	Description
bigint	Represents a 64-bit signed long.
counter	Represents a 64-bit signed integer counter. For more information, see <a href="#">the section called “Counters” (p. 195)</a> .
decimal	Represents a variable-precision decimal.
double	Represents a 64-bit IEEE 754 floating point.
float	Represents a 32-bit IEEE 754 floating point.
int	Represents a 32-bit signed int.
varint	Represents an integer of arbitrary precision.



## Counters

A `counter` column contains a 64-bit signed integer. The counter value is incremented or decremented using the [the section called “UPDATE” \(p. 206\)](#) statement, and it cannot be set directly. This makes `counter` columns useful for tracking counts. For example, you can use counters to track the number of entries in a log file or the number of times a post has been viewed on a social network. The following restrictions apply to `counter` columns:

- A column of type `counter` cannot be part of the `primary key` of a table.
- In a table that contains one or more columns of type `counter`, all columns in that table must be of type `counter`.

In cases where a counter update fails, for example, due to timeouts or loss of connection with Amazon Keyspaces, the client doesn't know whether the counter value was updated. If the update is retried, the update to the counter value might get applied a second time.

## Blob type

Data type	Description
<code>blob</code>	Represents arbitrary bytes.

## Boolean type

Data type	Description
<code>boolean</code>	Represents <code>true</code> or <code>false</code> .

## Time-related types

Data type	Description
<code>timestamp</code>	Represents a timestamp.
<code>timeuuid</code>	Represents a <a href="#">version 1 UUID</a> .

## Collection types

Data type	Description
<code>list</code>	Represents an ordered collection of literal elements.
<code>map</code>	Represents an unordered collection of key-value pairs.
<code>set</code>	Represents an unordered collection of one or more literal elements.
<code>tuple</code>	Represents a bounded group of literal elements.

## Other types

Data type	Description
inet	A string representing an IP address, in either IPv4 or IPv6 format.

## JSON encoding of Amazon Keyspaces data types

Amazon Keyspaces offers the same JSON data type mappings as Apache Cassandra. The following table describes the data types Amazon Keyspaces accepts in `INSERT` JSON statements and the data types Amazon Keyspaces uses when returning data with the `SELECT` JSON statement.

For single-field data types such as `float`, `int`, `UUID`, and `date`, you also can insert data as a `string`. For compound data types and collections, such as `tuple`, `map`, and `list`, you can also insert data as JSON or as an encoded JSON `string`.

JSON data type	Data types accepted in <code>INSERT</code> JSON statements	Data types returned in <code>SELECT</code> JSON statements	Notes
<code>ascii</code>	<code>string</code>	<code>string</code>	Uses JSON character escape <code>\u</code> .
<code>bigint</code>	<code>integer</code> , <code>string</code>	<code>integer</code>	String must be a valid 64-bit integer.
<code>blob</code>	<code>string</code>	<code>string</code>	String should begin with <code>0x</code> followed by an even number of hex digits.
<code>boolean</code>	<code>boolean</code> , <code>string</code>	<code>boolean</code>	String must be either <code>true</code> or <code>false</code> .
<code>date</code>	<code>string</code>	<code>string</code>	Date in format <code>YYYY-MM-DD</code> , timezone UTC.
<code>decimal</code>	<code>integer</code> , <code>float</code> , <code>string</code>	<code>float</code>	May exceed 32-bit or 64-bit IEEE-754 floating point precision in client-side decoder.
<code>double</code>	<code>integer</code> , <code>float</code> , <code>string</code>	<code>float</code>	String must be a valid integer or float.
<code>float</code>	<code>integer</code> , <code>float</code> , <code>string</code>	<code>float</code>	String must be a valid integer or float.
<code>inet</code>	<code>string</code>	<code>string</code>	IPv4 or IPv6 address.
<code>int</code>	<code>integer</code> , <code>string</code>	<code>integer</code>	String must be a valid 32-bit integer.
<code>list</code>	<code>list</code> , <code>string</code>	<code>list</code>	Uses the native JSON list representation.

JSON data type	Data types accepted in <code>INSERT JSON</code> statements	Data types returned in <code>SELECT JSON</code> statements	Notes
<code>map</code>	<code>map, string</code>	<code>map</code>	Uses the native JSON map representation.
<code>smallint</code>	<code>integer, string</code>	<code>integer</code>	String must be a valid 16-bit integer.
<code>set</code>	<code>list, string</code>	<code>list</code>	Uses the native JSON list representation.
<code>text</code>	<code>string</code>	<code>string</code>	Uses JSON character escape <code>\u</code> .
<code>time</code>	<code>string</code>	<code>string</code>	Time of day in format <code>HH-MM-SS[.ffffffffff]</code> .
<code>timestamp</code>	<code>integer, string</code>	<code>string</code>	A timestamp. String constants allow to store timestamps as dates. Date stamps with format <code>YYYY-MM-DD HH:MM:SS.SSS</code> are returned.
<code>timeuuid</code>	<code>string</code>	<code>string</code>	Type 1 UUID. See <a href="#">constants (p. 193)</a> for the UUID format.
<code>tinyint</code>	<code>integer, string</code>	<code>integer</code>	String must be a valid 8-bit integer.
<code>tuple</code>	<code>list, string</code>	<code>list</code>	Uses the native JSON list representation.
<code>uuid</code>	<code>string</code>	<code>string</code>	See <a href="#">constants (p. 193)</a> for the UUID format.
<code>varchar</code>	<code>string</code>	<code>string</code>	Uses JSON character escape <code>\u</code> .
<code>varint</code>	<code>integer, string</code>	<code>integer</code>	Variable length; might overflow 32-bit or 64-bit integers in client-side decoder.

## DDL statements (data definition language) in Amazon Keyspaces

*Data definition language* (DDL) is the set of Cassandra Query Language (CQL) statements that you use to manage data structures in Amazon Keyspaces (for Apache Cassandra), such as keyspaces and tables. You use DDL to create these data structures, modify them after they are created, and remove them when they're no longer in use. Amazon Keyspaces performs DDL operations asynchronously. For more

information about how to confirm that an asynchronous operation has completed, see [the section called “Asynchronous creation and deletion of keyspaces and tables”](#) (p. 7).

The following DDL statements are supported:

- [CREATE KEYSPACE](#) (p. 198)
- [ALTER KEYSPACE](#) (p. 199)
- [DROP KEYSPACE](#) (p. 199)
- [CREATE TABLE](#) (p. 200)
- [ALTER TABLE](#) (p. 202)
- [RESTORE TABLE](#) (p. 203)
- [DROP TABLE](#) (p. 204)

### Topics

- [Keyspaces](#) (p. 198)
- [Tables](#) (p. 199)

## Keyspaces

A *keyspace* groups related tables that are relevant for one or more applications. In terms of a relational database management system (RDBMS), keyspaces are roughly similar to databases, tablespaces, or similar constructs.

### Note

In Apache Cassandra, keyspaces determine how data is replicated among multiple storage nodes. However, Amazon Keyspaces is a fully managed service: The details of its storage layer are managed on your behalf. For this reason, keyspaces in Amazon Keyspaces are logical constructs only, and aren't related to the underlying physical storage.

For information about quota limits and constraints for Amazon Keyspaces keyspaces, see [Quotas](#) (p. 209).

The following is the only available replication strategy for keyspaces:

SingleRegionStrategy – Replicates data across three Availability Zones in its Region.

## CREATE KEYSPACE

Use the `CREATE KEYSPACE` statement to create a new keyspace.

### Syntax

```
create_keyspace_statement ::=  
    CREATE KEYSPACE [ IF NOT EXISTS ] keyspace_name  
    WITH options
```

Where:

- *keyspace\_name* is the name of the keyspace to be created.
- *options* are one or more of the following:
  - `REPLICATION` – A map indicating the replication strategy for the keyspace (SingleRegionStrategy), with additional values as necessary. (Required)
  - `DURABLE_WRITES` – Writes to Amazon Keyspaces are always durable, so this option isn't required. However, if specified, the value must be `true`.
  - `TAGS` – A list of key-value pair tags to be attached to the resource on creation.

### Example

Create a keyspace as follows.

```
CREATE KEYSPACE "myGSGKeyspace"  
  WITH REPLICATION = {'class': 'SingleRegionStrategy'} and TAGS ={'key1':'val1',  
  'key2':'val2'} ;
```

## ALTER KEYSPACE

Use the ALTER KEYSPACE to add or remove tags from a keyspace.

### Syntax

```
alter_keyspace_statement ::=  
  ALTER KEYSPACE keyspace_name  
  [[ADD | DROP] TAGS
```

Where:

- *keyspace\_name* is the name of the keyspace to be altered.
- TAGS – A list of key-value pair tags to be added or removed from the keyspace.

### Example

Alter a keyspace as follows.

```
ALTER KEYSPACE "myGSGKeyspace" ADD TAGS {'key1':'val1', 'key2':'val2'};
```

## DROP KEYSPACE

Use the DROP KEYSPACE statement to remove a keyspace—including all of its contents, such as tables.

### Syntax

```
drop_keyspace_statement ::=  
  DROP KEYSPACE [ IF EXISTS ] keyspace_name
```

Where:

- *keyspace\_name* is the name of the keyspace to be dropped.

### Example

```
DROP KEYSPACE "myGSGKeyspace";
```

## Tables

*Tables* are the primary data structures in Amazon Keyspaces. Data in a table is organized into rows and columns. A subset of those columns is used to determine partitioning (and ultimately data placement) through the specification of a partition key.

Another set of columns can be defined into clustering columns, which means that they can participate as predicates in query execution.

By default, new tables are created with *on-demand* throughput capacity. You can change the capacity mode for new and existing tables. For more information about read/write capacity throughput modes, see [the section called “Read/write capacity modes” \(p. 72\)](#).

For information about quota limits and constraints for Amazon Keyspaces tables, see [Quotas \(p. 209\)](#).

## CREATE TABLE

Use the `CREATE TABLE` statement to create a new table.

### Syntax

```
create_table_statement ::= CREATE TABLE [ IF NOT EXISTS ] table_name
    '('
        column_definition
        ( ',' column_definition )*
        [ ',' PRIMARY KEY '(' primary_key ')' ]
    ')' [ WITH table_options ]

column_definition      ::= column_name cql_type [ STATIC ][ PRIMARY KEY]

primary_key            ::= partition_key [ ',' clustering_columns ]

partition_key          ::= column_name
                        | '(' column_name ( ',' column_name )* ')'

clustering_columns     ::= column_name ( ',' column_name )*

table_options          ::= [table_options]
                        | CLUSTERING ORDER BY '(' clustering_order
                        | options
                        | CUSTOM_PROPERTIES
                        | WITH TAGS

clustering_order       ::= column_name (ASC | DESC) ( ',' column_name (ASC | DESC) )*
```

Where:

- *table\_name* is the name of the table to be created.
- *column\_definition* consists of the following:
  - *column\_name* – The name of the column.
  - *cql\_type* – An Amazon Keyspaces data type (see [Data types \(p. 194\)](#)).
  - *STATIC* – Designates this column as static. Static columns store values that are shared by all rows in the same partition.
  - *PRIMARY KEY* – Designates this column as the table's primary key.
- *primary\_key* consists of the following:
  - *partition\_key*
  - *clustering\_columns*
- *partition\_key*:
  - The partition key can be a single column, or it can be a compound value composed of two or more columns. The partition key portion of the primary key is required and determines how Amazon Keyspaces stores your data.
- *clustering\_columns*:
  - The optional clustering column portion of your primary key determines how the data is clustered and sorted within each partition.
- *table\_options* consist of the following:

- *CLUSTERING ORDER BY* – The default CLUSTERING ORDER on a table is composed of your clustering keys in the ASC (ascending) sort direction. Specify it to override the default sort behavior.
- *CUSTOM\_PROPERTIES* – A map of settings that are specific to Amazon Keyspaces.
  - *capacity\_mode*: Specifies the read/write throughput capacity mode for the table. The options are *throughput\_mode:PAY\_PER\_REQUEST* and *throughput\_mode:PROVISIONED*. The provisioned capacity mode requires *read\_capacity\_units* and *write\_capacity\_units* as inputs. The default is *throughput\_mode:PAY\_PER\_REQUEST*.
  - *encryption\_specification*: Specifies the encryption options for encryption at rest. If it's not specified, the default is *encryption\_type:AWS\_OWNED\_KMS\_KEY*. The encryption option customer managed key requires the AWS KMS key in Amazon Resource Name (ARN) format as input: *kms\_key\_identifier:ARN:kms\_key\_identifier:ARN*.
  - *point\_in\_time\_recovery*: Specifies if point-in-time restore is enabled or disabled for the table. The options are *status:enabled* and *status:disabled*. If it's not specified, the default is *status:disabled*.
- *TAGS*: A list of key-value pair tags to be attached to the resource when it's created.
- *clustering\_order* consists of the following:
  - *column\_name* – The name of the column.
  - *ASC | DESC* – Sets the ascendant (ASC) or descendant (DESC) order modifier. If it's not specified, the default order is ASC.

### Example

```
CREATE TABLE IF NOT EXISTS "my_keyspace".my_table (
    id text,
    name text,
    region text,
    division text,
    project text,
    role text,
    pay_scale int,
    vacation_hrs float,
    manager_id text,
    PRIMARY KEY (id,division))
WITH CUSTOM_PROPERTIES={
    'capacity_mode':{
        'throughput_mode': 'PROVISIONED',
        'read_capacity_units': 10, 'write_capacity_units': 20
    },
    'point_in_time_recovery':{'status':
        'enabled'},
    'encryption_specification':{
        'encryption_type':
        'CUSTOMER_MANAGED_KMS_KEY',
        'kms_key_identifier':'arn:aws:kms:eu-
west-1:555555555555:key/11111111-1111-1111-1111-111111111111'
    }
}
AND CLUSTERING ORDER BY (division ASC)
AND TAGS={'key1':'val1', 'key2':'val2'};
```

In a table that uses clustering columns, non-clustering columns can be declared as static in the table definition. For more information about static columns, see [the section called “Static columns” \(p. 88\)](#).

### Example

```
CREATE TABLE "my_keyspace".my_table (
```

```
id int,  
name text,  
region text,  
division text,  
project text STATIC,  
PRIMARY KEY (id,division));
```

## ALTER TABLE

Use the `ALTER TABLE` statement to add new columns, add tags, or change the table's custom properties.

### Syntax

```
alter_table_statement ::= ALTER TABLE table_name  
  
    [ ADD ( column_definition / column_definition_list) ]  
    [[ADD | DROP] TAGS {'key1':'val1', 'key2':'val2'}]  
    [ WITH table_options [ , ... ] ] ;  
  
column_definition      ::= column_name cql_type
```

Where:

- *table\_name* is the name of the table to be altered.
- *column\_definition* is the name of the column and data type to be added.
- *column\_definition\_list* is a comma-separated list of columns placed inside parentheses.
- *TAGS* is a list of key-value pair tags to be attached to the resource.
- *table\_options* consist of the following:
  - *CUSTOM\_PROPERTIES* – A map of settings specific to Amazon Keyspaces.
    - *capacity\_mode*: Specifies the read/write throughput capacity mode for the table. The options are *throughput\_mode:PAY\_PER\_REQUEST* and *throughput\_mode:PROVISIONED*. The provisioned capacity mode requires *read\_capacity\_units* and *write\_capacity\_units* as inputs. The default is *throughput\_mode:PAY\_PER\_REQUEST*.
    - *encryption\_specification*: Specifies the encryption option for encryption at rest. The options are *encryption\_type:AWS\_OWNED\_KMS\_KEY* and *encryption\_type:CUSTOMER\_MANAGED\_KMS\_KEY*. The encryption option customer managed key requires the AWS KMS key in Amazon Resource Name (ARN) format as input: *kms\_key\_identifier:ARN*.
    - *point\_in\_time\_recovery*: Specifies if point-in-time restore is enabled or disabled for the table. The options are *status:enabled* and *status:disabled*. The default is *status:disabled*.

### Note

With `ALTER TABLE`, you can either add columns, add tags, or change a custom property. You cannot combine more than one `ALTER TABLE` command in the same statement.

### Examples

```
ALTER TABLE "myGSGKeyspace".employees_tbl ADD (first_name text);
```



```

or

ALTER TABLE mykeyspace.mytable WITH CUSTOM_PROPERTIES={ 'capacity_mode':{ 'throughput_mode':
'PROVISIONED', 'read_capacity_units': 10, 'write_capacity_units': 20}} ;

or

ALTER TABLE mykeyspace.mytable WITH CUSTOM_PROPERTIES={
    'encryption_specification':{
        'encryption_type': 'CUSTOMER_MANAGED_KMS_KEY',
        'kms_key_identifier': 'arn:aws:kms:eu-
west-1:555555555555:key/11111111-1111-111-1111-111111111111'
    }
}

or

ALTER TABLE mykeyspace.mytable WITH CUSTOM_PROPERTIES={ 'point_in_time_recovery': { 'status':
'enabled'}};;

```

## RESTORE TABLE

Use the `RESTORE TABLE` statement to restore a table to a point in time. This statement requires point-in-time recovery to be enabled on a table. For more information, see [Point-in-time recovery \(p. 115\)](#).

### Syntax

```

restore_table_statement ::=
    RESTORE TABLE table_name FROM TABLE table_name
    [ WITH table_options [ , ... ] ] ;

```

Where:

- *restored\_table\_name* is the name of the restored table.
- *source\_table\_name* is the name of the source table.
- *table\_options* consists of the following:
  - *restore\_timestamp* is the restore point time in ISO 8601 format. If it's not specified, the current timestamp is used.
  - *CUSTOM\_PROPERTIES* – A map of settings specific to Amazon Keyspaces.
    - *capacity\_mode*: Specifies the read/write throughput capacity mode for the table. The options are `throughput_mode:PAY_PER_REQUEST` and `throughput_mode:PROVISIONED`. The provisioned capacity mode requires *read\_capacity\_units* and *write\_capacity\_units* as inputs. The default is the current setting from the source table.
    - *encryption\_specification*: Specifies the encryption option for encryption at rest. The options are `encryption_type:AWS_OWNED_KMS_KEY` and `encryption_type:CUSTOMER_MANAGED_KMS_KEY`. The encryption option customer managed key requires the AWS KMS key in Amazon Resource Name (ARN) format as input: *kms\_key\_identifier*:ARN. To restore a table encrypted with a customer managed key to a table encrypted with an AWS owned key, Amazon Keyspaces requires access to the AWS KMS key of the source table.
    - *point\_in\_time\_recovery*: Specifies if point-in-time restore is enabled or disabled for the table. The options are `status:enabled` and `status:disabled`. Unlike when you create new tables, the default status for restored tables is `status:enabled` because the setting is inherited from the source table. To disable PITR for restored tables, you must set `status:disabled` explicitly.
  - *TAGS* is a list of key-value pair tags to be attached to the resource.

**Note**

Deleted tables can only be restored to the time of deletion.

**Example**

```
RESTORE TABLE mykeyspace.mytable_restored from table mykeyspace.my_table
WITH restore_timestamp = '2020-06-30T04:05:00+0000'
AND custom_properties = {'point_in_time_recovery':{'status':'disabled'}, 'capacity_mode':
{'throughput_mode': 'PROVISIONED', 'read_capacity_units': 10, 'write_capacity_units': 20}}
AND TAGS={'key1':'val1', 'key2':'val2'};
```

## DROP TABLE

Use the `DROP TABLE` statement to remove a table from the keyspace.

**Syntax**

```
drop_table_statement ::=
    DROP TABLE [ IF EXISTS ] table_name
```

Where:

- `IF EXISTS` prevents `DROP TABLE` from failing if the table doesn't exist. (Optional)
- `table_name` is the name of the table to be dropped.

**Example**

```
DROP TABLE "myGSGKeyspace".employees_tbl;
```

# DML statements (data manipulation language) in Amazon Keyspaces

*Data manipulation language* (DML) is the set of CQL statements that you use to manage data in Amazon Keyspaces (for Apache Cassandra) tables. You use DML statements to add, modify, or delete data in a table.

You also use DML statements to query data in a table. (Note that CQL doesn't support joins or subqueries.)

**Topics**

- [SELECT](#) (p. 204)
- [INSERT](#) (p. 205)
- [UPDATE](#) (p. 206)
- [DELETE](#) (p. 207)

## SELECT

Use a `SELECT` statement to query data.

## Syntax

```

select_statement ::= SELECT [ JSON ] ( select_clause | '*' )
                        FROM table_name
                        [ WHERE where_clause ]
                        [ LIMIT (integer | bind_marker) ]
                        [ ALLOW FILTERING ]
select_clause      ::= selector [ AS identifier ] ( ',' selector [ AS identifier ] )
selector           ::= column_name
                        | term
                        | CAST '(' selector AS cql_type ')'
                        | function_name '(' [ selector ( ',' selector ) * ] ')'
where_clause      ::= relation ( AND relation ) *
relation           ::= column_name operator term
operator           ::= '=' | '<' | '>' | '<=' | '>=' | CONTAINS | CONTAINS KEY
ordering_clause   ::= column_name [ ASC | DESC ] ( ',' column_name [ ASC | DESC ] ) *

```

## Examples

```

SELECT name, id, manager_id FROM "myGSGKeyspace".employees_tbl ;

SELECT JSON name, id, manager_id FROM "myGSGKeyspace".employees_tbl ;

```

For a table that maps JSON encoded data types to Amazon Keyspaces data types see [the section called “JSON encoding of Amazon Keyspaces data types” \(p. 196\)](#).

## Ordering results

The ORDER BY clause specifies the sort order of the returned results. It takes as arguments a list of column names along with the sort order for each column. You can only specify clustering columns in ordering clauses. Non-clustering columns are not allowed. The sort order options are ASC for ascending and DESC for descending sort order. If the sort order is omitted, the default ordering of the clustering column is used. For possible sort orders, see [the section called “Ordering results” \(p. 93\)](#).

## Example

```

SELECT name, id, division, manager_id FROM "myGSGKeyspace".employees_tbl WHERE id =
'012-34-5678' ORDER BY division;

```

# INSERT

Use the INSERT statement to add a row to a table.

## Syntax

```

insert_statement ::= INSERT INTO table_name ( names_values | json_clause )
                        [ IF NOT EXISTS ]
names_values      ::= names VALUES tuple_literal
json_clause       ::= JSON string [ DEFAULT ( NULL | UNSET ) ]
names             ::= '(' column_name ( ',' column_name ) * ')'

```

## Example

```

INSERT INTO "myGSGKeyspace".employees_tbl (id, name, project, region, division, role,
pay_scale, vacation_hrs, manager_id)

```

```
VALUES ( '012-34-5678', 'Russ', 'NightFlight', 'US', 'Engineering', 'IC', 3, 12.5, '234-56-7890' ) ;
```

### JSON support

For a table that maps JSON encoded data types to Amazon Keyspaces data types see [the section called "JSON encoding of Amazon Keyspaces data types" \(p. 196\)](#).

The JSON keyword can be used to insert a JSON encoded map as a single row. For columns that exist in the table but are omitted in the JSON insert statement, use `DEFAULT UNSET` to preserve the existing values. Use `DEFAULT NULL` to write a NULL value into each row of omitted columns and overwrite the existing values (standard write charges apply). `DEFAULT NULL` is the default option.

### Example

```
INSERT INTO "myGSGKeyspace".employees_tbl JSON '{ "id": "012-34-5678",
                                                    "name": "Russ",
                                                    "project": "NightFlight",
                                                    "region": "US",
                                                    "division": "Engineering",
                                                    "role": "IC",
                                                    "pay_scale": 3,
                                                    "vacation_hrs": 12.5,
                                                    "manager_id": "234-56-7890" }';
```

If the JSON data contains duplicate keys, Amazon Keyspaces stores the last value for the key (similar to Apache Cassandra). In the following example, where the duplicate key is `id`, the value `234-56-7890` will be written.

### Example

```
INSERT INTO "myGSGKeyspace".employees_tbl JSON '{ "id": "012-34-5678",
                                                    "name": "Russ",
                                                    "project": "NightFlight",
                                                    "region": "US",
                                                    "division": "Engineering",
                                                    "role": "IC",
                                                    "pay_scale": 3,
                                                    "vacation_hrs": 12.5,
                                                    "id": "234-56-7890" }';
```

## UPDATE

Use the `UPDATE` statement to modify a row in a table.

### Syntax

```
update_statement ::= UPDATE table_name
                  [ USING update_parameter ( AND update_parameter ) * ]
                  SET assignment ( ',' assignment ) *
                  WHERE where_clause
                  [ IF ( EXISTS | condition ( AND condition ) * ) ]
update_parameter ::= ( integer | bind_marker )
assignment       ::= simple_selection '=' term
                  | column_name '=' column_name ( '+' | '-' ) term
                  | column_name '=' list_literal '+' column_name
simple_selection  ::= column_name
                  | column_name '[' term ']'
                  | column_name '.' `field_name`
condition        ::= simple_selection operator term
```

### Example

```
UPDATE "myGSGKeyspace".employees_tbl SET pay_scale = 5 WHERE id = '567-89-0123' AND  
division = 'Marketing' ;
```

To increment a counter, use the following syntax. For more information, see [the section called “Counters” \(p. 195\)](#).

```
UPDATE ActiveUsers SET counter = counter + 1 WHERE user = A70FE1C0-5408-4AE3-  
BE34-8733E5K09F14 AND action = 'click';
```

## DELETE

Use the DELETE statement to remove a row from a table.

### Syntax

```
delete_statement ::= DELETE [ simple_selection ( ',' simple_selection ) ]  
                        FROM table_name  
  
                        WHERE where_clause  
                        [ IF ( EXISTS | condition ( AND condition )*) ]  
  
simple_selection ::= column_name  
                        | column_name '[' term ']'  
                        | column_name '.' `field_name`  
  
condition           ::= simple_selection operator term
```

Where:

- *table\_name* is the table that contains the row you want to delete.

### Example

```
DELETE manager_id FROM "myGSGKeyspace".employees_tbl WHERE id='789-01-2345' AND  
division='Executive' ;
```

## Built-in functions in Amazon Keyspaces

Amazon Keyspaces (for Apache Cassandra) supports a variety of built-in functions that you can use in Cassandra Query Language (CQL) statements.

### Topics

- [Scalar functions \(p. 207\)](#)

## Scalar functions

A *scalar function* performs a calculation on a single value and returns the result as a single value. Amazon Keyspaces supports the following scalar functions.

Function	Description
<code>uuid</code>	Returns a random type 4 UUID.
<code>now</code>	Returns a new unique <code>timeuuid</code> .
<code>currentTimestamp</code>	Returns the current date/time as a timestamp.
<code>currentDate</code>	Returns the current date/time as a date.
<code>currentTime</code>	Returns the current date/time as a time.
<code>currentTimeUUID</code>	Returns the current date/time as a <code>timeuuid</code> .
<code>toDate</code>	Converts either a <code>timeuuid</code> or a timestamp to a date type.
<code>toTimestamp</code>	Converts either a <code>timeuuid</code> or a date to a timestamp.
<code>toUnixTimestamp</code>	Converts either a <code>timeuuid</code> or a timestamp into a <code>bigInt</code> .
<code>toJson</code>	Returns the column value of the selected column in JSON format.
<code>fromJson</code>	Converts the JSON string into the selected column's data type.
<code>dateOf</code>	<i>(Deprecated)</i> Extracts the timestamp of a <code>timeuuid</code> , and returns the value as a date.
<code>unixTimestampOf</code>	<i>(Deprecated)</i> Extracts the timestamp of a <code>timeuuid</code> , and returns the value as a raw, 64-bit integer timestamp.

# Quotas for Amazon Keyspaces (for Apache Cassandra)

This section describes current quotas and default values for Amazon Keyspaces (for Apache Cassandra).

## Topics

- [Amazon Keyspaces service quotas](#) (p. 209)
- [Increasing or decreasing throughput \(for provisioned tables\)](#) (p. 211)
- [Amazon Keyspaces encryption at rest](#) (p. 211)

## Amazon Keyspaces service quotas

The following table contains Amazon Keyspaces (for Apache Cassandra) quotas and the default values. Some quotas can be updated in the [Service Quotas](#) console. To increase all other quotas for your account, contact AWS Support.

Quota	Description	Amazon Keyspaces default
Max keyspaces per AWS Region	The maximum number of keyspaces for this subscriber per Region. This default value is adjustable in the <a href="#">AWS Service Quotas</a> console.	256
Max tables per AWS Region	The maximum number of tables across all keyspaces for this subscriber per Region. This default value is adjustable in the <a href="#">AWS Service Quotas</a> console.	256
Max table schema size	The maximum size of a table schema.	350 KB
Max concurrent DDL operations	The maximum number of concurrent DDL operations allowed for this subscriber per Region. This default value is adjustable in the <a href="#">AWS Service Quotas</a> console.	50
Max queries per connection	The maximum number of CQL queries that can be processed by a single client TCP connection per second.	3000
Max row size	The maximum size of a row, excluding static column data. For details, see <a href="#">the section called "Calculating row size"</a> (p. 91).	1 MB

Quota	Description	Amazon Keyspaces default
Max static data per logical partition	The maximum aggregate size of static data in a logical partition. For details, see <a href="#">the section called "Calculating static column size per logical partition" (p. 89)</a> .	1 MB
Max read throughput per second	The maximum read throughput per second—read request units (RRUs) or read capacity units (RCUs)—that can be allocated to a table per Region. This default value is adjustable in the <a href="#">AWS Service Quotas</a> console.	40,000
Max write throughput per second	The maximum write throughput per second—write request units (WRUs) or write capacity units (WCUs)—that can be allocated to a table per Region. This default value is adjustable in the <a href="#">AWS Service Quotas</a> console.	40,000
Account-level read throughput (provisioned)	The maximum number of aggregate read capacity units (RCUs) allocated for the account per Region; applicable only for tables in provisioned read/write capacity mode. This default value is adjustable in the <a href="#">AWS Service Quotas</a> console.	80,000
Account-level write throughput (provisioned)	The maximum number of aggregate write capacity units (WCU) allocated for the account per Region; applicable only for tables in provisioned read/write capacity mode. This default value is adjustable in the <a href="#">AWS Service Quotas</a> console.	80,000
Max partition key size	The maximum size of the compound partition key. Up to 3 bytes of additional storage are added to the raw size of each column included in the partition key for metadata.	2048 bytes
Max clustering key size	The maximum combined size of all clustering columns. Up to 4 bytes of additional storage are added to the raw size of each clustering column for metadata.	850 bytes



Quota	Description	Amazon Keyspaces default
Max concurrent table restores using Point-in-time Recovery (PITR)	The maximum number of concurrent table restores using PITR per subscriber is 4. This default value is adjustable in the <a href="#">AWS Service Quotas</a> console.	4
Max amount of data restored using point-in-time recovery (PITR)	The maximum size of data that can be restored using PITR within 24 hours. This default value is adjustable in the <a href="#">AWS Service Quotas</a> console.	5 TB

## Increasing or decreasing throughput (for provisioned tables)

### Increasing provisioned throughput

You can increase `ReadCapacityUnits` or `WriteCapacityUnits` as often as necessary, using the console or the `ALTER TABLE` statement. The new settings do not take effect until the `ALTER TABLE` operation is complete.

You can't exceed your per-account quotas when you add provisioned capacity and you can increase the provisioned capacity for your tables as much as you need. For more information about per-account quotas, see the preceding section, [the section called “Amazon Keyspaces service quotas” \(p. 209\)](#).

### Decreasing provisioned throughput

For every table in an `ALTER TABLE` statement, you can decrease `ReadCapacityUnits` or `WriteCapacityUnits` (or both). The new settings don't take effect until the `ALTER TABLE` operation is complete. A decrease is allowed up to four times, anytime per day. A day is defined according to Universal Coordinated Time (UTC). Additionally, if there was no decrease in the past hour, an additional decrease is allowed. This effectively brings the maximum number of decreases in a day to 27 (4 decreases in the first hour, and 1 decrease for each of the subsequent 1-hour windows in a day).

## Amazon Keyspaces encryption at rest

You can change encryption options between an AWS owned AWS KMS key and a customer managed AWS KMS key up to four times within a 24-hour window, on a per table basis, starting from when the table was created. If there was no change in the past six hours, an additional change is allowed. This effectively brings the maximum number of changes in a day to eight (four changes in the first six hours, and one change for each of the subsequent six-hour windows in a day).

You can change the encryption option to use an AWS owned AWS KMS key as often as necessary, even if the earlier quota has been exhausted.

These are the quotas unless you request a higher amount. To request a service quota increase, see <https://aws.amazon.com/support>.

# Document history for Amazon Keyspaces (for Apache Cassandra)

The following table describes the important changes to the documentation since the last release of Amazon Keyspaces (for Apache Cassandra). For notification about updates to this documentation, you can subscribe to an RSS feed.

- **Latest documentation update:** August 9, 2021

update-history-change	update-history-description	update-history-date
<a href="#">Migrating data to Amazon Keyspaces using DSBulk.</a>	Step-by-step tutorial for migrating data from Apache Cassandra to Amazon Keyspaces using the DataStax Bulk Loader (DSBulk).	August 9, 2021
<a href="#">Amazon Keyspaces support for VPC Endpoint entries in the <code>system.peers</code> table.</a>	Amazon Keyspaces allows you to populate the <code>system.peers</code> table with available interface VPC endpoint information to improve load balancing and increase read/write throughput.	July 29, 2021
<a href="#">Update to IAM managed policies to support customer managed AWS KMS keys.</a>	IAM managed policies for Amazon Keyspaces now include permissions to list and view available customer managed AWS KMS keys stored in AWS KMS.	June 1, 2021
<a href="#">Amazon Keyspaces support for customer managed AWS KMS keys.</a>	Amazon Keyspaces allows you to take control of customer managed AWS KMS keys stored in AWS KMS for encryption at rest.	June 1, 2021
<a href="#">Amazon Keyspaces support for JSON syntax</a>	Amazon Keyspaces helps you read and write JSON documents more easily by supporting JSON syntax for INSERT and SELECT operations.	January 21, 2021
<a href="#">Amazon Keyspaces support for static columns</a>	Amazon Keyspaces now helps you update and store common data between multiple rows efficiently by using static columns.	November 9, 2020
<a href="#">GA release of NoSQL Workbench support for Amazon Keyspaces</a>	NoSQL Workbench is a client-side application that helps you design and visualize nonrelational data models for	October 28, 2020

	Amazon Keyspaces more easily. NoSQL Workbench clients are available for Windows, macOS, and Linux.	
<a href="#">Preview release of NoSQL Workbench support for Amazon Keyspaces</a>	NoSQL Workbench is a client-side application that helps you design and visualize nonrelational data models for Amazon Keyspaces more easily. NoSQL Workbench clients are available for Windows, macOS, and Linux.	October 5, 2020
<a href="#">New code examples for programmatic access to Amazon Keyspaces</a>	We continue to add code examples for programmatic access to Amazon Keyspaces. Samples are now available for Java, Python, Go, C#, and Perl Cassandra drivers that support Apache Cassandra version 3.11.2.	July 17, 2020
<a href="#">Amazon Keyspaces point-in-time recovery (PITR)</a>	Amazon Keyspaces now offers point-in-time recovery (PITR) to help protect your tables from accidental write or delete operations by providing you continuous backups of your table data.	July 9, 2020
<a href="#">Amazon Keyspaces general availability</a>	With Amazon Keyspaces, formerly known during preview as Amazon Managed Apache Cassandra Service (MCS), you can use the Cassandra Query Language (CQL) code, Apache 2.0–licensed Cassandra drivers, and developer tools that you already use today.	April 23, 2020
<a href="#">Amazon Keyspaces automatic scaling</a>	Amazon Keyspaces (for Apache Cassandra) integrates with Application Auto Scaling to help you provision throughput capacity efficiently for variable workloads in response to actual application traffic by adjusting throughput capacity automatically.	April 23, 2020
<a href="#">Interface virtual private cloud (VPC) endpoints for Amazon Keyspaces</a>	Amazon Keyspaces offers private communication between the service and your VPC so that network traffic doesn't leave the Amazon network.	April 16, 2020

<a href="#">Tag-based access policies</a>	You can now use resource tags in IAM policies to manage access to Amazon Keyspaces.	April 8, 2020
<a href="#">Counter data type</a>	Amazon Keyspaces now helps you coordinate increments and decrements to column values by using counters.	April 7, 2020
<a href="#">Tagging resources</a>	Amazon Keyspaces now enables you to label and categorize resources by using tags.	March 31, 2020
<a href="#">AWS CloudFormation support</a>	Amazon Keyspaces now helps you automate the creation and management of resources by using AWS CloudFormation.	March 25, 2020
<a href="#">Support for IAM roles and policies and SigV4 authentication</a>	Added information on how you can use <a href="#">AWS Identity and Access Management (IAM)</a> to manage access permissions and implement security policies for Amazon Keyspaces and how to use the authentication plugin for the DataStax Java Driver for Cassandra to programmatically access Amazon Keyspaces using IAM roles and federated identities.	March 17, 2020
<a href="#">Read/write capacity mode</a>	Amazon Keyspaces now supports two read/write throughput capacity modes. The read/write capacity mode controls how you're charged for read and write throughput and how table throughput capacity is managed.	February 20, 2020
<a href="#">Initial release</a>	This documentation covers the initial release of Amazon Keyspaces (for Apache Cassandra).	December 3, 2019