Apache Cassandra : Getting Started

Cassandra Quickstart

Cassandra Quickstart

STEP 1: GET CASSANDRA USING DOCKER

You'll need to have Docker Desktop for Mac, Docker Desktop for Windows, or similar software installed on your computer.

Apache Cassandra is also available as a tarball or package **download**.

```
docker pull cassandra:latest
```

STEP 2: START CASSANDRA

A Docker network allows us to access the container's ports without exposing them on the host.

```
docker network create cassandra

docker run --rm -d --name cassandra --hostname cassandra --network cassandra cassandra
```

STEP 3: CREATE FILES

The Cassandra Query Language (CQL) is very similar to SQL but suited for the JOINless structure of Cassandra.

Create a file named data.cql and paste the following CQL script in it. This script will create a keyspace, the layer at which Cassandra replicates its data, a table to hold the data, and insert some data into that table:

```
CREATE KEYSPACE IF NOT EXISTS store WITH REPLICATION =
{ 'class' : 'SimpleStrategy',
'replication_factor' : '1'
};

CREATE TABLE IF NOT EXISTS store.shopping_cart (
    userid text PRIMARY KEY,
    item_count int,
    last_update_timestamp timestamp
);

INSERT INTO store.shopping_cart
```

```
    (userid, item_count, last_update_timestamp)
    VALUES ('9876', 2, toTimeStamp(now()));
INSERT INTO store.shopping_cart
    (userid, item_count, last_update_timestamp)
    VALUES ('1234', 5, toTimeStamp(now()));
```

STEP 4: LOAD DATA WITH CQLSH

The CQL shell, or `cqlsh`, is one tool to use in interacting with the database. We'll use it to load some data into the database using the script you just saved.

```
docker run --rm --network cassandra \
-v "$(pwd)/data.cql:/scripts/data.cql" \
-e CQLSH_HOST=cassandra -e CQLSH_PORT=9042 \
-e CQLVERSION=3.4.6 nuvo/docker-cqlsh
```

NOTE The cassandra server itself (the first docker run command you ran) takes a few seconds to start up. The above command will throw an error if the server hasn't finished its init sequence yet, so give it a few seconds to spin up.

STEP 5: INTERACTIVE CQLSH

Much like an SQL shell, you can also of course use `cqlsh` to run CQL commands interactively.

```
docker run --rm -it --network \
cassandra nuvo/docker-cqlsh cqlsh cassandra \
9042 --cqlversion='3.4.5'
```

This should get you a prompt like so:

```
Connected to Test Cluster at cassandra:9042.
[cqlsh 5.0.1 | Cassandra 4.0.4 | CQL spec 3.4.5 | Native protocol v5]
Use HELP for help.
cqlsh>
```

STEP 6: READ SOME DATA

```
SELECT * FROM store.shopping_cart;
```

STEP 7: WRITE SOME MORE DATA

```
INSERT INTO store.shopping_cart
    (userid, item_count)
    VALUES ('4567', 20);
```

STEP 8: CLEAN UP

```
docker kill cassandra
docker network rm cassandra
```

CONGRATULATIONS!

Hey, that wasn’t so hard, was it?

To learn more, we suggest the following next steps:

- Read through the **Cassandra Basics** to learn main concepts and how Cassandra works at a high level.
- Browse through the **Case Studies** to learn how other users in our worldwide community are getting value out of Cassandra.

Storage-attached Indexing (SAI) Quickstart

Storage-attached Indexing (SAI) Quickstart

To get started with Storage-Attached Indexing (SAI), we'll do the following steps:

- Create a keyspace.
- Create a table.
- Create an **index using SAI**.
- Add data.
- Create and run a query using SAI.

The examples in this quickstart topic show SAI indexes with non-partition key columns.

Create a keyspace

In `cqlsh`, define the `cycling` keyspace to try the commands in a test environment:

```
CREATE KEYSPACE IF NOT EXISTS cycling
   WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : '1' };
```

Create a database table

Using `cqlsh` or the CQL Console, create a `cyclist_semi_pro` database **table** in the `cycling` keyspace or the keyspace name of your choice:

```
CREATE TABLE IF NOT EXISTS cycling.cyclist_semi_pro (
  id int,
  firstname text,
  lastname text,
  age int,
  affiliation text,
  country text,
  registration date,
  PRIMARY KEY (id));
```

Create SAI indexes on the database table

To test a non-trivial query, you'll need some SAI indexes. Use **CREATE CUSTOM INDEX** commands to create SAI indexes on a few non-primary-key columns in the `cyclist_semi_pro` table:

```
CREATE INDEX lastname_sai_idx ON cycling.cyclist_semi_pro (lastname)
USING 'sai'
WITH OPTIONS = {'case_sensitive': 'false', 'normalize': 'true', 'ascii': 'true'};

CREATE INDEX age_sai_idx ON cycling.cyclist_semi_pro (age)
USING 'sai';

CREATE INDEX country_sai_idx ON cycling.cyclist_semi_pro (country)
USING 'sai'
WITH OPTIONS = {'case_sensitive': 'false', 'normalize': 'true', 'ascii': 'true'};

CREATE INDEX registration_sai_idx ON cycling.cyclist_semi_pro (registration)
USING 'sai';
```

Let's take a look at the description of the table and its indexes:

Query

```
DESCRIBE TABLE cycling.cyclist_semi_pro;
```

Result

```
CREATE TABLE cycling.cyclist_semi_pro (
    id int PRIMARY KEY,
    affiliation text,
    age int,
    country text,
    firstname text,
    lastname text,
    registration date
) WITH additional_write_policy = '99PERCENTILE'
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class':
'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy',
'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class':
'org.apache.cassandra.io.compress.LZ4Compressor'}
```

```
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND nodesync = {'enabled': 'true', 'incremental': 'true'}
    AND read_repair = 'BLOCKING'
    AND speculative_retry = '99PERCENTILE';
CREATE INDEX registration_sai_idx ON cycling.cyclist_semi_pro (registration) USING
'sai';
CREATE INDEX country_sai_idx ON cycling.cyclist_semi_pro (country) USING 'sai'
WITH OPTIONS = {'normalize': 'true', 'case_sensitive': 'false', 'ascii': 'true'};
CREATE INDEX age_sai_idx ON cycling.cyclist_semi_pro (age) USING 'sai';
CREATE INDEX lastname_sai_idx ON cycling.cyclist_semi_pro (lastname) USING 'sai'
WITH OPTIONS = {'normalize': 'true', 'case_sensitive': 'false', 'ascii': 'true'};
```

Add data to your table

Use CQLSH `INSERT` commands to add some data to the `cyclist_semi_pro` database table:

```
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (1, 'Carlos', 'Perotti', 22, 'Recco Club', 'ITA', '2020-01-12');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (2, 'Giovani', 'Pasi', 19, 'Venezia Velocità', 'ITA', '2016-05-15');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (3, 'Frances', 'Giardello', 24, 'Menaggio Campioni', 'ITA', '2018-
07-29');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (4, 'Mark', 'Pastore', 19, 'Portofino Ciclisti', 'ITA', '2017-06-
16');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (5, 'Irene', 'Cantona', 24, 'Como Velocità', 'ITA', '2012-07-22');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (6, 'Hugo', 'Herrera', 23, 'Bellagio Ciclisti', 'ITA', '2004-02-
12');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (7, 'Marcel', 'Silva', 21, 'Paris Cyclistes', 'FRA', '2018-04-28');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (8, 'Theo', 'Bernat', 19, 'Nice Cavaliers', 'FRA', '2007-05-15');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (9, 'Richie', 'Draxler', 24, 'Normandy Club', 'FRA', '2011-02-26');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (10, 'Agnes', 'Cavani', 22, 'Chamonix Hauteurs', 'FRA', '2020-01-
02');
```

```
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (11, 'Pablo', 'Verratti', 19, 'Chamonix Hauteurs', 'FRA', '2006-05-
15');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (12, 'Charles', 'Eppinger', 24, 'Chamonix Hauteurs', 'FRA', '2018-
07-29');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (13, 'Stanley', 'Trout', 30, 'Bolder Boulder', 'USA', '2016-02-12');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (14, 'Juan', 'Perez', 31, 'Rutgers Alumni Riders', 'USA', '2017-06-
16');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (15, 'Thomas', 'Fulton', 27, 'Exeter Academy', 'USA', '2012-12-15');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (16, 'Jenny', 'Hamler', 28, 'CU Alums Crankworkz', 'USA', '2012-07-
22');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (17, 'Alice', 'McCaffrey', 26, 'Pennan Power', 'GBR', '2020-02-12');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (18, 'Nicholas', 'Burrow', 26, 'Aberdeen Association', 'GBR', '2016-
02-12');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (19, 'Tyler', 'Higgins', 24, 'Highclere Agents', 'GBR', '2019-07-
31');
INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age, affiliation, country,
registration) VALUES (20, 'Leslie', 'Boyd', 18, 'London Cyclists', 'GBR', '2012-12-15');
```

Adding data in this quickstart topic shows simple `INSERT` commands. To load databases with many rows, consider using DataStax Bulk Loader for Apache Cassandra.

Try out CQL queries

Use the CQLSH `SELECT` command to submit queries.

TIP The supported query operators for tables with SAI indexes: * Numerics: `=`, `<`, `>`, `<=`, `>=`, `AND`, `OR`, `IN` * Strings: `=`, `CONTAINS`, `CONTAINS KEY`, `AND`, `OR`, `IN` * Strings or Numerics: `LIKE`

Find a specific semi-pro cyclist:

Query

```
SELECT * FROM cycling.cyclist_semi_pro WHERE lastname = 'Eppinger';
```

Result

```
id | affiliation       | age | country | firstname | lastname | registration
----+-------------------+-----+---------+-----------+----------+--------------
 12 | Chamonix Hauteurs |  24 |     FRA |   Charles | Eppinger |   2018-07-29

(1 rows)
```

Find semi-pro cyclists whose age is less than or equal to 23:

Query

```
SELECT * FROM cycling.cyclist_semi_pro WHERE age <= 23;
```

Result

```
id | affiliation        | age | country | firstname | lastname | registration
----+--------------------+-----+---------+-----------+----------+--------------
 10 |  Chamonix Hauteurs |  22 |     FRA |     Agnes |   Cavani |   2020-01-02
 11 |  Chamonix Hauteurs |  19 |     FRA |     Pablo | Verratti |   2006-05-15
  1 |         Recco Club |  22 |     ITA |    Carlos |  Perotti |   2020-01-12
  8 |     Nice Cavaliers |  19 |     FRA |      Theo |   Bernat |   2007-05-15
  2 |   Venezia Velocità |  19 |     ITA |   Giovani |     Pasi |   2016-05-15
  4 | Portofino Ciclisti |  19 |     ITA |      Mark |  Pastore |   2017-06-16
 20 |    London Cyclists |  18 |     GBR |    Leslie |     Boyd |   2012-12-15
  7 |    Paris Cyclistes |  21 |     FRA |    Marcel |    Silva |   2018-04-28
  6 |  Bellagio Ciclisti |  23 |     ITA |      Hugo |  Herrera |   2004-02-12

(9 rows)
```

Find semi-pro cyclists from Great Britain:

Query

```
SELECT * FROM cycling.cyclist_semi_pro WHERE country = 'GBR';
```

Result

```
id | affiliation          | age | country | firstname | lastname  | registration
----+----------------------+-----+---------+-----------+-----------+--------------
 19 |     Highclere Agents |  24 |     GBR |     Tyler |   Higgins |   2019-07-31
 18 | Aberdeen Association |  26 |     GBR |  Nicholas |    Burrow |   2016-02-12
 20 |      London Cyclists |  18 |     GBR |    Leslie |      Boyd |   2012-12-15
 17 |         Pennan Power |  26 |     GBR |     Alice | McCaffrey |   2020-02-12

(4 rows)
```

Find semi-pro cyclists who registered between a given date range:

Query

```
SELECT * FROM cycling.cyclist_semi_pro WHERE registration > '2010-01-01' AND
registration < '2015-12-31' LIMIT 10;
```

Result

```
id | affiliation         | age | country | firstname | lastname | registration
----+---------------------+-----+---------+-----------+----------+--------------
  5 |        Como Velocità |  24 |     ITA |     Irene |  Cantona |   2012-07-22
 16 | CU Alums Crankworkz |  28 |     USA |     Jenny |   Hamler |   2012-07-22
 15 |      Exeter Academy |  27 |     USA |    Thomas |   Fulton |   2012-12-15
 20 |     London Cyclists |  18 |     GBR |    Leslie |     Boyd |   2012-12-15
  9 |        Normandy Club |  24 |     FRA |    Richie |  Draxler |   2011-02-26

(5 rows)
```

TIP For query examples with `CONTAINS` clauses that take advantage of SAI collection maps, lists, and sets, be sure to see **SAI collection map examples with keys, values, and entries**.

Removing an SAI index

To remove an SAI index, use `DROP INDEX`.

Example:

```
DROP INDEX IF EXISTS cycling.age_sai_idx;
```

Resources

SAI overview

Vector Search Quickstart

Vector Search Quickstart

To enable your machine learning model, Vector Search uses data to be compared by similarity within a database, even if it is not explicitly defined by a connection. A vector is an array of floating point type that represents a specific object or entity.

The foundation of Vector Search lies within the embeddings, which are compact representations of text as vectors of floating-point numbers. These embeddings are generated by feeding the text through an API, which uses a neural network to transform the input into a fixed-length vector. Embeddings capture the semantic meaning of the text, providing a more nuanced understanding than traditional term-based approaches. The vector representation allows for input that is substantially similar to produce output vectors that are geometrically close; inputs that are not similar are geometrically further apart.

To enable Vector Search, a new `vector` data type is available in your **Cassandra** database with Vector Search.

Prerequisites

There are no prerequisite tasks.

In general, to use Vector Search with Apache Cassandra, you'll follow these instructions:

The embeddings were randomly generated in this quickstart. Generally, you would run both your source documents/contents through an embeddings generator, as well as the query you were asking to match. This example is simply to show the mechanics of how to use CQL to create vector search data objects.

Create vector keyspace

Create the keyspace you want to use for your Vector Search table. This example uses `cycling` as the `keyspace name`:

```
CREATE KEYSPACE IF NOT EXISTS cycling
   WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : '1' };
```

Use vector keyspace

Select the keyspace you want to use for your Vector Search table. This example uses `cycling` as the `keyspace name`:

```
USE cycling;
```

Create vector table

Create a new table in your keyspace, including the `comments_vector` column for vector. The code below creates a vector with five values:

```
CREATE TABLE IF NOT EXISTS cycling.comments_vs (
  record_id timeuuid,
  id uuid,
  commenter text,
  comment text,
  comment_vector VECTOR <FLOAT, 5>,
  created_at timestamp,
  PRIMARY KEY (id, created_at)
)
WITH CLUSTERING ORDER BY (created_at DESC);
```

Optionally, you can alter an existing table to add a vector column:

```
ALTER TABLE cycling.comments_vs
   ADD comment_vector VECTOR <FLOAT, 5> ;
```

Create vector index

Create the custom index with Storage Attached Indexing (SAI):

```
CREATE INDEX IF NOT EXISTS ann_index
  ON cycling.comments_vs(comment_vector) USING 'sai';
```

For more about SAI, see the **Storage Attached Indexing** documentation.

IMPORTANT

The index can be created with options that define the similarity function:

```
CREATE INDEX IF NOT EXISTS ann_index
    ON vsearch.com(item_vector) USING 'sai'
WITH OPTIONS = { 'similarity_function': 'DOT_PRODUCT' };
```

> Valid values for the `similarity_function` are `DOT_PRODUCT`, `COSINE`, or `EUCLIDEAN`.

Load vector data into your database

Insert data into the table using the new type:

```
INSERT INTO cycling.comments_vs (record_id, id, created_at, comment, commenter,
comment_vector)
   VALUES (
      now(),
      e7ae5cf3-d358-4d99-b900-85902fda9bb0,
      '2017-02-14 12:43:20-0800',
      'Raining too hard should have postponed',
      'Alex',
      [0.45, 0.09, 0.01, 0.2, 0.11]
);
INSERT INTO cycling.comments_vs (record_id, id, created_at, comment, commenter,
comment_vector)
   VALUES (
      now(),
      e7ae5cf3-d358-4d99-b900-85902fda9bb0,
      '2017-03-21 13:11:09.999-0800',
      'Second rest stop was out of water',
      'Alex',
      [0.99, 0.5, 0.99, 0.1, 0.34]
);
INSERT INTO cycling.comments_vs (record_id, id, created_at, comment, commenter,
comment_vector)
   VALUES (
      now(),
      e7ae5cf3-d358-4d99-b900-85902fda9bb0,
      '2017-04-01 06:33:02.16-0800',
      'LATE RIDERS SHOULD NOT DELAY THE START',
      'Alex',
      [0.9, 0.54, 0.12, 0.1, 0.95]
);

INSERT INTO cycling.comments_vs (record_id, id, created_at, comment, commenter,
comment_vector)
   VALUES (
      now(),
      c7fceba0-c141-4207-9494-a29f9809de6f,
      totimestamp(now()),
      'The gift certificate for winning was the best',
      'Amy',
      [0.13, 0.8, 0.35, 0.17, 0.03]
);

INSERT INTO cycling.comments_vs (record_id, id, created_at, comment, commenter,
comment_vector)
   VALUES (
      now(),
      c7fceba0-c141-4207-9494-a29f9809de6f,
      '2017-02-17 12:43:20.234+0400',
      'Glad you ran the race in the rain',
      'Amy',
      [0.3, 0.34, 0.2, 0.78, 0.25]
);

INSERT INTO cycling.comments_vs (record_id, id, created_at, comment, commenter,
comment_vector)
   VALUES (
      now(),
      c7fceba0-c141-4207-9494-a29f9809de6f,
      '2017-03-22 5:16:59.001+0400',
      'Great snacks at all reststops',
      'Amy',
      [0.1, 0.4, 0.1, 0.52, 0.09]
);
INSERT INTO cycling.comments_vs (record_id, id, created_at, comment, commenter,
comment_vector)
   VALUES (
      now(),
      c7fceba0-c141-4207-9494-a29f9809de6f,
      '2017-04-01 17:43:08.030+0400',
      'Last climb was a killer',
      'Amy',
      [0.3, 0.75, 0.2, 0.2, 0.5]
);
```

Query vector data with CQL

To query data using Vector Search, use a `SELECT` query:

```
SELECT * FROM cycling.comments_vs
    ORDER BY comment_vector ANN OF [0.15, 0.1, 0.1, 0.35, 0.55]
    LIMIT 3;
```

To obtain the similarity calculation of the best scoring node closest to the query data as part of the results, use a `SELECT` query:

```
SELECT comment, similarity_cosine(comment_vector, [0.2, 0.15, 0.3, 0.2, 0.05])
    FROM cycling.comments_vs
    ORDER BY comment_vector ANN OF [0.1, 0.15, 0.3, 0.12, 0.05]
    LIMIT 1;
```

The supported functions for this type of query are:

- similarity_dot_product
- similarity_cosine
- similarity_euclidean

with the parameters of (<vector_column>, <embedding_value>). Both parameters represent vectors.

NOTE

- The limit must be 1,000 or fewer.
- Vector Search utilizes Approximate Nearest Neighbor (ANN) that in most cases yields results almost as good as the exact match. The scaling is superior to Exact Nearest Neighbor (KNN).
- Least-similar searches are not supported.
- Vector Search works optimally on tables with no overwrites or deletions of the `item_vector` column. For an `item_vector` column with changes, expect slower search results.

With the code examples, you have a working example of our Vector Search. Load your own data and use the search function.

Installing Cassandra

Installing Cassandra

Apache Cassandra can be installed on a number of Linux distributions:

- AlmaLinux
- Amazon Linux Amazon Machine Images (AMIs)
- Debian
- RedHat Enterprise Linux (RHEL)
- SUSE Enterprise Linux
- Ubuntu

This is not an exhaustive list of operating system platforms, nor is it prescriptive. However, users are well-advised to conduct exhaustive tests if using a less-popular distribution of Linux. Deploying on older Linux versions is not recommended unless you have previous experience with the older distribution in a production environment.

Prerequisites

- Install the latest version of Java 11 or Java 17, from one of the following locations:
  - Oracle Java Standard Edition 11 Archived Version
  - Oracle Java Standard Edition 17
  - OpenJDK 11
  - OpenJDK 17

1. Verify the version of Java installed. For example:

Command

```
$ java -version
```

Result

```
openjdk version "11.0.20" 2023-07-18
OpenJDK Runtime Environment Temurin-11.0.20+8 (build 11.0.20+8)
OpenJDK 64-Bit Server VM Temurin-11.0.20+8 (build 11.0.20+8, mixed mode)
```

- To use the CQL shell `cqlsh`, install the latest version of Python 3.8-3.11.

To verify that you have the correct version of Python installed, type `python --version`.

NOTE Support for Python 2.7 is deprecated.

Choosing an installation method

There are three methods of installing Cassandra that are common:

- Docker image
- Tarball binary file
- Package installation (RPM, YUM)

If you are a current Docker user, installing a Docker image is simple. You'll need to install Docker Desktop for Mac, Docker Desktop for Windows, or have `docker` installed on Linux. Pull the appropriate image from the Docker Hub and start Cassandra with a `docker run` command.

For many users, installing the binary tarball is also a simple choice. The tarball unpacks all its contents into a single location with binaries and configuration files located in their own subdirectories. The most obvious advantage of a tarball installation is that it does not require `root` permissions and can be installed on any Linux distribution.

Packaged installations require `root` permissions, and are most appropriate for production installs. Install the RPM build on CentOS and RHEL-based distributions if you want to install Cassandra using YUM. Install the Debian build on Ubuntu and other Debian-based distributions if you want to install Cassandra using APT.

WARNING Note that both the YUM and APT methods required `root` permissions and will install the binaries and configuration files as the `cassandra` OS user.

Install with Docker

1. Pull the docker image. For the latest image, use:

   ```
   docker pull cassandra:latest
   ```

   This `docker pull` command will get the latest version of the official Docker Apache Cassandra image available from the Dockerhub.

2. Start Cassandra with a `docker run` command:

   ```
   docker run --name cass_cluster cassandra:latest
   ```

The `--name` option will be the name of the Cassandra cluster created. This example uses the name `cass_cluster`.

3. Start the CQL shell, `cqlsh` to interact with the Cassandra node created:

```
docker exec -it cass_cluster cqlsh
```

Install tarball file

1. Verify the version of Java installed. For example:

Command

```
$ java -version
```

Result

```
openjdk version "11.0.20" 2023-07-18
OpenJDK Runtime Environment Temurin-11.0.20+8 (build 11.0.20+8)
OpenJDK 64-Bit Server VM Temurin-11.0.20+8 (build 11.0.20+8, mixed mode)
```

1. Download the binary tarball from one of the mirrors on the {cass_url}download/[Apache Cassandra Download] site. For example, to download Cassandra {40_version}:

```
$ curl -OL http://apache.mirror.digitalpacific.com.au/cassandra/4.0.0/apache-
cassandra-4.0.0-bin.tar.gz
```

NOTE The mirrors only host the latest versions of each major supported release. To download an earlier version of Cassandra, visit the Apache Archives.

2. OPTIONAL: Verify the integrity of the downloaded tarball using one of the methods here. For example, to verify the hash of the downloaded file using GPG:

Command

```
$ gpg --print-md SHA256 apache-cassandra-4.0.0-bin.tar.gz
```

Result

```
apache-cassandra-4.0.0-bin.tar.gz: 28757DDE 589F7041 0F9A6A95 C39EE7E6
```

```
                              CDE63440 E2B06B91 AE6B2006 14FA364D
```

Compare the signature with the SHA256 file from the Downloads site:

Command

```
$ curl -L https://downloads.apache.org/cassandra/4.0.0/apache-cassandra-4.0.0-
bin.tar.gz.sha256
```

Result

```
28757dde589f70410f9a6a95c39ee7e6cde63440e2b06b91ae6b200614fa364d
```

3. Unpack the tarball:

```
$ tar xzvf apache-cassandra-4.0.0-bin.tar.gz
```

The files will be extracted to the `apache-cassandra-4.0.0/` directory. This is the tarball installation location.

4. Located in the tarball installation location are the directories for the scripts, binaries, utilities, configuration, data and log files:

```
<tarball_installation>/
    bin/         ①
    conf/        ②
    data/        ③
    doc/
    interface/
    javadoc/
    lib/
    logs/        ④
    pylib/
    tools/       ⑤
```

① location of the commands to run cassandra, cqlsh, nodetool, and SSTable tools

② location of cassandra.yaml and other configuration files

③ location of the commit logs, hints, and SSTables

④ location of system and debug logs <5>location of cassandra-stress tool

5. Start Cassandra:

```
$ cd apache-cassandra-4.0.0/ && bin/cassandra
```

NOTE This will run Cassandra as the authenticated Linux user.

6. Monitor the progress of the startup with:

Command

```
$ tail -f logs/system.log
```

Result

Cassandra is ready when you see an entry like this in the `system.log`:

+

```
INFO  [main] 2019-12-17 03:03:37,526 Server.java:156 - Starting listening for
CQL clients on localhost/127.0.0.1:9042 (unencrypted)...
```

NOTE See **configuring Cassandra** for configuration information.

1. Check the status of Cassandra:

```
$ bin/nodetool status
```

The status column in the output should report `UN` which stands for "Up/Normal".

Alternatively, connect to the database with:

```
$ bin/cqlsh
```

Install as Debian package

1. Verify the version of Java installed. For example:

Command

```
$ java -version
```

Result

```
openjdk version "11.0.20" 2023-07-18
OpenJDK Runtime Environment Temurin-11.0.20+8 (build 11.0.20+8)
OpenJDK 64-Bit Server VM Temurin-11.0.20+8 (build 11.0.20+8, mixed mode)
```

1. Add the Apache repository of Cassandra to the file `cassandra.sources.list`.

   The latest major version is {50_version} and the corresponding distribution name is `50x` (with an "x" as the suffix). For older releases use:

   - `50x` for C* {50_version} series
   - `41x` for C* {41_version} series
   - `40x` for C* {40_version} series

For example, to add the repository for version {50_version} (`50x`):

+

```
$ echo "deb [signed-by=/etc/apt/keyrings/apache-cassandra.asc]
https://debian.cassandra.apache.org 50x main" | sudo tee -a
/etc/apt/sources.list.d/cassandra.sources.list
deb https://debian.cassandra.apache.org 50x main
```

1. Add the Apache Cassandra repository keys to the list of trusted keys on the server:

   Command

   ```
   $ curl -o /etc/apt/keyrings/apache-cassandra.asc
   https://downloads.apache.org/cassandra/KEYS
   ```

   Result

   ```
   % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                   Dload  Upload   Total   Spent    Left  Speed
   100  266k  100  266k    0     0   320k      0 --:--:-- --:--:-- --:--:--  320k
   OK
   ```

2. Update the package index from sources:

```
$ sudo apt-get update
```

3. Install Cassandra with APT:

```
$ sudo apt-get install cassandra
```

4. Monitor the progress of the startup with:

Command

```
$ tail -f logs/system.log
```

Result

Cassandra is ready when you see an entry like this in the `system.log`:

+

```
INFO  [main] 2019-12-17 03:03:37,526 Server.java:156 - Starting listening for
CQL clients on localhost/127.0.0.1:9042 (unencrypted)...
```

NOTE See **configuring Cassandra** for configuration information.

1. Check the status of Cassandra:

```
$ nodetool status
```

The status column in the output should report `UN` which stands for "Up/Normal".

Alternatively, connect to the database with:

```
$ cqlsh
```

Install as RPM package

1. Verify the version of Java installed. For example:

Command

```
$ java -version
```

Result

```
openjdk version "11.0.20" 2023-07-18
OpenJDK Runtime Environment Temurin-11.0.20+8 (build 11.0.20+8)
OpenJDK 64-Bit Server VM Temurin-11.0.20+8 (build 11.0.20+8, mixed mode)
```

1. Add the Apache repository of Cassandra to the file `/etc/yum.repos.d/cassandra.repo` (as the `root` user).

   The latest major version is {50_version} and the corresponding distribution name is `50x` (with an "x" as the suffix). For older releases use:

   - `50x` for C* {50_version} series
   - `41x` for C* {41_version} series
   - `40x` for C* {40_version} series

For example, to add the repository for version {50_version} (`50x`):

+

```
[cassandra]
name=Apache Cassandra
baseurl=https://redhat.cassandra.apache.org/42x/
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://downloads.apache.org/cassandra/KEYS
```

1. Update the package index from sources:

   ```
   $ sudo yum update
   ```

2. Install Cassandra with YUM:

   ```
   $ sudo yum install cassandra
   ```

   NOTE A new Linux user `cassandra` will get created as part of the installation. The

Cassandra service will also be run as this user.

3. Start the Cassandra service:

```
$ sudo service cassandra start
```

4. Monitor the progress of the startup with:

Command

```
$ tail -f logs/system.log
```

Result

Cassandra is ready when you see an entry like this in the `system.log`:

+

```
INFO  [main] 2019-12-17 03:03:37,526 Server.java:156 - Starting listening for
CQL clients on localhost/127.0.0.1:9042 (unencrypted)...
```

NOTE See **configuring Cassandra** for configuration information.

1. Check the status of Cassandra:

```
$ nodetool status
```

The status column in the output should report `UN` which stands for "Up/Normal".

Alternatively, connect to the database with:

```
$ cqlsh
```

Further installation info

For help with installation issues, see the **Troubleshooting** section.

Configuring Cassandra

Configuring Cassandra

The `Cassandra` configuration files location varies, depending on the type of installation:

- docker: `/etc/cassandra` directory
- tarball: `conf` directory within the tarball install location
- package: `/etc/cassandra` directory

Cassandra's default configuration file, `cassandra.yaml`, is sufficient to explore a simple single-node `cluster`. However, anything beyond running a single-node cluster locally requires additional configuration to various Cassandra configuration files. Some examples that require non-default configuration are deploying a multi-node cluster or using clients that are not running on a cluster node.

- `cassandra.yaml`: the main configuration file for Cassandra
- `cassandra-env.sh`: environment variables can be set
- `cassandra-rackdc.properties` OR `cassandra-topology.properties`: set rack and datacenter information for a cluster
- `logback.xml`: logging configuration including logging levels
- `jvm-*`: a number of JVM configuration files for both the server and clients
- `commitlog_archiving.properties`: set archiving parameters for the `commitlog`

The sample configuration files can also be found in `./conf`:

- `cqlshrc.sample`: how the CQL shell, cqlsh, can be configured

Main runtime properties

Configuring Cassandra is done by setting yaml properties in the `cassandra.yaml` file. At a minimum you should consider setting the following properties:

- `cluster_name`: Set the name of your cluster.
- `seeds`: A comma separated list of the IP addresses of your cluster `seed nodes`.
- `storage_port`: Check that you don't have the default port of 7000 blocked by a firewall.
- `listen_address`: The `listen address` is the IP address of a node that allows it to communicate with other nodes in the cluster. Set to localhost by default. Alternatively, you can set `listen_interface` to tell Cassandra which interface to use, and consecutively which address to use. Set one property, not both.
- `native_transport_port`: Check that you don't have the default port of 9042 blocked by a firewall, so

that clients like cqlsh can communicate with Cassandra on this port.

Changing the location of directories

The following yaml properties control the location of directories:

- `data_file_directories`: One or more directories where data files, like `SSTables` are located.
- `commitlog_directory`: The directory where commitlog files are located.
- `saved_caches_directory`: The directory where saved caches are located.
- `hints_directory`: The directory where `hints` are located.

For performance reasons, if you have multiple disks, consider putting commitlog and data files on different disks.

Environment variables

JVM-level settings such as heap size can be set in `cassandra-env.sh`. You can add any additional JVM command line argument to the `JVM_OPTS` environment variable; when Cassandra starts, these arguments will be passed to the JVM.

Logging

The default logger is logback. By default it will log:

- **INFO** level in `system.log`
- **DEBUG** level in `debug.log`

When running in the foreground, it will also log at INFO level to the console. You can change logging properties by editing `logback.xml` or by running the nodetool setlogginglevel command.

Inserting and querying

Inserting and querying

The API for Cassandra is **`CQL`, the Cassandra Query Language**. To use CQL, you will need to connect to the cluster, using either:

- `cqlsh`, a shell for CQL
- a client driver for Cassandra
- for the adventurous, check out Apache Zeppelin, a notebook-style tool

CQLSH

`cqlsh` is a command-line shell for interacting with Cassandra using CQL. It is shipped with every Cassandra package, and can be found in the `bin` directory alongside the `cassandra` executable. It connects to the single node specified on the command line. For example:

```
$ bin/cqlsh localhost
```

```
Connected to Test Cluster at localhost:9042.
[cqlsh 5.0.1 | Cassandra 3.8 | CQL spec 3.4.2 | Native protocol v4]
Use HELP for help.
cqlsh> SELECT cluster_name, listen_address FROM system.local;

 cluster_name | listen_address
--------------+----------------
 Test Cluster |      127.0.0.1

(1 rows)
cqlsh>
```

If the command is used without specifying a node, `localhost` is the default. See the **`cqlsh` section** for full documentation.

Client drivers

A lot of **client drivers** are provided by the Community and a list of known drivers is provided. You should refer to the documentation of each driver for more information.

Client drivers

Client drivers

Here are known Cassandra client drivers organized by language. Before choosing a driver, you should verify the Cassandra version and functionality supported by a specific driver.

Java

- Achilles
- Astyanax
- Casser
- Datastax Java driver
- Kundera
- PlayORM

Python

- Datastax Python driver

Ruby

- Datastax Ruby driver

C# / .NET

- Cassandra Sharp
- Datastax C# driver
- Fluent Cassandra

Nodejs

- Datastax Nodejs driver

PHP

- CQL | PHP

- Datastax PHP driver
- PHP-Cassandra
- PHP Library for Cassandra

C++

- Datastax C++ driver
- libQTCassandra

Scala

- Datastax Spark connector
- Phantom
- Quill

Clojure

- Alia
- Cassaforte
- Hayt

Erlang

- CQerl
- Erlcass

Go

- CQLc
- Gocassa
- GoCQL

Haskell

- Cassy

Rust

- Rust CQL

Perl

- Cassandra::Client and DBD::Cassandra

Elixir

- Xandra
- CQEx

Dart

- dart_cassandra_cql

Production recommendations

Production recommendations

The `cassandra.yaml` and `jvm.options` files have a number of notes and recommendations for production usage. This page expands on some of the information in the files.

Tokens

Using more than one token-range per node is referred to as virtual nodes, or vnodes. `vnodes` facilitate flexible expansion with more streaming peers when a new node bootstraps into a cluster. Limiting the negative impact of streaming (I/O and CPU overhead) enables incremental cluster expansion. However, more tokens leads to sharing data with more peers, and results in decreased availability. These two factors must be balanced based on a cluster's characteristic reads and writes. To learn more, Cassandra Availability in Virtual Nodes, Joseph Lynch and Josh Snyder is recommended reading.

Change the number of tokens using the setting in the `cassandra.yaml` file:

```
num_tokens: 16
```

Here are the most common token counts with a brief explanation of when and why you would use each one.

Token Count	Description
1	Maximum availablility, maximum cluster size, fewest peers, but inflexible expansion. Must always double size of cluster to expand and remain balanced.
4	A healthy mix of elasticity and availability. Recommended for clusters which will eventually reach over 30 nodes. Requires adding approximately 20% more nodes to remain balanced. Shrinking a cluster may result in cluster imbalance.
8	Using 8 vnodes distributes the workload between systems with a ~10% variance and has minimal impact on performance.
16	Best for heavily elastic clusters which expand and shrink regularly, but may have issues availability with larger clusters. Not recommended for clusters over 50 nodes.

In addition to setting the token count, it's extremely important that `allocate_tokens_for_local_replication_factor` in `cassandra.yaml` is set to an appropriate number of replicates, to ensure even token allocation.

Read ahead

Read ahead is an operating system feature that attempts to keep as much data as possible loaded in the page cache. Spinning disks can have long seek times causing high latency, so additional throughput on reads using page cache can improve performance. By leveraging read ahead, the OS can pull additional data into memory without the cost of additional seeks. This method works well when the available RAM is greater than the size of the hot dataset, but can be problematic when the reverse is true (dataset > RAM). The larger the hot dataset, the less read ahead is useful.

Read ahead is definitely not useful in the following cases:

- Small partitions, such as tables with a single partition key
- Solid state drives (SSDs)

Read ahead can actually increase disk usage, and in some cases result in as much as a 5x latency and throughput performance penalty. Read-heavy, key/value tables with small (under 1KB) rows are especially prone to this problem.

The recommended read ahead settings are:

Hardware	Initial Recommendation
Spinning Disks	64KB
SSD	4KB

Read ahead can be adjusted on Linux systems using the `blockdev` tool.

For example, set the read ahead of the disk `/dev/sda1` to 4KB:

```
$ blockdev --setra 8 /dev/sda1
```

NOTE The `blockdev` setting sets the number of 512 byte sectors to read ahead. The argument of 8 above is equivalent to 4KB, or 8 * 512 bytes.

All systems are different, so use these recommendations as a starting point and tune, based on your SLA and throughput requirements. To understand how read ahead impacts disk resource usage, we recommend carefully reading through the **Diving Deep, using external tools** section.

Compression

Compressed data is stored by compressing fixed-size byte buffers and writing the data to disk. The buffer size is determined by the `chunk_length_in_kb` element in the compression map of a table's schema settings for `WITH COMPRESSION`. The default setting is 16KB starting with Cassandra {40_version}.

Since the entire compressed buffer must be read off-disk, using a compression chunk length that is too large can lead to significant overhead when reading small records. Combined with the default read ahead setting, the result can be massive read amplification for certain workloads. Therefore, picking an appropriate value for this setting is important.

LZ4Compressor is the default and recommended compression algorithm. If you need additional information on compression, read The Last Pickle blogpost on compression performance.

Compaction

There are different **compaction** strategies available for different workloads. We recommend reading about the different strategies to understand which is the best for your environment. Different tables may, and frequently do use different compaction strategies in the same cluster.

Encryption

It is significantly better to set up peer-to-peer encryption and client server encryption when setting up your production cluster. Setting it up after the cluster is serving production traffic is challenging to do correctly. If you ever plan to use network encryption of any type, we recommend setting it up when initially configuring your cluster. Changing these configurations later is not impossible, but mistakes can result in downtime or data loss.

Ensure keyspaces are created with NetworkTopologyStrategy

Production clusters should never use `SimpleStrategy`. Production keyspaces should use the `NetworkTopologyStrategy` (NTS). For example:

```
CREATE KEYSPACE mykeyspace WITH replication =     {
   'class': 'NetworkTopologyStrategy',
   'datacenter1': 3
};
```

Cassandra clusters initialized with `NetworkTopologyStrategy` can take advantage of the ability to configure multiple racks and data centers.

Configure racks and snitch

Correctly configuring or changing racks after a cluster has been provisioned is an unsupported process. Migrating from a single rack to multiple racks is also unsupported and can result in data loss. Using `GossipingPropertyFileSnitch` is the most flexible solution for on-premise or mixed cloud environments. `Ec2Snitch` is reliable for AWS EC2 only environments.

Getting started with mTLS authenticators

Getting started with mTLS authenticators

When a certificate based authentication protocol like TLS is used for client and Internode connections, `MutualTlsAuthenticator` & `MutualTlsInternodeAuthenticator` can be used for the authentication by leveraging the client certificates from the SSL handshake.

After SSL handshake, identity from the client certificates is extracted and only authorized users will be granted access.

What is an Identity

Operators can define their own identity for certificates by extracting some fields or information from the certificates. Implementing the interface `MutualTlsCertificateValidator` supports validating & extracting identities from the certificates that can be used by `MutualTlsAuthenticator` and `MutualTlsInternodeAuthenticator` to customize for the certificate conventions used in the deployment environment.

There is a default implementation of `MutualTlsCertificateValidator` with SPIFFE as the identity of the certificates.This requires spiffe to be present in the SAN of the certificate.

Instead of using `SPIFFE` based validator, a custom `CN` based validator that implements `MutualTlsCertificateValidator` could be configured by the operator if required.

Configuring mTLS authenticator for client connections

Note that the following steps uses SPIFFE identity as an example, If you are using a custom validator, use appropriate identity in place of `spiffe://testdomain.com/testIdentifier/testValue`.

STEP 1: Add authorized users to system_auth.identity_to_roles table

Note that only users with permissions to create/modify roles can add/remove identities. Client certificates with the identities in this table will be trusted by C*.

```
ADD IDENTITY 'spiffe://testdomain.com/testIdentifier/testValue' TO ROLE 'read_only_user'
```

STEP 2: Configure Cassandra.yaml with right properties

client_encryption_options configuration for mTLS connections

```
client_encryption_options:
  enabled: true
  optional: false
  keystore: conf/.keystore
  keystore_password: cassandra
  truststore: conf/.truststore
  truststore_password: cassandra
  require_client_auth: true // to enable mTLS
```

Configure mTLS authenticator and the validator for client connections . If you are implementing a custom validator, use that instead of Spiffe validator

```
authenticator:
  class_name : org.apache.cassandra.auth.MutualTlsAuthenticator
  parameters :
    validator_class_name: org.apache.cassandra.auth.SpiffeCertificateValidator
```

STEP 3: Bounce the cluster

After the bounce, C* will accept mTLS connections from the clients and if their identity is present in the identity_to_roles table, access will be granted.

Configuring mTLS authenticator for Internode connections

Internode authenticator trusts certificates whose identities are present in internode_authenticator.parameters.trusted_peer_identities if configured.

Otherwise, it trusts connections which have the same identity as the node. When a node is making an outbound connection to another node, it uses the certificate configured in server_encryption_options.outbound_keystore. During the start of the node, identity is extracted from the outbound keystore and connections from other nodes who have the same identity will be trusted if trusted_peer_identities is not configured.

For example, if a node has testIdentity embedded in the certificate in outbound keystore, It trusts connections from other nodes when their certificates have testIdentity embedded in them.

There is an optional configuration node_identity that can be used to verify identity extracted from the keystore to avoid any configuration errors.

STEP 1: Configure server_encryption_options in cassandra.yaml

```
server_encryption_options:
  internode_encryption: all
  optional: true
  keystore: conf/.keystore
  keystore_password: cassandra
  outbound_keystore: conf/.outbound_keystore
  outbound_keystore_password: cassandra
  require_client_auth: true  // for enabling mTLS
  truststore: conf/.truststore
  truststore_password: cassandra
```

STEP 2: Configure Internode Authenticator and Validator

Configure mTLS Internode authenticator and validator. If you are implementing a custom validator, use that instead of Spiffe validator

```
internode_authenticator:
  class_name : org.apache.cassandra.auth.MutualTlsInternodeAuthenticator
  parameters :
    validator_class_name: org.apache.cassandra.auth.SpiffeCertificateValidator
    trusted_peer_identities : "spiffe1,spiffe2"
```

STEP 3: Bounce the cluster Once all nodes in the cluster are restarted, all internode communications will be authenticated by mTLS.

Migration from existing password based authentication

- For client connections, since the migration will not happen overnight, the operators can run cassandra in optional mTLS mode and use `MutualTlsWithPasswordFallbackAuthenticator` which will accept both mTLS & password based connections, based on the type of connection client is making. These settings can be configured in `cassandra.yaml`. Once all the clients migrate to using mTLS, turn off optional mode and set the authenticator to be `MutualTlsAuthenticator`. From that point only mTLS client connections will be accepted.
- For Internode connections, while doing rolling upgrades from non-mTLS based configuration to mTLS based configuration, set `server_encryption_options.optional:true` for the new nodes to be able to connect to old nodes which are still using non-mTLS based configuration during upgrade. After this, change the internode authenticator to be `MutualTlsInternodeAuthenticator` and turn off the optional mode by setting `server_encryption_options.optional:false`.