FREE eBook

LEARNING cassandra

Free unaffiliated eBook created from
Stack Overflow contributors.

#cassandra

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: cassandra

It is an unofficial and free cassandra ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official cassandra.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with cassandra

Remarks

The Apache Cassandra database is the right choice when you need scalability and high availability without compromising performance. Linear scalability and proven fault-tolerance on commodity hardware or cloud infrastructure make it the perfect platform for mission-critical data. Cassandra's support for replicating across multiple datacenters is best-in-class, providing lower latency for your users and the peace of mind of knowing that you can survive regional outages.

PROVEN

Cassandra is in use at Constant Contact, CERN, Comcast, eBay, GitHub, GoDaddy, Hulu, Instagram, Intuit, Netflix, Reddit, The Weather Channel, and over 1500 more companies that have large, active data sets.

FAULT TOLERANT

Data is automatically replicated to multiple nodes for fault-tolerance. Replication across multiple data centers is supported. Failed nodes can be replaced with no downtime.

PERFORMANT

Cassandra consistently outperforms popular NoSQL alternatives in benchmarks and real applications, primarily because of fundamental architectural choices.

DECENTRALIZED

There are no single points of failure. There are no network bottlenecks. Every node in the cluster is identical.

SCALABLE

Some of the largest production deployments include Apple's, with over 75,000 nodes storing over 10 PB of data, Netflix (2,500 nodes, 420 TB, over 1 trillion requests per day), Chinese search engine Easou (270 nodes, 300 TB, over 800 million reqests per day), and eBay (over 100 nodes, 250 TB).

DURABLE

Cassandra is suitable for applications that can't afford to lose data, even when an entire data center goes down.

YOU'RE IN CONTROL

Choose between synchronous or asynchronous replication for each update. Highly available asynchronous operations are optimized with features like Hinted Handoff and Read Repair.

ELASTIC

Read and write throughput both increase linearly as new machines are added, with no downtime or interruption to applications.

PROFESSIONALLY SUPPORTED

Cassandra support contracts and services are available from third parties.

Versions

Version	Release Date
1.1.12	2013-11-19
1.1.9	2013-02-11
1.2.12	2013-11-28
1.2.13	2013-12-19
1.2.15	2014-02-19
1.2.16	2014-04-22
1.2.17	2014-06-25
1.2.18	2014-07-04
1.2.19	2014-11-14
1.2.6	2013-07-02
1.2.8	2013-07-27
2.0.10	2014-08-12
2.0.11	2014-10-17
2.0.12	2015-01-14
2.0.13	2015-03-20
2.0.14	2015-04-01
2.0.15	2015-06-01
2.0.16	2015-07-08
2.0.17	2015-09-18

Version	Release Date
2.0.5	2014-02-13
2.0.6	2014-04-02
2.0.7	2014-04-24
2.0.8	2014-06-13
2.0.9	2014-07-22
2.1.11	2015-10-12
2.1.12	2015-10-22
2.1.2	2014-11-20
2.1.3	2015-03-03
2.1.4	2015-04-01
2.1.5	2015-03-31
2.1.6	2015-06-09
2.1.7	2015-06-18
2.1.8	2015-07-03
2.1.9	2015-09-03
2.2.0	2015-05-14
2.2.0-beta1	2015-05-19
2.2.0-rc1	2015-06-04
2.2.0-rc2	2015-06-30
2.2.1	2015-08-25
2.2.2	2015-09-25
2.2.3	2015-10-12
2.2.4	2015-12-02
3.0.0	2015-01-26
3.0.0-alpha	2015-07-29

Version	Release Date
3.0.0-alpha1	2015-07-18
3.0.0-beta1	2015-07-10
3.0.0-beta2	2015-09-04
3.0.0-rc1	2015-07-16
3.0.0-rc2	2015-10-16
3.0.1	2015-12-04
3.0.2	2016-01-21
3.0.3	2015-11-24
3.0.4	2016-02-05
3.0.5	2016-04-02
3.0.6	2016-03-31
3.0.7	2016-05-24
3.0.8	2016-05-25
3.2.819	2016-01-05
3.4.950	2016-03-08
3.6.1076	2016-05-02
3.8.1199	2016-06-27
3.10.3004	2016-08-10

(Got this using a bit of awk: `git log --tags --simplify-by-decoration --pretty="format:%ai %d" |egrep "\(tag: [0-9]"| awk -F" " '{ print $1 " " $5}'|awk -F"." '{print $1 "." $2 "." $3}'| awk -F" " '{print $2 " |" $1}'| sed 's/)//'|sed 's/,//'| sort -n|sort -u -t" " -k1,1 | awk '{print "|" $0 "|"}'`)

Examples

Installation or Setup

Single node Installation

1. Pre-install NodeJS, Python and Java

2. Select your installation document based on your platform http://docs.datastax.com/en/cassandra/3.x/cassandra/install/installTOC.html
3. Download Cassandra binaries from http://cassandra.apache.org/download/
4. Untar the downloaded file to `<installation location>`
5. Start the cassandra using `<installation location>/bin/cassandra` OR start Cassandra as a service - `[sudo] service cassandra start`
6. Check whether cassandra is up and running using `<installation location>/bin/nodetool status`.

Ex:

1. On Windows environment run `cassandra.bat` file to start Cassandra server and `cqlsh.bat` to open CQL client terminal to execute CQL commands.

There are two ways that installation for a **Single Node** can be carried out.

You should have Oracle Java 8 or OpenJDk 8 (preferred for Cassandra versions > 3.0)

1. Installing a Debian package (installs Cassandra as a service)

Add the Cassandra version to the repository (replace the 22x with your own version for example for 2.7 use 27x)

```
echo "deb-src http://www.apache.org/dist/cassandra/debian 22x main" | sudo tee -a
/etc/apt/sources.list.d/cassandra.sources.list
# Update the repository
sudo apt-get update
# Then install it
sudo apt-get install cassandra cassandra-tools
```

Now Cassandra can be started and stopped using:

```
sudo service cassandra start
sudo service cassandra stop
```

Check the status using:

```
nodetool status
```

Logs and Data directories are `/var/log/cassandra` and `/var/lib/cassandra` respectively.

2. Installing any version of Cassandra in form of binary tarball (installs Cassandra as a standalone process)

Download the Datastax version:

```
curl -L  http://downloads.datastax.com/community/dsc-cassandra-version_number-bin.tar.gz | tar
xz
```

Or Apache Cassandra binary tarball manually (from the site http://www.apache.org/dist/cassandra/)

Now untar this:

```
tar -xvzf dsc-cassandra-version_number-bin.tar.gz
```

Change the directory to install location:

```
cd install_location
```

Start Cassandra using:

```
sudo sh ./bin/cassandra
```

Stop using:

```
sudo kill -9 pid
```

Check:

```
./bin/nodetool status
```

And viola, you have a single-node test cluster for Cassandra. So just use `cqlsh` in the terminal for Cassandra shell.

Configuration of Cassandra can be done in `cassandra.yaml` in `conf` folder in `install_location`.

Multi node installation

Multi DC Cluster Installation

Read Getting started with cassandra online: https://riptutorial.com/cassandra/topic/1682/getting-started-with-cassandra

Chapter 2: Cassandra - PHP

Examples

Simple console application

Download the Datastax PHP driver for Apache Cassandra from the Git project site, and follow the installation instructions.

We will be using the **"users"** table from the **KillrVideo** app and the Datastax PHP driver. Once you have Cassandra up and running, create the following keyspace and table:

```
//Create the keyspace
    CREATE KEYSPACE killrvideo WITH REPLICATION =
    { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };

//Use the keyspace
USE killrvideo;

// Users keyed by id
CREATE TABLE users (
    userid uuid,
    firstname text,
    lastname text,
    email text,
    created_date timestamp,
    PRIMARY KEY (userid)
);
```

Create a PHP file with your favorite editor. First, we need to connect to our **“killrvideo”** keyspace and cluster:

```
build();
$keyspace  = 'killrvideo';
$session  = $cluster->connect($keyspace);
```

Let’s insert a user into the **“users”** table:

```
execute(new Cassandra\SimpleStatement(
    "INSERT INTO users (userid, created_date, email, firstname, lastname)
      VALUES (14c532ac-f5ae-479a-9d0a-36604732e01d, '2013-01-01 00:00:00',
      'luke@example.com','Luke','Tillman')"
  ));
```

Using the Datastax PHP Cassandra driver, we can query the user by userid:

```
execute(new Cassandra\SimpleStatement
          ("SELECT firstname, lastname, email FROM killrvideo.users
            WHERE userid=14c532ac-f5ae-479a-9d0a-36604732e01d"));

foreach ($result as $row) {
```

```
    printf("user: \"%s\" \"%s\" email: \"%s\" \n", $row['firstname'],
    $row['lastname'], $row['email']);
  }
```

For a user to update their email address in the system:

```
execute(new Cassandra\SimpleStatement
          ("UPDATE users SET email = 'language_evangelist@example.com'
            WHERE userid = 14c532ac-f5ae-479a-9d0a-36604732e01d"));

execute(new Cassandra\SimpleStatement
          ("SELECT firstname, lastname, email FROM killrvideo.users
            WHERE userid=14c532ac-f5ae-479a-9d0a-36604732e01d"));

foreach ($result as $row) {
    printf("user: \"%s\" \"%s\" email: \"%s\" \n", $row['firstname'],
    $row['lastname'], $row['email']);
  }
```

Delete the user from the table and output all of the rows. You'll notice that the user's information no longer comes back after being deleted:

```
execute(new Cassandra\SimpleStatement
          ("DELETE FROM users WHERE userid = 14c532ac-f5ae-479a-9d0a-36604732e01d"));

execute(new Cassandra\SimpleStatement
          ("SELECT firstname, lastname, email FROM killrvideo.users
            WHERE userid=14c532ac-f5ae-479a-9d0a-36604732e01d"));

foreach ($result as $row) {
    printf("user: \"%s\" \"%s\" email: \"%s\" \n", $row['firstname'],
    $row['lastname'], $row['email']);
  }
```

The output should look something like this:

```
user: "Luke" "Tillman" email: "luke@example.com"
user: "Luke" "Tillman" email: "language_evangelist@example.com"
```

References:

Getting Started with Apache Cassandra and PHP, DataStax Academy

Read Cassandra - PHP online: https://riptutorial.com/cassandra/topic/2866/cassandra---php

Chapter 3: Cassandra as a Service

Introduction

This topic describes how to start Apache Cassandra as a service in windows and linux platforms. Remember you also start Cassandra from bin directory by running the batch or shell script.

Examples

Windows

1. Download the latest apache commons daemon from Apache Commons Project Distributions.
2. Extract the commons daemon in **<Cassandra installed directory>\bin**.
3. Rename the extracted folder as daemon.
4. Add **<Cassandra installed directory>** as **CASSANDRA_HOME** in windows environment variable.
5. Edit the **cassandra.yaml** file in **<Cassandra installed directory>\conf** and uncomment the data_file_directories, commitlog_directory, saved_cache_directory and set the absolute paths.
6. Edit **cassandra.bat** in **<Cassandra installed directory>\bin** and replace the value for the PATH_PRUNSRV as follows:

```
for 32 bit windows, set PATH_PRUNSRV=%CASSANDRA_HOME%\bin\daemon\
for 64 bit windows, set PATH_PRUNSRV=%CASSANDRA_HOME%\bin\daemon\amd64\
```

7. Edit **cassandra.bat** and configure SERVICE_JVM for required service name.

```
SERVICE_JVM="cassandra"
```

8. With administrator privileges, run cassandra.bat install from command prompt.

Linux

1. Create the /etc/init.d/cassandra startup script.
2. Edit the contents of the file:

```
#!/bin/sh
#
# chkconfig: - 80 45
# description: Starts and stops Cassandra
# update daemon path to point to the cassandra executable
```

```
DAEMON=<Cassandra installed directory>/bin/cassandra
start() {
        echo -n "Starting Cassandra... "
        $DAEMON -p /var/run/cassandra.pid
        echo "OK"
        return 0
}
stop() {
        echo -n "Stopping Cassandra... "
        kill $(cat /var/run/cassandra.pid)
        echo "OK"
        return 0
}
case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  restart)
        stop
        start
        ;;
  *)
        echo $"Usage: $0 {start|stop|restart}"
        exit 1
esac
exit $?
```

3. Make the file executable:

   sudo chmod +x /etc/init.d/cassandra

4. Add the new service to the list:

   sudo chkconfig --add cassandra

5. Now you can manage the service from the command line:

```
sudo /etc/init.d/cassandra start
sudo /etc/init.d/cassandra stop
sudo /etc/init.d/cassandra restart
```

Read Cassandra as a Service online: https://riptutorial.com/cassandra/topic/10544/cassandra-as-a-service

Chapter 4: Cassandra keys

Examples

Partition key, clustering key, primary key

Cassandra uses two kinds of keys:

- the **Partition Keys** is responsible for data distribution across nodes
- the **Clustering Key** is responsible for data sorting within a partition

A **primary key** is a combination of those to types. The vocabulary depends on the combination:

- **simple primary key**: only the partition key, composed of one column
- **composite partition key**: only the partition key, composed of multiple columns
- **compound primary key**: one partition key with one or more clustering keys.
- **composite and compound primary key**: a partition key composed of multiple columns and multiple clustering keys.

The PRIMARY KEY syntax

Declaring a key

The table creation statement should contain a `PRIMARY KEY` expression. The way you declare it is very important. In a nutshell:

```
PRIMARY KEY(partition key)
PRIMARY KEY(partition key, clustering key)
```

Additional parentheses group multiple fields into a composite partition key or declares a compound composite key.

Examples

Simple primary key:

```
PRIMARY KEY (key)
```

`key` is called the **partition key**.

(for simple primary key, it is also possible to put the `PRIMARY KEY` expression after the field, i.e. `key int PRIMARY KEY`, for example).

Compound primary key:

```
PRIMARY KEY (key_part_1, key_part_2)
```

Contrary to SQL, this does not exactly create a composite primary key. Instead, it declares `key_part_1` as the **partition key** and `key_part_2` as the **clustering key**. Any other field will also be considered part of the clustering key.

Composite+Compound primary keys:

```
PRIMARY KEY ((part_key_1, ..., part_key_n), (clust_key_1, ..., clust_key_n))
```

The first parenthese defines the **compound partition key**, the other columns are the clustering keys.

Syntax summary

- `(part_key)`
- `(part_key, clust_key)`
- `(part_key, clust_key_1, clust_key_2)`
- `(part_key, (clust_key_1, clust_key_2))`
- `((part_key_1, part_key_2), clust_key)`
- `((part_key_1, part_key_2), (clust_key_1, clust_key_2))`

Key ordering and allowed queries

The **partition key** is the **minimum specifier** needed to perform a query using a where clause.

If you declare a **composite clustering key**, the order matters.

Say you have the following primary key:

```
PRIMARY KEY((part_key1, part_key_2), (clust_key_1, clust_key_2, clust_key_3))
```

Then, the *only valid queries* use the following fields in the `where` clause:

- `part_key_1`, `part_key_2`
- `part_key_1`, `part_key_2`, `clust_key_1`
- `part_key_1`, `part_key_2`, `clust_key_1`, `clust_key_2`
- `part_key_1`, `part_key_2`, `clust_key_1`, `clust_key_2`, `clust_key_3`

Example of invalid queries are:

- `part_key_1`, `part_key_2`, `clust_key_2`
- Anything that does not contain both `part_key_1`, `part_key_2`
- ...

If you want to use `clust_key_2`, you have to also specify `clust_key_1`, and so on.

So the order in which you declare your clustering keys will have an impact on the type of queries you can do. In the opposite, the order of the partition key fields is not important, since you always

have to specify all of them in a query.

Read Cassandra keys online: https://riptutorial.com/cassandra/topic/9071/cassandra-keys

Chapter 5: Connecting to Cassandra

Remarks

The Cassandra Driver from Datastax very much mirrors the Java JDBC MySQL driver.

`Session`, `Statement`, `PreparedStatement` are present in both drivers.

The Singleton Connection is from this question and answer:
http://stackoverflow.com/a/24691456/671896

Feature wise, Cassandra 2 and 3 are identical. Cassandra 3 introduced a complete rewrite of the data storage system.

Examples

Java: Include the Cassandra DSE Driver

In your Maven project, add the following to your `pom.xml` file. The following versions are for Cassandra 3.x.

```
    <dependency>
        <groupId>com.datastax.cassandra</groupId>
        <artifactId>cassandra-driver-core</artifactId>
        <version>3.1.0</version>
    </dependency>
    <dependency>
        <groupId>com.datastax.cassandra</groupId>
        <artifactId>cassandra-driver-mapping</artifactId>
        <version>3.1.0</version>
    </dependency>
    <dependency>
        <groupId>com.datastax.cassandra</groupId>
        <artifactId>cassandra-driver-extras</artifactId>
        <version>3.1.0</version>
    </dependency>
    <dependency>
        <groupId>com.datastax.cassandra</groupId>
        <artifactId>dse-driver</artifactId>
        <version>1.1.0</version>
    </dependency>
```

Java: Connect to a Local Cassandra Instance

Connecting to Cassandra is very similar to connecting to other datasources. With Cassandra, credentials are not required.

```
String cassandraIPAddress = "127.0.0.1";
String cassandraKeyspace = "myKeyspace";
String username = "foo";
```

```
String password = "bar";

com.datastax.driver.core.Cluster cluster = Cluster.builder()
    .addContactPoint(cassandraIPAddress)
    .withCredentials(username, password) // If you have setup a username and password for your
node.
    .build();

com.datastax.driver.core.Session session = cluster.connect(cassandraKeyspace);

com.datastax.driver.core.Metadata metadata = cluster.getMetadata();

// Output Cassandra connection status
System.out.println("Connected to Cassandra cluster: " + metadata.getClusterName() + " with
Partitioner: " + metadata.getPartitioner());

// Loop through your entire Cluster.
for (Host host : metadata.getAllHosts()) {
    System.out.println("Cassandra Host Address: " + host.getAddress() + " | Is Up = " +
host.isUp());
}
```

Java: Connect Using a Singleton

```
public enum Cassandra {

    DB;

    private Session session;
    private Cluster cluster;
    private static final Logger LOGGER = LoggerFactory.getLogger(Cassandra.class);

    /**
     * Connect to the cassandra database based on the connection configuration provided.
     * Multiple call to this method will have no effects if a connection is already
established
     * @param conf the configuration for the connection
     */
    public void connect(ConnectionCfg conf) {
        if (cluster == null && session == null) {
            cluster =
Cluster.builder().withPort(conf.getPort()).withCredentials(conf.getUsername(),
conf.getPassword()).addContactPoints(conf.getSeeds()).build();
            session = cluster.connect(conf.getKeyspace());
        }
        Metadata metadata = cluster.getMetadata();
        LOGGER.info("Connected to cluster: " + metadata.getClusterName() + " with partitioner:
" + metadata.getPartitioner());
        metadata.getAllHosts().stream().forEach((host) -> {
            LOGGER.info("Cassandra datacenter: " + host.getDatacenter() + " | address: " +
host.getAddress() + " | rack: " + host.getRack());
        });
    }

    /**
     * Invalidate and close the session and connection to the cassandra database
     */
    public void shutdown() {
        LOGGER.info("Shutting down the whole cassandra cluster");
```

```
        if (null != session) {
            session.close();
        }
        if (null != cluster) {
            cluster.close();
        }
    }

    public Session getSession() {
        if (session == null) {
            throw new IllegalStateException("No connection initialized");
        }
        return session;
    }
}
```

Using the Singleton connection

```
public void cassandra() throws Exception {
    Cassandra.DB.connect();
    Cassandra.DB.getSession().execute(/* CQL | Statement | PreparedStatement */)
    Cassandra.DB.close();
}
```

Read Connecting to Cassandra online: https://riptutorial.com/cassandra/topic/7845/connecting-to-cassandra

Chapter 6: Repairs in Cassandra

Parameters

Option/Flag	Description
option	*option description*
-h	hostname. Defaults to "localhost." If you do not specify a host repair is run on the same host that the command is executed from.
-p	JMX port. The default is 7199.
-u	username. Only required if JMX security is enabled.
-pw	password. Only required if JMX security is enabled.
flag	*flag description*
-local	Only compare and stream data from nodes in the "local" data center.
-pr	"Partitioner Range" repair. Only repair the primary token range for a replica. Faster than repairing all ranges of your replicas, as it prevents repairing the same data multiple times. Note that if you use this option for repairing one node, you must also use it for the rest of your cluster, as well.
-par	Run repairs in parallel. Gets repairs done faster, but significantly restricts the cluster's ability to handle requests.
-hosts	Allows you to specify a comma-delimited list of nodes to stream your data from. Useful if you have nodes that are known to be "good." While it is documented as a valid option for Cassandra 2.1+, it also works with Cassandra 2.0.

Remarks

Cassandra Anti-Entropy Repairs:

Anti-entropy repair in Cassandra has two distinct phases. To run successful, performant repairs, it is important to understand both of them.

- **Merkle Tree calculations**: This computes the differences between the nodes and their replicas.
- **Data streaming**: Based on the outcome of the Merkle Tree calculations, data is scheduled to be streamed from one node to another. This is an attempt to synchronize the data

between replicas.

Stopping a Repair:

You can stop a repair by issuing a STOP VALIDATION command from nodetool:

```
$ nodetool stop validation
```

How do I know when repair is completed?

You can check for the first phase of repair (Merkle Tree calculations) by checking `nodetool compactionstats`.

You can check for repair streams using `nodetool netstats`. Repair streams will also be visible in your logs. You can `grep` for them in your system logs like this:

```
$ grep Entropy system.log

INFO [AntiEntropyStage:1] 2016-07-25 07:32:47,077 RepairSession.java (line 164) [repair
#70c35af0-526e-11e6-8646-8102d8573519] Received merkle tree for test_users from /192.168.14.3
INFO [AntiEntropyStage:1] 2016-07-25 07:32:47,081 RepairSession.java (line 164) [repair
#70c35af0-526e-11e6-8646-8102d8573519] Received merkle tree for test_users from /192.168.16.5
INFO [AntiEntropyStage:1] 2016-07-25 07:32:47,091 RepairSession.java (line 221) [repair
#70c35af0-526e-11e6-8646-8102d8573519] test_users is fully synced
INFO [AntiEntropySessions:4] 2016-07-25 07:32:47,091 RepairSession.java (line 282) [repair
#70c35af0-526e-11e6-8646-8102d8573519] session completed successfully
```

Active repair streams can also be monitored with this (Bash) command:

```
$ while true; do date; diff <(nodetool -h 192.168.0.1 netstats) <(sleep 5 && nodetool -h
192.168.0.1 netstats); done
```

ref: how do i know if nodetool repair is finished

How to check for stuck or orphaned repair streams?

On each node, you can monitor this with `nodetool tpstats`, and check for anything "blocked" on the "AntiEntropy" lines.

```
$ nodetool tpstats
Pool Name                    Active   Pending      Completed   Blocked  All time blocked
...
AntiEntropyStage                  0         0         854866         0                 0
...
AntiEntropySessions               0         0           2576         0                 0
...
```

Examples

Examples for running Nodetool Repair

Usage:

```
$ nodetool repair [-h | -p | -pw | -u] <flags> [ -- keyspace_name [table_name]]
```

Default Repair Option

```
$ nodetool repair
```

This command will repair the current node's primary token range (i.e. range which it owns) along with the replicas of other token ranges it has in all tables and all keyspaces on the current node:

For e.g. If you have replication factor of 3 then total of 5 nodes will be involved in repair: 2 nodes will be fixing 1 partition range 2 nodes will be fixing 2 partition ranges 1 node will be fixing 3 partition ranges. (Command was run on this node)

Repair in Parallel

```
$ nodetool repair -par
```

This command will run do perform the same task as default repair but by running the repair in parallel on the nodes containing replicas.

Repair Primary Token Range

This command repairs only the primary token range of the node in all tables and all keyspaces on the current node:

```
$ nodetool repair -pr
```

Repair only the local Data Center on which the node resides:

```
$ nodetool repair -pr -local
```

Repair only the primary range for all replicas in all tables and all keyspaces on the current node, only by streaming from the listed nodes:

```
$ nodetool repair -pr -hosts 192.168.0.2, 192.168.0.3, 192.168.0.4
```

Repair only the primary range for all replicas in the stackoverflow keyspace on the current node:

```
$ nodetool repair -pr -- stackoverflow
```

Repair only the primary range for all replicas in the test_users table of the stackoverflow keyspace on the current node:

```
$ nodetool repair -pr -- stackoverflow test_users
```

Read Repairs in Cassandra online: https://riptutorial.com/cassandra/topic/4873/repairs-in-cassandra

Chapter 7: Running Repair on Cassandra

Syntax

- **Synopsis**
- `nodetool [node-options] repair [other-options]`
- **Node options**
- `[(-h <host> | --host <host>)]`
- `[(-p <port> | --port <port>)]`
- `[(-pw <password> | --password <password>)]`
- `[(-pwf <passwordFilePath> | --password-file <passwordFilePath>)]`
- `[(-u <username> | --username <username>)]`
- **Other options**
- `[(-dc <specific_dc> | --in-dc <specific_dc>)...]`
- `[(-dcpar | --dc-parallel)]`
- `[(-et <end_token> | --end-token <end_token>)]`
- `[(-full | --full)]`
- `[(-hosts <specific_host> | --in-hosts <specific_host>)...]`
- `[(-j <job_threads> | --job-threads <job_threads>)]`
- `[(-local | --in-local-dc)]`
- `[(-pl | --pull)]`
- `[(-pr | --partitioner-range)]`
- `[(-seq | --sequential)]`
- `[(-st <start_token> | --start-token <start_token>)]`
- `[(-tr | --trace)]`
- `[--]`
- `[<keyspace> <tables>...]`

Parameters

Parameter	Details
`-dc <specific_dc>`, `--in-dc <specific_dc>`	Use `-dc` to repair specific datacenters
`-dcpar`, `--dc-parallel`	Use `-dcpar` to repair data centers in parallel.
`-et <end_token>`, `--end-token <end_token>`	Use `-et` to specify a token at which repair range ends
`-full`, `--full`	Use `-full` to issue a full repair.
`-h <host>`, `--host <host>`	Node hostname or ip address
`-hosts <specific_host>`, `--in-hosts <specific_host>`	Use `-hosts` to repair specific hosts
`-j <job_threads>`, `--job-threads <job_threads>`	Number of threads to run repair jobs. Usually this means number of CFs to repair concurrently. WARNING: increasing this puts more load on repairing nodes, so be careful. (default: 1, max: 4)
`-local`, `--in-local-dc`	Use `-local` to only repair against nodes in the same datacenter
`-p <port>`, `--port <port>`	Remote jmx agent port number
`-pl`, `--pull`	Use `--pull` to perform a one way repair where data is only streamed from a remote node to this node.
`-pr`, `--partitioner-range`	Use `-pr` to repair only the first range returned by the partitioner
`-pw <password>`, `--password <password>`	Remote jmx agent password
`-pwf <passwordFilePath>`, `--password-file <passwordFilePath>`	Path to the JMX password file
`-seq`, `--sequential`	Use `-seq` to carry out a sequential repair
`-st <start_token>`, `--start-token <start_token>`	Use `-st` to specify a token at which the repair range starts
`-tr`, `--trace`	Use `-tr` to trace the repair. Traces are logged to `system_traces.events`.
`-u <username>`, `--username <username>`	Remote jmx agent username
`--`	This option can be used to separate command-line options from the list of argument, (useful when arguments might be mistaken for command-line options

Parameter	Details
`[<keyspace> <tables>...]`	The keyspace followed by one or many tables

Examples

Running repair on Cassandra

Run repair on a particular partition range.

```
nodetool repair -pr
```

Run repair on the whole cluster.

```
nodetool repair
```

Run repair in parallel mode.

```
nodetool repair -par
```

Read Running Repair on Cassandra online: https://riptutorial.com/cassandra/topic/6556/running-repair-on-cassandra

Chapter 8: Security

Remarks

Cassandra security resources

- CQL: Database roles syntax definition
- CQL: List of object permissions
- DataStax Documentation: Internal authentication
- DataStax Documentation: Internal authorization

Examples

Configuring internal authentication

Cassandra will not require users to login using the default configuration. Instead password-less, anonymous logins are permitted for anyone able to connect to the `native_transport_port`. This behaviour can be changed by editing the `cassandra.yaml` config to use a different authenticator:

```
# Allow anonymous logins without authentication
# authenticator: AllowAllAuthenticator

# Use username/password based logins
authenticator: PasswordAuthenticator
```

The login credentials validated by `PasswordAuthenticator` will be stored in the internal `system_auth` keyspace. By default, the keyspace will not be replicated accross all nodes. You'll have to change the replication settings to make sure that Cassandra will still be able to read user credentials from local storage in case other nodes in the cluster cannot be reached, or else you might not be able to login!

For `SimpleStrategy` (where `N` is the number of nodes in your cluster):

```
ALTER KEYSPACE system_auth WITH replication = {'class': 'SimpleStrategy',
'replication_factor': N};
```

For `NetworkTopologyStrategy` (where `N` is the number of nodes in the corresponding data center):

```
ALTER KEYSPACE system_auth WITH replication =  { 'class' : 'NetworkTopologyStrategy',
'datacenter1' : N };
```

Restart each node after the changes described above. You'll now only be able to login using the default superuser:

```
cqlsh -u cassandra -p cassandra
```

(Optional) Replace default superuser with custom user

Using a default superuser with a standard password isn't much safer than using no user at all. You should create your own user instead using a safe and unique password:

```
CREATE ROLE myadminuser WITH PASSWORD = 'admin123' AND LOGIN = true AND SUPERUSER = true;
```

Log in using your new user: `cqlsh -u myadminuser -p admin123`

Now disable login for the standard cassandra user and remove the superuser status:

```
ALTER ROLE cassandra WITH LOGIN = false AND SUPERUSER = false;
```

Configuring internal authorization

By default each user will be able to access all data in Cassandra. You'll have to configuring a different authorizer in your `cassandra.yaml` to grant individual object permissions to your users.

```
# Grant all permissions to all users
# authorizer: AllowAllAuthorizer

# Use object permissions managed internally by Cassandra
authorizer: CassandraAuthorizer
```

Permissions for individual users will be store in the internal `system_auth` keyspace. You should change the replication settings in case you haven't already done so while enabling password based authentication.

For `SimpleStrategy` (where `N` is the number of nodes in your cluster):

```
ALTER KEYSPACE system_auth WITH replication = {'class': 'SimpleStrategy',
'replication_factor': N};
```

For `NetworkTopologyStrategy` (where `N` is the number of nodes in the corresponding data center):

```
ALTER KEYSPACE system_auth WITH replication =  { 'class' : 'NetworkTopologyStrategy',
'datacenter1' : N };
```

Restart each node after the changes described above. You'll now be able to set permissions using e.g. the following commands.

Grants all permissions for specified keyspace and role:

```
GRANT ALL ON KEYSPACE keyspace_name TO role_name;
```

Grant read permissions on all keyspaces:

```
GRANT SELECT ON ALL KEYSPACES TO role_name;
```

Allow execution of INSERT, UPDATE, DELETE and TRUNCATE statements on a certain

keyspace:

```
GRANT MODIFY ON KEYSPACE keyspace_name TO role_name;
```

Allow changing keyspaces, tables and indices for certain keyspace:

```
GRANT ALTER ON KEYSPACE keyspace_name TO role_name;
```

Please note that permissions will be cached for `permissions_validity_in_ms` (`cassandra.yaml`) and changes might not be effective instantly.

Read Security online: https://riptutorial.com/cassandra/topic/5646/security

Credits

S. No	Chapters	Contributors
1	Getting started with cassandra	Community, muru, perennial_noob, phact, Ravinder Matte, sourav, Stephen Leppik
2	Cassandra - PHP	Aaron, ethrbunny, Renjith VR
3	Cassandra as a Service	Shoban Sundar
4	Cassandra keys	Derlin
5	Connecting to Cassandra	geeves
6	Repairs in Cassandra	Aaron, Akshay
7	Running Repair on Cassandra	muru, Ravinder Matte
8	Security	Stefan Podkowinski