CONFIGURING APACHE CASSANDRA DATA CONSISTENCY

Let's look at how Apache Cassandra handles data consistency.

If you write data to n - 1 nodes in a cluster and then an application reads it from node n Cassandra could report that the data is not there as replication is not instant. That could be a big problem or not a problem at all. It depends on the application. If it's a revenue generating application then it's a problem.

So, Cassandra lets administrators configure data replication and consistency at the application level. The trade off is response time versus data accuracy.

This ability to configure this at the application level is called **eventual consistency**. As the name implies you can tell Cassandra to wait after an operation to write all data to all data centers and racks or move to the next operation after the first write is done with the understanding that Cassandra will eventually catch up.

This idea is the same as determining what kind of storage to rent or buy when designing an application. Applications that are deemed less mission critical might use lower cost storage and even fewer drives and make fewer replicas or no replicas at all than mission critical ones.

Consistency can be set at these levels:

- data center
- cluster
- I/O operation type (i.e., read or write)

(This article is part of our Cassandra Guide. Use the right-hand menu to navigate.)

Write and Read Consistency

Write consistency has lots of different options. Here are a couple.

ALL	Data must be written to the memtable and commit log on each node. The steps that Cassandra goes through as it writes data is to: • write data to the memtable (memory) • write to the commit log • write to disk • compact the data
QUORUM	A quorum at the data center level. A **quorum** is ((number of replicas)) / 2 + 1 rounded up. Adding 1 makes this number the next integer when that division results in a fraction. And you cannot write to a fraction of a data center, so add one more
ONE, TWO, THREE	1, 2, or 3 replicate, meaning at the node level.

Read Consistency

Depending on the write consistency set, some nodes will have older copies of data than others. So it makes sense that you can configure the read consistency as well.

ALL	Waits until all replicas have responded.
...	etc.

Replication Strategy

Replication is a factor in data consistency. A **replica** means a copy of the data.

in order to whether a write has been successful, and whether replication is working, Cassandra has an object called a **snitch**, which determines which datacenter and rack nodes belong to and the network topology.

There are two replication stations:

1. **SimpleStrategy**—one datacenter and one rack.
2. **NetworkTopologyStrategy**—multiple racks and multiple data centers. This is preferred even when there is only one data center.

Example

Here we illustrate with an example.

First follow these instructions to set up a cluster.

Use **nodetool** to determine the name of the data center as reported by the snitch. Here the name is **datacenter1**. This shows that there are two nodes in this cluster. Of courses you would run this on each node to see what data center each node belongs to.

```
nodetool status
```

```
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address       Load       Tokens       Owns    Host ID
Rack
UN  172.31.46.15  532.22 MiB  256          ?       58999e76-
cf3e-4791-8072-9257d334d945  rack1
UN  172.31.47.43  532.59 MiB  256          ?       472fd4f0-9bb3-48a3-
a933-9c9b07f7a9f6  rack1
```

Create a keyspace with **NetworkTopologyStrategy** with 1 replica to **datacenter1**. So that means write the data to two racks, since there is only one data center.

```
CREATE KEYSPACE Customers
       WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy',
'datacenter1' : 1};
```

Now create a table:

```
CREATE TABLE  Customers.Customers (
Name text PRIMARY KEY,
credit int
);
```

Put some data in it:

```
insert into Customers.Customers (name, credit) values ('Hardware Store',
1000);
```

Turn tracing on.

```
cqlsh> TRACING on;
Now Tracing is enabled
```

And write another record:

```
insert into Customers.Customers (name, credit) values ('Book Store', 1000);
```

Cassandra responds with some information on latency. It does not show rack level operations.

```
Tracing session: bf8e0a90-0c43-11e9-9628-9d6a90b241c5

Parsing insert into Customers.Customers (name, credit) values ('Book Store',
1000);

activity| timestamp | source | source_elapsed | client

 | 2018-12-30 15:01:12.761000 | 172.31.46.15 |  203 | 127.0.0.1

Preparing statement  | 2018-12-30 15:01:12.762000 | 172.31.46.15 | 359 |
```

127.0.0.1

Determining replicas for mutation | 2018-12-30 15:01:12.762000 | 172.31.46.15 | 662 | 127.0.0.1

Appending to commitlog | 2018-12-30 15:01:12.762000 | 172.31.46.15 | 753 | 127.0.0.1

Adding to customers memtable | 2018-12-30 15:01:12.762000 | 172.31.46.15 | 827 | 127.0.0.1

Request complete | 2018-12-30 15:01:12.762121 | 172.31.46.15 | 1121 | 127.0.0.1