# Cloud Computing and Big Data

# Apache Cassandra

Oxford University
Software Engineering
Programme
July 2020

# Apache Cassandra

- Masterless / Symmetric
  - Every node is equal and you can write to any node as well as read
- Shared Nothing architecture
  - Each server has its own disk
- Based on Dynamo
  - for automatic sharding and eventual consistency
- And BigTable
  - For “Column Families”
- Donated to Apache by Facebook
  - Now mostly developed by DataStax

# Cassandra Write Model

## Single Datacentre

1. Client Writes to any Cassandra Node
2. Coordinator Node replicates to nodes and Zones
3. Nodes return ack to coordinator
4. Coordinator returns ack to client
5. Data written to internal commit log disk


If a node goes offline, hinted handoff completes the write when the node comes back up.

Requests can choose to wait for one node, a quorum, or all nodes to ack the write

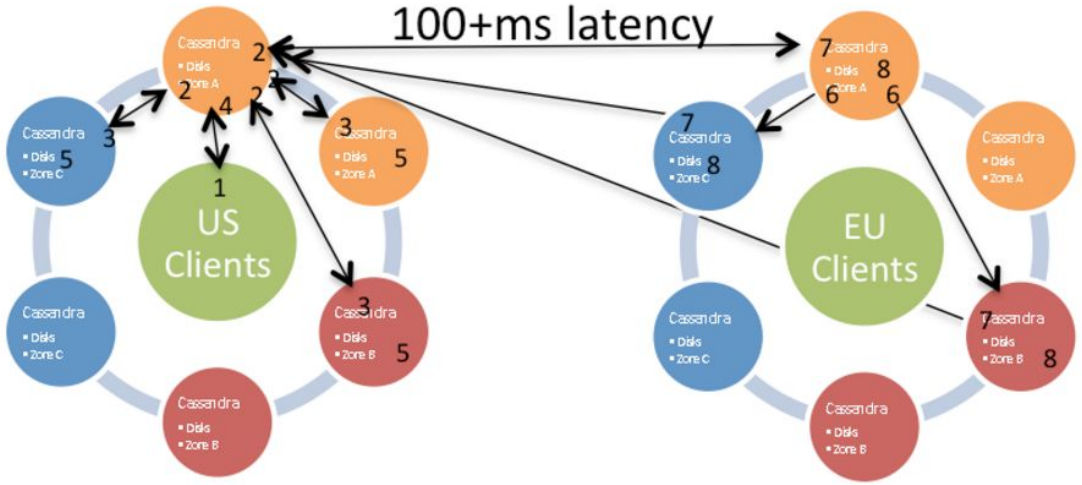
SSTable disk writes and compactions occur asynchronously

Source: Netflix

# Multi Datacentre Writes

1. Client Writes to any Cassandra Node
2. Coordinator node replicates to other nodes Zones and regions
3. Local write acks returned to coordinator
4. Client gets ack when 2 of 3 local nodes are committed
5. Data written to internal commit log disks
6. When data arrives, remote node replicates data
7. Ack direct to source region coordinator
8. Remote copies written to commit log disks

If a node or region goes offline, hinted handoff completes the write when the node comes back up. Nightly global compare and repair jobs ensure everything stays consistent.

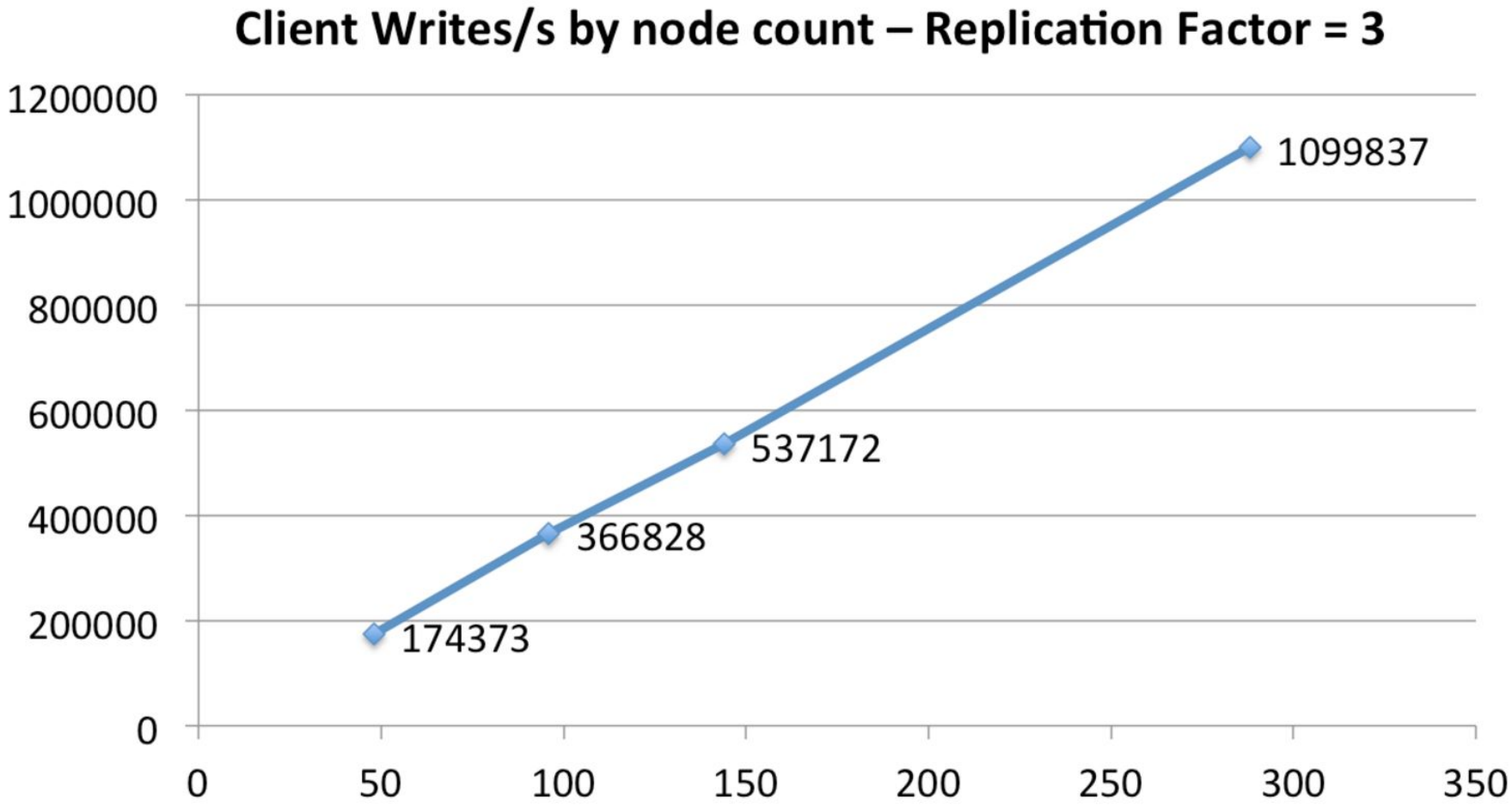





Source: Netflix

# Cassandra Scale Up

## In Amazon EC2




Source: Netflix

# The numbers

| Per Node | 48 Nodes | 96 Nodes | 144 Nodes | 288 Nodes |
|---|---|---|---|---|
| Per Server Writes/s | 10,900 w/s | 11,460 w/s | 11,900 w/s | 11,456 w/s |
| Mean Server Latency | 0.0117 ms | 0.0134 ms | 0.0148 ms | 0.0139 ms |
| Mean CPU %Busy | 74.4 % | 75.4 % | 72.5 % | 81.5 % |
| Disk Read | 5,600 KB/s | 4,590 KB/s | 4,060 KB/s | 4,280 KB/s |
| Disk Write | 12,800 KB/s | 11,590 KB/s | 10,380 KB/s | 10,080 KB/s |
| Network Read | 22,460 KB/s | 23,610 KB/s | 21,390 KB/s | 23,640 KB/s |
| Network Write | 18,600 KB/s | 19,600 KB/s | 17,810 KB/s | 19,770 KB/s |



Source: Netflix

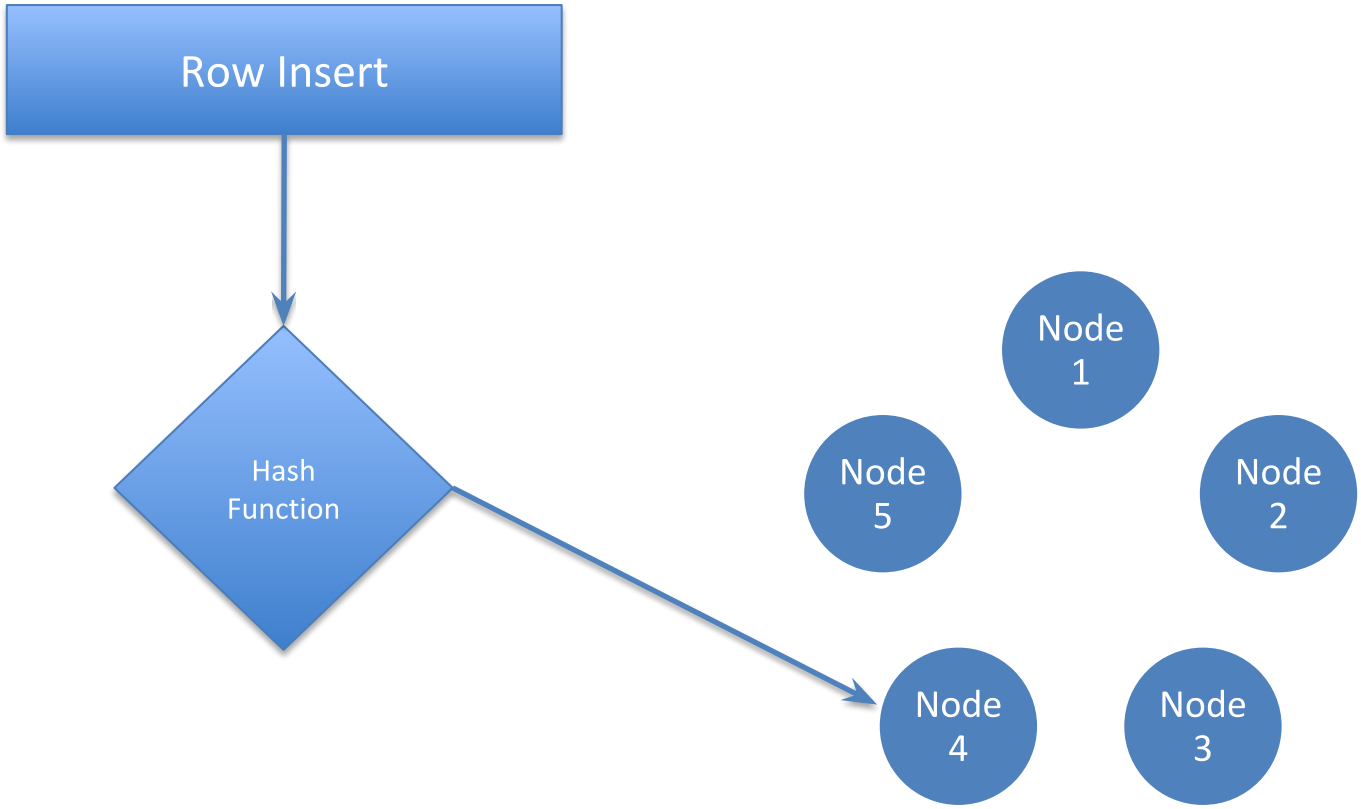
# Cassandra Model

- **Keyspaces** are roughly equivalent to SQL Databases
  - Encapsulate replication strategies
- **Column Families** roughly equivalent to SQL tables
- Generally a different approach vs SQL
  - Writes are cheap
  - Indexes are expensive
  - Normalization is not the goal

# Cassandra Model cont.

- Inserts are the same as updates
  - No read first
- Data can be marked with a Time to Live (TTL)
  - Automatically deleted
- Deletes are not instant
  - Deleted rows are marked with a tombstone
  - Eventually cleaned up
  - Can re-appear if you do not run node repair after a node failure

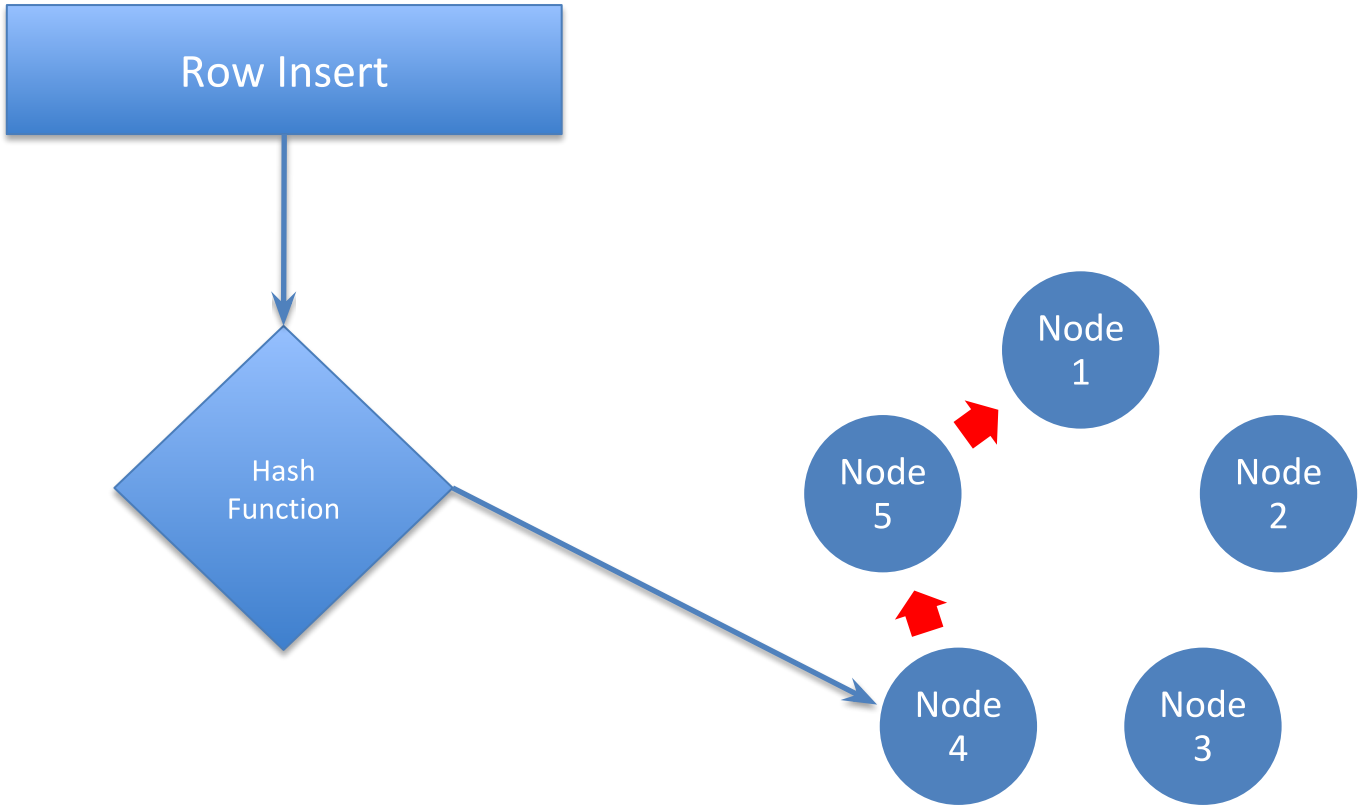
# Partitioning



# Partitioning / Hashing

- Cassandra partitions your data via a Hash function onto different nodes
  - Based on the row key
  - This can be random (MD5 hash), or specific to the data (ordered)
  - Random is recommended as it is guaranteed to be balanced
  - The latest random partitioner is the Murmur3Partitioner based on the Murmur3 hash function
    - https://en.wikipedia.org/wiki/MurmurHash
  - Specified in cassandra.yaml

# Replication



# Replication

- Each row is replicated to other servers based on the *replication factor*
  - Replication factor 1 means no copies
  - Set per keyspace
- SimpleStrategy
  - Copied onto the next *n* servers clockwise in the cluster
- NetworkTopologyStrategy
  - Tries to get onto a different rack
  - Or a different datacentre if you specify a **Replica Group**

# The snitch

- Manages the Replication
  - Simple Snitch
    - Simple replication strategy
  - Rack Inferring Snitch
    - Assumes your IP address octets define the datacentres and racks
  - Property File Snitch
    - Let's you specify your topology using a properties File
  - EC2 snitch
    - Makes calls to EC2 to understand the topology

# CQL

- A variant of SQL written specifically for Cassandra
  - The preferred model of access
  - Replaces the old “Thrift” API
- Attempts to have some compatibility with normal SQL
  - e.g. you can use either KEYSPACE or TABLE interchangeably

# CQL examples

SELECT name, occupation FROM users WHERE userid IN (199, 200, 207);

However, some queries are not permitted:

```
SELECT firstname, lastname FROM users WHERE
    birth_year = 1981 AND country = 'FR';
```

Requires a large scan of the database and cannot give a predictable time response:

ALLOW FILTERING will make this run anyway

# INSERT / UPDATE

```
INSERT INTO NerdMovies (movie, director, main_actor, year)
    VALUES ('Serenity', 'Joss Whedon', 'Nathan Fillion', 2005)
        USING TTL 86400;
```

- Every row can have a specified expiry time
- Inserts work even if the data is already there, unless you specify:

```
INSERT INTO NerdMovies (movie, director, main_actor, year)
    VALUES ('Serenity', 'Joss Whedon', 'Nathan Fillion', 2005)
    IF NOT EXISTS
        USING TTL 86400;
```

This can have unpredictable timing because it requires read-before-write

# Non-SQL data types

- Sets
  - CREATE TABLE cycling.cyclist_career_teams ( id UUID PRIMARY KEY, lastname text, teams **set**<text> );
- Lists
  - CREATE TABLE cycling.upcoming_calendar ( year int, month int, events **list**<text>, PRIMARY KEY ( year, month) );
- Maps
  - CREATE TABLE cycling.cyclist_teams ( id UUID PRIMARY KEY, lastname text, firstname text, teams **map**<int,text> );
- Tuples
  - CREATE TABLE cycling.popular (rank int PRIMARY KEY, cinfo **tuple**<text,text,int> );

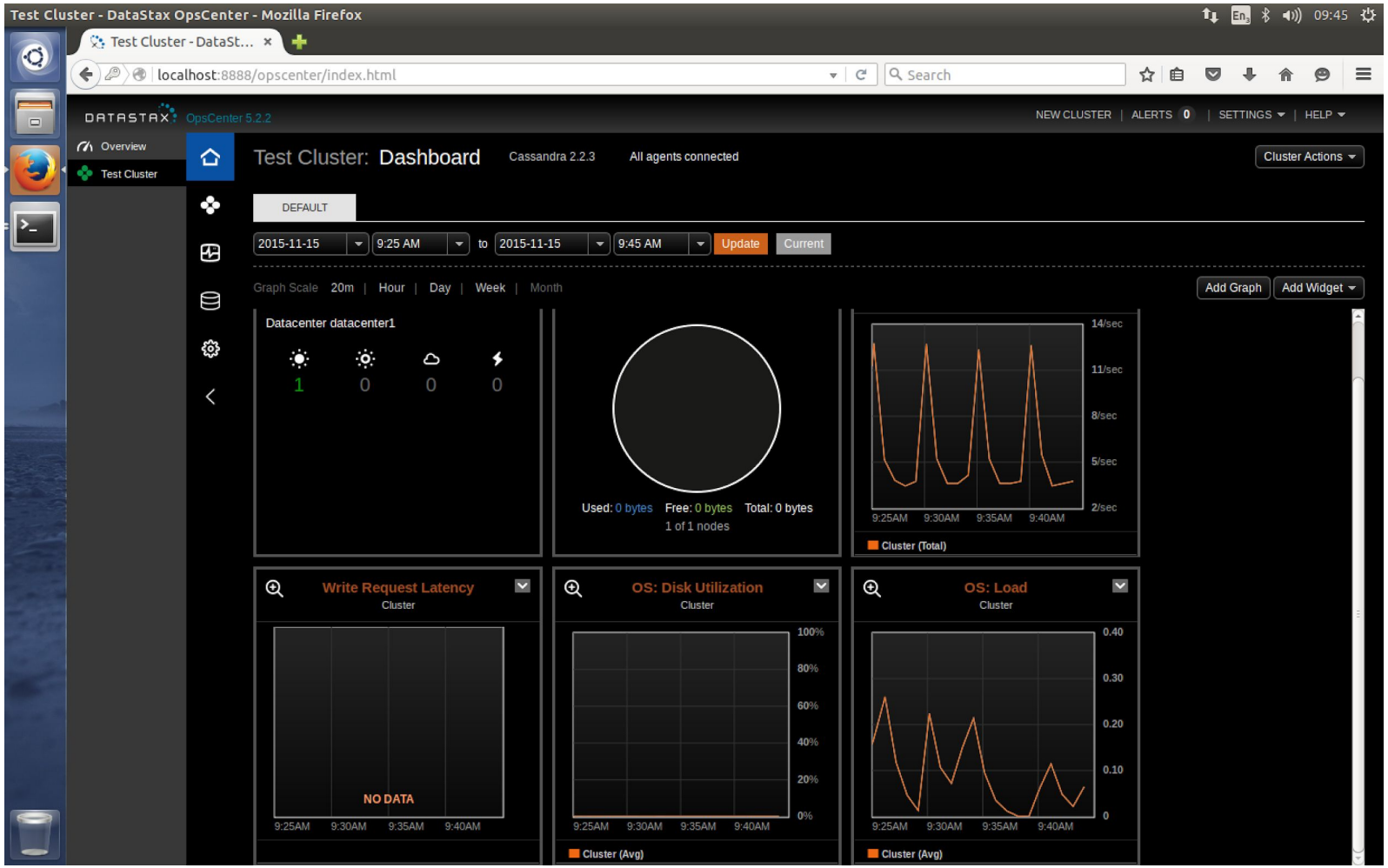
# Direct support for JSON

```
INSERT INTO cycling.cyclist_category
JSON '{
  "category" : "GC",
  "points" : 780,
  "id" :
"829aa84a-4bba-411f-a4fb-38167a987cd
a",
  "lastname" : "SUTHERLAND" }';
```

# cassandra.yaml

- Configuration of the major parts of the system
  - Datacentres, Racks, Cluster name
  - Authentication and Authorization
  - Partitioner
  - Data Storage location
  - Cacheing
  - Network topology and ports
  - Etc, etc

# DataStax OpsCenter

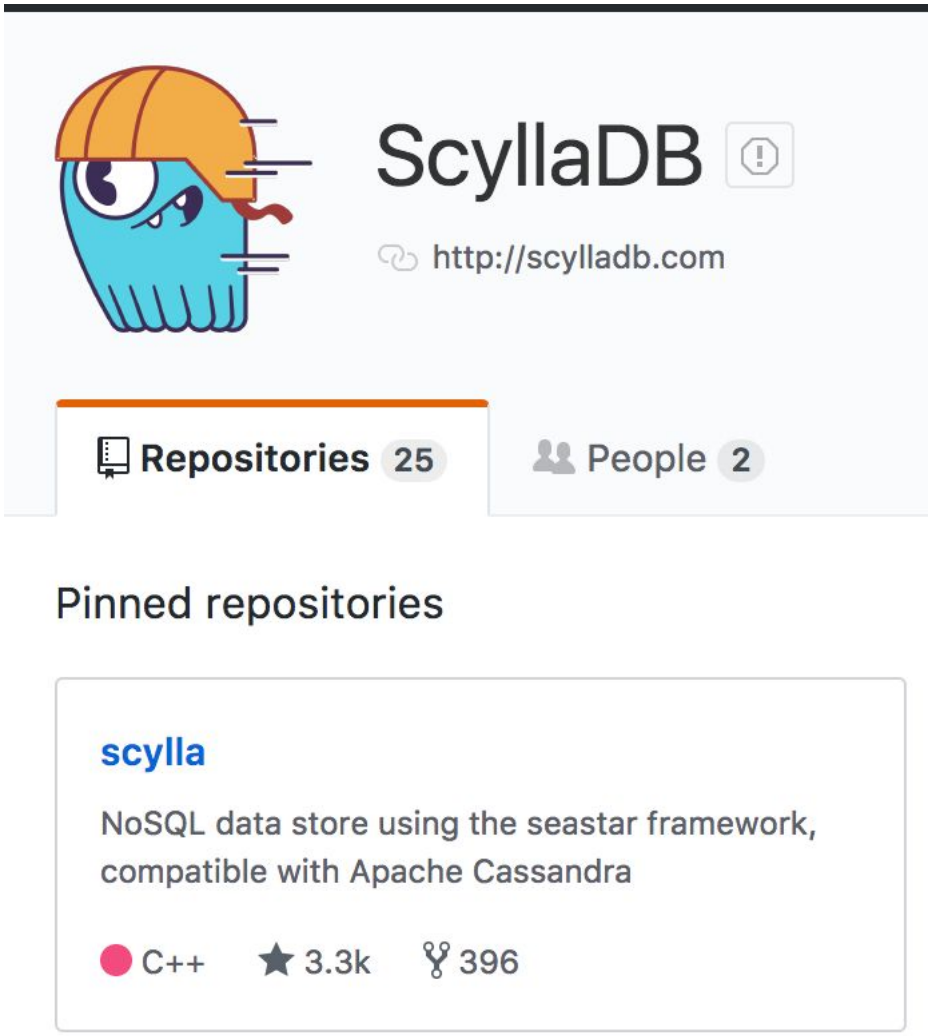


# OpsCenter

- Part of DataStax Cassandra distribution
  - Community edition has limited features
  - Enterprise edition expands these
- Not open source, but free to use in the community edition
  - Requires an agent on each Cassandra node
  - It will install this via SSH if possible

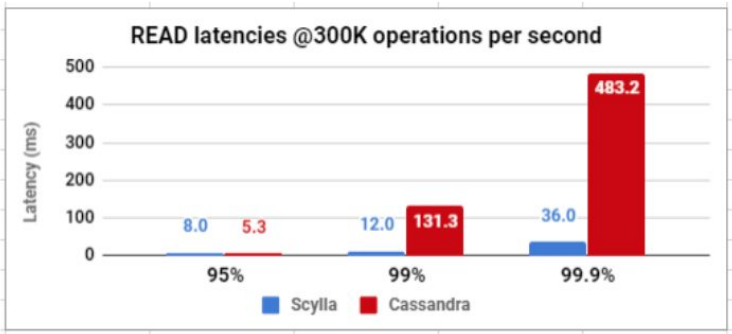
# ScyllaDB

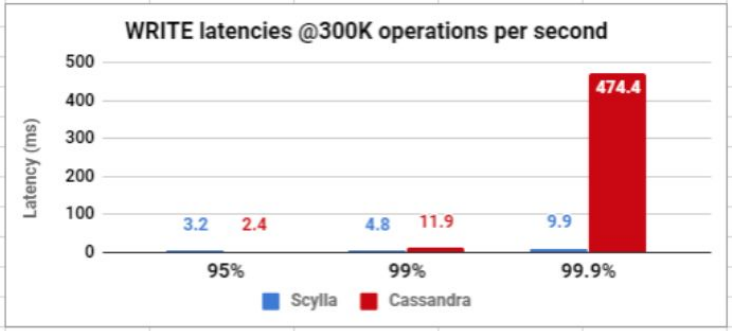
- A C++ “clone” of Cassandra
- Also Open Source
- Claims to be significantly faster

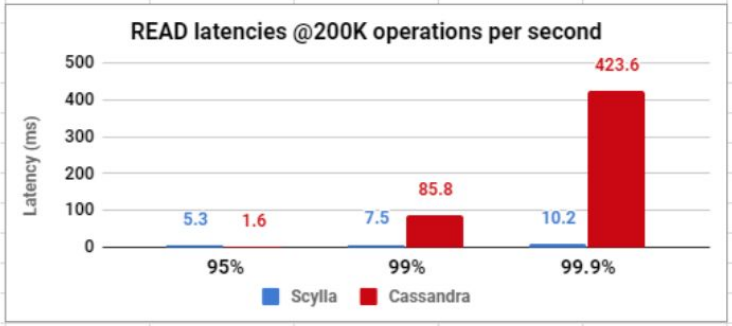


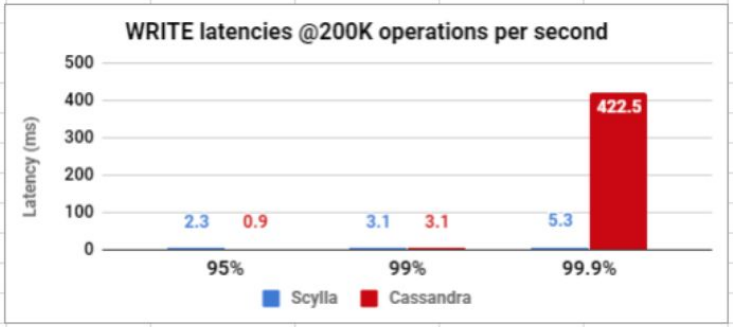
# Scylla vs Cassandra

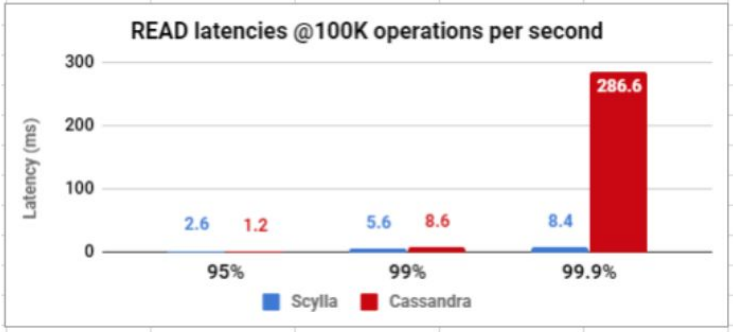
## Performance Results (Graphs)

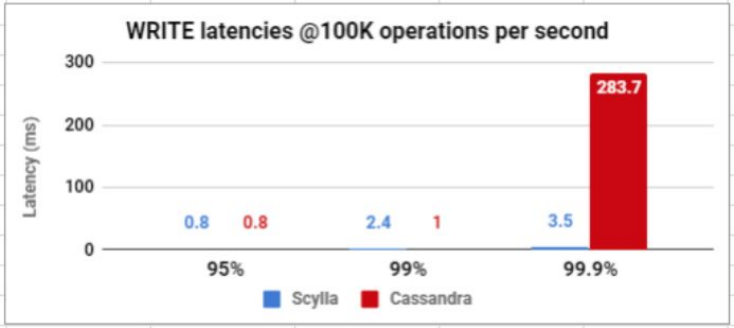


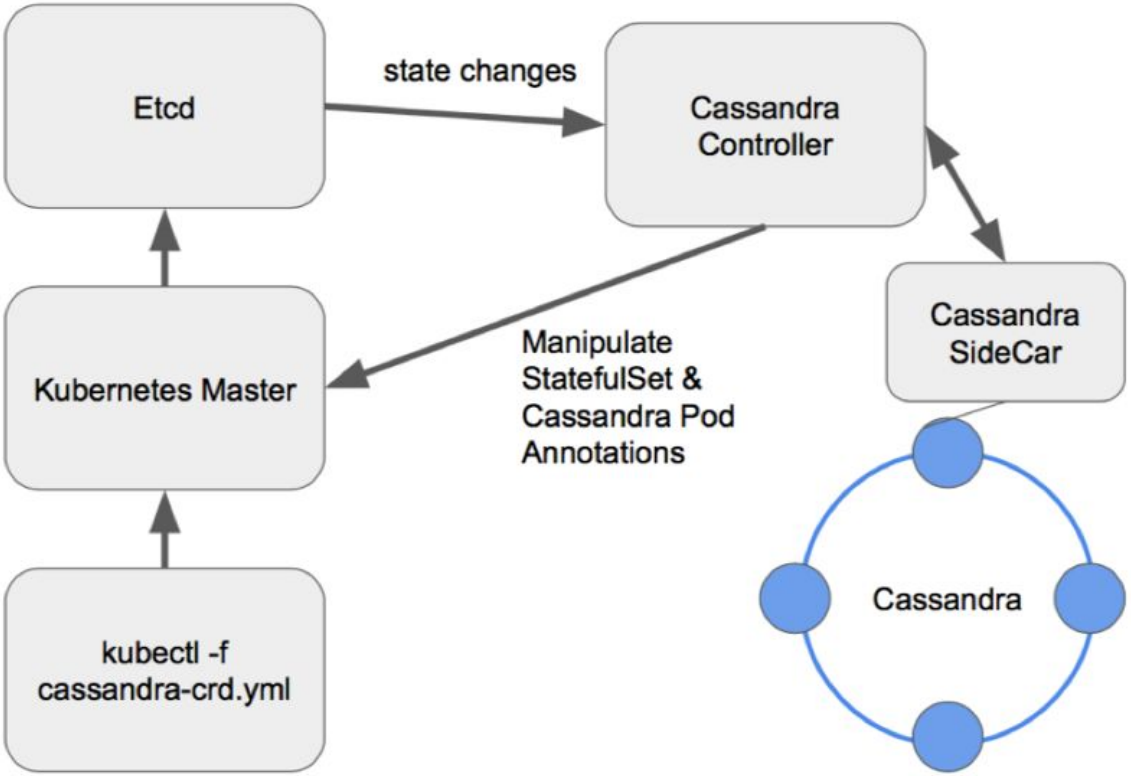



# Cassandra on Kubernetes

https://kubernetes.io/docs/tutorials/stateful-application/cassandra/

https://medium.com/flant-com/running-cassandra-in-kubernetes-challenges-and-solutions-9082045a7d93



# Questions?