greatlearning
Learning for Life

DynamoDB

amazon web services

Course 4 - Big Data Management On cloud

By
Nirmallya Mukherjee

Certified Cassandra Developer
Certified Cassandra Administrator

amazon web services | Certified
Solutions Architect - Associate

What's the challange?

Databases are doing just fine!

- Consumer oriented applications / Users in millions
- Is consumer growth predictable? Then how to do capacity planning for infrastructure?
- Data produced in hundred/thousand of TB
- Hardware failure happens no matter what
- Bottom line is we have to handle
  - V3 [Volume, Velocity, Variety]

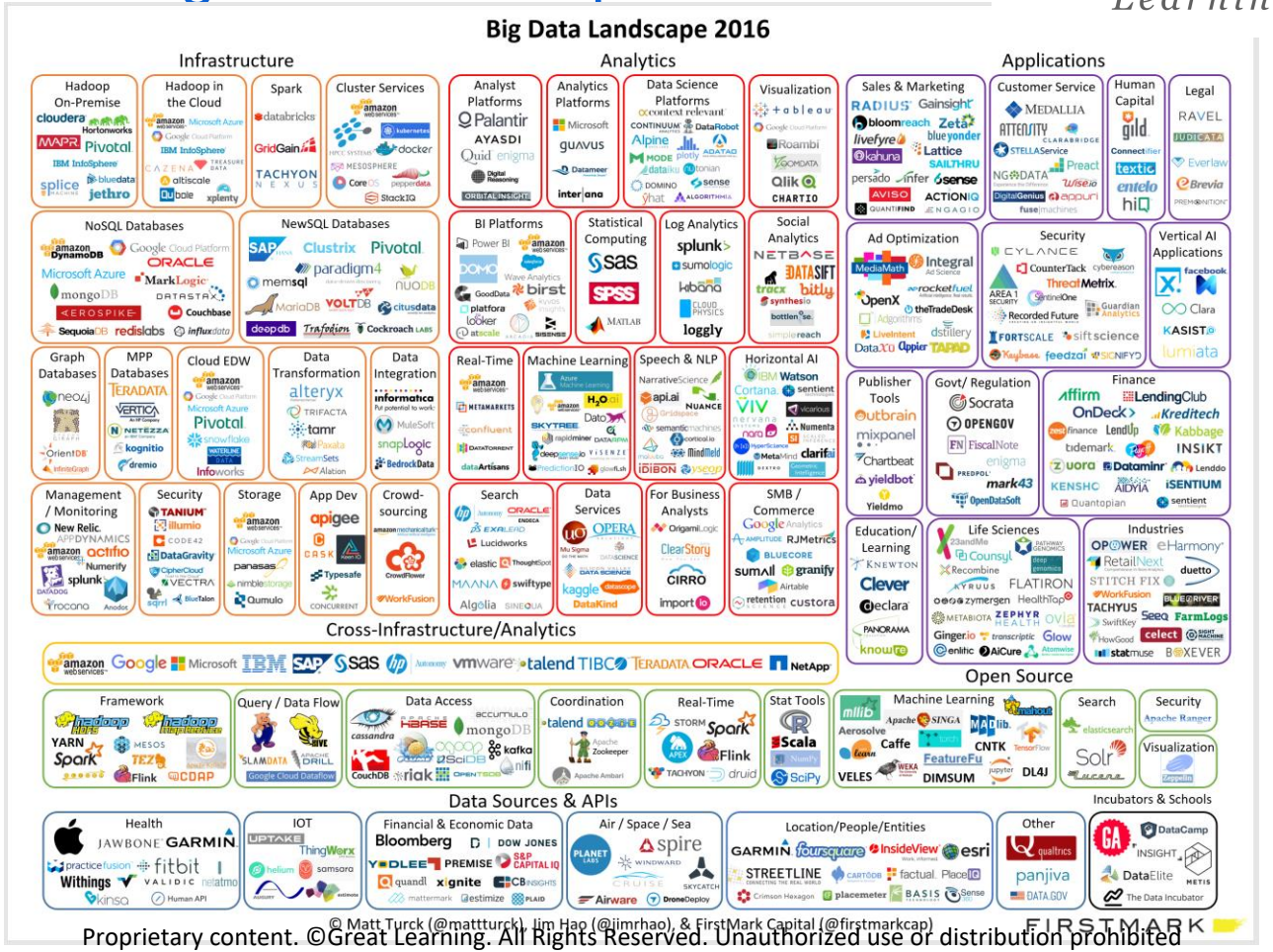
How big is big data?

• Byte of data	1 grain of rice
• Kilobyte	Fills a cup
• Megabyte	8 Bags
• Gigabytes	3 container trucks
• Terabytes	2 container ships
• Petabytes	Covers Manhattan
• Exabytes	Covers UK 3 times over
• Zettabytes	Fills the pacific ocean

Credit: David Wellman, Myraid Genetics

How BIG is big data landscape?

greatlearning
Learning for Life



© Matt Turck (@mattturck), Jim Hao (@jimrhao), & FirstMark Capital (@firstmarkcap)

Types of NoSQL

greatlearning
Learning for Life

- **Wide Row** - Also known as wide-column stores, these databases store data in rows and users are able to perform some query operations via column-based access. A wide-row store offers very high performance and a highly scalable architecture. Examples include: Cassandra, HBase, and Google BigTable.
- **Columnar** - Also known as column oriented store. Here the columns of all the rows are stored together on disk. A great fit for analytical queries because it reduces disk seek and encourages array like processing. Amazon Redshift, Google BigQuery, Teradata (with column partitioning), Druid.
- **Key/Value** - These NoSQL databases are some of the least complex as all of the data within consists of an indexed key and a value. Examples include Amazon DynamoDB, Riak, and Oracle NoSQL database
- **Document** - Expands on the basic idea of key-value stores where "documents" are more complex, in that they contain data and each document is assigned a unique key, which is used to retrieve the document. These are designed for storing, retrieving, and managing document-oriented information, also known as semi-structured data. Examples include MongoDB and CouchDB
- **Graph** - Designed for data whose relationships are well represented as a graph structure and has elements that are interconnected; with an undetermined number of relationships between them. Examples include: Neo4J, OrientDB and TitanDB

What's after NoSQL

- "NewSQL" - a class of modern relational database management systems that seek to provide the same scalable performance of NoSQL systems for online transaction processing (OLTP) read-write workloads while still maintaining the ACID guarantees of a traditional database system

- VoltDB, the most mature of these systems, combines streaming analytics, strong ACID guarantees and native clustering. This allows VoltDB to be the system-of-record for data-intensive applications, while offering an integrated high-throughput, low-latency ingestion engine. It’s a great choice for policy enforcement, fraud/anomaly detection, or other fast-decisioning apps

SQL and NoSQL

greatlearning
Learning for Life

1. Database, Relational, strict models
2. Data in rows, pre-defined schema, sql supports join
3. Vertically scalable (high data density)
4. Random access pattern support
5. Good fit for online transactional systems
6. Master slave model
7. Periodic data replication as read only copies in slave
8. Availability model includes a slight downtime in case of outages

1. Datastore, distributed & Non relational
2. Data in key value pairs, flexible schema, no joins
3. Horizontally scalable (low data density)
4. Designed for access patterns
5. Good for optimized read based system or high availability write systems
6. Seamless data replication
7. C* is masterless model
8. Masterless allows for no downtime

Applicability of SQL / NoSQL

- No one single rule and very usecase specific
  - High read oriented
  - High write oriented
  - Document based storage
  - KV based storage
- Important to know the access patterns upfront
  - Focus on modeling specific to the use case
- Very difficult to fix an improper model later on unlike a database

CAP Theorem

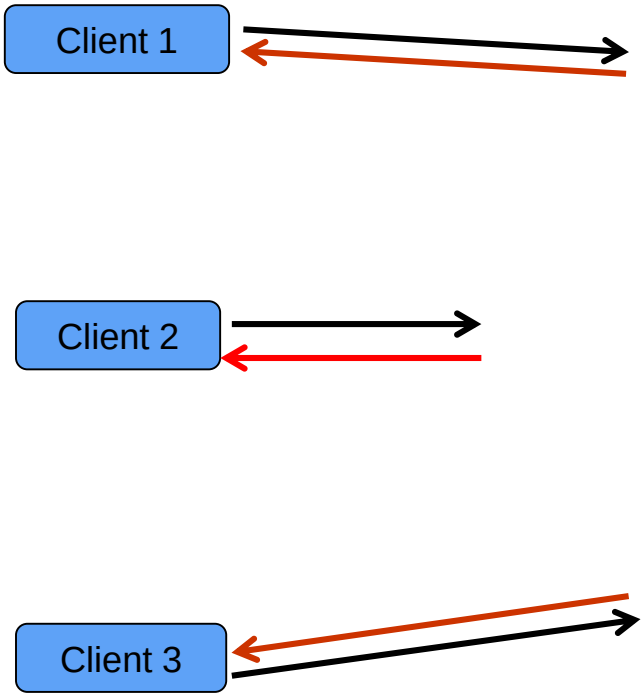
- ACID
  - **A**tomicity - Atomicity requires that each transaction be "all or nothing"
  - **C**onsistency - The consistency property ensures that any transaction will bring the database from one valid state to another
  - **I**solation - The isolation property ensures that the concurrent execution of transactions result in a system state that would be obtained if transactions were executed serially
  - **D**urability - Durability means that once a transaction has been committed, it will remain so under all circumstances

- What is it? Can I have all?
  - **C**onsistency - all replica nodes have the same data at all times
  - **A**vailability - Request must receive a response
  - **P**artition tolerance - Should run even if there is a part failure

- CAP theorem applies only to distributed systems only
  - https://en.wikipedia.org/wiki/CAP_theorem

- CAP leads to BASE theory
  - **B**asically **A**vailable, **S**oft state, **E**ventual consistency

Why large malls have more than one entrance?



Gossip

- I think we all know what this means in English! ☺
- It is a bi-directional communication mechanism among nodes and a way any node learns about the cluster and other nodes
- At the time of boostrap 3 nodes are picked at random for the first time, this is done by the seed node because a new node does not know about the cluster yet
- Runs every second and talks to upto 1 to 3 nodes at random everytime
- Passing state information (it's own + others)
  - Available / down / bootstraping
  - Data "Load" it is under + "Severity" indicates I/O pressure
- Process
  - One node sends the digest to the other node about all the nodes gossip timestamp
  - Other node checks if the timestamp is latest (maybe the transmitting node has latest or itself may have the latest)
  - Receiving node sends the ack of the nodes for which it needs data and sends the data which it has as the latest
  - The sending node now accepts latest data from the receiving node and updates itself with any latest data from the receiving node
  - Transmit the other node data which the receiving node needs
- Communication among the nodes is not a guarantee, but that's not an issue because one node may learn about another node from more than 1 other node
- Once gossip has started on a node, the stats about other nodes are stored locally so that a re-start can be fast
- It is versioned, older states are erased
- Local gossip info can be cleaned if needed. Eg when an admin wants to reset the gossip state of a node (perhaps after bringing up a node that was unavailable). Start with -Dcassandra.load_ring_state=false option
- Helps in detecting failed "not responsive since" nodes (next ...)

Detecting a failed node

greatlearning
Learning for Life

- C* uses a mechanism called "Accrual Failure Detector"
- The basic idea is that a node's state is not necessarily up or down. What else? "Busy" maybe!
- It is an educated guess which takes multiple factors into account
- A server (node n1) suspects that a node is down because it hasn't received the two last heartbeats from node n2
- n1 assigns a specific phi value to node n2 which denotes a level of "suspicion" that something could be wrong with n2
- Further lost heartbeats increases the phi value until it reaches a threshold
- When threshold > acceptable value the node is marked down
- The parameter is called "phi_convict_threshold" but it is always not required to be changed
- Higher the value the less chance of getting a false positive about a failure (Good range is between 5 and 12, default is 8)
- See "FailureDetector.java" class for details & JIRA CASSANDRA-2597

Partitioner

- How do you all organize your cubicle/office space? Where will your stuff be? Assume the floor you are on is about 25,000 sq ft.
- A partitioner determines how to distribute the data across the nodes in the cluster
- Murmur3Partitioner - recommended for most purposes (default strategy as well)
- Once a partitioner is set for a cluster, cannot be changed without data reload
- The partition/row key is hashed using the murmer hash to determine which node it needs to go
- There is nothing called as the "Master/Primary/Original/Gold replica", all replicas are identical - first/second/third ... replica
- The token range it can produce is -2^{63} to $2^{63} - 1$ (Range of Long, ROL)
- Wikipedia details
  - MurmurHash is a non-cryptographic hash function suitable for general hash-based lookup. It was created by Austin Appleby in 2008, and exists in a number of variants, all of which have been released into the public domain. When compared to other popular hash functions, MurmurHash performed well in a random distribution of regular keys.

Replication

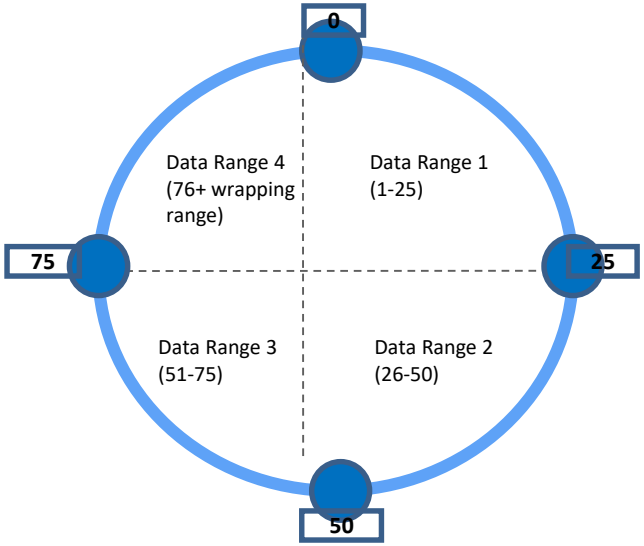
- How many of you have multiple copies of the most critical files - pwd file, IT return confirmation etc on external drives?
- Replication determines how many copies of the data will be maintained in the cluster across nodes
- There is no single magic formula to determine the correct number
- The widely accepted number is 3 but your use case can be different
- This has an impact on the number of nodes in the cluster - cannot have a high replication with less nodes
  - Replication factor <= number of nodes
- Seamless synchronization of data across DC/DR
- In a DC/DR scenario using NetworkTopology strategy different replication can be specified per keyspace per DR/DC
  - strategy_options:{data-center-name}={rep-factor-value}
- Also has a performance impact during
  - "Insert/Update" using a particular consistency level
  - "Select" using a particular consistency level
- System tables are never replicated, they remain local to each node

Partitioner and Replication

- Position of the first copy of the data is determined by the partitioner and copies are placed by walking the cluster in a clockwise direction
- For example, consider a simple 4 node cluster where all of the partition/row keys managed by the cluster were numbers in the PRIMARY range of 0 to 100.
- Each node is assigned a token that represents a point in this range. In this simple example, the token values are 0, 25, 50, and 75.
- The first node, the one with token 0, is responsible for the wrapping range (76-0).
- The node with the lowest token also accepts partition/row keys less than the lowest token and more than the highest token.

- Once the first node is determined the replicas will be placed in the nodes in a clockwise order

- The process is orchestrated by "Coordinator" .. a bit later

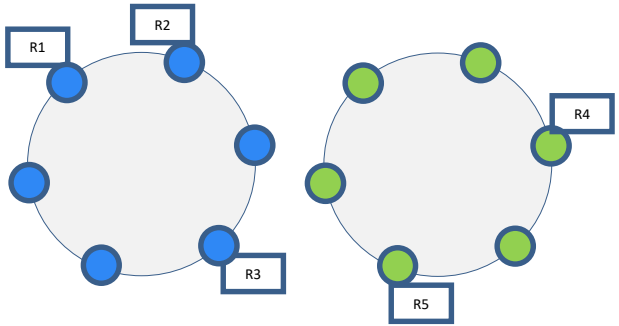


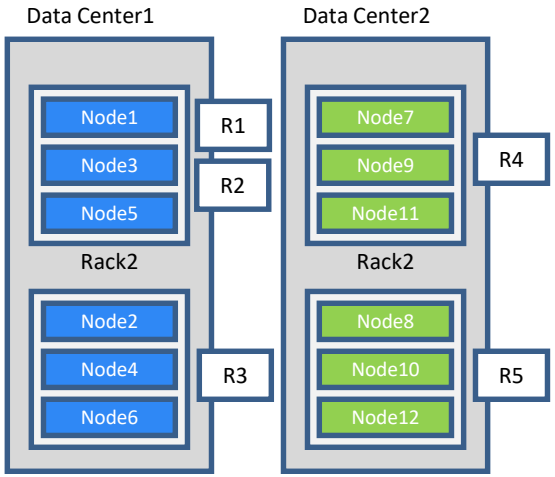
Multi DC replication

- When using NetworkToplogyStrategy, you set the number of replicas per data center
- For example if you set the replicas as 5 (3+2) then this is what you get ...

```
Create KEYSPACE stockdb WITH REPLICATION =
  {'class' : 'NetworkTopologyStrategy', 'DC1' : 3, 'DC2' : 2};

If altering RF, run nodetool repair on each affected node, wait for it
to come up and move on to the next node.
```





Snitch

- What can you typically find at the entrance of a very large mall? What's the need for "Can I help you?" desk?
- Informs the partitioner about the rack and DC locations - determines which nodes the replicas need to go
- Identify which DC and rack a node belongs to
- Think of this as the "Google map" for directions for the replication process
- "Tries" not to have more than one replica in the same rack (may not be a physical grouping)
- Routing requests efficiently
- Allows for a truly distributed fault tolerant cluster
- To be selected at the time of C* installation in C* yaml file
- Changing a snitch is a long drawn up process especially if data exists in the keyspace - you have to run a full repair in your cluster

Concepts - summary

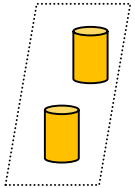
- Seed node - like a recruitment department
- Gossip - share load information
- Partitioner - distribute data around the cluster based on Murmer3hash
- Replication - walk cluster clockwise, use snitch and try to avoid replicas on same rack
- Vnodes - split the token range (ROL) into smaller chunks. Auto calculation of the ranges
- Snitch - a yaml setting to help in positioning the replicas

Deployment - 4 dimensions

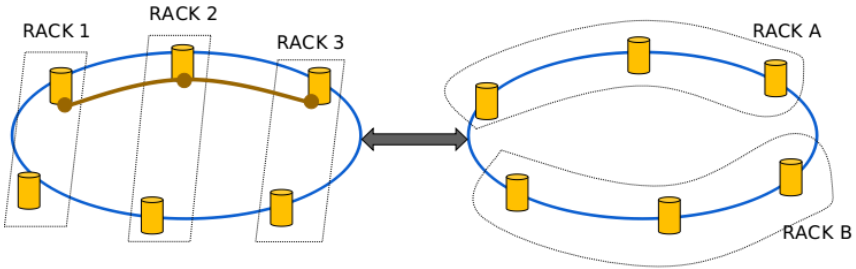
greatlearning
Learning for Life

1 Node - One C* instance

2 Rack - Logical set of nodes



3 DC-DR - Logical set of racks



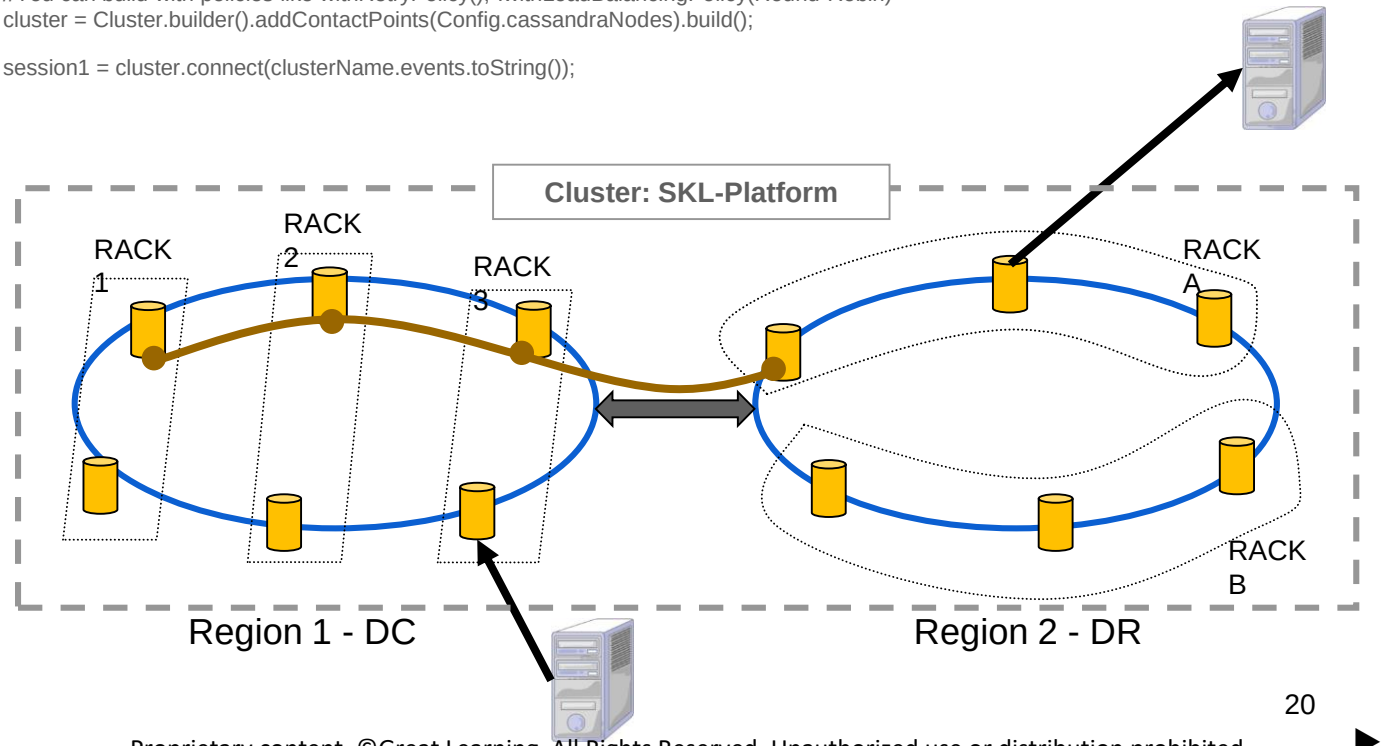
4 Cluster - Nodes that map to a single token ring (can be across DC), next slide ...

```
//These are the C* nodes that the DAO will look to connect to
public static final String[] cassandraNodes = { "10.24.37.1", "10.24.37.2", "10.24.37.3" };

public enum clusterName { events, counter }

//You can build with policies like withRetryPolicy(), .withLoadBalancingPolicy(Round Robin)
cluster = Cluster.builder().addContactPoints(Config.cassandraNodes).build();

session1 = cluster.connect(clusterName.events.toString());
```



Setup & Installation

Cassandra.yaml - an introduction

- cluster_name *(default: 'Test Cluster')*
  - All *nodes* in a *cluster* must have the same value.
- listen_address *(default: localhost)*
  - IP address or hostname other nodes use to connect to this *node*
- commitlog_directory *(default: /var/lib/cassandra/commitlog)*
  - Best practice to mount on a separate disk in production (unless SSD)
- data_file_directories *(default: /var/lib/cassandra/data)*
  - Storage directory for data tables (*SSTables*)
- saved_caches_directory *(default: /var/lib/cassandra/saved_caches)*
  - Storage directory *for key caches* and *row caches*
- rpc_address / rpc_port *(default: localhost / 9160)*
  - listen address / port for *Thrift* client connections
- native_transport_port *(default: 9042)*
  - listen address for *Native CQL Driver* binary protocol

* hints_directory: /var/lib/cassandra/hints
  - This is where the hints will be stored in segments
* disk_optimization_strategy: ssd (default, other is spinning)

There is another undocumented parameter called auto_bootstrap: [true/false]
This can start C* node but will not start the bootstrap process until you explicitly join using nodetool
There are more configuration parameters, but we will cover those as we move along ...

Concepts II

Keyspace aka Schema

greatlearning
Learning for Life

It is like the database (MySQL) or schema/user (Oracle)

Cassandra	Database
create keyspace [IF NOT EXISTS] meterdata with replication strategy (optionally with DC/DR setup), durable writes (True/False)	create database meterdata;

durable writes = false bypasses the commit log. You can lose data!

Table (older nomenclature CF)

greatlearning
Learning for Life

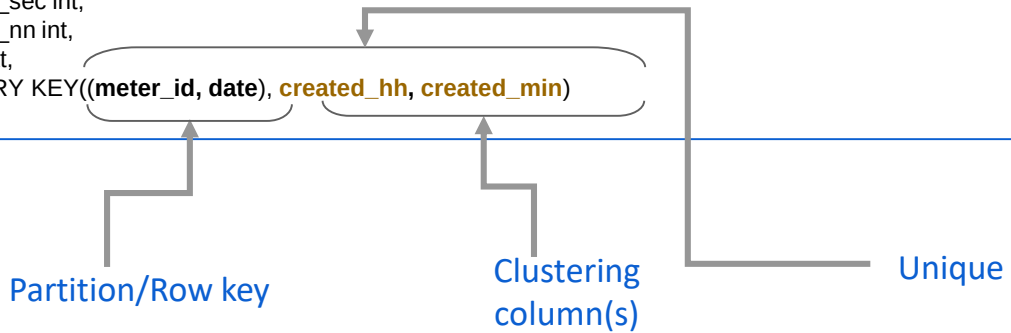
Cassandra	Database
create table meterdata.bill_data (...) primary key (compound key) with compaction, clustering order Primary Key is mandatory in C*	create table meterdata.bill_date (...) pk, references, engine, charset etc
Insert into bill_data () values (); Update bill_data set=.. where .. Delete from bill_data where ..	Standard SQL DML statements

What happens if we insert with a primary key that already exists?
Hold on to your thoughts ...

Visualizing the Primary Key

What are the components of Primary key?

```
CREATE TABLE meter_data (
  meter_id text,
  date int,              //format as yyyymmdd number (more efficient)
  created_hh int,
  created_min int,
  created_sec int,
  created_nn int,
  data text,
  PRIMARY KEY((meter_id, date), created_hh, created_min)
);
```



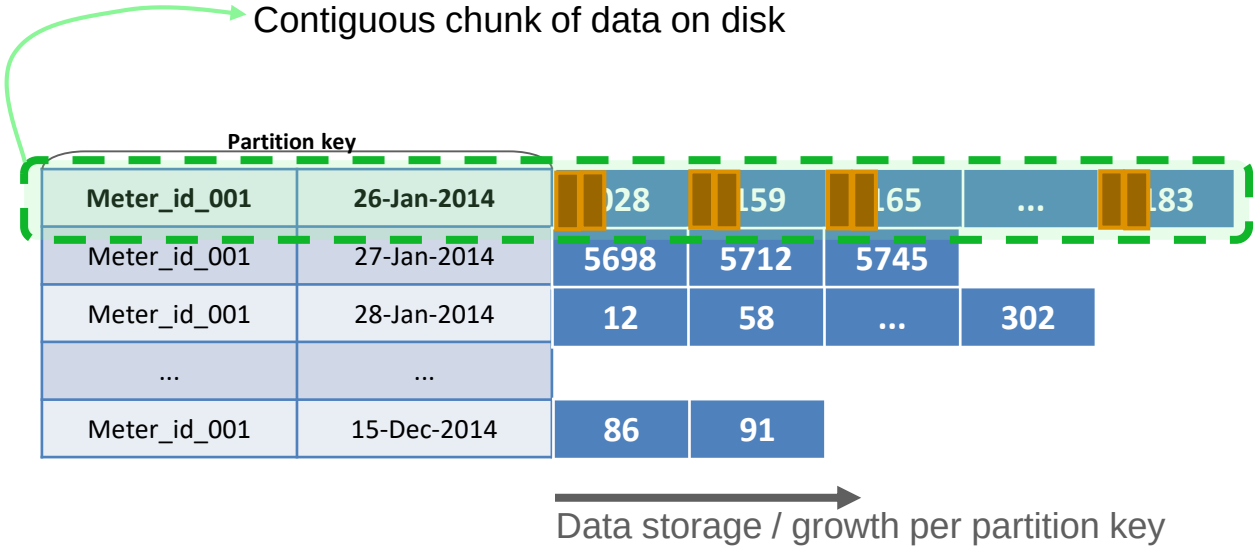
Primary key = Partition/Row key columns(s) + Clustering column(s)

Naturally by definition the primary key must be unique

Visualizing PK based storage

greatlearning
Learning for Life

- Question - how does C* keep the data based on the partition/row key definition of a given table?



Note: C* creates a **key/name** using the clustering column values with regular column names and then stores the **value** of the regular column as a "**cell**". Visuals on next slide...

The brown boxes indicate the clustering column values being part of the key of the cell that contains the meter data.

ANOTHER note: the internal representation has undergone a change in version 3.0 (Rows concept)

Partition size - Guideline

"*For efficient operation, partitions must be sized within certain limits in Apache Cassandra™. Two measures of partition size are the number of values in a partition and the partition size on disk. The practical limit of cells per partition is two billion.*

Sizing the disk space is more complex, and involves the number of rows and the number of columns, primary key columns and static columns in each table.

Each application will have different efficiency parameters, but a good rule of thumb is to keep the maximum ***number of rows below 100,000 items and the disk size under 100 MB***"

http://docs.datastax.com/en/landing_page/doc/landing_page/planning/planningPartitionSize.html

Failure/Partition Tolerance By Replicatio

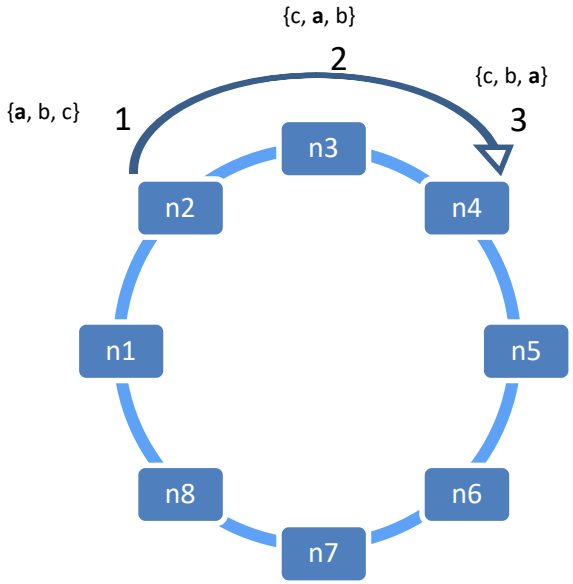
greatlearning
Learning for Life

Remember CAP theorem? C* gives availability and partition tolerance

Quick Q: Availability is achieved by?

Assume Replication Factor (RF) = 3

Let's insert a record with Pk=a




```
private static final String allEventCql = "insert into all_events (event_type, date, created_hh, created_min,
                                          created_sec, created_nn, data) values(?, ?, ?, ?, ?, ?, ?)";
Session session = CassandraDAO.getEventSession();
BoundStatement boundStatement = getWritableStatement(session, allEventCql);
session.execute(boundStatement.bind(.., .., ..);
```

Coordinator Node - Single DC replicatior

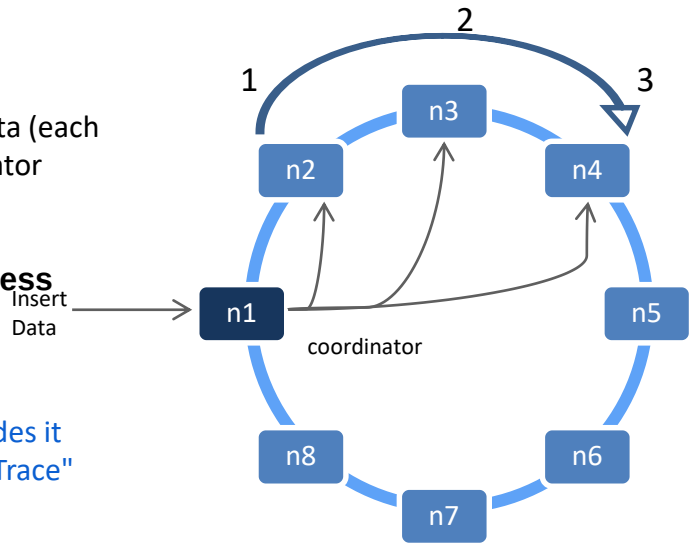
greatlearning
Learning for Life

Incoming Requests (read/Write)

Coordinator handles the request

An interesting aspect to note is that the data (each column) is timestamped using the coordinator node's time

Every node can be **coordinator -> masterless**

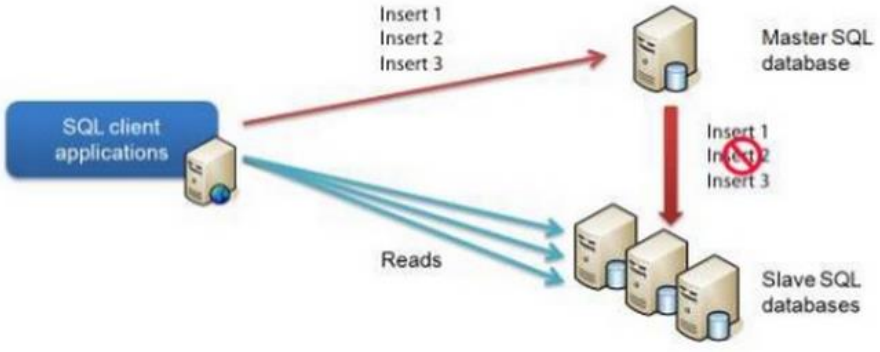


You can see the coordinator and all the nodes it connects to in the new DevCenter "Query Trace" tab!

Hinted handoff a bit later ...

Eventual consistency

- 1 + 1 = 2 OR eventually 2?
- Do you really have strong consistency in RDBMS?



Consistency Level

greatlearning
Learning for Life

What is CL? Can you define it?

"At what point in time the coordinator can respond to the client?"

Tunable at runtime

- ONE (default)
- QUORUM (strict majority w.r.t RF)
- ALL

Can be applied both to read and write

```
protected static BoundStatement getWritableStatement(Session session, String cql,
                                                     boolean setAnyConsistencyLevel) {
        PreparedStatement statement = session.prepare(cql);  //Ideally prepare a statement once per session
        if(setAnyConsistencyLevel) {
              statement.setConsistencyLevel(ConsistencyLevel.QUORUM);
        }

        BoundStatement boundStatement = new BoundStatement(statement);
        return boundStatement;
}


cqlsh:meterdb> consistency;
cqlsh:meterdb> consistency all;
```

Other consistency levels are ANY, LOCAL_QUORUM, TWO, THREE, LOCAL_ONE (see documentation on the web)

What is a Quorum?

quorum = RoundDown(sum_of_replication_factors / 2) + 1

sum_of_replication_factors = datacenter1_RF + datacenter2_RF + . . . + datacentern_RF

- Using a replication factor of 3, a quorum is 2 nodes. The cluster can tolerate 1 replica down.
- Using a replication factor of 6, a quorum is 4. The cluster can tolerate 2 replicas down.
- In a two data center cluster where each data center has a replication factor of 3, a quorum is 4 nodes. The cluster can tolerate 2 replica nodes down.
- In a five data center cluster where two data centers have a replication factor of 3 and three data centers have a replication factor of 2, a quorum is 7 nodes.

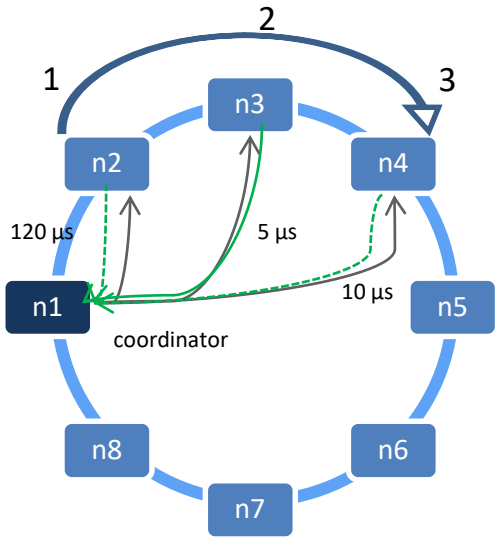
Write Consistency level

Write ONE

Send requests to **all replicas** in the cluster applicable to the PK

Wait for ONE ack before returning to client

Other acks later , asynchronously

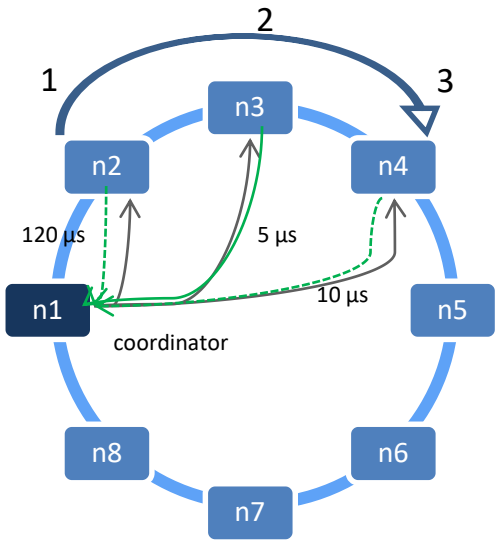


Write Consistency level

Write QUORUM

Send requests to **all replicas**

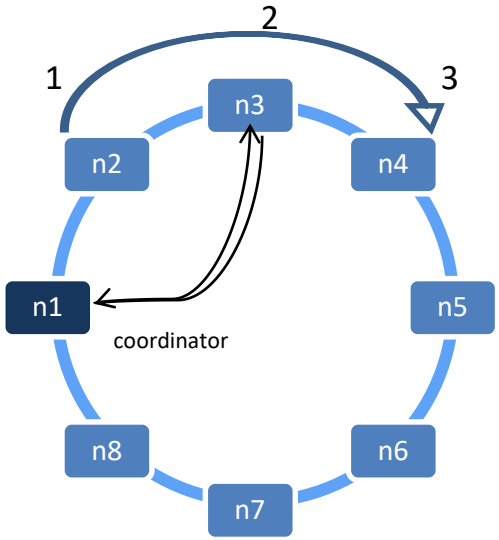
Wait for QUORUM ack before returning to client



Read Consistency level

Read ONE

Read from one node among all replicas



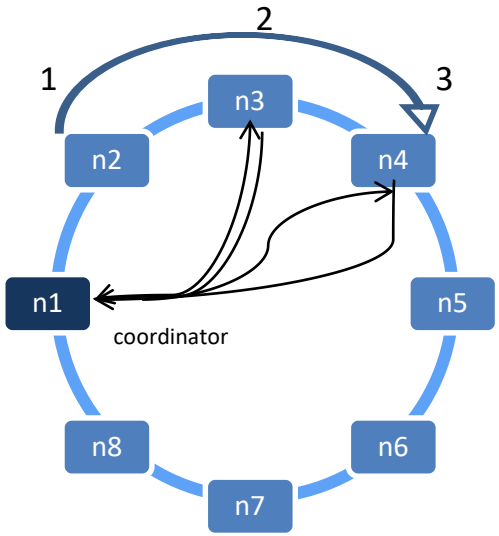
Read Consistency level

Read QUORUM

Read from one fastest/preferred node

AND **request digest** from other replicas to reach QUORUM

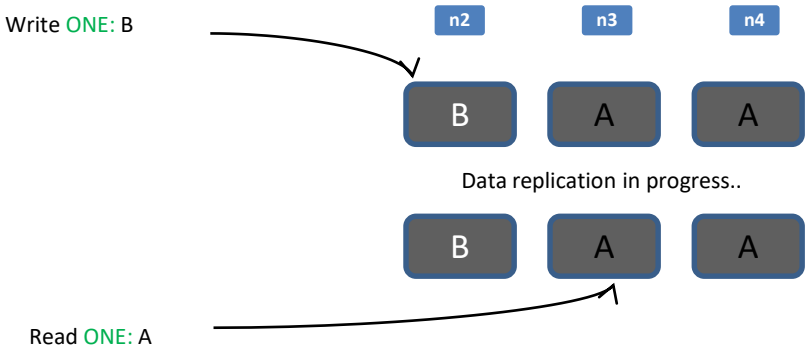
Return most up-to-date data to client



Consistency level in Action

RF = 3, Write ONE Read ONE

A write must be written to the commit log and memtable of at least one replica node.

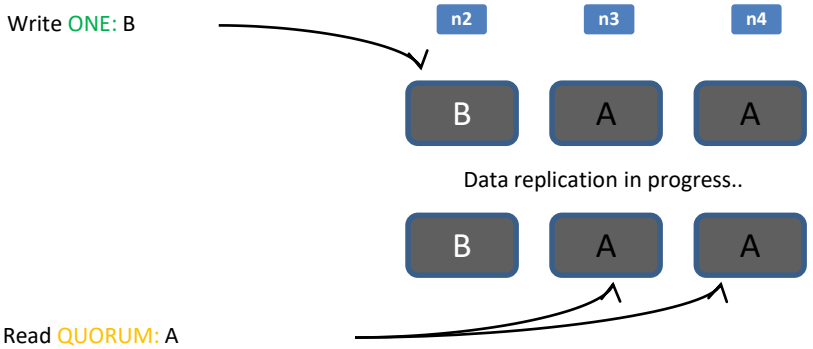


Returns a response from the closest replica, as determined by the snitch. By default, a read repair runs in the background to make the other replicas consistent.

Consistency level in Action

RF = 3, Write ONE Read QUORUM

A write must be written to the commit log and memtable of at least one replica node.



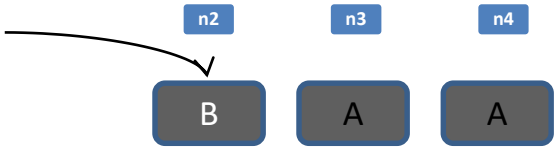
Returns the record after a quorum of replicas has responded from any data center.

Consistency level in Action

RF = 3, Write ONE Read ALL

A write must be written to the commit log and memtable of at least one replica node.

Write ONE: B



Read ALL: B

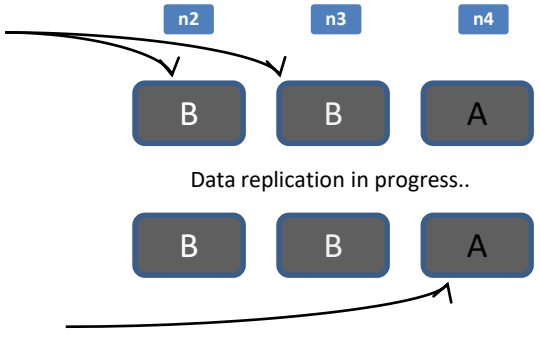
Returns the record after all replicas have responded. The read operation will fail if a replica does not respond.

Consistency level in Action

RF = 3, Write QUORUM Read ONE

A write must be written to the commit log and memtable on a quorum of replica nodes.

Write QUORUM: B



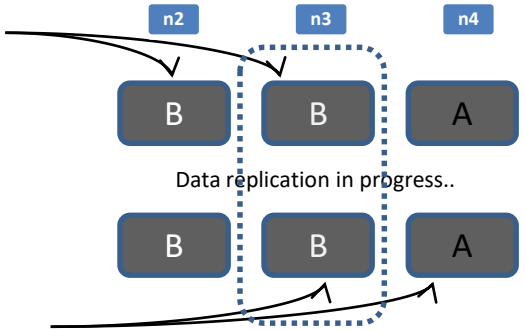
Read ONE: A

Returns a response from the closest replica, as determined by the snitch. By default, a read repair runs in the background to make the other replicas consistent.

Consistency level in Action

RF = 3, Write QUORUM Read QUORUM

Write QUORUM: B



Read QUORUM: B

Consistency Level - summary

greatlearning
Learning for Life

ONE

Fast write, may not read latest written value

QUORUM / LOCAL_QUORUM

Strict majority w.r.t. Replication Factor
Good balance

ALL

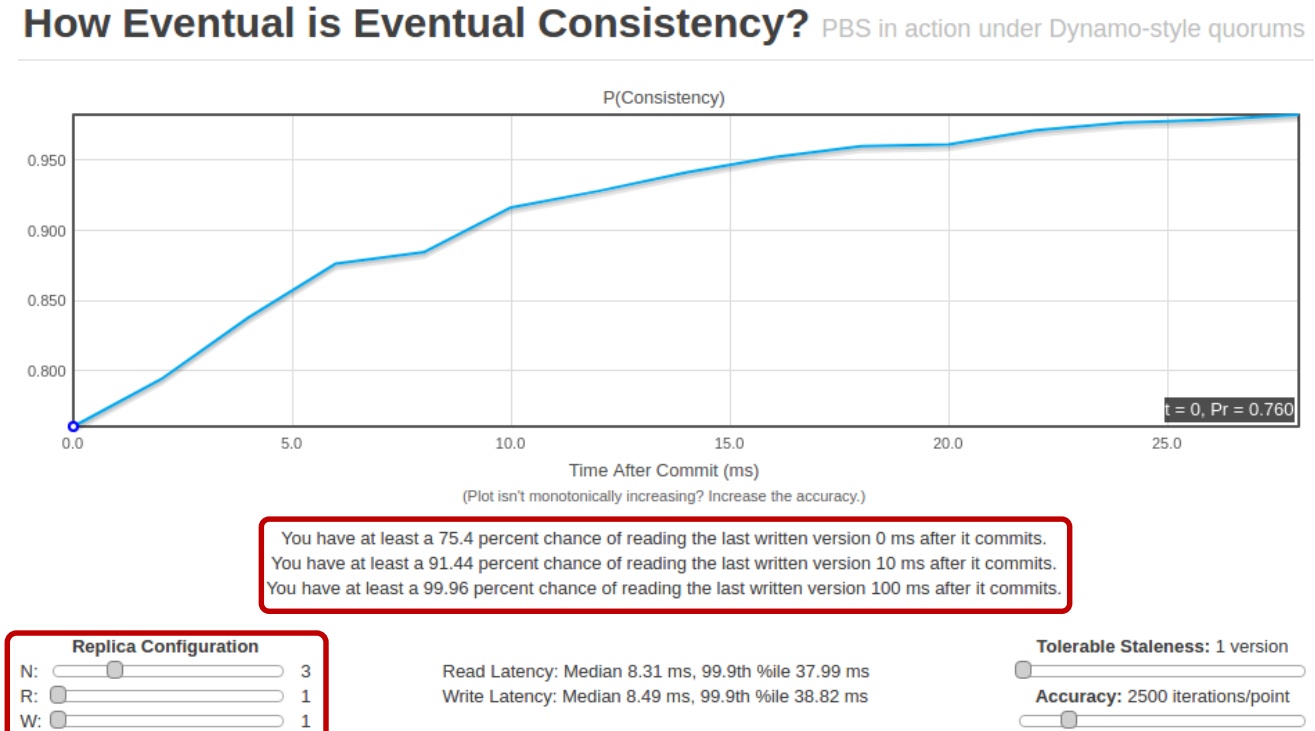
Not the best choice
Slow, no high availability

if(nodes_written + nodes_read) > replication factor then you can get **immediate consistency**

Debate - Immediate consistency

- The following consistency level gives immediate consistency
  - Write CL = ALL, Read CL = ONE
  - Write CL = ONE, Read CL = ALL
  - Write CL = QUORUM, Read CL = QUORUM
- Debate which CL combination you will consider in what scenarios?
- The combinations are
  - Few write but read heavy
  - Heavy write but few reads
  - Balanced

How eventual?



Awesome article...

https://www.datastax.com/dev/blog/your-ideal-performance-consistency-tradeoff

Write and Read path

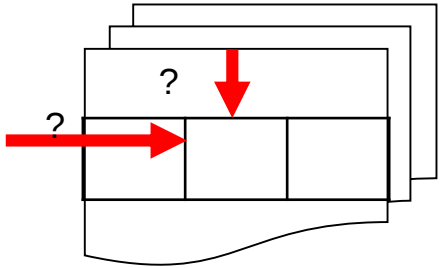
Try reading a book that is organized as follows

P1, P23, P45, P2, P6, P99, P125, P8 ...

How about a book that looks like

P1, P2, P3, P4, P5, P6, ...

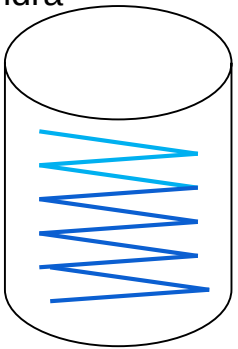
RDBMS



Seeks and writes values to various pre-set locations

Is this random?

Cassandra



Continuously appends, merging and compacting when appropriate

What about this disk access?

Storage

- Writes are harder than reads to scale
- Spinning disks aren't good with random I/O
- Goal: minimize random I/O
  - Log Structured Merged Tree (LSM)
  - Push changes from a memory-based component to one or more disk components

Some more on LSM

- An LSM-tree is composed of two or more tree-like component data structures
- Smaller component is C0 sits in memory - aka Memtables
- Larger component C1 sits in the disk
  - aka SSTables (Sorted String Table)
  - Immutable component
- Writes are inserted in C0 and logs
- In time flush C0 to C1
- C1 is optimized for sequential access

Key components for Write

greatlearning
Learning for Life

Each node implements the following key components (3 storage mechanism and one process) to handle writes

1. Memtables - the in-memory tables corresponding to the CQL tables along with the related indexes

2. CommitLog - an append-only log, replayed to restore node Memtables

3. SSTables - Memtable snapshots periodically flushed (written out) to free up heap

4. Compaction - periodic process to merge and streamline multiple generations of SSTables

Last Write Win (LWW)

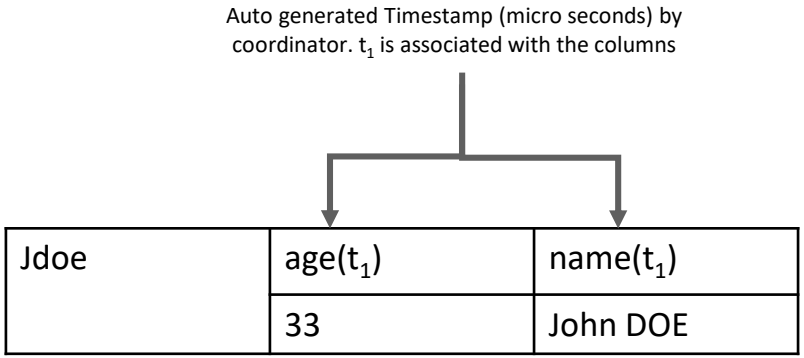
```
INSERT INTO users(login,name,age) VALUES ('jdoe', 'John DOE', 33);
```

#Partition

Jdoe	age	name
	33	John DOE

Last Write Win (LWW)

```
INSERT INTO users(login,name,age) VALUES ('jdoe', 'John DOE', 33);
```



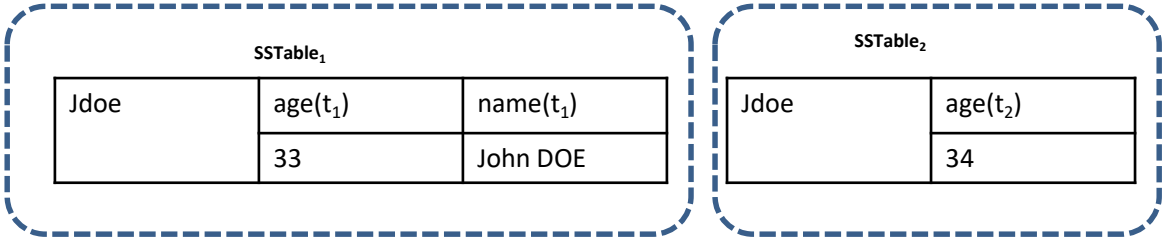
Q: How do I know what was the timestamp associated with the column?

A: select writetime(age) from users where user_id = 'Jdoe';

Last Write Win (LWW)

```
UPDATE users SET age = 34 WHERE login = 'jdoe';
```

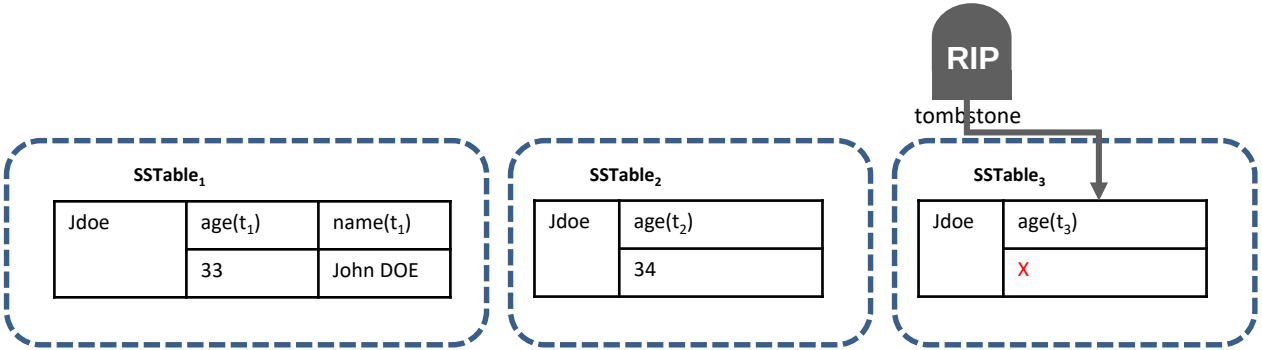
Assume a flush occurs



Remember that SSTables are immutable, once written it cannot be updated.
Creates a new SSTable

Last Write Win (LWW)

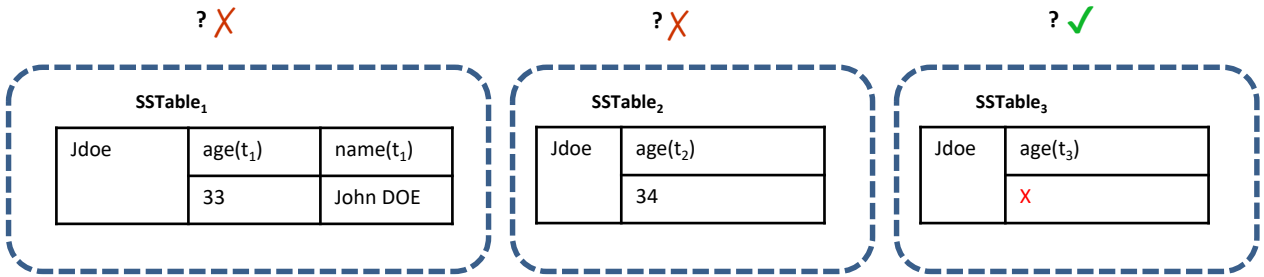
```
DELETE age from users WHERE login = 'jdoe';
```



Last Write Win (LWW)

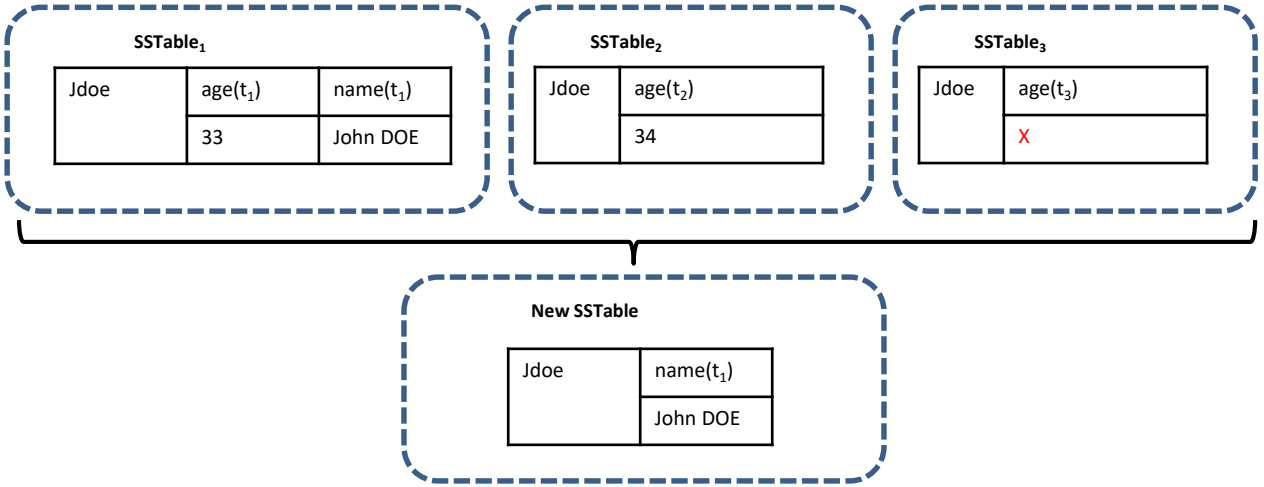

```
SELECT name, age from users WHERE login = 'jdoe';
```

Where to read from? How to construct the response?



Debate: Writes in C* are idempotent. Why?

Compaction



- Related SSTables are merged
- Most recent version of each column is compiled to one partition in one new SSTable
- Columns marked for eviction/deletion & tombstones older than gc_grace_seconds are removed
- Old generation SSTables are deleted
- A new generation SSTable is created (hence the generation number keep increasing over time)

Disk space freed = sizeof(SST1 + SST2 + .. + SSTn) - sizeof(new compact SST)
Commit logs (containing segments) are versioned and reused as well
Newly freed space becomes available for reuse

WHERE clause & restrictions

greatlearning
Learning for Life

- Deeper look at WHERE http://www.datastax.com/dev/blog/a-deep-look-to-the-cql-where-clause
- All queries (INSERT/UPDATE/DELETE/SELECT) must provide #partition (where can filter in contiguous data)
- WHERE clause is only possible on columns defined in primary key unless there is a secondary index available for that column
- "Allow filtering" can override the default behaviour of cross partition access but it is not a good practice at all (incorrect data model)
  - select .. from meter_reading where record_date > '2014-12-15'

  (assuming there is a secondary index on record_date)
- Exact match (=) or IN predicate on any partition key is allowed, range queries (<,<=,<=,>) not allowed (C* 2.2+)
  - Prior to C* 2.2, in case of a compound partition key IN is allowed in the last column of the compound key (if partition key has one only column then it works on that column)
- On clustering columns, exact match and range query predicates (<,<=,<=,>, IN) are allowed

  Consider clustering columns col1, col2, col3

  col1=1 and col2>3 and col3=4 //col3 cannot be restricted because the previous uses non eq restriction

  col1=1 and col2>3 and col3>4 //col3 cannot be restricted because the previous uses non eq restriction

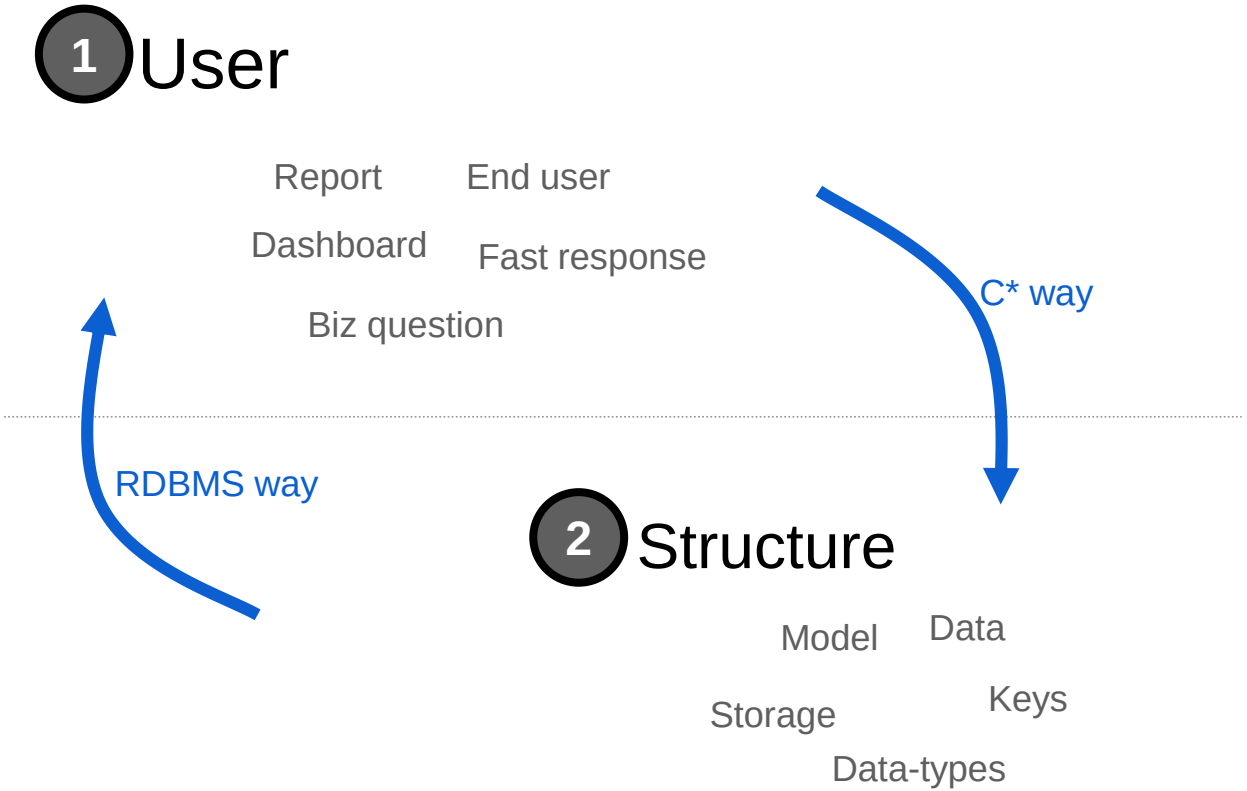
  col1=1 and col2 in (3,5) and col3=4 //OK

  col1=1 and col2 in (3,5) and col3>4 //OK
- Order of the clustering filters must match the order of primary key definition otherwise create secondary index (anti-pattern)

greatlearning
Learning for Life

Modeling II and development

Before you model ...



C* modelling is "Query Driven Design"

If you want to be a "Super Modeler" ask this ...

"Give me the query and I will organize the data via a model to get you blazing performance for that query"

Thinking "outwards" vs "inwards"!

Ex 1 - stock model

- Remember we have to work backwards from how we would like to retrieve the data

- Here are the queries
  - Users have stocks that they track
  - User's portfolio value should be current
  - Latest value for a given stock

Stock CQL model

create keyspace stockdbwith replication = { 'class':'SimpleStrategy', 'replication_factor': 1 };

```
CREATE TABLE stockdb.user(
  user_id int,
  display_name VARCHAR,
  first_name VARCHAR,
  last_name VARCHAR
  PRIMARY KEY (user_id));
```

```
CREATE TABLE exchange(
  exchange_id VARCHAR,
  exchange_name VARCHAR,
  currency_code VARCHAR,
  active BOOLEAN,
  PRIMARY KEY (exchange_id));
```

```
CREATE TABLE stockdb.userstocklist(
  stock_symbol VARCHAR,
  user_id int,
  display_name VARCHAR,
  holding int,
  PRIMARY KEY (user_id, stock_symbol);
```

```
CREATE TABLE industry(
  industry_id INT,
  industry_name VARCHAR,
  sector_id INT,
  PRIMARY KEY (industry_id));
```

```
CREATE TABLE stockdb.stockprice (
  stock_symbol VARCHAR,
  trade_date INT,
  trade_time INT,
  company_name VARCHAR,
  start_price DECIMAL,
  current_price DECIMAL,
  exchange_id VARCHAR,
  industry_id INT,
  PRIMARY KEY ((trade_date, stock_symbol)), trade_time)
  with clustering order by (trade_time desc);
```

```
CREATE TABLE sector(
  sector_id INT,
  sector_name VARCHAR,
  PRIMARY KEY (sector_id));
```

Q: How to get a list of stocks given an industry?

Stock model "clustering order by" caveat

- Assume the primary key is

  ((industry_id), exchange_id, stock_symbol)

  then the following sort orders are valid
  - order by exchange_id desc, stock_symbol desc
  - order by exchange_id asc, stock_symbol asc

- Following is an example of invalid sort order
  - order by exchange_id desc, stock_symbol asc

Ex 2 - Relationship/search table

Relationship tables are not uncommon to support different search/fetch criteria - get the list of stocks given an industry in this case from previous example it is not possible with the tables!

Solution is to create a new table StockSearch (like a search index) and reference to industry, exchange and stock tables - tradeoff!

```
CREATE TABLE StockSearch (
  industry_id INT,
  exchange_id VARCHAR,
  stock_symbol VARCHAR,
  PRIMARY KEY (industry_id, exchange_id, stock_symbol)
);
```

Model below shows how to access a video by tag or by release date

The video information can be embedded in the "inverted index" tables as well BUT that will take up lots of space. There needs to be a balance of denormalization vs space consumed!

Ex 3 - "Comment" Model

Consider a scenario where the user can give comments about a video like youtube. How to create a model if the queries are-

1. Get all comments for a given video (optionally filter by user)
2. Get all comments given by a user (optionally filter by video)

```
create table comments_by_video (
  video_id uuid,
  user_id varchar,
  comment_id timeuuid,
  comment text,
  primary key ((video_id), user_id, comment_id)
);
```

```
create table comments_by_user(
  user_id uuid,
  video_id varchar,
  comment_id timeuuid,
  comment text,
  primary key ((user_id), video_id, comment_id)
);
```

- Model both ways - create the same comment twice
- A good example of many to many relationship
- Totally ok to write multiple times!
- Do keep in mind the degree of de-normalization (notice the video and user details are not duplicated)
- Can read both ways - very efficient

greatlearning
Learning for Life

Relational Data Services

DynamoDB

DynamoDB - concepts

- Tables - DynamoDB stores data in tables
- Items - Each table contains multiple items. An item is a group of attributes that is uniquely identifiable among all of the other items (yes like rows)
- Attribute - is a fundamental data element, something that does not need to be broken down any further (like columns)
  - Usually scalar (single values)
  - Can also support nested attributes upto 32 levels deep (nested JSON is a good way to visualize)
- Primary Key - uniquely identifies each item in the table (must be scalar)
  - Partition key (PK) - hashed to find out which partition the data will be stored
  - Sort key (SK) - items with the same partition key are stored together, in sorted order by sort key value
- Secondary index - to fetch based on non PK columns
  - Global - where the PK and SK are different as that of the base table
  - Local - where the PK is same but SK is different from the base table
- Can define up to 5 global secondary indexes and 5 local secondary indexes per table
- Query Driven OR Access Pattern Based Design principles
- Allows low latency read and write access to items ranging from 1 byte up to 400KB (Key & Value put together)
- Stored on SSD, spread across 3 different distinct DC
- Supports Eventual consistent reads / Strongly consistent reads (app needs to decide)
- Can be expensive for writes but not so for reads (must see "capacity units" at the time of creating a table)
- Also, supports auto scale to handle variable demand
- Can have a downloadable version that helps in development without the need for cloud services, hence reducing development costs

DynamoDB - concepts

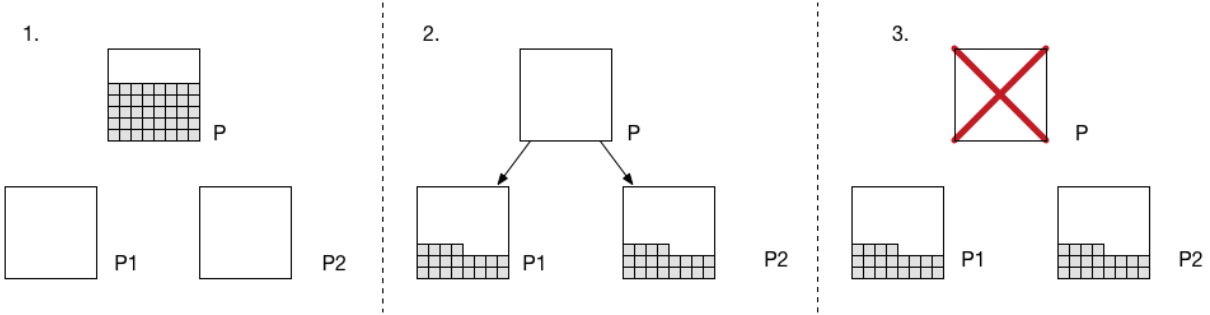
- DynamoDB Streams is an optional feature that captures data modification events in DynamoDB tables (think of CDC)
- If enabled; provides the following -
  - If a new item is added to the table, the stream captures an image of the entire item, including all of its attributes
  - If an item is updated, the stream captures the "before" and "after" image of any attributes that were modified in the item
  - If an item is deleted from the table, the stream captures an image of the entire item before it was deleted
- Stream records have a lifetime of 24 hours; after that, they are automatically removed from the stream
- You can use DynamoDB Streams together with AWS Lambda to create a trigger—code that executes automatically whenever an event of interest appears in a stream
  - If a new customer is created then the Lambda can use SES to send a welcome email

DynamoDB - concepts

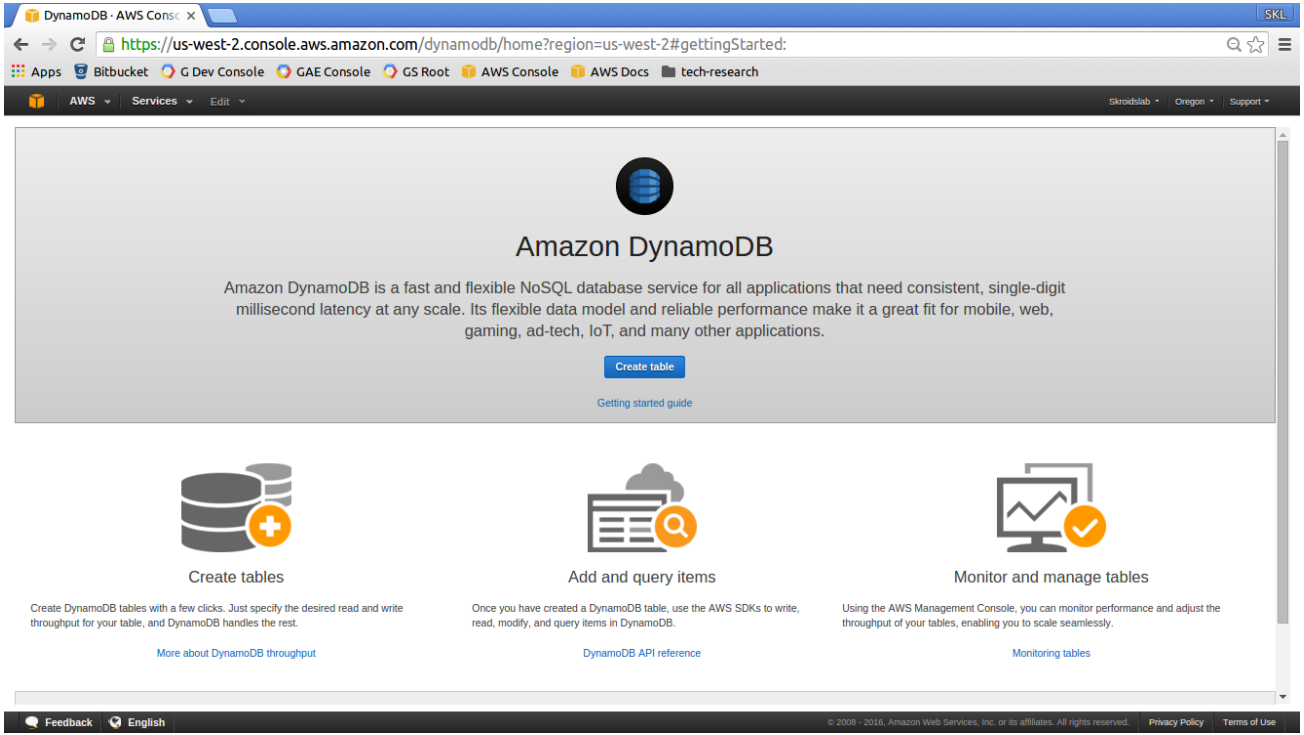
- Every secondary index is associated with exactly one table, from which it obtains its data called the "base table"
- When you create an index, you define an alternate key for the index (partition key and sort key)
- You also define the attributes that you want to be projected, or copied, from the base table into the index
- DynamoDB copies these attributes into the index, along with the primary key attributes from the base table
- You can then query or scan the index just as you would query or scan a table
- Types
  - A global secondary index is considered "global" because queries on the index can span all of the data in the base table, across all partitions (can be created on an existing table)
  - A local secondary index is "local" in the sense that every partition of a local secondary index is scoped to a base table partition (can be created at the time of creating the base table, not possible afterwards)

DynamoDB - concepts

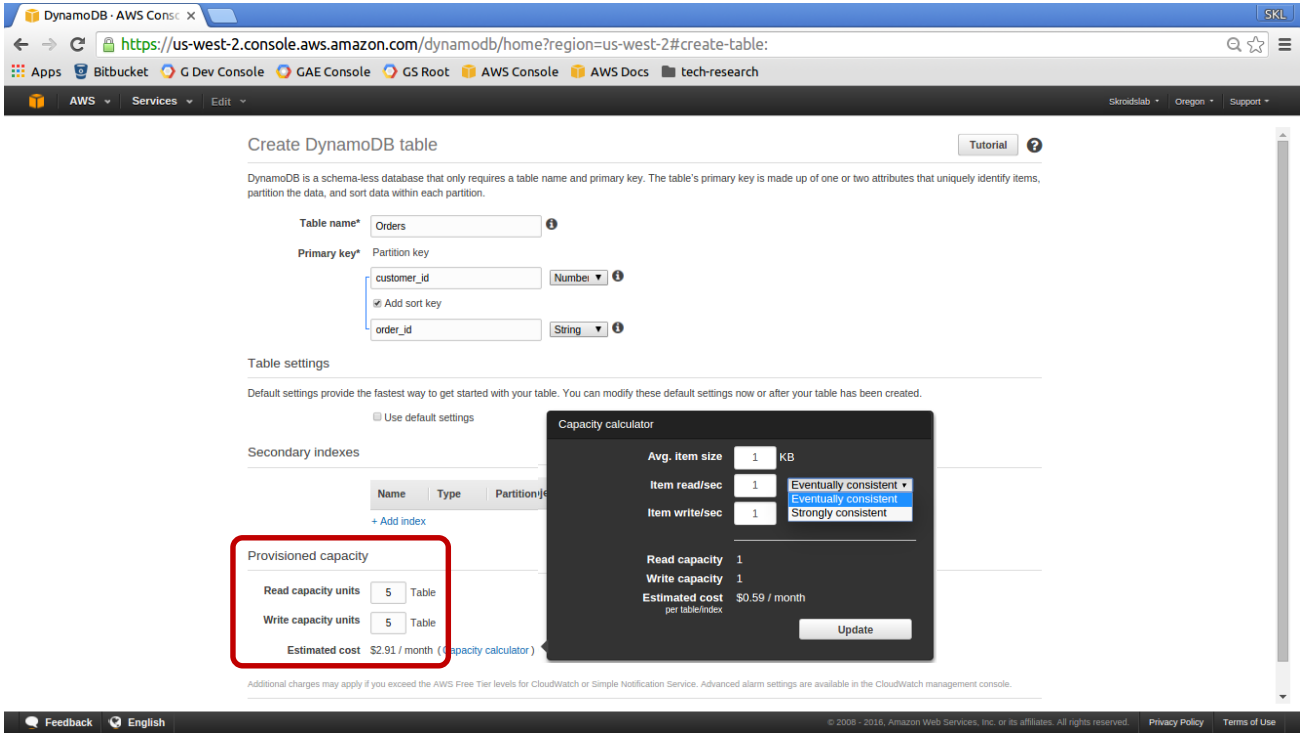
- When you create a table or index in DynamoDB, you must specify your capacity requirements for read and write activity
  - One read capacity unit represents one strongly consistent read per second, or two eventually consistent reads per second, for an item up to 4 KB in size
  - One write capacity unit represents one write per second for an item up to 1 KB in size
- A single partition can hold approximately 10 GB of data, and can support a maximum of 3,000 read capacity units or 1,000 write capacity units
  - (readCapacityUnits / 3,000) + (writeCapacityUnits / 1,000) = initialPartitions (rounded up)
- A partition can split due to the need for more capacity or need to store more data per partition



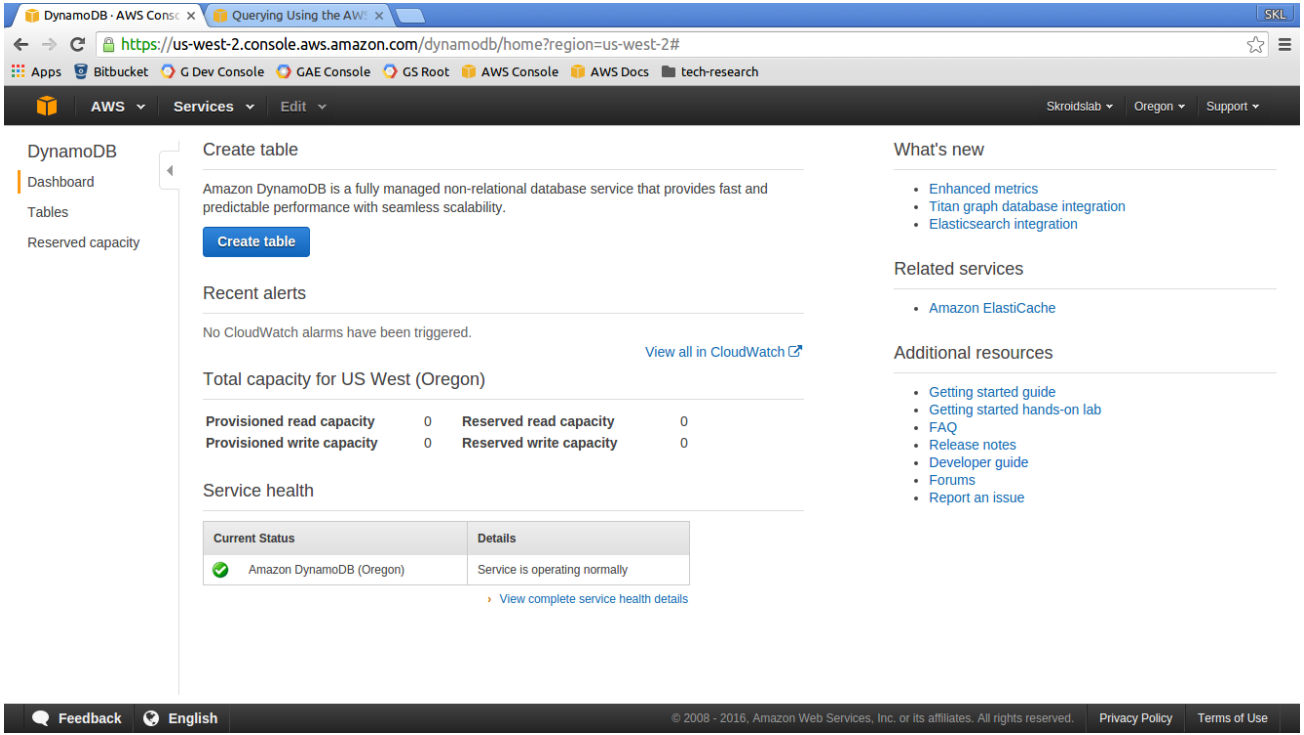
Activity - DynamoDB



Activity - DynamoDB



Activity - DynamoDB



Activity - DynamoDB