NoSQL: Redis, Cass, ES, Kafka

Diego Pacheco

About Me

- Cat's Father
- Principal Software Architect
- Agile Coach
- SOA/Microservices Expert
- DevOps Practitioner
- Speaker
- Author

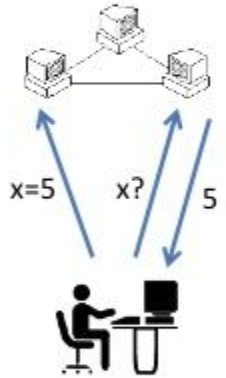
diegopacheco

@diego_pacheco

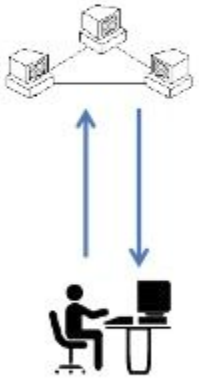
http://diego-pacheco.blogspot.com.br/

CAP Theorem

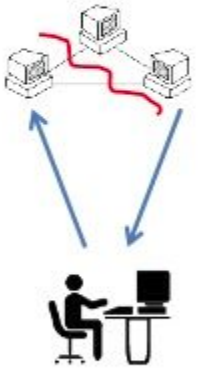
Consistency



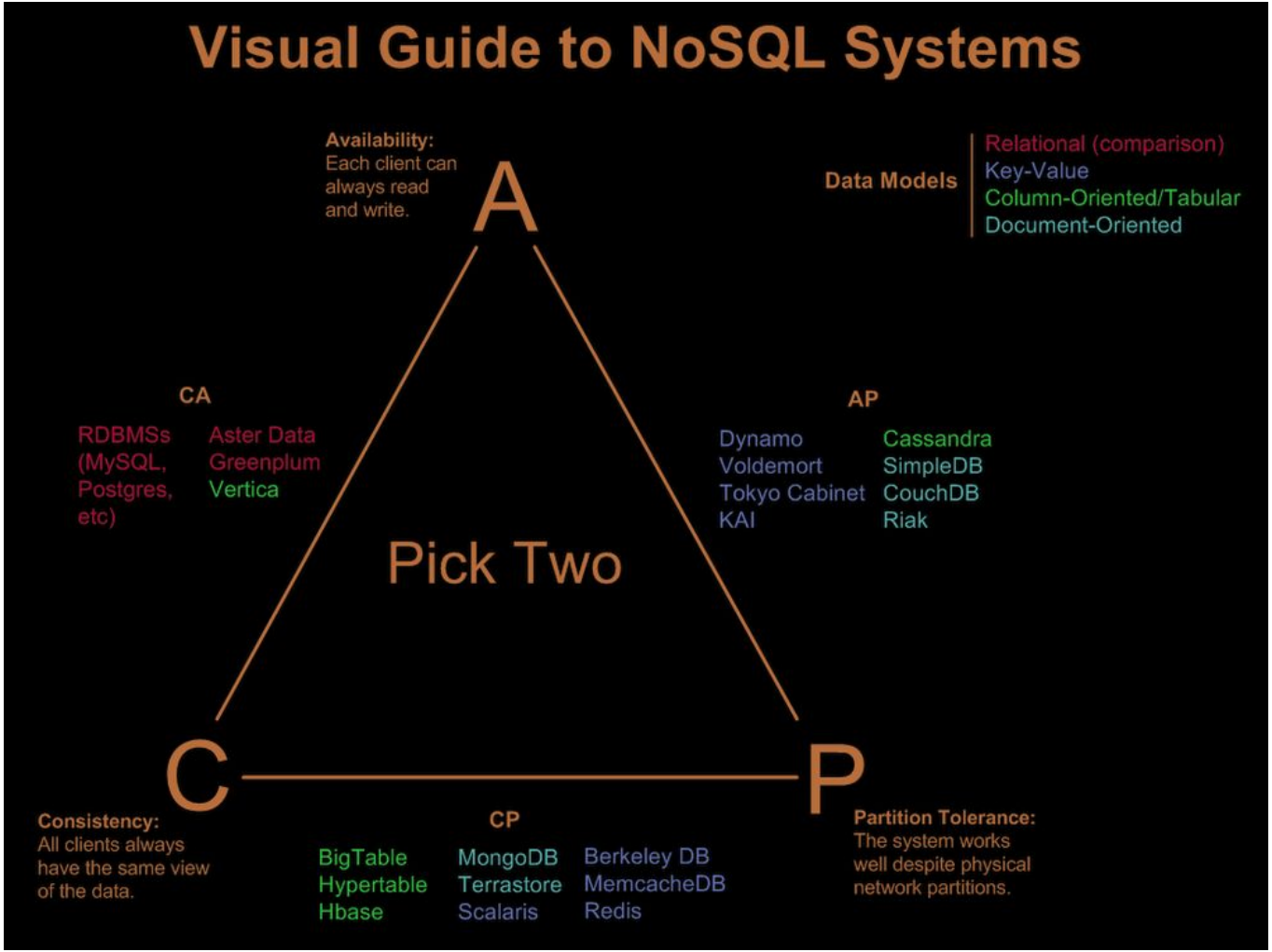
Availability



Partition tolerance



Visual Guide to NoSQL Systems



key-value

Amazon DynamoDB (Beta) · ORACLE BERKELEY DB 11g · redis

graph

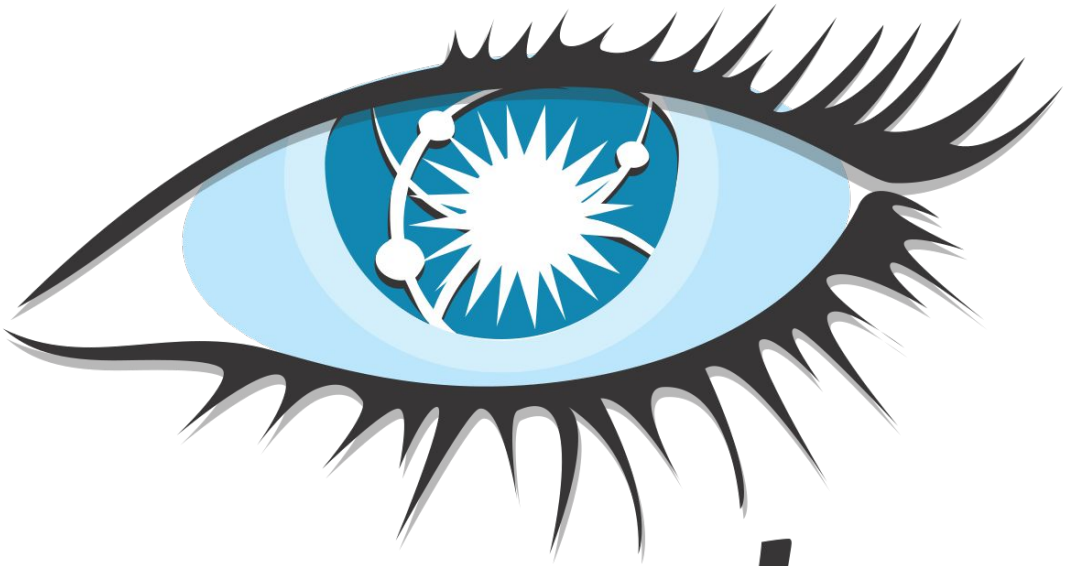
Neo4j the graph database · InfiniteGraph · sones

column

HBASE · riak · Cassandra

document

CouchDB relax · mongoDB · terrastore

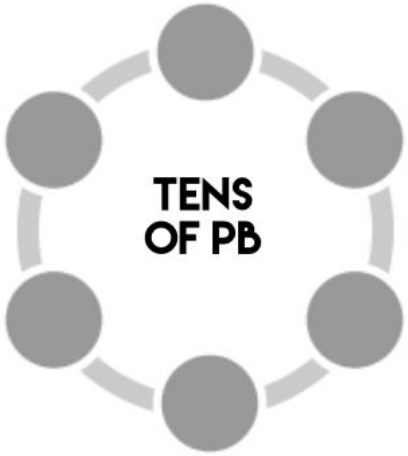


cassandra

Cassandra

- Column oriented NoSQL
- Created by Facebook for Inbox Search
- Inspired from Amazon Dynamo Paper
- Written in Java
- Symetrics Nodes - NO SPOF
- Scalable
- Awesome Write throughput
- High Availability
- Eventual Consistency
- Consistency is tunable

Cassandra Battle Tested

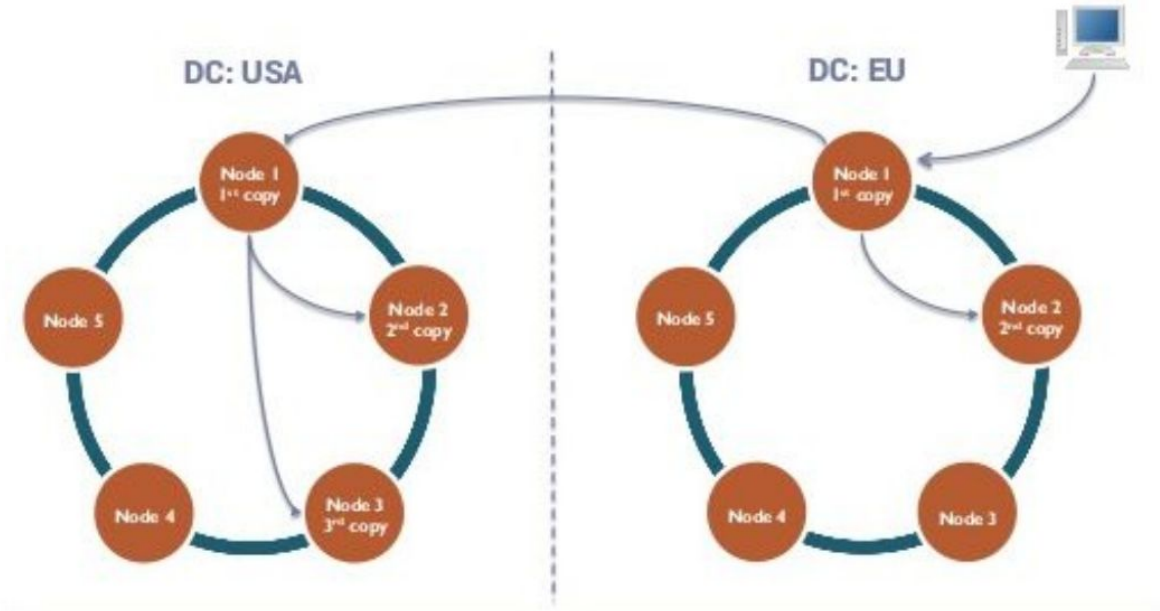


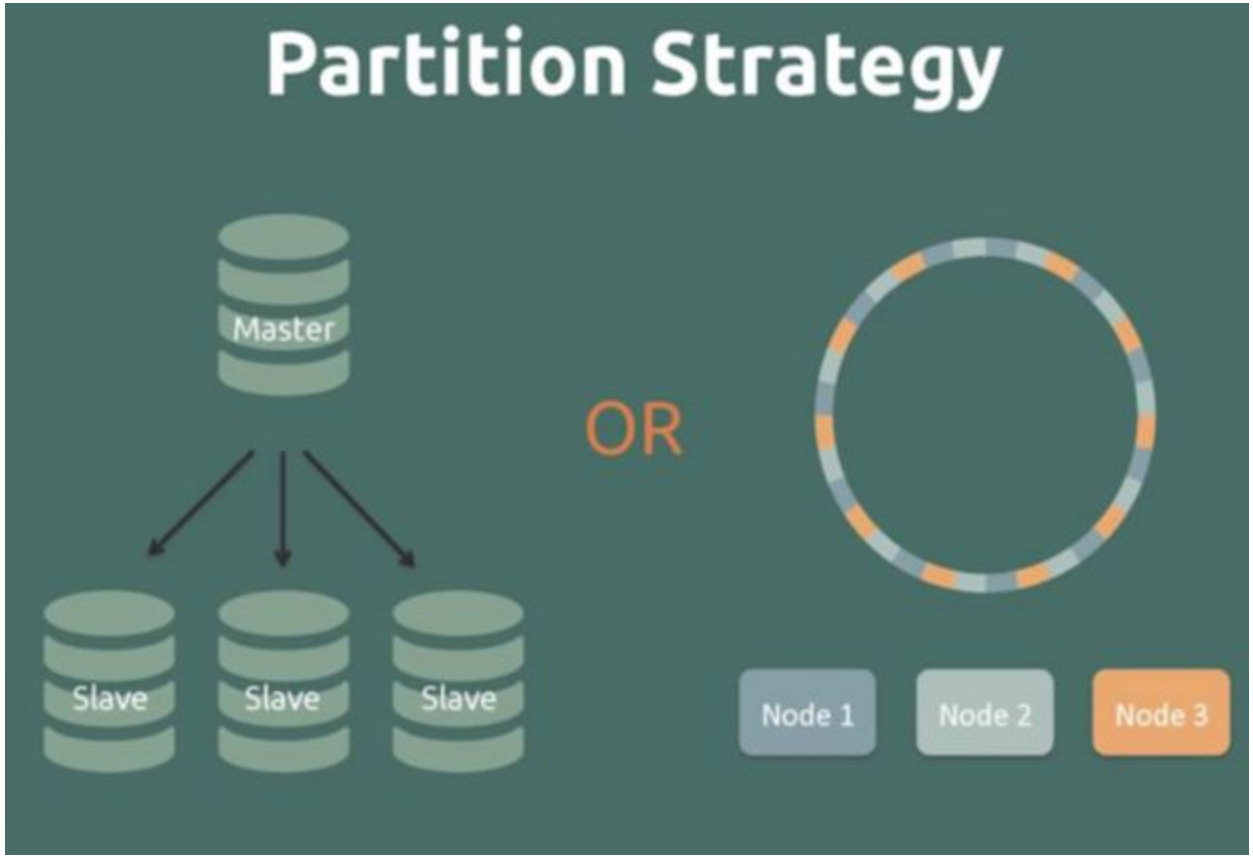
NETFLIX

Spotify

Multi-DC | Cloud Native

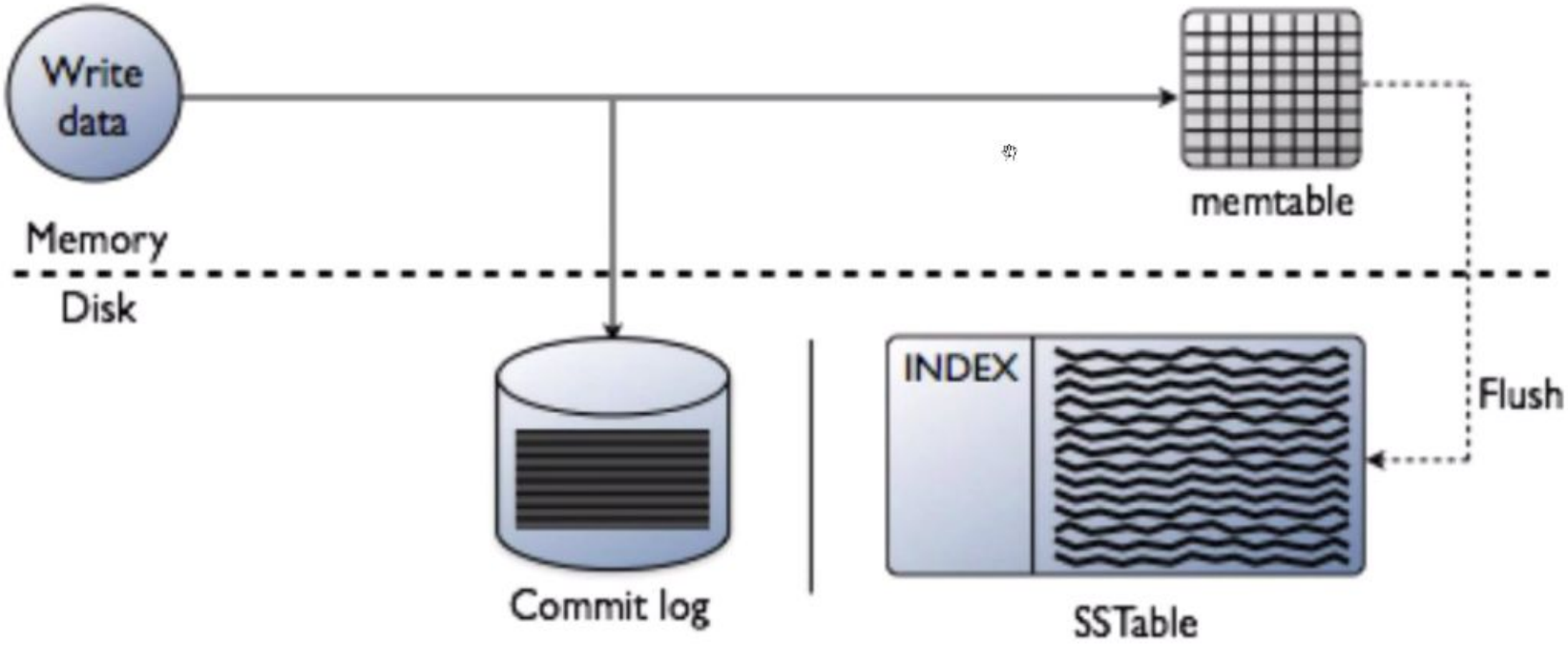
```
CREATE KEYSPACE johnny WITH REPLICATION =
{'class':'NetworkTopologyStrategy', 'USA':3, 'EU': 2};
```




Partition Strategy
Master
Slave
Slave
Slave
OR
Node 1
Node 2
Node 3

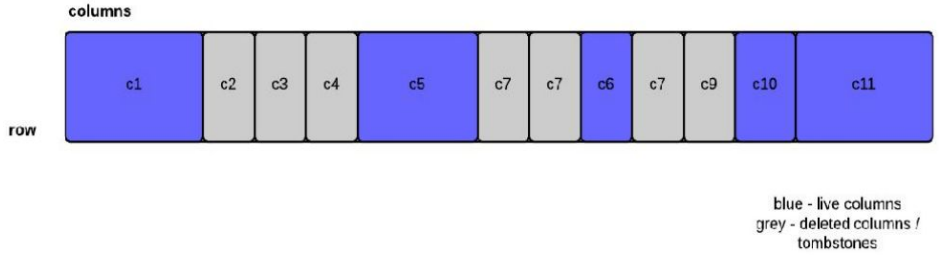
Write in Cassandra

- 1 SSTable per column Family
- Bloom Filter index for Read performance

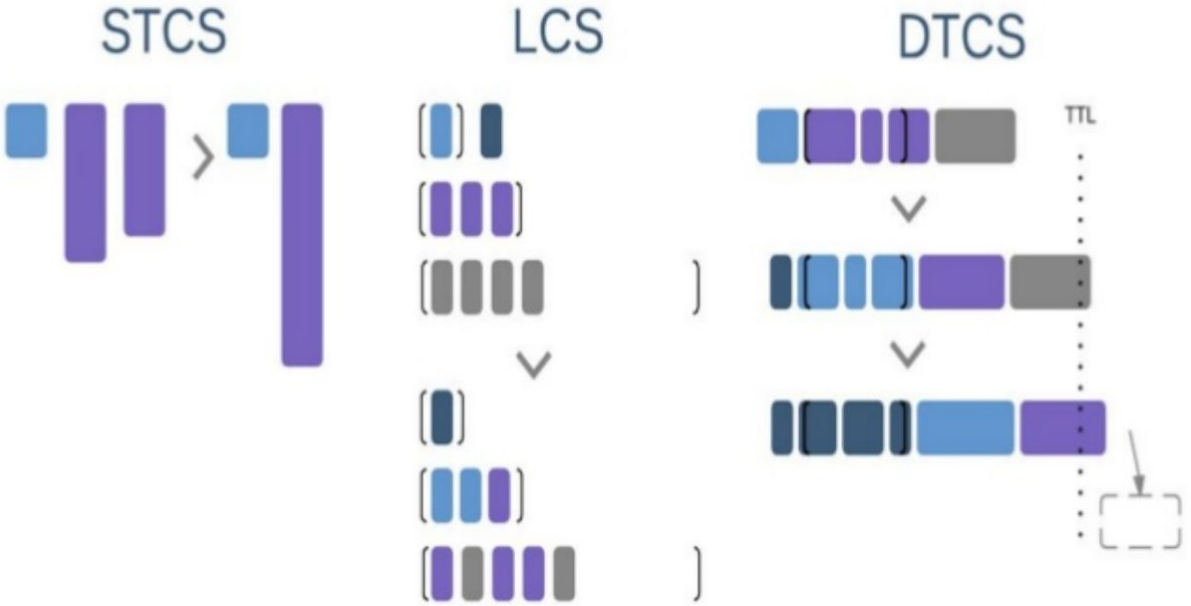


Tombstones

- Deleted data is MARKED as Removed == Tombstone
- Data is deleted and removed during compaction
- Compaction can happen in few days depending of the configs.
- Queries on partition with lots of tombstones requires lots of filtering which can slow down the CASS performance.
- Collections operations can lead to tombstones depending on what you do.
- There are Compaction Trade-Offs.



Compaction Strategy



- **STCS**
  - Default
  - Insert-Heavy
  - General Workloads
- **LCS**
  - Read Heavy
  - More Updates than Inserts
- **DTCS**
  - Time Series
  - Inserts out of order
  - Updates for old data

Anti-Patterns

Anti-Patterns

- Using Cassandra as a queue or queue-like table
  - Tombstones
  - Lots of deleted columns(expiry) and slice-queries don't play well
  - http://www.datastax.com/dev/blog/cassandra-anti-patterns-queues-and-queue-like-datasets
- CQL Nulls
  - Reading Tombstones
  - Write NULL create tombstones
- Intensive Updates on SAME column
  - Sensor table (**ID**,VALUE)
  - Physical Limits
  - Solution: Timestamp as cluster key.

Benchmark: 1 million writes per second

Per Node Activity

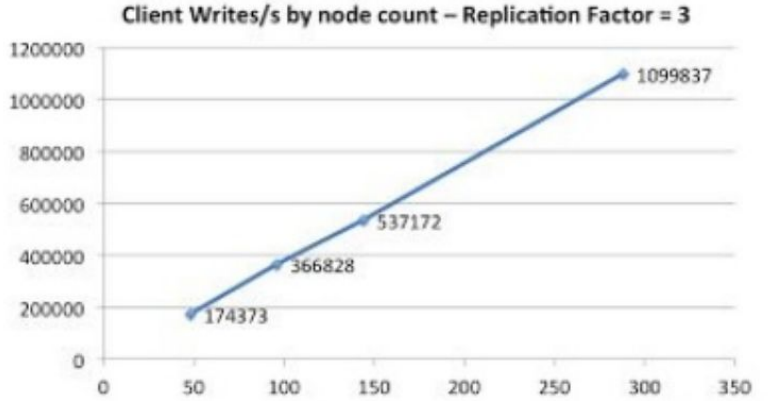
Per Node	48 Nodes	96 Nodes	144 Nodes	288 Nodes
Per Server Writes/s	10,900 w/s	11,460 w/s	11,900 w/s	11,456 w/s
Mean Server Latency	0.0117 ms	0.0134 ms	0.0148 ms	0.0139 ms
Mean CPU %Busy	74.4 %	75.4 %	72.5 %	81.5 %
Disk Read	5,600 KB/s	4,590 KB/s	4,060 KB/s	4,280 KB/s
Disk Write	12,800 KB/s	11,590 KB/s	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

Node specification – Xen Virtual Images, AWS US East, three zones

- Cassandra 0.8.6, CentOS, SunJDK6
- AWS EC2 m1 Extra Large – Standard price $ 0.68/Hour
- 15 GB RAM, 4 Cores, 1Gbit network
- 4 internal disks (total 1.6TB, striped together, md, XFS)

NETFLIX

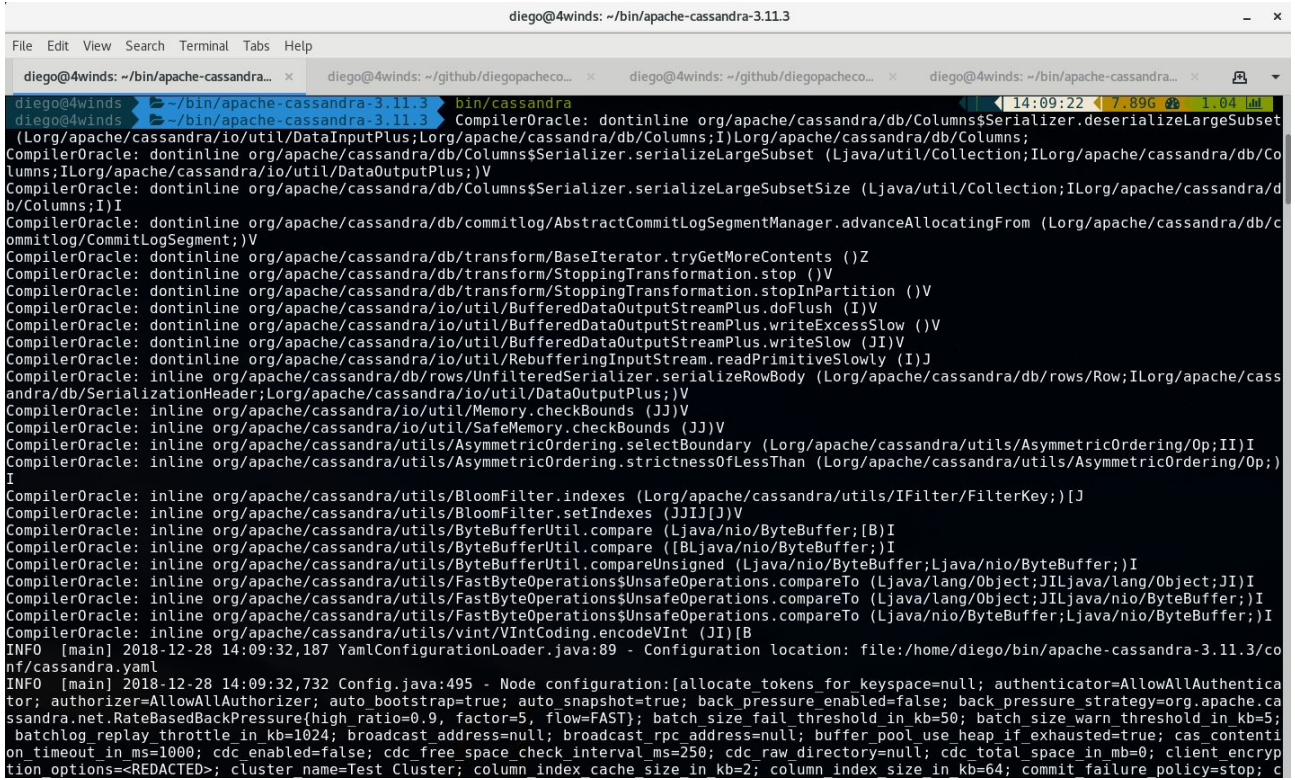
Scale-Up Linearity



NETFLIX

http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on.html

Running Cass (local)



Create Keyspace

keyspace.cql

```
1  CREATE KEYSPACE CLUSTER_TEST WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };
2  USE CLUSTER_TEST;
3  CREATE TABLE TEST ( key text PRIMARY KEY, value text);
4  INSERT INTO TEST (key,value) VALUES ('1', 'works');
5  SELECT * from CLUSTER_TEST.TEST;
```

```
29 bin/nodetool status
30
31 Datacenter: datacenter1
32 =======================
33 Status=Up/Down
34 |/ State=Normal/Leaving/Joining/Moving
35 --  Address    Load       Tokens       Owns (effective)  Host ID                               Rack
36 UN  127.0.0.1  204.48 KiB  256          100.0%            a9cbbaab-aa9a-4396-8d0a-1c78b8b38b17  rack1
37
38 bin/cqlsh
39
40 Connected to Test Cluster at 127.0.0.1:9042.
41 [cqlsh 5.0.1 | Cassandra 3.11.3 | CQL spec 3.4.4 | Native protocol v4]
42 Use HELP for help.
43 cqlsh> SELECT * from CLUSTER_TEST.TEST;
44
45  key | value
46 -----+-------
47    1 | works
48
49 (1 rows)
50 cqlsh>
```

Go Cassandra

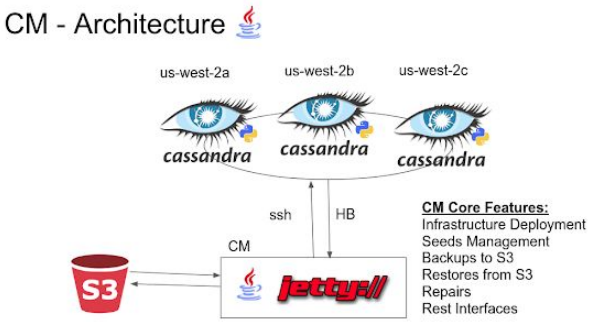
go-cass-sample.go

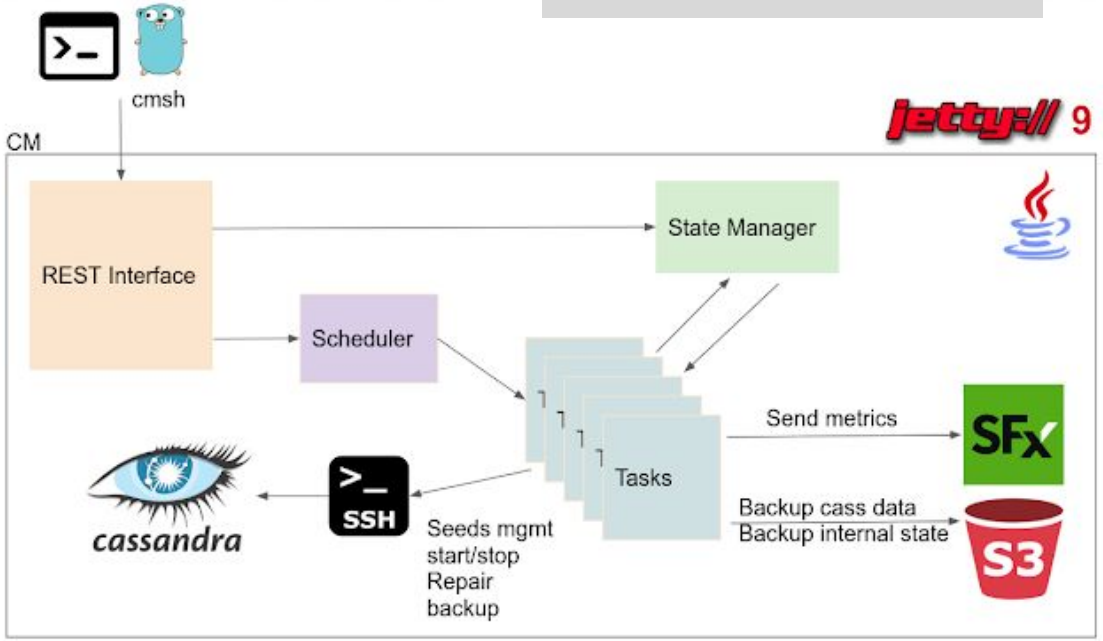
```go
package main

import (
    "fmt"
    "log"

    "github.com/gocql/gocql"
)

func main() {
    cluster := gocql.NewCluster("localhost")
    cluster.Keyspace = "cluster_test"
    cluster.Consistency = gocql.LocalOne
    session, _ := cluster.CreateSession()
    defer session.Close()

    var id string
    var text string

    iter := session.Query("SELECT * from cluster_test.TEST").Iter()
    for iter.Scan(&id, &text) {
        fmt.Println("Data:", id, text)
    }
    if err := iter.Close(); err != nil {
        log.Fatal(err)
    }
}
```

https://github.com/gocql/gocql

Personal Case Study CM

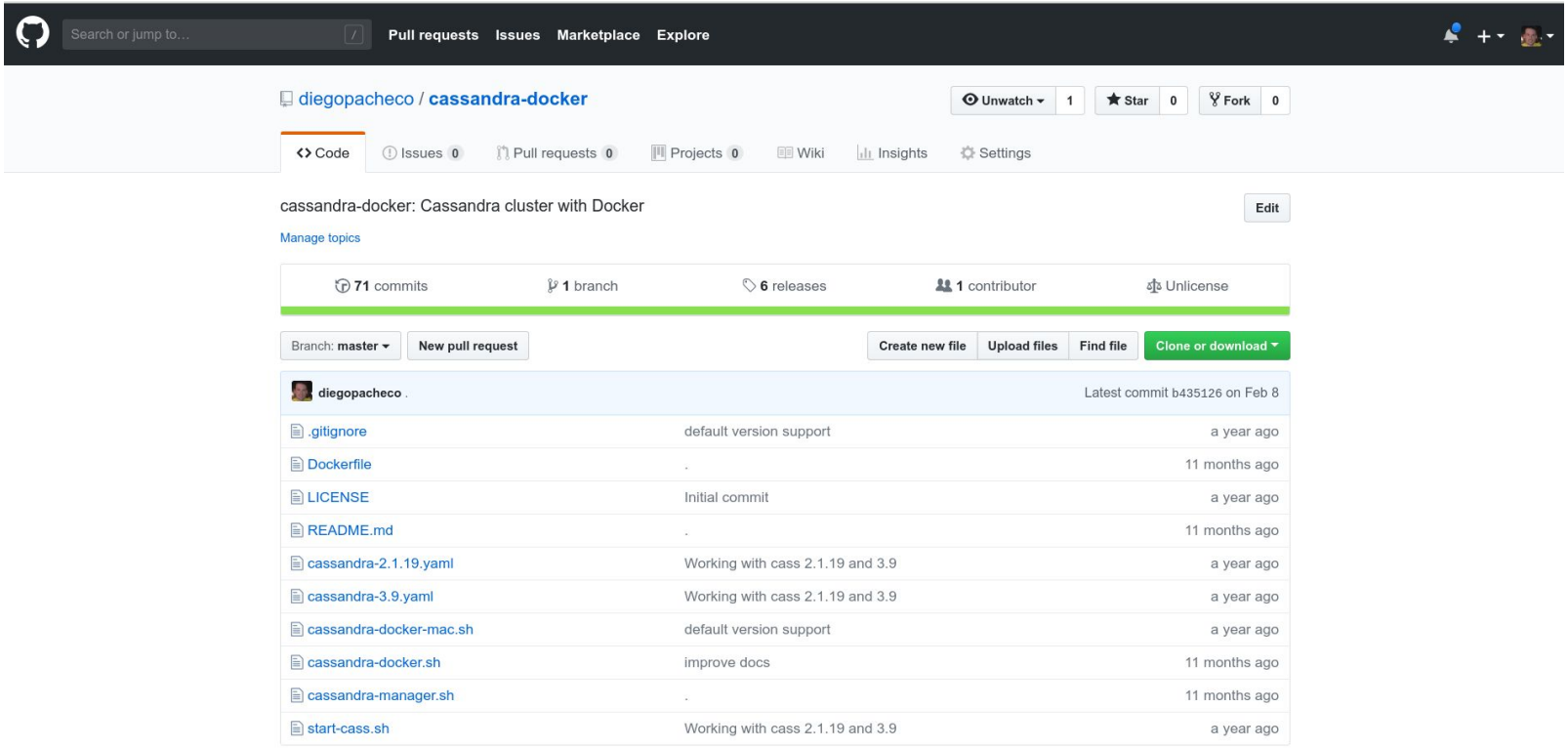
CM



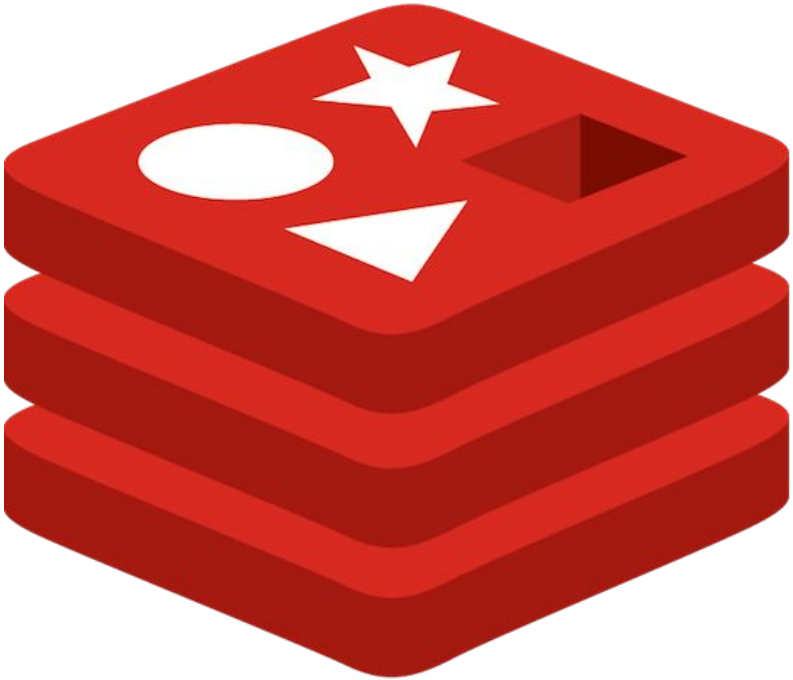


http://diego-pacheco.blogspot.com/2018/07/experiences-building-cassandra.html

Learn & Play Local - diegopacheco/cass-docker



https://github.com/diegopacheco/cassandra-docker



Redis

- In-memory
- Key/Value store
- Written in C
- FAST
- Very robust
- Lots of specific commands / data structures
- Clients for several languages
- Redis-cluster: Master/Slave

Redis Use Cases

- Cache
- Message Queue
- Primary Data Store * -- Terms Apply
- Real Time ingestion buffer
- Handle Real Time analytics
- For management

Redis data strata

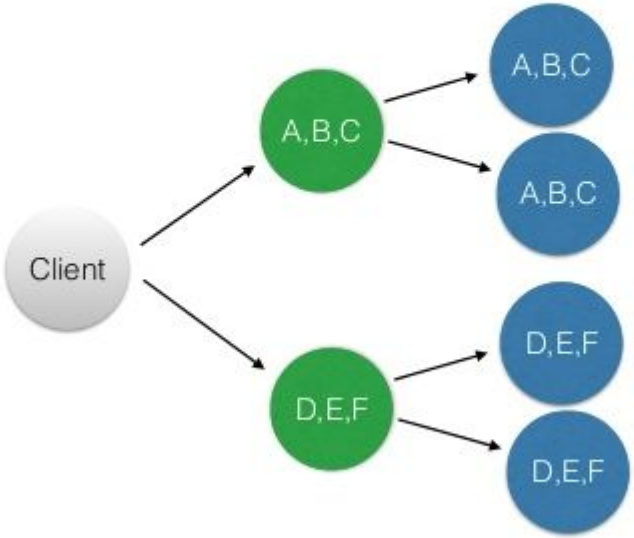
v1.0	Strings
	Lists
	Sets
v1.2	Sorted Sets
v2.0	Hashes
v2.2	Bit arrays
v2.8.9	HyperLogLog
v3.2	Geo Sets
	Bit fields
v4	Streams (?)
	MODULES!

Redis CLI

```
diego@4winds:~$ redis-cli
127.0.0.1:6379> set users-count 0
OK
127.0.0.1:6379> set users-count 1
OK
127.0.0.1:6379> set users-count 2
OK
127.0.0.1:6379> get users-count
"2"
127.0.0.1:6379> keys *
1) "users-count"
127.0.0.1:6379>
```

Redis Cluster

Sharding and replication (asynchronous).

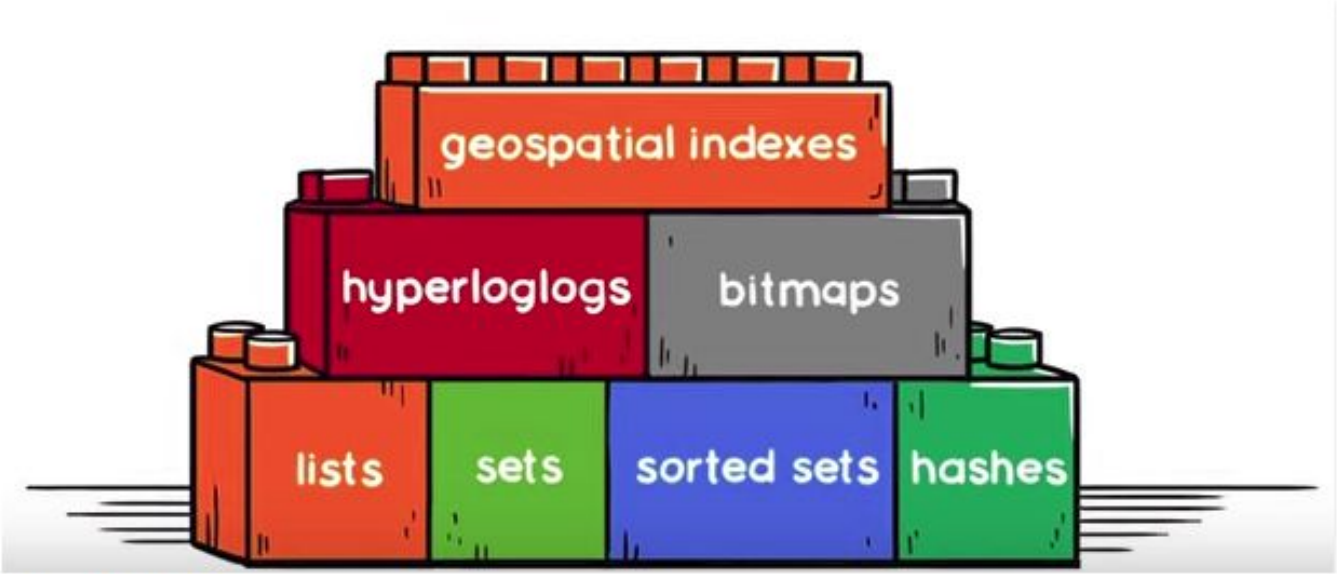


Redis Cluster Issues | Dynomite to Rescue

Redis (w/failover)

- Split-brain
- Drops all data on one component
- Not even close to consistent.

Redis data types

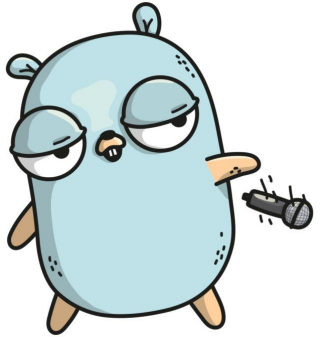


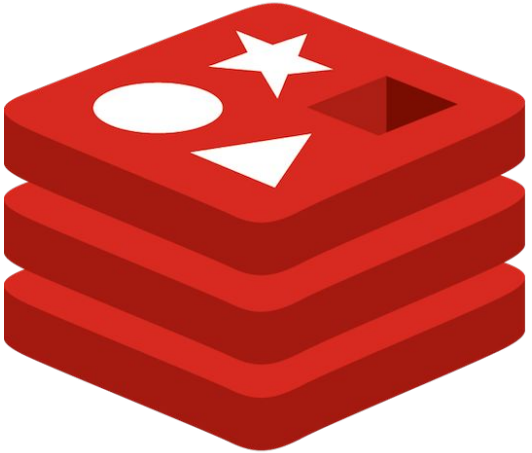
Picture credit : RedisLabs

Redis Go

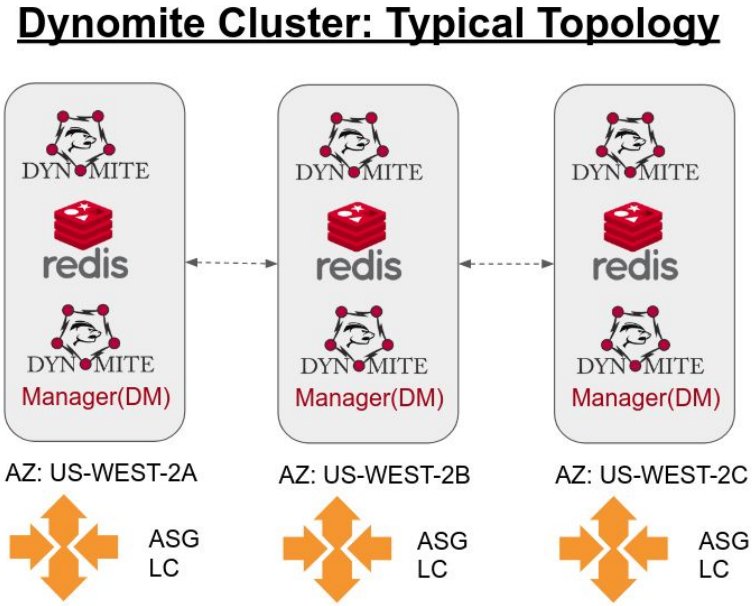
go-redis-sample.go

```go
package main

import (
    "fmt"
    "net/http"

    "github.com/go-redis/redis"
)

func handle(w http.ResponseWriter, r *http.Request) {
    client := redis.NewClient(&redis.Options{Addr: "localhost:6379"})
    client.Incr("kcount")

    val, err := client.Get("kcount").Result()
    if err != nil {
        panic(err)
    }
    result := string("key count: " + string(val))

    fmt.Print(result)
    fmt.Fprintf(w, result)
}

func main() {
    fmt.Print("Serving at 0.0.0.0:9090... ")
    http.HandleFunc("/", handle)
    http.ListenAndServe("0.0.0.0:9090", nil)
}
```





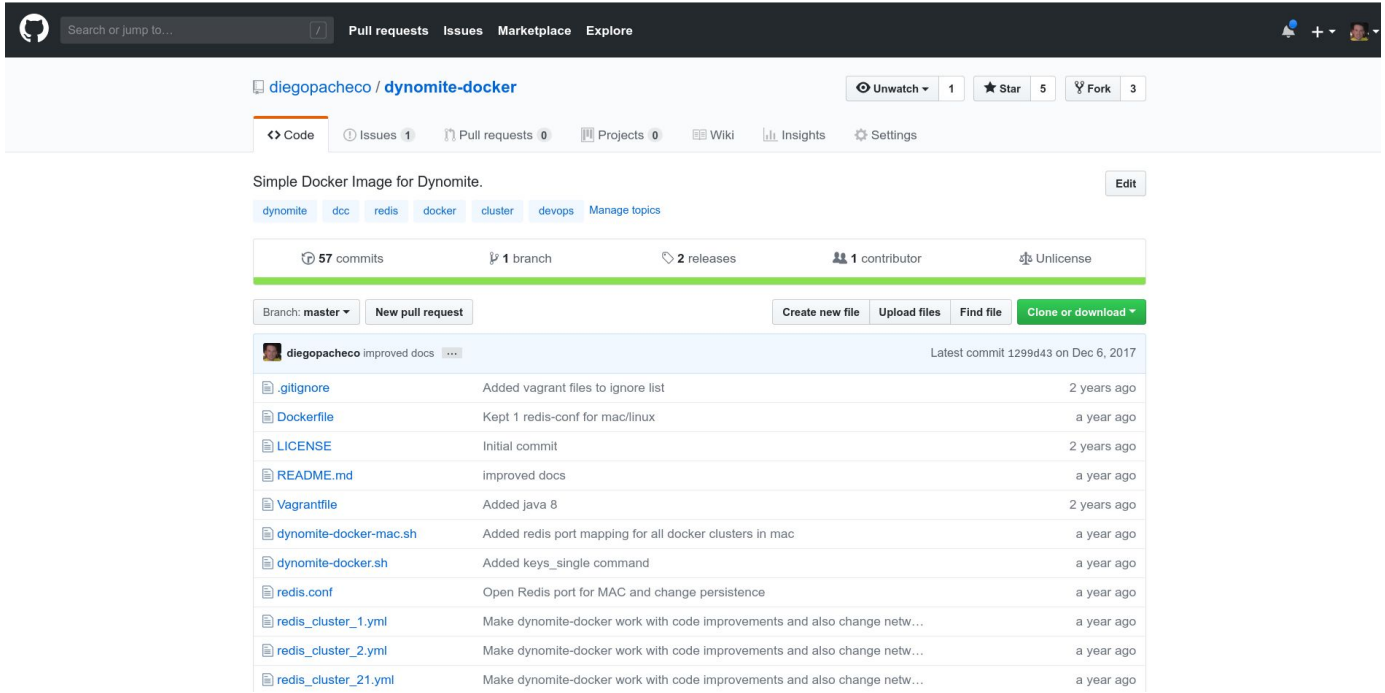
Personal Case Study - Dynomite / DM use Case



NETFLIX OSS

https://www.youtube.com/watch?v=Z4_rzsZd70o&feature=youtu.be

Learn & Play Local - diegopacheco/dynomite-docker



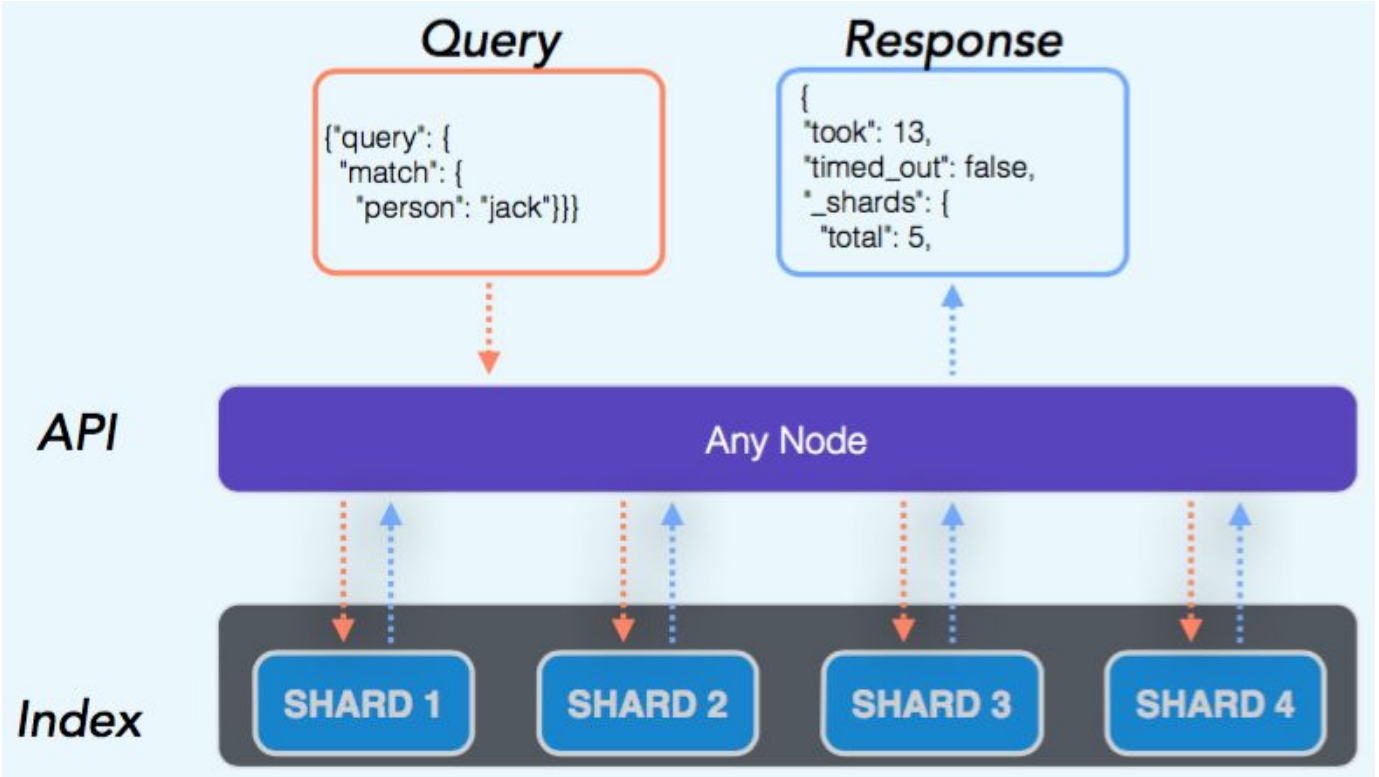
https://github.com/diegopacheco/dynomite-docker

elasticsearch

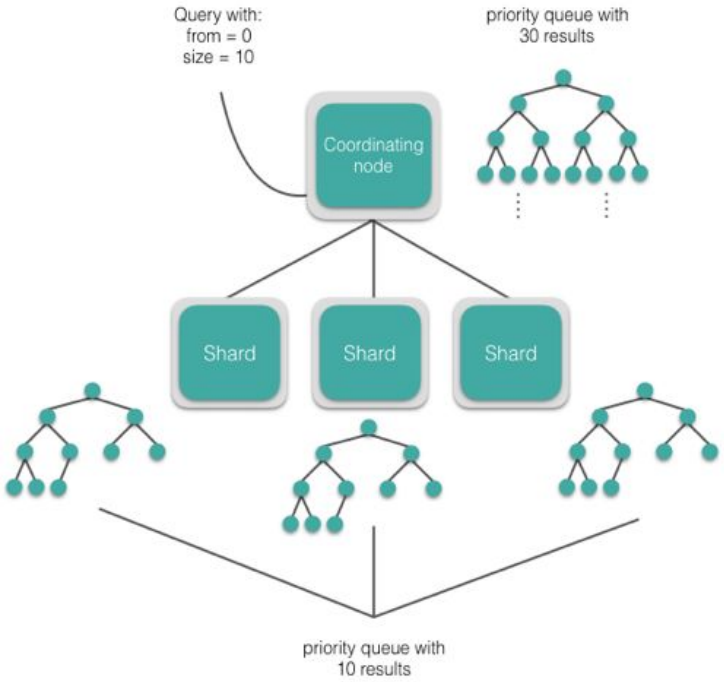
ES

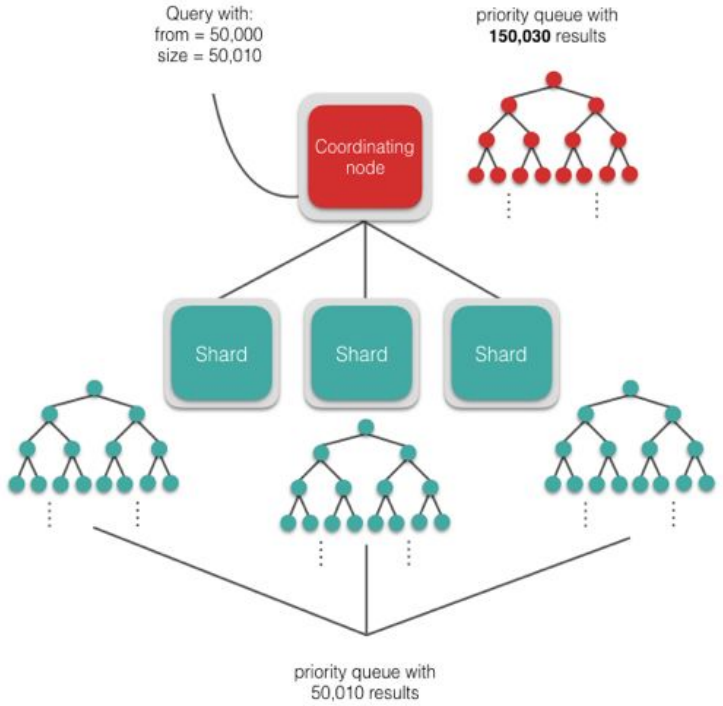
- 2010
- Cluster
- Uses Lucene Engine(Apache)
- Full Text Search Engine
- Documents (JSON) - Indexing / Feeding Process
- Distributed
- Geo Processing Support - Geo Search
- It can't be used as Source of Truth - Because loses data(Split Brain).

REST API

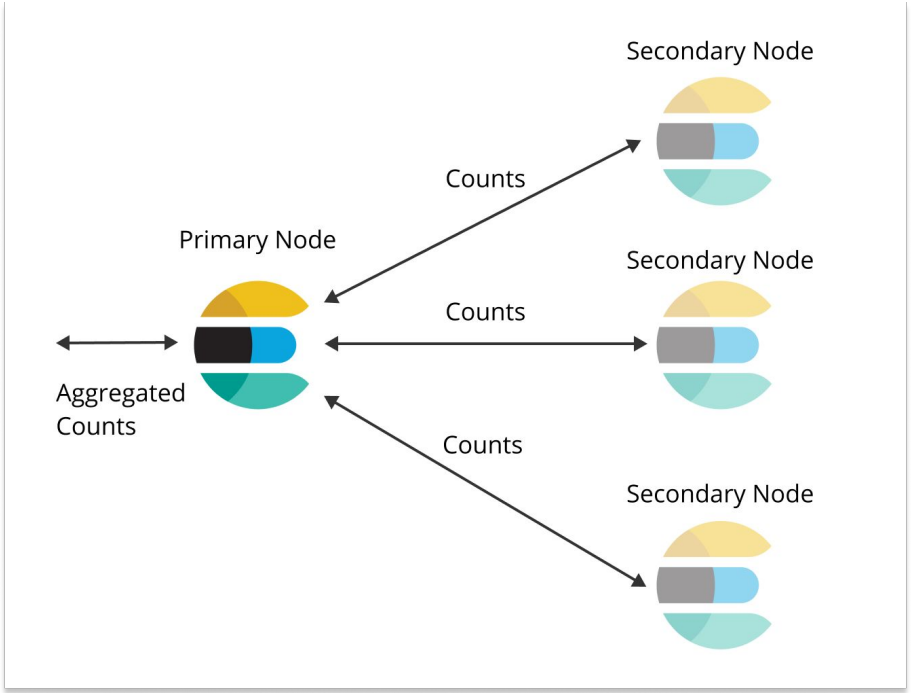

Query
{"query": {
"match": {
"person": "jack"}}}
Response
{
"took": 13,
"timed_out": false,
"_shards": {
"total": 5,
API
Any Node
Index
SHARD 1
SHARD 2
SHARD 3
SHARD 4

ES Cluster


Query with:
from = 0
size = 10
priority queue with
30 results
Coordinating
node
Shard
Shard
Shard
priority queue with
10 results


Query with:
from = 50,000
size = 50,010
priority queue with
150,030 results
Coordinating
node
Shard
Shard
Shard
priority queue with
50,010 results

ES Cluster


Secondary Node
Counts
Primary Node
Secondary Node
Counts
Aggregated
Counts
Counts
Secondary Node

ES Cluster - Create Docs

```
POST localhost:9200/accounts/person/1
{
    "name" : "John",
    "lastname" : "Doe",
    "job_description" : "Systems administrator and Linux specialit"
}
```

The response will return information about the document creation:

```
{
    "_index": "accounts",
    "_type": "person",
    "_id": "1",
    "_version": 1,
    "result": "created",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "created": true
}
```

ES Cluster - Retrieve it

```
GET localhost:9200/accounts/person/1
```

The result will contain metadata and also the full document (shown in the `_source` field) :

```
{
    "_index": "accounts",
    "_type": "person",
    "_id": "1",
    "_version": 1,
    "found": true,
    "_source": {
        "name": "John",
        "lastname": "Doe",
        "job_description": "Systems administrator and Linux specialit"
    }
}
```

ES Cluster - Search

```
GET localhost:9200/_search?q=john
```

This search will return both documents, since both of them include *john*:


```
{
    "took": 58,
    "timed_out": false,
    "_shards": {
        "total": 5,
        "successful": 5,
        "failed": 0
    },
    "hits": {
        "total": 2,
        "max_score": 0.2876821,
        "hits": [
            {
                "_index": "accounts",
                "_type": "person",
                "_id": "2",
                "_score": 0.2876821,
                "_source": {
                    "name": "John",
                    "lastname": "Smith",
                    "job_description": "Systems administrator"
                }
            },
            {
                "_index": "accounts",
                "_type": "person",
                "_id": "1",
                "_score": 0.28582606,
                "_source": {
                    "name": "John",
                    "lastname": "Doe",
                    "job_description": "Systems administrator and Linux specialist"
                }
            }
        ]
    }
}
```


APACHE
kafka®

KAFKA's Distributed Log

Append Only

Linear Scans

Kafka Partitions

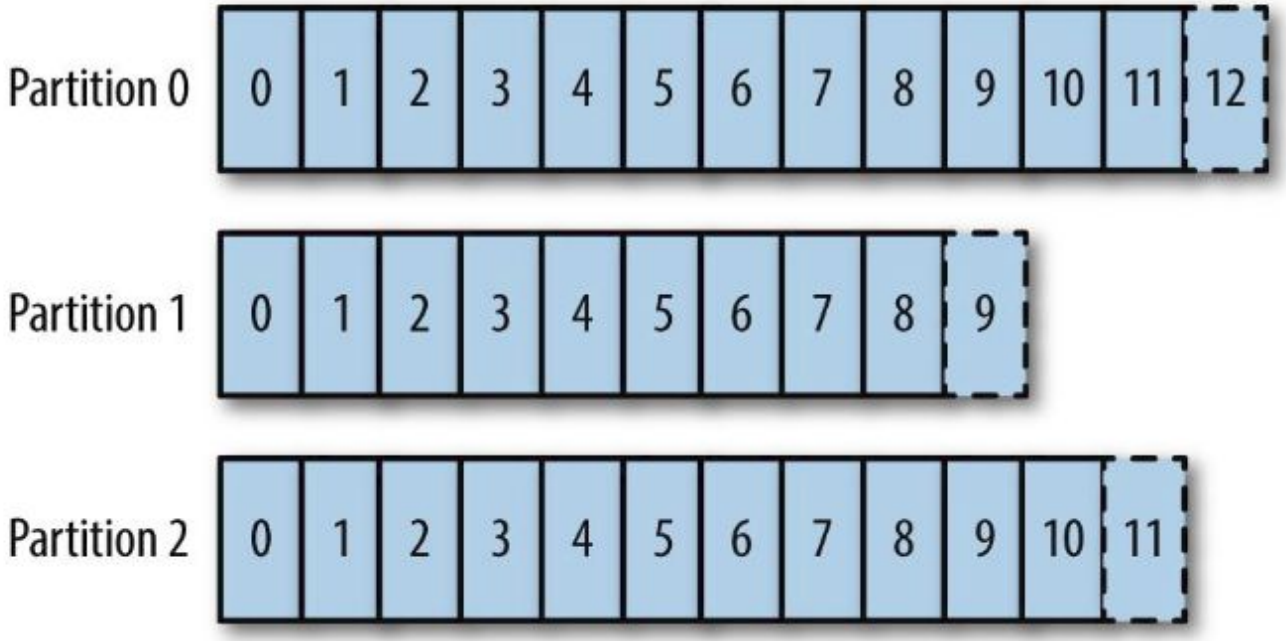


Figure 2-9. By partitioning the log, we allow each partition to act independently of all other partitions. This lets us horizontally scale the write throughput.

Log Compaction

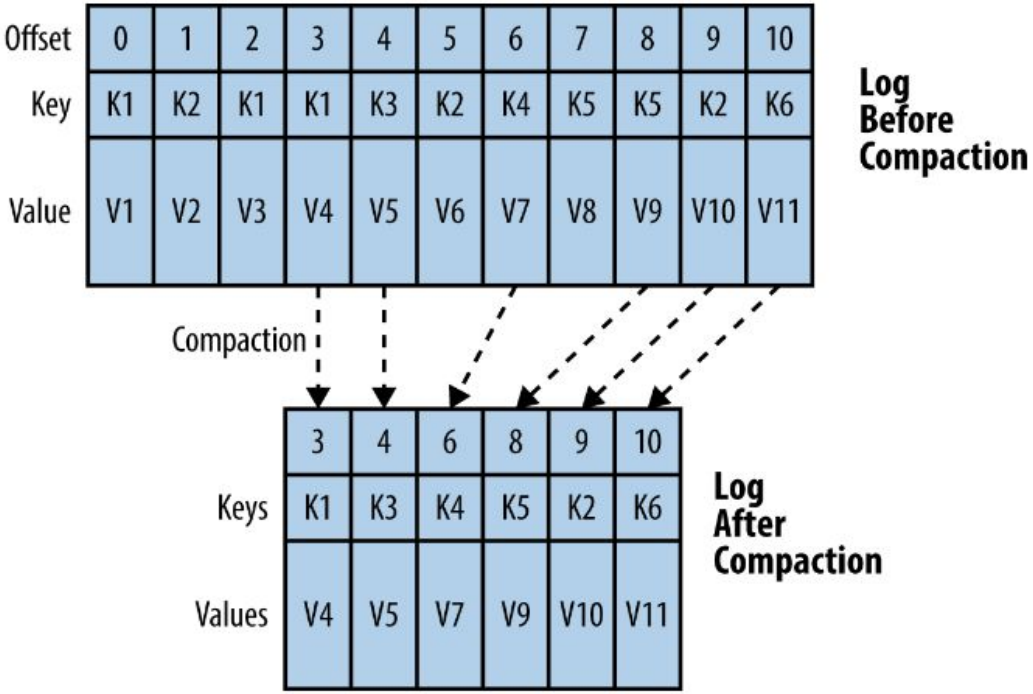
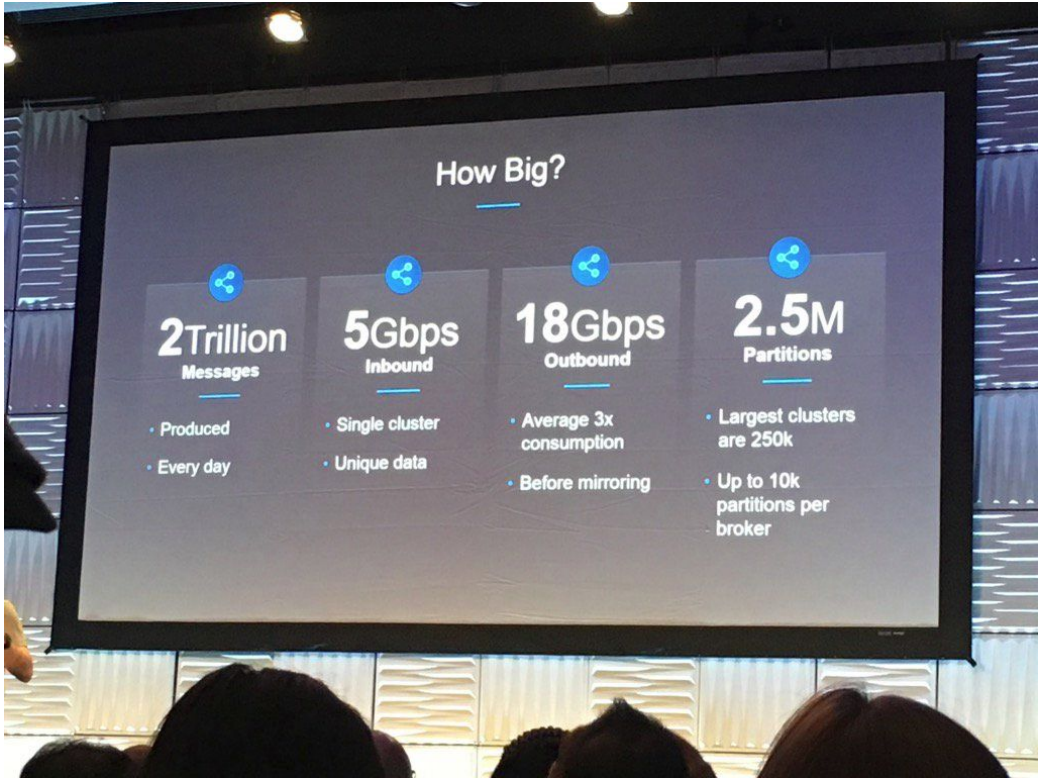
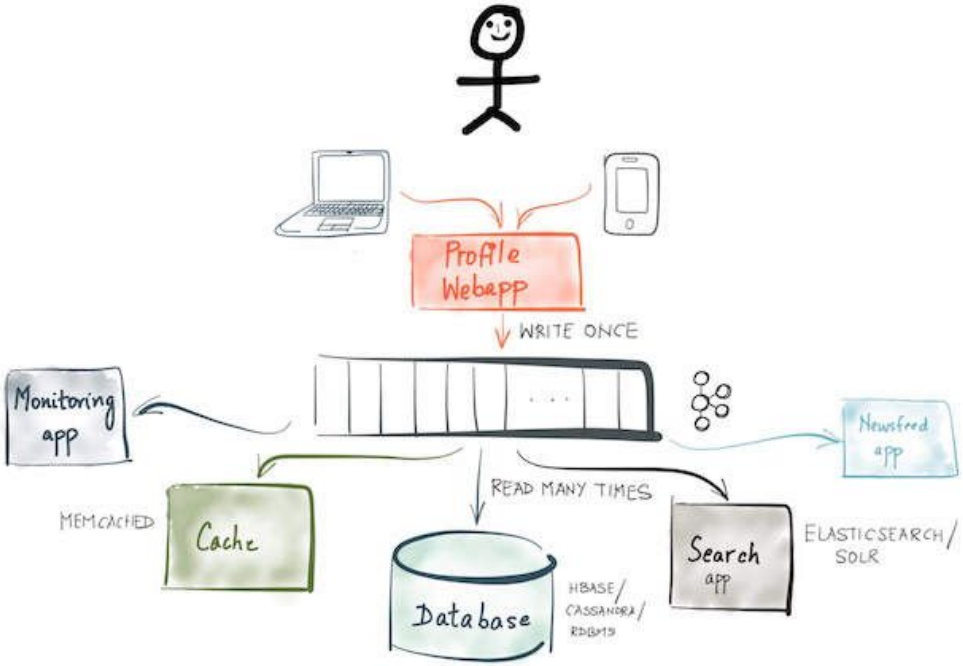


Figure 3-5. Log compaction ensures that the log retains only the latest update for each key. This is useful for modeling updates to mutable data as a log.

Kafka LinkedIN Numbers 2017



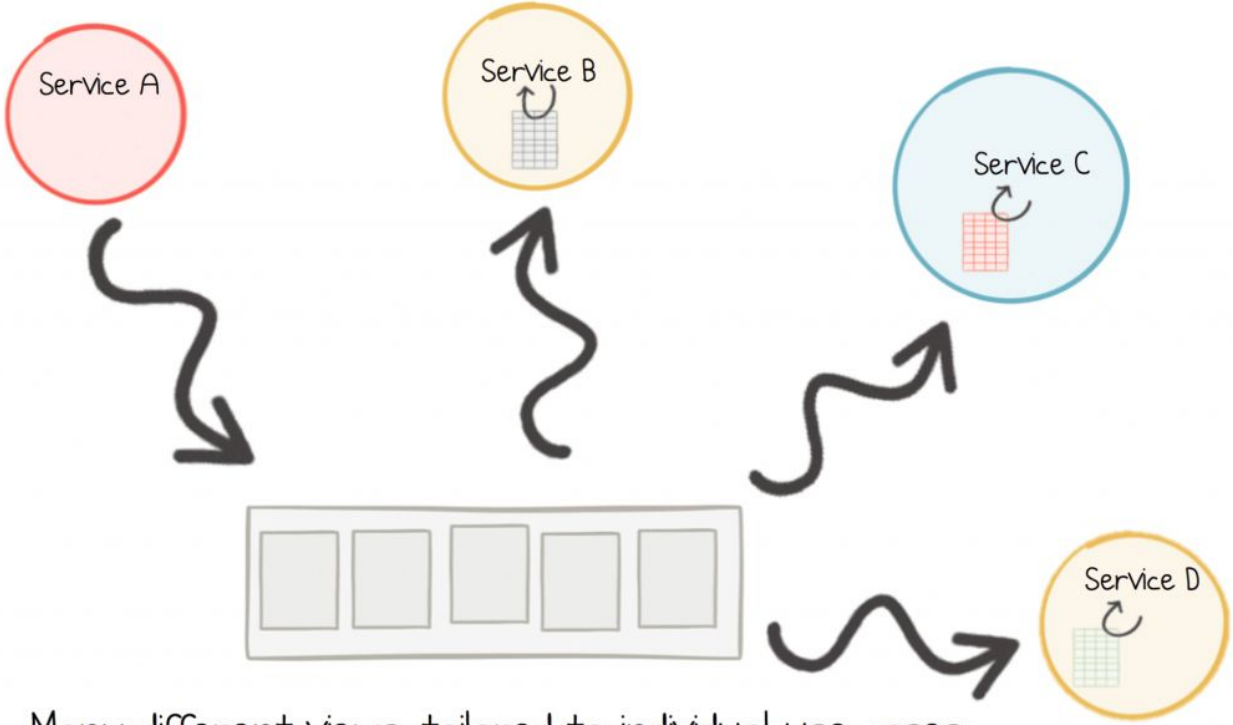
CQRS / ES



1) How we do aggregate queries without a central DB?

2) How do we propagate and notify changes?

Kafka and Microservices



Amazon Kinesis

Google Cloud Pub/Sub

Spark

Flink

confluent
Real-time streams powered by Apache Kafka

Learn & Play Local

Having fun with Kafka

Installing Kafka

```
sudo wget http://www.eu.apache.org/dist//kafka/0.8.2.0/kafka_2.10-0.8.2.0.tgz
tar -xzf kafka_2.10-0.8.2.0.tgz
cd kafka_2.10-0.8.2.0
```

START Zookeper and KAFKA SERVER

```
sudo nohup bin/zookeeper-server-start.sh config/zookeeper.properties &
sudo nohup bin/kafka-server-start.sh config/server.properties &
```

Create a Topic and List all Topics

```
sudo bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic
sudo bin/kafka-topics.sh --list --zookeeper localhost:2181 &
```

Create and send message to kafka topic

```
sudo bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
```

Start a consumer and consume all messages from begining

```
sudo bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test --from-beginning
```

APACHE kafka®

Apache Zookeeper

https://gist.github.com/diegopacheco/b1979c18357e57cc8218

NoSQL: Redis, Cass, ES, Kafka

Diego Pacheco