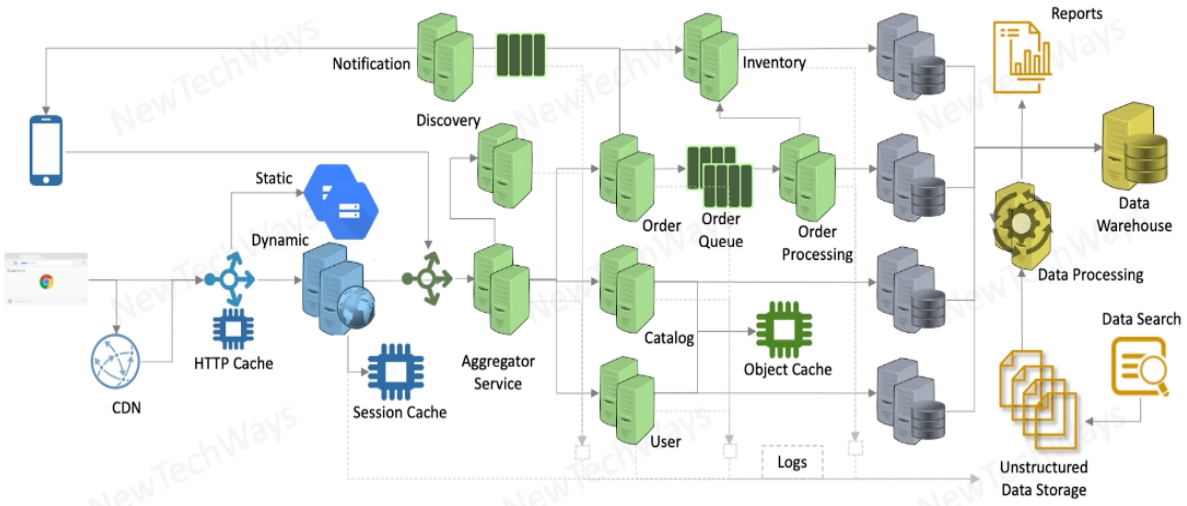
System Design Tech Stack

Reference System:



Web application – Solutions:

Static Content →

- Apache Web Server
- Nginx Web Server
- Cloud Storage

Dynamic Content →

- Web Server – Apache HTTPD, NodeJS
- Java Container – Tomcat, Jetty, Spring Boot

Content Caching →

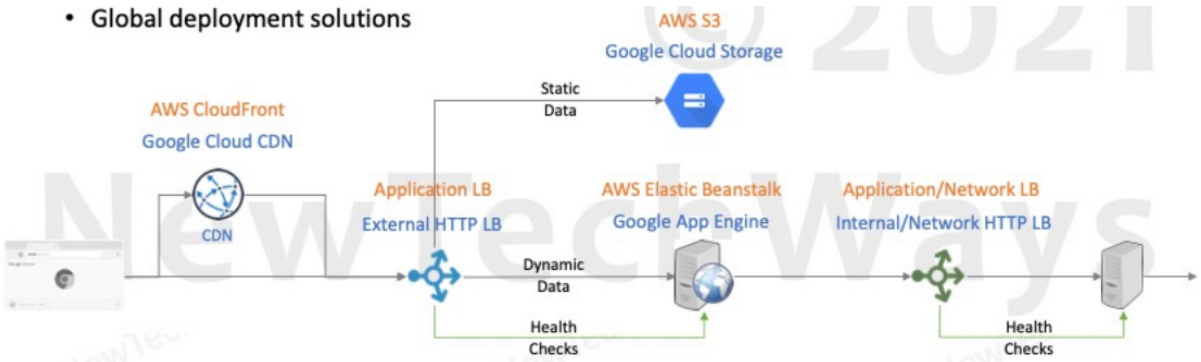
- Nginx

Content Distribution →

- CDN

Cloud Solutions for Web:

- Global deployment solutions



Load Balancer types based on content type:

- Static data – Cloud Storage, Dynamic Data – Web Application

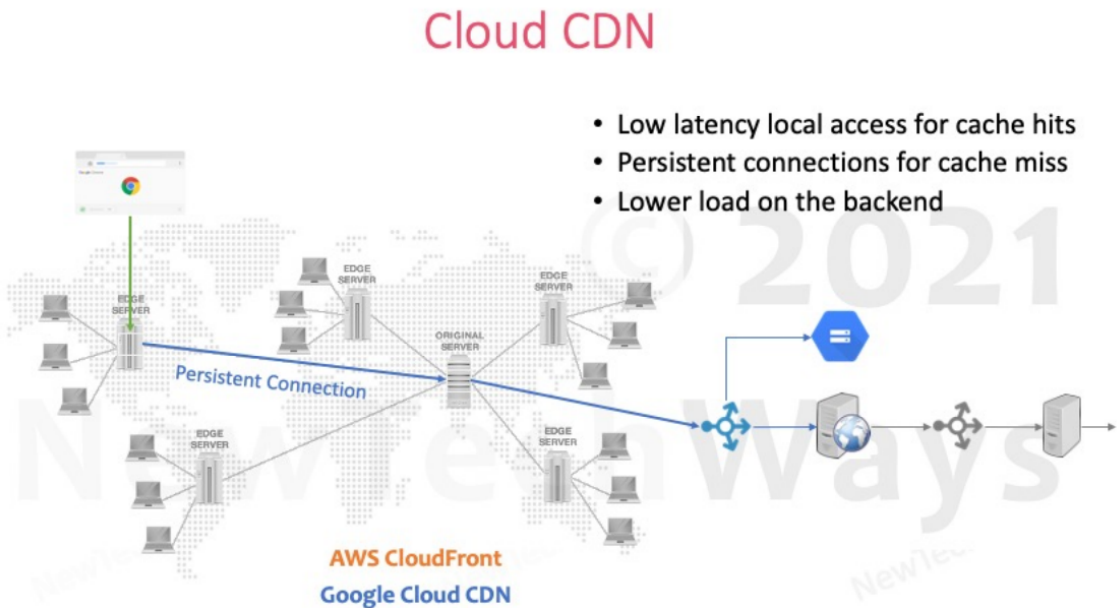
- AWS – Application LB
- Google – External HTTP LB
- Level 7 LB can inspect content of the request.

Load Balancer based on URL:

- AWS – Network LB
- Google – Network HTTP LB

If it's a Network LB CDN can directly talk to Cloud Storage

Cloud CDN →



Services – Solutions:

Web Containers →

- Rest and Spring Containers

Object Caching →

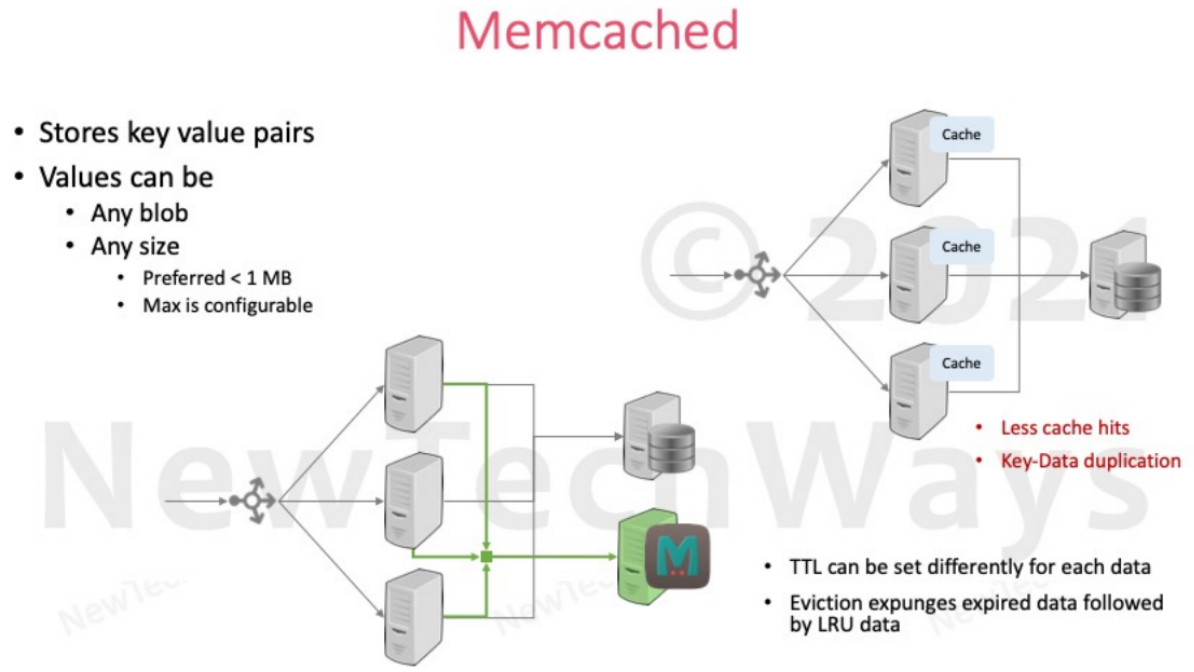
- Memcache
- Redis

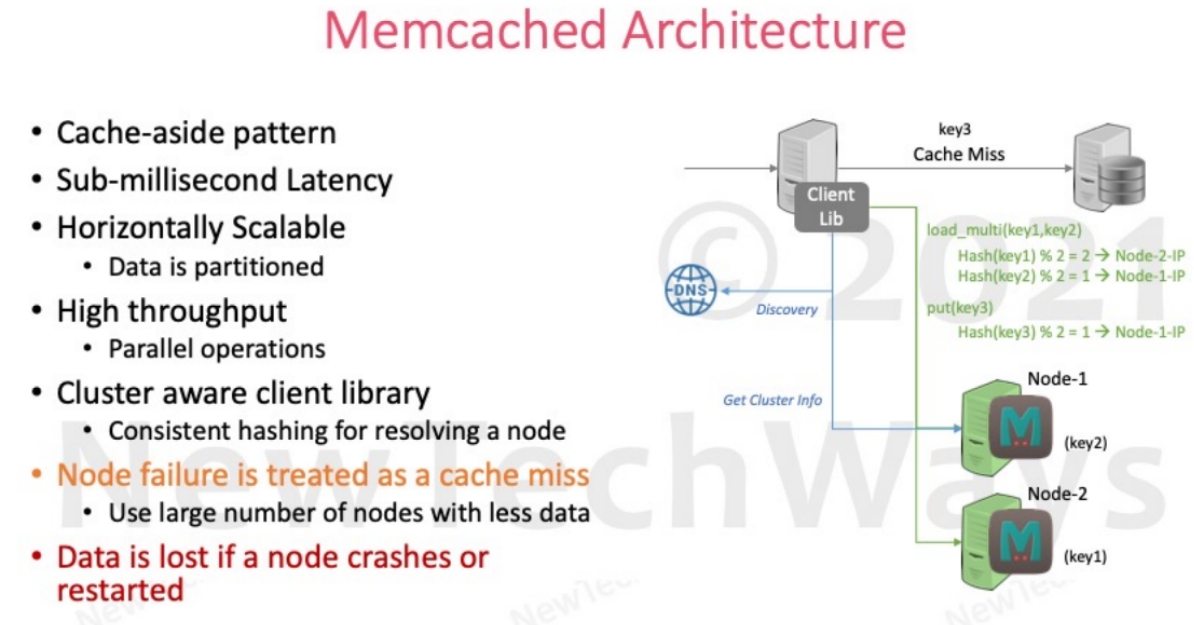
Async Messaging →

- Redis
- RabbitMQ
- Kafka

Object Caching →

Memcache:



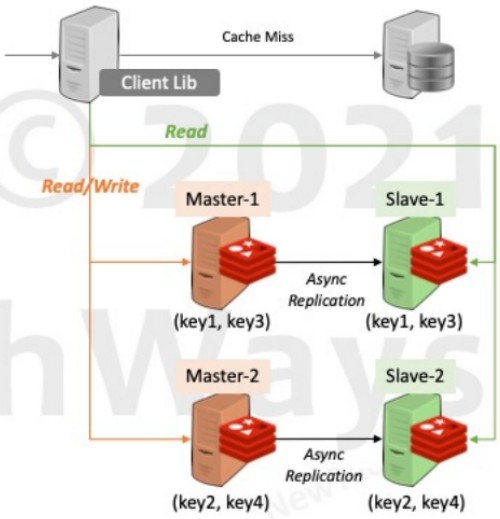


- Client knows the IP address of all the nodes in a cluster. Client uses a memcache client lib. Client lib makes a DNS call to get any one of the Node info. Client makes a call to the Node to get the cluster info and in return gets the info of the entire cluster (how any nodes are there, IP address of all the nodes)
- Cache-aside pattern – Client can only retrieve those value which client has put.
- Client knows which Node has the value by **Hash(key) % N = Node number** (N – Number of nodes) or **Consistent hashing**.
- Load multiple key into one API

Redis Cache:

Redis Cache

- Much like Memcached
- It is a [Key → Data structure] store
  - Strings, Lists, SortedSets, Maps, ...
- Data-Store mode for Persistence
  - Stores data on a disk
  - Allows backups
  - Can be started pre-populated with a backup
- Data Replication
  - Asynchronous and synchronous
  - Read load distribution
  - High Availability
- Can also be used as a messaging queue
- Data-Store mode requires fixed number of nodes
  - Cache mode is suitable for node scaling



Cloud Caching Solution:

Cloud Caching Solutions

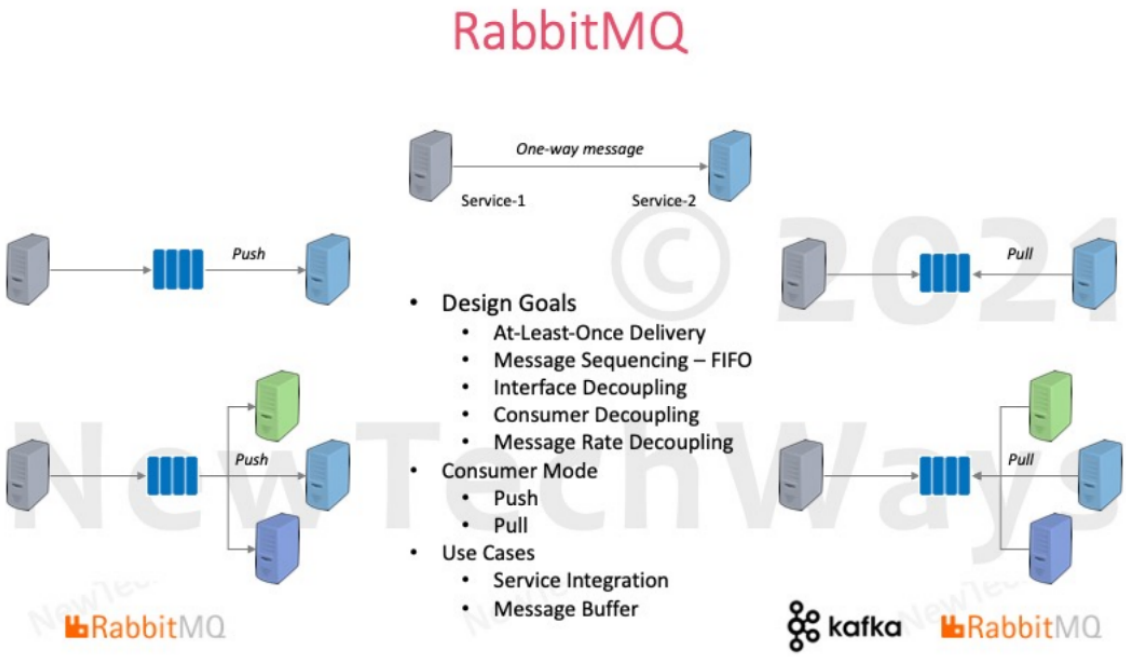
- AWS Elastic Cache
  - Memcached
  - Redis

- Google Memorystore
  - Memcached
  - Redis

- Fully Managed
- Sub-millisecond latency
- Scalable to 5 TB
- Highly available (99.9%)

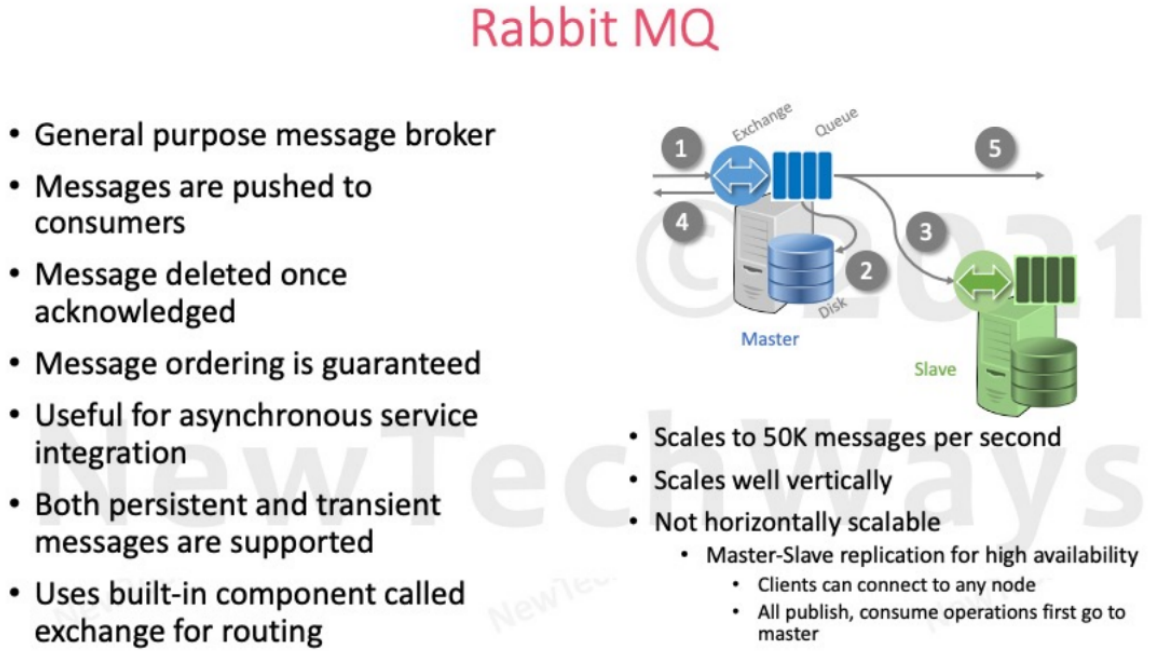
Async Messaging →

RabbitMQ:



- RabbitMQ can be a push or a pull-based messaging whereas Kafka can only be a pull-based messaging.
- RabbitMQ is generally used as push-based messaging system.
- Service Integration - At times when there is message or there might not be message in the queue push-based messaging system is recommended (RabbitMQ) as in this case the consumer will unnecessarily poll and waste resources.
- Message Buffer - At times when the volume of message is very high (Streaming workflows) and queue is always full then pull-based messaging system is recommended (Kafka).

RabbitMQ Architecture:

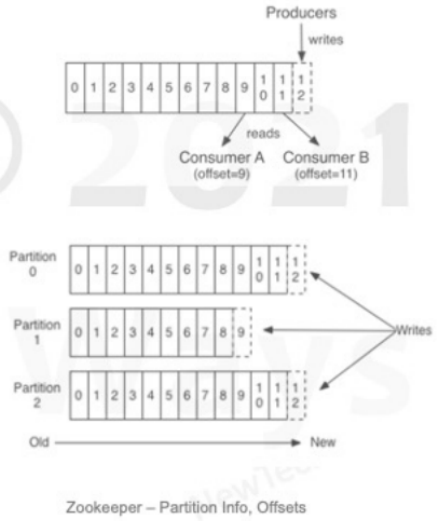


- Transient messaging – when we need very high-speed processing of messages and we do not care if messages are lost. Messages not stored to DB.
- Persistent messaging – when we need guarantee that the message is at least delivered once. Messages stored to DB.

Kafka Architecture:

Kafka

- Performance
  - Million messages per second
  - Sequential writes and reads on log files
  - Relies on page cache for quick reads
- Horizontally Scalable
  - Topics are partitioned
- Order of data guaranteed only within a partition
  - Producer can decide the partition
- Messages are not deleted, so can be replayed
- Consumers can only pull the data
  - No push by Kafka
  - Not designed for service integration
- Useful for streaming analytical workloads
  - Where high throughput is required
  - Click streams, Page views, Logging, Ingestion, Security

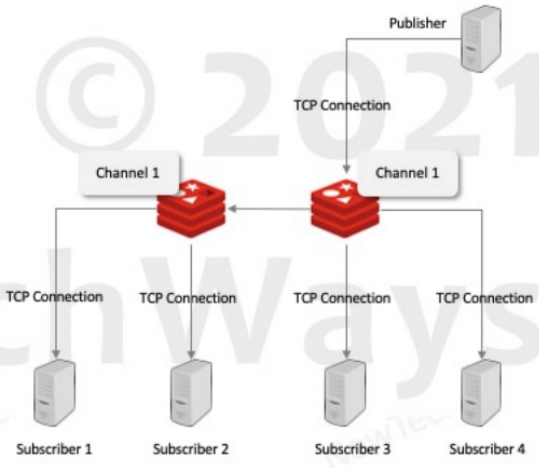


Zookeeper – Partition Info, Offsets

Redis Pub/Sub:

Redis Pub/Sub

- For short lived messages with no persistence
  - Much like a synchronous call
  - Fire and forget
    - No delivery guarantee
- Million operations per second
- Useful for making dashboards
  - Leaderboard
- Comparison
  - Kafka
    - No push
    - Writes to log
  - RabbitMQ Transient
    - Delivery acknowledgement
    - Deletion of delivered messages



- Fast messaging without any persistence, we should consider Redis.
- In Redis topics are called as Channel.

Cloud MQ Solutions:

Cloud MQ Solutions

Community	AWS	Google Cloud
Rabbit MQ	SQS	Pub/Sub
Kafka	Kinesis	

Datastore – Solutions:

RDBMS →

- Oracle
- SQL Server

Distributed Databases →

- Key-Value – Dynamo
- Column Family – Big Table, Cassandra HBase
- Document Oriented – MongoDB

RDBMS →

RDBMS

- General purpose database (< 1-5 TB data, 10K connections)
- ACID Transactions
  - Update of multiple records or tables
- Data Consistency
  - Data same for all readers at any given time
- Fixed Schema
  - Data analysis
    - Columns can be selected
    - Rows can be filtered
    - Queries can Join Tables
  - Query pattern can change or evolve
    - Any column can be indexed
    - Indexes can be created whenever required
- Normalized Data
  - Efficient storage and writes
- Overwrite for updates

Possible only on a single node or else it requires 2PC/3PC

Leads to low availability

Impedes application evolution

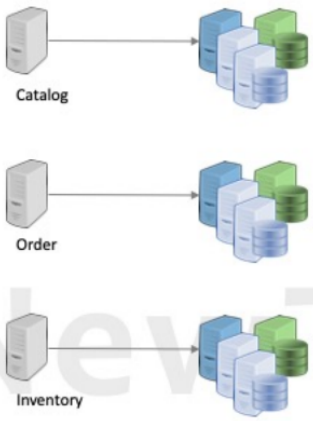
Joins may slow down a system

Allows only one version of data

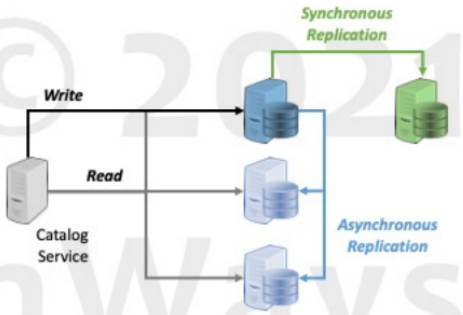
Old design for expensive disk space

RDBMS Scalability Architecture:

RDBMS Scalability Architecture



Vertical Partitioning for Scalability



Replication for Reliability/Availability

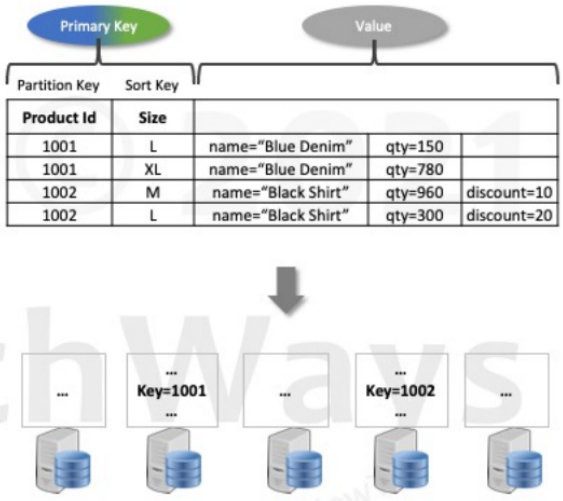
- Separate DB for each service.
- Read replicas (Async data replication).

Distributed Databases →

Dynamo DB:

Amazon DynamoDB

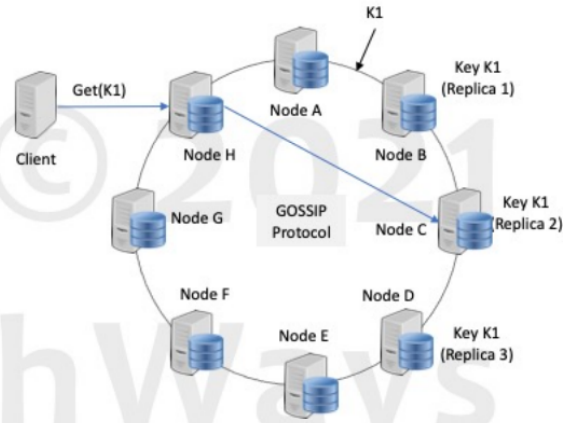
- Key-Value Pair Datastore
- Used for High Scalability & Availability
- Table is a Hash-Map
  - Persistent
  - Distributed
- API
  - Put, Get, Update, Delete, Query
- Index Key – has two parts
  - Partition Key
    - Can have only one attribute
    - Key is hashed to determine the partition
  - Sort Key
    - Determines sort order of an item within a partition
    - Can be used for range query <, >, like
- R/W ops for a key are atomic
  - Table as HashMap



Dynamo DB architecture:

Dynamo DB Architecture

- Suitable for storing small chunks of data
  - Key values less than 1MB
- Peer-To-Peer cluster
  - No master – write on any node
  - Each node responsible for a set of keys
  - Consistent hashing of virtual nodes for key allocation
- Highly Scalable
  - Practically any number of nodes
  - Petabytes of data
  - Can handle 10 million RW ops requests/second
- Highly Available
  - Updates are not rejected even in case of network partitions or server failures
  - Vector clocks and business rules to resolve merge conflicts
- Consistency guarantee is adjustable
  - R+W > N for strong consistency

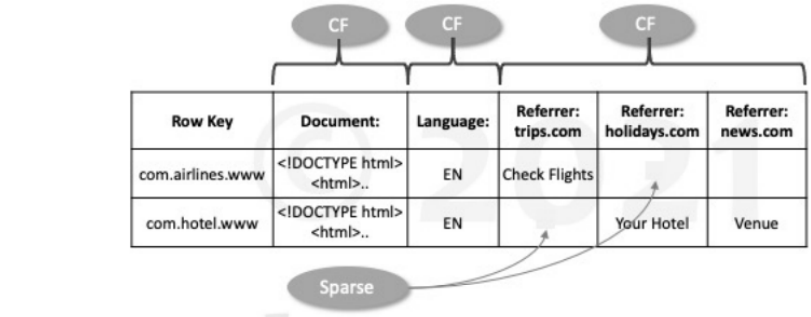


- Ring represents range of values for a partition key column
- Machine IP/Name is hashed to occupy a position on the ring

Google Big Table:

Google Big Table

- Basis for Apache HBase
- Column-Family Storage
- Table as Tree-Map
  - Sparse
  - **Sorted**
  - Persistent
  - Distributed
- Limited CF (< 100)
- CFs are compressed
- Unlimited columns
- R/W ops atomic for a key
- Timestamps for versioning



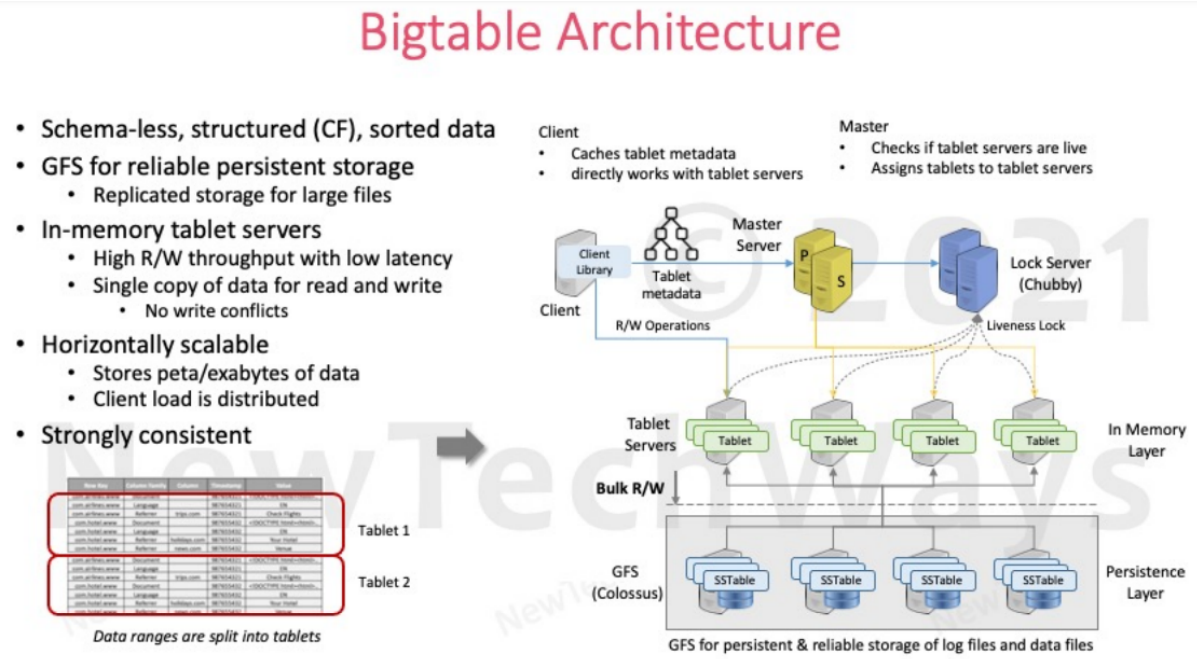
Index Key: Row Key/Col Family:Col Name/timestamp

Sorted

Row Key	Column Family	Column	Timestamp	Value
com.airlines.www	Document		987654321	<!DOCTYPE html><html>..
com.airlines.www	Language		987654321	EN
com.airlines.www	Referrer	trips.com	987654321	Check Flights
com.hotel.www	Document		987655432	<!DOCTYPE html><html>..
com.hotel.www	Language		987655432	EN
com.hotel.www	Referrer	holidays.com	987655432	Your Hotel
com.hotel.www	Referrer	news.com	987655432	Venue

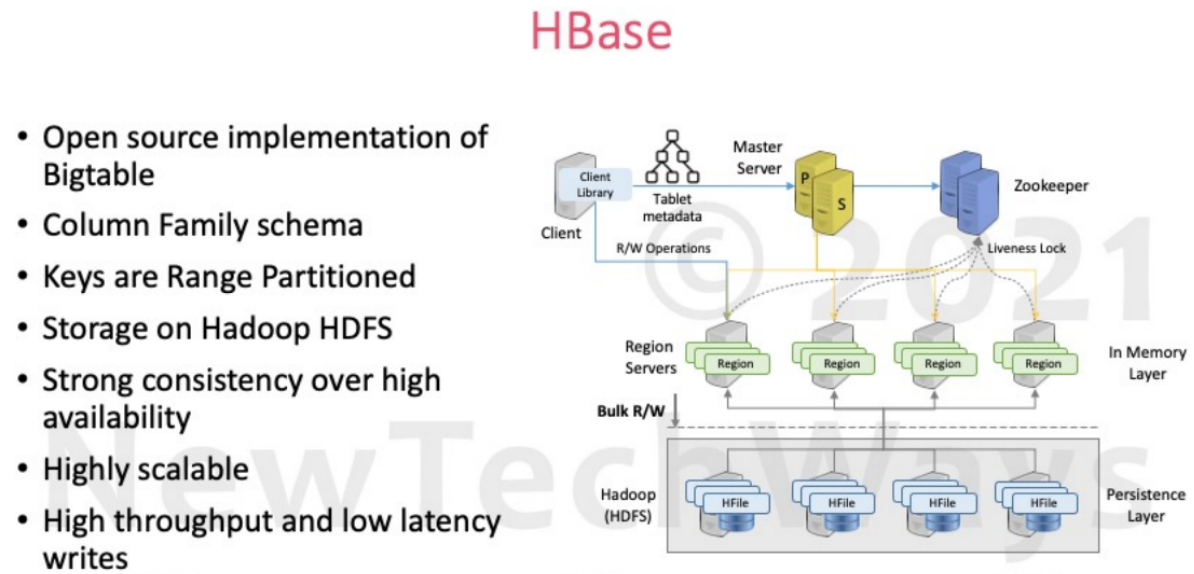
- Apache HBase is open-source implementation of Bigtable.
- Table as TreeMap
- If a new value needs to be updated, then the old value is also kept and a new row is added for the new value.
- Rows are in a sorted order.

Bigtable Architecture:



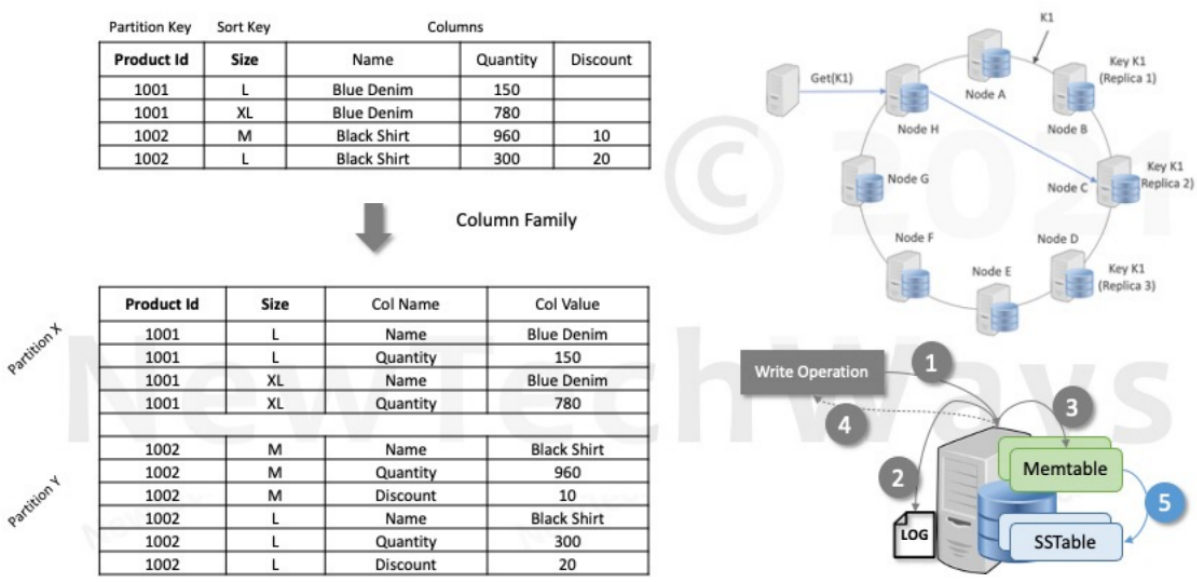
- Data is stored in sorted order that helps in range queries.
- Bigtable uses GFS (Google File System) underneath as the persistent storage.
- Hadoop is the open-source implementation if GFS.
- Tablet Servers stores the String Sorted table (SSTable) or GFS data in memory for faster access (as these files are very huge and stored in GFS disk and reading data will take time). It acts as a cache.
- **Read** – During a read operation data is served from the Tablet Servers to the client.
- If the data is not present, then it is first loaded into the Tablet servers from the GFS and then sent to client.
- **Write** – During a write operation first data is written to a log (WAL – Write Ahead Log) then to the Tablet server and then to the SSTable (Write through Cache).
- Client never connects to master server or Lock Server for any read write operations.
- Client connects to the master server and lock server only once to get the Tablet Metadata which is a BTree kind of index which it will store in memory and that guide the client which Tablet to connect for Read and Write data.
- Master role is to check if the nodes are alive and to assign tablets to nodes servers and it re-assigns the tablets if the node is not alive.
- Every Tablet server is supposed to hold a lock in the lock server and if it fails to do so then it is considered as dead. Master also consider that server as dead and re-distributes the data to other nodes.

HBase:



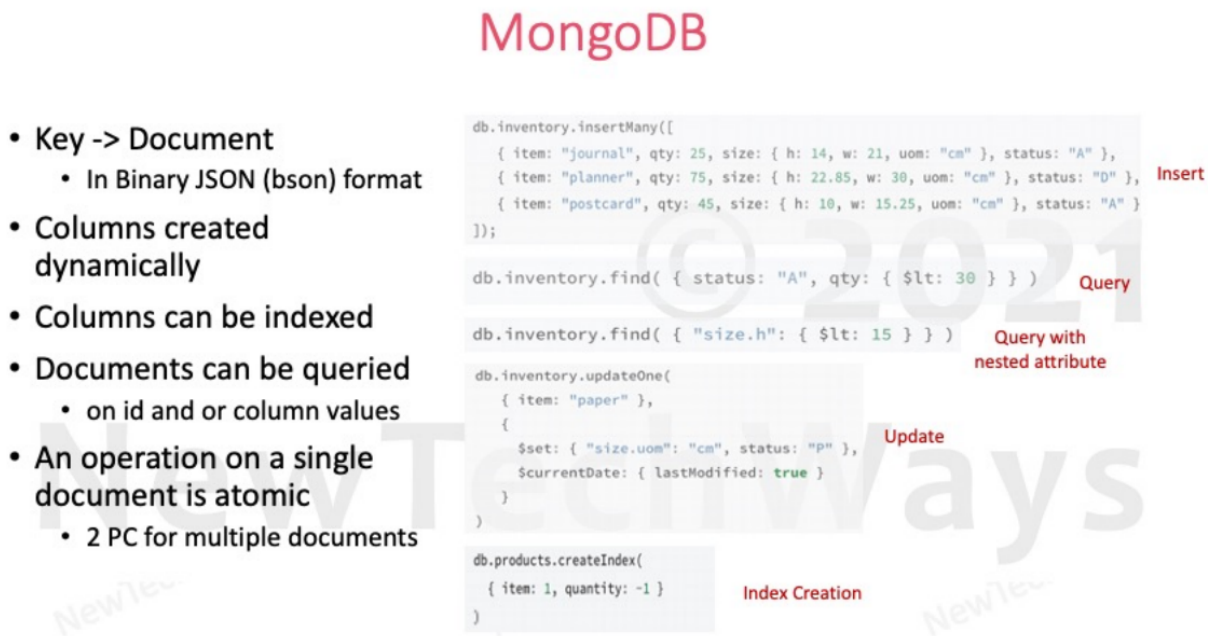
- Tablet Servers (BigTable) →Region Servers (HBase)
- Underling storage GFS (BigTable) → Hadoop HDFS (HBase)
- Lock Server Chubby (BigTable) → Zookeeper (HBase)

Cassandra:

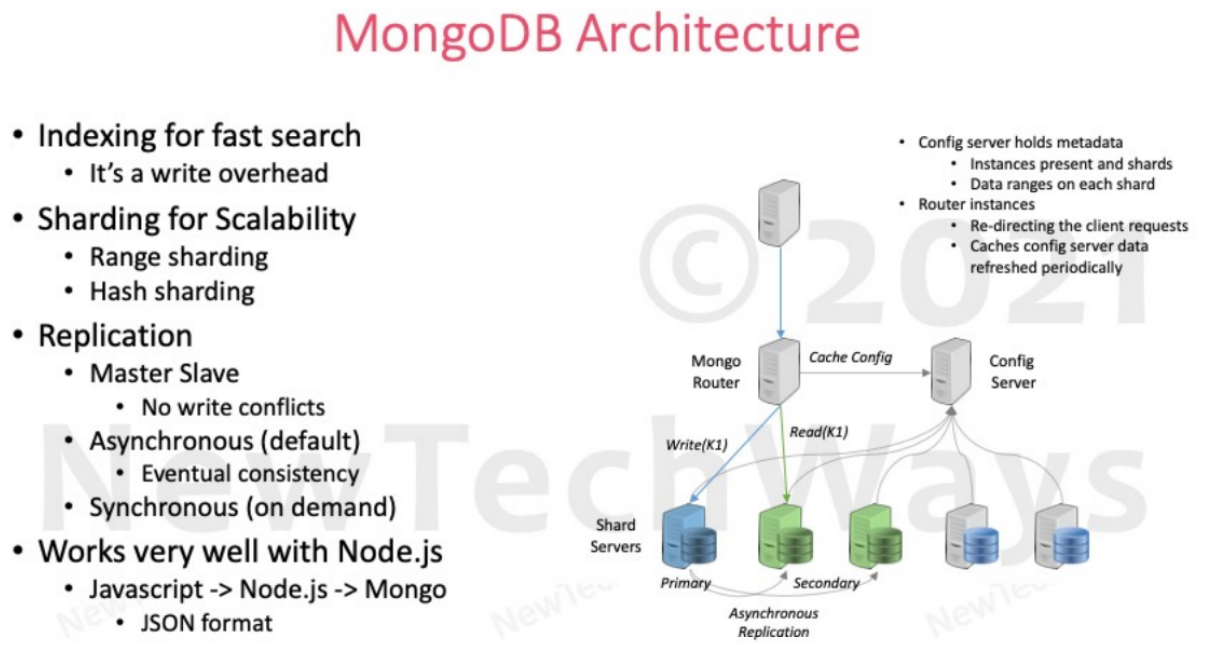


- In Cassandra the sorting is only in a partition.
- Data is sent to a partition based on the hash of the partition key.
- Data model is mix of DynamoDB and BigTable.
- Peer to Peer like DynamoDB.
- Write operation is similar to BigTable → 1. log file (WAL), 2. Memtable ,3. SSTable
- Tablet Server (Big Table)→ Memtable (Cassandra)
- Read operation is similar to BigTable.

MongoDB:



MongoDB Architecture:



- In terms of latency, it is like RDBMS. The more the number if indexes the slower the write operations.
- MongoDB differs from RDBMS in terms of horizontal scalability.

Analytics:

Analytics – Solutions:

Data Movement →

- Logstash

- Fluentd

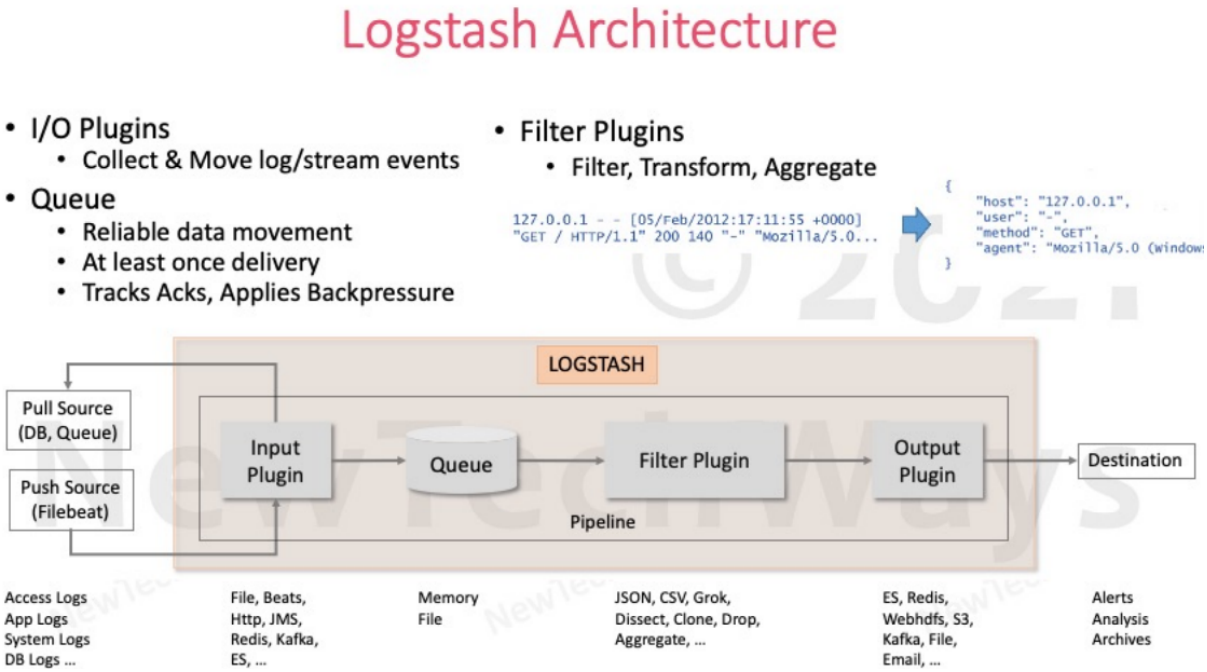
Storage →

- Hadoop HDFS – MapReduce
- Elastic Search – Search

Stream processing →

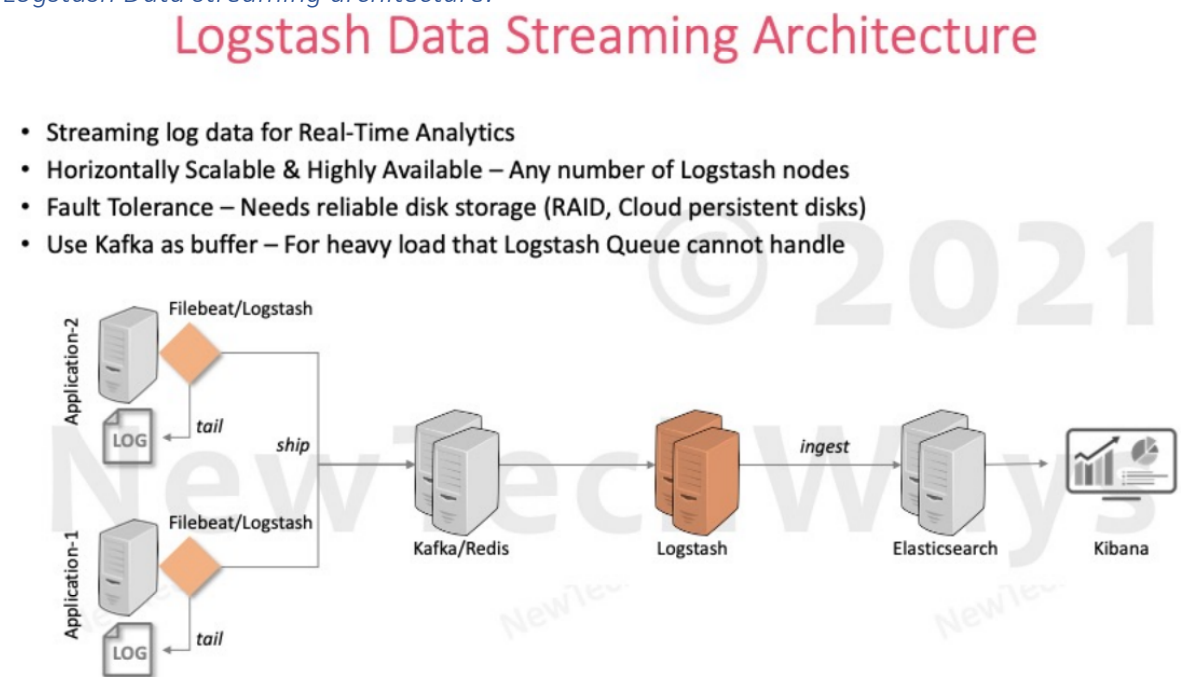
- Kafka – Buffer
- Storm, Flink – Processing

Logstash:



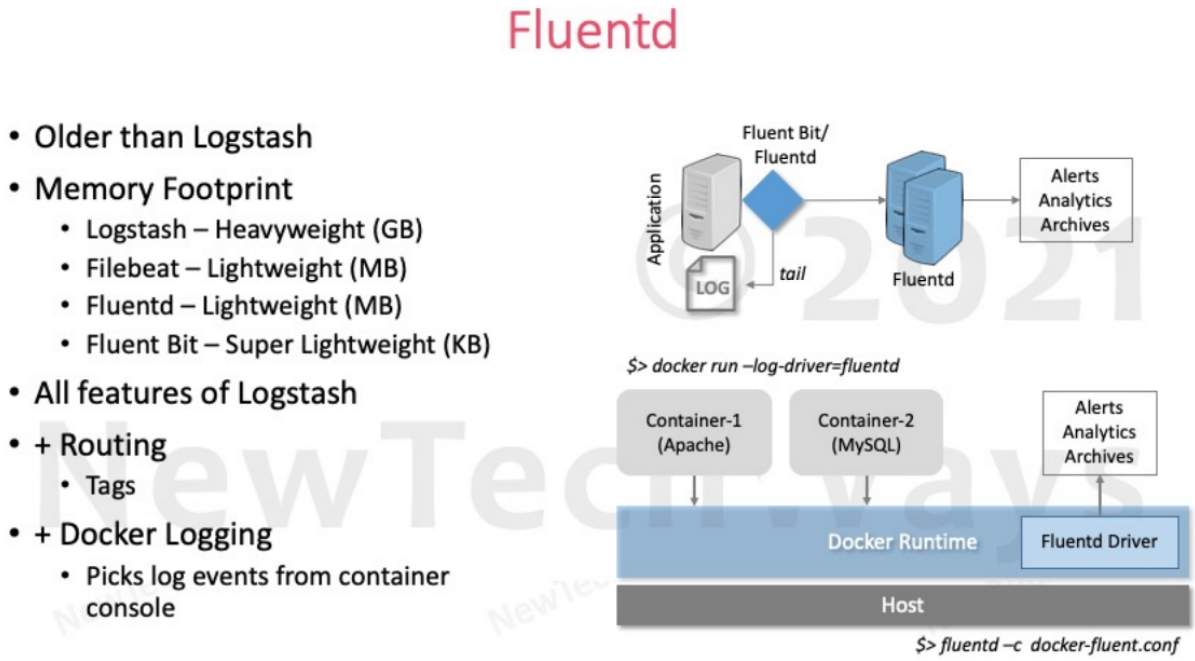
- Move files from services to storage.
- Filtering of data can also be done before storage.

Logstash Data streaming architecture:



- Logstash is a heavy application, so it's not recommended to install on the same machine where the app is installed. Filebeat is another tool developed by Logstash team which is very light weight, and it can collect the data and ship it.
- FileBeat can send data directly to analytic engine but that won't be a scalable solution as when more number of application generate log the ES might not be able to handle.

Fluentd: (d – demon)



Elasticsearch:

Elasticsearch

- Full-Text Search
  - Filter, Group, Aggregate
- Stores JSON Documents
- Document Fetched using id
- Indexes JSON keys and values
- Structure
  - Index -> Database
  - Type -> Table
  - Document -> Row
    - JSON keys are flattened
  - Supports data types
- Users can specify mapping between terms & documents

Documents

ID	Document
Doc-1	Exception in Order module
Doc-2	Your Order is created
Doc-3	Your Order is delivered

Inverted Index

Term	Frequency	Documents
created	1	Doc-2
delivered	1	Doc-3
Exception	1	Doc-1
in	1	Doc-1
is	2	Doc-2, Doc-3
module	1	Doc-1
Order	3	Doc-1, Doc-2, Doc-3
Your	2	Doc-2, Doc-3

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

GET localhost:9200/accounts/person/1

GET localhost:9200/_search?q=john

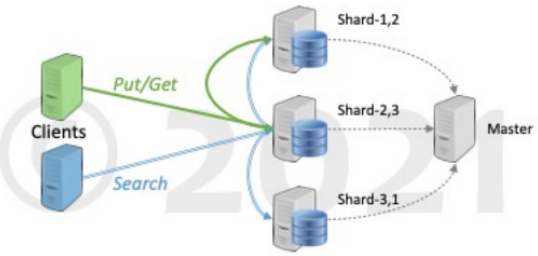
GET localhost:9200/_search?q=job_description:linux

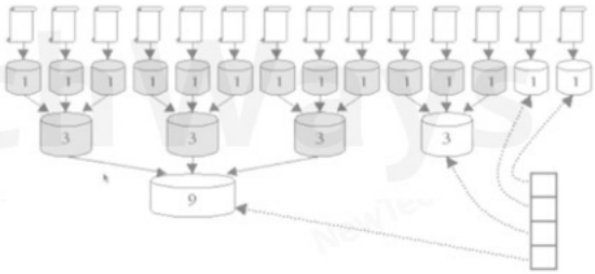
GET localhost:9200/accounts/person/_search?q=job_description:linux

Elasticsearch architecture:

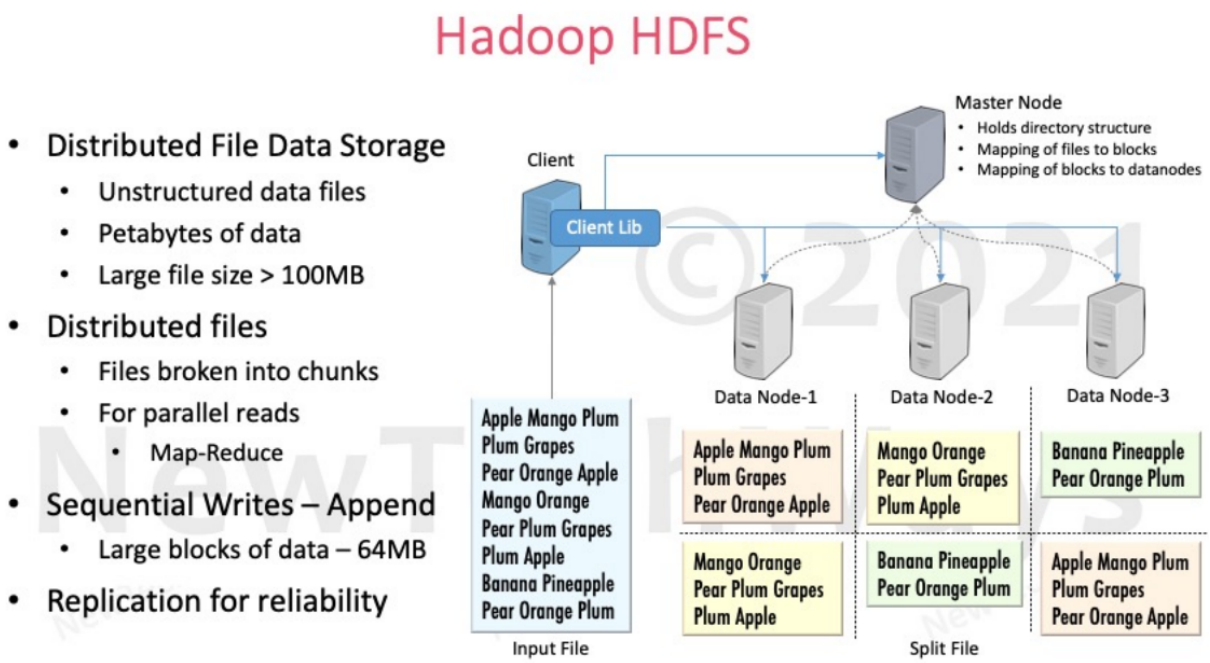
Elasticsearch Architecture

- Document Oriented data-model
- Horizontally Scalable
  - Petabytes of data
  - Data is sharded with key as Document Id
  - Put/Get request goes to specific shard
  - Search queries go to all shards
- Highly Available
  - Shards are replicated
- Index structure is based on merge sort
- Index not updated with every update or insert
- Index maintained in memory
  - Occasionally flushed to disk



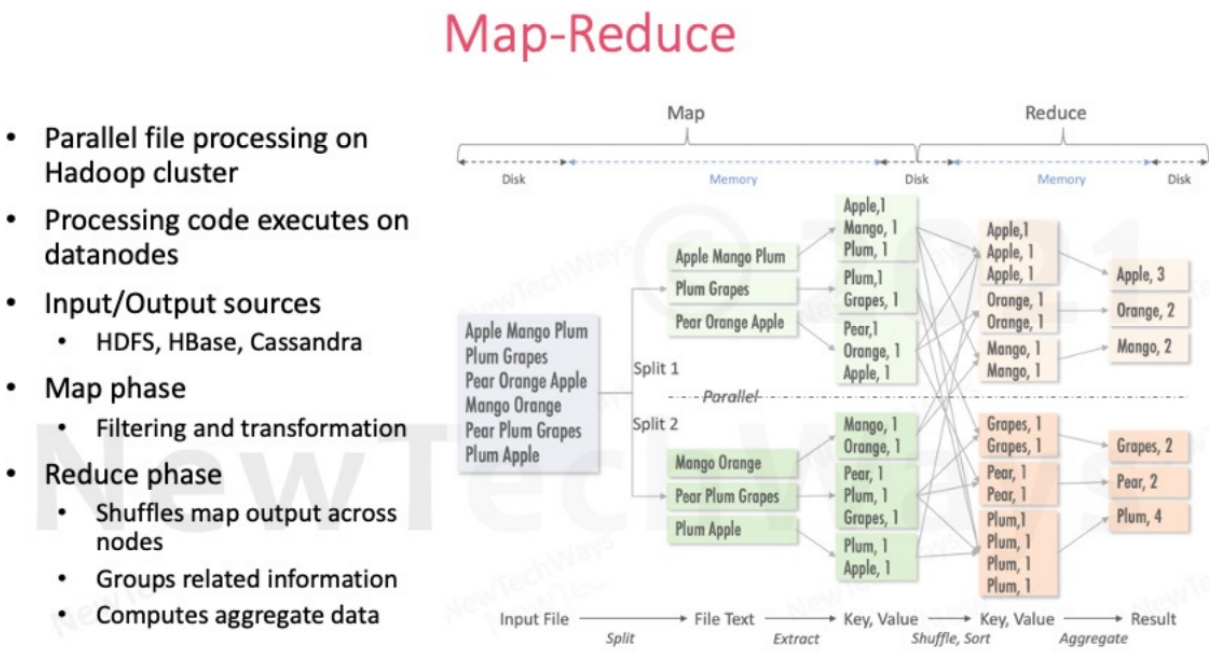


Hadoop HDFS:



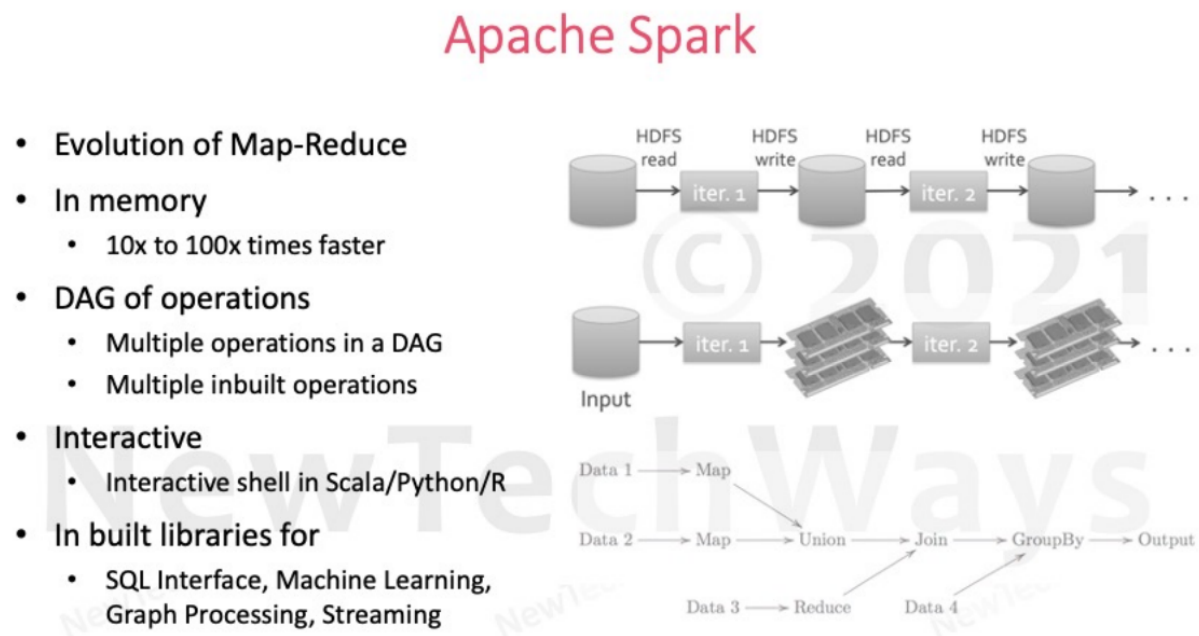
- Client needs to interact with the Master node to find the information of the node and then it can directly talk to the nodes.
- This way master node does not become a bottle neck.

MapReduce:



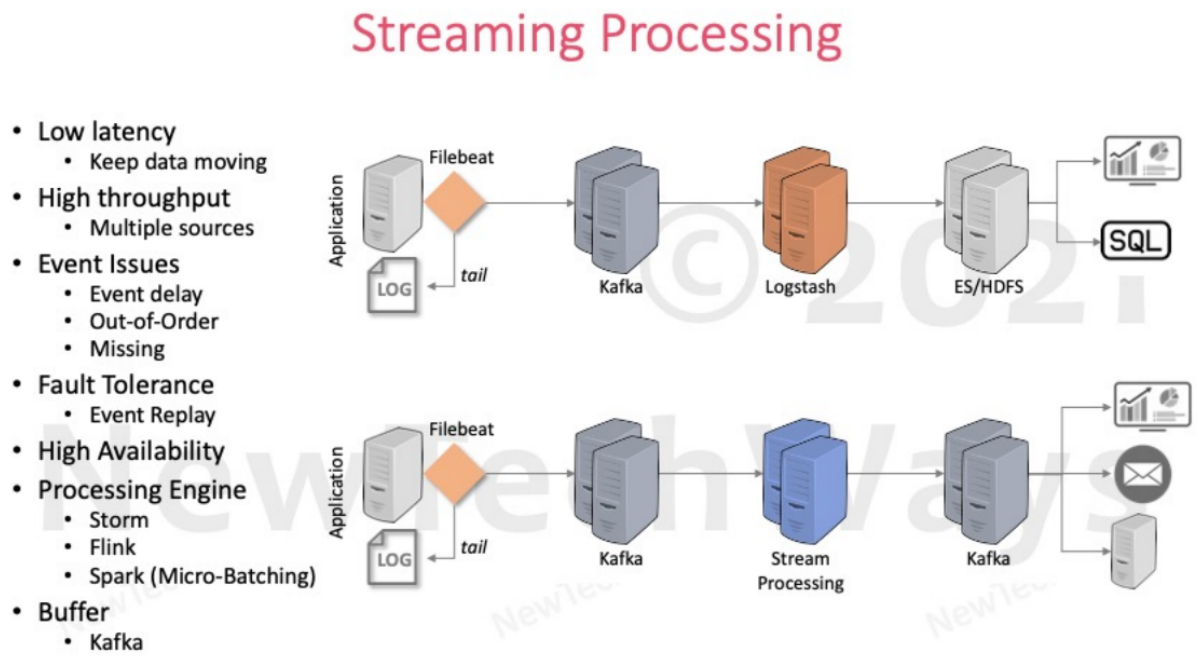
- Input source to MapReduce can be from HDFS or HBase or Cassandra but the intermediate step for storing into disk should be Hadoop Cluster.
- Used for bulk processing which can take hours or days or significant amount of time.

Apache Spark:



- In case of Hadoop MapReduce data is read from disk to memory and then after map function it is again written onto disk and then again when reduce function start data is read into memory and then again written into disk.
- In case of Apache Spark the data is kept in memory during the intermediate steps rather than writing into disk which speeds up the process by 10 – 100x.

Stream Processing:



When we want analysis of data in real time like fraud detection, we do stream processing.

Transactions in Microservices

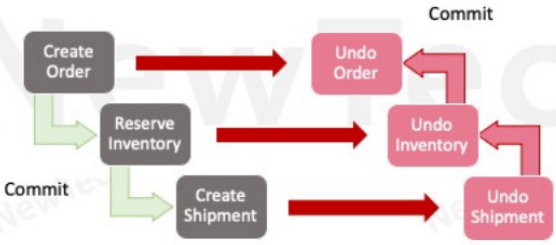
Micro-Services Transactions

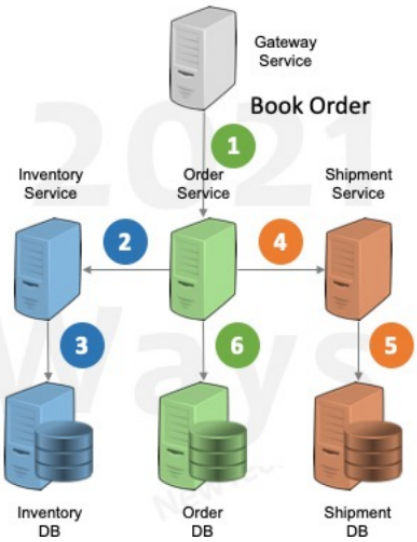
- Transaction involves multiple machines
  - Distributed services with their own DB
  - Local transaction not possible
- Options
  - Distributed ACID Transactions
    - 2PC/3PC
    - Completely ACID
  - Compensating Transactions
    - SAGA pattern
    - Eventually consistent model
      - Relaxes consistency and isolation

SAGA Pattern:

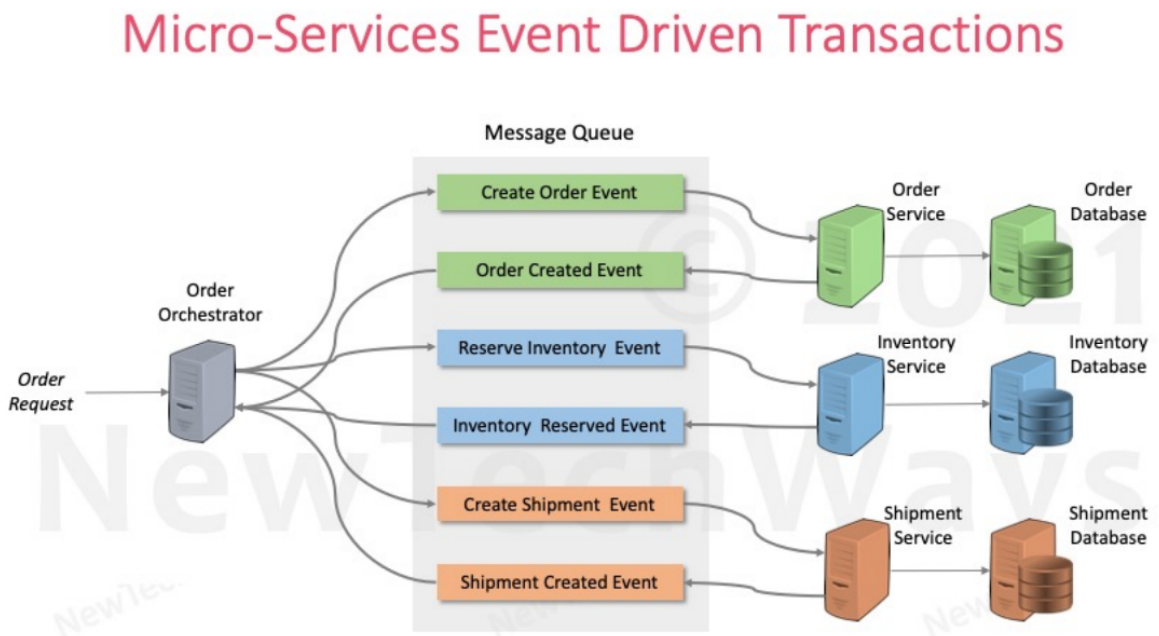
Compensating Transactions – SAGA Pattern

- 'Logical Undo' of a partially committed transactions
  - Flow of reversal may not be exactly opposite, and some steps can be executed in parallel
  - Compensation itself can fail. Should be able to restart itself and retry
- Asynchronous processing for reliability

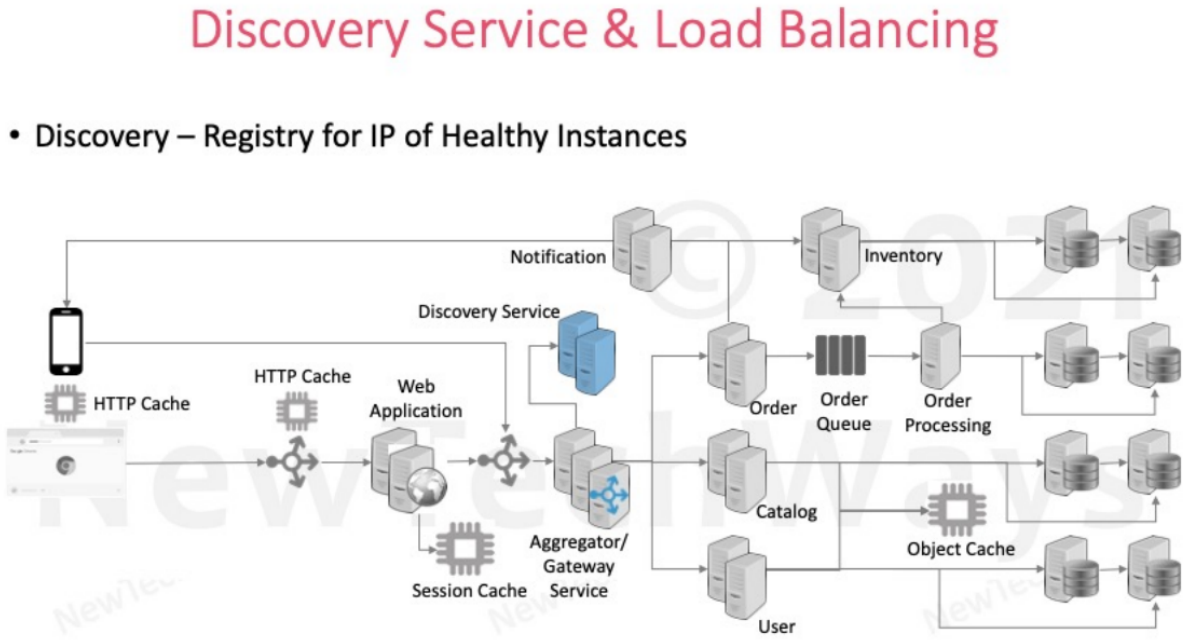




Microservices Event Driven Transactions:



Discovery Service & Load Balancing:



- Gateway Service has an embedded router and checks the discovery service for the list of the IPs for a service.
- Based on the algorithm in the router the requests are routed.