System Design Development Tools

Ramakant Bharti

Revision History

Sno	Who	When	What
1	Ramakant	5 Dec 2019	Created
2	Ramakant	6 Jun 2020	Updated - Redis, Overview

TABLE OF CONTENTS

OVERVIEW

Used Ubuntu 18.04 to install and play around all the relevant system design tools.
Most of the diagrams/materials are taken from the respective tools authoritative sites.

DESIGN TEMPLATE

This is a generic template taken from the book **Designing Data Intensive Applications** (awesome book btw!). I have just added the binary objects and object store to it.

The basic idea is that a complex system can be built by using the commonly available tools and stitching them together efficiently with application code.

For a good system (broadly referred to be available, scalable and maintainable) we need the following:

1. Good knowledge of the common tools and their trade-offs
2. Robust application code to glue these together

There are six categories of these system design tools (numbered in red in the diagram below) and we will try to cover them briefly in the pages to follow.

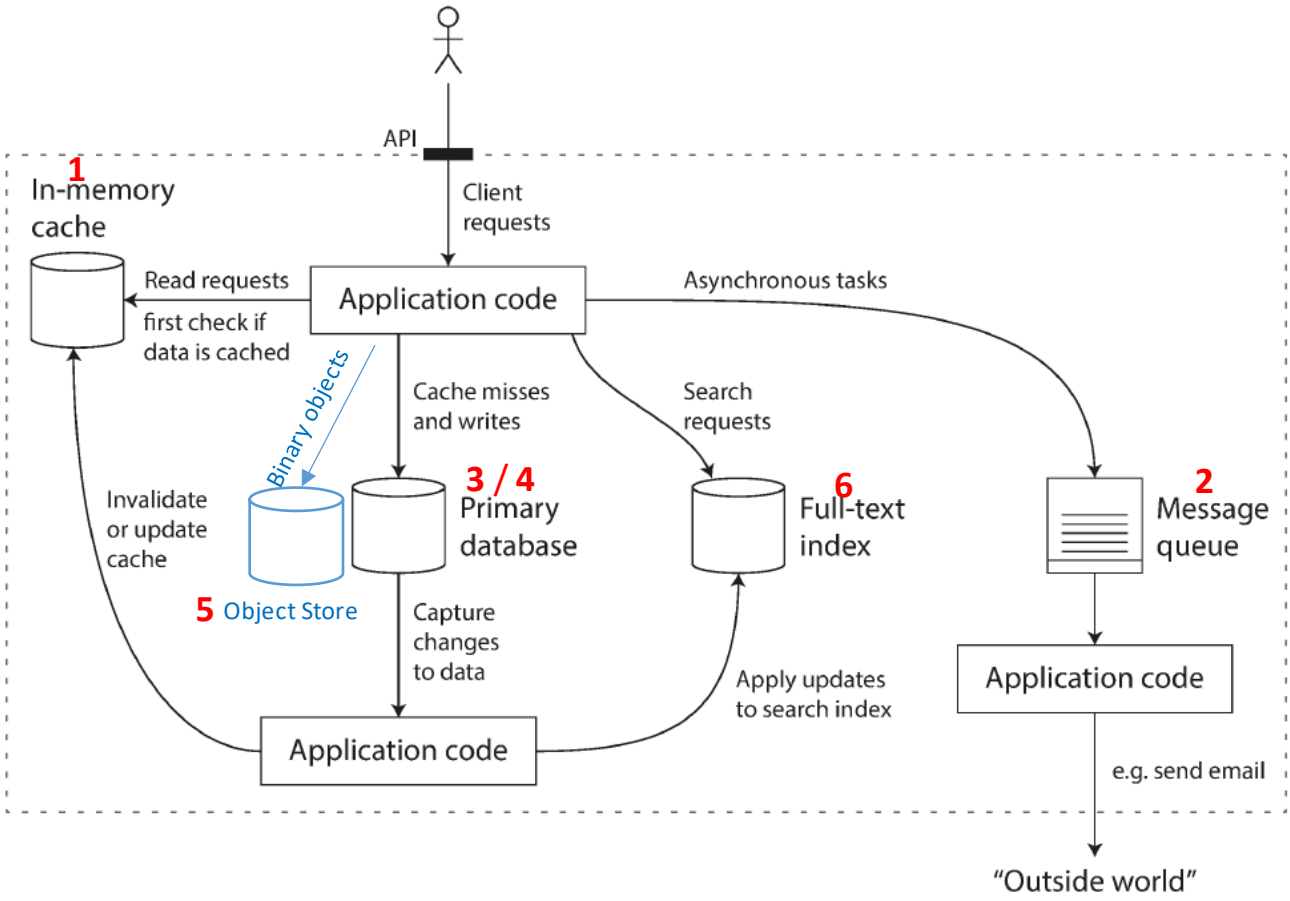


Figure 1-1. One possible architecture for a data system that combines several components.

1. **In memory cache**
   - Memcached (in memory distributed key-value cache)
   - Redis (+document store+durability+partitioning+Replication...)
2. **Message Queues**
   - RabbitMQ (Written in ErLang, AMQP, Flexible routing)

- Kafka (Written in Scala, uses Zookeeper, Multi-consumers)

3. **SQL DB(RDBMS)**
   - MySQL(RDBMS)
   - PostgreSQL (ORDBMS, better SQL compliance)
4. **NoSQL DB**
   - Key-Value Store - Memcached, Redis
   - Document Store - Mongo DB (JSON based CRUD)
   - Column Oriented Store - Apache Cassandra (Distributed DB, uses SQL like syntax)
   - Graph Store - Neo4j (Written in Java, Cypher Query Language, Web interface)
5. **Special Purpose Store**
   - Time Series Store - Influx DB (Written in Go, inbuilt REST API support)
   - Object Store - Store the object in a file system and index it on an RDBMS
6. **Full Text Search**
   - ElasticSearch (Lucene based text search engine written in Java, inbuilt REST API support)

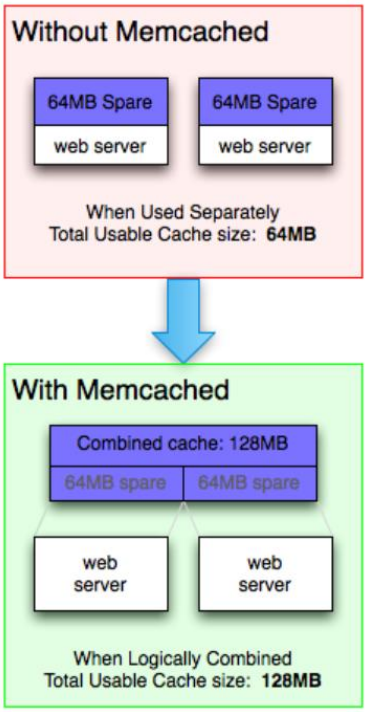
CHAPTER 1: IN MEMORY CACHE

Application caching

In-memory caches such as **Memcached** and **Redis** are key-value stores between your application and your data storage. Since the data is held in RAM, it is much faster than typical databases where data is stored on disk. RAM is more limited than disk, so cache invalidation algorithms such as least recently used (LRU) can help invalidate 'cold' entries and keep 'hot' data in RAM.

1. MEMCACHED

1. OVERVIEW



Memcached is an open source, high-performance, **distributed(?)** memory cache (LRU) intended to speed up dynamic web applications by reducing the database load. It is a key-value store of strings, objects, etc., stored in the memory.

Memcached allows you to take memory from parts of your system where you have more than you need and make it accessible to areas where you have less than you need.

2. INSTALLATION

Update Your System

```
sudo apt-get update -y
sudo apt-get upgrade -y
```

Install Memcached

```
sudo apt-get install memcached libmemcached-tools -y
```

Start the service

```
sudo systemctl start memcached
sudo systemctl enable memcached
sudo systemctl status memcached
```

Configure Memcached

```
sudo nano /etc/memcached.conf
```

Change the following lines as per your requirements:

```
# Default connection port is 11211
-p 11211

# Specify which IP address to listen on
-l 192.168.0.101

#Define the maximum number of Memory can be used by Memcached deamon
-m 256
```

Restart the service

Save and close the file then restart the service for the changes to apply.

```
sudo systemctl restart memcached
```

3. HELLO WORLD!

Using python to interact with memcached. Need to install the plugin.

Install pymemcache

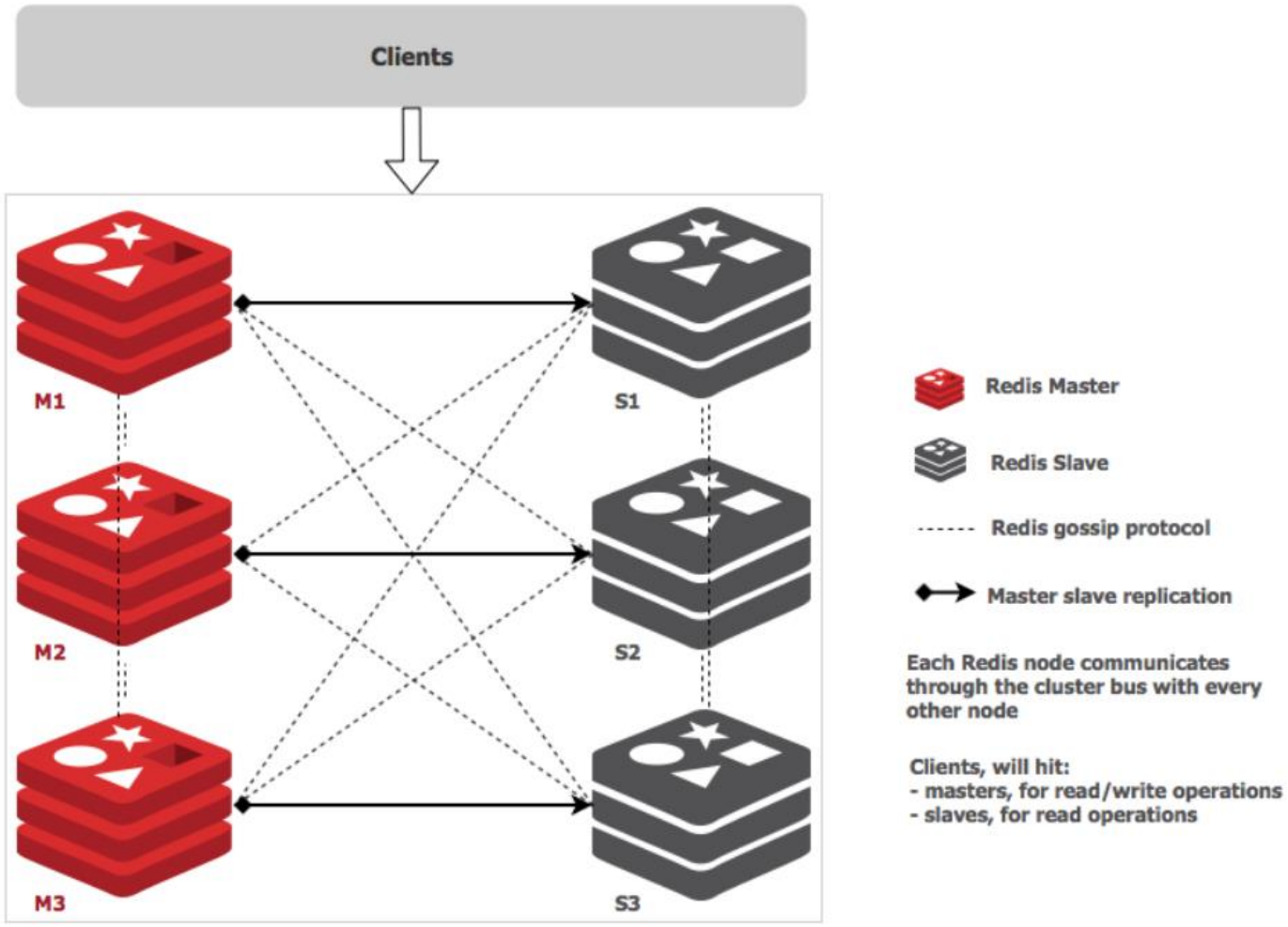
```
pip install pymemcache
```

Run in python

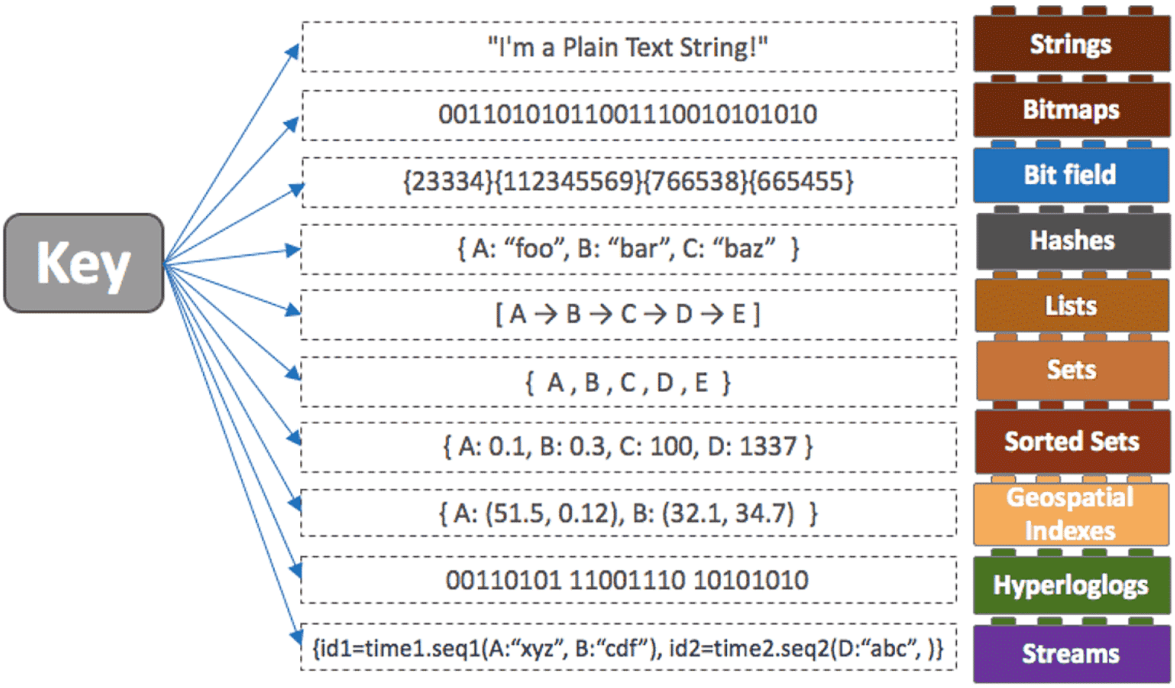
```
>>> from pymemcache.client import base
>>> client = base.Client(('localhost', 11211))
>>> client.set('name', 'Ramakant')
>>> client.get('name') #try this on different consoles
```

2. REDIS

1. OVERVIEW



1. **Intro -** Redis is an open source, advanced key-value store and an apt solution for building high-performance, scalable web applications.
2. **Replication** - Redis supports clustering, multiple Redis instances(nodes) can be part of the same cluster, hence providing horizontal scaling and reliability (when used with replication) – **minimum one master-slave (for reliability)**
3. **Partitioning** - Incoming data needs to be partitioned to figure out in which instance they will land:
    - Range partitioning – needs a mapping table
    - Hash partitioning – hash(data)%n, n= no of instances. Consistent hashing can be implemented in application code if needed.
4. **Persistence** - snapshot & commit log.
5. **Data Types -**
    - Key-value pairs only. key={String}, value = {string, list, set, sorted set etc.}, there's no namespace



- Some main commands, many more exists as per the ADT

	Create	Read	Delete
List	**LPUSH**	LPOP	LREM
Set	**SADD**	SPOP SMEMBERS	SREM
Sorted-Set	**ZADD**	ZPOPMAX ZPOPMIN	ZREM

2. INSTALLATION

Update Your System

```
sudo apt-get update -y
```

Install Redis Server

```
sudo apt-get install redis-server
```

Start the service

```
sudo systemctl start redis-server
sudo systemctl enable redis-server
```

```
sudo systemctl status redis-server
```

3. HELLO WORLD!

Using redis-cli to interact with redis-server

```
127.0.0.1:6379> set msg "Hello World!"
OK
```

```
127.0.0.1:6379> get msg #Try this on a different consoles
"Hello World!"
```

3. CONCLUSION

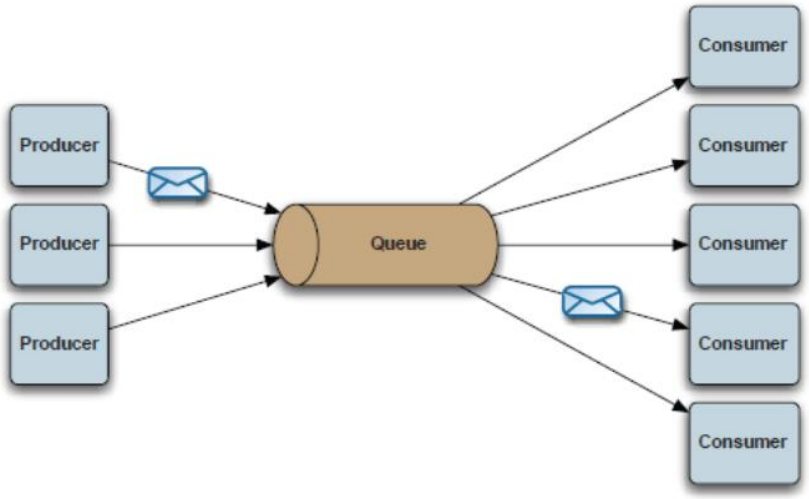
Tool Trade-off

Sno.	Memcached	Redis
1	Simple architecture - very fast an effective	-
2	Single instance(?), clustering has to be done from client code	Inbuilt clustering capabilities, supports partitioning & replication
3	Scalability and reliability suffers due to it being single instance	Highly scalable and reliable
4	No persistence option	Supports persistence – snapshot, persistence logs
5	Values are simple strings	Supports complex datatypes as values – Hash, List, Set & Sorted Set
6	Utilizes multiple cores	Only single core

Doubts

1. How is memcached distributed?

CHAPTER 2: MESSAGE QUEUE

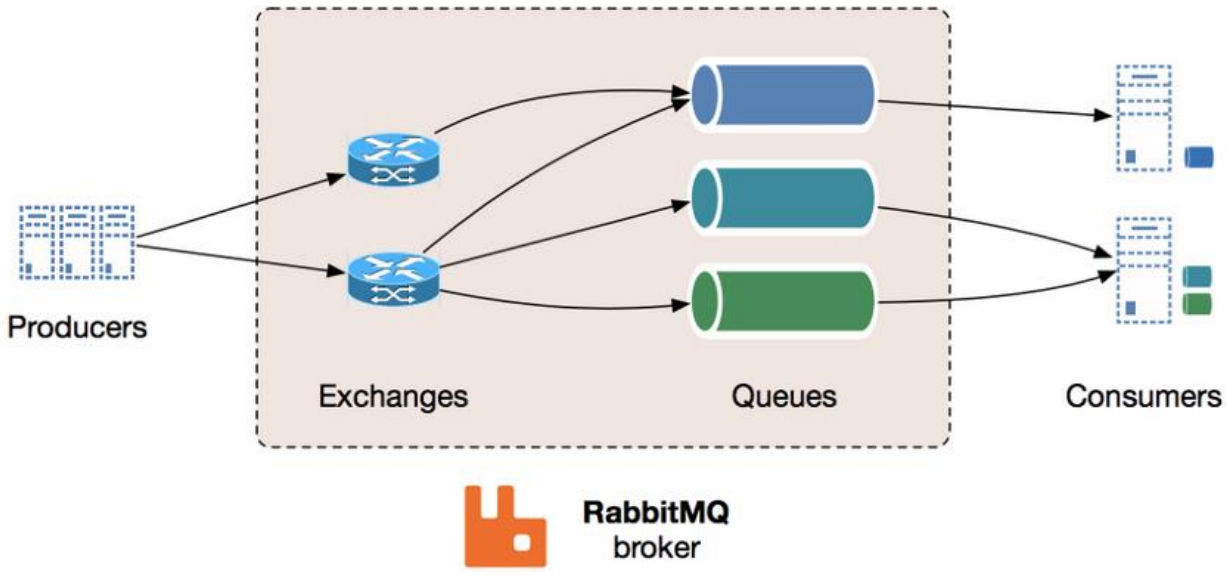


Message queues receive, hold, and deliver messages. If an operation is too slow to perform inline, you can use a message queue with the following workflow:

- An application publishes a job to the queue, then notifies the user of job status
- A worker picks up the job from the queue, processes it, then signals the job is complete

The user is not blocked and the job is processed in the background asynchronously. During this time, the client might optionally do a small amount of processing to make it seem like the task has completed. For example, if posting a tweet, the tweet could be instantly posted to your timeline, but it could take some time before your tweet is actually delivered to all of your followers.

1. RABBITMQ



1. OVERVIEW

- Designed as a **general purpose message broker**, employing producer-consumer model.
- Producers send messages to the exchanges and Consumers receive messages from Queues (Producers are decoupled from message routing decisions)
- Can be setup as multi-node cluster

- Only one copy of the messages even in multi-node cluster, unless we use the Quorum queue feature
- Implements the Advanced Message Queuing Protocol (AMQP)
- It is mature, performs well and has good support from libraries and plugins

2. INSTALLATION

Pre-req – Install Erlang

RabbitMQ is written in Erlang, install it

Import Erlang GPG Key

```
wget -O- https://packages.erlang-solutions.com/ubuntu/erlang_solutions.asc  | sudo apt-key add -
```

Add Erlang Repo to Ubuntu

```
echo "deb https://packages.erlang-solutions.com/ubuntu  bionic contrib" | sudo tee /etc/apt/sources.list.d/rabbitmq.list
```

Install Erlang

```
sudo apt update
sudo apt -y install erlang
```

Install RabbitMQ

Import RabbitMQ

```
wget -O- https://dl.bintray.com/rabbitmq/Keys/rabbitmq-release-signing-key.asc  | sudo apt-key add -
wget -O- https://www.rabbitmq.com/rabbitmq-release-signing-key.asc  | sudo apt-key add -
```

Add RabbitMQ repo to Ubuntu

```
echo "deb https://dl.bintray.com/rabbitmq/debian  $(lsb_release -sc) main" | sudo tee /etc/apt/sources.list.d/rabbitmq.list
```

Install RabbitMQ server

```
sudo apt update
sudo apt -y install rabbitmq-server
```

Start the service

```
sudo systemctl start rabbitmq-server
sudo systemctl enable rabbitmq-server
sudo systemctl status rabbitmq-server
```

Enable dashboard

```
sudo rabbitmq-plugins enable rabbitmq_management
```

```
sudo ufw allow proto tcp from any to any port 5672,15672
```

http://localhost:15672 - guest/guest

3. HELLO WORLD!

Install Python Pika client

```
sudo python -m pip install pika -upgrade
```

Console1 (Producer)

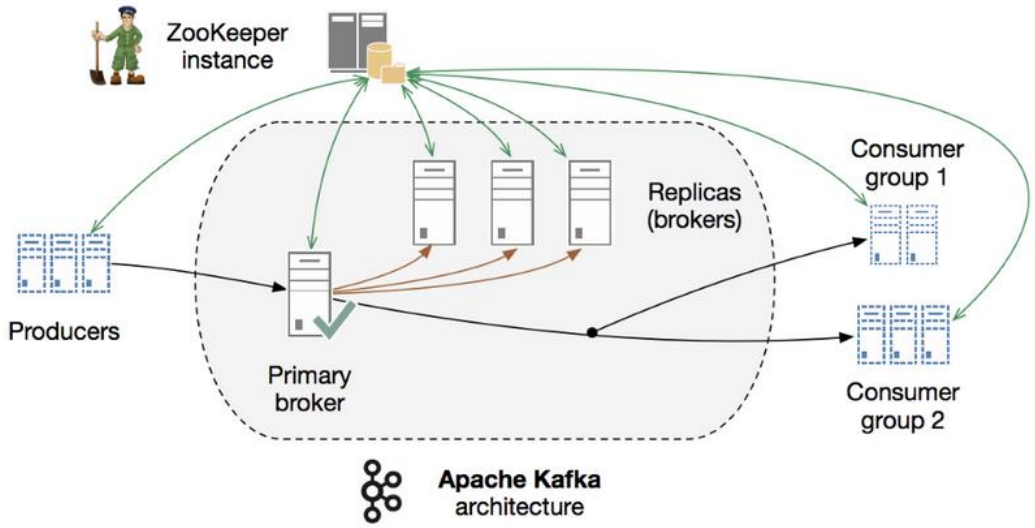
```
>>> import pika
>>> connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
>>> channel = connection.channel()
>>> channel.queue_declare(queue='hello')
<METHOD(['channel_number=1', 'frame_type=1', "method=<Queue.DeclareOk(['consumer_count=0', 'message_count=0', 'queue=hello'])>"])>
>>> channel.basic_publish(exchange='',
...                       routing_key='hello',
...                       body='Hello World!')
>>> print(" [x] Sent 'Hello World!'")
[x] Sent 'Hello World!'
>>> connection.close()
```

Console2 (Consumer)

```
>>> import pika
>>> connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
>>> channel = connection.channel()
>>> channel.queue_declare(queue='hello')
<METHOD(['channel_number=1', 'frame_type=1', "method=<Queue.DeclareOk(['consumer_count=0', 'message_count=1', 'queue=hello'])>"])> <-- Till here same as above!
>>> def callback(ch, method, properties, body):
...     print(" [x] Received %r" % body)
>>> channel.basic_consume(queue='hello',
...                       auto_ack=True,
...                       on_message_callback=callback)
'ctag1.4b5cae2475bf4792b531d69a267e0705'
>>> print(' [*] Waiting for messages. To exit press CTRL+C')
[*] Waiting for messages. To exit press CTRL+C
>>> channel.start_consuming()
[x] Received 'Hello World!'
```

2. KAFKA

OVERVIEW



- Originally created as a **distributed commit log**, hence providing features like message durability and message replay
- Stores messages in the HDD/SSD and uses OS features (Page Cache, Zero Copy Send Files) to achieve redis level of performance in best case scenarios
- Designed for high volume publish-subscribe messages and **streams**
- Is durable, fast, and scalable
- Works well even with high number of nodes in the cluster

INSTALLATION

Pre-req – Install Java

```
sudo apt update
sudo apt install default-jdk
```

Download and install Apache Kafka

```
wget http://www-us.apache.org/dist/kafka/2.2.1/kafka_2.12-2.2.1.tgz
tar xzf kafka_2.12-2.2.1.tgz
mv kafka_2.12-2.2.1 /usr/local/kafka
```

Start Kafka server

```
cd /usr/local/kafka
bin/zookeeper-server-start.sh config/zookeeper.properties
bin/kafka-server-start.sh config/server.properties
```

HELLO WORLD!

Create a topic in Kafka

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

```
Created topic testTopic.
```

Kafka Producer

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

> Start typing here

Kafka Consumer (in another console, can be more than one)

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic testTopic --from-beginning
```

> Appears here

3. CONCLUSION

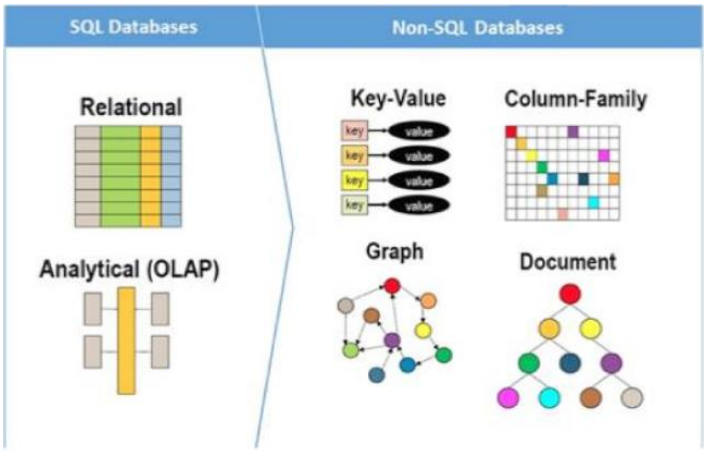
Tool Trade-off

	Kafka	RabbitMQ
Distributed consumers	✅	✅
Distributed producers	✅	✅
Distributed queues	✅	✅
Replicated queues	✅	✅
Distributed leader to manage cluster state	✅ zookeeper	❌
performance	✅ less hardware	✅ many nodes with high throughput
Commit log/ Time travel	✅	❌ msg removed once consumed
Multi consumers for one msg	✅ by design	❌ only by routing this msg to different queues
Msg ordering in parallel processing	✅ offer ordering in the partition level	❓ can be done by smart routing/binding
Log compaction	✅	❌
Msg encryption	✅	❌
Msg persistence	✅ retention period, can be unlimited time	❓ from the docs **the persistence guarantees aren't strong**
Consumer msg acknowledgement	✅	✅
Producer msg acknowledgement	✅	❌
Pull/push model	✅ consumers request batches of messages from a given offset	❌ RabbitMQ push notification to consumers
Stream processing	✅	❌
Flexible routing to topic/queue	❌ by msg key hash	✅ 5 different ways of routing including wildcard and pattern
Msg priority	❌	✅
Msg protocols	Binary	AMQP, STOMP (Text based), MQTT (binary) and HTTP.
Transaction support	✅	✅
Monitoring (UI)	✅	✅
Open source	✅	✅
Written in	Scala	erlang

Doubts

CHAPTER 3: SQL DB (RDBMS)

- ACID Properties – Atomicity, Consistency, Isolation, Durability
- Important concepts – Normalization, Joins, Indexing
- SQL vs NoSQL



SQL	NoSQL
RDBMS with **ACID** properties	Non-relational or distributed DB with **BASE** properties
Predefined schema - **schema on write**	Schema-less – **schema on read**
Good for **structured data**, works well with complex queries and joins	Good for document, hierarchical data, graph storage
Use when **consistency** is important	Use when **availability** is important
Limited scalability – only **vertically** scalable	Very scalable - **horizontally** scalable

- 2 popular SQL DBs are **MySQL**(RDBMS) and **PostgreSQL**(**ORDBMS**)

1. MYSQL

1. OVERVIEW

- World's most popular open source DB
- Proven performance, reliability, and ease-of-use, leading database choice for web-based applications
- Supports multiple storage engines – **MyISAM** (simple & fast), **InnoDB** (most common) and more

2. INSTALLATION

Install mysql

```
sudo apt-get update
sudo apt-get install mysql-server
```

Allow remote access

```
sudo ufw enable
sudo ufw allow mysql
```

Start the service

```
sudo systemctl start mysql
```

Install workbench

```
sudo apt install mysql-workbench
```

3. HELLO WORLD!

Command line

```
sudo su <- Default installation of mysql logs in root user through Ubuntu's root user auth
mysql -u root -p
mysql> UPDATE mysql.user SET Password = PASSWORD('root') WHERE User = 'root';
mysql> UPDATE mysql.user SET plugin = 'mysql_native_password' where User = 'root'; <- now root/root can be used to login from command line or workbench
```

Now, login again with mysql -u root -p with root as password and have fun!!

Workbench

```
$ mysql-workbench
```

Create a new connection with root/root and login into the workbench and you are all set!

2. POSTGRESQL (COMING LATER…)

3. CONCLUSION

Tool Trade-off

Parameter	MYSQL	PostgreSQL
Performance	Mostly used for web-based projects that need a database for straightforward data transactions	Used in large systems where read and write speeds are important
Best suited	OLAP & OLTP systems (read heavy)	Complex queries
JSON	JSON data type support	Also allows indexing JSON data
Materialized views	Supports materialized views and temporary tables	Supports temporary tables only
Object	Fairly good object statistics	Very good object statistics
Joins	Limit join capabilities	Good join capabilities
Companies	Airbnb, Uber, Twitter	Netflix, Instagram, Groupon

Doubts

1. Does postgreSQL support storage engines like MySQL?

CHAPTER 4: NOSQL DB

NoSQL is a collection of data items represented in a **key-value store**, **document store**, **wide column store**, or a **graph database**. Data is denormalized, and joins are generally done in the application code. Most NoSQL stores lack true ACID transactions and favour **eventual consistency**.

BASE is often used to describe the properties of NoSQL databases. In comparison with the CAP Theorem, BASE chooses availability over consistency.

- **Basically Available** - the system guarantees availability.
- **Soft state** - the state of the system may change over time, even without input.
- **Eventual consistency** - the system will become consistent over a period of time, given that the system doesn't receive input during that period

1. KEY-VALUE STORE

Memcached or Redis can be used as a key-value store, already discussed.

2. DOCUMENT STORE

- Stores documents (**XML**, **JSON**, binary, etc)
- Provides APIs or a query language to query the document
- Based on the underlying implementation, documents are organized by collections, tags, metadata, or directories
- Often used for working with occasionally changing data

1. OVERVIEW

- MongoDB stores data in flexible, **JSON-like documents**, meaning fields can vary from document to document and data structure can be changed over time
- The document model maps to the objects in your application code, making data easy to work with
- Ad hoc queries, indexing, and real time aggregation provide powerful ways to access and analyse your data
- MongoDB is a distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use

2. INSTALLATION

Import MongoDB public GPG key

```
wget -qO - https://www.mongodb.org/static/pgp/server-4.2.asc  | sudo apt-key add -
```

Create a list file for MongoDB

```
echo "deb [ arch=amd64 ] https://repo.mongodb.org/apt/ubuntu  bionic/mongodb-org/4.2 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.2.list
```

Reload local package database

```
sudo apt-get update
```

Install the MongoDB packages

```
sudo apt-get install -y mongodb-org
```

Start MongoDB

```
sudo service mongod start
```

3. HELLO WORLD!

Type **mongo** to launch the console

Inserting a document into a **collection** (student), creates the collection if not present since schema is not needed

```
> db.student.insert({
... regNo: "3014",
... name: "Ramakant",
... course: {
... courseName: "MS",
... duration: "2 Years"
... },
... address: {
... city: "Bangalore",
... state: "KA",
... country: "India"
... }
... })
WriteResult({ "nInserted" : 1 })
```

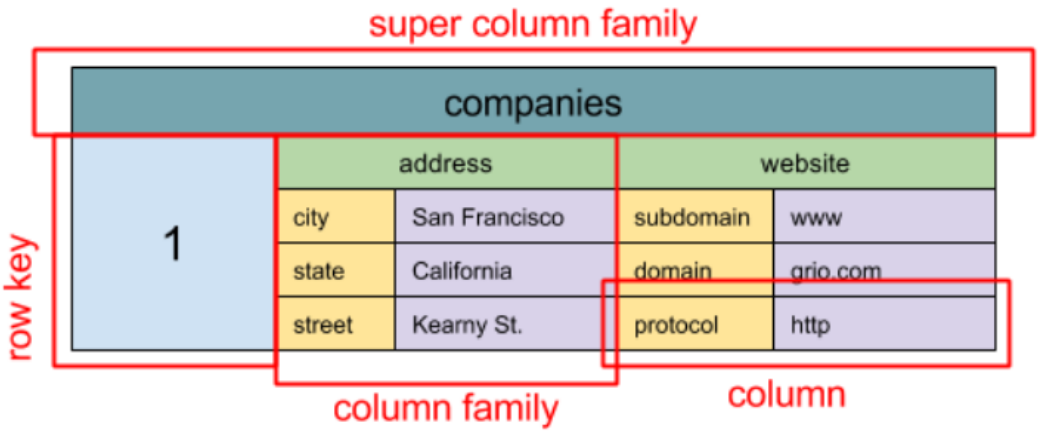
List the documents in a collection

```
> db.student.find()
{ "_id" : ObjectId("5dd50d6e369241e45479f55a"),
 "regNo" : "3014", "name" : "Ramakant", "course
" : { "courseName" : "MS", "duration" : "2 Year
s" }, "address" : { "city" : "Bangalore", "stat
e" : "KA", "country" : "India" } }
```

General syntax like where clause: collection.find(**{"fieldname":"value"}**)

CRUD – collection.insert(), collection.find(), collection.update(), collection.remove()

3. COLUMNN ORIENTED STORE



- Column based storage in:

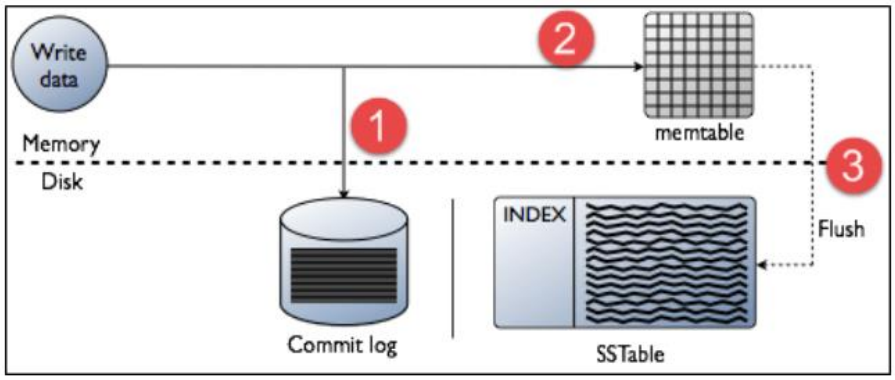
- High level – data is stored in cells grouped in columns, which in turn are grouped in Column Families (Cells > Columns > Column Family > Super Column Family)
- Low level – Unlike RDBMS where rows of data is stored together (in the same block or contiguous blocks), here columns of data is stored together. This enables us to read related columns from a HDD much faster (sequential read)

- Good for data aggregation needs and OLAP queries
- Often used for very large data sets

Google introduced Bigtable as the first wide column store, which influenced the open-source HBase often-used in the Hadoop ecosystem, and Cassandra from Facebook. Stores such as BigTable, HBase, and Cassandra maintain keys in lexicographic order, allowing efficient retrieval of selective key ranges.

APACHE CASSANDRA

1. OVERVIEW



- Cassandra implements Amazon's Dynamo-style replication model with no single point of failure, but adds a more powerful “column family” data model like Google's Bigtable
- Apart from the regular benefits of the class of store(NoSQL, Column Oriented), following are it's notable features:
  - **Transaction support** – Cassandra supports properties like Atomicity, Consistency, Isolation, and Durability (ACID)
  - **Fast writes** – It performs blazingly fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency

Cassandra Write Operation:

1. Write request first written to commit log
2. Then it's written to the Memtable
3. When Memtable is full, the data is flushed to the SSTable

2. INSTALLATION

Add the Cassandra Repository File

```
echo "deb http://www.apache.org/dist/cassandra/debian 39x main" | sudo tee -a /etc/apt/sources.list.d/cassandra.sources.list
```

Add the GPG Key

```
curl https://www.apache.org/dist/cassandra/KEYS  | sudo apt-key add -
```

Install Cassandra on Ubuntu

```
sudo apt update
sudo apt install cassandra
```

Start Cassandra

```
sudo systemctl start cassandra
sudo systemctl enable cassandra
sudo systemctl status cassandra
```

3. HELLO WORLD!

Creating a Keyspace (analogous to Schema) using Cqlsh (syntax similar to SQL)

```
CREATE KEYSPACE test
WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 1};
```

Create

```
Use test;
CREATE TABLE emp(
   emp_id int PRIMARY KEY,
   emp_name text,
   emp_city text,
   emp_sal varint,
   emp_phone varint
   );
```

Insert

```
INSERT INTO emp (emp_id, emp_name, emp_city, emp_phone, emp_sal) VALUES(1,'ram', 'Hyderabad', 9848022338, 50000);
INSERT INTO emp (emp_id, emp_name, emp_city, emp_phone, emp_sal) VALUES(2,'robin', 'Delhi', 9848022339, 50000);
INSERT INTO emp (emp_id, emp_name, emp_city, emp_phone, emp_sal) VALUES(3,'rahman', 'Chennai', 9848022330, 45000);
```

Read

```
cqlsh:test> select * from emp;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 |     Delhi |    robin | 9848022339 |   50000
      3 |   Chennai |   rahman | 9848022330 |   45000
```

Update

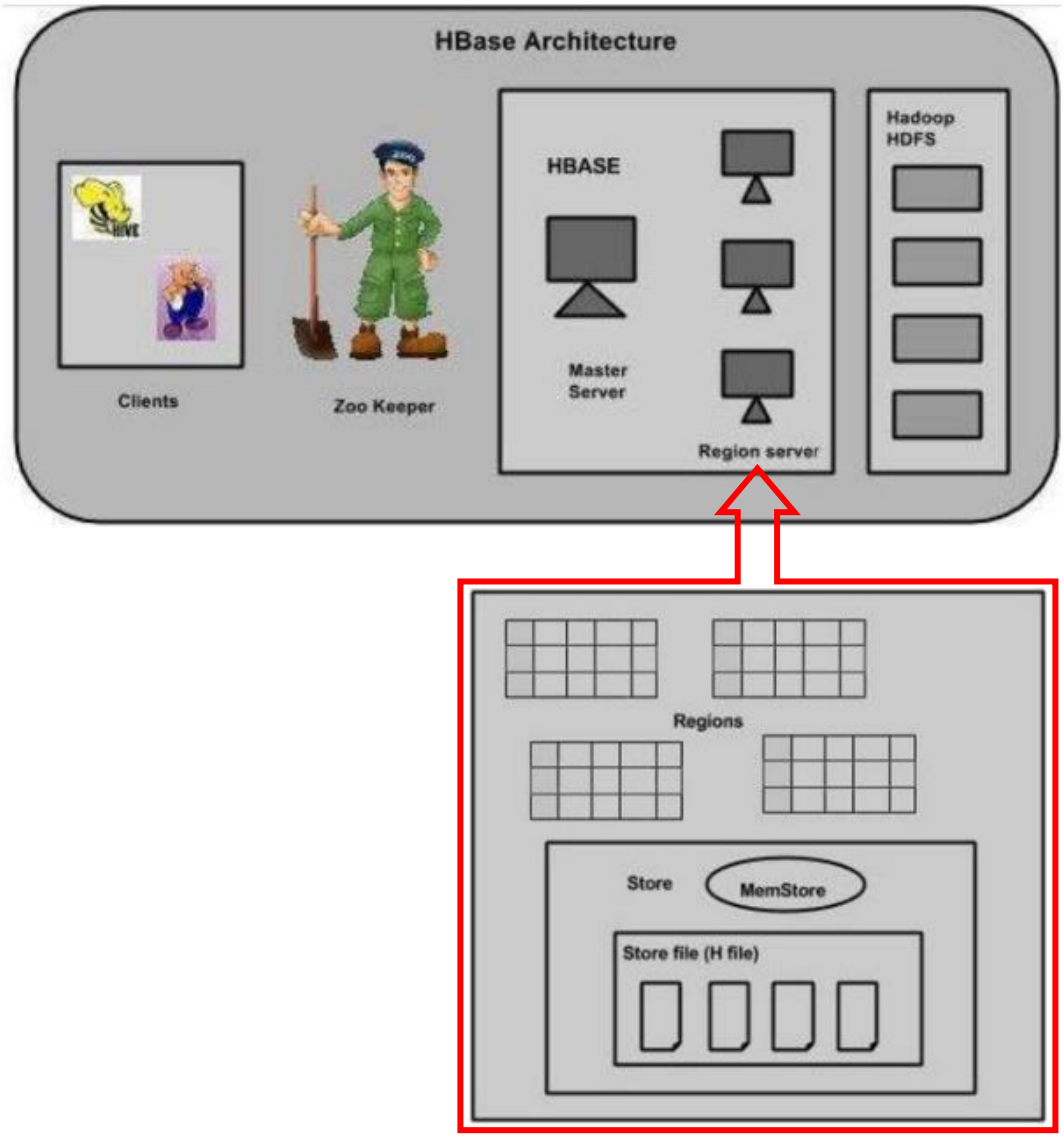
UPDATE emp SET emp_city='Bangalore',emp_sal=75000 WHERE emp_id=2;

```
cqlsh:test> select * from emp where emp_id=2;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      2 | Bangalore |    robin | 9848022339 |   75000
```

Delete

```
cqlsh:test> DELETE from emp where emp_id=2;
cqlsh:test> select * from emp;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      3 |   Chennai |   rahman | 9848022330 |   45000
```

1. OVERVIEW



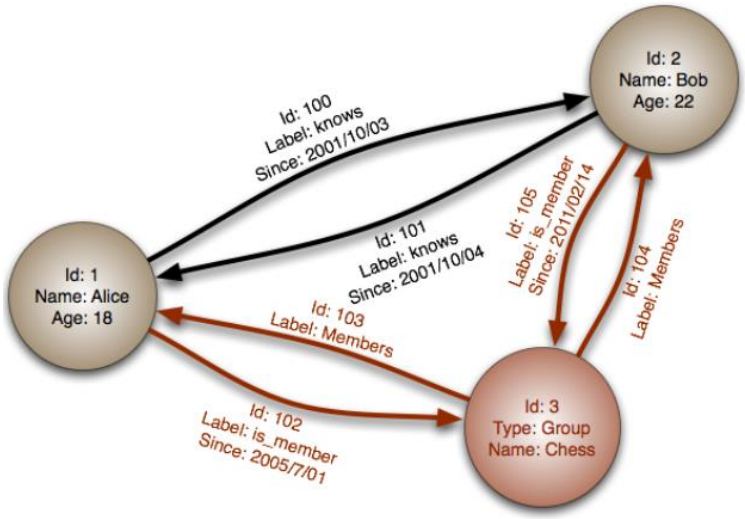
Year	Event
Nov 2006	Google released the paper on BigTable.
Feb 2007	Initial HBase prototype was created as a Hadoop contribution.
Oct 2007	The first usable HBase along with Hadoop 0.15.0 was released.
Jan 2008	HBase became the sub project of Hadoop.
May 2010	HBase became Apache top-level project.

- HBase is a distributed column-oriented **data store** built on top of the Hadoop file system.
- Data store instead of a database as it misses out on some important features of traditional RDBMs like typed columns, triggers, advanced query languages and **secondary indexes**.
- Enables fast random read/writes on HDFS (vanilla HDFS reads/writes sequentially)

2. INSTALLATION

Pre-req is Apache Hadoop, HBASE is much harder to install, maybe we will do it later!

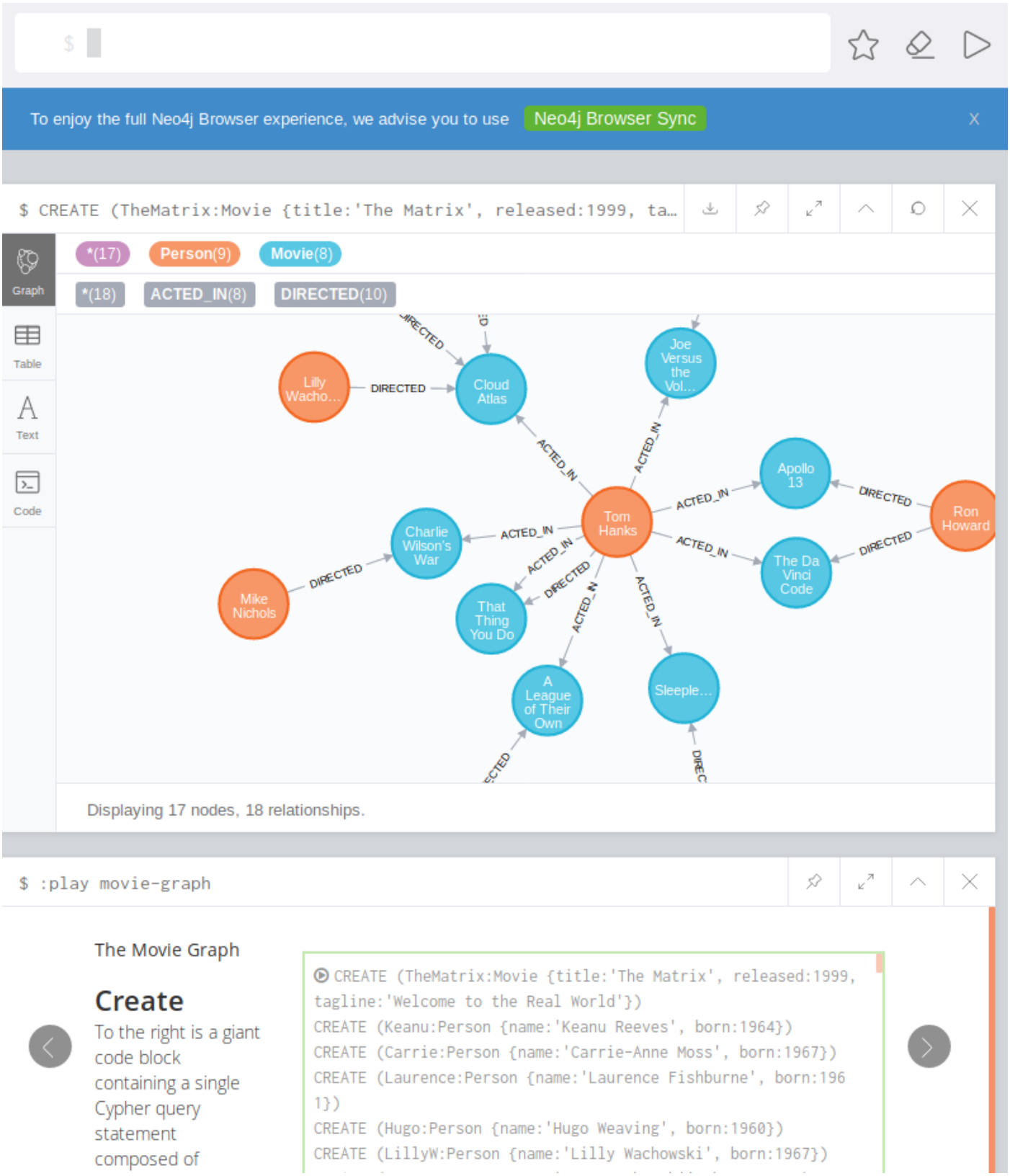
3. HELLO WORLD!



- In a graph database, each **vertex** is a record and each **edge** is a relationship between two nodes. Graph databases are optimized to represent complex relationships with multiple many-to-many relationships.
- Graphs databases offer high performance for data models with complex relationships, such as a social network.
- They are relatively new and are not yet widely-used; it might be more difficult to find development tools and resources. Many graphs can only be accessed with REST APIs.
- RDBMS vs Graph DB

Sn	RDBMS	Graph Database
1	Tables	Graphs
2	Rows	Nodes
3	Columns and Data	Properties and its values
4	Constraints	Relationships
5	Joins	Traversal

1. OVERVIEW



Written in Java, provides an awesome graphical web interface, also supports interaction through REST API

- Uses CQL(Cypher Query Language) to interact with the graph
- Supports ACID properties

2. INSTALLATION

Enter as root

```
sudo su
```

Install Neo4J

1. Import neo4j public GPG key

```
wget --no-check-certificate -O - https://debian.neo4j.org/neotechnology.gpg.key  | sudo apt-key add -
```

2. Create the list file for neo4j

```
echo 'deb http://debian.neo4j.org/repo  stable/' > /etc/apt/sources.list.d/neo4j.list
```

3. Update the system

```
apt update
```

4. Install neo4j

```
apt install neo4j
```

Start the service

```
systemctl start neo4j
```

3. HELLO WORLD!

http://localhost:7474/browser/

Very intuitive UI that helps you learn and experiment live with graph DB, that's it!

CHAPTER 5: SPECIAL PURPOSE STORE

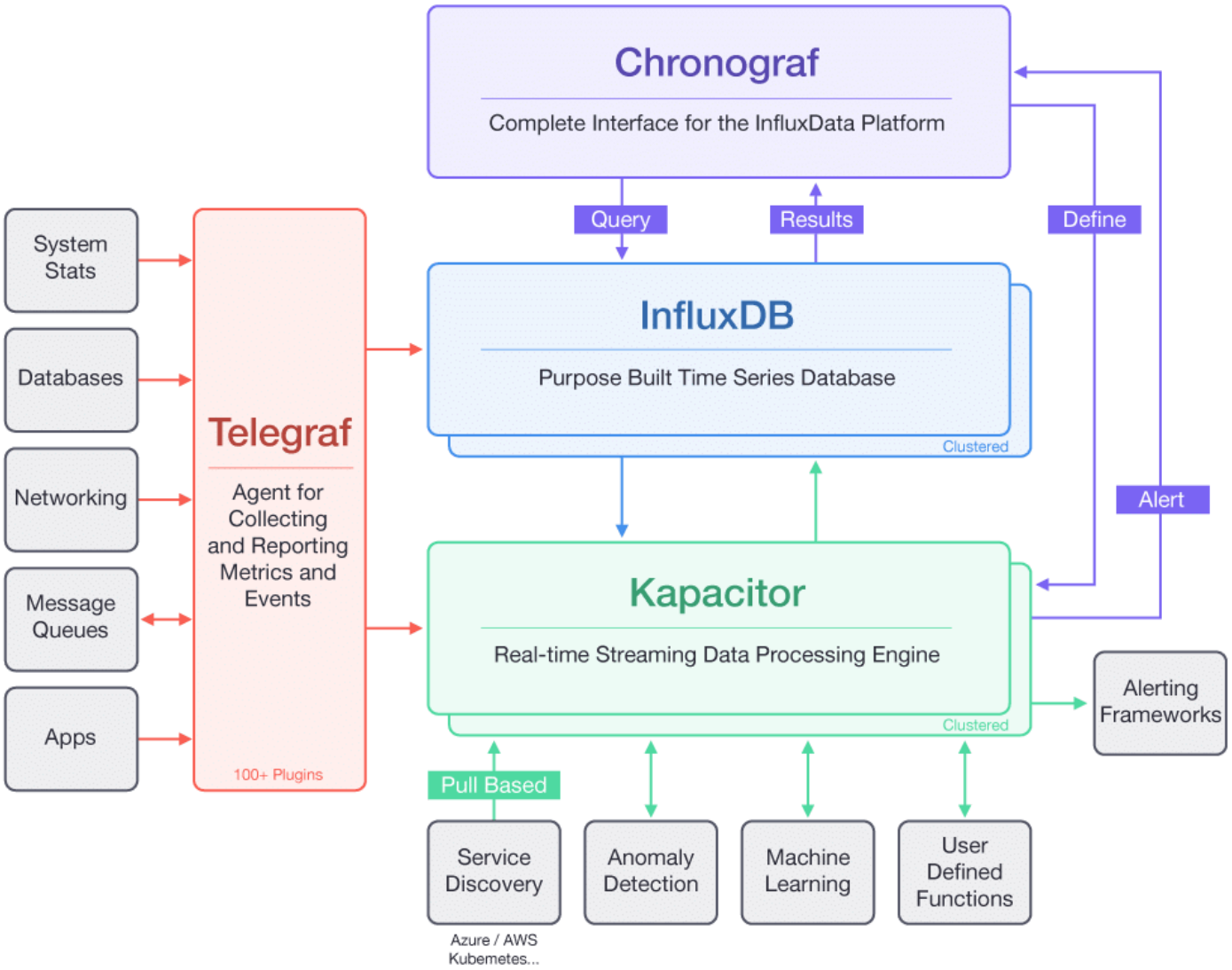
May be based on SQL or NoSQL DB or a combination of other technologies to cater to some very specialized requirements:

- Time series data store – log/sensor data coming at regular intervals (in large volumes)
- Object store – storing images, videos and other docs

1. TIME SERIES STORE

- what do self-driving Teslas, autonomous Wall Street trading algorithms, smart homes etc have in common? - constant data generation that can give a lot of insight into it, ideal case for time series DB!
- Time is not just a metric here but a primary axis
- Such stores can be optimized to utilize:
    - Data typically is an **insert** and not an update (read append only) except for corrections/delayed data
    - Data typically arrives in **timely order**
    - **Huge inflow** of data
- Relational databases fare poorly with very large datasets; NoSQL databases fare better at scale, but can still be outperformed by a database fine-tuned for time-series database (which can be based on **relational or NoSQL databases**)

1. OVERVIEW



- Written in **Go**, part of TICK stack (Telegraf, InfluxDB, Chronograph and Kapacitor)
- Designed to handle high write and query loads
- provides a **SQL-like** query language called InfluxQL
- Some quick metrics comparison between InfluxDB vs Others(from official docs):

Cassandra	*Elasticsearch*	*MongoDB*
Write throughput: **4.5x faster**	Write throughput: **6.1x faster**	Write throughput: **2.4x faster**
Disk storage: **2.1x less**	Disk storage: **2.5x less**	Disk storage: **20x less**
Query performance: **45x faster**	Query performance: **8.2x faster**	Query performance: **5.7x faster**
Download the technical paper	Download the technical paper	Download the technical paper

2. INSTALLATION

Install InfluxDB

Add the InfluxData repository to the file /etc/apt/sources.list.d/influxdb.list

```
echo "deb https://repos.influxdata.com/ubuntu bionic stable" | sudo tee /etc/apt/sources.list.d/influxdb.list
```

Import apt key

```
sudo curl -sL https://repos.influxdata.com/influxdb.key  | sudo apt-key add -
```

Update apt index and install influxdb

```
sudo apt-get update
sudo apt-get install -y influxdb
```

Start InfluxDB

```
sudo systemctl start influxdb
```

...(status, enable, stop, restart)

3. HELLO WORLD!

```
$ Influx
```

```
> create database test
```

Create - Inserting temperature & sound data:

```
> use test
Using database test
> INSERT temperature,officename=01 value=23.2
> INSERT temperature,officename=01 value=23.1
> INSERT temperature,officename=01 value=23.2
> INSERT temperature,officename=01 value=23.1
> INSERT temperature,officename=01 value=23.3
> INSERT sound,officename=01 value=50.1
> INSERT sound,officename=01 value=50.2
> INSERT sound,officename=01 value=55
> INSERT sound,officename=01 value=50.1
> INSERT sound,officename=01 value=50.25
```

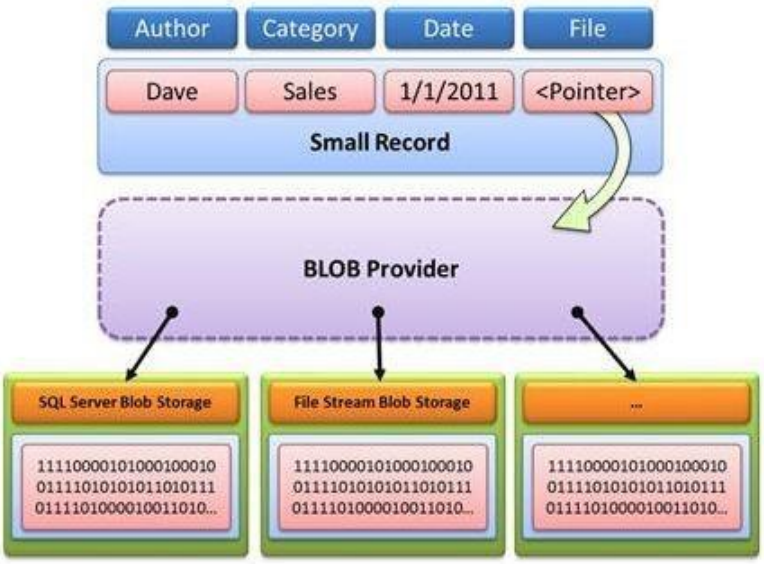
Read

```
> SELECT * from temperature;
name: temperature
time                officename value
----                ---------- -----
1574858691881496254 01         23.2
1574858711358863573 01         23.1
1574858711371003280 01         23.2
1574858711382594198 01         23.1
1574858713029557562 01         23.3
```

2. OBJECT STORE

PROBLEM STATEMENT

A common problem while building systems is how to store large binary objects (BLOB) like audio, video, image or documents.



1. Brute Force Solution

We can store them in an RDBMS using the BLOB datatype or equivalent, this will store them in the DB itself.

Why you should not do this?

- Impacts the DB performance drastically
- DB size becomes huge, creates problem for maintenance activities
- Most importantly, they are immutable and don't need any guarantees that an RDBMS provides

2. In-house Solution

Store the binary object in a filesystem and the metadata in an RDBMS (MySQL). Metadata can be id, the path of the file, author, category, number of views etc and can be indexed for faster search. Some sort of pre-processing can be done before storing the binary object in the file system like:

- Size/Quality reduction
- Common compressed format for storage
- De-duplication etc

Disadvantages: What will happen if a lot of users are trying to download/stream a large file say a video? Two issues:

1. This will create a lot of contention on the fileserver
2. as well as the network channel

3. Cloud Solution

Store the binary object in a CDN (Akamai, Cloudflare etc) or Amazon S3 and the link in MySQL. Now there are multiple copies of the binary object and possibly closer to the user hence solving the above problems.

CHAPTER 6: FULL TEXT SEARCH

- Means searching the entire text space (documents, tables etc)
- 2 ways of doing it:
    1. Serial search like GREP
    2. Index based search like Elasticsearch, involves 2 steps –
        1. Creation of index and then
        2. Searching in the index. [Every term]* will have an index entry pointing to all the locations where it's present.

    *Except for common stop words or derived words like drive, driving, drove – will be under a single entry 'drive'

ELASTICSEARCH

1. OVERVIEW

- Clustered architecture
- Built in Java and uses Apache Lucene
- Part of the ELK stack (Elasticsearch, Logstash & Kibana)
- Provides REST API based interface to interact with it

2. INSTALLATION

Install Elasticsearch

- Install Elasticsearch GPG key

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
```

- Install the apt-transport-https package

```
sudo apt-get install apt-transport-https
```

- Add Elasticsearch repo to Ubuntu

```
echo "deb https://artifacts.elastic.co/packages/6.x/apt stable main" | sudo tee -a /etc/apt/sources.list.d/elastic-6.x.list
```

- Install it

```
sudo apt-get update
sudo apt-get install elasticsearch
```

Configure Elasticsearch

```
sudo nano /etc/elasticsearch/elasticsearch.yml
network.host: "localhost"
```

http.port:9200

Start Elasticsearch

```
sudo systemctl start elasticsearch
```

3. HELLO WORLD!

Test Elasticsearch - http://localhost:9200

Create

```
curl -XPOST 'http://localhost:9200/blog/user/ramakant' -H 'Content-Type: application/json' -d '{ "name" : "Ramakant Bharti" }'
```

Read

```
curl -XGET 'http://localhost:9200/blog/user/ramakant'
```

```
{"_index":"blog","_type":"user","_id":"ramakant","_version":7,"_seq_no":9,"_primary_term":7,"found":true,"_source":{ "name" : "Ramakant Bharti" }}
```