ETL of Big Data and Kafka

张春阳

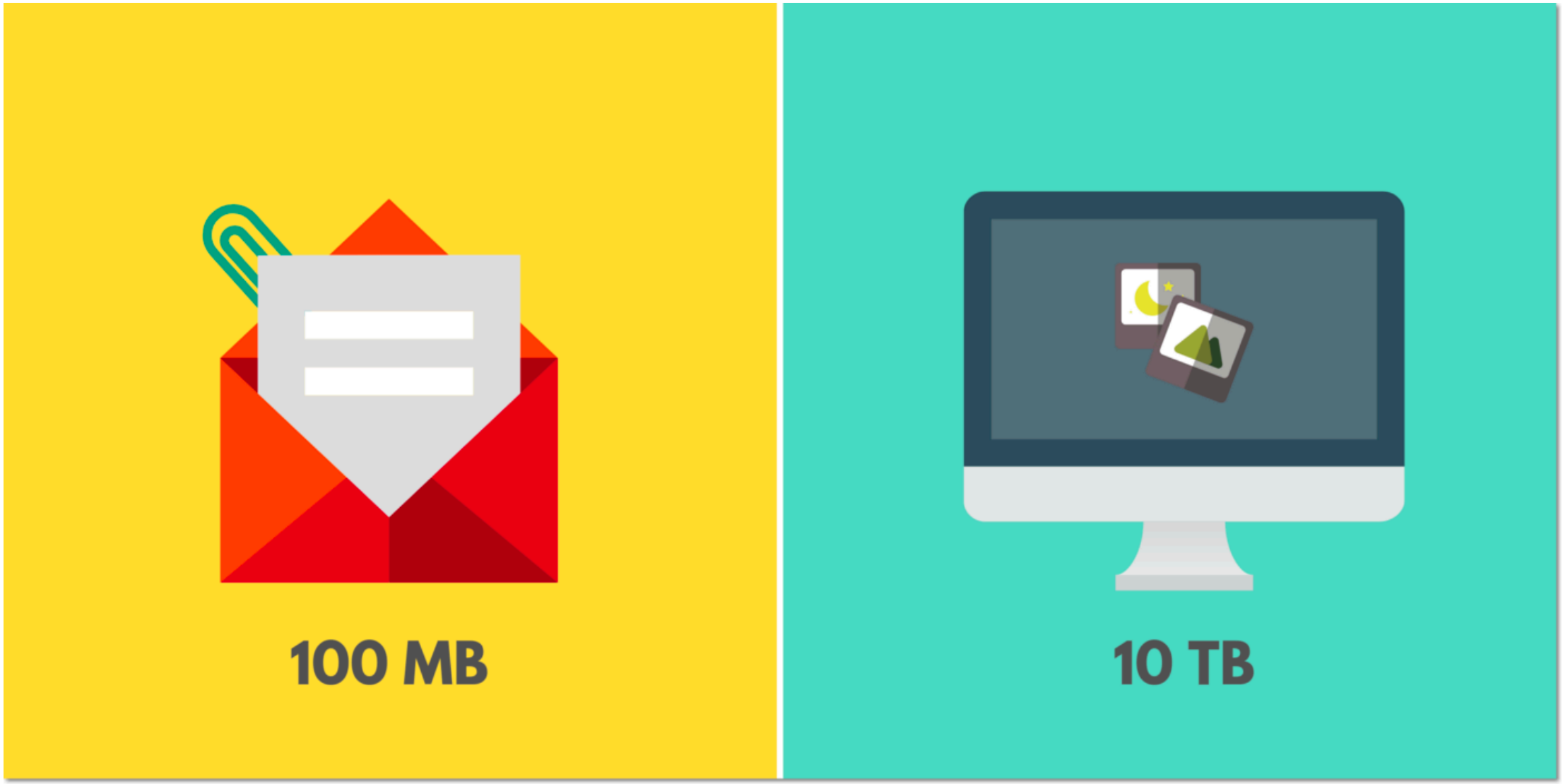
张春阳
开课吧人工智能高级导师
清华大学（MEM）硕士
曾任新东方教育研究院人工智能部门部门主管 百度大数据算法工程师

Review

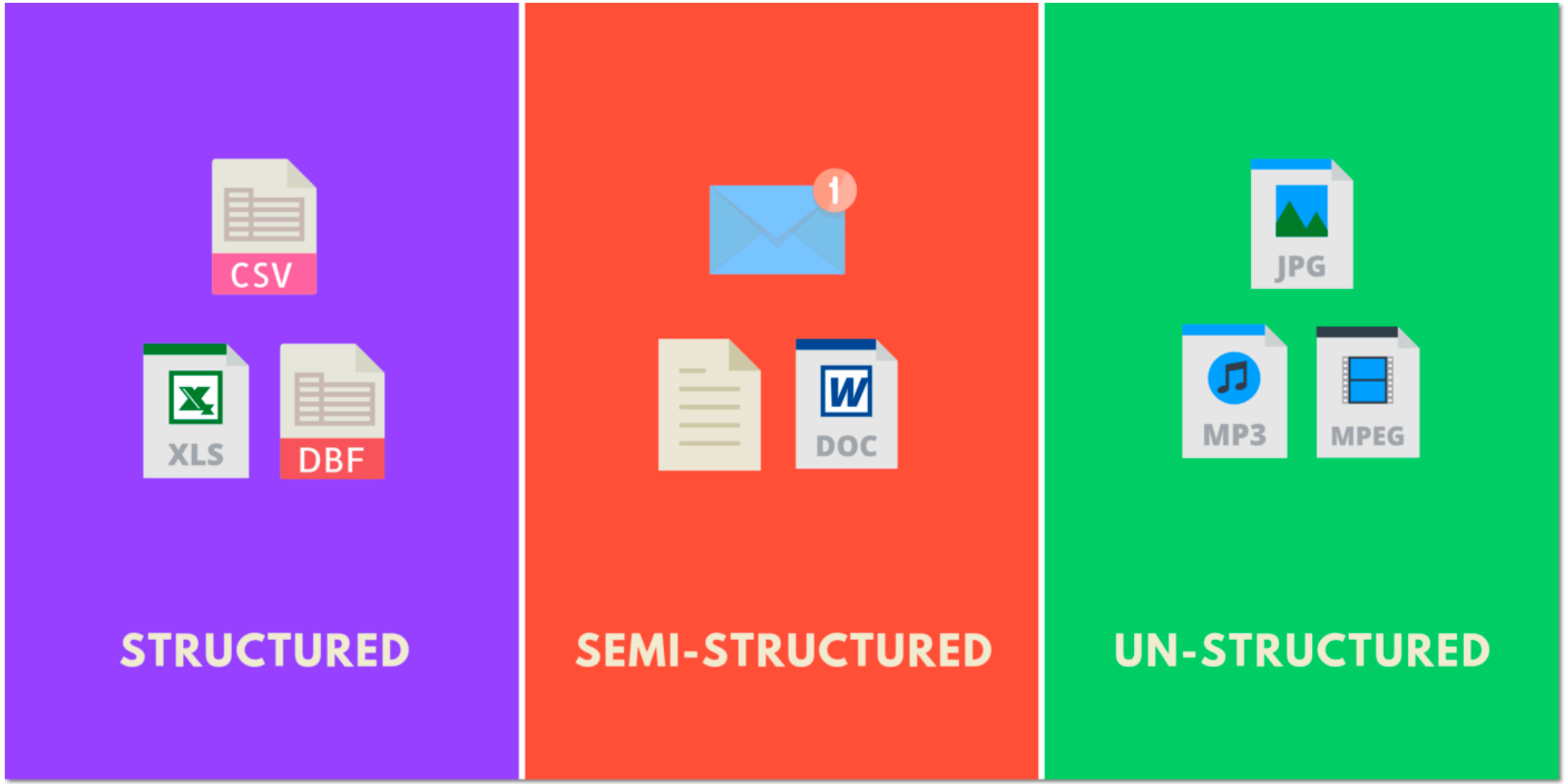
What is Big Data?

Big Data refers to a huge volume of data, that cannot be stored and processed using the traditional computing approach within a given time frame.

How huge this data needs to be? To be termed as Big Data?



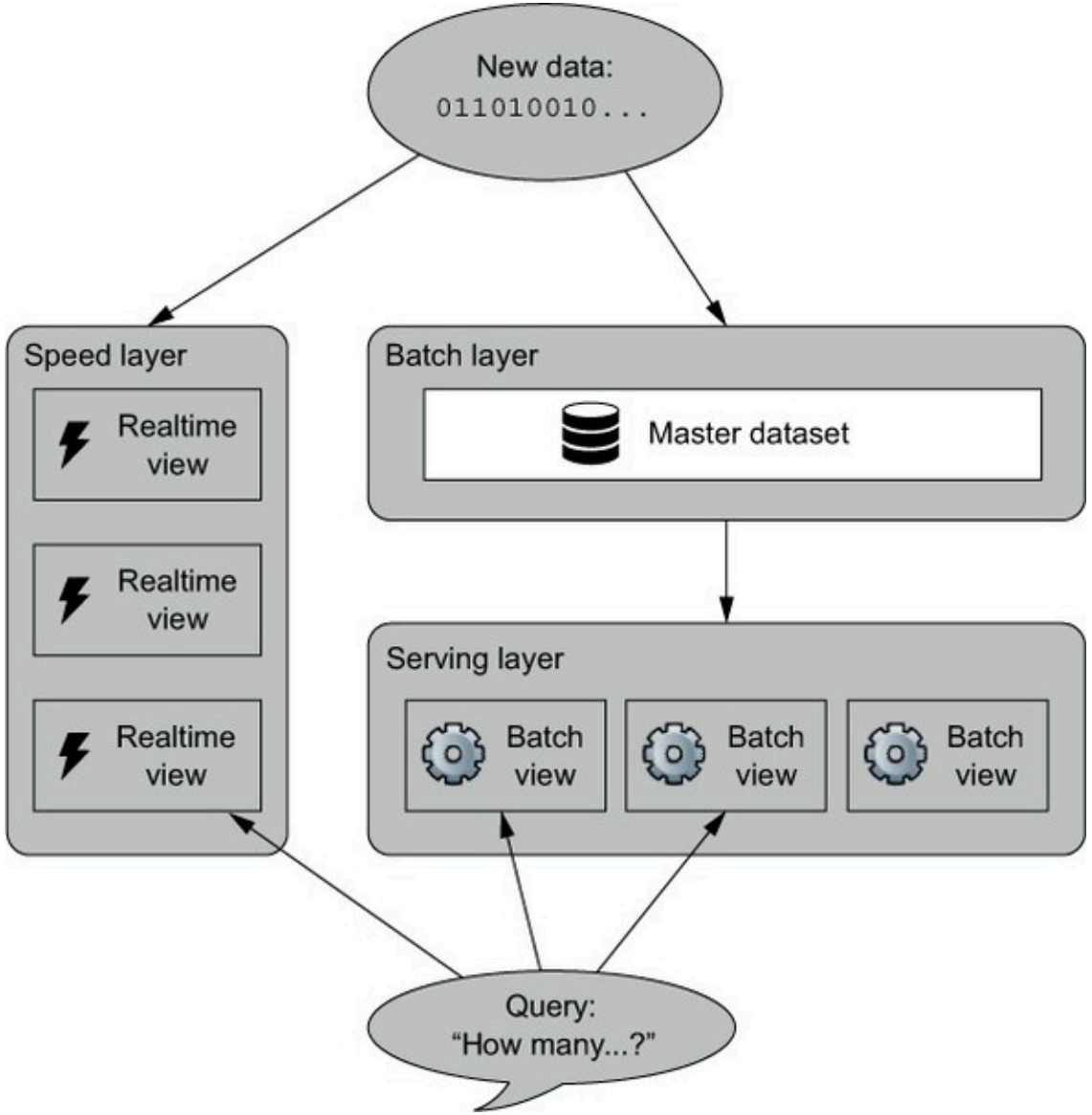
How Big Data is Classified?



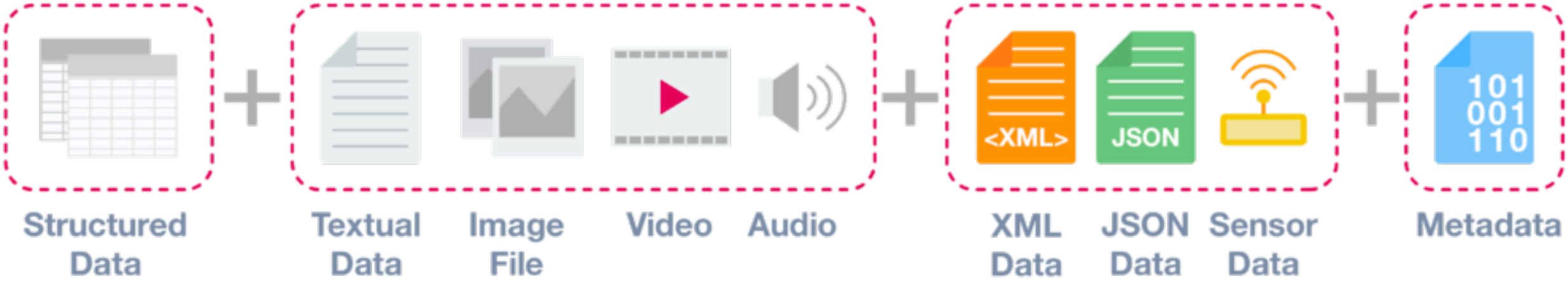
Characteristics of Big Data

Lambda Architecture

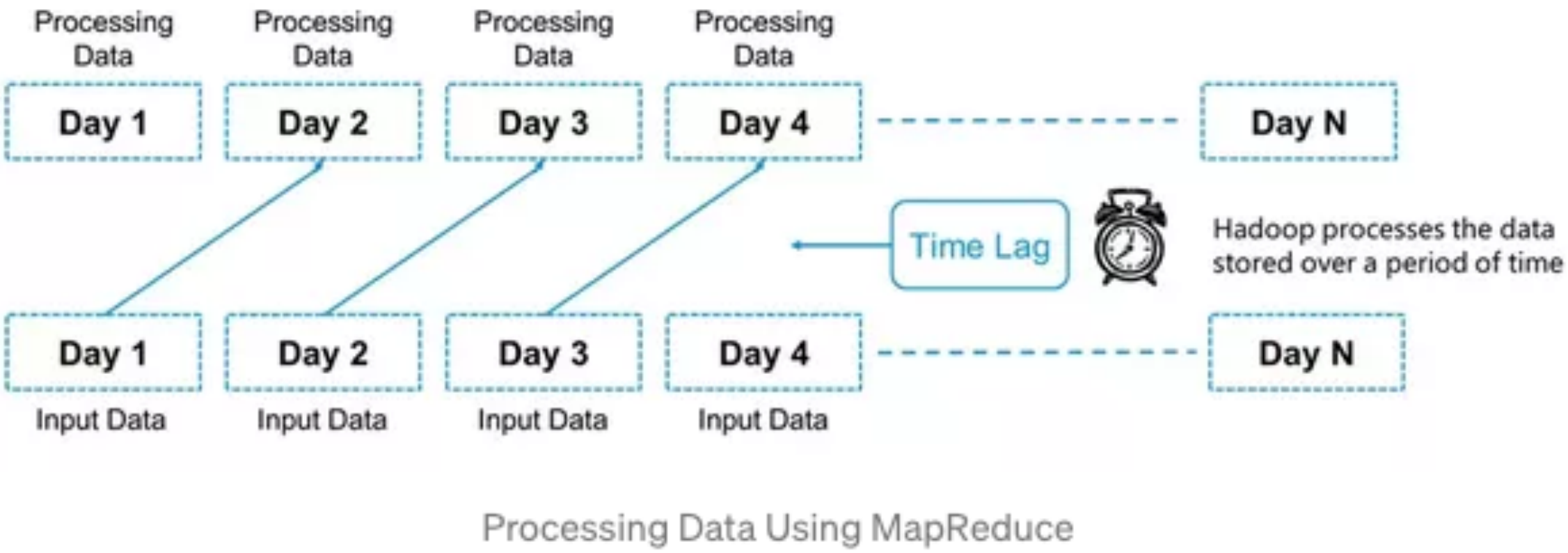
Popular Lambda Architecture of Big Data



Popular Architecture of Big Data - Input



Popular Architecture of Big Data - Batch Layer



Processing Data Using MapReduce

Popular Architecture of Big Data - Serving Layer

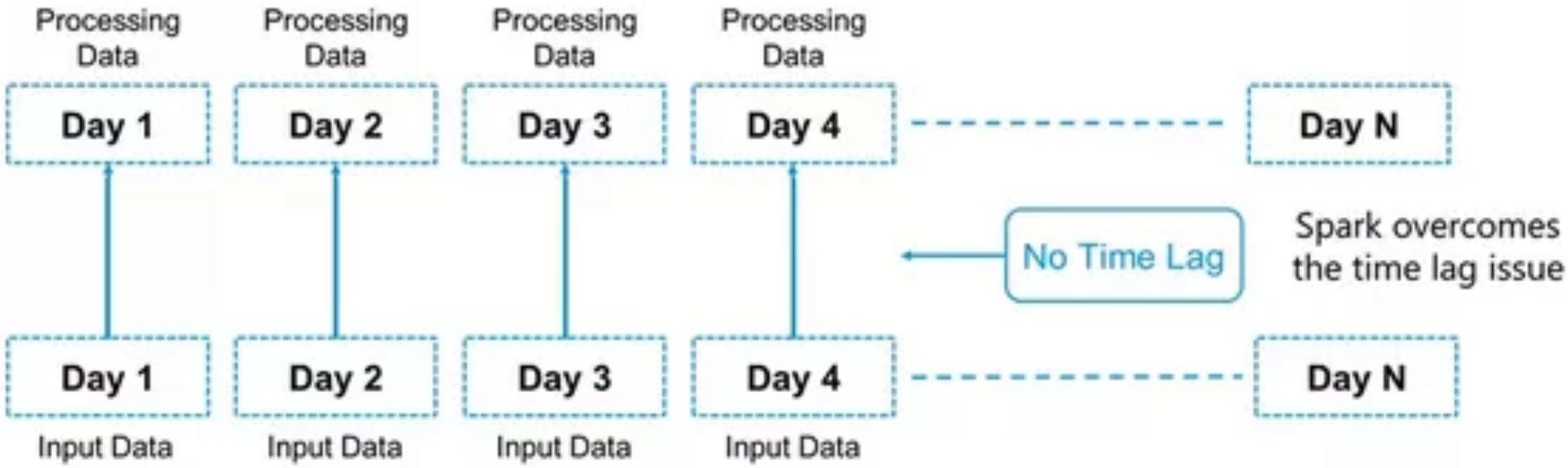
Serving Components

30 June 2014

Merge/Low-Latency Databases

Technology	Does it fit	Maturity	Ease of use	API Language	Comments
ElephantDB	★★★	★	★	Clojure	
SploutSQL	★★★	★	★★	Java	
Voldemort (with a ReadOnly backend)	★★	★★	★★	Java	
HBase (bulk loading)	★★	★★	★★	Java	
Druid	★★★	★★	★	Java	originates from Metamarkets

Popular Architecture of Big Data - Speed Layer



How to Create Machines for Experiments

VirtualBox

- About
- Screenshots
- Downloads
- Documentation
  - End-user docs
  - Technical docs
- Contribute
- Community

VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. Not only is VirtualBox an extremely feature rich, high performance product for enterprise customers, it is also the only professional solution that is freely available as Open Source Software under the terms of the GNU General Public License (GPL) version 2. See "About VirtualBox" for an introduction.

Presently, VirtualBox runs on Windows, Linux, Macintosh, and Solaris hosts and supports a large number of guest operating systems including but not limited to Windows (NT 4.0, 2000, XP, Server 2003, Vista, Windows 7, Windows 8, Windows 10), DOS/Windows 3.x, Linux (2.4, 2.6, 3.x and 4.x), Solaris and OpenSolaris, OS/2, and OpenBSD.

VirtualBox is being actively developed with frequent releases and has an ever growing list of features, supported guest operating systems and platforms it runs on. VirtualBox is a community effort backed by a dedicated company: everyone is encouraged to contribute while Oracle ensures the product always meets professional quality criteria.

Hot picks:

- Pre-built virtual machines for developers at ⇨Oracle Tech Network
- **Hyperbox** Open-source Virtual Infrastructure Manager ⇨project site
- **phpVirtualBox** AJAX web interface ⇨project site

ORACLE

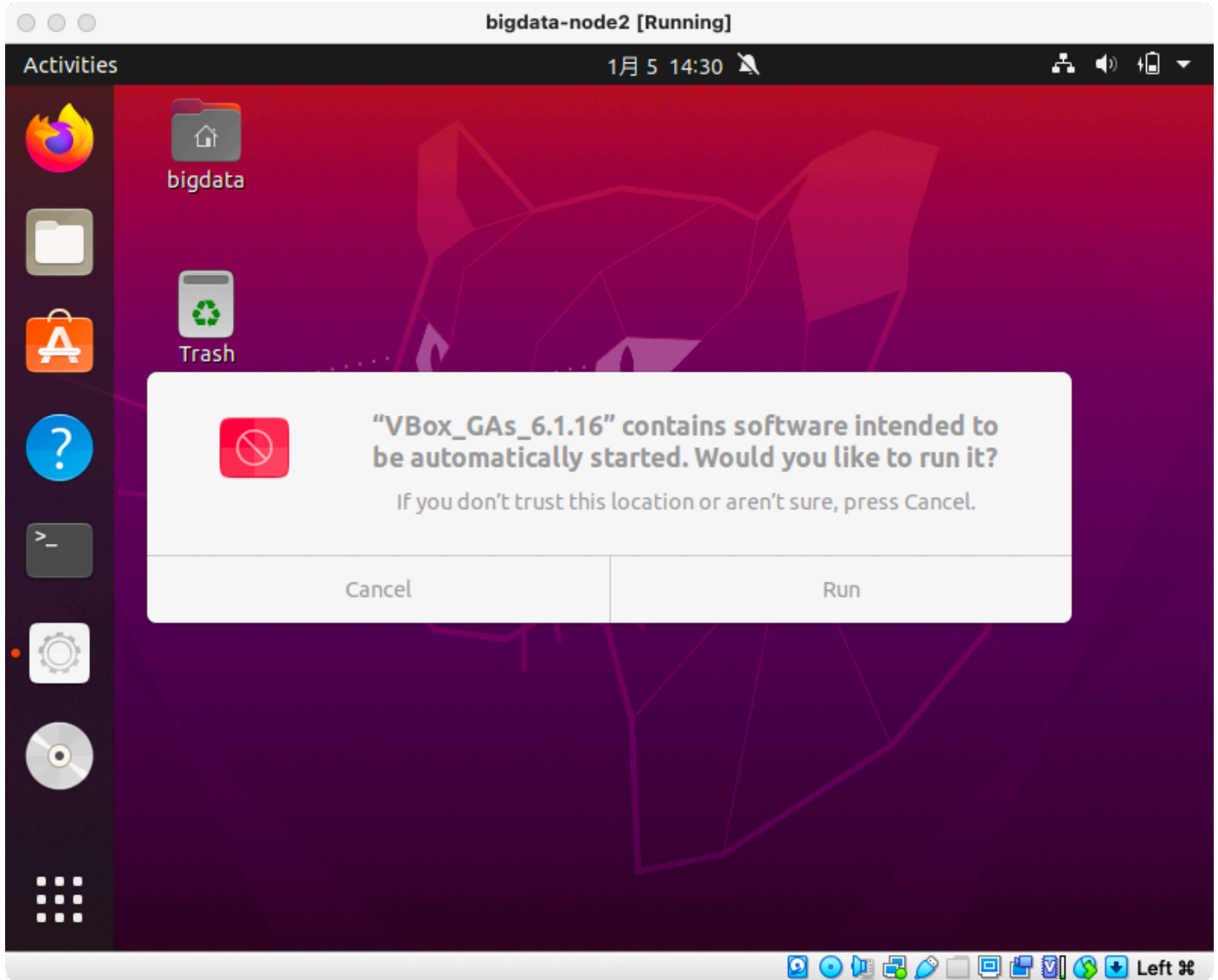
Contact – Privacy policy – Terms of Use

https://www.virtualbox.org/

Machines Setting

	bigdata-node1	bigdata-node2	bigdata-node3
Memory	4G(3G)	4G(3G)	4G(3G)
Disk	40G	40G	40G

Clipboard Share



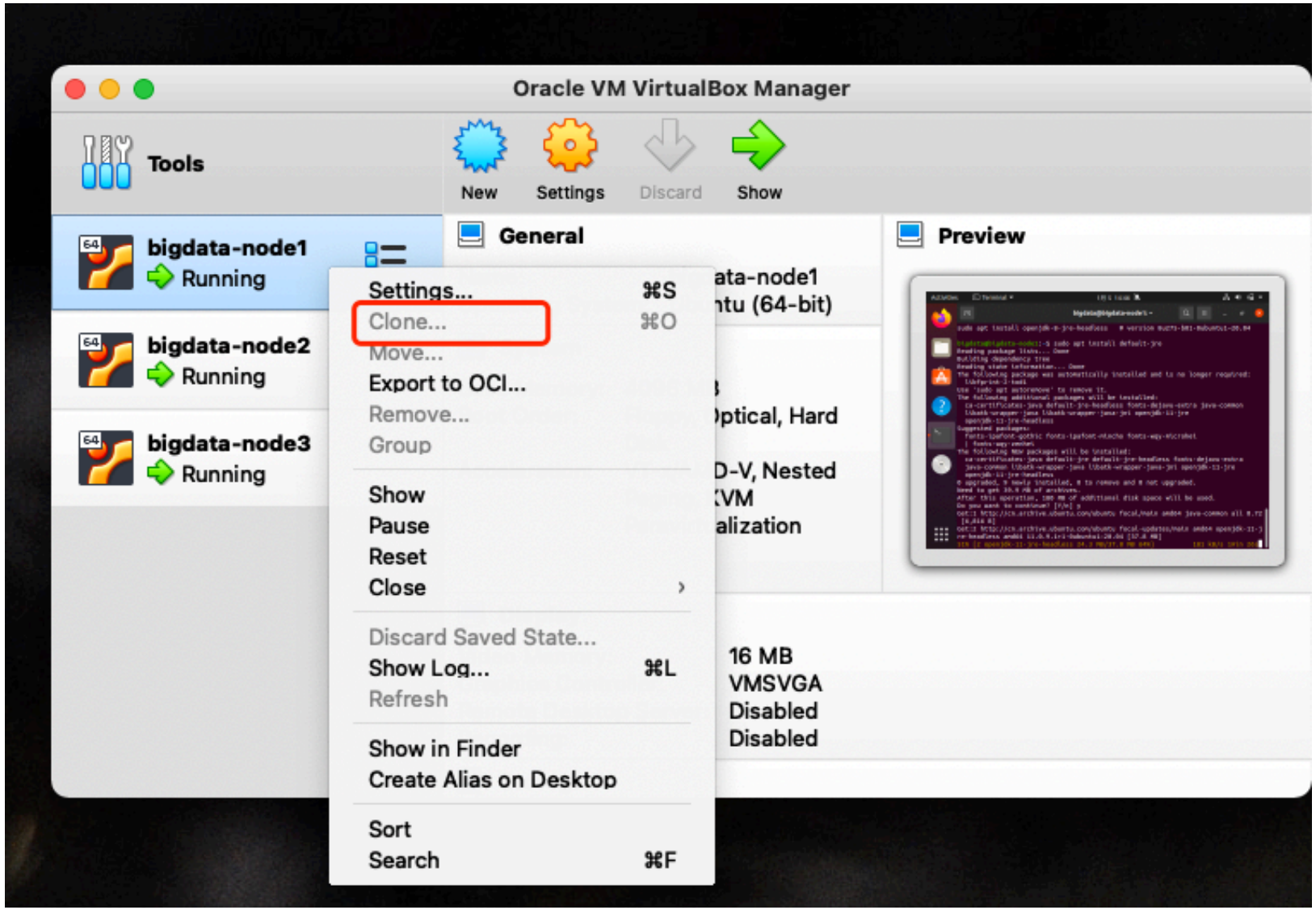
1. Devices/Shared Clipboard
2. Insert Guest Addition

Base Environment Setup

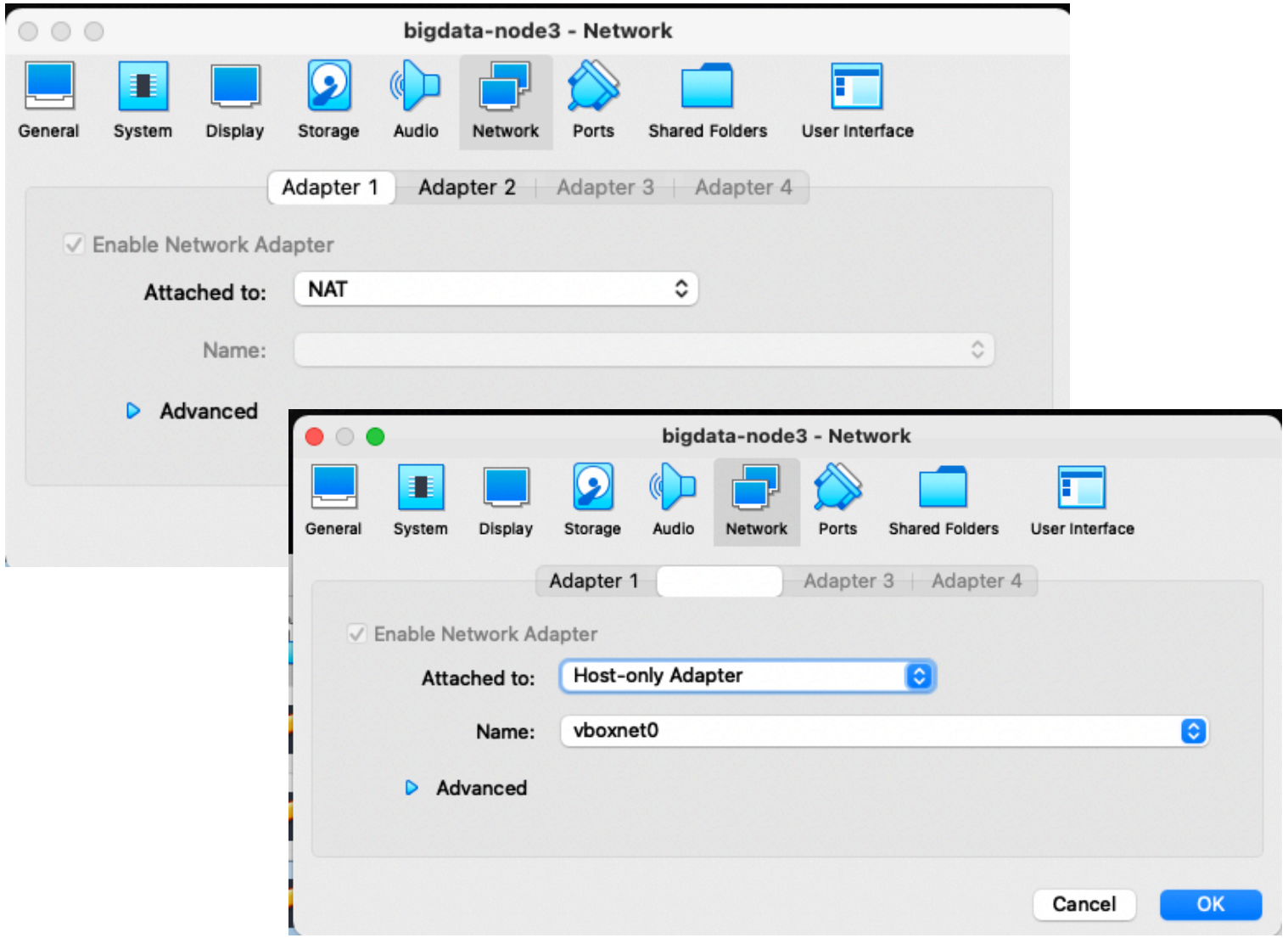
```
# change root pass
sudo su -
passwd root
sudo chown -R bigdata:bigdata /opt/bigdata
# install essential software
sudo apt upgrade -y
sudo apt install vim net-tools -y
sudo apt install software-properties-common
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt install python3.8 -y
sudo apt install default-jre -y
sudo apt install openjdk-11-jdk-headless -y
sudo apt install openssh-server -y
systemctl status ssh
```

```
# install essential software
sudo mkdir /opt/bigdata
sudo mkdir /data
```

Machine Clone

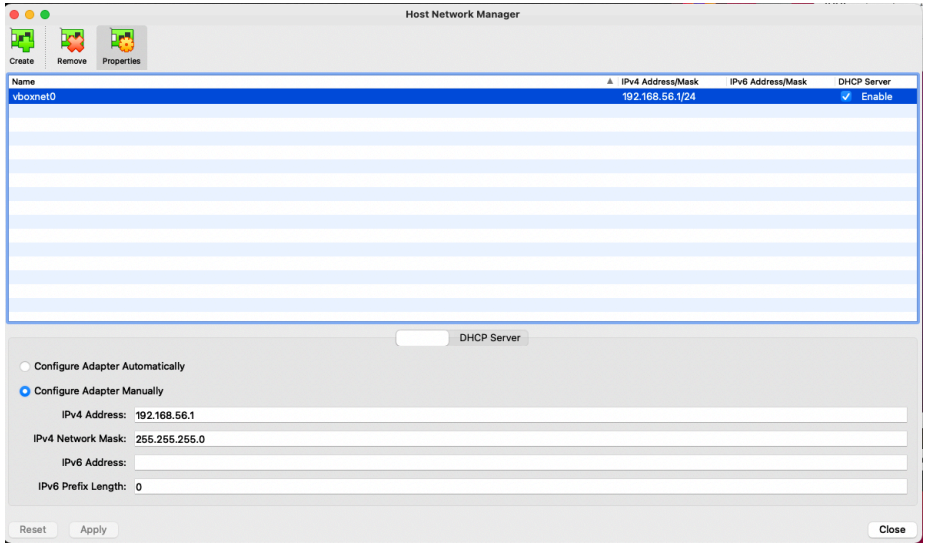


Network - NAT & HostOnly

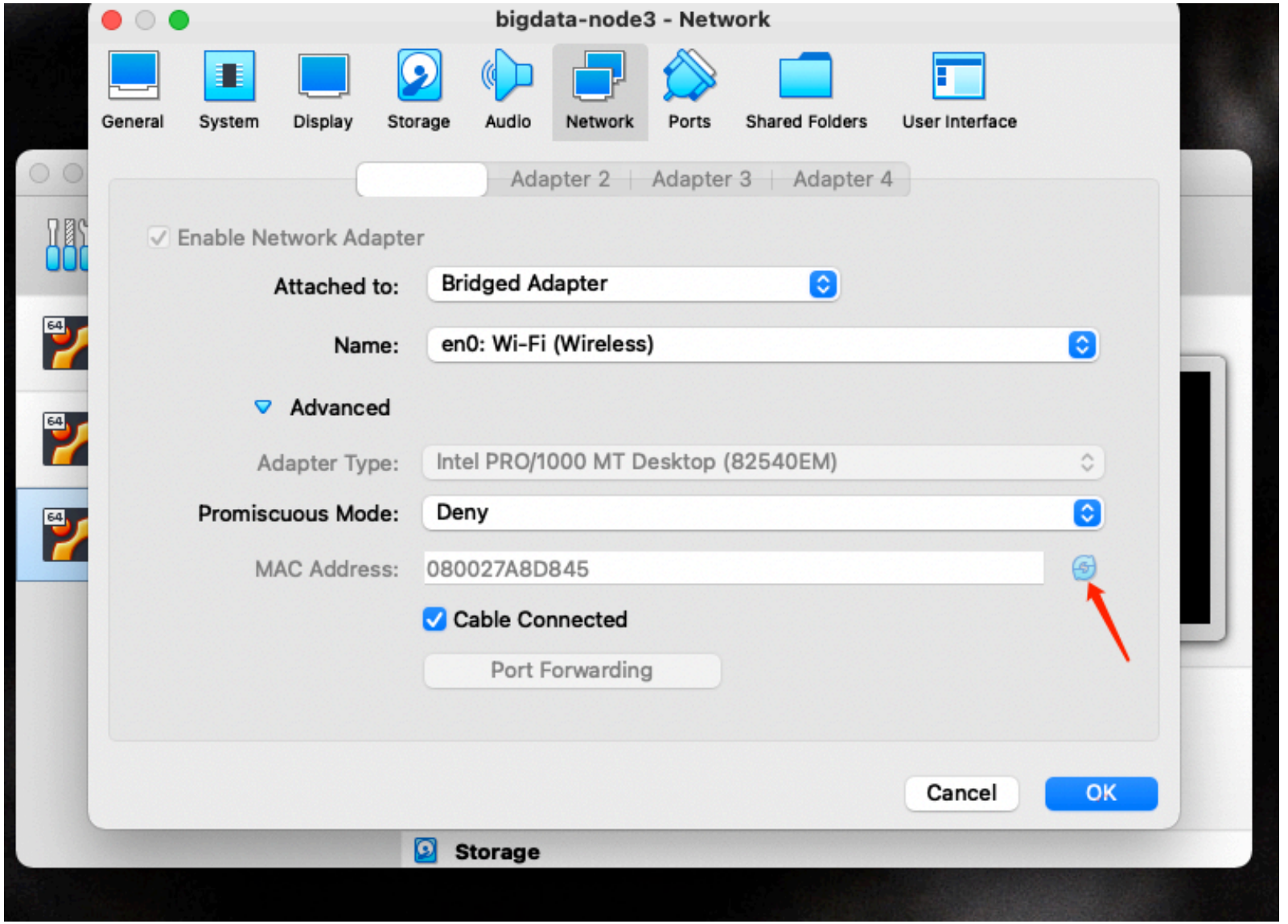


network setting

File/HostNetworkManager/Add



Network - Mac Conflict

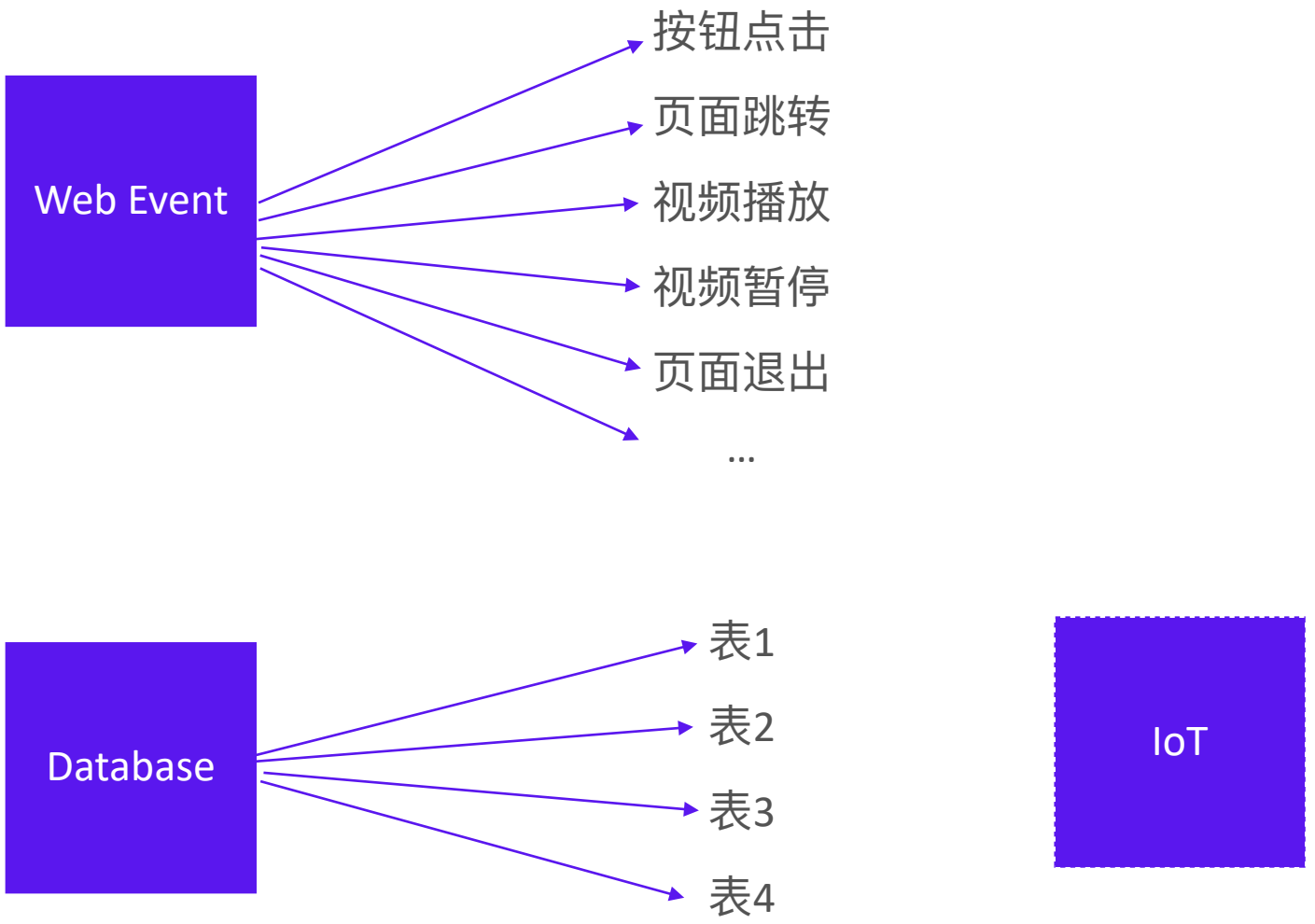


Machine Name and Hostname

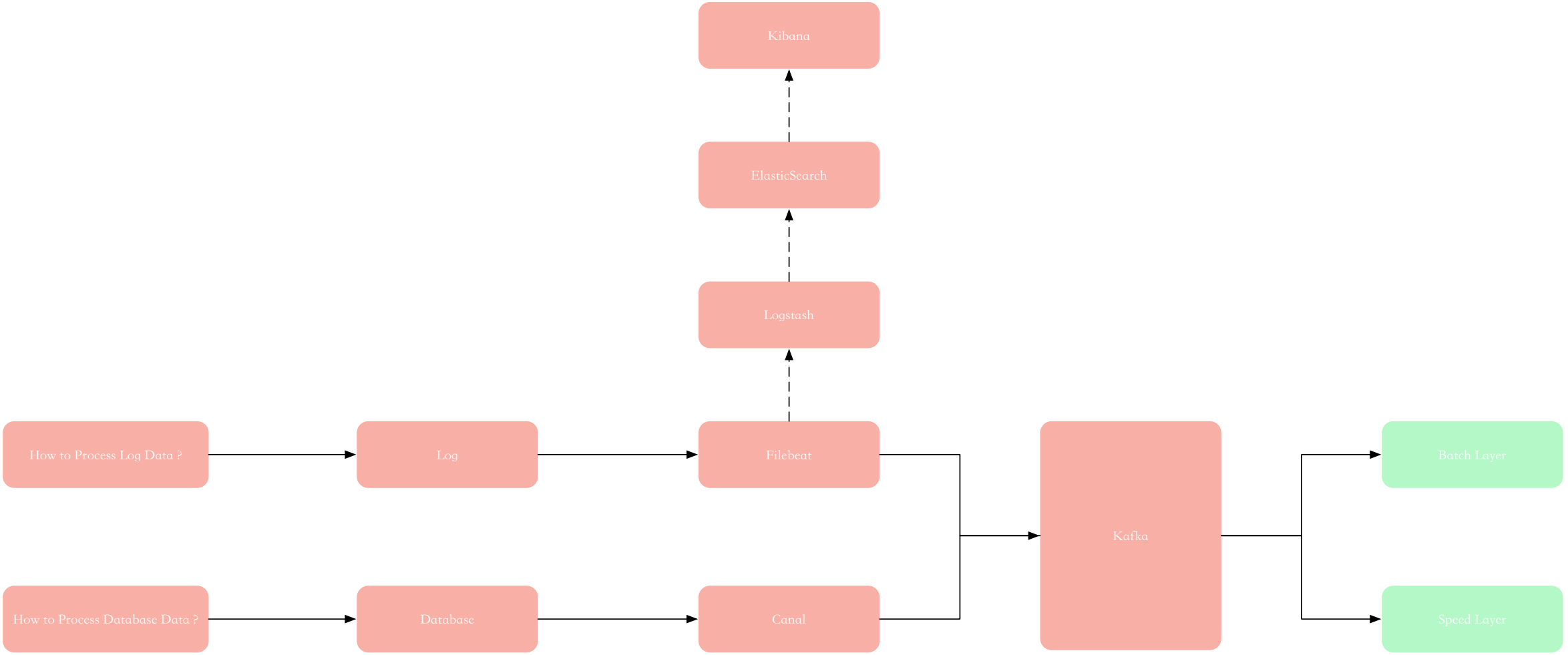
1. sudo vim /etc/hostname
2. sudo vim /etc/hosts

Data in your System

What’s data in our System ?

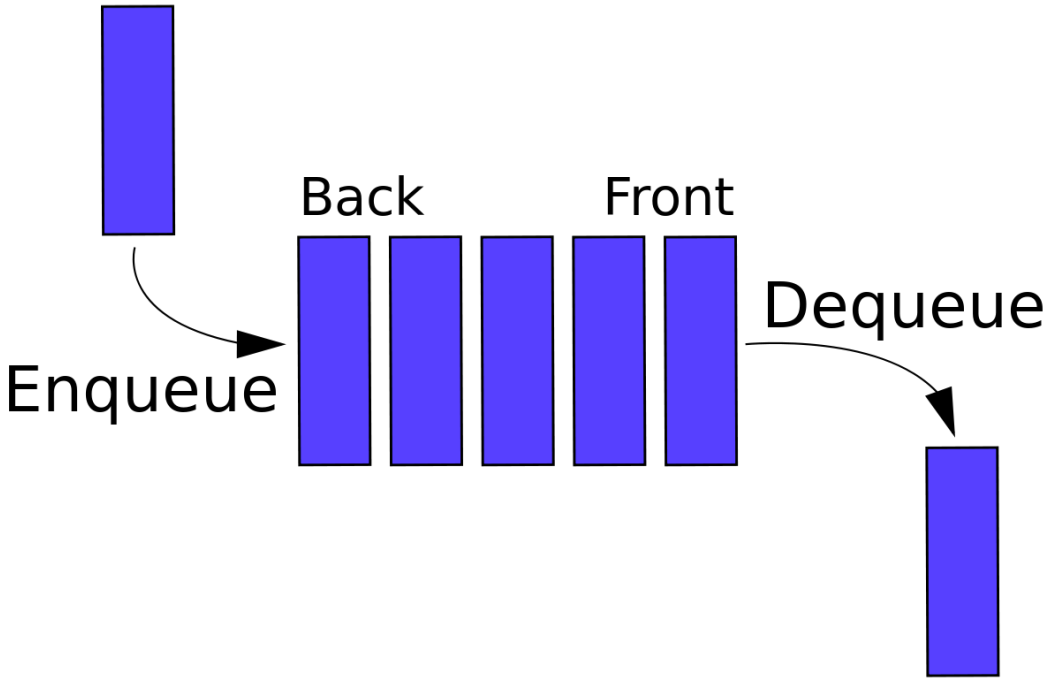


Our Data Target



Kafka

What is Kafka?

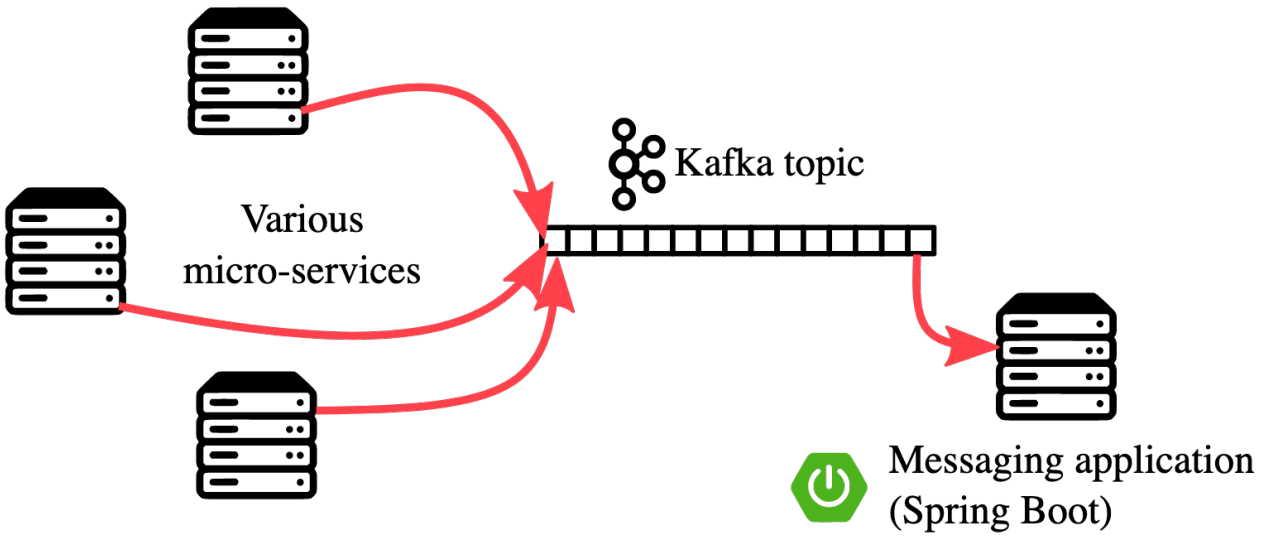


What is Kafka?

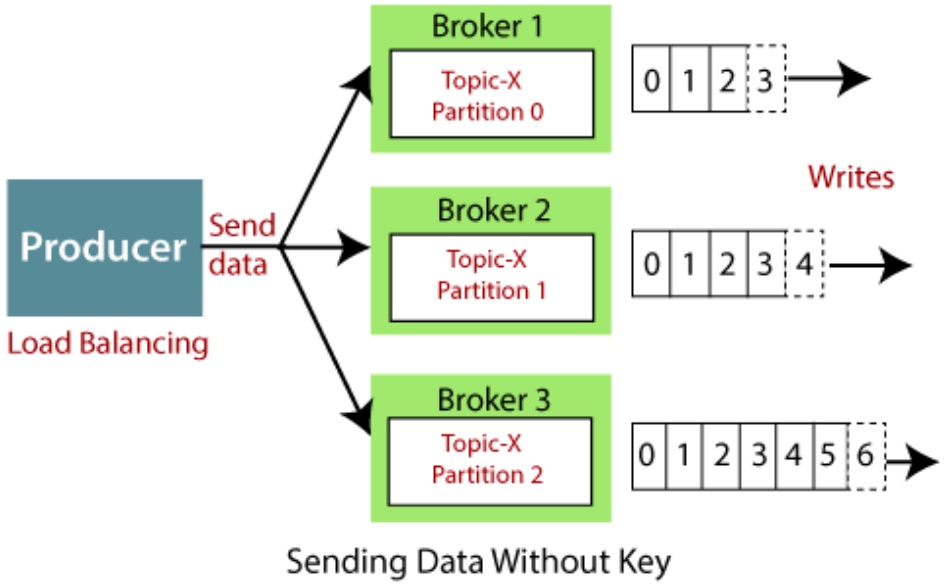
Kafka was originally developed at LinkedIn in 2011 and has improved a lot since then. Nowadays, it's a whole platform, allowing you to redundantly store absurd amounts of data, have a message bus with huge throughput (millions/sec), and use real-time stream processing on the data that goes through it all at once.

This is all well and great, but stripped down to its core, Kafka is a distributed, horizontally scalable, fault-tolerant commit log.

Distributed



A *distributed* system is one which is split into multiple running machines, all of which work together in a cluster to appear as one single node to the end user. Kafka is distributed in the sense that it stores, receives, and sends messages on different nodes (called *brokers*).
The benefits to this approach are high scalability and fault tolerance.



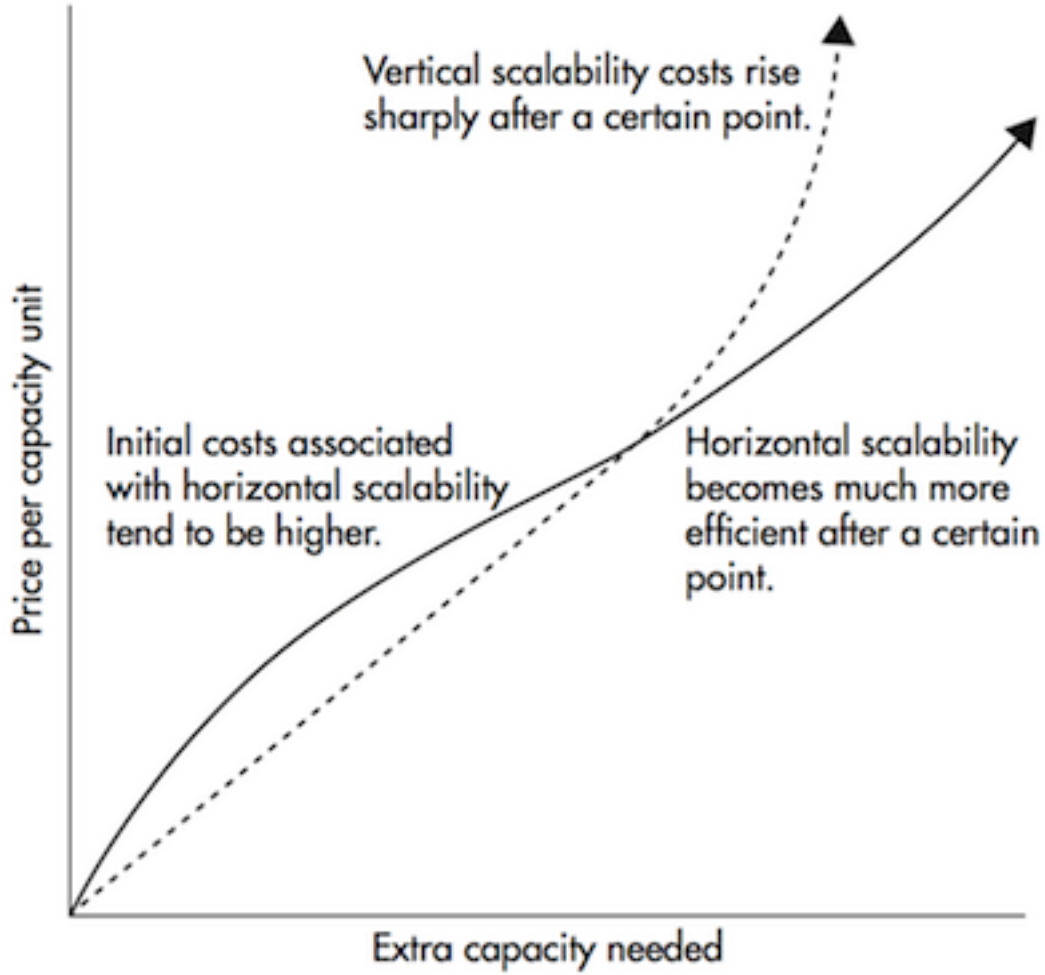
Horizontally scalable

For instance, you have a traditional database server that's starting to get overloaded.

The way to get this solved is to simply increase the resources (CPU, RAM, SSD) on the server. This is called *vertical scaling* — where you add more resources to the machine. There are two big disadvantages to scaling upwards:

- There are limits defined by the hardware. You cannot scale upwards indefinitely
- It usually requires downtime, something which big corporations can't afford

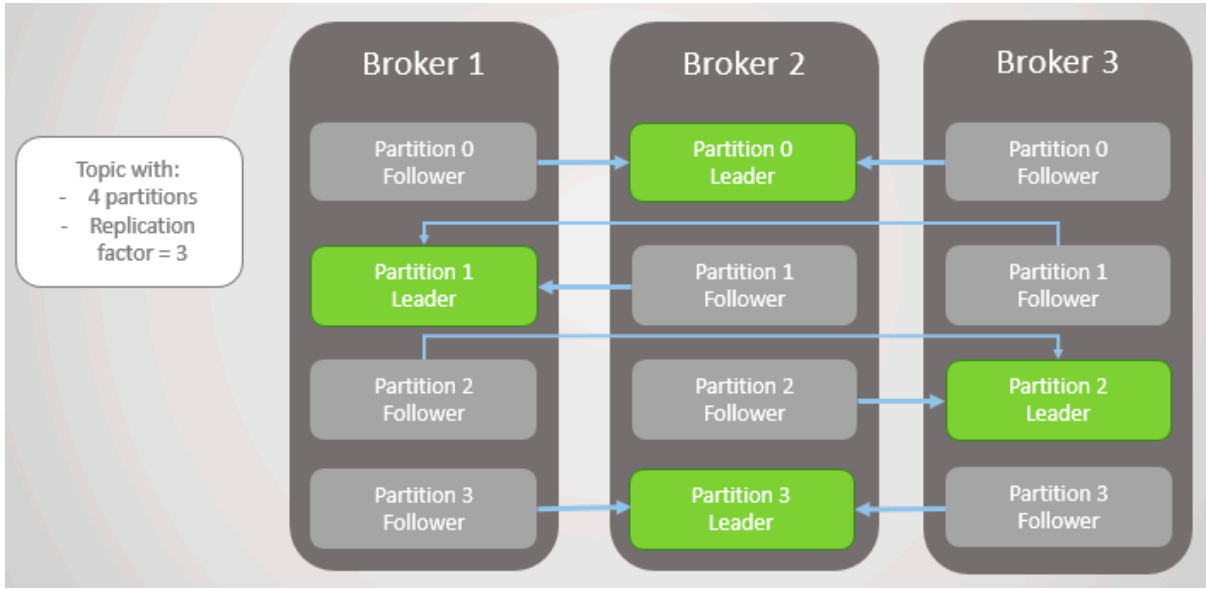
Horizontal scalability is solving the same problem by throwing more machines at it. Adding a new machine doesn't require downtime, nor are there any limits to the amount of machines you can have in your cluster. The catch is not all systems support horizontal scalability, as they're not designed to work in a cluster, and those that are are usually more complex to work with.



Fault-tolerant

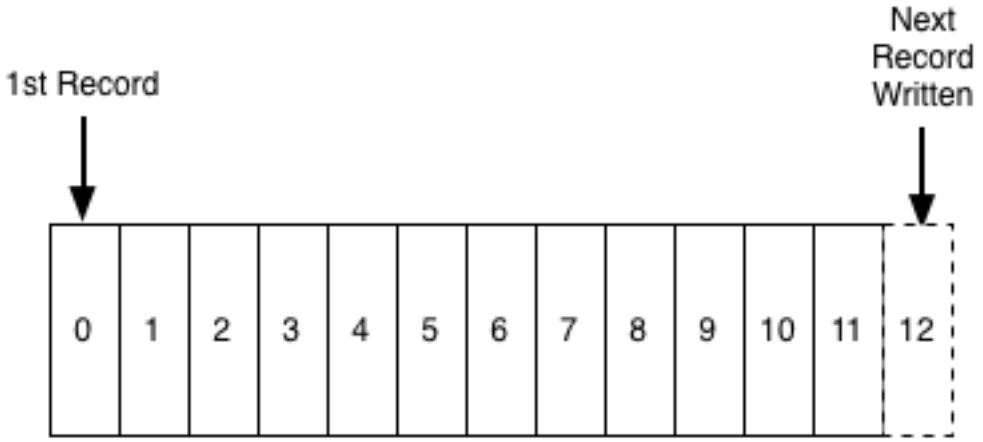
Something that emerges in nondistributed systems is they have a single point of failure (SPoF). If your single database server fails (as machines do) for whatever reason, you're screwed.

Distributed systems are designed in such a way to accommodate failures in a configurable way. In a 5-node Kafka cluster, you can have it continue working even if two of the nodes are down. It's worth noting that fault tolerance is at a direct trade-off with performance, as in the more fault-tolerant your system is, the less performant it is.



Commit log

A *commit log* (also referred to as *write-ahead log* or a *transaction log*) is a persistent-ordered data structure that only supports appends. You can't modify or delete records from it. It's read from left to right and guarantees item ordering.



https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

Are you telling me that Kafka is such a simple data structure?

In many ways, yes. This structure is at the heart of Kafka and is invaluable, as it provides ordering, which in turn provides deterministic processing. Both of which are nontrivial problems in distributed systems.

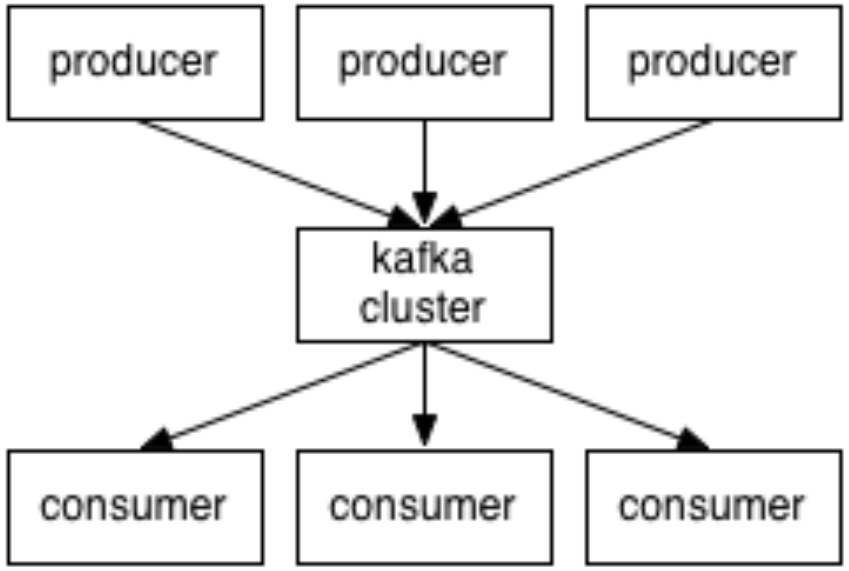
Kafka actually stores all of its messages to disk (more on that later), and having them ordered in the structure lets it take advantage of sequential disk reads.

- Reads and writes are a constant time O(1) *(knowing the record ID)*, which compared to other structure's O(log N) operations on disk is a huge advantage, as each disk seek is expensive
- Reads and writes don't affect each other. Writing wouldn't lock reading and vice versa (as opposed to balanced trees).

These two points have huge performance benefits, since the data size is completely decoupled from performance. Kafka has the same performance whether you have 100 KB or 100 TB of data on your server.

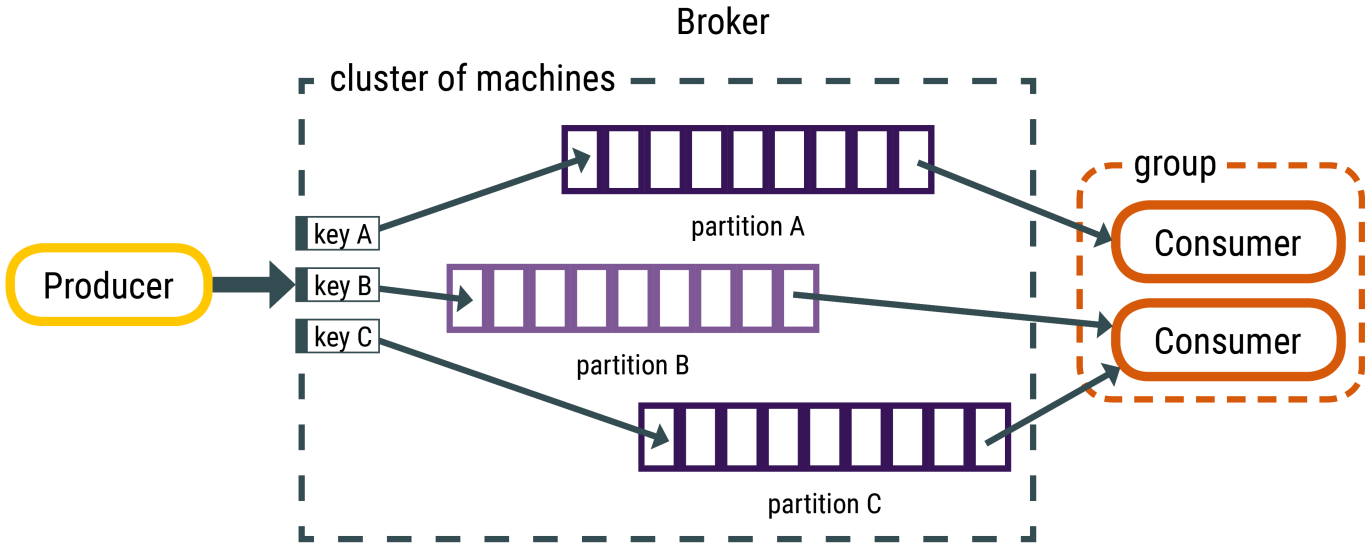
How Does it Work?

Applications *(producers)* send messages *(records)* to a Kafka node *(broker)*, and said messages are processed by other applications called *consumers*. Said messages get stored in a *topic* and consumers subscribe to the topic to receive new messages.



Topic

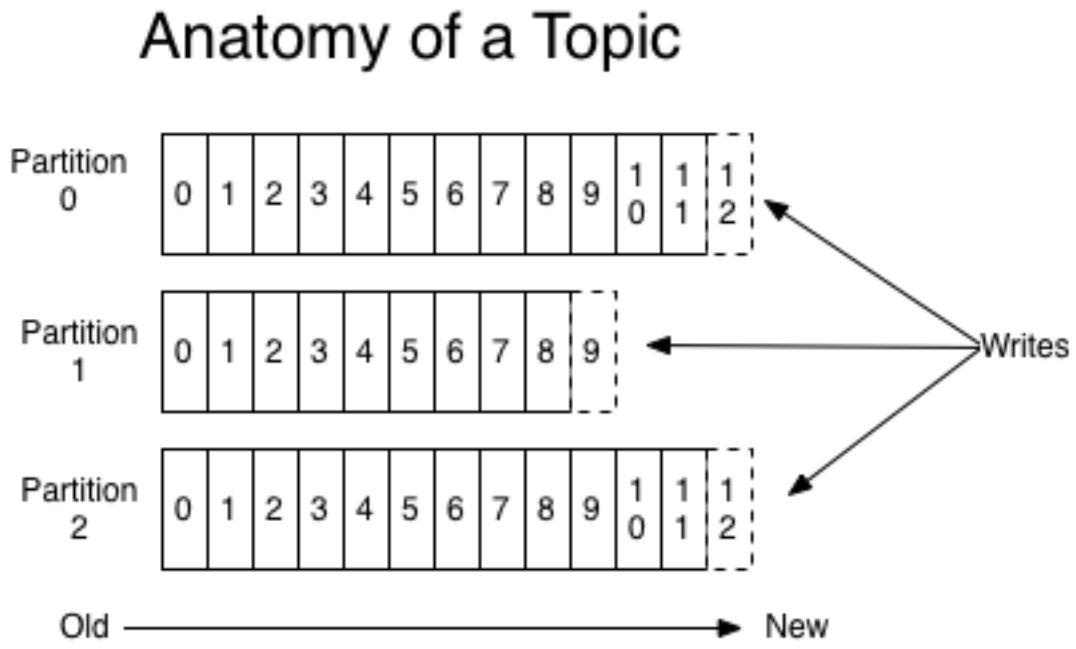
As topics can get quite big, they get split into *partitions* of a smaller size for better performance and scalability. (For example, say you were storing user login requests. You could split them by the first character of the user's username.)



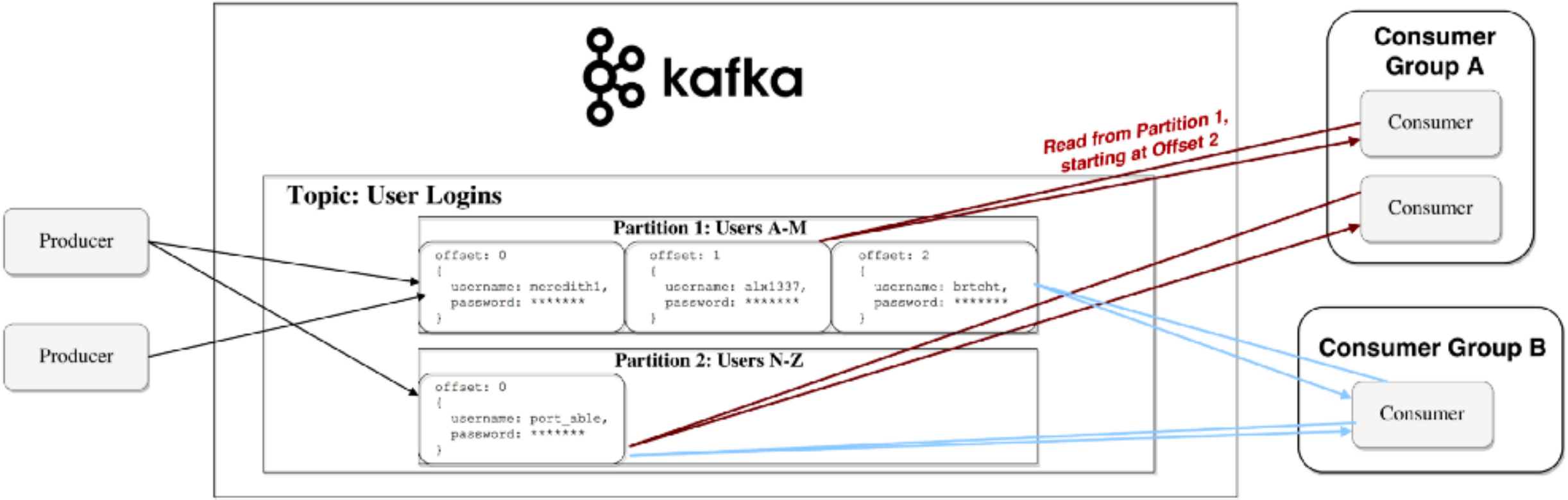
How does it Work?

Kafka guarantees that all messages inside a partition are ordered in the sequence they came in. The way you distinct a specific message is through its *offset,* which you could look at as a normal array index, a sequence number which is incremented for each new message in a partition.

Kafka follows the principle of a dumb broker and smart consumer. This means that Kafka doesn't keep track of what records are read by the consumer, then deleting them. Rather, it stores them for a set amount of time (e.g., one day) or until some size threshold is met. Consumers, themselves, poll Kafka for new messages and say what records they want to read. This allows them to increment/ decrement the offset they're at as they wish, thus being able to replay and reprocess events.



Avoid two processes reading the same message twice



It's worth noting consumers are actually consumer groups that have one or more consumer processes inside. In order to avoid two processes reading the same message twice, each partition is tied to only one consumer process per group.

Persistence to Disk

Kafka actually stores all of its records to disk and doesn't keep anything in RAM. You might be wondering how this is in the slightest way a sane choice. There are numerous optimizations behind this that make it feasible:

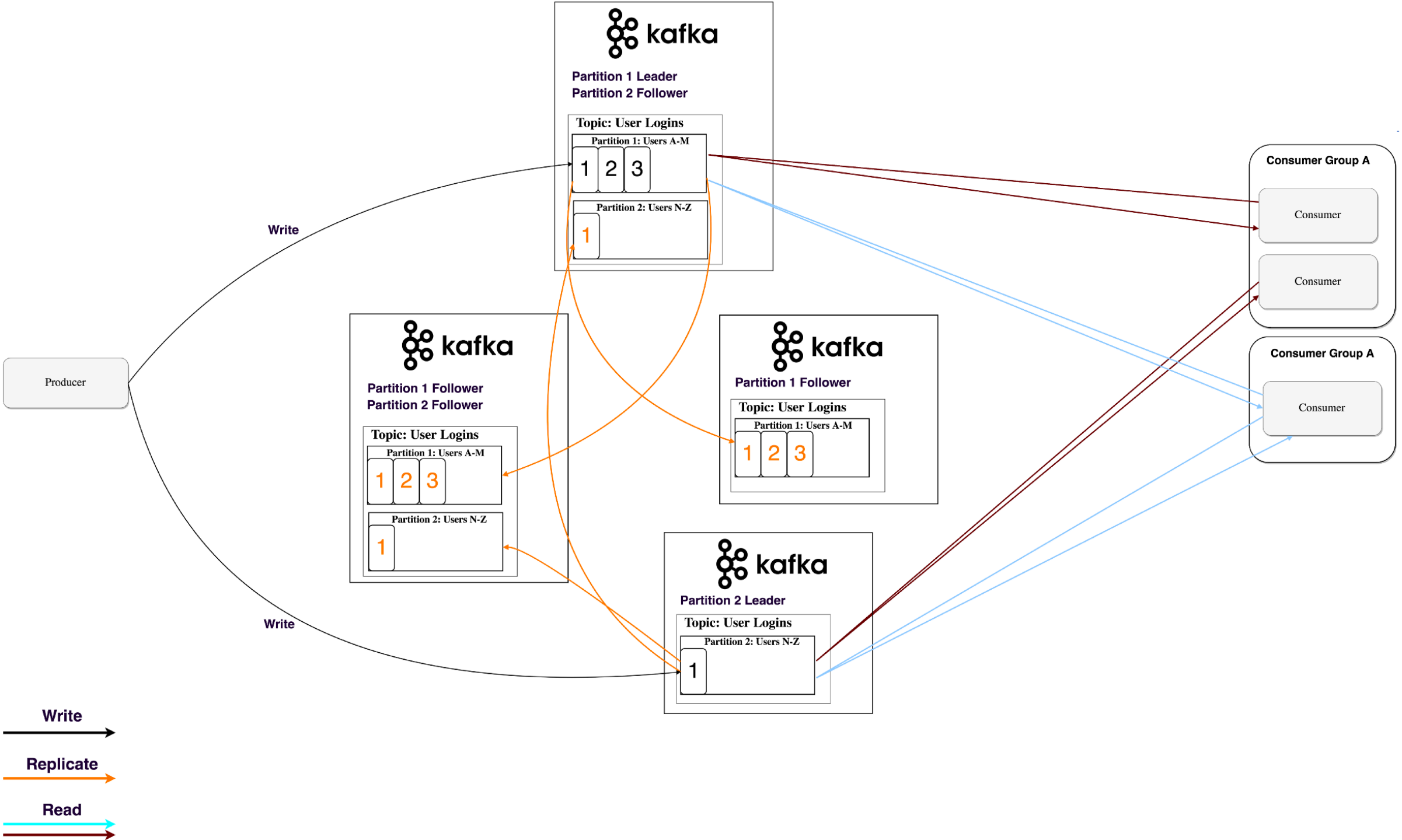
- Kafka has a protocol that groups messages together. This allows network requests to group messages together and reduce network overhead; the server, in turn, persists chunk of messages in one go, and consumers fetch large linear chunks at once.
- Linear reads/writes on a disk are fast. The concept that modern disks are slow is because of numerous disk seeks, something that's not an issue in big linear operations.
- Said linear operations are heavily optimized by the OS, via *read-ahead* (prefetch large block multiples) and *write-behind* (group small logical writes into big physical writes) techniques.
- Modern OSes cache the disk in free RAM. This is called ***pagecache***.
- Since Kafka stores messages in a standardized binary format unmodified throughout the whole flow (producer ➡ broker ➡ consumer), it can make use of the *zero-copy* optimization. That's when the OS copies data from the pagecache directly to a socket, effectively bypassing the Kafka broker application entirely.

Data replication

Partition data is replicated across multiple brokers in order to preserve the data in case one broker dies.

At all times, one broker *owns* a partition and is the node through which applications write/read from the partition. This is called a *partition leader*. It replicates the data it receives to *n* other brokers, called *followers*. They store the data as well and are ready to be elected as leader in case the leader node dies.

This helps you configure the guarantee that any successfully published message won't be lost. Having the option to change the replication factor lets you trade performance for stronger durability guarantees, depending on the criticality of the data.


kafka
Partition 1 Leader
Partition 2 Follower
Topic: User Logins
Partition 1: Users A-M
1 2 3
Partition 2: Users N-Z
1
kafka
Partition 1 Follower
Partition 2 Follower
Topic: User Logins
Partition 1: Users A-M
1 2 3
Partition 2: Users N-Z
1
kafka
Partition 1 Follower
Topic: User Logins
Partition 1: Users A-M
1 2 3
kafka
Partition 2 Leader
Topic: User Logins
Partition 2: Users N-Z
1
Producer
Write
Write
Consumer Group A
Consumer
Consumer
Consumer Group A
Consumer
Write
Replicate
Read

Data replication

In this way, if one leader ever fails, a follower can take his place.

You may be asking, though:

“How does a producer/consumer know who the leader of a partition is?”

For a producer/consumer to write/read from a partition, they need to know its leader, right? This information needs to be available from somewhere.

Kafka stores such metadata in a service called Zookeeper.

What's Zookeeper?

Zookeeper is a distributed key-value store. It's highly optimized for reads, but writes are slower. It's most commonly used to store metadata and handle the mechanics of clustering (heartbeats, distributing updates/configurations, etc).

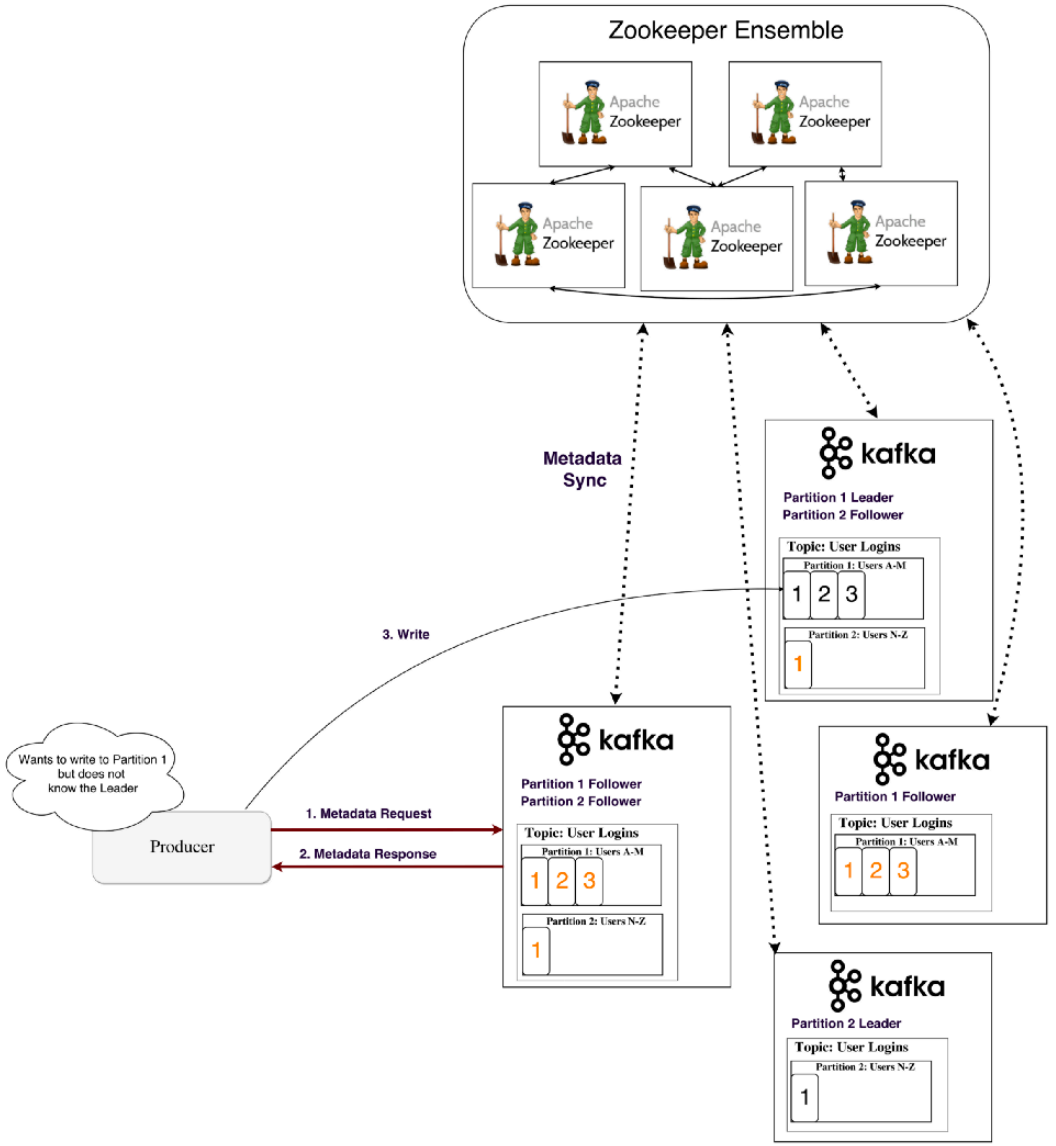
It allows clients of the service (the Kafka brokers) to subscribe and have changes sent to them once they happen. This is how brokers know when to switch partition leaders. Zookeeper is also extremely fault tolerant, and it ought to be, as Kafka heavily depends on it.

It's used for storing all sort of metadata, to mention some:

- Consumer groups offset per partition (although modern clients store offsets in a separate Kafka topic)
- Access control lists (ACLs) — used for limiting access/authorization
- Producer and consumer quotas —maximum message/sec boundaries
- Partition leaders and their health

How does a producer/consumer know who is the leader of a partition

Producers and consumers used to directly connect and talk to Zookeeper to get this (and other) information. Kafka has been moving away from this coupling, and since versions 0.8 and 0.9, respectively, clients fetch metadata information from Kafka brokers directly, who themselves talk to Zookeeper.

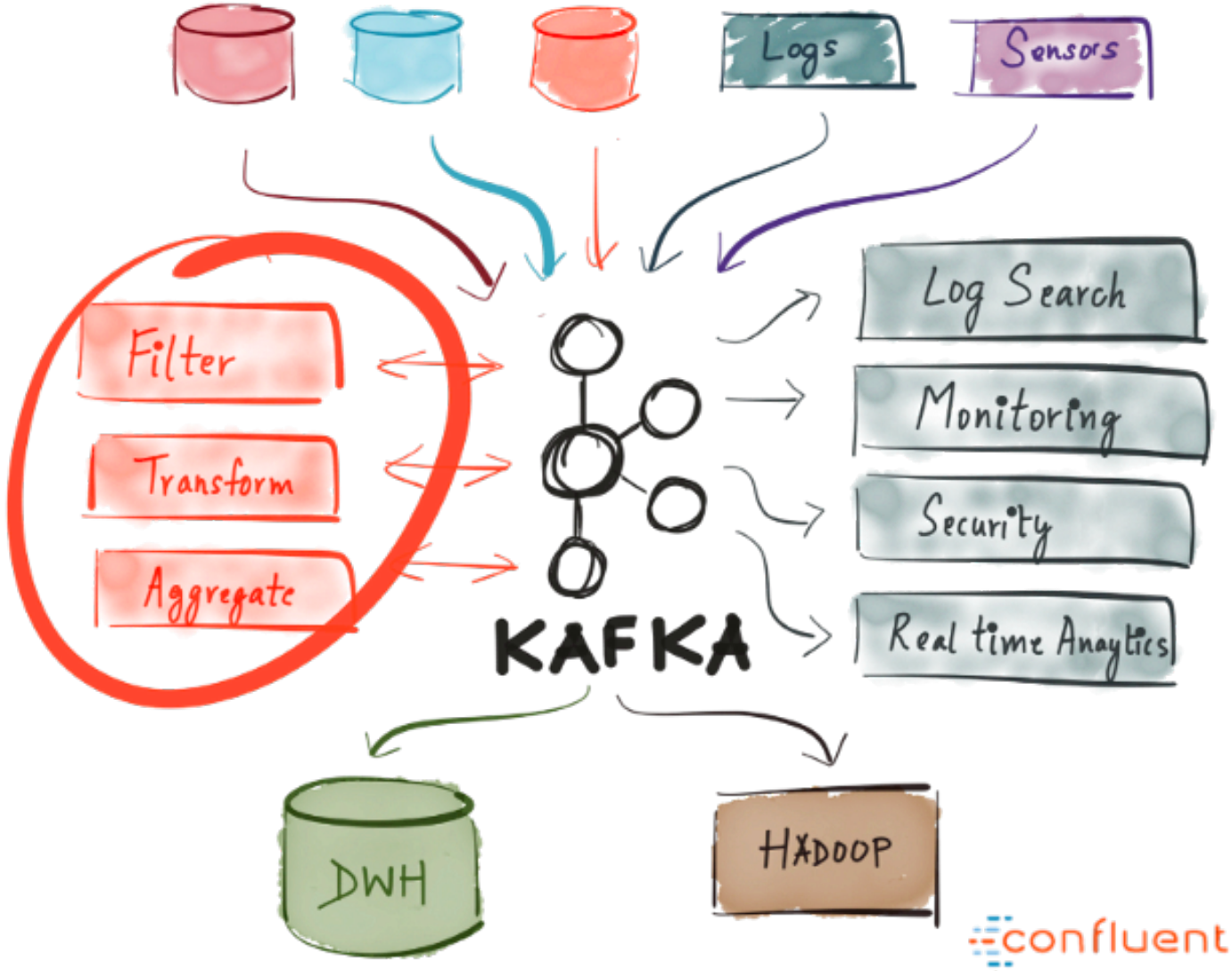


When Would You Use Kafka?

As we already covered, Kafka allows you to have a huge amount of messages go through a centralized medium and to store them without worrying about things like performance or data loss.

This means it's perfect for use as the heart of your system's architecture, acting as a centralized medium that connects different applications. Kafka can be the center piece of an event-driven architecture and allows you to truly decouple applications from one another.

Kafka allows you to easily decouple communication between different (micro)services.



Why Has It Seen So Much Use?

High performance, availability, and scalability alone aren't strong enough reasons for a company to adopt a new technology. There are other systems that boast similar properties, but none have become so widely used. Why is that?

The reason Kafka has grown in popularity (and continues to do so) is because of one key thing — businesses nowadays benefit greatly from event-driven architecture. This is because the world has changed — an enormous (and ever-growing) amount of data is being produced and consumed by many different services (internet of things, machine learning, mobile, microservices).
A single real-time event-broadcasting platform with durable storage is the cleanest way to achieve such an architecture. Imagine what kind of a mess it'd be if streaming data to/from each service used a different technology specifically catered to it.

Why Has It Seen So Much Use?

High performance, availability, and scalability alone aren't strong enough reasons for a company to adopt a new technology. There are other systems that boast similar properties, but none have become so widely used. Why is that?

The reason Kafka has grown in popularity (and continues to do so) is because of one key thing — businesses nowadays benefit greatly from event-driven architecture. This is because the world has changed — an enormous (and ever-growing) amount of data is being produced and consumed by many different services (internet of things, machine learning, mobile, microservices).
A single real-time event-broadcasting platform with durable storage is the cleanest way to achieve such an architecture. Imagine what kind of a mess it'd be if streaming data to/from each service used a different technology specifically catered to it.

Deploy Zookeeper

```
dataDir
server.x
```

sudo vim conf/zoo.cfg
sudo mkdir -p /data/zookeeper
sudo chown -R bigdata:bigdata /data/zookeeper
cd /data/zookeeper && sudo echo x(id) > myid
bin/zkServer.sh start
Jps
    QuorumPeerMain
bin/zkServer.sh status
bin/zkCli.sh -server xxx:2181

```
ls /
[zookeeper]

create /test sample
Created /test

get /test
sample

set /test bar

get /test
bar

delete /test

ls /
[zookeeper]
```

Deploy Kafka

`broker.id` `log.dirs`

`listeners` `zookeeper.connect`

sudo vim config/server.properties
sudo mkdir -p /data/kafka-logs
sudo chown -R bigdata:bigdata /data/kafka-logs
scp config/server.properties bigdata-node2:/opt/bigdata/kafka/config/
scp config/server.properties bigdata-node3:/opt/bigdata/kafka/config/
bin/kafka-server-start.sh config/server.properties &
jps
  Kafka

https://kafka.apache.org/quickstart

Deploy Kafka Test

bin/kafka-topics.sh --create --bootstrap-server bigdata-node1:9092 --replication-factor 3 --partitions 1 --topic repl-test
bin/kafka-topics.sh --create --bootstrap-server bigdata-node1:9092 --replication-factor 3 --partitions 1 --topic bigdata-database
bin/kafka-topics.sh --describe --bootstrap-server bigdata-node1:9092 --topic repl-test
producer
bin/kafka-console-producer.sh --bootstrap-server bigdata-node2:9092 --topic repl-test-topic
consumer
bin/kafka-console-consumer.sh --bootstrap-server bigdata-node2:9092 --from-beginning --topic repl-test-topic

test fault-tolerance
kill bigdata-node2 kafka, it will auto switch to other node.

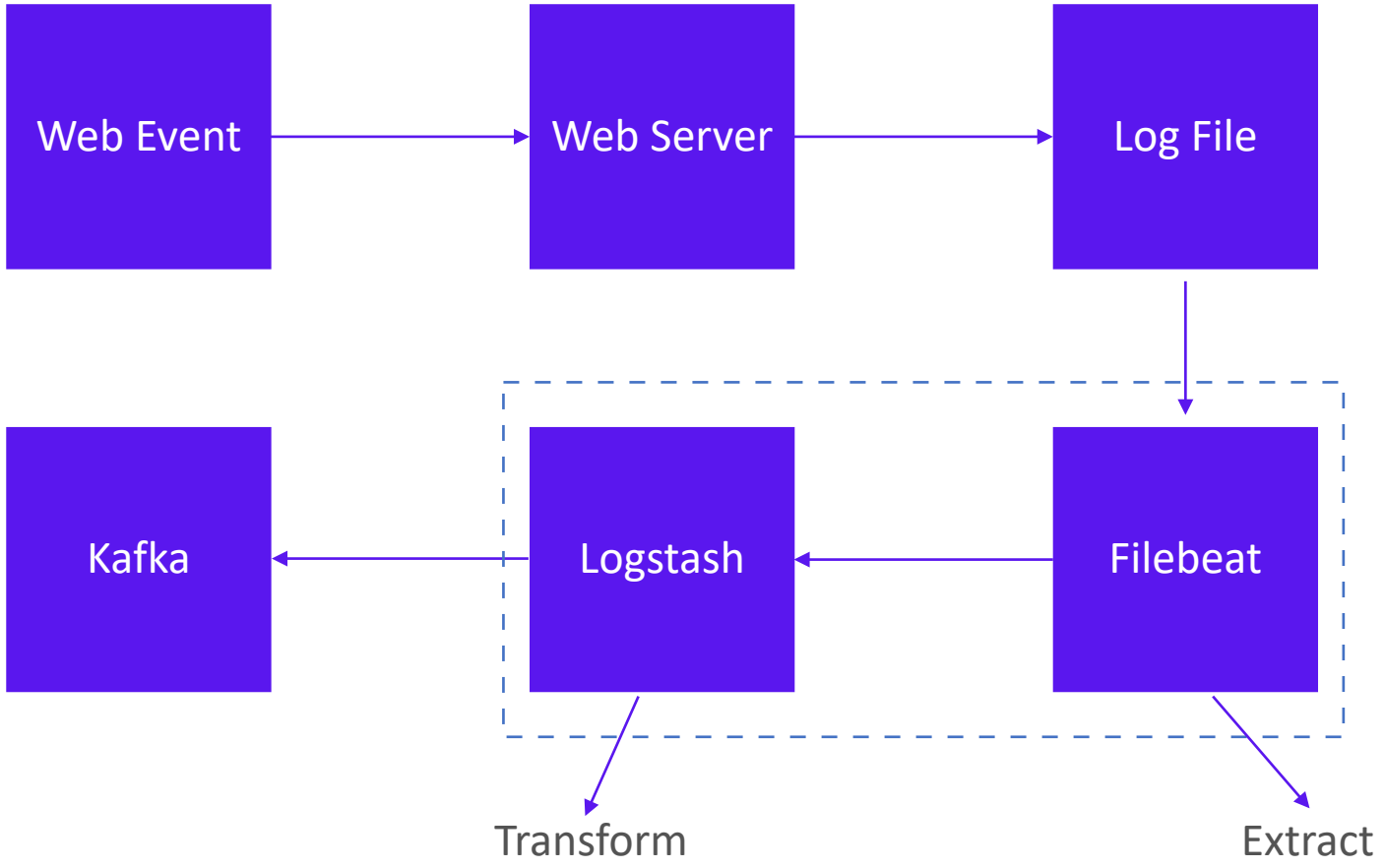
https://kafka.apache.org/quickstart

Deploy Kafka Manager

vim conf/application.conf
bin/cmak
http://192.168.56.101:9000/

How to Process Log Data?

How to Capture Change of Log - Web Event



How to Capture Change of Log - Web Event


```json
[
  {
    "header": {
      "name": "EcoScope Data",
      "well": "35/12-6S",
      "field": "Fram",
      "date": "2020-06-14",
      "operator": "Logtek Petroleum",
      "startIndex": 2907.79,
      "endIndex": 2907.84,
      "step": 0.01
    },
    "curves": [
      {
        "name": "MD",
        "description": "Measured depth",
        "quantity": "length",
        "unit": "m",
        "valueType": "float",
        "dimensions": 1
      },
      {
        "name": "A40H",
        "description": "Attenuation resistivity 40 inch",
        "quantity": "electrical resistivity",
        "unit": "ohm.m",
        "valueType": "float",
        "dimensions": 1
      }
    ],
    "data": [
      [2907.79, 29.955],
      [2907.80, 28.892],
      [2907.81, 27.868],
      [2907.82, 31.451],
      [2907.83, 28.080],
      [2907.84, 27.733]
    ]
  }
]
```


https://www.graylog.org/post/log-formats-a-complete-guide

https://jsonwelllogformat.org/

https://github.com/madzak/python-json-logger

How to Capture Change of Log - Log Extract

轻量型

从源头采集。简单明了。

Beats 是数据采集的得力工具。将 Beats 和您的容器一起置于服务器上，或者将 Beats 作为功能加以部署，然后便可在 Elasticsearch 中集中处理数据。Beats 能够采集符合 Elastic Common Schema (ECS) 要求的数据，如果您希望拥有更加强大的处理能力，Beats 能够将数据转发至 Logstash 进行转换和解析。

https://www.elastic.co/cn/beats/

https://www.elastic.co/cn/beats/filebeat

Deploy Filebeat and Logstash - Filebeat

vim filebeat.yml
./filebeat modules list
./filebeat modules enable system
./filebeat setup -e
./filebeat -e

consumer
bin/kafka-console-consumer.sh --bootstrap-server bigdata-node2:9092 --from-beginning --topic repl-test-topic

https://www.elastic.co/guide/en/beats/filebeat/7.10/filebeat-installation-configuration.html

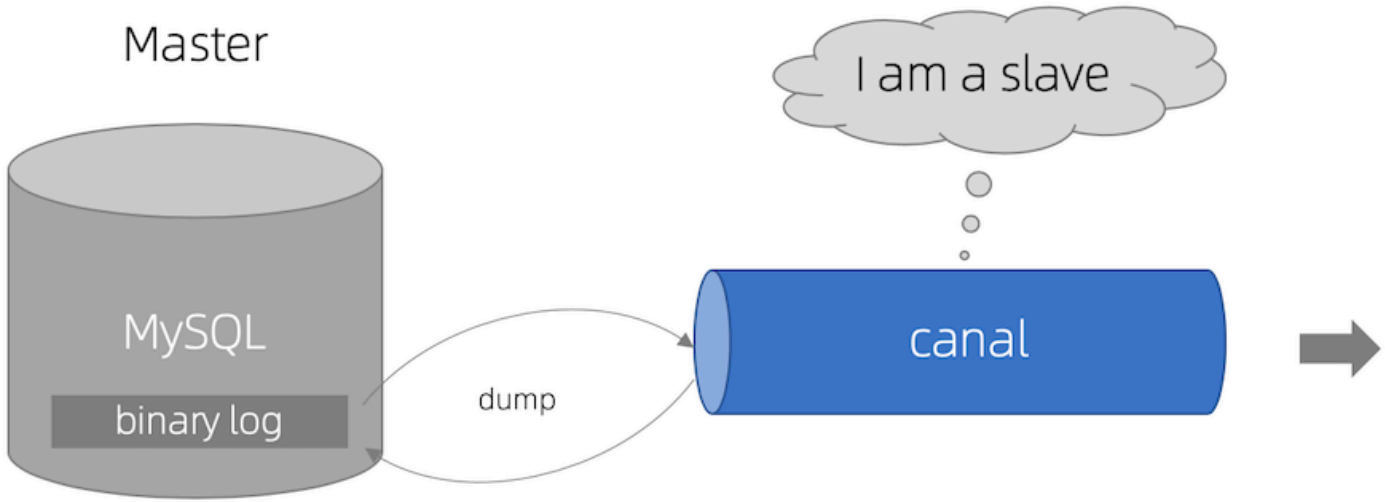
How to Capture Change of Log - Log Extract & Transform

	Flume	Logstash	Filebeat
内存	大	大	小
cpu		大	小
背压敏感协议	否	否	是
插件	需要些API	多	多
功能	从多种输入端采集数据并输出到多种输出端	从多种输入端采集并实时解析和转换数据并输出到多种输出端	传输
轻重	相对较重	相对较重	轻量级二进制文件
过滤能力	自带了分区和拦截器功能	强大的过滤能力	有过滤能力但是弱
进程	一台服务器可以有多个进程，挂掉之后需要手动拉起	一台服务器只允许一个logstash进程,挂掉之后需要手动拉起	十分稳定
原理	当source接收到Event时，它将其存储到一个或多个channel中。channel是一个被动存储，它将事件保持到被Flume消耗为止。接收器将事件从channel中移除，并将其放入外部存储库（如HDFS）或将其转发到流中下一个Flume代理的source。给代理内的source和sink与在通道中分段的事件异步运行。	Logstash使用管道的方式进行日志的搜集和输出,分为输入input --> 处理filter（不是必须的） --> 输出output,每个阶段都有不同的替代方式	开启进程后会启动一个或多个探测器（prospectors）去检测指定的日志目录或文件，对于探测器找出的每一个日志文件，filebeat启动收割进程（harvester），每一个收割进程读取一个日志文件的新内容，并发送这些新的日志数据到处理程序（spooler），处理程序会集合这些事件，最后filebeat会发送集合的数据到你指定的地点。
编写语言	Java	Jruby	go语言
集群	分布式	单节点	单节点
输出到多个接收方	支持	支持	6.0之前支持
二次开发或者扩展开发	一般	难	易

How to Process Database Data?

How to Capture Change of Database

Canal



MySQL

APACHE kafka
A distributed streaming platform

elasticsearch

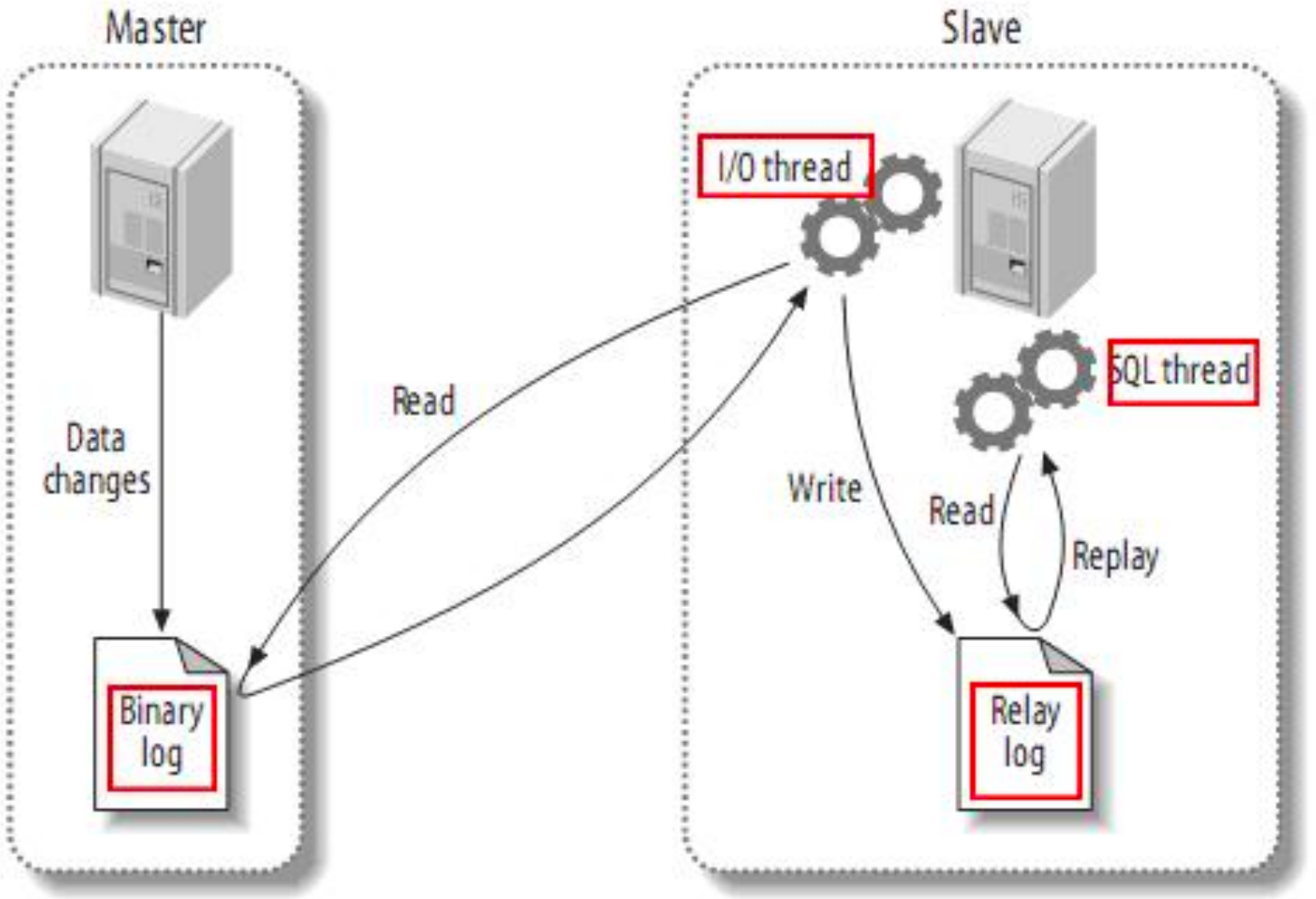
APACHE HBASE

RocketMQ

PULSAR

...

Mysql master to slave



https://github.com/alibaba/canal/wiki/Canal-Kafka-RocketMQ-QuickStart

Deploy Mysql

```
sudo apt install mysql-server
sudo mysql_secure_installation
sudo mysql
CREATE USER canal IDENTIFIED BY 'canal';
GRANT SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO
'canal'@'%';
-- GRANT ALL PRIVILEGES ON *.* TO 'canal'@'%' ;
grant all privileges on *.* to 'canal'@'%' with grant option;
ALTER USER 'canal'@'%' IDENTIFIED BY 'canal' PASSWORD EXPIRE
NEVER;
ALTER USER 'canal'@'%' IDENTIFIED WITH mysql_native_password BY
'canal';
FLUSH PRIVILEGES;
vim /etc/mysql/mysql.conf.d/mysqld.cnf
systemctl restart mysql.service
systemctl status mysql.service
```

Deploy Canal

vim conf/example/instance.properties
vim conf/canal.properties
vim bin/startup.sh
bash bin/startup.sh/bash bin/stop.sh

bin/kafka-console-consumer.sh --bootstrap-server bigdata-node3:9092 --topic bigdata.test —from-beginning

https://github.com/alibaba/canal/wiki/Canal-Kafka-RocketMQ-QuickStart

Assignment

`Setup ETL environment and take screenshot`

1、减分项：抄袭作业；原封不动提交老师代码；作业提交超时。
2、加分项：代码有注释；代码运行结果正确；代码格式规范；在作业的基础上有对比实验或者自己总结的项目心得。
3、满分作业要求：
a.提交了非抄袭代码。
b.符合讲师的作业目标要求。
c.代码有合理的注释。
d.代码书写整洁规范、代码逻辑严谨。
e.有模型实验结果的对比总结或者从项目中总结的心得体会。
f.代码能够运行成功。