Introduction to Kafka

=> Apache Spark
Apache Airflow
Apache Kafka

=> What is Apache Kafka?

⊙ Kafka is open-source,
distributed,
event-streaming platform.

⊙ Kafka is designed to handle large quantities of data, & to be scalable to various situations.

⊙ Even-streaming varies based on contex.

↓

in Kafka, it means obtaining information from various sources, reliably storing that data, & providing it to any users or systems needing access.

⊙ Kafka has many uses & can fit into many different scenarios.

1> Ecommerce :- allowing different systems to monitor sales data in up to real-time timeframes.

2> Kafka works well with various types of order tracking, maintaining the status of various orders, such as new orders, processing & packing, & shipment & delivery

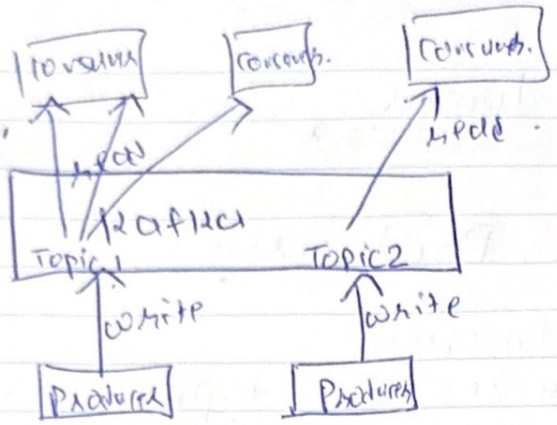
3) building on order tracking, kafka is often used by ride-share or food delivery services. adding geo-graphical location information & providing live status update (info) for all drivers, riders & customers.

4) kafka works well for sensor data networks, such as closely monitoring temperature data in a warehouse or various information from the safety systems in a car (or even a roller coaster).

5) Use for cyber security purposes, like spam tracking, patch monitoring & more.

⇒ Components of kafka :-

- Topics :- A topic is a common message type stored within kafka. (eg:- like notepad where kafka stores the events or messages it receives. There can be any # of topics in a kafka system)
- kafka Producers :- (writers) which write events to various topics. Producers can write to single or multiple topics.
- kafka Consumer :- (readers) which reads information from topics. There can be any # of consumers.



- `bin/kafka-topics.sh --help` → will output → give all the options available on the commands.

Options → ⋮

description → ⋮

- You want to see a list of topics stored within kafka server. Then, use command,

```
bin/kafka-topics.sh --bootstrap-server
    localhost:9092 --list.
```

→ give output

cleared - devices
compromised - systems.
know - exploits
phising - sites.

⇒ Kafka Producers :-

⊙ What is a Producer?

→ refer to publisher
→ Write messages to topics (kafka topics)
→ Messages sent via producers stored on kafka for later use. (goal is to retain data on kafka ASAP.)
→ Many producer can write on kafka at a time & each producer can write on single or multiple topics as required.

⊙ Types of Producers :-

→ Many types of producers are availble, with varying features depending on needs.
→ Most basic producer is command-line producer.
→ This is included in kafka-software installation & is called `kafka-console-producer.sh` ↓ primary producer.
→ for command-line tool API libraries are available for python, Java & many other languages.
→ Besides API libraries & kafka command line tools, there is another option called kafka-connect.
  ↳ is designed to interact within existing systems, such as Relational databases & automatic transfer data between them.

1. kafka-console-producer.sh (tool)

→ found in `bin/` folder, with other kafka tools.

→ `bin/kafka-console-producer.sh`

→ Several command-line options available, including some required, others are not. This means they must always be present when using this command.

→ Required options:

(1) --bootstrap-server ← specifies which server to use. (kafka-system)

`--bootstrap-server localhost:9092` for communication with & defaults to localhost-9092.

(2) --topic ← which requires the topic to write to, such as topic phising-sites.

`--topic phising-sites`

→ By default, this opens a connection where you can type messages.

In this case, you can write any messages & then use `Ctrl+C` to exit the tool.

→ You can also use the pipe (|) symbol to pass any message data directly into the kafka-console-producer tool.

(.) Examples:-

(1) → Interactive examples

```
bin/kafka-console-producer.sh
        --bootstrap-server localhost:9092
                --topic testing.
```

↓

When this command will run we can enter the messages to console & use Ctrl+C to exit.

```
>This is the 1st message.
> "   "   "  2nd  ".
> "   "   "  3rd  ".
(Ctrl+C to exit).
```

→ Interactive means program waits for our input before continuing.

(2) → Non - Interactive examples

- Pretty similar to interactive eg. but using echo command.

```
echo "This is the first message" |
bin/kafka-console-producer.sh
    --bootstrap-server localhost:9092
    --topic testing.
```

↓

to write our message & pass it directly to the kafka-console-producer. So, this would automatically write the message to kafka without having to use (Ctrl+C) to exit.

- Producers are primary component of kafka.

⇒ Kafka consumer :-

→ Reads message from kafka topics.
→ Sometimes called subscribers
→ consumer read messages stored within kafka.topics.
→ can have many consumers.
→ Consumers can read from one or multiple topics.
→ Many different kind of consumers.
→ most popular is command-line tool.

`kafka-console-consumer.sh`

→ there are API library available for python, java, & many other languages.
→ There are also component within kafka connect that allow users to read data from kafka topics.
→ `kafka-console-consumer.sh tool.`

↓

normally, found in `bin/` folder, with other kafka tools.

→ Many option for consumer, but most required ones are:
  --topic
  --bootstrap-servers.

→ By default, opens a connection where messages appear (real-time) & maintain position.
→ Use `ctl+C` for exit.

⇒ Optional Arguments :-

- -- from-beginning
  - Tells kafka system to send all messages in topic, even ones that may have been already read.
- --max-messages <number>
  - Used to read up to a maximum number of messages before exiting.

⇒ Interactive example :-

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

↓

can see all messages
ctrl+c to exit.

⇒ Non-interactive example :-

```
bin/kafka-console-consumer.sh --bootstrap-server
localhost:9092 / --topic testing --from-beginning
--max-messages 2
```

beginning

↓

return first 2 events & then exit automatically.

-: Kafka - Architecture :-

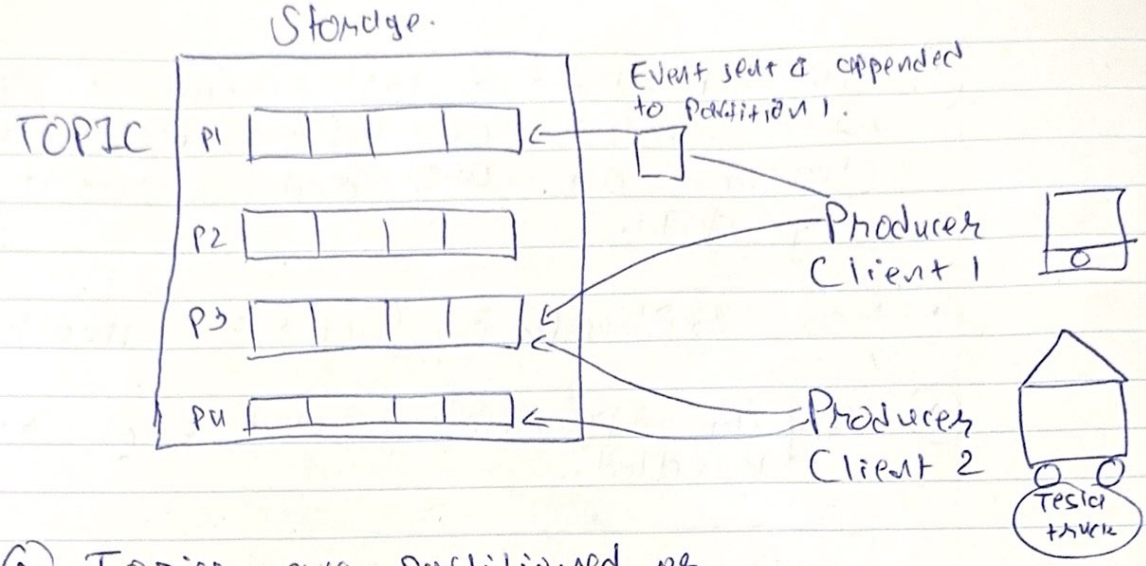
⇒ Major component of kafka → Servers, Clients.

servers:
- In kafka, server is a cluster of one or more computers handle storing data & manage communications with kafka clients.
- It can also handle integration with other systems including databases, log files, & so on.

Clients:
- Kafka clients, which read data via kafka consumers or write data via producers.
- Kafka clients can also process data locally and then store that information elsewhere or back to kafka itself.

⇒ Kafka Server :-

- Acting as a kafka broker.
- Storage, also known as the kafka broker
  - server also handles data storage.
  - handles communication via consumers & producers.
- Data written by producers is stored & organized via topics.

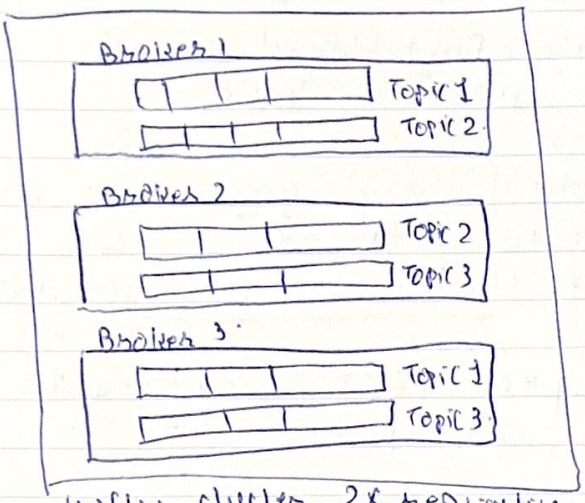


- Topics are partitioned, or stored in seperate topics.
- The individual messages are stored in a given partition based on an event id, such as a customerid or locationid.
- Each messages are retrieved in same order as written.

⟹ Partitions & replication :-

- Kafka is a fault-tolerant system, meaning that if a system within cluster goes offline, the other can provide the data.
- If one system goes offline, others can provide the requested data.
- For each kafka cluster, there is a replication factor. → [Max # of failures - 1 = replication factor]

- The replication factor minus 1 equals the ~~clust~~ number of system failures the cluster can withstand without loosing any data.
- Max replication factor == number of servers
- Copying partitions are how replication is handled.



Kafka cluster, 2x replication.

consider an example cluster with 3 brokers.

3 topics defined & set with 2x replication.

2 copy of each partition within the cluster.

2x replication == can lose 1 system. before data failure.

if Broker 2 fails

↓

replication factor 2x
2 - 1 == can handle 1 failed system.

we are ok, because each topic is exists even Broker 2 fails within cluster.

if Broker 1, Broker 2 fails

↓

More failure than supported (1 system).

Topic 1 & 3 are available, so not a complete failure

Topic 2 is no longer available in system.

⇒ `/home/repl/check_broker.sh.` — find a # of broker in the cluster.

(*) Creating & Managing Kafka Cluster:

⇒ It can possible by <u>Apache Zookeeper.</u>

⇒ What is <u>Zookeeper ?</u>

- Zookeeper is a framework to manage information & provide services necessary for running distributed systems.
- Zookeeper is primary used by developers to create distributed applications, but users do interact with it to manage those applications.
- eg. Of applications using Zookeeper include
  1. Kafka.
  2. Hadoop → distributed processing framework.
  3. Neo4j → Graph db.

⇒ What zookeeper actually does?

- It provides various services necessary to run distributed applications. This includes handling any configuration information & the naming of systems to prevent conflicts

- It also provides the ability to synchronize across systems, such as determining what systems are available, when they should start, how services can reach them, & so forth.

- Zookeeper also provides any other basic service that would be required for a group of systems to communicate.

⇒ Zookeeper is designed as a framework to prevent individual distributed applications from implementing custom versions of services.

for eg:- how power plug or water hose nozzle implements common standards vs each having its own.

This allow for easier configuration, implementation & interaction.

⇒ So, kafka use Zookeeper for cluster management

- Newer cluster management tool available called **KRaft**.

- Kafka uses two primary configuration files (two files) for server / cluster.

config / zookeeper.properties.

config / server.properties

```
# zookeeper.properties

# The directory where the
# snapshot is stored.
dataDir = /tmp/zookeeper

# The port at which the
# clients will connect
clientPort = 2181

.....
```

handles information needed for a basic zookeeper settings setup including where to store zookeeper data & what network port to run on.

Portion of zookeeper.properties.

⟹ The primary kafka server configuration file is the config/server.properties file.

- This contains considerably more information than the zookeeper.properties file & defines information specific to kafka installation.
- Includes info regarding kafka brokers details, any network configuration, where to store events, a basic topic configuration including any replication details.

```
....
# A comma seperated list of directories
under which to store log files.
logs.dirs = /tmp/kafka-logs.

# The default number of log partitions per topic.
num.partitions = 1.
```

⇒ Kafka server can be started in two steps.

(*) Starting a Kafka cluster:-

→ 1) using command

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

This starts up the basic zookeeper server & the configuration details found in the zookeeper.properties file.

There's an extensive amount of output that will vary based on your system.

→ 2)

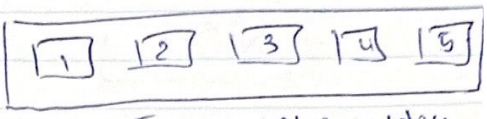
```
bin/kafka-server-start.sh config/server.properties
```

This will start Kafka services as defined in the server.properties file.

→ To stop a Kafka cluster, we do the reverse using kafka-server-stop.sh & zookeeper-server-stop.sh

NOTE:- because Kafka must cleanly shutdown any open connections, & because it uses zookeeper services, the shutdown order is reversed.

Kafka topics :-



E-commerce orders

:- Introduction to Apache Kafka :-

⇒ Create & manage kafka topics.

- Kafka topics are a logical grouping of events, where each event or message is referring to the same type of thing, such as orders, downloads, virus detections, etc.
- Topic is similar to a table in a relational database, where each event has a similar structure and meaning.
- Topic sometimes refer to as "Event log." & messages written to the log are immutable. (means messages in a topic can be read or created but can't modified.).
- Messages can be removed based on age, but otherwise, it is available to be read at any point in the future, given enough storage.

⇒ multiple ways to create a topics in a kafka, but we'll primarily use the bin/kafka-topics.sh script.

(Create new topic on kafka server.)

```
bin/kafka-topics.sh --bootstrap-server <server>
--topic <topicname> --create.
```

NOTE:- Still we need to add the --bootstrap-server option.

eg:- `$ bin/kafka-topics.sh --bootstrap-server localhost:9092 \ --topic Orders --create`

↓

System will return text "Created topic Orders."

⇒ Other Variations :-

- Optional arguments for `bin/kafka-topics.sh --create`

`--replication-factor <x>`

↓

Define the replication factor of the topic.

↓

How many copies of a topic exist within a kafka cluster.

`--partitions <x>`

↓

specifies the numbers of partitions to split the topic into.

→ This is a performance optimization but allows kafka to handle higher throughput.

for eg:-

```
$ bin/ kafka-topics.sh --bootstrap-server
  localhost:9092 --topic orders --create
  --replication-factor 3 --partitions 3
```

↓ created topic orders.

⇒ `--describe` → gives us details about the configurations of the kafka topics.

argument in ↓ kafka-topics.sh

```
bin/kafka-topics.sh --bootstrap-server <servers>
--topic <topicname> --describe
```

for eg:-

```
$ bin/kafka-topics.sh --bootstrap-server localhost:
9092 --topic orders --describe.
```

↓

Output:

```
Topic: Orders-new    TopicId: <topicid>
PartitionCount: 3    ReplicationFactor: 1

Configs:
  Topic: Orders-new  Partition: 0  Leader: 0  Replicas: 0 Isr:
  Topic:  "          "  : 1       "          "          "
  Topic:  "          "  : 2       "          "          "
```

Hilroy

⟹ Removing topics :-

```
$ bin/kafka-topics.sh --bootstrap-server
<server> --topic <topicname> --delete.
```

↓

- will remove the topic from a kafka server/cluster.
- Removing topic also removes all messages in the topic.

for eg:-

```
$ bin/kafka-topics.sh --bootstrap-server
localhost:9092 --topic Orders --delete
```

↓

simply returns to prompt.

✡ Kafka - troubleshooting :-

→ One common issue you'll run into is that kafka tries to be helpful to users in its default configuration. Unfortunately, it sometimes causes problems instead.

sounds confusing but it may lead to significant issue. →

One such issue is that, by default, a topic doesn't need to exist before a producer writes to it.

Consider what happens if producer misspelled a topic name, such as ordrs instead orders.

→ The events would be written to the wrong topic, & consumers would not see these events.

→ list of topics

```
$ bin/kafka-topics.sh --bootstrap.sh localhost:9092 --list.
```

Orders.

```
$ echo "Test message" | bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic
Ordrs ← mispelled here.
```

```
$ bin/kafka-topics.sh --bootstrap.sh localhost:9092 --list.
```

↓

Orders } two topics.
Ordrs.

⟹ Connectivity Problems:-

- Kafka is a networked service, even on a single-node installation.
- Any networking issue can cause problems with Kafka communications.
- Best way to identify these problems is to look at the output of commands to determine the potential issue

let's consider trying to get a list of topics from an instance using the kafka-topics.sh script.

```
$ bin/kafka-topics.sh --bootstrap-server
localhost:9092 --list
```

but error like (output) ↓

```
WARN [AdminClient clientId=adminclient-1
Connection to node -1 ( ) could not be
established.
Node may not be available.
```

(Connection not may be established.)

=> To mitigate it, The first is to check that the service is running using. something like ps & check if the kafka process is present.

tlnp.

```
(PS ax | grep kafka, netstat -tlnp | grep 9092)
```

↓ lists for processes running on a system.

↓ provide list of networking connection.

grep is basic search text tool.

=> We can also check for any firewall issues. & verify that we have the correct IP port combo for our kafka installation.

① Verify correct port / IP.

⇒ Another Common Problem :-

- Consumers :-
  - Use `--from-beginning` if you need older messages.

    Otherwise, only new messages in the topic, since the consumer started, will be received.
  - Remember `--max-messages` to only read a specific messages quantity, otherwise the consumer script will never exit.

- All tools :-
  - Remember to include `--bootstrap-server` options.

    ↓

    This is initial kafka server that all programs communicate with to run kafka commands.
  - Most error messages are pretty clear & will point you to likely possible solutions.
  - Tools also have a `--help` option to get further details when using these scripts.