Kafka and Zookeeper

Zookeeper:
ZooKeeper is a high-performance coordination service for distributed applications.
Download link: https://www.apache.org/dyn/closer.cgi/zookeeper/

Standalone Operation Minimal Configuration

After downloading just unpack the *.jar file.
Enter the unpack folder as shown below.
> cd zookeeper-version

Edit **zoo.cfg** in the conf folder as shown below.
`tickTime=2000 →` It is used to do heartbeats and the minimum session timeout will be twice the tickTime.

`dataDir=/var/lib/zookeeper →` the location to store the in-memory database snapshots.
`clientPort=2181 →` the port to listen for client connections

Starting Zookeeper server
While you are in the zookeeper root directory you start it as shown below.
> ./bin/zkServer.sh start

After starting zookeeper we can connect using client to perform file like operations as shown below.
> ./bin/zkCli.sh -server 127.0.0.1:2181
> [zkshell: 0] help → for getting list of commands.

Creating node and cluster configuration will be done later.

Kafka:

Apache Kafka is *a distributed streaming platform*

A streaming platform has three key capabilities:

- Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system.
- Store streams of records in a fault-tolerant durable way.
- Process streams of records as they occur.

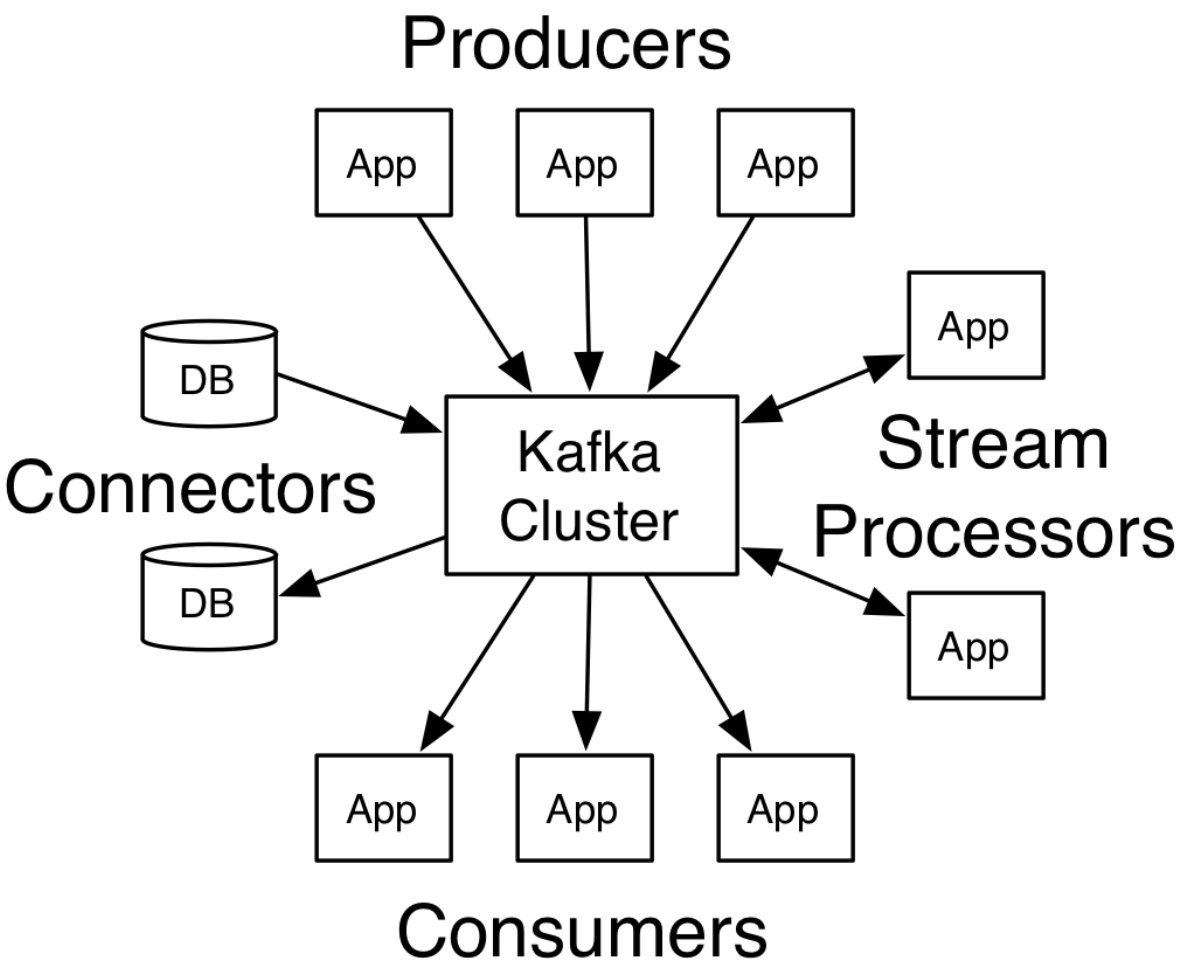
Kafka is generally used for two broad classes of applications:

- Building real-time streaming data pipelines that reliably get data between systems or applications

- Building real-time streaming applications that transform or react to the streams of data

To understand how Kafka does these things, let's dive in and explore Kafka's capabilities from the bottom up.
First a few concepts:

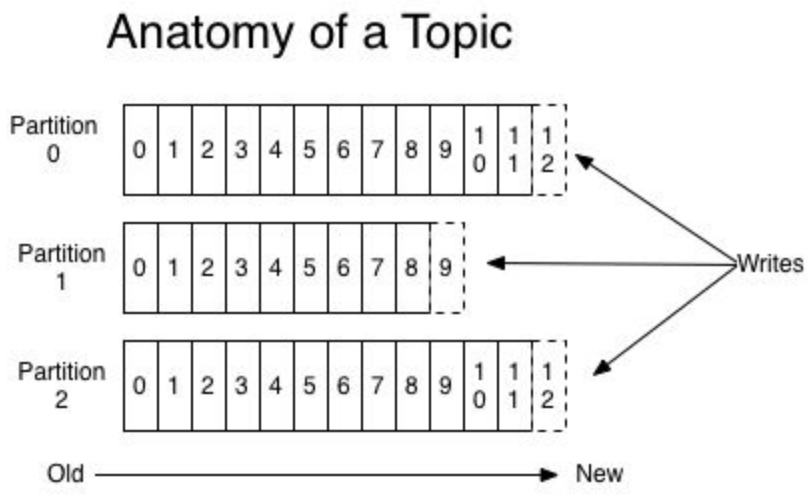
- Kafka is run as a cluster on one or more servers that can span multiple datacenters.
- The Kafka cluster stores streams of *records* in categories called *topics*.
- Each record consists of a key, a value, and a timestamp.



Topics and Logs

A topic is a category or feed name to which records are published. Topics in Kafka are always multi-subscriber; that is, a topic can have zero, one, or many consumers that subscribe to the data written to it.

For each topic, the Kafka cluster maintains a partitioned log that looks like this:



The Kafka cluster durably persists all published records—whether or not they have been consumed—using a configurable retention period.

Distribution

Start the server

After downloading and unpacking the kafka, you can start it using the following command shown below.
> cd kafka_version

Then execute the command below to start kafka server.
> ./bin/kafka-server-start.sh

Create a topic

The command below is use to create a topic while you are in kafka root directory.
> ./bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test

With the below command we can then list the topics in our server.
> ./bin/kafka-topics.sh --list --zookeeper localhost:2181

Send some messages (creating producer to send message)

Kafka comes with a command line client that will take input from a file or from standard input and send it out as messages to the Kafka cluster. By default, each line will be sent as a separate message.

Using kafka producer and consumer console clients we can easily test by sending messages using the producer client and consuming them using the consumer client. With the below commands we can start console producer and send message to our test topic that we created above.

> ./bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
> This is a message
This is another message

Start a consumer (creating consumer to consume messages sent by the producer above)

The below command can be used to start console consumer client while you in kafka root directory to consume the message sent by the producer above.

> ./bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning.
When the client start you will see the message sent to test topic printed in the console.