Kafka: All you need to know

Apache Kafka's popularity is exploding. Learn about what it is, and why it's becoming a solution of big data and microservices applications

Maria Valcam Follow
May 23 · 7 min read ★

APACHE kafka®
A distributed streaming platform

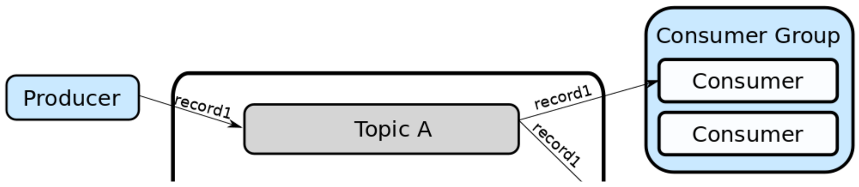
What is Kafka

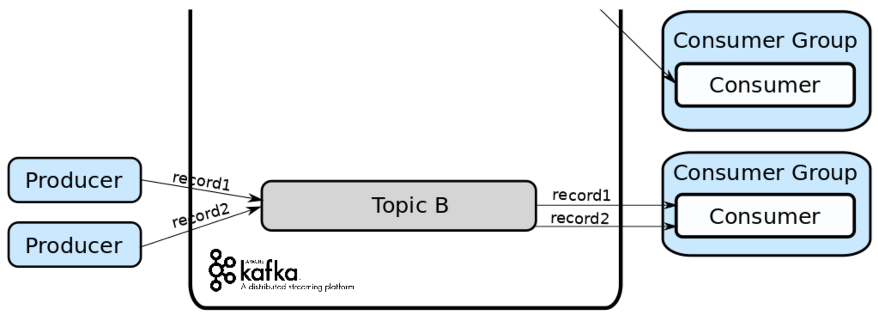
Kafka is a Publish/Subscribe messaging system. It allows **producers** to write ***records*** into Kafka that can be read by one or more **consumers**. These records that cannot be deleted or modify once they are sent to Kafka (this is known as "distributed commit log").

Records are published into **topics**. Topics are like channels where all the records are stored. Consumers subscribe to one or more topics to read its records.

Kafka has a retention period, so it will store your records for the time or size you specify and can be set by topic.

Consumers label themselves with a **consumer group** name. In a consumer group, one or more consumers work together to consume a topic. When a new record arrives to the topic, it will be sent just to one consumer instance in the consumer group.





Architecture

Kafka is distributed as in the sense that it stores, receives and sends records on different nodes (called **brokers**). This makes it easy to scale it horizontally and makes it fault-tolerant. Brokers receive records from producers, assigns offsets to them, and commits them to storage on disk.

To run Kafka, you need Zookeeper. Zookeeper is used for:

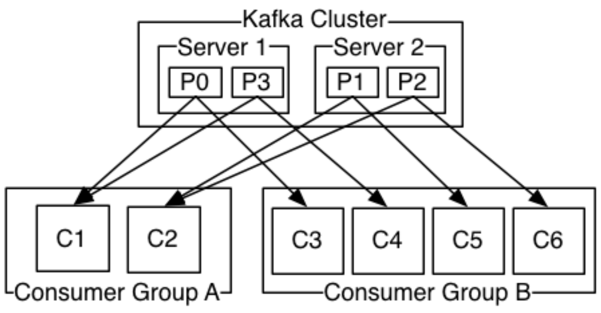
- **Controller election**. The controller is one of the brokers and is responsible for maintaining the leader/follower relationship for all the partitions. When a node shuts down, it is the controller that tells other replicas to become partition leaders to replace the partition leaders on the node that is going away. Zookeeper is used to elect a controller, make sure there is only one and elect a new one it if it crashes.
- **Cluster membership.** Zookeeper maintains a list of all the brokers that are functioning and are a part of the cluster at any given moment.
- **Topic configuration** — which topics exist, how many partitions each has, where are the replicas, who is the preferred leader, what configuration overrides are set for each topic
- **Quotas** . How much data is each client allowed to read and write
- **Access Control Lists (ACL)** . Who is allowed to read and write to which topic (old high level consumer) — Which consumer groups exist, who are their members and what is the latest offset each group got from each partition.

Deep Dive

As topics can get quite big, they get split into **partitions** (this improves performance and scalability). So a record will be published in a single partition of a topic. Producers can choose the partition in which a record will be sent to, otherwise, the partition is chosen by Kafka.

A partition is owned by a single broker in the cluster, and that broker is called the leader of the partition. A partition may be assigned to multiple brokers, which will result in the partition being replicated. This provides redundancy of records in the partition, such that another broker can take over leadership if there is a broker failure.

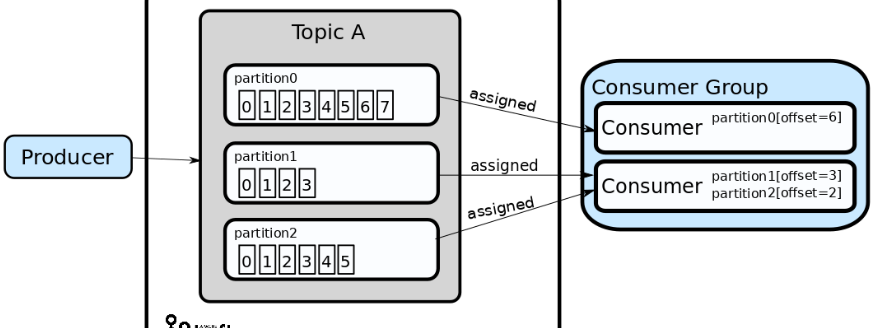
About consumer groups, In order to avoid two consumer instances within the same consumer groups reading the same record twice, each partition is tied to only one consumer process per consumer group.



Kafka follows the principle of a dumb broker and smart consumer. This means that Kafka does not keep track of what records are read by the consumer. By storing the offset of the last consumed record for each partition, either in Zookeeper or in Kafka itself, a consumer can stop and restart without losing its place.

Top highlight

Each record consists of a key, a value, and a timestamp. This key is assigned by Kafka when producers publish a record. Keys are used when records are to be written to partitions in a more controlled manner. The simplest such scheme is to generate a consistent hash of the key, and then select the partition number for that record by taking the result of the hash modulo, the total number of partitions in the topic. This assures that records with the same key are always written to the same partition.



Note: While records are opaque byte arrays to Kafka itself, it is recommended that additional structure, or schema, be imposed on the record content so that it can be easily understood. Typically, this schema can be JSON or Avro. It is also normal to add some versioning for when it changes.

Note on Kafka Streams

Kafka's Streams API allows an application to act as a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.

It is normally used for enrichment and transformation. For example, we could have a `users` topic and an `enriched-users` topic. So a `UserEnricher` service could read users events, add information about its company and college and publish the same events but with more information into our enriched-users topic.

Run it locally

You can easily run it using docker:

```
$ docker network create kafka
$ docker run --network kafka -p 2181:2181 -p 9092:9092 --env
ADVERTISED_HOST=`docker-machine ip \`docker-machine active\`` --env
ADVERTISED_PORT=9092 spotify/kafka
```

To test that it is working, let's create a topic and run a consumer and a producer. Kafka exposes an API for these actions. To make it easy, Kafka brings some script that we can use to easily do this. To use these scripts, we can run an image of Kafka in a container and tell this container to execute one of these scripts.

So to create a topic, we can run a container with `bitnami/kafka:latest` image in interactive mode (`-it`) using the same network that our Kafka and Zookeeper containers are using (`– network kafka`). Let's check what is the IP of our container:

```
$ docker network inspect kafka
```

```
# Check IPv4Address, for me it shows 172.19.0.2
```

And now run `kafka-topics.sh` script:

```
docker run -it --network kafka bitnami/kafka:latest kafka-topics.sh
--create --zookeeper 172.19.0.2:2181 --replication-factor 1 --
partitions 1 --topic amazingtopic
```

Now let's check our topic was created properly:

```
docker run -it --network kafka bitnami/kafka:latest kafka-topics.sh
--zookeeper 172.19.0.2:2181 --describe --topic amazingtopic
```

And now let's do the cool stuff: create a Consumer and a Producer. Start creating a consumer by typing:

```
docker run -it --network kafka bitnami/kafka:latest kafka-console-
consumer.sh --bootstrap-server 172.19.0.2:9092 --topic amazingtopic
--from-beginning
```

Now open a new terminal and create a producer:

```
docker run -it --network kafka bitnami/kafka:latest kafka-console-
producer.sh --broker-list 172.19.0.2:9092 --topic amazingtopic
```

When you run this producer, it lets you type words and it publishes a record to Kafka once you click enter. Try to create some records and see how they arrive to the consumer :D

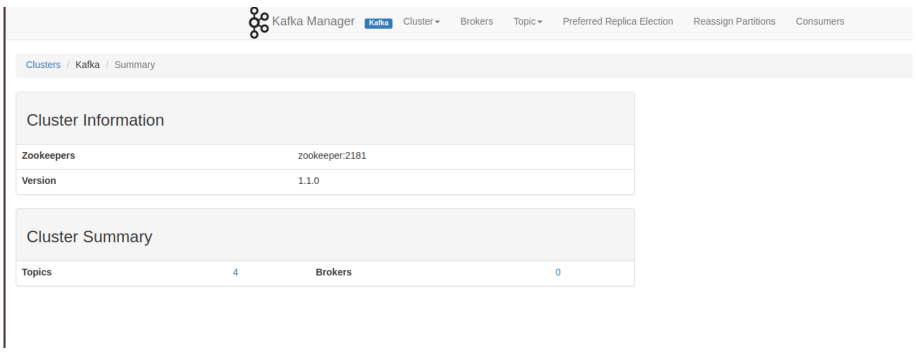
Note: if you kill the consumer, then produce some more records into our `amazingtopic` and finally run a new consumer, you will see that all the records appear again so there is no lost of information while the consumer was down.

Troubleshooting

I recommend you to use KafkaManager. This is an application that fetches metrics from Kafka and shows them in a nice interface.

Its interface is really simple, if you access to `localhost:9000` using your browser, its first page will load and you will see a list of all Kafka cluster that it is connected to.

If you click in one of the clusters, a new page with the summary of it will appear and also, you will see several options on the top menu:



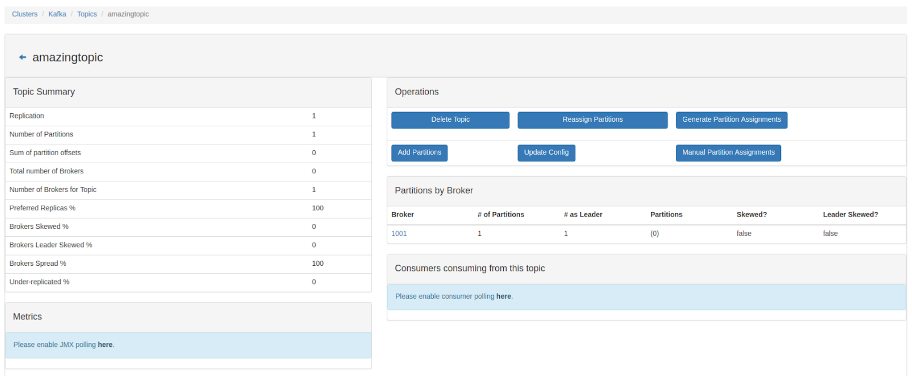
Cluster Information

Zookeepers	zookeeper:2181
Version	1.1.0

Cluster Summary

Topics	4	**Brokers**	0

- **Brokers**. In the brokers main page, you can see some basic info for the brokers (like bytes in and bytes out). If you click on one of the brokers ids, you will see information about records and topics handle by this specific broker.
- **Topic**. Then go to List, you will be redirected to a topics page. Here you can see the number of partitions and replicas for every topic. You can also click on the topic name to find much more information about it.


Clusters / Kafka / Topics / amazingtopic
amazingtopic
Operations
Delete Topic
Reassign Partitions
Generate Partition Assignments
Add Partitions
Update Config
Manual Partition Assignments

Topic Summary

Replication	1
Number of Partitions	1
Sum of partition offsets	0
Total number of Brokers	0
Number of Brokers for Topic	1
Preferred Replicas %	100
Brokers Skewed %	0
Brokers Leader Skewed %	0
Brokers Spread %	100
Under-replicated %	0

Metrics

Please enable JMX polling **here**.

Partitions by Broker

Broker	# of Partitions	# as Leader	Partitions	Skewed?	Leader Skewed?
1001	1	1	(0)	false	false

Consumers consuming from this topic

Please enable consumer polling **here**.

- **Consumers**. You will get a list of consumers. Click on one of the consumers name to see the list of topics it is reading from and click on a topic name to see information like % of partitions assigned to a consumer instance, total lag and consumer offset.

Thanks for reading!

That is all for this overview of Kafka :)

If you want to see code examples of consumers or producers for Kafka, please let me know in a comment below or send a message to my twitter account @Marvalcam1

Kafka DevOps Message Queue Distributed Systems Engineering

Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. Watch

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. Explore

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just $5/month. Upgrade

About Help Legal