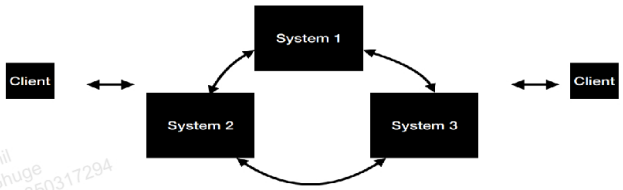
Apache Kafka

Distributed System

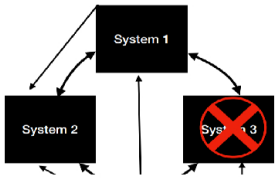
A Distributed System in general collection of systems working together to deliver a value In Distributed Systems we have the clients, who interact with these systems, and the system receives the request from client and responds to the client.



Characteristics of Distributed System

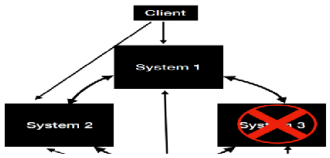
1. Availability and Fault Tolerance:

Lets say one of the system is down, still this wont impact the overall availability of the system. In this situation the client request will handled gracefully.



2. Reliable Work Distribution

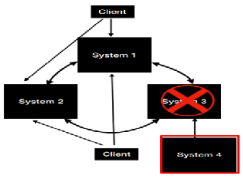
The request's done by the clients are equally distributed for the available systems.



3. Easy Scalable

we can easily add the new systems to the existing to the setup is really easy

Apache Kafka

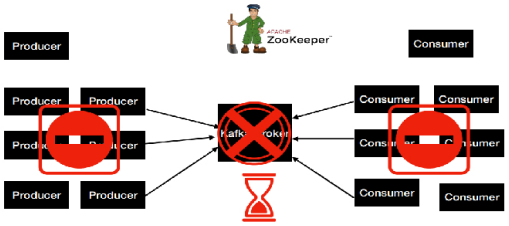


4. Handling Concurrency:

Concurrency can be handled easily with the distributed systems.

Kafka As Distributed System

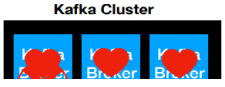
In Kafka we have zookeeper and Broker. In any enterprise its very common to have a lot of producers and consumers. Lets assume with one broker, all the consumers and producers requests will go to the same broker, then there is a possibility of over burden the Broker with the bunch of requests as a result the broker will perform very poor.



If the Broker goes down then there is a no way for serve the producer and consumer requests, this leads to single point of failure. Hence we have to use Kafka as Cluster.

A cluster can have one or more brokers, will be managed by the zookeeper.

Kafka Cluster

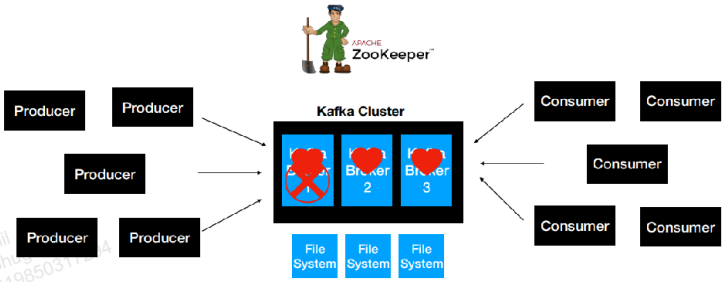


All the brokers will send the heart beats to the zookeeper at a regular intervals, to ensure the zookeeper to be consider the broker as active and be able to serve the client requests. With the 3 brokers in place, the client requests can be distributed among them and it handles the load very well.

if one of the broker goes down then the zookeeper gets notify, then all the client requests will be routed to the other available brokers. even client doesn't know that a broker was down.

Apache Kafka

It is easy to scale the no.of brokers in the cluster without effecting the clients.



- ✓ Client requests are distributed between brokers
- ✓ Easy to scale by adding more brokers based on the need File System File System File System
- ✓ Handles data loss using Replication

Setting up Kafka Cluster with Three Brokers:Ubuntu 20.0.4.LTS

Step-1: create a directory for kafka distribution storage in /opt as shown below:

```
root@ip-172-31-11-4:/opt# mkdir apache-kafka-distro
root@ip-172-31-11-4:/opt# ls -l
total 8
drwxr-sr-x 2 root root 4096 Apr 29 17:42 apache-kafka-distro
```

Step-2: download the kafka distrubution into this location as shown below:

wget https://dlcdn.apache.org/kafka/3.1.0/kafka_2.13-3.1.0.tgz

```
root@ip-172-31-11-4:/opt# cd apache-kafka-distro/
root@ip-172-31-11-4:/opt/apache-kafka-distro#
root@ip-172-31-11-4:/opt/apache-kafka-distro# wget https://dlcdn.apache.org/kafka/3.1.0/kafka_2.13-3.1.0.tgz
--2022-04-29 17:44:30--  https://dlcdn.apache.org/kafka/3.1.0/kafka_2.13-3.1.0.tgz
Resolving dlcdn.apache.org (dlcdn.apache.org)... 151.101.2.132, 2a04:4e42::644
Connecting to dlcdn.apache.org (dlcdn.apache.org)|151.101.2.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 88130011 (84M) [application/x-gzip]
Saving to: 'kafka_2.13-3.1.0.tgz'

kafka_2.13-3.1.0.tgz        100%[=====================================>]  84.05M   133MB/s    in 0.6s

2022-04-29 17:44:30 (133 MB/s) - 'kafka_2.13-3.1.0.tgz' saved [88130011/88130011]

root@ip-172-31-11-4:/opt/apache-kafka-distro# ls -l
total 86068
-rw-r--r-- 1 root root 88130011 Jan 21 14:54 kafka_2.13-3.1.0.tgz
root@ip-172-31-11-4:/opt/apache-kafka-distro#
```

Step-3: extract the kafka distribution using **tar -xvzf kafka_2.13-3.1.0.tgz --strip 1**

Apache Kafka

```
root@ip-172-31-11-4:/opt/apache-kafka-distro# tar -xvzf kafka_2.13-3.1.0.tgz --strip 1
kafka_2.13-3.1.0/LICENSE
kafka_2.13-3.1.0/NOTICE
kafka_2.13-3.1.0/bin/
kafka_2.13-3.1.0/bin/kafka-delete-records.sh
kafka_2.13-3.1.0/bin/trogdor.sh
kafka_2.13-3.1.0/bin/connect-mirror-maker.sh
kafka_2.13-3.1.0/bin/kafka-console-consumer.sh
kafka_2.13-3.1.0/bin/kafka-consumer-perf-test.sh
kafka_2.13-3.1.0/bin/kafka-log-dirs.sh
kafka_2.13-3.1.0/bin/zookeeper-server-stop.sh
kafka_2.13-3.1.0/bin/kafka-verifiable-consumer.sh
```

```
root@ip-172-31-11-4:/opt/apache-kafka-distro# ls -l
total 86136
-rw-r--r-- 1 root root    14536 Jan 12 09:01 LICENSE
-rw-r--r-- 1 root root    28184 Jan 12 09:01 NOTICE
drwxr-xr-x 3 root root     4096 Jan 12 09:07 bin
drwxr-xr-x 3 root root     4096 Jan 12 09:07 config
-rw-r--r-- 1 root root 88130011 Jan 21 14:54 kafka_2.13-3.1.0.tgz
drwxr-xr-x 2 root root     4096 Apr 29 17:45 libs
drwxr-xr-x 2 root root     4096 Jan 12 09:07 licenses
drwxr-xr-x 2 root root     4096 Jan 12 09:07 site-docs
root@ip-172-31-11-4:/opt/apache-kafka-distro#
```

Step-4: Start Zookeeper

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

Step-5: Start Broker-01

> **./bin/kafka-server-start.sh config/server.properties**

Step-6: Start Broker-02

Create new server.properties like **server-01.properties** and configure the broker id and other following information:

broker.id=<unique-broker-d>

listeners=PLAINTEXT://localhost:<unique-port>

log.dirs=/tmp/<unique-kafka-folder>

auto.create.topics.enable=false(optional)

Example:

broker.id=1

listeners=PLAINTEXT://localhost:9093

log.dirs=/tmp/kafka-logs-1

auto.create.topics.enable=false(optional)

And start the broker with the new properties file as shown below:

> **.bin/kafka-server-start.sh config/server-01.properties**

Step-7: Start Broker-03

Create new server.properties like **server-02.properties** and configure the broker id and other following information:

broker.id=<unique-broker-d>

listeners=PLAINTEXT://localhost:<unique-port>

log.dirs=/tmp/<unique-kafka-folder>

auto.create.topics.enable=false(optional)

Example:

broker.id=2

listeners=PLAINTEXT://localhost:9094

log.dirs=/tmp/kafka-logs-2

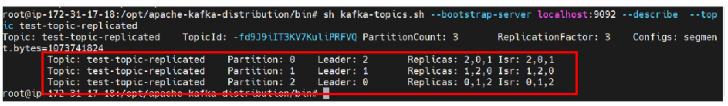
auto.create.topics.enable=false(optional)

And start the broker with the new properties file as shown below:

> **.bin/kafka-server-start.sh config/server-02.properties**

Step-8: Create the topic with the zookeeper and replication factor

>./bin/kafka-topics.sh --create --topic test-topic-replicated --bootstrap-server localhost:9092 --replication-factor 3 --partitions 3



Step-9: Start Kafka Console Producer without key

>**.bin/kafka-console-producer.bat --broker-list localhost:9092 --topic test-topic-replicated**

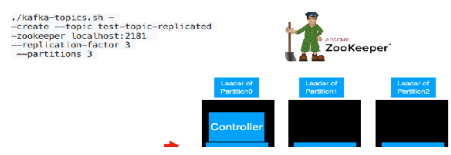
Step-10: Start Kafka Console Consumer without Key

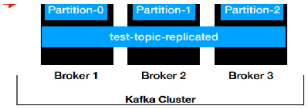
>.bin/kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic **test-topic-replicated** --from-beginning

Kafka Cluster-Producer Flow

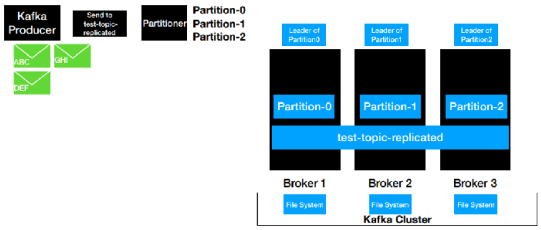
Lets say a Cluster with 3 Brokers and Zookeeper. Out of 3 brokers one of the available broker will act as a Controller. usually this is the first broker to join the cluster.

Anil Ghuge 919850317294





when we create topic, the command will be receive by the Zookeeper, and Zookeeper will redirect the request to the controller broker. the role of the controller is to distribute the partitions to the available brokers. In Distributed Systems, this Concept of distributing the partitions to the available brokers is called as **Leader Assignment.**



So the test topic is distributed across the Kafka Cluster. We know Producer has a layer called partitioner which cares of determining, which partition the message is going to.

The producer sends the first message, its goes to the partitioner before the messages sent to the kafka topic.

The partitioner will determine this message goes to partition-0, in this case the leader of the partition-0 is broker-01, so the message will be sent to broker-01.

The client will always invokes the leader of the Partition after that the message will be persist into the file system of broker-01

when the second message came from producer, then the partitioner will resolves the partition-01 and the leader of the partition-1is broker-2 so the message will go to broker-2 file system.

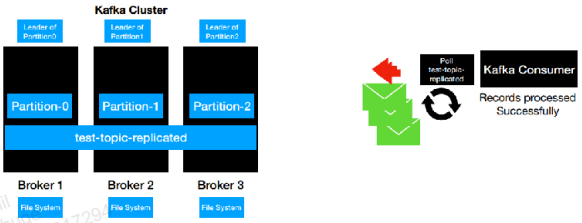
when the third message came from producer, then the partitioner will resolves the partition-02 and the leader of the partition-2 is broker-3 so the message will go to broker-3 file system.

So the messages coming from the producer are distributing among the brokers based on the partition which in-directly means that the load is distributed among the brokers.

Apache Kafka

Kafka Cluster-Consumer Flow

Lets Consider, we have the consumer ready to poll the test-topic created.

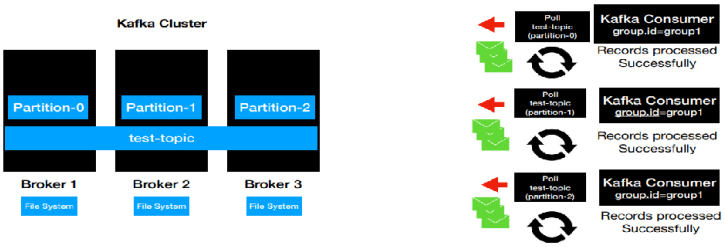


when the poll loop is executed, the request goes to all the partitions and retrieves the records from them here each broker owns a partition. In this Case the poll loop request will goes to all the brokers and retrieves the records from them and the retrieve records are handed over to the kafka consumer and the consumer will process the records successfully and the same flow repeats.

In a nut shell even from the consumer end the request to retrieve the data are distributed among the brokers. basically the consumer call only go to the partition leaders of the topic and retrieve the data

Kafka Consumer Flow with Consumer Groups

It is a common practice, to run the multiple instances of the consumer and process the records from the kafka topic. Lets take the 3 consumers with the same group id.



According to the concept of consumer groups, of there are one more instances of the consumer, with the same group Id, then the partitions are distributed for the scalability message consumption.

so in our case shown above, each consumer will be assigned with one partition when the poll loop gets executed, each instance is going to poll the data from the partition that they are matching with and the poll call will goes to the leader of the partition of the topic and retrieve the data.

So here Partition-0 call goes to broker-1 and Partition-1 call goes to broker-2 and partition-2 call goes to broker-3 and so on and the process repeats.

Note:

- ✓ Partition leaders are assigned during topic Creation
- ✓ Clients will only invoke leader of the partition to produce and consume data
- ✓ Load will evenly distributed between the brokers