APACHE KAFKA

Experiments:

7. Write a script to create a topic with specific partition and replication factor settings.

```
package org.example;

import org.apache.kafka.clients.admin.AdminClient;

import org.apache.kafka.clients.admin.NewTopic;

import org.apache.kafka.common.serialization.StringSerializer;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.Properties;


public class MultipleCluster {

    public static void main(String[] args) {

        Properties properties=new Properties();

        properties.put("bootstrap.servers","localhost:9092");

        properties.put("key.serializer", StringSerializer.class);

        properties.put("value.serializer",StringSerializer.class);

       try {

           AdminClient adminClient = AdminClient.create(properties);

           NewTopic topic1 = new NewTopic("Multitopic", 7, (short) 5);

           adminClient.createTopics(Collections.singleton(topic1)).all().get();
```

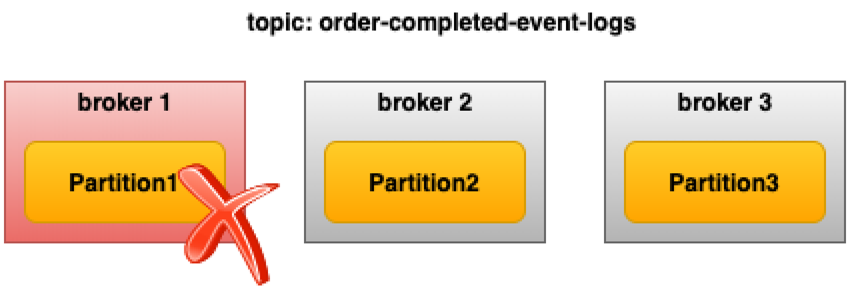
```
            System.out.println("Topic created successfully");
            adminClient.close();
        }
        catch (Exception e) {
            e.printStackTrace();
            System.out.println("Error while creating topics");
        }
        }
    }
```

O/P:

8. Simulate fault tolerance by shutting down one broker and observing the cluster behavior.

Kafka is a distributed system. The topic is divided into partitions and kept in different brokers. If any broker fails, data should not be lost. For fault-tolerance purposes, the partition is replicated and stored in different brokers. If leader brokers fail, then the controller will elects one of the replicas as the leader. Even controller brokers can fail, in this case, Zookeeper will help in electing the broker as the controller.

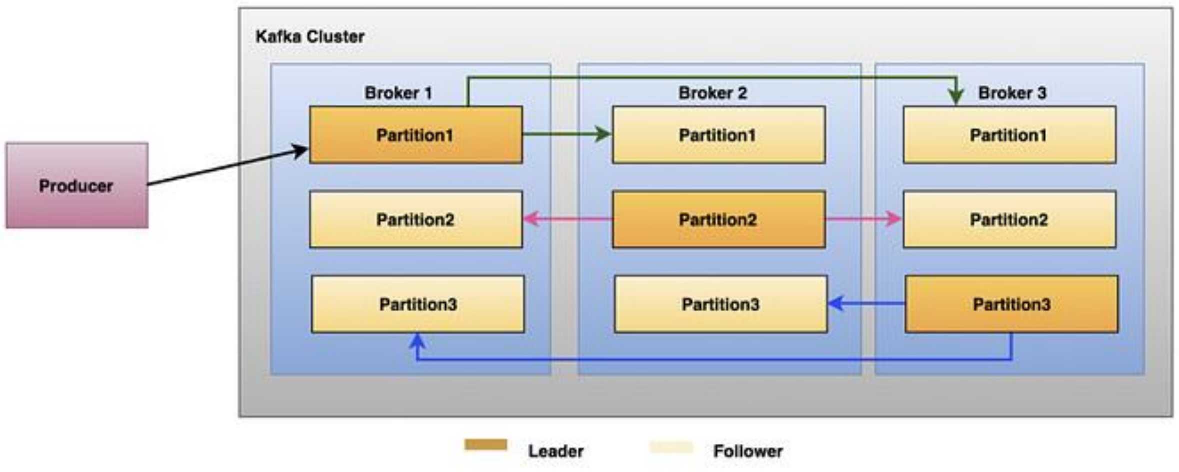
What happens when the broker goes down? Is all the data lost?



Broker Failure

Fault tolerance in Kafka is done by copying the partition data to other brokers which are known as replicas. There is a configuration that specifies how many copies of the partition you need. Its called a replication factor.

Each broker will hold one or more partitions. And each of these partitions can either be a replica or leader for the topic. All the writes and reads to a topic go through the leader and the leader coordinates to update replicas with new data.



Replication

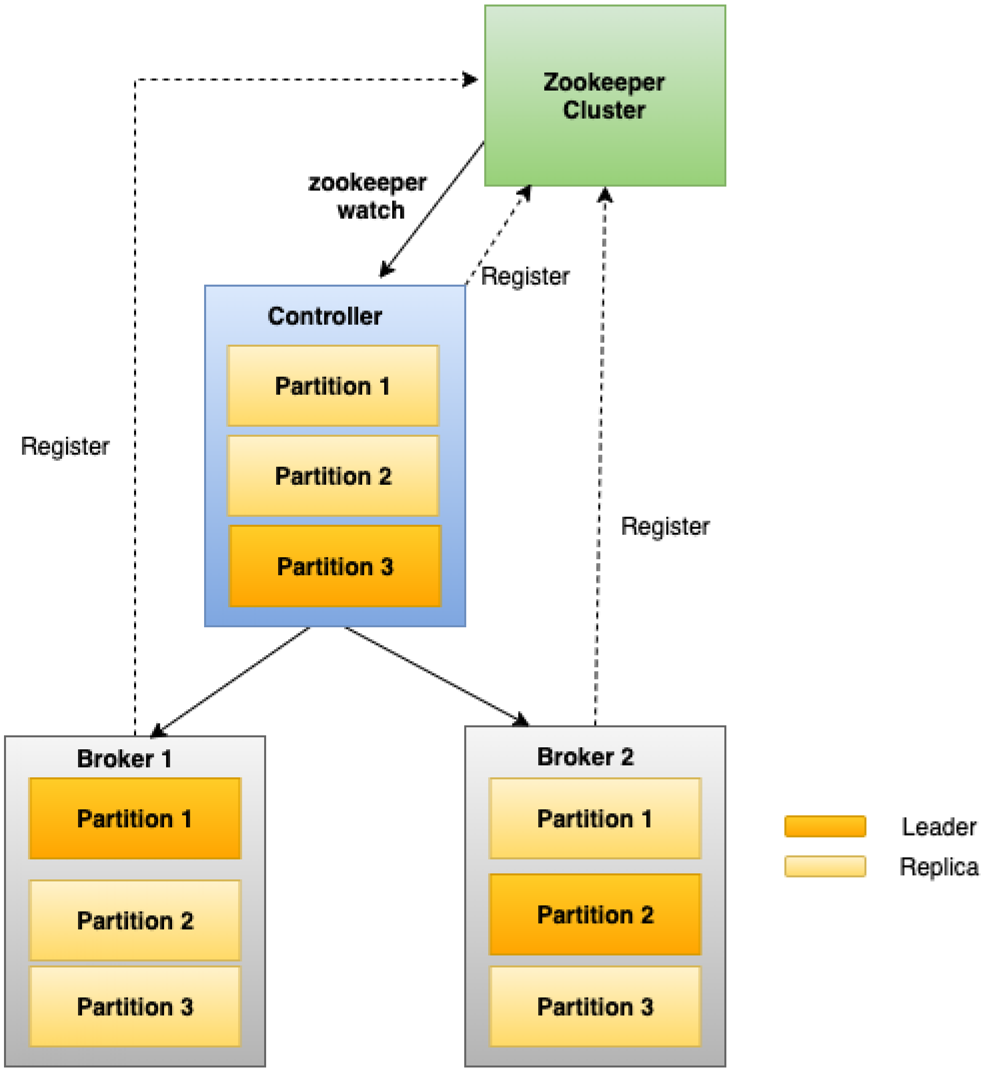
In the above example, there are three brokers. For partition-1, Broker-1 is a leader. Broker-2 and Broker-3 are the replica brokers.

The leader partitions and replica partitions are kept in separate brokers because if a leader partition goes down, one of the replica partition brokers can serve as the leader.

But how are leaders elected? There should be somebody who is responsible for electing leaders in the Kafka cluster. That's where Controller brokers come into the picture.

Controller broker:

The controller Broker takes care of electing the leader broker for the partitions. It’s just a normal broker with extra responsibility. **There will be only one controller for the Kafka cluster**. The controller broker keeps track of brokers joining and leaving the cluster with the help of Zookeeper.



controller

As we saw earlier, **Zookeeper** is the centralized service for storing metadata of topic, partition, and broker. Every time a broker starts up, it registers itself to the zookeeper. And the zookeeper keeps track of each broker by calling a health check on it. Just like Kafka brokers, even Zookeepers run as a cluster, known as an **ensemble**.

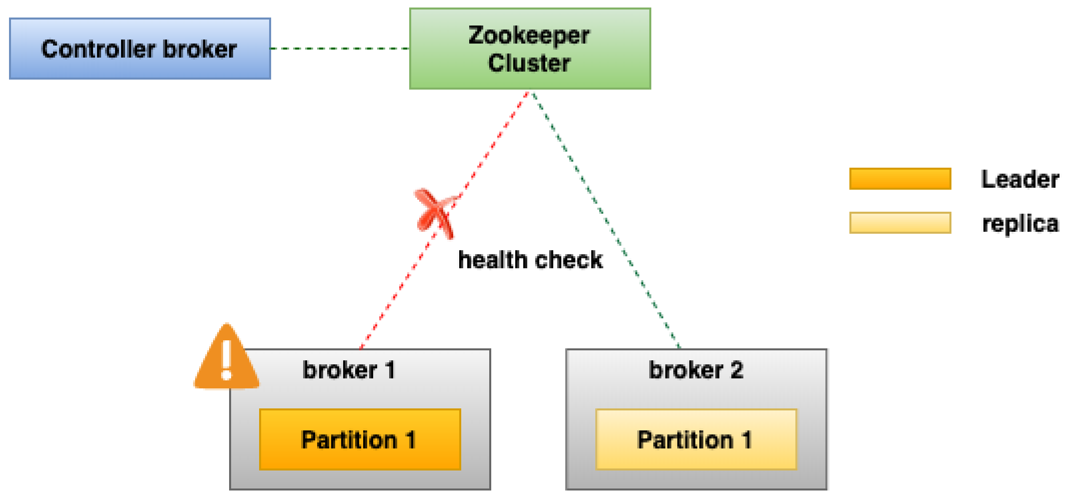
Leader partition election:

When the leader broker goes down,

1. The Zookeeper informs the Controller.

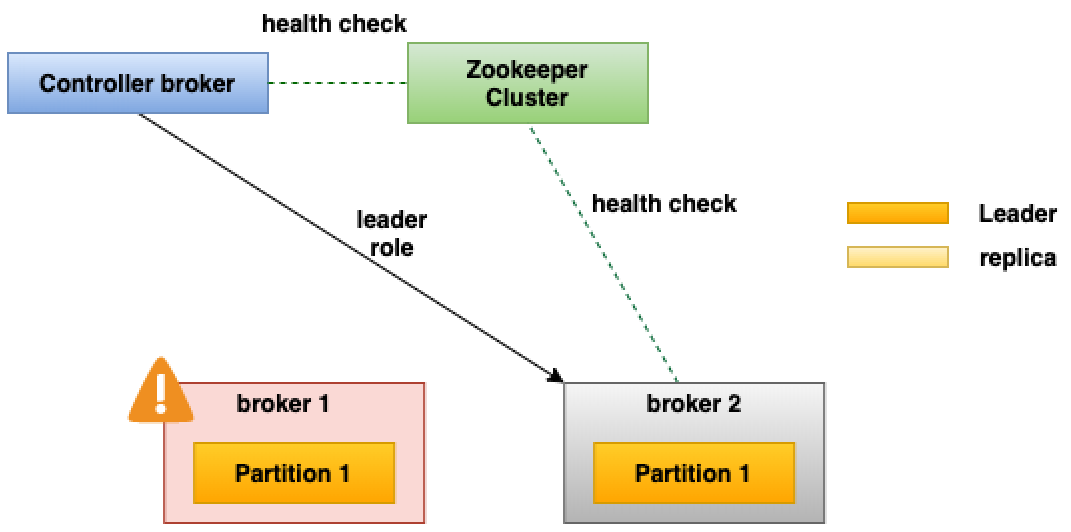
2. The controller selects one of the **in-sync replicas (ISR)** as the leader. ISR is a replica broker that is fully caught up with the changes of the leader broker. The leader is responsible for keeping track of ISR and sending this information to the zookeeper.

3. When the broker comes back up, then it will be assigned again as the leader.



Leader failure

In the above example, there is broker-1 which is down due to some issue. And the zookeeper keeps track of it because of health checks. Zookeeper sends a notification to the controller regarding the broker-1 unavailability.

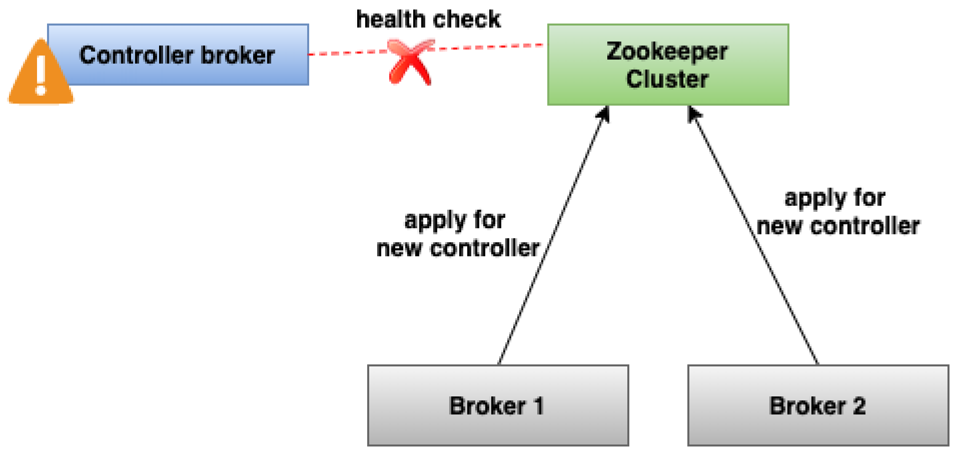


Leader election

The controller chooses one of the replica partitions as the leader for this partition. In the above example, it is broker-2. When the broker-1 comes back up, it will be assigned again as the leader.

But what if the controller itself goes down?

Controller election



Controller failure

When the controller goes down,

- Zookeeper informs all the brokers that the controller failed.
- All the brokers will apply to be the controller.
- The first broker who applies for this position will become the controller.

9. Implement operations such as listing topics, modifying configurations, and deleting topics.

```
package org.example;

import org.apache.kafka.clients.admin.AdminClient;

import org.apache.kafka.clients.admin.ListTopicsResult;

import org.apache.kafka.clients.admin.NewPartitions;

import org.apache.kafka.common.errors.TopicExistsException;

import java.util.Collections;

import java.util.Properties;

import java.util.concurrent.ExecutionException;

public class multiFunction {

    public static void main(String[] args) throws ExecutionException,
InterruptedException

    {

        Properties properties = new Properties();

        properties.put("bootstrap.servers","localhost:9092");

        AdminClient adminClient=AdminClient.create(properties);

        System.out.println("FOLLOWING IS THE LIST OF TOPICS");

        ListTopicsResult topics=adminClient.listTopics();

        topics.names().get().forEach(System.out::println);
        String topicName = "newtopic";

        int newPartitionCount =2;

        System.out.println("Changing Configuration ( partitions ) for Topic "+
topicName);

        try( AdminClient adminClient1 = AdminClient.create(properties))

{

            NewPartitions newPartitions =
NewPartitions.increaseTo(newPartitionCount);

            adminClient1.createPartitions(Collections.singletonMap(topicName,
newPartitions)).all().get();

            System.out.println("Successfully increased partitions for topic: " +
topicName);

           }

catch (InterruptedException | ExecutionException e)

{

                    if (e.getCause() instanceof TopicExistsException)

{

               System.out.println("Topic already exists: " + topicName);

            } else {

               System.err.println("Error increasing partitions: " + e.getMessage());

               e.printStackTrace();

            }

        }
```

```
    }
}
```

O/P:

❖ **Deleting Topics:**

```
package org.example;
import org.apache.kafka.clients.admin.AdminClient;
import org.apache.kafka.clients.admin.ListTopicsResult;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.StringSerializer;
import java.io.IOException;
import java.util.HashSet;
import java.util.Properties;
import java.util.Set;
import java.util.concurrent.ExecutionException;

public class DeleteKafkaTopics {

    public static void main(String[] args) throws IOException,
ExecutionException, InterruptedException {
        Properties properties=new Properties();
        properties.put("bootstrap.servers","localhost:9092");
        properties.put("key.serializer", StringSerializer.class);
        properties.put("value.serializer", StringSerializer.class);
        AdminClient adminClient = AdminClient.create(properties);

        Set<String> topics = new HashSet<>();
        topics.add("topic-from-java");
        adminClient.deleteTopics(topics);
        adminClient.close();
    }
}
```

10. Introduce Kafka Connect and demonstrate how to use connectors to integrate with externalsystems.

Kafka Connect is a tool to reliably stream data between Kafka topic and external datastores using out of the box libraries,
without writing even a single line of code. It is usually used to replicate data between standard external datastores,
for example excel to MySql , via Kafka topic in between.

Following are the steps to stream data from a source file to a target file using Kafka FileStream Connectors.
Prerequisite
Install Kafka as per this guideline. Kafka connect comes along with standard Kafka installation, so no need to install it separately.

Step 1
Start zookeeper server from Kafka installation bin directory using below command.

```
./zookeeper-server-start.bat ./config/zookeeper.properties
```

Step 2
Start Kafka server from Kafka installation bin directory using below command.

```
./kafka-server-start.bat ./config/server.properties
```

Step 3
Add “connect-file-3.3.2.jar” to plugin.path in “config/connect-standalone.properties” file as per below
(Replace your installation directory accordingly).

This jar version could be different if you are working on different Kafka version than here.
Replace JsonConvertor with String convertor for data transfer for text document.

```
key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.storage.StringConverter
plugin.path=C:/kafka/libs/connect-file-3.8.0.jar
```

Step 4
Create a source file (source.txt) and update “config/connect-file-source.properties” file accordingly.
Save changes as below.

```
name=local-file-source
connector.class=FileStreamSource
tasks.max=1
file=C:/kafka/connect/source.txt
topic=connect-test
```

Step 5
Create a target file(target.txt) and update “config/connect-file-sink.properties” file accordingly.
Save changes as below.

```
name=local-file-sink
connector.class=FileStreamSink
tasks.max=1
file=C:/kafka/connect/target.txt
topics=connect-test
```

Step 6
Start Kafka connector in standalone mode using below command from Kafka installation bin directory.

```
.\bin\windows\connect-standalone.bat .\config\connect-standalone.properties
.\config\connect-file-source.properties .\config\connect-file-sink.properties
```

Once successfully up, any change made on the source file will immediately reflect on the target file.
Kafka connect FileStreamSource will stream any change in the source file into kafka topic.
And FileStreamSink will steam it from kafka topic into the target file.

11. Implement a simple word count stream processing application using Kafka Stream.

```
package org.example;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.kstream.KTable;
import org.apache.kafka.streams.kstream.Produced;

import java.util.Arrays;
import java.util.Properties;
import java.util.regex.Pattern;

public class wordcount1 {
    final static Pattern pattern = Pattern.compile("\\W+",
Pattern.UNICODE_CHARACTER_CLASS);

    public static void main(String[] args) {
        Properties props = new Properties();
        props.put(StreamsConfig.APPLICATION_ID_CONFIG,
"word-count-ap");

props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
"localhost:9092");

props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CO
NFIG, Serdes.String().getClass());

props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_
CONFIG, Serdes.String().getClass());

        StreamsBuilder builder = new StreamsBuilder();
        KStream<String, String> textLines =
builder.stream("sentences");

        KTable<String, Long> wordsCount = textLines
                .flatMapValues(value ->
Arrays.asList(pattern.split(value.toLowerCase())))
                .groupBy((key, value) -> value)
                .count();

        wordsCount.toStream().to("wordcount1",
Produced.with(Serdes.String(), Serdes.Long()));

        KafkaStreams kafkaStreams = new
KafkaStreams(builder.build(), props);
        kafkaStreams.start();

        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
                System.out.println("Shutting down stream");
                kafkaStreams.close();
            }));
        }
};
```

For this,

1. Create a topic called ***word-count*** and create another topic called ***sentences***.

   .\bin\windows\kafka-topics.bat --create –topic word-count –bootstrap-server localhost:9092

   .\bin\windows\kafka-topics.bat --create –topic sentences –bootstrap-server localhost:9092

2. Then, in sentences topic produce messages
   .\bin\windows\kafka-console-producer.bat –topic sentences –bootstrap-server localhost:9092
   and consume messages using the following command in new terminal.
   .\bin\windows\kafka-console-consumer.bat –topic word-count –bootstrap-server localhost:9092 –from-beginning –property print.key=true –property key.seperator=":" –key-deserializer "org.apache.kafka.common.serialization.StringDeserializer" –value-deserializer "org.apache.kafka.common.serialization.LongDeserializer"