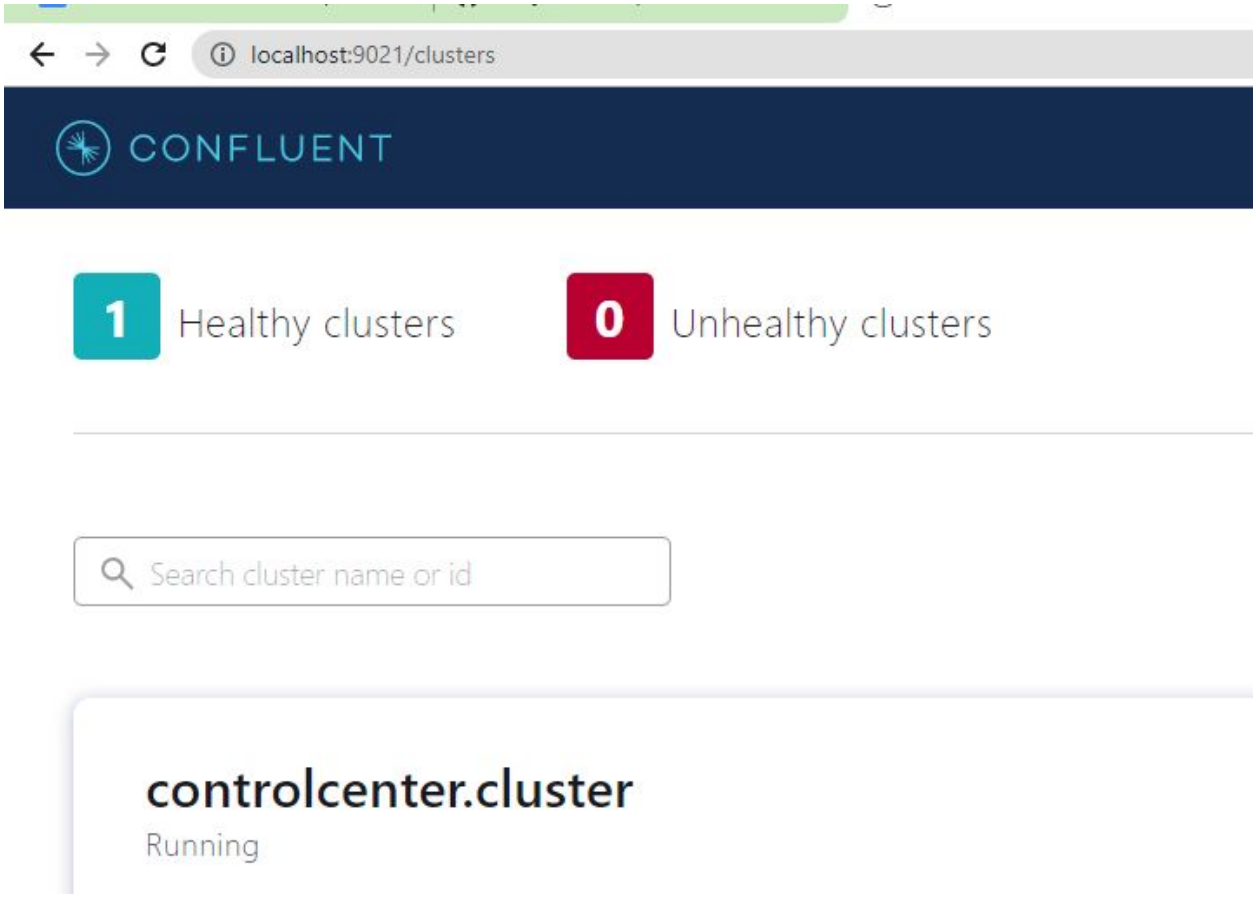
Confluent Certified Developer for Apache Kafka Certification Examination -Notes

Setting Up Environment

We will be using confluent kafka docker images to run the kafka services. Follow

Running Apache Kafka in docker container - First time

- Clone the repository
-
```
# git clone
https://github.com/balajich/CCD-Apache-Kafka-Certification-Examination-Notes.git
```
- Change to cloned directory
-
```
# cd CCD-Apache-Kafka-Certification-Examination-Notes
```
- Start the linux virtual machine using Vagrant
-
```
# vagrant up
```
- Take ssh to linux
-
```
# vagrant ssh
```
- Switch to root user in linux machine
-
```
[vagrant@kserver ~]$ sudo su -
```
- Go to vagrant folder
-
```
[root@kserver ~]# cd /vagrant/
```
- Start the kafka server using docker-compose, The below command will start all the dependent services
-
```
[root@kserver vagrant]# docker-compose up -d
```
- You should see output as
-
```
zookeeper is up-to-date
Starting broker ... done
schema-registry is up-to-date
rest-proxy is up-to-date
Starting connect ... done
Starting ksqldb-server ... done
Starting control-center ...
Starting ksql-datagen ...
Starting control-center ... done
```
-
- Access confluent command center UI. VM is configured to do port forward of command center UI port. http://localhost:9021/

- 

Stopping Apache Kafka that is running in docker container

```
[root@kserver vagrant]# docker-compose stop
Stopping ksql-datagen    ... done
Stopping control-center  ... done
Stopping ksqldb-cli      ... done
Stopping ksqldb-server   ... done
Stopping connect         ... done
Stopping rest-proxy      ... done
Stopping schema-registry ... done
Stopping broker          ... done
Stopping zookeeper       ... done
```

Starting the container that was stopped

```
[root@kserver vagrant]# docker-compose start
Starting zookeeper       ... done
Starting broker          ... done
Starting schema-registry ... done
Starting connect         ... done
Starting ksqldb-server   ... done
Starting control-center  ... done
Starting ksqldb-cli      ... done
Starting ksql-datagen    ... done
Starting rest-proxy      ... done
```

Deleting Kafka environment

```
[root@kserver vagrant]# docker-compose down
Stopping control-center ... done
Stopping ksqldb-cli     ... done
Stopping ksqldb-server  ... done
Stopping zookeeper      ... done
Removing ksql-datagen    ... done
Removing control-center  ... done
Removing ksqldb-cli      ... done
Removing ksqldb-server   ... done
Removing connect         ... done
Removing rest-proxy      ... done
Removing schema-registry ... done
Removing broker          ... done
Removing zookeeper       ... done
Removing network vagrant_default
```

Troubleshooting

Checking state of kafka process that are running in docker container

```
[root@kserver vagrant]# docker-compose ps
     Name                   Command               State                    Ports
-------------------------------------------------------------------------------------------------------
broker            /etc/confluent/docker/run        Up      0.0.0.0:9092->9092/tcp,
0.0.0.0:9101->9101/tcp
connect           /etc/confluent/docker/run        Up      0.0.0.0:8083->8083/tcp, 9092/tcp
control-center    /etc/confluent/docker/run        Up      0.0.0.0:9021->9021/tcp
ksql-datagen      bash -c echo Waiting for K ...   Up
ksqldb-cli        /bin/sh                          Up
ksqldb-server     /etc/confluent/docker/run        Up      0.0.0.0:8088->8088/tcp
rest-proxy        /etc/confluent/docker/run        Up      0.0.0.0:8082->8082/tcp
schema-registry   /etc/confluent/docker/run        Up      0.0.0.0:8081->8081/tcp
zookeeper         /etc/confluent/docker/run        Up      0.0.0.0:2181->2181/tcp, 2888/tcp,
3888/tcp
```

Stop the Linux virtual machine

```
CCD-Apache-Kafka-Certification-Examination-Notes> vagrant halt
==> kserver: Attempting graceful shutdown of VM...
```

Application Design

Overall Apache Kafka architecture and design

Kafka’s command line tools

Kafka provides several command-line interface (CLI) utilities that helps to manage topics, clusters etc

Topic Operations

The **kafka-topics.sh** tool provides an easy way to perform most topic operations. This allows you to create, modify, delete, and list information about topics in a cluster.

Creating a New Topic in a cluster

```
docker-compose exec kafka-1 kafka-topics \
  --create \
  --bootstrap-server localhost:19092 \
  --replication-factor 2 \
  --partitions 8 \
  --topic my-topic
```

Creates a new topic named my-topic with replication factor as 2 and partitions as 8

Key Points

- It is not recommend to have topic names starts with two underscore like __my-topic
- Kafka internal topics starts with two underscores example **__consumer_offsets**
- In a single cluster we shouldn't have topic names with both underscores and period

Increasing partitions of topic

```
# Increase partitions of a topic
docker-compose exec kafka-1 kafka-topics \
 --alter \
 --bootstrap-server localhost:19092 \
 --partitions 16 \
 --topic my-topic
```

Increase the partitions of topic my-topic form 8 to 16

Key Points

- It is not best practice to increase the size of partitions once created.
- Topics that are having messages that is produced with Keys are difficult to change partitions

Decreasing partitions of topic

It is not possible to reduce the number of partitions of a topic

Deleting a Topic

```
# Delete a topic
```

```
docker-compose exec kafka-1 kafka-topics \
  --delete \
  --bootstrap-server localhost:19092 \
  --topic my-topic
```

Delete my-topic from the clustger

Key point

Delete a topic will also delete all the messages. This is a non reversible action.

Listing all topics in a Cluster

```
 # list all topics in cluster
docker-compose exec kafka-1 kafka-topics \
  --list \
  --bootstrap-server localhost:19092
```

Describe all the topics

```
# Describe all the topics in cluster
docker-compose exec kafka-1 kafka-topics \
  --describe \
  --bootstrap-server localhost:19092
```

Console Producer

The **kakfa-console-producer**.sh tool can be used to write messages into a Kafka topic in your cluster

```
#  Write messages to my-topic using console producer
docker-compose exec kafka-1 kafka-console-producer \
  --topic my-topic \
  --bootstrap-server localhost:19092 \
```

Console Consumer

The **kafka-console-consumer**.sh tool is used to read messages from one or more topics

```
# Read messages from my-topic
docker-compose exec kafka-1 kafka-console-consumer \
  --topic my-topic \
  --bootstrap-server localhost:19092
```

By default the tool reads latest messages from topic

To read messages from beginning

```
# Read messages from my-topic from-beginning
docker-compose exec kafka-1 kafka-console-consumer \
  --topic my-topic \
  --from-beginning \
  --bootstrap-server localhost:19092
```

Consumer Groups

The **kafka-consumer-groups**.sh tool can be used to list and describe both old and new consumer groups.
For old consumer the group information is maintained in Zookeeper
For new consumers the group information is maintained in Broker

List Consumer Groups in a cluster

```
# List new consumer groups
docker-compose exec kafka-1 kafka-consumer-groups \
  --list \
  --bootstrap-server localhost:19092
```

Describe a Consumer Group

```
# Describe a consumer group
docker-compose exec kafka-1 kafka-consumer-groups \
  --describe \
  --group console-consumer-51714 \
  --bootstrap-server localhost:19092
```

Pub/Sub and Streaming

Publisher (Pub) or Producers

In this section we are going to learn about publishing messages to a topic in Kafka Cluster using console-producer, java and python clients.

Understand various options of a producer

Java- HelloWorldProducer

Write a hello world message topic . The message will be placed in a buffer and will be sent to the broker in a separate thread.

```
public class HelloWorldProducer {
    public static void main(String[] args) {
        Properties kafkaProps = new Properties();
        kafkaProps.put("bootstrap.servers", "broker-1:19092,broker-2:29092,broker-3:39092");
        kafkaProps.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        kafkaProps.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

        ProducerRecord<String, String> record = new ProducerRecord<>("hello-world-topic",
null, "Hello World");
        KafkaProducer<String, String> producer = new KafkaProducer<>(kafkaProps);
        try {
            producer.send(record); //send record
            producer.flush(); //flush the accumulated records
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Java -HelloWorldProducerSynchronously

Send data to broker Synchronously

```
public class HelloWorldProducerSynchronously {
    public static void main(String[] args) {
        Properties kafkaProps = new Properties();
        kafkaProps.put("bootstrap.servers", "broker-1:19092,broker-2:29092,broker-3:39092");
        kafkaProps.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        kafkaProps.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

        ProducerRecord<String, String> record = new ProducerRecord<>("hello-world-topic",
```

```
null, "Hello World");
        KafkaProducer<String, String> producer = new KafkaProducer<>(kafkaProps);
        try {
            producer.send(record).get(); //send synchronously
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Java- HelloWorldProducerSynchronouslyRecordMetaData

Get metadata of the published record on the broker

```
public static void main(String[] args) {
        Properties kafkaProps = new Properties();
        kafkaProps.put("bootstrap.servers", "broker-1:19092,broker-2:29092,broker-3:39092");
        kafkaProps.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        kafkaProps.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

        ProducerRecord<String, String> record = new ProducerRecord<>("hello-world-topic",
null, "Hello World");
        KafkaProducer<String, String> producer = new KafkaProducer<>(kafkaProps);
        try {
            RecordMetadata metadata = producer.send(record).get(); //send synchronously
            System.out.println("Topic name : "+metadata.topic());
            System.out.println("partition : "+metadata.partition());
            System.out.println("offset : "+metadata.offset());
            System.out.println("timestamp : "+metadata.timestamp());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Java-HelloWorldProducerASynchronouslyWithCallBack

Register a callback object

```
try {
            producer.send(record, new HelloWorldProducerCallback());// provide a call back
object
            producer.flush();// flush the records
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
@Override
   public void onCompletion(RecordMetadata recordMetadata, Exception e) {
      if (e != null) {
         e.printStackTrace();
      }
      System.out.println("Topic name : " + recordMetadata.topic());
      System.out.println("partition : " + recordMetadata.partition());
      System.out.println("offset : " + recordMetadata.offset());
      System.out.println("timestamp : " + recordMetadata.timestamp());
   }
```

Development

Deployment/Testing/Monitoring

Frequently Asked Questions

1. What is Kafka?
    a. Kafka is a distributed pub sub system, it combines three important capabilities.
    b. To publish (write) and subscribe to (read) streams of events
    c. To store streams of events durably and reliably for as long as you want
    d. To process streams of events as they occur or retrospectively.
2. What are the main components in the kafka system?
    a. Topic – is a palace where records are stored and published.
    b. Producer – Publishes message to topic
    c. Consumer – Subscribes to topic and pull data from it
    d. Brokers – a set of servers where published messages are stored
    e. Zookeeper - Keeps track of status of the kafka cluster nodes, It is a configuration management service for cluster
3. What is a Zookeeper?
    a. It is a centralized configuration management service. It keeps track of nodes health, topics, offsets and partition information etc.
4. What is Leader and Follower in kafka cluster
    a. For every partition in Kafka cluster one broker plays a role of Leader and there will be zero or more followers
    b. Leader is responsible for reading and writing to that partition, whereas followers passively replicate the data.

c. If Leader is failed one for the follower will take the role of a leader

5. In which language Kafka is written?
    a. Majority part of the Kafka is written in scala and runs in a JVM
6. What is offset and its significance?
    a. An offset is a unique number given to a message in a Partition. This helps us to uniquely identify a message in Partition.
7. Explain the importance of the consumer group?
    a. Consumer group is used to uniquely distribute messages across the consumers.
    b. You cannot read messages from kafka topics without a consumer group.
    c.
8. Name a few errors that are retriable by Kafka Producer?
    a. LEADER_NOT_AVAILABLE
    b. NOT_LEADER_FOR_PARTITION
    c. UNKNOWN_TOPIC_OR_PARTITION
9. How can you uniquely identify a message in a kafka cluster?
    a. The combination of topicname , partition number and offset number will form a unique id
10. How message order is guaranteed in kafka?
    a. Message order in kafka is guaranteed using partitions, All the messages with the same key go to the same partition. Message order is guaranteed within a partition but not across the partitions.
11. What is the default nature of send() in org.apache.kafka.clients.producer.KafkaProducer?
    a. send() is asynchronous in nature. By default the places the message in buffer and returns. There will be a separate thread that sends messages to the broker.
12.

Quizz

1. Is message order guaranteed in the Kafka cluster?
    a. No, it is not possible to get message order guaranteed in kafka cluster
    b. Yes, it is possible but within a partition.
    c. Yes, it is possible across the partitions.
    d. None of the above

Answer B

Message order in kafka is guaranteed using partitions, All the messages with the same key go to the same partition. Message order is guaranteed within a partition but not across the partitions.

2. Do we need to provide all the brokers hostname and ports for the property '**bootstrap.servers**' when connecting to the kafka cluster?
    a. Yes, we need to provide all the broker hostnames and ports
    b. No, we need only zookeeper hostnames and ports
    c. No, we need at least one broker hostname and port
    d. None of the above

Answer C

We need at least one brokername and port. We never specify zookeeper hostnames while connecting to kafka cluster

3. What is the default nature of send(ProducerRecord<K, V> record) in org.apache.kafka.clients.producer.KafkaProducer?
    a. ASynchronous -doesn't wait till the message is send to broker
    b. Synchronous
4. Which of the following is not a part of the Kafka ecosystem?
    a. Broker
    b. Partition
    c. Offset
    d. Relational database

Answer D
Kafka ecosystem does not contain a relation database.

5. Can two messages in a partition have the same offset?
    a. Yes
    b. No

c. Randomly sometimes they can have the same offset.
d. None of the above

Answer B An offset is a unique number given to a message in a Partition. This helps us to uniquely identify a message in Partition. No two messages in a partition can have common offset

6. Can Kafka cluster run without a Zookeeper?

a. Yes
b. No

Answer B Clients cannot connect to Kafka server without zookeeper.It is a centralized configuration management service. It keeps track of nodes health, topics, offsets and partition information etc.

7.
8.

References

Confluent Docker	https://docs.confluent.io/platform/current/quickstart/cos-docker-quickstart.html
Github url for confluent docker images	https://github.com/confluentinc/cp-all-in-one
Setting up multi node cluster	https://better-coding.com/building-apache-kafka-cluster-using-docker-compose-and-virtualbox/
Good Book	Kafka: The Definitive Guide- Real-Time Data and Stream Processing at Scale
Python client	https://docs.confluent.io/clients-confluent-kafka-python/current/overview.html

Kafka Listeners - Explained	https://rmoff.net/2018/08/02/kafka-listeners-explained/
Kafka Intro	https://kafka.apache.org/intro
Exam notes	https://codingnconcepts.com/post/apache-kafka-ccdak-exam-notes/

Miscellaneous

Docker

Bring all the services up

```
[root@kserver vagrant]# docker-compose up -d
vagrant_zookeeper-3_1 is up-to-date
vagrant_zookeeper-1_1 is up-to-date
vagrant_zookeeper-2_1 is up-to-date
vagrant_broker-1_1 is up-to-date
vagrant_broker-2_1 is up-to-date
vagrant_broker-3_1 is up-to-date
```

Check the logs of all the services

```
[root@kserver vagrant]# docker-compose logs
```

Stop and delete all the services

```
root@kserver vagrant]# docker-compose down
Stopping vagrant_broker-3_1    ... done
Stopping vagrant_broker-1_1    ... done
Stopping vagrant_broker-2_1    ... done
```

```

```

Stop a particular service

```
[root@kserver vagrant]# docker-compose stop broker-1
Stopping vagrant_broker-1_1 ... done
```

Start a particular service

```
[root@kserver vagrant]# docker-compose start broker-1
Starting broker-1 ... done
```

Access logs of a particular service

```
[root@kserver vagrant]# docker-compose logs broker-2

[root@kserver vagrant]# docker-compose logs -f broker-2
```