Lab 01 Fundamentals of Apache Kafka

a. Introduction

This document provides Hands-On Exercises for the course **Confluent Developer Skills for Building Apache Kafka**. You will use a setup that includes a virtual machine (VM) configured as a Docker host to demonstrate the distributed nature of Apache Kafka.

The main Kafka cluster includes the following components, each running in a container:

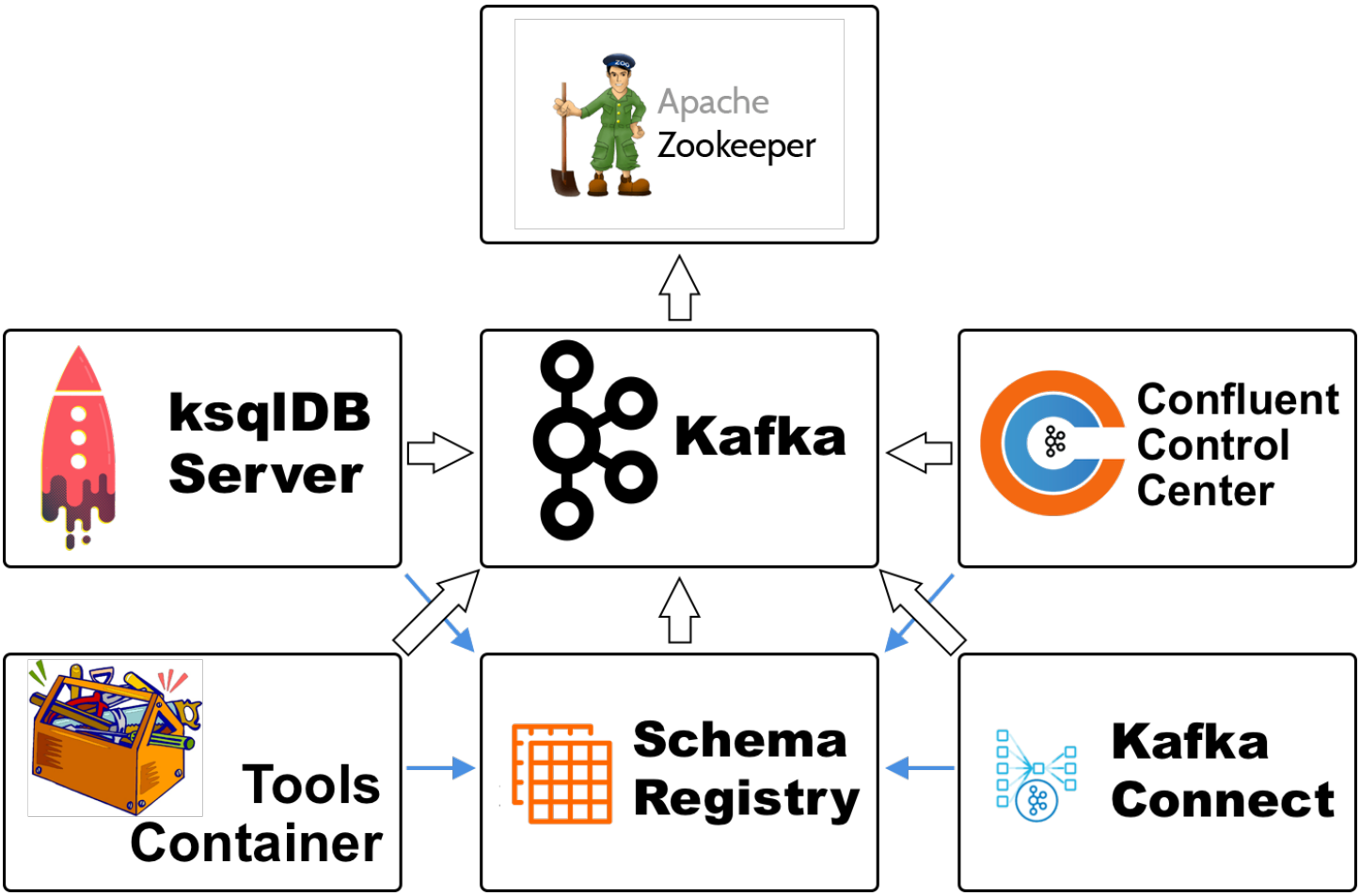


Table 1. Components of the Confluent Platform

Alias	Description
zookeeper	ZooKeeper
kafka	Kafka Broker
schema-registry	Schema Registry

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.

Alias	Description
connect	Kafka Connect
ksqldb-server	ksqlDB Server
control-center	Confluent Control Center
tools	secondary location for tools run against the cluster

As you progress through the exercises you will selectively turn on parts of your cluster as they are needed.

You will use Confluent Control Center to monitor the main Kafka cluster. To achieve this, we are also running the Control Center service which is backed by the same Kafka cluster.

In this course we are using Confluent Platform version 7.0.0 which includes Kafka 3.0.0.

In production, Control Center should be deployed with its own dedicated Kafka cluster, separate from the cluster with production traffic. Using a dedicated metrics cluster is more resilient because it continues to provide system health monitoring even if the production traffic cluster experiences issues.

Alternative Lab Environments

As an alternative you can also download the VM to your laptop and run it in VirtualBox. Make sure you have the newest version of VirtualBox installed. Download the VM from this link:

- https://s3.amazonaws.com/confluent-training-images-us-east-1/training-ubuntu-20-04-jan2022.ova

If you have installed Docker for Desktop on your Mac or Windows 10 Pro machine then you can run the labs there. But please note that your trainer might not be able to troubleshoot any potential problems if you are running the labs locally. If you choose to do this, follow the instructions at → Running Labs in Docker for Desktop.

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.

Command Line Examples

Most exercises contain commands that must be run from the command line. These commands will look like this:

```
$ pwd
/home/training
```

Commands you should type are shown in **bold**; non-bold text is an example of the output produced as a result of the command.

Preparing the Labs

Welcome to your lab environment! You are connected as user **training**, password **training**.

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.

If you haven't already done so, you should open the **Exercise Guide** that is located on the lab virtual machine. To do so, open the **Confluent Training Exercises** folder that is located on the lab virtual machine desktop. Then double-click the shortcut that is in the folder to open the **Exercise Guide**.

Copy and paste works best if you copy from the Exercise Guide on your lab virtual machine.

- Standard Ubuntu keyboard shortcuts will work: `Ctrl+C` → Copy, `Ctrl+V` → Paste
- In a Terminal window: `Ctrl+Shift+C` → Copy, `Ctrl+Shift+V` → Paste.

If you find these keyboard shortcuts are not working you can use the right-click context menu for copy and paste.

1. Open a terminal window
2. Clone the source code repository to the folder `confluent-dev` in your **home** directory:

```
$ cd ~
$ git clone --depth 1 --branch 7.0.0-v1.0.5 \
    https://github.com/confluentinc/training-developer-src.git \
    confluent-dev
```

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.

If you chose to select another folder for the labs then note that many of our samples assume that the lab folder is `~/confluent-dev`. You will have to adjust all those command to fit your specific environment.

3. Navigate to the `confluent-dev` folder:

```
$ cd ~/confluent-dev
```

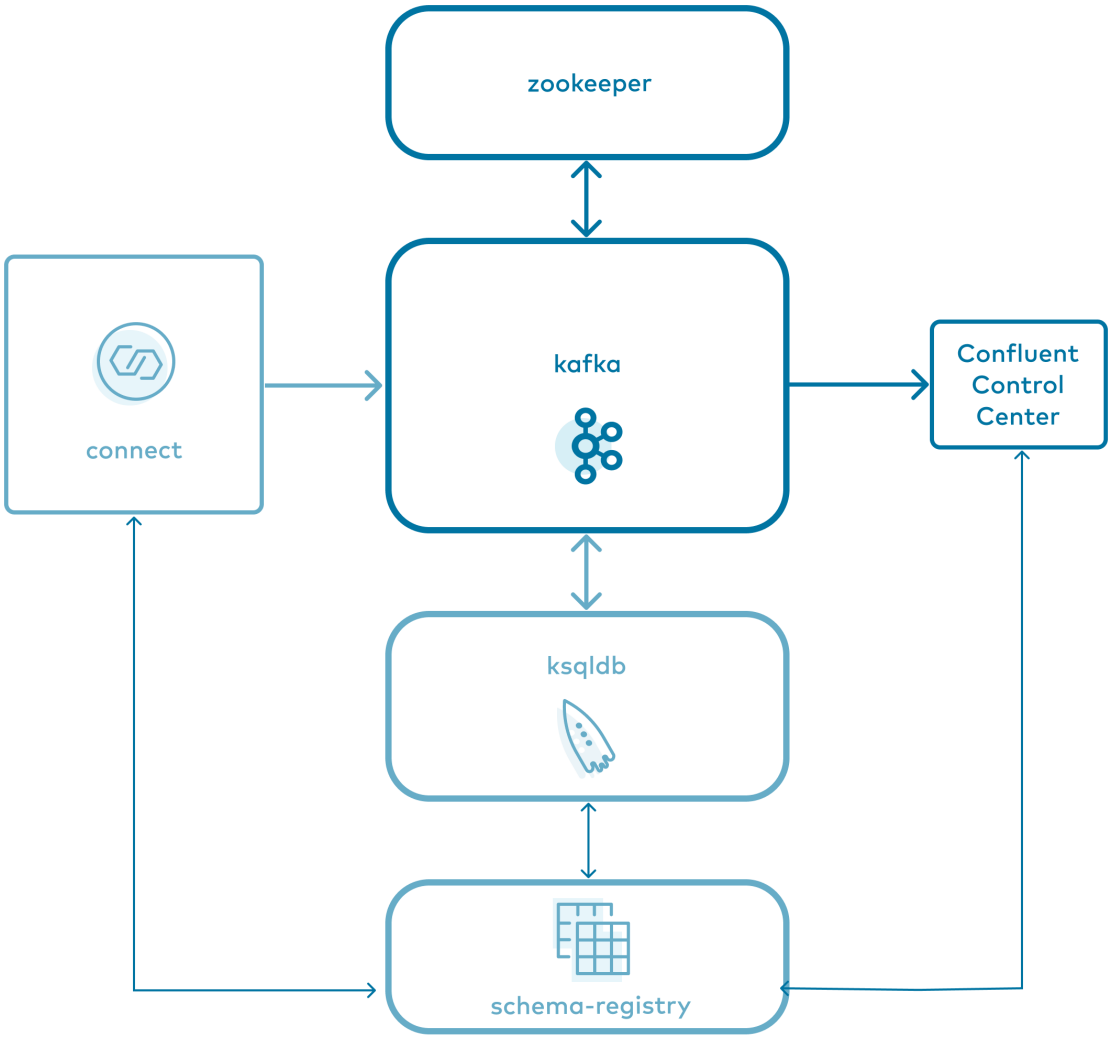
4. Start the Kafka cluster:

```
$ docker-compose up -d zookeeper kafka control-center
```

You should see something similar to this:

```
Creating network "confluent-dev_default" with the default driver
Creating control-center ... done
Creating kafka          ... done
Creating zookeeper      ... done
```

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.



In the first steps of each exercise, you launch the containers needed for the exercise with `docker-compose up`.

If at any time you want to get your environment back to a clean state use `docker-compose down` to end all of your containers. Then return to your last `docker-compose up` to get back to the beginning of an exercise.

Exercises do not need to be completed in order. You can start from the beginning of any exercise at any time.

If you want to completely clear out your docker environment use the script on the VM at `~/docker-nuke.sh`. The nuke script will forcefully end all of your running docker containers.

5. Monitor the cluster with:

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.

```
$ docker-compose ps

     Name                   Command            State
Ports
------------------------------------------------------------------------
--------------------------
control-center   /etc/confluent/docker/run   Up      0.0.0.0:9021-
>9021/tcp
kafka            /etc/confluent/docker/run   Up      0.0.0.0:9092-
>9092/tcp
zookeeper        /etc/confluent/docker/run   Up      0.0.0.0:2181-
>2181/tcp, 2888/tcp, 3888/tcp
```

All services should have `State` equal to `Up`.

6. You can also observe the stats of Docker on your VM:

```
$ docker stats

CONTAINER ID        NAME                CPU %         MEM USAGE /
LIMIT    MEM %    ...
e174ec2aaa51        zookeeper           0.00%         86.88MiB /
7.787GiB  1.09%
2bfac54019a2        kafka               0.01%         450.9MiB /
7.787GiB  5.65%
14c813cf0cf1        control-center      0.01%         376.7MiB /
7.787GiB  4.72%
```

Press `Ctrl+C` to exit the Docker statistics.

Testing the Installation

1. Use the `zookeeper-shell` command to verify that all Brokers have registered with ZooKeeper. You should see a single Broker listed as `[101]` in the last line of the output.

```
$ zookeeper-shell zookeeper:2181 ls /brokers/ids
Connecting to zookeeper:2181

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[101]
```

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.

OPTIONAL: Analyzing the Docker Compose File

1. Open the file `docker-compose.yml` in your editor and:
   a. locate the various services that are listed in the table earlier in this section
   b. note that the container name (e.g. `zookeeper` or `kafka`) are used to resolve a particular service
   c. note how the broker (kafka)
      i. gets a unique ID assigned via environment variable `KAFKA_BROKER_ID`
      ii. defines where to find the ZooKeeper instance

```
KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
```

      iii. sets the replication factor for the offsets topic to 1:

```
KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
```

      iv. configures the broker to send metrics to Confluent Control Center:

```
KAFKA_METRIC_REPORTERS:
"io.confluent.metrics.reporter.ConfluentMetricsReporter"
CONFLUENT_METRICS_REPORTER_BOOTSTRAP_SERVERS: "kafka:9092"
```

   d. note how various services use the environment variable `..._BOOTSTRAP_SERVERS` to define the list of Kafka brokers that serve as bootstrap servers (in our case it's only one instance):

```
..._BOOTSTRAP_SERVERS: kafka:9092
```

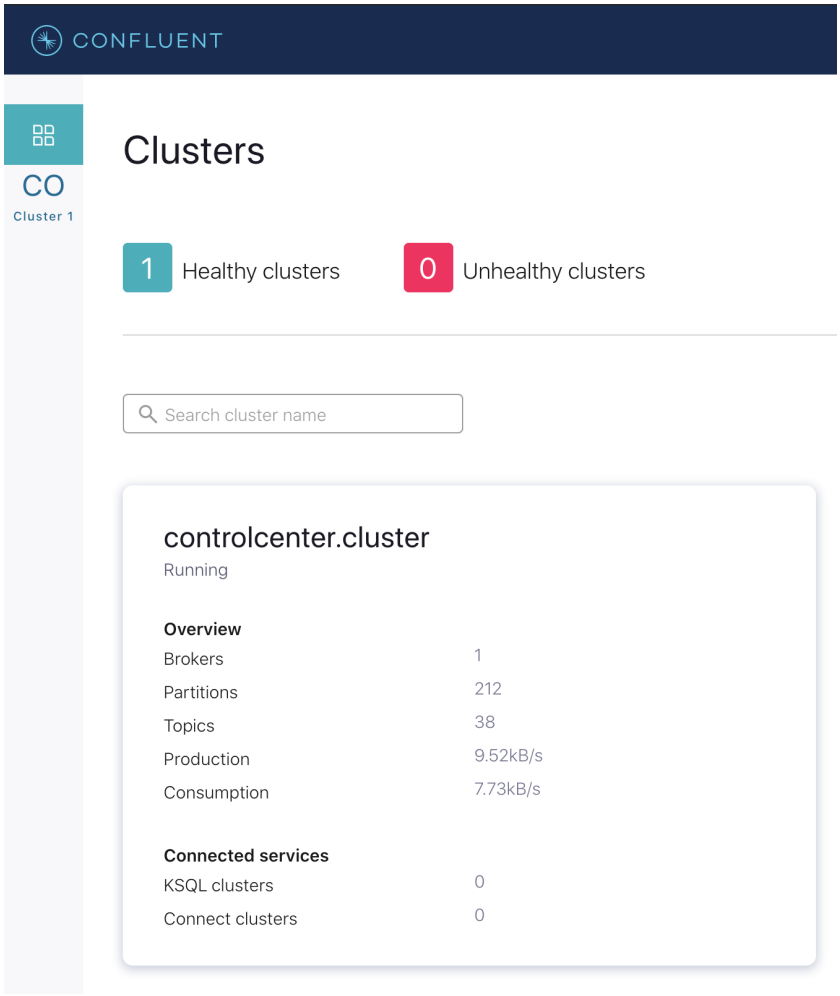
   e. note how e.g. the `connect` service and the `ksqldb-server` service define producer and consumer interceptors that produce data which can be monitored in Confluent Control Center:

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.

```
io.confluent.monitoring.clients.interceptor.MonitoringProducerInte
rceptor
io.confluent.monitoring.clients.interceptor.MonitoringConsumerInte
rceptor
```

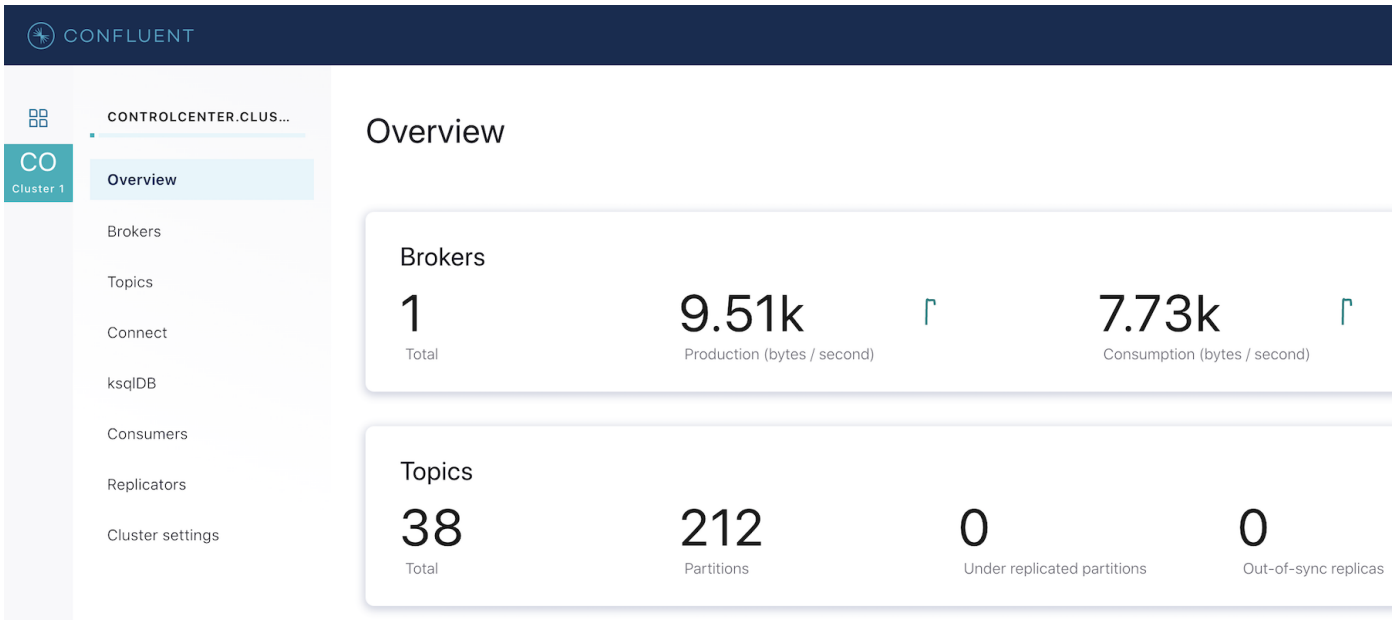
Using Confluent Control Center

1. On your host machine, open a new browser tab in Google Chrome.
2. Navigate to Control Center at the URL http://localhost:9021:



3. Select the cluster **CO** and you will see this:

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.



We have a single broker in our cluster. Also note the other important metrics of our Kafka cluster on this view.

4. Optional: Explore the other tabs of Confluent Control Center, such as **Topics** or **Cluster Settings**.

b. Using Kafka's Command-Line Tools

In this Hands-On Exercise you will start to become familiar with some of Kafka's command-line tools. Specifically you will:

- Use a tool to **create** a topic
- Use a console program to **produce** a message
- Use a console program to **consume** a message
- Use a tool to explore data stored in ZooKeeper

Prerequisites

1. Navigate to the `confluent-dev` folder:

```
$ cd ~/confluent-dev
```

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.

2. Run the Kafka cluster, including Confluent Control Center:

```
$ docker-compose up -d zookeeper kafka control-center
```

If your containers are running from the previous exercise this command will simply tell you each container is up-to-date.

Console Producing and Consuming

Kafka has built-in command line utilities to produce messages to a Topic and read messages from a Topic. These are extremely useful to verify that Kafka is working correctly, and for testing and debugging.

1. Before we can start writing data to a topic in Kafka, we need to first create that topic using a tool called `kafka-topics`. From within the terminal window run the command:

```
$ kafka-topics
```

This will bring up a list of parameters that the `kafka-topics` program can receive. Take a moment to look through the options.

2. Now execute the following command to create the topic `testing`:

```
$ kafka-topics --bootstrap-server kafka:9092 \
    --create \
    --partitions 1 \
    --replication-factor 1 \
    --topic testing
```

We create the topic with a single partition and `replication-factor` of one.

We could have configured Kafka to allow **auto-creation** of topics. In this case we would not have had to do the above step and the topic would automatically be created when the first record is written to it. But this behavior is **strongly discouraged** in production. Always create your topics explicitly!

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.

3. Now let's move on to start writing data into the topic just created. From within the terminal window run the command:

```
$ kafka-console-producer
```

This will bring up a list of parameters that the `kafka-console-producer` program can receive. Take a moment to look through the options. We will discuss many of their meanings later in the course.

4. Run `kafka-console-producer` again with the required arguments:

```
$ kafka-console-producer --bootstrap-server kafka:9092 --topic
testing
```

The tool prompts you with a `>`.

5. At this prompt type:

```
> some data
```

And click **Enter**.

6. Now type:

```
> more data
```

And click **Enter**.

7. Type:

```
> final data
```

And click **Enter**.

8. Now we will use a Consumer to retrieve the data that was produced. Open a new terminal window and run the command:

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.

```
$ kafka-console-consumer
```

This will bring up a list of parameters that the `kafka-console-consumer` can receive. Take a moment to look through the options.

9. Run `kafka-console-consumer` again with the following arguments:

```
$ kafka-console-consumer \
    --bootstrap-server kafka:9092 \
    --from-beginning \
    --topic testing
```

After a short moment you should see all the messages that you produced using `kafka-console-producer` earlier:

```
some data
more data
final data
```

10. Press `Ctrl+D` to exit the `kafka-console-producer` program.
11. Press `Ctrl+C` to exit `kafka-console-consumer`.

OPTIONAL: Working with record keys

By default, `kafka-console-producer` and `kafka-console-consumer` assume null keys. They can also be run with appropriate arguments to write and read keys as well as values.

1. Re-run the Producer with additional arguments to write (key,value) pairs to the Topic:

```
$ kafka-console-producer \
    --bootstrap-server kafka:9092 \
    --topic testing \
    --property parse.key=true \
    --property key.separator=,
```

2. Enter a few values such as:

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.

```
> 1,my first record
> 2,another record
> 3,Kafka is cool
```

3. Press `Ctrl+D` to exit the producer.

4. Now run the **Consumer** with additional arguments to print the key as well as the value:

```
$ kafka-console-consumer \
    --bootstrap-server kafka:9092 \
    --from-beginning \
    --topic testing \
    --property print.key=true

null    some data
null    more data
null    final data
1   my first record
2   another record
3   Kafka is cool
```

   Note the `NULL` values for the first 3 records that we entered earlier...

5. Press `Ctrl+C` to exit the consumer.

The ZooKeeper Shell

1. Kafka's data in ZooKeeper can be accessed using the `zookeeper-shell` command:

```
$ zookeeper-shell zookeeper
Connecting to zookeeper
Welcome to ZooKeeper!
JLine support is disabled

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
```

2. From within the `zookeeper-shell` application, type `ls /` to view the directory structure in ZooKeeper. Note the `/` is required.

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.

```
ls /
[admin, brokers, cluster, config, consumers, controller,
controller_epoch, isr_change_notification, latest_producer_id_block,
log_dir_event_notification, zookeeper]
```

3. Type `ls /brokers` to see this next level of the directory structure.

```
ls /brokers
[ids, seqid, topics]
```

4. Type `ls /brokers/ids` to see the broker ids for the Kafka cluster.

```
ls /brokers/ids
[101]
```

Note the output `[101]`, indicating that we have a single broker with ID `101` in our cluster.

5. Type `get /brokers/ids/101` to see the metadata for broker 101.

```
get /brokers/ids/101
{"listener_security_protocol_map":{"PLAINTEXT":"PLAINTEXT"},"endpoint
s":["PLAINTEXT://kafka:9092"],"jmx_port":-
1,"host":"kafka","timestamp":"1581126250804","port":9092,"version":4}
```

6. Type `get /brokers/topics/testing/partitions/0/state` to see the metadata for partition 0 of topic `testing`.

```
get /brokers/topics/testing/partitions/0/state
{"controller_epoch":1,"leader":101,"version":1,"leader_epoch":0,"isr"
:[101]}
```

Note: During client startup, it requests cluster metadata from a broker in the `bootstrap.servers` list. The output of the two previous commands reflects a bit of this cluster metadata included in the broker response. We will cover this metadata request in more detail later in this course.

7. Press `Ctrl+D` to exit the ZooKeeper shell.

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.

Conclusion

In this lab you have used Kafka command line tools to create a topic, write and read from this topic. Finally you have used the ZooKeeper shell tool to access data stored within ZooKeeper.

STOP HERE. THIS IS THE END OF THE EXERCISE.

© 2014-2022 Confluent, Inc. Do not reproduce without prior written consent.