



# **Confluent Operations Training for Apache Kafka**

## **Exercise Manual**

5.0.0-v1.2.0

# Table of Contents

- Copyright & Trademarks . . . . . 1
- Introduction. . . . . 2
- Preparing the Lab . . . . . 5
- Investigating the Distributed Log . . . . . 11
- Examine Checkpoint Files . . . . . 15
- Examining Topic Partitions . . . . . 16
- Kafka Administrative Tools . . . . . 26
- Modifying Partitions and Viewing Offsets . . . . . 37
- Performance Tuning . . . . . 43
- Defining Over Consumption Trigger and Action . . . . . 50
- Simulating Over Consumption . . . . . 52
- Hands-On Exercise: Securing the Kafka Cluster . . . . . 57
- Generating Certificate . . . . . 57
- Running Kafka Connect . . . . . 67
- Appendix: Running Labs in Docker for Desktop . . . . . 74
- Appendix: Reassigning Partitions in a Topic - Alternate Method . . . . . 74

# Copyright & Trademarks

Copyright © Confluent, Inc. 2014-2019. Privacy Policy | Terms & Conditions.  
Apache, Apache Kafka, Kafka and the Kafka logo are trademarks of the  
Apache Software Foundation

# Introduction

This document provides Hands-On Exercises for the course *Confluent Operations Training for 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
zk-1	ZooKeeper 1
zk-2	ZooKeeper 2
zk-3	ZooKeeper 3
kafka-1	Kafka Broker 1
kafka-2	Kafka Broker 2
kafka-3	Kafka Broker 3
schema-registry	Schema Registry,
connect	Kafka Connect,
ksql-server	KSQL Server,
control-center	Confluent Control Center
base	a container used to run tools against the cluster

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.

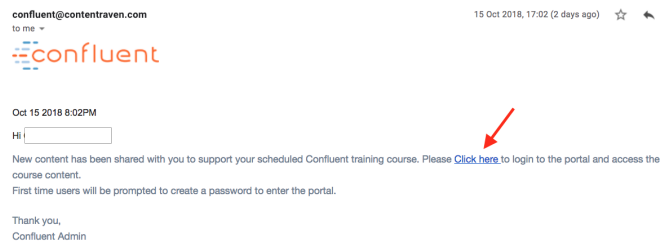


In production, Control Center should be deployed with its own dedicated Kafka cluster, separate from the cluster with production traffic. Using a dedicated metrics cluster is more resilient because it continues to provide system health monitoring even if the production traffic cluster experiences issues.

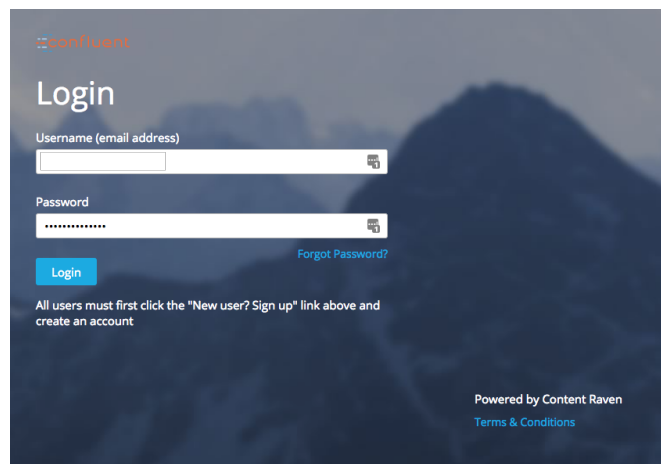
## Accessing your Lab Environment

You will receive an email with a link to your lab environment from Confluent. The labs are running on a VM which is based on Ubuntu 18.04 Desktop edition. On it we have installed Docker Community Edition (CE), Google Chrome and Visual Studio Code.

1. Click the link in the email you received from Confluent:



## 2. Login (or signup) to Content Raven:

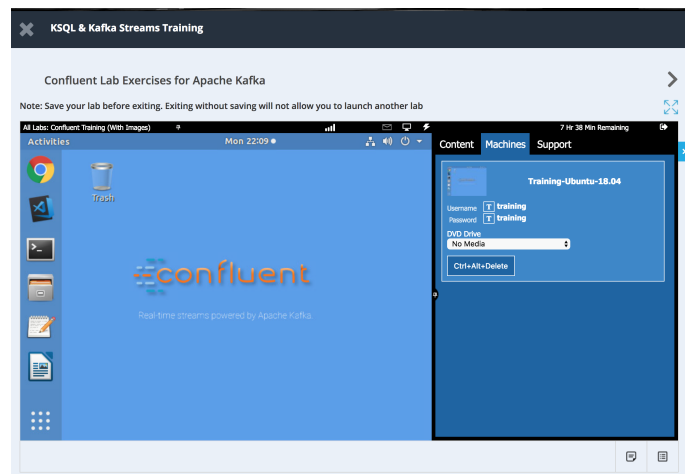


## 3. Access your Learning Path

## 4. Launch the accompanying VM

## 5. Login to the VM using the following credentials:

- **Username:** training
- **Password:** training



Alternatively 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-18.04-apr2019.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](#).

## 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.

## Continued Learning After Class

Once the course ends, the VM will terminate and you will no longer have access to it, but you can continue to use this Exercise Manual for reference. We encourage you to bring up your own test environment, using Docker for Mac or Windows, or Linux and Docker. Options include:

- Download the Confluent Platform: <https://www.confluent.io/download/>
- Run the Confluent Platform Demo including Control Center and KSQL: <https://github.com/confluentinc/cp-demo>
- More information on clustered deployments for testing: <https://docs.confluent.io/current/cp-docker-images/docs/tutorials/clustered-deployment.html>

# Preparing the Lab

1. Download all Docker images needed in this course:



This step is only needed if you're not running the VM in the cloud but rather executing the labs in Docker for Desktop on your computer. This may take a couple of minutes!

```
$ CE_VERSION=5.0.0
for IMAGE in cp-zookeeper cp-enterprise-kafka cp-schema-registry \
             cp-kafka-connect cp-kafka-rest cp-ksql-server \
             cp-enterprise-control-center
do
    docker image pull confluentinc/${IMAGE}:${CE_VERSION}
done

5.0.0: Pulling from confluentinc/cp-zookeeper
Digest: sha256:f5cd4ac4782da1de36f263ecae0ec2d0894c96abe97fb76fd757cc1df9373a77
...
```

2. Create a folder `confluent-ops` in your **home** directory for the labs and navigate to it:

```
$ mkdir -p ~/confluent-ops/data && cd ~/confluent-ops
```



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

3. Download the file `docker-compose.yml` into this folder:

```
$ curl -L https://cnfl.io/docker-compose-apr19 -o docker-compose.yml
```

This file will be used to run and manage the Confluent Platform with all its components.

4. Start the whole cluster with the following simple command:

```
$ docker-compose up -d
```

```
Creating network "confluent-ops_confluent" with the default driver
Creating volume "confluent-ops_data-zk-1" with default driver
Creating volume "confluent-ops_data-zk-2" with default driver
Creating volume "confluent-ops_data-zk-3" with default driver
Creating volume "confluent-ops_data-kafka-1" with default driver
Creating volume "confluent-ops_data-kafka-2" with default driver
Creating volume "confluent-ops_data-kafka-3" with default driver
Creating kafka-2      ... done
Creating schema-registry ... done
Creating ksql-server  ... done
Creating control-center ... done
Creating base         ... done
Creating kafka-3      ... done
Creating zk-1         ... done
Creating zk-3         ... done
Creating connect      ... done
Creating kafka-1      ... done
Creating zk-2         ... done
```

## 5. Monitor the cluster with:

```
$ docker-compose ps
```

Name	Command	State	Ports
base	/bin/sh	Up	8083/tcp, 9092/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
kafka-1	/etc/confluent/docker/run	Up	9092/tcp
kafka-2	/etc/confluent/docker/run	Up	9092/tcp
kafka-3	/etc/confluent/docker/run	Up	9092/tcp
ksql-server	/etc/confluent/docker/run	Up	0.0.0.0:8088->8088/tcp
schema-registry	/etc/confluent/docker/run	Up	8081/tcp
zk-1	/etc/confluent/docker/run	Up	2181/tcp, 2888/tcp, 3888/tcp
zk-2	/etc/confluent/docker/run	Up	2181/tcp, 2888/tcp, 3888/tcp
zk-3	/etc/confluent/docker/run	Up	2181/tcp, 2888/tcp, 3888/tcp

All services should have State equal to Up.

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



```
$ docker stats
```

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	...
877986e719fd	base	0.00%	400KiB / 7.786GiB	0.00%	...
42170ab1d9ec	zk-2	0.25%	66.02MiB / 7.786GiB	0.83%	...
eab22caal51e	connect	1.20%	1.912GiB / 7.786GiB	24.56%	...
e64d1f839bbf	kafka-1	16.30%	396.9MiB / 7.786GiB	4.98%	...
1fabbed67d2b	zk-3	0.24%	65.97MiB / 7.786GiB	0.83%	...
d597e1de0f2c	zk-1	0.22%	59.47MiB / 7.786GiB	0.75%	...
0624d0cc170e	schema-registry	0.50%	236.8MiB / 7.786GiB	2.97%	...
2de9ad477a6b	control-center	70.87%	377.4MiB / 7.786GiB	4.73%	...
30e2509018d0	kafka-3	4.30%	374.6MiB / 7.786GiB	4.70%	...
582b31d8ac68	kafka-2	3.95%	381.9MiB / 7.786GiB	4.79%	...
288d45be137d	ksql-server	0.57%	223.4MiB / 7.786GiB	2.80%	...

## Testing the Installation

1. Use the `zookeeper-shell` command to verify that all Brokers have registered with ZooKeeper. You should see the three Brokers listed as `[101, 102, 103]` in the last line of the output.

```
$ zookeeper-shell zk-1:2181 ls /brokers/ids
Connecting to zk-1:2181

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[101, 102, 103]
```

## 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 alias (e.g. `zk-1` or `kafka-2`) are used to resolve a particular service
  - c. note how each ZooKeeper instance (`zk-1`, `zk-2`, `zk-3`)
    - i. gets a unique ID assigned via environment variable `ZOOKEEPER_SERVER_ID`
    - ii. gets the information about all members of the ensemble:

```
ZOOKEEPER_SERVERS=zk-1:2888:3888;zk-2:2888:3888;zk-3:2888:3888
```

- d. note how each broker (`kafka-1`, `kafka-2`, `kafka-3`)
  - i. gets a unique ID assigned via environment variable `KAFKA_BROKER_ID`
  - ii. defines where to find the ZooKeeper instances

```
KAFKA_ZOOKEEPER_CONNECT: zk-1:2181,zk-2:2181,zk-3:2181
```

- iii. sets the replication factor for the offsets topic to 3:

```
KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 3
```

- 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-1:9092,kafka-2:9092,kafka-3:9092"
```

- e. note how various services use the environment variable `..._BOOTSTRAP_SERVERS` to define the list of Kafka brokers that serve as bootstrap servers:

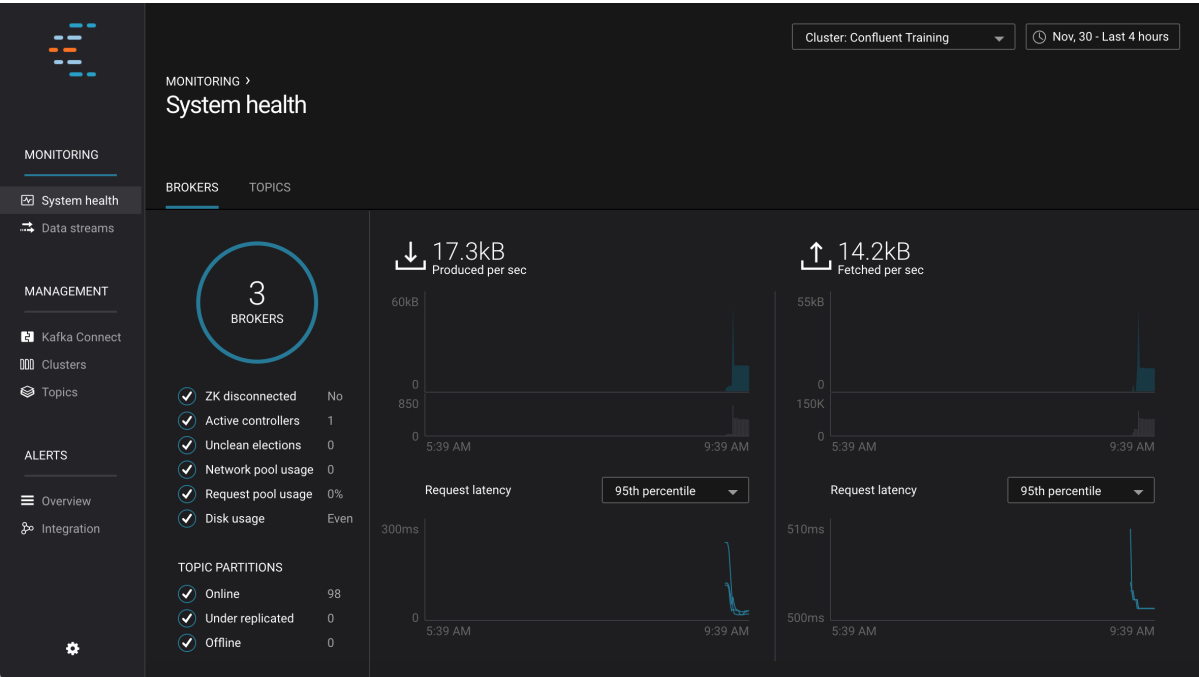
```
..._BOOTSTRAP_SERVERS: kafka-1:9092,kafka-2:9092,kafka-3:9092
```

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

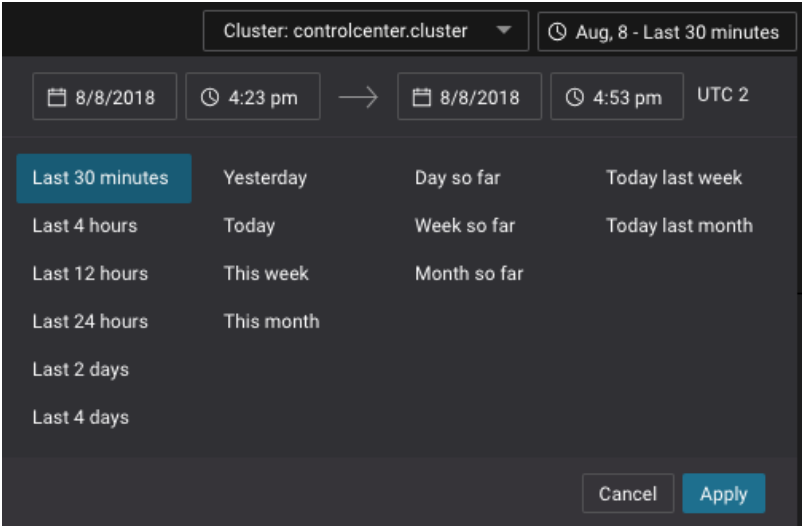
```
io.confluent.monitoring.clients.interceptor.MonitoringProducerInterceptor
io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor
```

## Using Confluent Control Center

1. In the VM, open a new browser tab in Google Chrome.
2. Navigate to the Control Center GUI at <http://localhost:9021>



3. In the Control Center GUI, click on the top right button that shows the current date, and change Last 4 hours to Last 30 minutes.



4. In the Control Center System health landing page, observe the brokers in your cluster. Scroll down to see the table with three brokers: 1, 2, 3.

Broker ID		Throughput		Latency (produce)			
		Bytes in/sec	Bytes out/sec	99.9th %ile	99th %ile	95th %ile	Median
101	...	10.8kB	8.10kB	1825ms	831ms	189ms	5ms
102	...	7.27kB	5.61kB	19s	2368ms	218ms	7ms
103	...	7.10kB	5.52kB	52s	3351ms	280ms	4ms

## Running the Confluent Platform in Kubernetes

For more information on how to run Confluent OSS or Confluent Enterprise in Kubernetes please refer to the following links:

1. Confluent Platform Helm Charts: <https://docs.confluent.io/current/quickstart/cp-helm-charts/docs/index.html>
2. Helm Charts on GitHub: <https://github.com/confluentinc/cp-helm-charts>



**STOP HERE. THIS IS THE END OF THE EXERCISE.**

# Investigating the Distributed Log

In this exercise, you will investigate the distributed log. You will then simulate a Broker failure, and see how to recover the Broker.

## Prerequisites

1. Please make sure you have prepared your environment by following → Preparing the Labs
2. Make sure your Kafka cluster is started, **otherwise** execute this command:

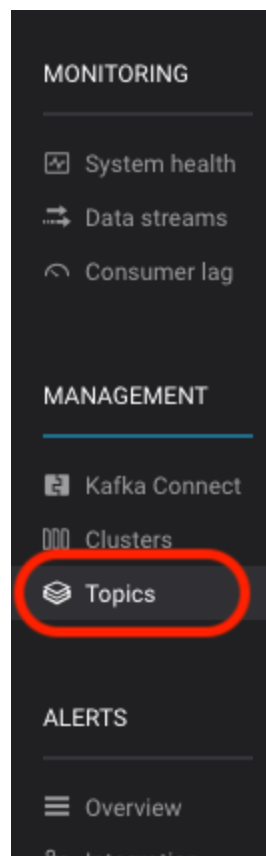
```
$ cd ~/confluent-ops  
$ docker-compose up -d
```

## Observing the Distributed Log

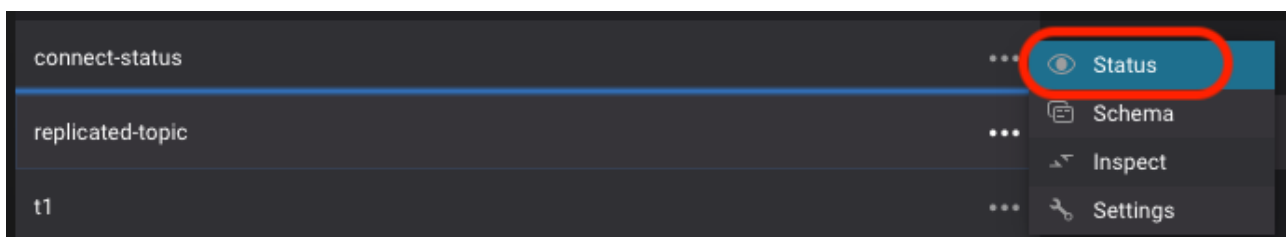
1. Create a new Topic called `replicated-topic` with six Partitions and two replicas:

```
$ kafka-topics \  
  --create \  
  --zookeeper zk-1:2181 \  
  --topic replicated-topic \  
  --partitions 6 \  
  --replication-factor 2  
  
Created topic "replicated-topic".
```

2. View the Topic information to see that it has been created correctly. The exact output may vary depending on which brokers the topic partitions and replicas were assigned.
  - a. Connect to Confluent Control Center. If necessary, open a browser tab to the URL <http://localhost:9021>
  - b. In Confluent Control Center, under the **MANAGEMENT** header, click on **Topics**:





- c. Scroll through the topic list until you find `replicated-topic`. Click on the 3 dots next to the topic name and click on **Status**:

































- d. Notice the Topic's partition list and the corresponding brokers where each partition replica resides. Blue replicas indicate they are in sync. A later exercise where a broker goes offline will result in some orange replicas:

MANAGEMENT > TOPICS > replicated-topic

STATUS SCHEMA INSPECT SETTINGS

 In sync replica  Out of sync replica

Partitions		Replica placement		
id	Replicas	Broker list		
0	 	102,101	  	
1	 	103,102	  	
2	 	101,103	  	
3	 	102,103	  	
4	 	103,101	  	
5	 	101,102	  	

- e. You can also view the same Topic information via Kafka command line tools. In your terminal, from base, describe the Topic:

```
$ kafka-topics \
  --describe \
  --zookeeper zk-1:2181 \
  --topic replicated-topic
```

The output should be similar to this:

```

Topic:replicated-topic PartitionCount:6 ReplicationFactor:2 Configs:
  Topic: replicated-topic Partition: 0 Leader: 102 Replicas: 102,101 Isr:
102,101
  Topic: replicated-topic Partition: 1 Leader: 103 Replicas: 103,102 Isr:
103,102
  Topic: replicated-topic Partition: 2 Leader: 101 Replicas: 101,103 Isr:
101,103
  Topic: replicated-topic Partition: 3 Leader: 102 Replicas: 102,103 Isr:
102,103
  Topic: replicated-topic Partition: 4 Leader: 103 Replicas: 103,101 Isr:
103,101
  Topic: replicated-topic Partition: 5 Leader: 101 Replicas: 101,102 Isr:
101,102

```

3. From base produce some data to send to the Topic `replicated-topic`. Leave this process running until it finishes. The following command line sends 6000 messages, 100 bytes in size, at a rate of 1000 messages/sec:

```

$ kafka-producer-perf-test \
  --topic replicated-topic \
  --num-records 6000 \
  --record-size 100 \
  --throughput 1000 \
  --producer-props bootstrap.servers=kafka-1:9092,kafka-2:9092,kafka-3:9092

```

The output should look similar to this:

```

4999 records sent, 999.8 records/sec (0.10 MB/sec), 5.2 ms avg latency, 298.0 max
latency.
6000 records sent, 998.336106 records/sec (0.10 MB/sec), 4.69 ms avg latency, 298.00 ms
max latency, 2 ms 50th, 25 ms 95th, 35 ms 99th, 46 ms 99.9th.

```

4. Start the console Producer for the same Topic `replicated-topic`:

```

$ kafka-console-producer \
  --broker-list kafka-1:9092,kafka-2:9092,kafka-3:9092 \
  --topic replicated-topic

```

5. At the `>` prompt, type “I heart logs” and press Enter. Add five more messages and then press `Ctrl-d` to exit the console Producer:

```

> I heart logs
> Hello world
> All your base
> Kafka rules
> Don't worry
> Be happy
<Ctrl-d>

```



# Examine Checkpoint Files

For each Broker (kafka-1, kafka-2 and kafka3) examine the contents of the checkpoint files `recovery-point-offset-checkpoint` and `replication-offset-checkpoint` at `/var/lib/kafka/data`. Let's start with the first broker called kafka-1:

1. The next steps will be run on the broker. From your host system, open a new terminal and connect to the broker, here kafka-1:

```
$ cd ~/confluent-ops
$ docker-compose exec kafka-1 /bin/bash
root@kafka-1:/#
```

2. The `recovery-point-offset-checkpoint` file records the point up to which data has been flushed to disks. This is important as, on hard failure, the Broker needs to scan unflushed data, verify the CRC, and truncate the corrupted log.

```
root@kafka-1:/# cat /var/lib/kafka/data/recovery-point-offset-checkpoint | \
  grep replicated-topic

replicated-topic 5 0
replicated-topic 1 0
replicated-topic 4 0
replicated-topic 0 0
```

In the output above, the first number is the Partition number, and the second number is the offset of the last flushing point. In the output above, the offset of the last flushing point is expected to be 0 because the open file segment has not been flushed yet

3. The `replication-offset-checkpoint` file is the offset of the last committed offset. The follower uses this during software upgrades if leader epoch is not stored in messages yet:

```
root@kafka-1:/# cat /var/lib/kafka/data/replication-offset-checkpoint | \
  grep replicated-topic

replicated-topic 5 1001
replicated-topic 1 1001
replicated-topic 4 1001
replicated-topic 0 1001
```

In the output above, the first number is the Partition number, and the second number is the offset of the high water mark. The offset of the high water mark is expected to be 1001 because the number of produced messages, 6006 (= 6000 + 6) was distributed across the six Partitions.

4. Disconnect from the broker container by pressing CTRL-d.

5. Repeat the previous steps for the other two brokers `kafka-2` and `kafka-3`.

## Examining Topic Partitions

On each of the Brokers, examine the contents of one of the data directories for one of the Topic-Partitions. For example, the directory `replicated-topic-0` has log files for Partition 0 of Topic `replicated-topic`:

1. From your host system connect to the broker container, here `kafka-1`:

```
$ cd ~/confluent-ops
$ docker-compose exec kafka-1 /bin/bash
root@kafka-1:/#
```

2. **QUESTION:** The directory `replicated-topic-0` does not exist on all three Brokers in your cluster. Why not?

```
root@kafka-1:/# ls -l /var/lib/kafka/data/replicated-topic-0

-rw-r--r-- 1 root root 10485760 Aug  1 14:35 00000000000000000000.index
-rw-r--r-- 1 root root   165626 Aug  1 14:40 00000000000000000000.log
-rw-r--r-- 1 root root 10485756 Aug  1 14:35 00000000000000000000.timeindex
-rw-r--r-- 1 root root          8 Aug  1 14:35 leader-epoch-checkpoint
```



Remember that a broker may not store every partition for a topic. As shown in the partition lists earlier, each broker only contains a subset of the partitions for `replicated-topic`. If you get an error like the following then the partition is on another broker.

```
ls: cannot access /var/lib/kafka/data/replicated-topic-0: No such file or
directory
```

The data files have the following meaning:

- `.log` is for the log itself
- `.index` maps the message offset to the physical offset within the log file
- `.timeindex` maps the timestamp to an offset
- `leader-epoch-checkpoint` has the leader epoch and its corresponding start offset

3. On the Brokers, view the contents of the file `leader-epoch-checkpoint`:

```
root@kafka-1:/# cat /var/lib/kafka/data/replicated-topic-0/leader-epoch-checkpoint
```

This file tracks the lineage of the Partition leadership by noting which offset starts a new leader generation, in this case leader generation 0 starts at offset 0:

```
0
1
0 0
```

4. On the broker, use the `kafka-dump-log` tool to view details of the `.log` file:

```
root@kafka-1:/# kafka-dump-log \
  --print-data-log \
  --files /var/lib/kafka/data/replicated-topic-0/00000000000000000000.log
```

which should give (shortened):

```
Dumping /var/lib/kafka/data/replicated-topic-0/00000000000000000000.log
Starting offset: 0
offset: 0 position: 0 CreateTime: 1533195082851 invalid: true keysize: -1 valuesize: 100
magic: 2 compresscodec: NONE producerId: -1 producerEpoch: -1 sequence: -1
isTransactional: false headerKeys: [] payload:
SSXVJHPDQDXVCRAS TVBCWVMGNYKR XVZ XKGXTSPSJDGYLUEGQFLAQLOCFLJBEPWFNSOMYARHAOPUFOJHHDXE HXJ
BHWGSMZJGNL
offset: 1 position: 0 CreateTime: 1533195082851 invalid: true keysize: -1 valuesize: 100
magic: 2 compresscodec: NONE producerId: -1 producerEpoch: -1 sequence: -1
isTransactional: false headerKeys: [] payload:
SSXVJHPDQDXVCRAS TVBCWVMGNYKR XVZ XKGXTSPSJDGYLUEGQFLAQLOCFLJBEPWFNSOMYARHAOPUFOJHHDXE HXJ
BHWGSMZJGNL
offset: 2 position: 0 CreateTime: 1533195082851 invalid: true keysize: -1 valuesize: 100
magic: 2 compresscodec: NONE producerId: -1 producerEpoch: -1 sequence: -1
isTransactional: false headerKeys: [] payload:
SSXVJHPDQDXVCRAS TVBCWVMGNYKR XVZ XKGXTSPSJDGYLUEGQFLAQLOCFLJBEPWFNSOMYARHAOPUFOJHHDXE HXJ
BHWGSMZJGNL
...
offset: 1000 position: 165547 CreateTime: 1533134422950 invalid: true keysize: -1
valuesize: 11 magic: 2 compresscodec: NONE producerId: -1 producerEpoch: -1 sequence: -1
isTransactional: false headerKeys: [] payload: Don't worry
```

## 5. Questions:

- Are the offsets in the `.log` file sequential?
- Can you work out what is the offset of the message “All your base” that you entered earlier? If you can’t find the value in the directory `replicated-topic-0`, where else might it be?

6. On the broker, use the `DumpLogSegments` tool to view the offsets in the `.index` file:

```

root@kafka-1:/# kafka-run-class kafka.tools.DumpLogSegments \
  --files /var/lib/kafka/data/replicated-topic-0/00000000000000000000.index

Dumping /var/lib/kafka/data/replicated-topic-0/00000000000000000000.index
offset: 50 position: 5293
offset: 73 position: 9482
offset: 98 position: 13671
...
offset: 952 position: 157448
offset: 977 position: 161637
Mismatches in :/var/lib/kafka/data/replicated-topic-0/00000000000000000000.index
  Index offset: 512, log offset: 511

```

7. Disconnect from the broker (container) by pressing CTRL-d.
8. Repeat the above steps for the other two brokers `kafka-2` and `kafka-3`.

## Taking a Broker Offline

1. First let's define an **action** in Confluent Control Center for when an under replication happens. We will see later why that is important:
  - a. Open Control Center
  - b. Navigate to **ALERTS → Overview**
  - c. Select the **TRIGGERS** tab
  - d. Click the button **Add a trigger**
  - e. Enter **Under Replicated Partitions** as trigger name, select **Cluster** as component type, from the cluster id dropdown select your cluster, select **Under replicated topic partitions** for metric, **Greater than** as condition and finally 0 as value. And then click **Submit**.

ALERTS > OVERVIEW > TRIGGERS >

## Under Replicated Partitions

General

Trigger name\*  
Under Replicated Partitions

Components

Component type\*  
Cluster

Cluster id\*  
controlcenter.cluster [tRToXS-NQoeprmaRRy6a7EQ] x

Criteria

Metric* Under replicated topic partitions	Condition* Greater than
Value* 0	

- f. In the **Trigger saved** confirmation box, when asked to "Set actions to perform when your trigger goes off.", select **Create an action**

✓ Trigger saved

Set actions to perform when your trigger goes off.


Create an action I'll do this later

- g. Add **My action** as Action Name, **Under Replicated Partitions** as Triggers, enter your email address as Recipient email address, **Under replicated partitions** as Subject, and finally **1 Per minute** as Max send rate. Then click **Save action**:

ALERTS > OVERVIEW > ACTIONS >


## New

General


Action Name\*  
My action 

enabled ☒

Triggers

Triggers\*  
Under Replicated Partitions 


Actions

Action\*  
Send email 

Recipient email address\*  
gnschenker@gmail.com

Subject\*  
Under replicated partitions

Max send rate\*  
1

Per minute 

2. From a terminal window discover which of the 3 brokers currently is the controller by using this command:

```
$ zookeeper-shell zk-1:2181 get /controller
Connecting to zk-1:2181

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
{"version":1,"brokerid":101,"timestamp":"1538556180456"}
cZxid = 0x100000466
ctime = Wed Oct 03 08:43:00 UTC 2018
mZxid = 0x100000466
mtime = Wed Oct 03 08:43:00 UTC 2018
pZxid = 0x100000466
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x10004d005af0001
dataLength = 56
numChildren = 0
```

In my particular case broker with ID=101 is the controller.

3. Let's simulate the worst case and stop this broker who is at the same time the controller, and then see what happens. From your host system execute:

```
$ ~/confluent-ops
$ docker-compose stop kafka-1

Stopping kafka-1 ... done
```

Note, it takes a few seconds to stop.

4. Verify that only Brokers 2 and 3 (with ids 102 and 103) are running e.g.:

- a. From your host system use the following command:

```
$ docker-compose ps
```

Name	Command	State	Ports
base	/bin/sh	Up	8083/tcp, 9092/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
kafka-1	/etc/confluent/docker/run	Exit 143	
kafka-2	/etc/confluent/docker/run	Up	9092/tcp
kafka-3	/etc/confluent/docker/run	Up	9092/tcp
ksql-server	/etc/confluent/docker/run	Up	0.0.0.0:8088->8088/tcp
schema-registry	/etc/confluent/docker/run	Up	8081/tcp
zk-1	/etc/confluent/docker/run	Up	2181/tcp, 2888/tcp, 3888/tcp
zk-2	/etc/confluent/docker/run	Up	2181/tcp, 2888/tcp, 3888/tcp
zk-3	/etc/confluent/docker/run	Up	2181/tcp, 2888/tcp, 3888/tcp

and make sure that the State of services kafka-2 and kafka-3 are marked as Up and the one of kafka-1 as Exit 143.

- b. From a terminal window use zookeeper-shell command-line tool to see which Brokers are registered with ZooKeeper:

```
$ zookeeper-shell zk-1:2181 ls /brokers/ids

Connecting to zk-1:2181

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[102, 103]
```

which indeed shows us that currently only broker 2 and 3 are active.

5. From the host system review the server log for Broker 1 (/var/log/kafka/server.log). Look for confirmation of the controlled shutdown succeeding:

```
$ docker-compose logs kafka-1 | tail
```

```
...
kafka-1          | [2018-08-02 08:00:41,766] INFO [Producer clientId=confluent-metrics-
kafka-1          | [2018-08-02 08:00:41,775] INFO [KafkaServer id=1] shut down
completed (kafka.server.KafkaServer)
```



All logs produced by the brokers running inside a container are written to STDOUT and STDERR and reflected in the Docker logs.

6. From the host system review the controller log for Broker 1 (`/var/log/kafka/controller.log`). Look for confirmation of the controller role being released:

```
$ docker-compose logs kafka-1 | grep Resigned
```



```
kafka-1 | [2018-08-02 08:00:40,293] INFO [Controller id=1] Resigned
(kafka.controller.KafkaController)
```

























7. **Question:** If you don't find the above entry in the log of broker 1, where else could it be? Could another broker have been the active controller?
8. Wait up to five minutes and then observe the cluster in Control Center.
  - a. Under **MONITORING** → **System health**, observe the Broker count decrease to 2:
  - b. In the **System health** view as shown above, observe the **Under replicated** Topic Partition count (should be non-zero).
  - c. Under the **MANAGEMENT** header, click on **Topics**. Scroll through the topic list until you find `replicated-topic`. Click on the 3 dots next to the topic name and click on **Status**. Notice the Topic's partition list and the corresponding brokers where each partition replica resides. The replicas assigned to Broker 1 are now orange:



MANAGEMENT > TOPICS > replicated-topic

STATUS SCHEMA INSPECT SETTINGS

 In sync replica  Out of sync replica

Partitions		Replica placement		
id	Replicas	Broker list		
0	 	102,101		
1	 	103,102		
2	 	101,103		
3	 	102,103		
4	 	103,101		
5	 	101,102		

- d. Under the **ALERTS** header, click on **Overview** and view the alert history and notice that the broker down event triggered an alert. It is critical to monitor under replicated partitions to ensure message durability in your Kafka cluster:

ALERTS > OVERVIEW > History

HISTORY TRIGGERS ACTIONS

Time	Trigger	Action Count	
a few seconds ago	Under Replicated Partitions	1	<a href="#">view</a>

- e. Start the console Producer for the same Topic `replicated-topic`:

```
$ kafka-console-producer \
  --broker-list kafka-2:9092,kafka-3:9092 \
  --topic replicated-topic
```

- f. At the > prompt, type six more messages and then press Ctrl-d to exit the console Producer:

```
> Kafka
> Distributed
> Secure
> Real-time
> Scalable
> Fast
<Ctrl-d>
```

9. From your host system view the impact to leader epoch:

- a. On either Broker 2 or Broker 3, view the contents of the file `leader-epoch-checkpoint` in one of the Topic-Partitions subdirectory for which Broker 1 was one of the replicas:

```
$ docker-compose exec kafka-3 \
  cat /var/lib/kafka/data/replicated-topic-0/leader-epoch-checkpoint

0
2
0 0
1 1001
```

- b. Compare the `partitionLeaderEpoch` value in the message produced before Broker 1 stopped and after Broker 1 stopped. The epoch was 0 before Broker 1 stopped, and the epoch is now 1 after Broker 1 stopped:

```
$ docker-compose logs kafka-3 | grep PartitionLeaderEpoch

...
kafka-3          | [2018-08-02 07:31:22,879] INFO Updated PartitionLeaderEpoch. New:
{epoch:0, offset:0}, Current: {epoch:-1, offset:-1} for Partition:replicated-topic-0.
Cache now contains 0 entries. (kafka.server.epoch.LeaderEpochFileCache)
kafka-3          | [2018-08-02 07:31:22,881] INFO Updated PartitionLeaderEpoch. New:
{epoch:0, offset:0}, Current: {epoch:-1, offset:-1} for Partition:replicated-topic-4.
Cache now contains 0 entries. (kafka.server.epoch.LeaderEpochFileCache)
kafka-3          | [2018-08-02 08:27:09,973] INFO Updated PartitionLeaderEpoch. New:
{epoch:1, offset:1001}, Current: {epoch:0, offset:0} for Partition: replicated-topic-
4. Cache now contains 1 entries. (kafka.server.epoch.LeaderEpochFileCache)
kafka-3          | [2018-08-02 08:27:09,977] INFO Updated PartitionLeaderEpoch. New:
{epoch:1, offset:1001}, Current: {epoch:0, offset:0} for Partition: replicated-topic-
0. Cache now contains 1 entries. (kafka.server.epoch.LeaderEpochFileCache)
```

Note the first and last line in the output snippet where the epoch is first set to 0 for Partition `replicated-topic-0` and then it is set to 1 for the same partition.

## Bringing a Broker Online

Bringing a Broker online is as simple as restarting the Broker and letting Kafka automatically rebalance the leaders.

1. From your host system restart broker 1:

```
$ docker-compose start kafka-1
```

2. Wait five minutes and then observe the cluster in the Control Center GUI:
  - a. Under **MONITORING** → **System health**, observe the Broker count, as it returns to 3.
  - b. Observe the **Under replicated** Topic Partition count, as it returns to 0.
  - c. Under the **MANAGEMENT** header, click on **Topics**. Scroll through the topic list until you find `replicated-topic`. Click on the 3 dots next to the topic name and click on **Status**. Notice the Topic's partition list and the corresponding brokers where each partition replica resides. All replicas on Broker 1 are back to blue.



**STOP HERE. THIS IS THE END OF THE EXERCISE.**

# Kafka Administrative Tools

In this exercise, you will delete a Topic, reassign Partitions, and simulate a completely failed Broker.

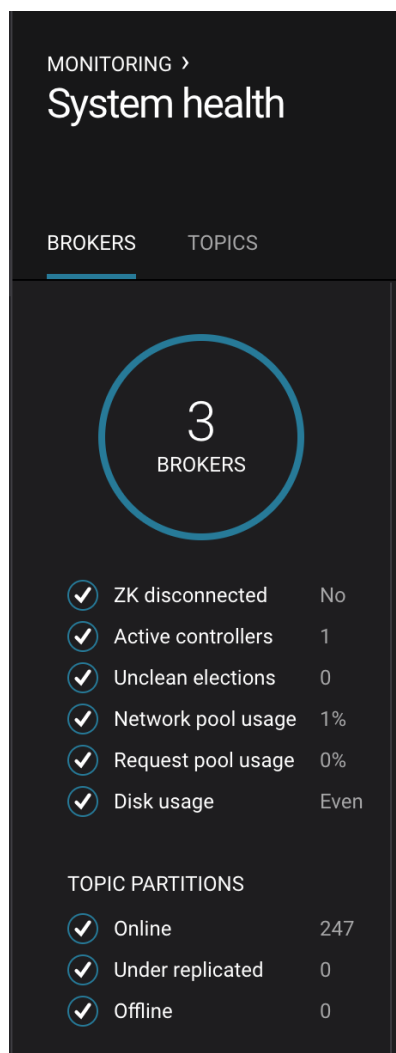
## Prerequisites

1. Please make sure you have prepared your environment by following → [Preparing the Labs](#)
2. Make sure your Kafka cluster is started, **otherwise** execute this command:

```
$ cd ~/confluent-ops  
$ docker-compose up -d
```

## Deleting Topics in the Cluster

1. Connect to Control Center. If necessary, open a browser tab to the URL `http://localhost:9021`.
2. Verify that three Brokers (with ids 101, 102, 103) are running. In the Control Center **System health** view, observe the Broker count is three.



3. Delete the topic `replicated-topic`:

```
$ kafka-topics \
  --zookeeper zk-1:2181 \
  --delete \
  --topic replicated-topic
```

Topic `replicated-topic` is marked for deletion.

Note: This will have no impact if `delete.topic.enable` is not set to true.



When deleting a topic always make sure you don't have any Producers or Consumers running for your topic before deleting it. Otherwise, the Topic will be re-created automatically.

4. In the Control Center, check that the Topic `replicated-topic` is now gone from the list of topics in the Topic Management view.



The topic may not disappear immediately. If necessary wait a few minutes.

## Rebalancing the Cluster

1. Create a new Topic called `moving` with 6 Partitions and 2 replicas, on only Broker 1 and Broker 2 (with IDs 101 and 102):

```
$ kafka-topics \
  --zookeeper zk-1:2181 \
  --create \
  --topic moving \
  --replica-assignment 101:102,102:101,101:102,102:101,101:102,102:101

Created topic "moving".
```

2. Run the command line tool `kafka-producer-perf-test` to produce 2GB of data to Topic `moving`.



Wait until this command has completed before continuing.

```
$ kafka-producer-perf-test \
  --topic moving \
  --num-records 2000000 \
  --record-size 1000 \
  --throughput 1000000000 \
  --producer-props bootstrap.servers=kafka-1:9092,kafka-2:9092
```



3. Verify which brokers contain partitions for the topic `moving`.
  - a. In the Control Center System health view, click on the Topics tab. Scroll down to the topic `moving`. Scroll right to see Segment Size. It shows a little over 2GB, equivalent to what was produced:

























Topic Name		Segment	
		Count	Size
moving	...	6	2.03GB

- b. Click on the 3 dots next to the topic name and click on **View details**. Notice the Topic's partition list and the corresponding brokers where each partition replica resides. They are all on Brokers 1 and 2:

MANAGEMENT › TOPICS ›

## moving

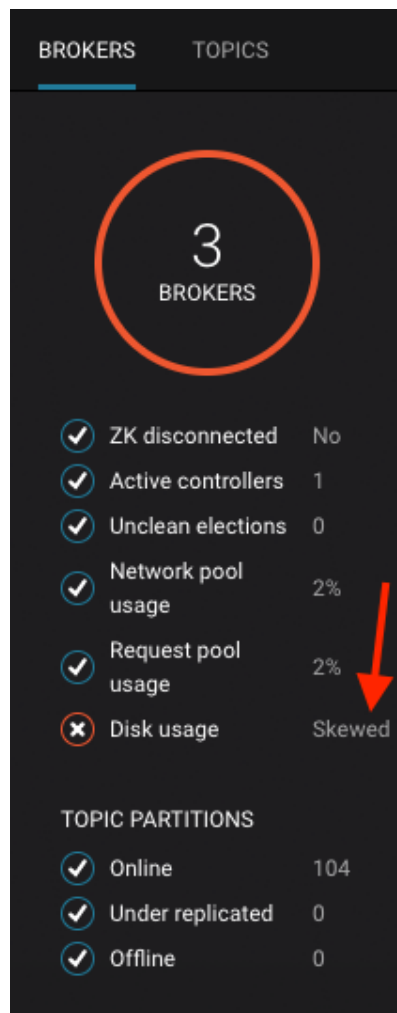
 In sync replica  Out of sync replica

Partitions		Replica placement	
id	Replicas	Broker list	
0	 	101,102	 
1	 	102,101	 
2	 	101,102	 
3	 	102,101	 
4	 	101,102	 
5	 	102,101	 

- c. In the `System health` view, click on the `Brokers` tab. Scroll down to the list of brokers and scroll to the right to see `Segment Size`. Brokers 1 and 2 show a little over 2GB, equivalent to what was produced to topic `moving`. It is significantly more than Broker 3:

Broker ID	Throughput		Latency (produce)				Latency (fetch)				Replication Total Count	Segment Size	R
	Bytes in/sec	Bytes out/sec	99.9th %ile	99th %ile	95th %ile	Median	99.9th %ile	99th %ile	95th %ile	Median			
101	12.0kB	10.6kB	159ms	60ms	19ms	1ms	553ms	514ms	504ms	500ms	226	2.05GB	
102	10.2kB	8.01kB	153ms	46ms	17ms	1ms	576ms	514ms	504ms	501ms	225	2.05GB	
103	8.16kB	6.89kB	373ms	124ms	58ms	13ms	598ms	527ms	504ms	501ms	220	19.7MB	

- d. In the Control Center `System health` view, look at the metric `Disk usage`. Notice that it has changed from `Even` to `Skewed` and is now marked with an `X` to indicate a cluster imbalance:



4. Consume some data from the Topic moving. Leave this consumer running for the duration of the exercise:



```
$ kafka-consumer-perf-test \
  --broker-list kafka-1:9092,kafka-2:9092 \
  --topic moving \
  --group test-group \
  --threads 1 \
  --show-detailed-stats \
  --timeout 1000000 \
  --reporting-interval 5000 \
  --messages 10000000
time, threadId, data.consumed.in.MB, MB.sec, data.consumed.in.nMsg, nMsg.sec,
rebalance.time.ms, fetch.time.ms, fetch.MB.sec, fetch.nMsg.sec
2018-10-03 12:21:58:622, 0, 172.8697, 34.1437, 181267, 35802.2911, 3033, 2030, 85.1575,
89294.0887
2018-10-03 12:22:03:622, 0, 684.5884, 102.3438, 717843, 107315.2000, 0, 5000, 102.3438,
107315.2000
2018-10-03 12:22:08:622, 0, 1249.4726, 112.9768, 1310167, 118464.8000, 0, 5000, 112.9768,
118464.8000
2018-10-03 12:22:13:624, 0, 1646.5025, 79.3742, 1726483, 83229.9080, 0, 5002, 79.3742,
83229.9080
...
```



The next few steps use the Confluent Auto Data Balancer to rebalance the cluster. If you prefer to use the Apache open source tool instead, you may view the Appendix section of this manual called **Reassigning Partitions in a Topic: Alternate Method** (→ [Jump here](#)). Note the open source tool requires users to enumerate every topic to rebalance and does not take into consideration disk utilization per broker.

5. Run the Confluent Auto Data Balancer to rebalance the cluster. This will distribute all cluster partitions, including the topic `moving`, so that the brokers are more evenly utilized. Wait for a few seconds for it to compute the rebalance plan.

```
$ confluent-rebalancer execute \
  --zookeeper zk-1:2181 \
  --metrics-bootstrap-server kafka-1:9092,kafka-2:9092 \
  --throttle 1000000 \
  --verbose \
  --force
```

Computing the rebalance plan (this may take a while) ...  
 You are about to move 4 replica(s) for 4 partitions to 1 broker(s) with total size 1,350.4 MB.  
 The preferred leader for 5 partition(s) will be changed.  
 In total, the assignment for 6 partitions will be changed.  
 The minimum free volume space is set to 20.0%.

Min/max stats for brokers (before -> after):

Type	Leader Count	Replica Count	Size (MB)
Min	77 (id: 103) -> 82 (id: 102)	220 (id: 103) -> 223 (id: 102)	32.8 (id: 103) -> 1,383.2 (id: 101)
Max	86 (id: 101) -> 83 (id: 101)	226 (id: 101) -> 224 (id: 101)	2,058.4 (id: 101) -> 1,383.2 (id: 103)

No racks are defined.

Broker stats (before -> after):

Broker	Leader Count	Replica Count	Size (MB)	Free Space (%)
101	86 -> 83	226 -> 224	2,058.4 -> 1,383.2	77.3 -> 78.4
102	84 -> 82	225 -> 223	2,058.4 -> 1,383.2	77.3 -> 78.4
103	77 -> 82	220 -> 224	32.8 -> 1,383.2	77.3 -> 75.1

The rebalance has been started, run status to check progress.

Warning: You must run the status or finish command periodically, until the rebalance completes, to ensure the throttle is removed. You can also alter the throttle by re-running the execute command passing a new value.

## 6. Observe the configured throttling limits.

```
$ kafka-configs \
  --describe \
  --zookeeper zk-1:2181 \
  --entity-type brokers
```

Configs for brokers '101' are  
 leader.replication.throttled.rate=1000000,follower.replication.throttled.rate=1000000  
 Configs for brokers '102' are  
 leader.replication.throttled.rate=1000000,follower.replication.throttled.rate=1000000  
 Configs for brokers '103' are  
 leader.replication.throttled.rate=1000000,follower.replication.throttled.rate=1000000

## 7. Monitor the progress of the rebalancing:

```
$ confluent-rebalancer status \
  --zookeeper zk-1:2181
Partitions being rebalanced:
Topic moving: 0,1,3,5
```

8. Increase the throttle limit configuration to 1GBps by rerunning the `confluent-rebalancer` command with the new throttle limit:

```
$ confluent-rebalancer execute \
  --zookeeper zk-1:2181 \
  --metrics-bootstrap-server kafka-1:9092,kafka-2:9092 \
  --throttle 1000000000 \
  --verbose
The throttle rate was updated to 1000000000 bytes/sec.
A rebalance is currently in progress for:
Topic moving: 0,1,3,5
```

9. Note the updated throttle limit configuration values.

```
$ kafka-configs \
  --describe \
  --zookeeper zk-1:2181 \
  --entity-type brokers
Configs for brokers '101' are
leader.replication.throttled.rate=1000000000,follower.replication.throttled.rate=1000000000
Configs for brokers '102' are
leader.replication.throttled.rate=1000000000,follower.replication.throttled.rate=1000000000
Configs for brokers '103' are
leader.replication.throttled.rate=1000000000,follower.replication.throttled.rate=1000000000
```

10. Check the status of the Auto Data Balancer again.

- a. Note that it has completed.

```
$ confluent-rebalancer status \
  --zookeeper zk-1:2181
```

No rebalance is currently in progress. If you have called status after a rebalance was started successfully, the rebalance has completed. Run the execute command to check if the cluster is balanced.

- b. Issuing the `status` command above automatically removed the throttle limit configuration from the brokers.

```
$ kafka-configs \
  --describe \
  --zookeeper zk-1:2181 \
  --entity-type brokers

Configs for brokers '101' are
Configs for brokers '102' are
Configs for brokers '103' are
```

11. Wait up to five minutes. In the Control Center System health view, look at the metric `Disk usage`. Notice that it has changed from `Skewed` to `Even`.
12. View the Topic information to see that the topic `moving` has its partitions moved across all three Brokers.

MANAGEMENT > TOPICS > **moving**

■ In sync replica ■ Out of sync replica

Partitions		Replica placement
id	Replicas	Broker list
0	<span style="color: blue;">■</span> <span style="color: blue;">■</span>	103,102 <span style="color: blue;">■</span> <span style="color: blue;">■</span> <span style="color: blue;">■</span>
1	<span style="color: blue;">■</span> <span style="color: blue;">■</span>	102,103 <span style="color: blue;">■</span> <span style="color: blue;">■</span> <span style="color: blue;">■</span>
2	<span style="color: blue;">■</span> <span style="color: blue;">■</span>	101,102 <span style="color: blue;">■</span> <span style="color: blue;">■</span> <span style="color: blue;">■</span>
3	<span style="color: blue;">■</span> <span style="color: blue;">■</span>	103,101 <span style="color: blue;">■</span> <span style="color: blue;">■</span> <span style="color: blue;">■</span>
4	<span style="color: blue;">■</span> <span style="color: blue;">■</span>	101,102 <span style="color: blue;">■</span> <span style="color: blue;">■</span> <span style="color: blue;">■</span>
5	<span style="color: blue;">■</span> <span style="color: blue;">■</span>	103,101 <span style="color: blue;">■</span> <span style="color: blue;">■</span> <span style="color: blue;">■</span>

13. Return to the terminal running the consumer. Press `Ctrl-c` to exit the consumer.

## Simulate a Completely Failed Broker

In previous exercises, the Broker failures were such that the Broker could be simply restarted to recover. Now you will simulate a completely failed Broker which requires a replacement system.

1. In a new terminal from the host observe the logs on Broker 1.

```
$ docker-compose exec kafka-1 ls /var/lib/kafka/data

...
connect-offsets-7
connect-status-0
connect-status-3
log-start-offset-checkpoint
meta.properties
moving-2
moving-3
moving-4
moving-5
recovery-point-offset-checkpoint
replication-offset-checkpoint
```

2. Stop and remove Broker 1 (this also removes the data of Broker 1).

```
$ docker-compose stop kafka-1 && \
  docker-compose rm kafka-1 && \
  docker volume rm confluent-ops_data-kafka-1

Stopping kafka-1 ... done
Going to remove kafka-1
Are you sure? [yN] y
Removing kafka-1 ... done
confluent-ops_data-kafka-1
```

3. In the Confluent Control Center under **System health** verify that broker 1 (with ID=101) is gone.



You may have to wait up to five minutes for the broker to disappear.

4. Restart Broker 1:

```
$ docker-compose up -d

control-center is up-to-date
zk-1 is up-to-date
zk-2 is up-to-date
ksql-cli is up-to-date
ksql-server is up-to-date
zk-3 is up-to-date
kafka-2 is up-to-date
schema-registry is up-to-date
kafka-3 is up-to-date
connect is up-to-date
Creating kafka-1 ... done
```

5. In the Confluent Control Center under **System health** verify that all three Brokers (with ids 101, 102, 103) are running.



You may have to wait up to five minutes for the previously failed broker to reappear.

6. Notice that there are under replicated and offline topic partitions. After a moment of recovery they will return back to zero though.
7. Look at the logs in `/var/lib/kafka/data`. Verify the data is re-replicated back on Broker 1:

```
$ docker-compose exec kafka-1 ls /var/lib/kafka/data
```



**STOP HERE. THIS IS THE END OF THE EXERCISE.**

# Modifying Partitions and Viewing Offsets

In this exercise, you will increase the number of Partitions in a Topic and view offsets in an active Consumer Group.

## Prerequisites

1. Please make sure you have prepared your environment by following → [Preparing the Labs](#)
2. Make sure your Kafka cluster is started, **otherwise** execute this command:

```
$ cd ~/confluent-ops  
$ docker-compose up -d
```

## Increasing the Number of Partitions in a Topic

1. Create a new Topic called `grow-topic` with six Partitions and three replicas:

```
$ kafka-topics \  
  --zookeeper zk-1:2181 \  
  --create \  
  --topic grow-topic \  
  --partitions 6 \  
  --replication-factor 3  
  
Created topic "grow-topic"
```

2. View the `grow-topic` configuration and replica placement in the cluster:
  - a. In the Control Center GUI, view the `grow-topic` Topic configuration from the Management view:

MANAGEMENT > TOPICS > **grow-topic**

■ In sync replica
 ■ Out of sync replica

Partitions		Replica placement	
id	Replicas	Broker list	
0	<span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span>	102,101,103	<span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span>
1	<span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span>	103,102,101	<span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span>
2	<span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span>	101,103,102	<span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span>
3	<span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span>	102,103,101	<span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span>
4	<span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span>	103,101,102	<span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span>
5	<span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span>	101,102,103	<span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span> <span style="color: #0070C0;">■</span>

3. Start the console Producer for Topic `grow-topic`.

```
$ kafka-console-producer \
  --broker-list kafka-1:9092,kafka-2:9092 \
  --topic grow-topic
```

At the `>` prompt, type three messages and then press `Ctrl-d` to exit the console Producer:

```
> KSQL
> Streaming
> Engine
<Ctrl-d>
```

4. Start the console Consumer for Topic `grow-topic` specifying the `group.id` property:



```
$ kafka-console-consumer \
  --consumer-property group.id=test-consumer-group \
  --from-beginning \
  --topic grow-topic \
  --bootstrap-server kafka-1:9092,kafka-2:9092
```

Streaming  
KSQL  
Engine

Leave this Consumer running for during the next step. Think about why it needs to keep running.

5. From another terminal window, describe the Consumer Group `test-consumer-group`:

```
$ kafka-consumer-groups \
  --bootstrap-server kafka-1:9092,kafka-2:9092 \
  --group test-consumer-group \
  --describe
```

TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG	CONSUMER-ID	HOST
CLIENT-ID						
grow-topic	1	1	1	0	consumer-1-a9e...	/192.168.0.6
consumer-1						
grow-topic	4	0	0	0	consumer-1-a9e...	/192.168.0.6
consumer-1						
grow-topic	0	1	1	0	consumer-1-a9e...	/192.168.0.6
consumer-1						
grow-topic	3	1	1	0	consumer-1-a9e...	/192.168.0.6
consumer-1						
grow-topic	2	0	0	0	consumer-1-a9e...	/192.168.0.6
consumer-1						
grow-topic	5	0	0	0	consumer-1-a9e...	/192.168.0.6
consumer-1						

6. Alter the Topic to increase the number of Partitions to 12:

```
$ kafka-topics \
  --zookeeper zk-1:2181 \
  --alter \
  --topic grow-topic \
  --partitions 12
```



WARNING: If partitions are increased for a topic that has a key, the partition logic or ordering of the messages will be affected  
Adding partitions succeeded!









































































7. Wait for a refresh of the metadata, which happens after `metadata.max.age.ms` (default: 5 minutes). Then describe the Consumer Group `test-consumer-group` again. What has changed?

```
$ kafka-consumer-groups \
  --bootstrap-server kafka-1:9092,kafka-2:9092 \
  --group test-consumer-group \
  --describe
```

8. In Control Center, view the `grow-topic` topic configuration from the Management view. You may need to refresh your browser:

MANAGEMENT > TOPICS > **grow-topic**

 In sync replica    Out of sync replica

Partitions		Replica placement
id	Replicas	Broker list
0	  	102,101,103   
1	  	103,102,101   
2	  	101,103,102   
3	  	102,103,101   
4	  	103,101,102   
5	  	101,102,103   
6	  	102,103,101   
7	  	103,101,102   
8	  	101,102,103   
9	  	102,101,103   
10	  	103,102,101   
11	  	101,103,102   

9. In the terminal window where the Consumer is running, press `Ctrl-C` to terminate the process.

## Kafka-based Offset Storage

1. Start the console Producer for Topic `new-topic`:

```
$ kafka-console-producer \
  --broker-list kafka-1:9092,kafka-2:9092 \
  --topic new-topic
```

At the `>` prompt, type some messages into the console. After the first message, you will see a warning message because the Topic `new-topic` did not exist prior to producing to the Topic. Press `Ctrl-d` to return to the command line when you are done:

```
> I
[2018-08-02 14:01:26,083] WARN [Producer clientId=console-producer] Error while fetching
metadata with correlation id 2 : {new-topic=LEADER_NOT_AVAILABLE}
(org.apache.kafka.clients.NetworkClient)
> Love
> Kafka
<Ctrl-d>
```

2. Start the console consumer for Topic `__consumer_offsets` to view the offsets. Leave this running for the duration of the exercise:

```
$ kafka-console-consumer \
  --topic __consumer_offsets \
  --bootstrap-server kafka-1:9092,kafka-2:9092 \
  --formatter "kafka.coordinator.group.GroupMetadataManager$OffsetsMessageFormatter" \
  | grep new-topic
```

3. Start the console Consumer for Topic `new-topic`:

```
$ kafka-console-consumer \
  --group consumer-1 \
  --from-beginning \
  --topic new-topic \
  --bootstrap-server kafka-1:9092,kafka-2:9092
I
Love
Kafka
```

4. Press `Ctrl-c` once your messages have been displayed.
  - a. When reading from the topic `new-topic`, did you see the messages you typed earlier?
  - b. In the other terminal window where you were reading from the topic `__consumer_offsets`, did you see any `OffsetMetadata` specifically for the topic `new-topic` such as shown below?

```
[consumer-1,new-topic,0]::[OffsetMetadata[3,NO_METADATA],CommitTime  
1533218612765,ExpirationTime 1533823412765]  
[consumer-1,new-topic,0]::[OffsetMetadata[3,NO_METADATA],CommitTime  
1533218617762,ExpirationTime 1533823417762]  
...
```

5. If they are currently running, terminate the Producer and Consumers.



**STOP HERE. THIS IS THE END OF THE EXERCISE.**

# Performance Tuning

In this exercise, you will observe Kafka performance and use some of Kafka's settings to optimize the behavior of the Producer and Consumer.

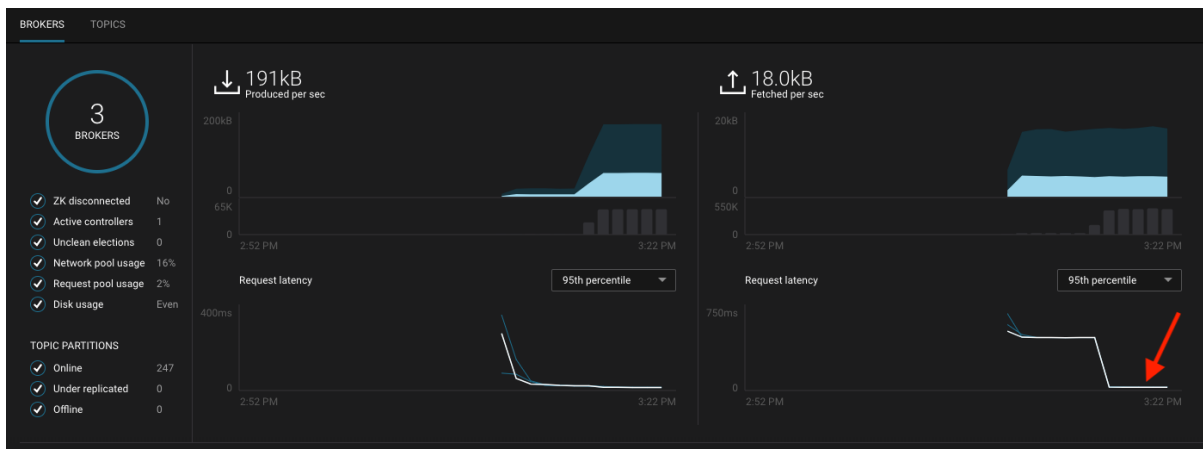
## Prerequisites

1. Please make sure you have prepared your environment by following → Preparing the Labs
2. Make sure your Kafka cluster is started, **otherwise** execute this command:

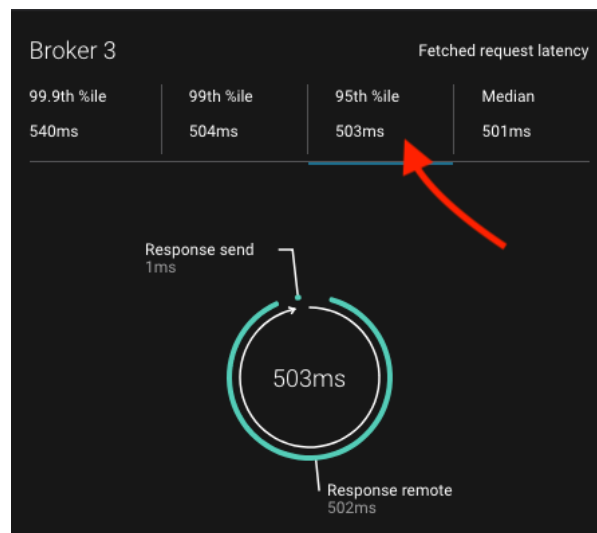
```
$ cd ~/confluent-ops
$ docker-compose up -d
```

## Observing Replica Fetch Times

1. Connect to Control Center. If necessary, open a browser tab to the URL `http://localhost:9021`.
2. In the Control Center `System health` view, observe the fetch request latency for five minutes. Click on one of the lines in the fetch request latency line graph to open the breakdown of times in the request latency lifecycle.  
Note: your exact view may differ, and the y-axis in the request latency graphs may not start at 0.



3. In the fetch request latency breakdown, select the **95th %ile** view.



- Observe that most of the fetch request time is in `Response remote` time waiting for the `replica.fetch.wait.max.ms` timeout. When Producers are not writing records, the fetch request latency will go up to `replica.fetch.wait.max.ms` (default 500ms).
- If you haven't done so in the previous exercise, create a new Topic called `grow-topic` with six Partitions and three replicas:

```
$ kafka-topics \
  --zookeeper zk-1:2181 \
  --create \
  --topic grow-topic \
  --partitions 6 \
  --replication-factor 3

Created topic "grow-topic"
```

- Produce some data to send to the Topic `grow-topic` and let it run:

```
$ kafka-producer-perf-test \
  --topic grow-topic \
  --num-records 1000000 --record-size 100 --throughput 1000 \
  --producer-props bootstrap.servers=kafka-1:9092,kafka-2:9092

4988 records sent, 997.6 records/sec (0.10 MB/sec), 19.3 ms avg latency, 278.0 max
latency.
5004 records sent, 999.2 records/sec (0.10 MB/sec), 10.5 ms avg latency, 83.0 max
latency.
5028 records sent, 1005.4 records/sec (0.10 MB/sec), 12.6 ms avg latency, 123.0 max
latency.
...
```

- Let the producer run for five minutes. In Control Center, in the fetch request latency breakdown, observe the **response remote time** at the 95th %ile, and it drops to a few milliseconds.



8. If it is currently running, terminate the Producer by pressing CTRL-C.

## Observe Broker Load When Sending Messages

1. Create a new Topic called `i-love-logs` with one Partition and one replica.

```
$ kafka-topics \
  --zookeeper zk-1:2181 \
  --create \
  --topic i-love-logs \
  --partitions 1 \
  --replication-factor 1

Created topic "i-love-logs".
```

2. In the Control Center GUI, under the Management header, click on Topics. Scroll through the topic list until you find `i-love-logs`. Click on the 3 dots next to the topic name and click on Status. Take note of which Broker has the Partition. Note: the Broker listed in the output may vary.

MANAGEMENT > TOPICS >

## i-love-logs

■ In sync replica
 ■ Out of sync replica

Partitions		Replica placement
id	Replicas	Broker list
0	<span style="color: #0070C0;">■</span>	102 <span style="color: #0070C0;">■</span>

3. Produce some data to send to the Topic `i-love-logs`. Leave this process running until instructed to end it.

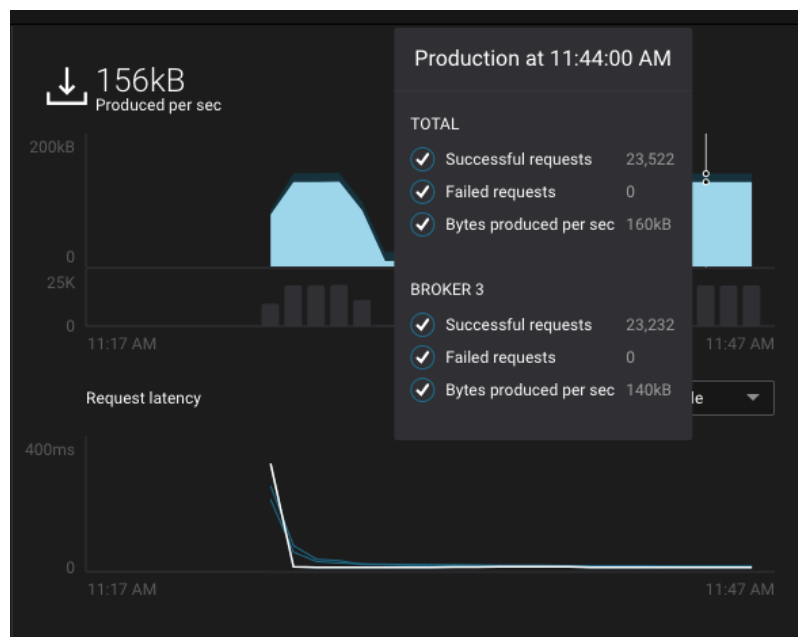
The following command line sends 10 million messages, 100 bytes in size, at a rate of 1000 messages/sec. The `producer-props` environment variable defines interceptors that enable stream monitoring in Control Center.

```
$ NS=io.confluent.monitoring.clients.interceptor && \
kafka-producer-perf-test \
  --topic i-love-logs \
  --num-records 10000000 \
  --record-size 100 \
  --throughput 1000 \
  --producer-props \
    bootstrap.servers=kafka-1:9092,kafka-2:9092 \
    interceptor.classes=${NS}.MonitoringProducerInterceptor

5001 records sent, 1000.0 records/sec (0.10 MB/sec), 7.9 ms avg latency, 371.0 max
latency.
5004 records sent, 1000.6 records/sec (0.10 MB/sec), 1.4 ms avg latency, 31.0 max
latency.
5000 records sent, 1000.0 records/sec (0.10 MB/sec), 1.2 ms avg latency, 7.0 max latency.
...
```

4. Wait for up to five minutes. Return to the Broker tab in the Control Center `System health` view. Select the Broker with the `i-love-logs` Topic in the Broker list at the bottom of the screen. Hover over the charts and pay attention to changes in the `Bytes produced per sec` and `Successful produced requests`. After five minutes, the graphs should look similar to this:





5. Create a file named `consumer.properties` in the `~/confluent-ops/data` folder. Add this content to the file:

```
interceptor.classes=io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor
```

The `consumer.config` file defines interceptors that enable stream monitoring in Control Center.

6. Consume some data from the Topic `i-love-logs` with a consumer group called `cg`. Leave this process running until instructed to end it.

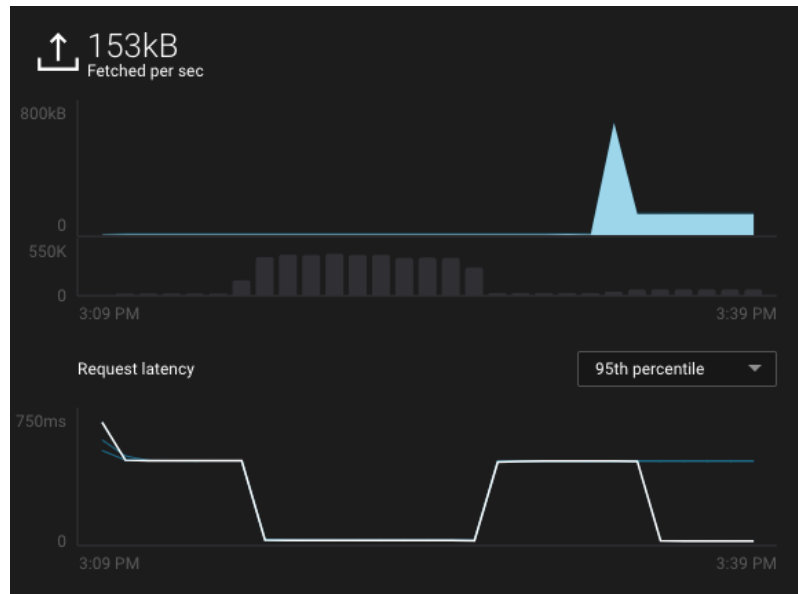
```
$ kafka-consumer-perf-test \
  --broker-list kafka-1:9092,kafka-2:9092 \
  --topic i-love-logs \
  --group cg \
  --messages 10000000 \
  --threads 1 \
  --show-detailed-stats \
  --reporting-interval 5000 \
  --consumer.config data/consumer.properties
```

```
time, threadId, data.consumed.in.MB, MB.sec, data.consumed.in.nMsg, nMsg.sec,
rebalance.time.ms, fetch.time.ms, fetch.MB.sec, fetch.nMsg.sec
2018-08-06 12:24:33:271, 0, 26.2823, 5.2554, 275590, 55106.9786, 3072, 1929, 13.6248,
142866.7703
2018-08-06 12:24:38:271, 0, 118.8769, 18.5189, 1246515, 194185.0000, 0, 5000, 18.5189,
194185.0000
2018-08-06 12:24:43:274, 0, 218.6112, 19.9349, 2292305, 209032.5805, 0, 5003, 19.9349,
209032.5805
...
```



Since the tool `kafka-consumer-perf-test` (contrary to the tool `kafka-producer-perf-test` we used previously) does not support a command line parameter such as `--consumer-props` by which we could pass key-value pairs we had to change our approach and pass a `.properties` file. That's the reason we had to start a new container instance to run that tool.

- Observe the Control Center System health dashboard for five minutes. Pay attention to changes in the Bytes fetched per sec and Successful fetched requests. After five minutes, the graphs should look similar to this:



- Observe the CPU load for the `java` process on the Broker with the partition for the `i-love-logs` Topic (in my case `kafka-3`):

```
$ docker-compose exec kafka-3 top -n10
...
%Cpu(s):  4.4 us,  4.6 sy,   0.0 ni, 90.5 id,   0.2 wa,   0.0 hi,   0.2 si,   0.0 st
KiB Mem:  8164628 total, 8039964 used,  124664 free,  144172 buffers
KiB Swap: 2097148 total,  187988 used, 1909160 free. 1398920 cached Mem

  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM     TIME+ COMMAND
    1 root        20   0 6396520 937384 21204 S   62.9  11.5  13:39.28 java
   207 root        20   0  23652    2616  2288 R    0.0   0.0   0:00.03 top
...
```

Observe CPU utilization until `top` exits. In the output above, the `java` process CPU load is 62.9%.

- Append the entry `fetch.min.bytes=100000` to the file `data/consumer.properties`:

```
$ echo "fetch.min.bytes=100000" >> data/consumer.properties
```

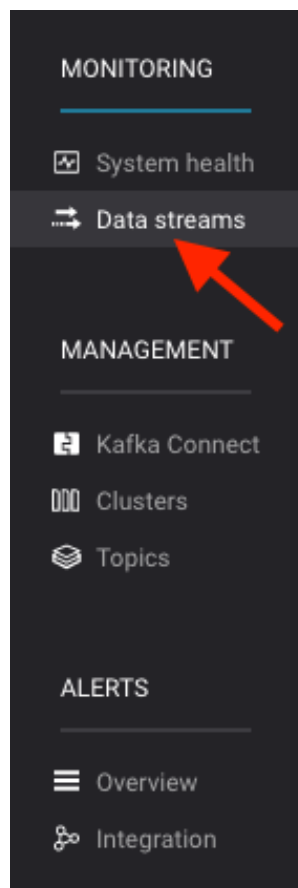
This configuration tells the Broker to wait for larger amounts of data to accumulate before responding to the Consumer. This generally improves throughput and reduces load on the Broker but can increase latency

10. From this new terminal consume some data from for the Topic `i-love-logs` with a new consumer group called `cg-fetch-min`. Leave this process running until instructed to end it.

```
$ kafka-consumer-perf-test \
  --broker-list kafka-1:9092,kafka-2:9092 \
  --topic i-love-logs \
  --group cg-fetch-min \
  --messages 10000000 \
  --threads 1 \
  --show-detailed-stats \
  --reporting-interval 5000 \
  --consumer.config data/consumer.properties
```

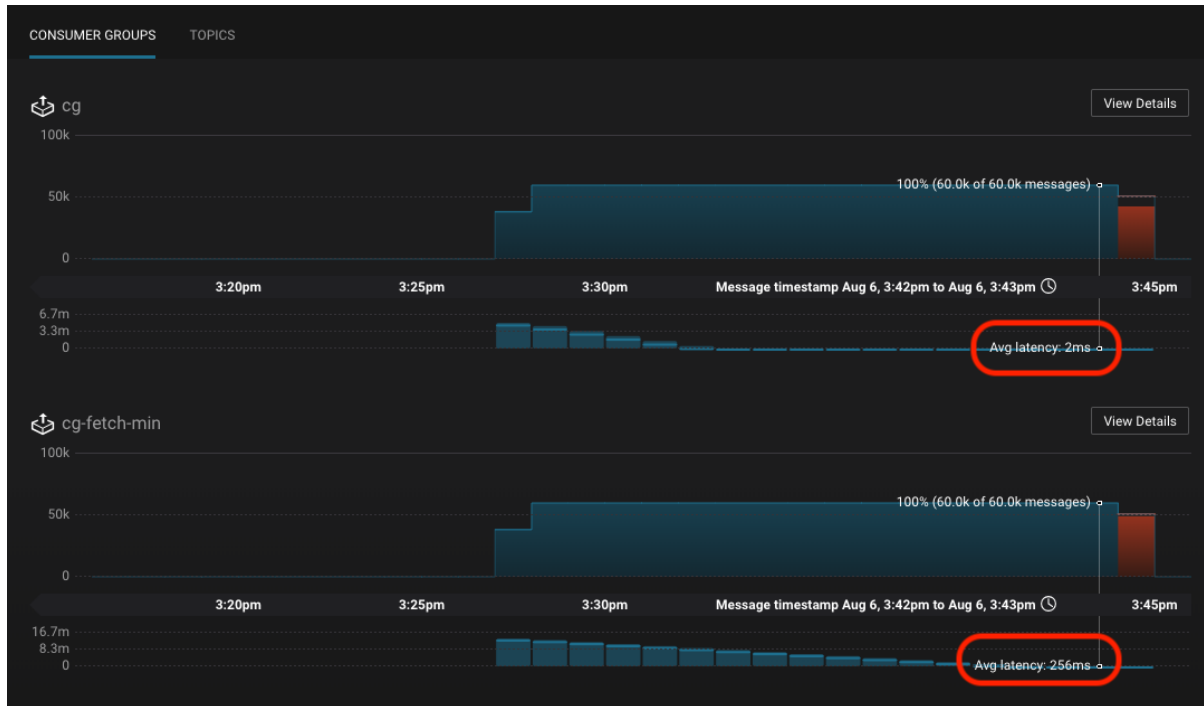
11. Compare the relative latency for these two consumer groups `cg` (with default consumer group configuration) and `cg-fetch-min` (with `fetch.min.bytes=100000`).

- a. In the Control Center GUI, click on `Data streams`. If at first the graphs don't appear, wait a minute and refresh your browser.



- b. Scroll down to view the consumer groups `cg` and `cg-fetch-min`. Hover over the bar graphs to see the messages produced and latencies for each consumer group. It is critical to monitor latency for real-time

applications where consumers should consume produced messages with as low latency as possible.



## 12. Questions:

- Which consumer group has a higher consumption rate (number of messages consumed per time bucket), or are they equal?
- Which consumer group has a higher latency?

13. Go to the terminal window with the first perf Consumer that was using consumer group `cg` and press `Ctrl-c` to end the process. The other Consumer that was using consumer group `cg-fetch-min` should still be running.

14. Observe the CPU load again for the `java` process on the Broker:

```
$ docker-compose exec kafka-3 top -n10
...
  PID USER      PR  NI   VIRT    RES    SHR S  %CPU  %MEM    TIME+  COMMAND
    1 root        20   0 6398568 1.026g 16696 S   36.8  13.2   19:27.56 java
...
```

Why does the CPU load decrease with consumer group `cg-fetch-min` as compared to consumer group `cg`?

## Defining Over Consumption Trigger and Action

- For the following exercise (over consumption) we need to define a trigger and corresponding action in Confluent Control Center:
  - Open Confluent Control Center

- b. Navigate to **ALERTS → Overview**
- c. Select the **TRIGGERS** tab
- d. Click the button **New trigger**
- e. Enter **Over consumption** as "Trigger name", select **Consumer group** as "Component type", select **cg** from the "Consumer group name" dropdown, select **Consumption difference** for metric, **Greater than** as condition and finally 0 as value. And then click **Submit**.

ALERTS > OVERVIEW > TRIGGERS >

## New

General

Trigger name\*  
Over consumption

Components

Component type\*  
Consumer group

Consumer group name\*  
cg [f6h9Pa4uSjyxikfX2v87tQ]

Criteria

Metric* Consumption difference	Value* 0
Condition* Greater than	Buffer (seconds)* 60

- f. In the **Trigger saved** confirmation box, when asked to "Set actions to perform when your trigger goes off.", select **Create an action**

✓ Trigger saved

Set actions to perform when your trigger goes off.

Create an action I'll do this later

- g. Add **Over consumption** as "Action Name", **Over consumption** as "Triggers", enter you email address as "Recipient email address", **Over consumption** as "Subject", and finally **1 Per hour** as Max send rate. The click **Save action**:

ALERTS > OVERVIEW > ACTIONS >

## New

General

Action Name\*  
Over consumption

enabled ☒

Triggers

Triggers\*  
Over consumption x

Actions

Action\*  
Send email

Recipient email address\*  
admin@confluent.io

Subject\*  
Over consumption

Max send rate\*  
1

Per hour

## Simulating Over Consumption

1. Reset the offsets of consumer group `cg` using this command:

```
$ kafka-consumer-groups \
  --bootstrap-server kafka-1:9092,kafka-2:9092 \
  --group cg \
  --reset-offsets \
  --to-earliest \
  --all-topics \
  --execute
```

TOPIC	PARTITION	NEW-OFFSET
i-love-logs	0	0

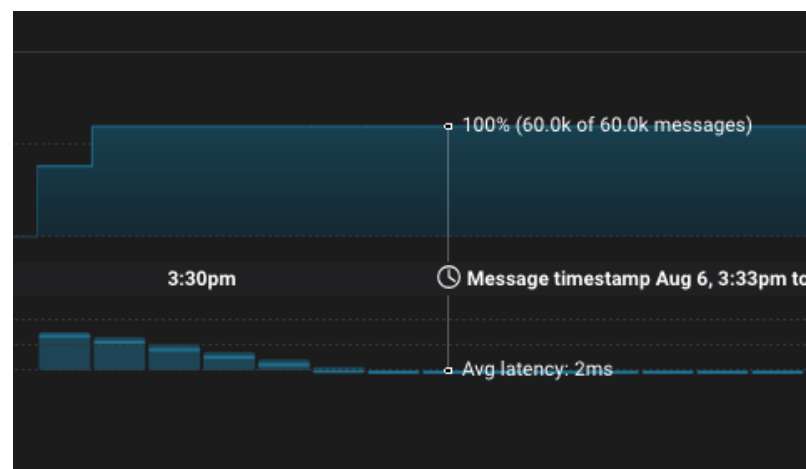
2. Restart the first perf Consumer that was using consumer group `cg`, which restarts consuming messages from the topic `i-love-logs` at the very beginning (we have reset the offset just before), thereby processing the same messages again. This is a good simulation of consumer group overconsumption:

```
$ kafka-consumer-perf-test \
  --broker-list kafka-1:9092,kafka-2:9092 \
  --topic i-love-logs \
  --group cg \
  --messages 10000000 \
  --threads 1 \
  --show-detailed-stats \
  --reporting-interval 5000 \
  --consumer.config data/consumer.properties

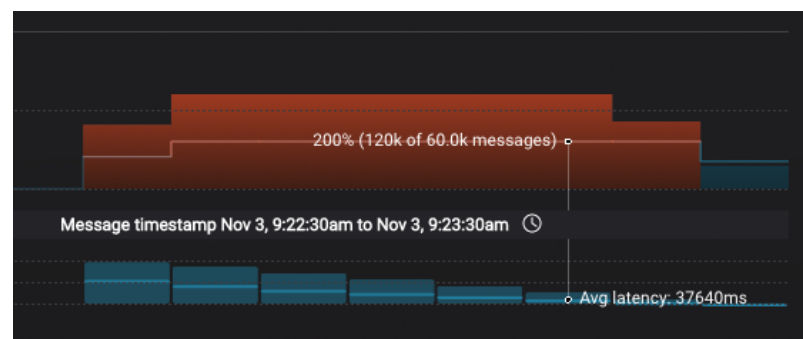
time, threadId, data.consumed.in.MB, MB.sec, data.consumed.in.nMsg, nMsg.sec,
rebalance.time.ms, fetch.time.ms, fetch.MB.sec, fetch.nMsg.sec
2018-08-06 13:05:16:033, 0, 29.0271, 5.8043, 304371, 60862.0276, 3017, 1984, 14.6306,
153412.8024
2018-08-06 13:05:21:351, 0, 33.7141, 0.8814, 353518, 9241.6322, 0, 5318, 0.8814,
9241.6322
2018-08-06 13:05:26:374, 0, 34.1927, 0.0953, 358536, 999.0046, 0, 5023, 0.0953, 999.0046
...
```

### 3. Observe the cluster in the Control Center GUI for a couple minutes:

- Click on Data streams header and observe the consumer group `cg` for a few minutes. This is what it looked like before:

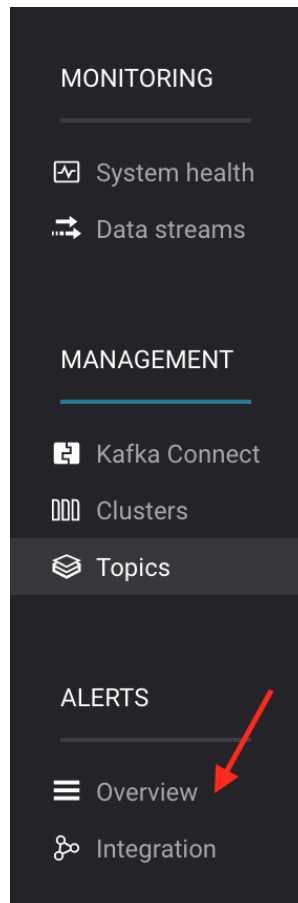


- Observe the consumer group `cg` data streams monitoring change after a few minutes:



Observe that:

- The consumption bar changed from blue to red
  - The consumption bar is double the height of the expected consumption line
  - The latency jumped because the latency calculation is relative to produce time
- c. Under the Alerts header, click on Overview.



- d. View the alert history and notice that the over consumption event triggered an alert. It is critical to monitor applications for message consumption to know how your applications are behaving. Overconsumption may happen intentionally, or it may happen unintentionally, for example if an application crashes before committing processed messages.

ALERTS >  
Overview

Time	Trigger	Action Count	
23 minutes ago	Over consumption	1	<a href="#">view</a>

4. For the Producer and both Consumers, go to the respective terminal window and press `Ctrl-c` to end each process.



## Performance Tuning the Producer

1. Create a new Topic called `performance` with six Partitions and three replicas.

```
$ kafka-topics \
  --zookeeper zk-1:2181 \
  --create \
  --topic performance \
  --partitions 6 \
  --replication-factor 3

Created topic "performance".
```

2. Run Producer performance tests using different permutations of the following configuration parameters:

- `acks` (Default: 1)
- `batch.size` (Default: 16384)
- `linger.ms` (Default: 0)

Use the following command line and replace `<VARIABLE_HERE>` with the specified options in the table below:

```
$ kafka-producer-perf-test \
  --topic performance \
  --num-records 10000000 \
  --record-size 100 \
  --throughput 10000000 \
  --producer-props bootstrap.servers=kafka-1:9092,kafka-2:9092 \
  <VARIABLE_HERE>
```

Let each test run until it terminates and provides a performance summary. Record the throughput and latency results in the following table.

Variables	Throughput (MB/sec)	Latency (ms avg)
<code>acks=1</code>		
<code>acks=all</code>		
<code>acks=1 batch.size=1000000 linger.ms=1000</code>		
<code>acks=all batch.size=1000000 linger.ms=1000</code>		

3. Questions:

- a. Do you get better throughput with `acks=1` or `acks=all`? What is the ratio between them?
- b. Do you get better latency with `acks=1` or `acks=all`? Why?
- c. What happens to Producer throughput and latency if you set `batch.size` to be 1MB and `linger.ms` to

1000?

## Cleanup

1. Run the following commands to stop the cluster and clean up unused resources:

```
$ docker-compose down -v
```



**STOP HERE. THIS IS THE END OF THE EXERCISE.**

# Hands-On Exercise: Securing the Kafka Cluster

In this exercise, you will configure one-way SSL authentication, two-way SSL authentication, and explore the SSL performance impact.

## Prerequisites

1. Please make sure you have prepared your environment by following → Preparing the Labs
2. Make sure your Kafka cluster you used in the previous exercises is stopped, **otherwise** execute this command:

```
$ cd ~/confluent-ops
$ docker-compose down -v
```

3. In you labs folder `~/confluent-ops` create a folder `secure-cluster` and navigate to it:

```
$ mkdir -p ~/confluent-ops/secure-cluster && cd ~/confluent-ops/secure-cluster
```

4. In the folder `secure-cluster` create a new file `docker-compose.yml`. From within the `secure-cluster` folder execute this command:

```
$ curl -L https://cnfl.io/docker-compose-ssl-apr19 -o docker-compose.yml
```

## Generating Certificate

Next, we generate all the necessary certificates and credentials that are used to create a secure (3-node) Kafka cluster. Note that we need to run the script that will generate the certs in a container since the script expects certain Java tools to be available, but our lab environment does not have Java installed. Thus we will just use a Java based container to address this problem as you will see further down:

1. In the project folder `secure-cluster` create a subfolder `scripts/security` and navigate to it:

```
$ mkdir -p scripts/security && cd scripts/security
```

2. From within the that folder download the script `certs-create.sh`:

```
$ curl -L https://cnfl.io/securing-kafka-certs-apr19 -o certs-create.sh
```

Open the file and analyze the code. If necessary discuss it with your peers.

3. Make the script you just downloaded executable:

```
$ chmod +x ./certs-create.sh
```

- Execute the following command to run the script, generating the desired certificates in the mounted folder:

```
$ ./certs-create.sh
```

After the script has ended browse through the files that were generated in folder `~/confluent-ops/secure-cluster/scripts/security`.

## Enabling SSL on the Brokers

- Open the file `~/confluent-ops/secure-cluster/docker-compose.yml` in your editor and note the following environment variables (in addition to the existing ones) that have been added to each of the three brokers (kafka-1, kafka-2 and kafka-3), e.g. for kafka-1 they are:

```
KAFKA_ADVERTISED_LISTENERS: SSL://kafka-1:9093,PLAINTEXT://kafka-1:9092
KAFKA_SSL_KEYSTORE_FILENAME: kafka.kafka-1.keystore.jks
KAFKA_SSL_KEYSTORE_CREDENTIALS: kafka-1_keystore_creds
KAFKA_SSL_KEY_CREDENTIALS: kafka_sslkey_creds
KAFKA_SSL_TRUSTSTORE_FILENAME: kafka.kafka-1.truststore.jks
KAFKA_SSL_TRUSTSTORE_CREDENTIALS: kafka-1_truststore_creds
KAFKA_SSL_ENDPOINT_IDENTIFICATION_ALGORITHM: "HTTPS"
```



The variable names shown above might at a first sight not look familiar. Your trainer has shown you the variable names in the configuration files for e.g. the brokers to be such as `ssl.keystore.filename` or `ssl.truststore.credentials`. A startup script inside each container (such as the ones for the brokers) converts the Docker variable names by a standard algorithm into the Kafka specific variable names and adds them to the respective configuration file. Upper case is converted to lowercase and underscores are converted to periods. Furthermore each component (Kafka, REST Proxy, Kafka Connect, etc.) has a well defined prefix that is removed from the Docker variable. In the above case it is `KAFKA_`.

- Each of the three brokers also has a volume mapping defined for the certificates:

```
volumes:
  ...
  - $PWD/scripts/security:/etc/kafka/secrets
```

- Please answer the following questions. Discuss with your peers if necessary:
  - Why is `KAFKA_LISTENERS` configured for both `SSL` and `PLAINTEXT`?
  - What would happen if `KAFKA_LISTENERS` were configured for just `SSL`?
  - What does `KAFKA_SSL_ENDPOINT_IDENTIFICATION_ALGORITHM` do?

- From within the folder `secure-cluster` run the cluster using:

```
$ cd ~/confluent-ops/secure-cluster
$ docker-compose up -d
```

and monitor the progress with:

```
$ watch docker-compose ps
```

Name	Command	State	Ports
control-center	/etc/confluent/docker/run	Up	0.0.0.0:9021->9021/tcp
kafka-1	/etc/confluent/docker/run	Up	9092/tcp
kafka-2	/etc/confluent/docker/run	Up	9092/tcp
kafka-3	/etc/confluent/docker/run	Up	9092/tcp
schema-registry	/etc/confluent/docker/run	Up	8081/tcp
zk-1	/etc/confluent/docker/run	Up	2181/tcp, 2888/tcp, 3888/tcp
zk-2	/etc/confluent/docker/run	Up	2181/tcp, 2888/tcp, 3888/tcp
zk-3	/etc/confluent/docker/run	Up	2181/tcp, 2888/tcp, 3888/tcp



Press CTRL-C to stop watching.

## Verifying SSL is Working

- From a terminal window verify that the keystore and truststore of each Broker are setup properly. The `openssl` command below should return a key and certificate:

```
$ openssl s_client -connect kafka-1:9093 -tls1

CONNECTED(00000003)
depth=1 CN = cal.test.confluent.io, OU = TEST, O = CONFLUENT, L = PaloAlto, C = US
verify error:num=19:self signed certificate in certificate chain
verify return:0
139729781544592:error:1408E0F4:SSL routines:SSL3_GET_MESSAGE:unexpected
message:s3_both.c:477:
---
Certificate chain
 0 s:/C=US/ST=Ca/L=PaloAlto/O=CONFLUENT/OU=TEST/CN=kafka-1
  i:/CN=cal.test.confluent.io/OU=TEST/O=CONFLUENT/L=PaloAlto/C=US
 1 s:/CN=cal.test.confluent.io/OU=TEST/O=CONFLUENT/L=PaloAlto/C=US
  i:/CN=cal.test.confluent.io/OU=TEST/O=CONFLUENT/L=PaloAlto/C=US
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDQTCCAikCCQDHiHJcd4/LAzANBgkqhkiG9w0BAQUFADBjMR4wHAYDVQQDDBVj
YTEudGVzdC5jb25mbHVlbnQuaW8xDTALBgNVBAsMBFRFU1QxEjAQBGNVBAoMCUNP
TkZMVUVVOVDERMA8GA1UEBwwIUUGFsb0FsdG8xCzAJBgNVBAYTAlVTMB4XDTE4MDgw
...

```



Press CTRL-C to stop the above command.

2. Repeat the same for the other two brokers:

```
$ openssl s_client -connect kafka-2:9093 -tls1
$ openssl s_client -connect kafka-3:9093 -tls1
```

If instead of seeing the certificate you see the output below, then the Broker is not properly setup for SSL. Please review the steps above. We simulate this here by providing the wrong port:



```
$ openssl s_client -connect kafka-1:9094 -tls1

socket: Connection refused
connect:errno=111
```

3. In the code editor create a file named `producer_ssl.properties` in the folder `~/confluent-ops/secure-cluster/scripts/security`. Add this content to the file:

```
security.protocol=SSL
ssl.truststore.location=scripts/security/kafka.client.truststore.jks
ssl.truststore.password=confluent
ssl.key.password=confluent
```

4. Send messages to the cluster via the SSL port.

a. Create a new Topic called `ssl-topic` with one Partition and three replicas:

```
$ kafka-topics \
  --zookeeper zk-1:2181 \
  --create \
  --topic ssl-topic \
  --partitions 1 \
  --replication-factor 3

Created topic "ssl-topic".
```

b. Start the console Producer for Topic `ssl-topic` using the `producer_ssl.properties` file. Here the client is connecting to the SSL port 9093, whereas in the previous exercise the client was connecting to the PLAINTEXT port 9092:

```
$ kafka-console-producer \
  --broker-list kafka-1:9093,kafka-2:9093 \
  --topic ssl-topic \
  --producer.config scripts/security/producer_ssl.properties
```

- c. At the `>` prompt, type “Security is good” and press Enter. Add a couple more messages and then press `Ctrl-d` to exit the console Producer.

```
> Security is good
> Certificates
> Keys
<Ctrl-d>
```

5. Again in your code editor create a file named `consumer_ssl.properties` in the folder `~/confluent-ops/secure-cluster/scripts/security` to provide SSL configuration parameters when clients connect to the cluster. Add this content to the file:

```
security.protocol=SSL
ssl.truststore.location=scripts/security/kafka.client.truststore.jks
ssl.truststore.password=confluent
ssl.key.password=confluent
```



This is the same content we used for the producer above.

6. Start the console Consumer for Topic `ssl-topic` using the `consumer_ssl.properties` file. You should see the messages you typed above. Press `Ctrl-c` once your messages have been displayed.

```
$ cd ~/confluent-ops/secure-cluster
$ kafka-console-consumer \
  --consumer.config scripts/security/consumer_ssl.properties \
  --from-beginning \
  --topic ssl-topic \
  --bootstrap-server kafka-1:9093,kafka-2:9093

Security is good
Certificates
Keys
```

Press `CTRL-c` to stop the consumer.

## Enabling SSL Two-Way Authentication

1. In the `docker-compose.yml` file in folder `~/confluent-ops/secure-cluster`, for each of the three brokers uncomment the following environment variable:

```
KAFKA_SSL_CLIENT_AUTH: "required"
```

2. Restart all 3 brokers by executing:

```
$ cd ~/confluent-ops/secure-cluster
$ docker-compose up -d

zk-1 is up-to-date
control-center is up-to-date
schema-registry is up-to-date
zk-3 is up-to-date
Recreating kafka-2 ...
secure-cluster_base_1 is up-to-date
Recreating kafka-2 ... done
Recreating kafka-1 ... done
Recreating kafka-3 ... done
```



Docker will realize that the definition of the 3 broker definitions have changed and reschedule the respective container.

### 3. Try to run the kafka-console-consumer:

```
$ kafka-console-consumer \
  --consumer.config scripts/security/consumer_ssl.properties \
  --from-beginning \
  --topic ssl-topic \
  --bootstrap-server kafka-1:9093,kafka-2:9093

[2018-08-07 11:35:07,521] ERROR [Consumer clientId=consumer-1, groupId=console-consumer-83215] Connection to node -1 failed authentication due to: SSL handshake failed
(org.apache.kafka.clients.NetworkClient)
[2018-08-07 11:35:07,530] ERROR Authentication failed: terminating consumer process
(kafka.tools.ConsoleConsumer$)
org.apache.kafka.common.errors.SslAuthenticationException: SSL handshake failed
Caused by: javax.net.ssl.SSLProtocolException: Handshake message sequence violation, 2
...
```



This fails with a Java IO Exception. Why?

### 4. Return to the ~/confluent-ops/secure-cluster/scripts/security folder. With your code editor add the following snippet to **both**, the producer\_ssl.properties and consumer\_ssl.properties files:

```
ssl.keystore.location=scripts/security/kafka.client.keystore.jks
ssl.keystore.password=confluent
```

### 5. Start the console Consumer for Topic ssl-topic again, this time passing in the updated consumer\_ssl.properties file. Now it should succeed and you should see the messages you typed earlier:



```
$ kafka-console-consumer \
  --consumer.config scripts/security/consumer_ssl.properties \
  --from-beginning \
  --topic ssl-topic \
  --bootstrap-server kafka-1:9093,kafka-2:9093

Security is good
Certificates
Keys
```

Press `Ctrl-c` to quit the consumer once your messages have been displayed.

## SSL Performance Impact

In this section, you will run two performance tests and compare the results:

- Connecting to the cluster with no SSL via the configured PLAINTEXT port (9092)
- Connecting to the cluster with SSL via the configured SSL port (9093)
  1. Create a new Topic called `no-ssl-topic` with one Partition and three replicas.

```
$ kafka-topics \
  --zookeeper zk-1:2181 \
  --create \
  --topic no-ssl-topic \
  --partitions 1 \
  --replication-factor 3

Created topic "no-ssl-topic".
```

2. Produce a high rate of data to Topic `no-ssl-topic` without SSL. Notice that the Broker port numbers used below are 9092, the PLAINTEXT port.

```
$ kafka-producer-perf-test \
  --topic no-ssl-topic \
  --num-records 400000 \
  --record-size 1000 \
  --throughput 1000000 \
  --producer-props bootstrap.servers=kafka-1:9092,kafka-2:9092

38801 records sent, 7760.2 records/sec (7.40 MB/sec), 1978.4 ms avg latency, 3582.0
max latency.
54064 records sent, 10810.6 records/sec (10.31 MB/sec), 3084.3 ms avg latency, 3578.0
max latency.
71280 records sent, 14250.3 records/sec (13.59 MB/sec), 2749.2 ms avg latency, 3457.0
max latency.
67600 records sent, 13520.0 records/sec (12.89 MB/sec), 2152.5 ms avg latency, 2569.0
max latency.
74048 records sent, 14809.6 records/sec (14.12 MB/sec), 2267.4 ms avg latency, 2405.0
max latency.
76528 records sent, 15302.5 records/sec (14.59 MB/sec), 2211.3 ms avg latency, 2414.0
max latency.
400000 records sent, 12824.212112 records/sec (12.23 MB/sec), 2400.34 ms avg latency,
3582.00 ms max latency, 2291 ms 50th, 3395 ms 95th, 3479 ms 99th, 3560 ms 99.9th.
```



Performance results will vary.

- a. What is the average throughput? Look in the last line for the value associated with MB/sec.
  - b. What is the average latency? Look in the last line for the value associated with ms avg latency.
3. Now produce a high rate of data to Topic `ssl-topic` with SSL. Notice that you now connect to Broker port number 9093, the SSL port.

```
$ kafka-producer-perf-test \
  --topic ssl-topic \
  --num-records 400000 \
  --record-size 1000 \
  --throughput 1000000 \
  --producer-props bootstrap.servers=kafka-1:9093,kafka-2:9093 \
  --producer.config scripts/security/producer_ssl.properties

32401 records sent, 6477.6 records/sec (6.18 MB/sec), 1942.5 ms avg latency, 3575.0
max latency.
53792 records sent, 10756.2 records/sec (10.26 MB/sec), 3433.0 ms avg latency, 4072.0
max latency.
57008 records sent, 11401.6 records/sec (10.87 MB/sec), 2997.2 ms avg latency, 3423.0
max latency.
62784 records sent, 12556.8 records/sec (11.98 MB/sec), 2446.4 ms avg latency, 3097.0
max latency.
51072 records sent, 10212.4 records/sec (9.74 MB/sec), 3398.1 ms avg latency, 3978.0
max latency.
77184 records sent, 15436.8 records/sec (14.72 MB/sec), 2197.0 ms avg latency, 2541.0
max latency.
65152 records sent, 13030.4 records/sec (12.43 MB/sec), 2394.6 ms avg latency, 2674.0
max latency.
400000 records sent, 11406.410403 records/sec (10.88 MB/sec), 2681.58 ms avg latency,
4072.00 ms max latency, 2594 ms 50th, 3794 ms 95th, 3968 ms 99th, 4061 ms 99.9th.
```



Performance results will vary.

- What is the average throughput? Look in the last line for the value associated with MB/sec. Is this higher or lower than without SSL?
- What is the average latency? Look for the value associated with ms avg latency. Is this higher or lower than without SSL?

## Cleanup

- Run the following commands to stop the cluster and clean up unused resources:

```
$ docker-compose down -v
```



**STOP HERE. THIS IS THE END OF THE EXERCISE.**

# Running Kafka Connect

In this exercise, you will run Connect in distributed mode, and use the JDBC source Connector and File sink Connector.

## Prerequisites

1. Please make sure you have prepared your environment by following → [Preparing the Labs](#)
2. Make sure your Kafka cluster is started, **otherwise** execute this command:

```
$ cd ~/confluent-ops
$ docker-compose up -d
```

## Connect Pipeline

In this section, you will run Connect in distributed mode with two Connectors: a JDBC source Connector and a file sink Connector. The JDBC source Connector writes the contents of a database table to a Kafka Topic. The file sink Connector reads data from the same Kafka Topic and writes those messages to a file. It will update when new rows are added to the database.



## Prerequisites

1. Create the SQLite database `my.db` in the folder `~/confluent-ops/data`:

```
$ cd ~/confluent-ops
$ sqlite3 data/my.db
```

2. Run the following statments in SQLite3:

```
create table years(id INTEGER PRIMARY KEY AUTOINCREMENT, name VARCHAR(50), year INTEGER);
insert into years(name,year) values('Hamlet',1600);
insert into years(name,year) values('Julius Caesar',1599);
insert into years(name,year) values('Macbeth',1605);
insert into years(name,year) values('Merchant of Venice',1595);
insert into years(name,year) values('Othello',1604);
insert into years(name,year) values('Romeo and Juliette',1594);
insert into years(name,year) values('Anthony and Cleopatra',1606);
```

3. Make sure the data is there:

```
sqlite> SELECT * FROM years;

1|Hamlet|1600
2|Julius Caesar|1599
3|Macbeth|1605
4|Merchant of Venice|1595
5|Othello|1604
6|Romeo and Juliette|1594
7|Anthony and Cleopatra|1606
```

4. Press CTRL-d to exit SQLite3.

5. Create a new Topic called `shakespeare_years` with one Partition and one replica.

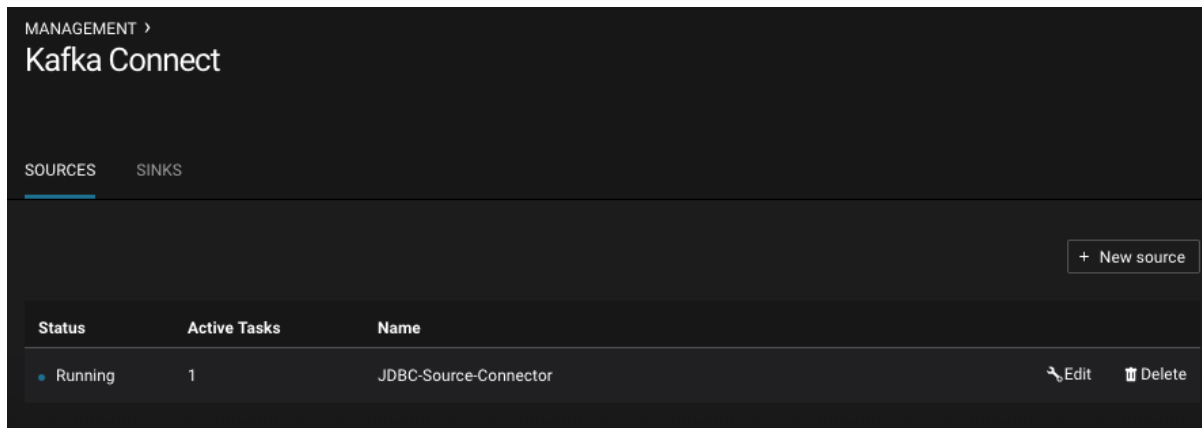
```
$ kafka-topics \
  --zookeeper zk-1:2181 \
  --create \
  --topic shakespeare_years \
  --partitions 1 \
  --replication-factor 1

Created topic "shakespeare_years".
```

## Configuring the Source Connector

1. Add a new source connector to read data from the database `my.db` and write to the Kafka topic `shakespeare_years` by using the Connect REST API:
  - a. Open Control Center at <http://localhost:9021>. In the **MANAGEMENT** sidebar, click **Kafka Connect**.
  - b. In the **SOURCES** tab, click the button **+ New Source**.
  - c. Configure the JDBC Source Connector
    - i. In the **Connector Class** dropdown, choose `JdbcSourceConnector`.
    - ii. In the **Name** text box, type `JDBC-Source-Connector`.
    - iii. Under **Database**, in the **JDBC URL** text box, type `jdbc:sqlite:/data/my.db`.

- iv. Under **Database**, in the **Table Whitelist** dropdown, choose `years`.
  - v. Under **Mode**, in the **Table Loading Mode** text box, type `incrementing`.
  - vi. Under **Mode**, in the **Incrementing Column Name** text box, type `id`.
  - vii. Under **Connector**, in the **Topic Prefix** text box, type `shakespeare_`.
- d. Click **Continue**.
  - e. Click **Save & Finish**.
  - f. Verify you see the new connector **JDBC-Source-Connector** running.



- a. Alternatively add the source connector via command line and **REST API** of Connect:

```
$ curl -s -X POST \
  -H "Content-Type: application/json" \
  --data '{
    "name": "JDBC-Source-Connector",
    "config": {
      "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
      "connection.url": "jdbc:sqlite:/data/my.db",
      "table.whitelist": "years",
      "mode": "incrementing",
      "incrementing.column.name": "id",
      "table.types": "TABLE",
      "topic.prefix": "shakespeare_"
    }
  }' http://connect:8083/connectors
```

- 2. Launch another terminal and start the console Consumer for Topic `shakespeare_years`:

```
$ kafka-avro-console-consumer \
  --bootstrap-server kafka-1:9092, kafka-2:9092 \
  --property schema.registry.url=http://schema-registry:8081 \
  --from-beginning \
  --topic shakespeare_years

{"id":1,"name":{"string":"Hamlet"},"year":{"long":1600}}
{"id":2,"name":{"string":"Julius Caesar"},"year":{"long":1599}}
{"id":3,"name":{"string":"Macbeth"},"year":{"long":1605}}
{"id":4,"name":{"string":"Merchant of Venice"},"year":{"long":1595}}
{"id":5,"name":{"string":"Othello"},"year":{"long":1604}}
{"id":6,"name":{"string":"Romeo and Juliette"},"year":{"long":1594}}
{"id":7,"name":{"string":"Anthony and Cleopatra"},"year":{"long":1606}}
...
```

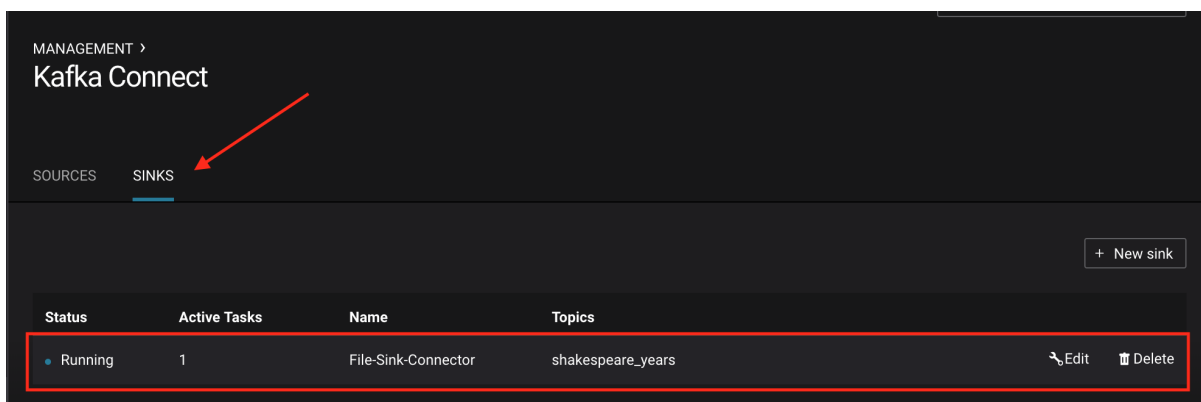
Leave the consumer running until instructed to terminate it. See what messages have been and will be produced.

## Configuring the Sink Connector

1. Add a new sink connector to read data from the Kafka topic `shakespeare_years` and write to the file `/data/test.sink.txt` on the Connect worker system:

```
$ curl -s -X POST \
  -H "Content-Type: application/json" \
  --data '{
    "name": "File-Sink-Connector",
    "config": {
      "topics": "shakespeare_years",
      "connector.class":
"org.apache.kafka.connect.file.FileStreamSinkConnector",
      "value.converter": "io.confluent.connect.avro.AvroConverter",
      "value.converter.schema.registry.url": "http://schema-registry:8081",
      "file": "/data/test.sink.txt"
    }
  }' http://connect:8083/connectors
```

2. Use Control Center to verify the creation of the connector.







Due to a limitation of the Confluent Control Center it is not possible to correctly define the sink in the UI. More specific: it is not possible in the Control Center to define a value for the mandatory property value `.converter.schema.registry.url`. That is why we had to resort to method that uses the Connect REST API.

3. Verify that a new file called `test.sink.txt` has been created in the folder `~/confluent-ops/data`. View this file to confirm that the sink connector worked:

```
$ cat ~/confluent-ops/data/test.sink.txt

Struct{id=1,name=Hamlet,year=1600}
Struct{id=2,name=Julius Caesar,year=1599}
Struct{id=3,name=Macbeth,year=1605}
Struct{id=4,name=Merchant of Venice,year=1596}
Struct{id=5,name=Othello,year=1604}
Struct{id=6,name=Romeo and Juliet,year=1594}
Struct{id=7,name=Antony and Cleopatra,year=1606}
```



This file was created by the **File Sink Connector** in folder `/data/` on the connect container. Since this container folder is mapped to the folder `~/confluent-ops/data` of your host system we can see it there too.

4. Insert a few new rows into the table `years`:

- a. From your host, run SQLite3:

```
$ sqlite3 data/my.db
```

- b. Insert two records:

```
sqlite> INSERT INTO years(name,year) VALUES('Tempest',1611);
INSERT INTO years(name,year) VALUES('King Lear',1605);
```

- c. In the window where the consumer is still running observe that two new records have been output:

```
{"id":8,"name":{"string":"Tempest"},"year":{"long":1611}}
{"id":9,"name":{"string":"King Lear"},"year":{"long":1605}}
```

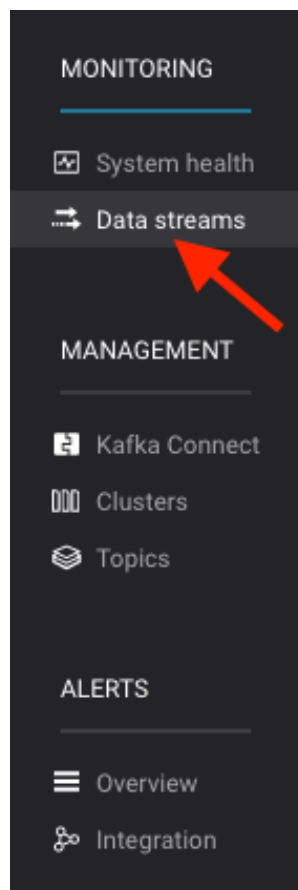
- d. Quit SQLite3 by pressing CTRL-d.

5. View the sink file, `test.sink.txt` again and notice that the new log lines are added to it:

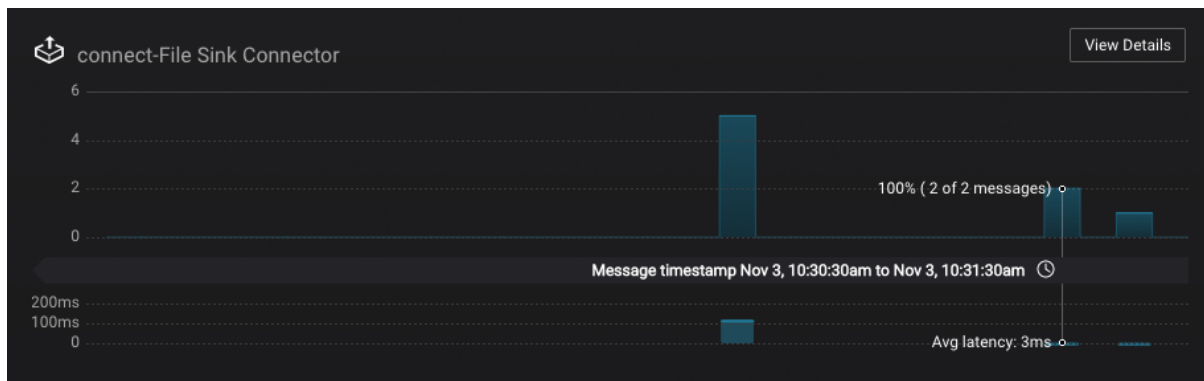
```
$ cat ~/confluent-ops/data/test.sink.txt

Struct{id=1,name=Hamlet,year=1600}
Struct{id=2,name=Julius Caesar,year=1599}
Struct{id=3,name=Macbeth,year=1605}
Struct{id=4,name=Merchant of Venice,year=1596}
Struct{id=5,name=Othello,year=1604}
Struct{id=6,name=Romeo and Juliet,year=1594}
Struct{id=7,name=Antony and Cleopatra,year=1606}
Struct{id=8,name=Tempest,year=1611}
Struct{id=9,name=King Lear,year=1605}
```

6. In Control Center, observe the consumer group performance for Kafka Connect. You could use this to ensure that Kafka Connect is performing well in your cluster, just like other production traffic.
  - a. In the Control Center GUI, click on **Data streams**. If at first the graphs don't appear, wait a minute and refresh your browser.



- b. Scroll down to view the consumer group **connect-File-Sink-Connector**.



7. In the terminal window where the consumer is running, press `Ctrl-c` to terminate the process.

## Cleanup

1. Run the following commands to stop the cluster and clean up unused resources:

```
$ docker-compose down -v
```



**STOP HERE. THIS IS THE END OF THE EXERCISE.**

## Appendix: Running Labs in Docker for Desktop

If you have installed Docker for Desktop on your Mac or Windows 10 Pro machine you are able to complete the course by building and running your applications from the command line.

- Increase the memory available to Docker Desktop to a minimum of 6 GiB. See the advanced settings for Docker Desktop for Mac, and Docker Desktop for Windows.
- Download the `docker-compose.yml` files from the exercises **Preparing the Lab** and **Securing the Kafka Cluster** to your local computer.

```
mkdir -p ~/confluent-ops/data && cd ~/confluent-ops
curl -L https://cnfl.io/docker-compose-apr19 -o docker-compose.yml
mkdir -p ~/confluent-ops/secure-cluster && cd ~/confluent-ops/secure-cluster
curl -L https://cnfl.io/docker-compose-ssl-apr19 -o docker-compose.yml
```

- Open both `docker-compose.yml` files and uncomment the base container at the end of the file.
- In each exercise follow the instructions to launch the cluster containers with `docker-compose` on your host machine.
- Begin the exercises by first opening a bash shell on the base container. All the command line instructions will work from the base container. This container has been preconfigured with all of the tools you use in the exercises, e.g. `kafka-topics`, `confluent-rebalancer`, and `kafka-configs`.

```
$ docker-compose exec base /bin/bash
bash-4.4#
```

- Anywhere you are instructed to open additional terminal windows you can `exec` additional bash shells on the base container with the same command as above on your host machine.
- Any subsequent `docker` or `docker-compose` instructions should be run on your host machine.

## Appendix: Reassigning Partitions in a Topic - Alternate Method

In the **Kafka Administrative Tools** exercise, one of the sections uses the Confluent Auto Data Balancer to rebalance the cluster and to reassign partitions for the topic `moving`. If you prefer to use the Apache open source tool instead, you may follow the instructions below.



This appendix assumes that the topic `moving` exists and has been populated as described in the **Kafka Administrative Tools** exercise. If this is not the case then please first follow the instructions → [Rebalancing the Cluster](#).

1. Launch a terminal and `exec` into the base container which will be used as starting point for subsequent commands:

```
$ cd ~/confluent-ops
$ docker-compose exec base /bin/bash
root@base:/#
```

2. Switch to the data folder (remember this folder is mapped to the file system of the host system):

```
root@base:/# cd /data
root@base:/data#
```

3. In that folder create a file `topics-to-move.json` as follows:

```
root@base:/data# echo '{"topics": [{"topic": "moving"}], "version": 1}' > topics-to-move.json
```

4. Generate the reassignment plan.

```
root@base:/data# kafka-reassign-partitions \
  --zookeeper zk-1:2181 \
  --topics-to-move-json-file topics-to-move.json \
  --broker-list "101,102,103" \
  --generate > reassignment.json
```

The content of the file `reassignment.json` should look like this:

```
root@base:/data# cat reassignment.json

Current partition replica assignment
{"version":1,"partitions":[{"topic":"moving","partition":2,"replicas":[101,102],"log_dirs":["any","any"]},{ "topic":"moving","partition":3,"replicas":[102,101],"log_dirs":["any","any"]},{ "topic":"moving","partition":0,"replicas":[101,102],"log_dirs":["any","any"]},{ "topic":"moving","partition":4,"replicas":[101,102],"log_dirs":["any","any"]},{ "topic":"moving","partition":5,"replicas":[102,101],"log_dirs":["any","any"]},{ "topic":"moving","partition":1,"replicas":[102,101],"log_dirs":["any","any"]}]}

Proposed partition reassignment configuration
{"version":1,"partitions":[{"topic":"moving","partition":2,"replicas":[101,102],"log_dirs":["any","any"]},{ "topic":"moving","partition":4,"replicas":[103,102],"log_dirs":["any","any"]},{ "topic":"moving","partition":1,"replicas":[103,101],"log_dirs":["any","any"]},{ "topic":"moving","partition":0,"replicas":[102,103],"log_dirs":["any","any"]},{ "topic":"moving","partition":3,"replicas":[102,101],"log_dirs":["any","any"]},{ "topic":"moving","partition":5,"replicas":[101,103],"log_dirs":["any","any"]}]}

```

5. Open another terminal window and navigate to the mapped data folder:

```
$ cd ~/confluent-ops/data
```

6. Find the file `reassignment.json` and edit it (with your code editor) so it only includes the "Proposed partition reassignment configuration" in JSON (i.e., just the last line). The resulting file should look like this:

```
$ cat reassignment.json
```

```
{ "version":1, "partitions": [{ "topic": "moving", "partition": 2, "replicas": [101, 102], "log_dirs": [ "any", "any" ] }, { "topic": "moving", "partition": 4, "replicas": [103, 102], "log_dirs": [ "any", "any" ] }, { "topic": "moving", "partition": 1, "replicas": [103, 101], "log_dirs": [ "any", "any" ] }, { "topic": "moving", "partition": 0, "replicas": [102, 103], "log_dirs": [ "any", "any" ] }, { "topic": "moving", "partition": 3, "replicas": [102, 101], "log_dirs": [ "any", "any" ] }, { "topic": "moving", "partition": 5, "replicas": [101, 103], "log_dirs": [ "any", "any" ] } ] }
```

7. Back from within the base container observe the current throttling limits configured:

```
root@base:/data# kafka-configs \
  --describe \
  --zookeeper zk-1:2181 \
  --entity-type brokers
```



By default, there should be none (i.e. **no output** in the command above).

8. Execute the reassignment plan with throttling set to 1MBps.

```
root@base:/data# kafka-reassign-partitions \
  --zookeeper zk-1:2181 \
  --reassignment-json-file reassignment.json \
  --execute \
  --throttle 1000000
```

Current partition replica assignment

```
{ "version":1, "partitions": [{ "topic": "moving", "partition": 2, "replicas": [101, 102], "log_dirs": [ "any", "any" ] }, { "topic": "moving", "partition": 3, "replicas": [102, 101], "log_dirs": [ "any", "any" ] }, { "topic": "moving", "partition": 0, "replicas": [101, 102], "log_dirs": [ "any", "any" ] }, { "topic": "moving", "partition": 4, "replicas": [101, 102], "log_dirs": [ "any", "any" ] }, { "topic": "moving", "partition": 5, "replicas": [102, 101], "log_dirs": [ "any", "any" ] }, { "topic": "moving", "partition": 1, "replicas": [102, 101], "log_dirs": [ "any", "any" ] } ] }
```

Save this to use as the `--reassignment-json-file` option during rollback

Warning: You must run `Verify` periodically, until the reassignment completes, to ensure the throttle is removed. You can also alter the throttle by rerunning the `Execute` command passing a new value.

The inter-broker throttle limit was set to 1000000 B/s

Successfully started reassignment of partitions.

9. Observe the new throttling limits configured.

```

root@base:/data# kafka-configs \
  --describe \
  --zookeeper zk-1:2181 \
  --entity-type brokers

Configs for brokers '101' are
leader.replication.throttled.rate=1000000,follower.replication.throttled.rate=1000000
Configs for brokers '102' are
leader.replication.throttled.rate=1000000,follower.replication.throttled.rate=1000000
Configs for brokers '103' are
leader.replication.throttled.rate=1000000,follower.replication.throttled.rate=1000000

```

# 10. Monitor the progress of the reassignment.

- a. Run the `kafka-reassign-partitions` command with the `--verify` option and note that some reassignments are “still in progress”.

```

root@base:/data# kafka-reassign-partitions \
  --zookeeper zk-1:2181 \
  --reassignment-json-file reassignment.json \
  --verify

Status of partition reassignment:
Reassignment of partition moving-2 completed successfully
Reassignment of partition moving-3 completed successfully
Reassignment of partition moving-0 is still in progress
Reassignment of partition moving-4 is still in progress
Reassignment of partition moving-5 is still in progress
Reassignment of partition moving-1 is still in progress

```

- b. Run the `kafka-topics` command with the `--describe` option and notice the listed replicas for the partitions that are still in progress for reassignment.

```

root@base:/data# kafka-topics \
  --zookeeper zk-1:2181 \
  --describe \
  --topic moving

Topic:moving    PartitionCount:6    ReplicationFactor:3
Configs:leader.replication.throttled.replicas=0:101,0:102,4:101,4:102,5:102,5:101,1:10
2,1:101,follower.replication.throttled.replicas=0:103,4:103,5:103,1:103
  Topic: moving    Partition: 0    Leader: 101 Replicas: 102,103,101    Isr: 101,102
  Topic: moving    Partition: 1    Leader: 102 Replicas: 103,101,102    Isr: 102,101
  Topic: moving    Partition: 2    Leader: 101 Replicas: 101,102    Isr: 101,102
  Topic: moving    Partition: 3    Leader: 102 Replicas: 102,101    Isr: 102,101
  Topic: moving    Partition: 4    Leader: 101 Replicas: 103,102,101    Isr: 101,102
  Topic: moving    Partition: 5    Leader: 102 Replicas: 101,103,102    Isr: 102,101

```

11. Increase the throttle limit configuration to 1GBps by rerunning the `kafka-reassign-partitions` command with the new throttle limit.

```
root@base:/data# kafka-reassign-partitions \
  --zookeeper zk-1:2181 \
  --reassignment-json-file reassignment.json \
  --execute \
  --throttle 1000000000
```

There is an existing assignment running.

12. Note the updated throttle limit configuration values.

```
root@base:/data# kafka-configs \
  --describe \
  --zookeeper zk-1:2181 \
  --entity-type brokers
```

```
Configs for brokers '101' are
leader.replication.throttled.rate=1000000000,follower.replication.throttled.rate=10000000
00
Configs for brokers '102' are
leader.replication.throttled.rate=1000000000,follower.replication.throttled.rate=10000000
00
Configs for brokers '103' are
leader.replication.throttled.rate=1000000000,follower.replication.throttled.rate=10000000
00
```

13. The throttle limit is still in place. This is expected: the throttling is not removed automatically when reassignment completes.

```
root@base:/data# kafka-configs \
  --describe \
  --zookeeper zk-1:2181 \
  --entity-type brokers
```

```
Configs for brokers '101' are
leader.replication.throttled.rate=1000000000,follower.replication.throttled.rate=10000000
00
Configs for brokers '102' are
leader.replication.throttled.rate=1000000000,follower.replication.throttled.rate=10000000
00
Configs for brokers '103' are
leader.replication.throttled.rate=1000000000,follower.replication.throttled.rate=10000000
00
```

- a. Run the `kafka-reassign-partitions` command with the `--verify` option again. Note the “completed successfully” output and “Throttle was removed”.



```
root@base:/data# kafka-reassign-partitions \
  --zookeeper zk-1:2181 \
  --reassignment-json-file reassignment.json \
  --verify
```

```
Status of partition reassignment:
Reassignment of partition moving-2 completed successfully
Reassignment of partition moving-3 completed successfully
Reassignment of partition moving-0 completed successfully
Reassignment of partition moving-4 completed successfully
Reassignment of partition moving-5 completed successfully
Reassignment of partition moving-1 completed successfully
Throttle was removed.
```

b. Confirm that the throttle limit is now removed.

```
root@base:/data# kafka-configs \
  --describe \
  --zookeeper zk-1:2181 \
  --entity-type brokers
```

```
Configs for brokers '101' are
Configs for brokers '102' are
Configs for brokers '103' are
```

14. Exit the base container by pressing CTRL-d.
15. Return to (the end of) the exercise **Rebalancing the Cluster** (→ [Jump here](#)).



**STOP HERE. THIS IS THE END OF THE EXERCISE.**