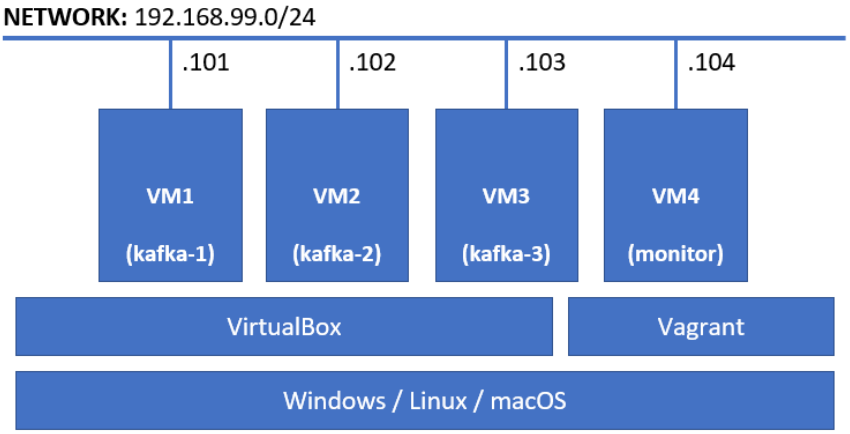
Practice M6: Apache Kafka

For this practice, our lab environment will look like this



We are going to use a homogeneous setup. All machines will be based either on **CentOS Stream 9** or on **Debian 11**

All configurations and supplementary files are provided as a ZIP archive and can be downloaded from the module section in the official site

Part 1

First, bring up the environment using the provided **Vagrantfile**

Install Apache Kafka

We can download the files from here: https://kafka.apache.org/downloads

Preparation

Before we start, we should take care for a few preparation steps

We should install a recent version of **Java**

Let's start with the first machine (**kafka-1**)

On CentOS

Install the needed packages

```
sudo dnf install java-17-openjdk
```

For easier interaction, we can disable the firewall if running

```
sudo systemctl disable --now firewalld
```

On Debian/Ubuntu

Install the required packages

```
sudo apt-get update
```

```
sudo apt-get install openjdk-17-jre
```

Actual Installation

Download the recent package. For example, with this command

```
wget https://archive.apache.org/dist/kafka/3.3.1/kafka_2.13-3.3.1.tgz
```

Now, extract the archive

```
tar xzvf kafka_2.13-3.3.1.tgz
```

And rename it for easier interaction

```
mv kafka_2.13-3.3.1 kafka
```

Enter the folder

```
cd kafka
```

And explore the contents of the **Zoo Keeper**'s file

```
cat config/zookeeper.properties
```

We can see the default values and we are fine with them

Now, let's start it in but in foreground mode

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

After a few seconds it will be up and running

Now, open a new terminal session to the same machine

Let's first see that the **Zoo Keeper** is working

Execute the following command

```
telnet localhost 2181
```

If the command if not installed, then install it

Once connected, execute

```
srvr
```

Okay, our instance is working and ready

Navigate to the folder where the files are extracted

```
cd kafka
```

Explore the default configuration file for **Apache Kafka**

```
cat config/server.properties
```

Now, start a broker with

```
bin/kafka-server-start.sh config/server.properties
```

After a few seconds it will be ready

Single Node Experiments

Topics

Let's create a topic and see what will happen

Open another session (***third*** one) to the same machine (**kafka-1**)

Now, let's create a topic named demo with one partition and replication factor set to one as well

First, go to the folder

```
cd kafka
```

And then execute

```
bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic demo
```

We can check the messages on the broker's session (the ***second session***)

Notice where the files are created - **/tmp/kafka-logs/demo-0**

Let's see what we have there (go back to the ***third session***)

```
ls -lh /tmp/kafka-logs/
```

```
ls -lh /tmp/kafka-logs/demo-0/
```

The log file is empty as we do not have any messages yet

Create another topic

```
bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic test
```

And check the folders again

```
ls -lh /tmp/kafka-logs/
```

```
ls -lh /tmp/kafka-logs/test-0/
```

Now, list the topics with

```
bin/kafka-topics.sh --list --bootstrap-server localhost:9092
```

And ask for more information about one of them (for example, for the demo one)

```
bin/kafka-topics.sh --describe --topic demo --bootstrap-server localhost:9092
```

Producers

Start a producer to generate some messages

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic demo
```

Enter some text, like

```
message1
```

```
message2
```

```
message3
```

Now, either press **Ctrl+D** or **Ctrl+C** to end the message entering process

Check the files in the appropriate folder

```
ls -lh /tmp/kafka-logs/demo-0/
```

The log file finally consumed some space

Try to see its contents with

```
cat /tmp/kafka-logs/demo-0/00000000000000000000.log
```

It is not the best possible way to explore the data but at least we can identify our messages

Let's use another approach

```
bin/kafka-run-class.sh kafka.tools.DumpLogSegments --deep-iteration --print-data-log --files /tmp/kafka-logs/demo-0/00000000000000000000.log
```

This is another story

Start again the producer

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic demo
```

Consumers

Open another session (a ***fourth one***) to the same machine (**kafka-1**)

Navigate to the folder

```
cd kafka
```

Start a consumer

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic demo
```

Hmm, no messages

Let's switch to the session of the producer (the ***third one***) and type a message (for example, **message4**)

Return to the session of the consumer (the ***fourth one***)

Aha, it appeared

Now, press **Ctrl+C** to stop and close the consumer

Start it again, but this time with the following command

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic demo --from-beginning
```

Now, it fetched all messages

Return to the producer's session (the ***third one***) and type another message (for example, **message5**)

Then return again in the consumer's session (the ***fourth one***). The message should be there

Can we have multiple consumers of the same topic? Sure, we can

Open another session (a ***fifth one***) to the same machine (**kafka-1**)

Navigate to the folder

```
cd kafka
```

Start another consumer

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic demo
```

Return to the producer's session (the ***third one***) and type a few messages (for example, **message6** and **message7**)

Check again the session of the two consumers (the ***fourth*** and the ***fifth*** ones)

All new messages appear on both of them

Clean Up

Stop all the processes – consumers, producer, broker and zookeeper

Delete the data

```
rm -rf /tmp/kafka-logs /tmp/zookeeper
```

And close all the five sessions

Simple Cluster

Installation and Configuration

Now, open three new sessions – one for every node (**kafka-1**, **kafka-2** and **kafka-3**)

Installation

Repeat the preparation and installation steps on the other two machines (**kafka-2** and **kafka-3**)

Configuration

Once done with the installation, adjust the configurations an all three nodes (kafka-1, kafka-2 and kafka-3) as follows

Start from the first node (**kafka-1**)

Open the **Zoo Keeper** main configuration file for editing

```
vi config/zookeeper.properties
```

Add the following at the end of the file

```
tickTime = 2000
initLimit = 5
syncLimit = 2
server.1=kafka-1:2888:3888
server.2=kafka-2:2888:3888
server.3=kafka-3:2888:3888
```

Save and close the file

Prepare the default folder

```
mkdir /tmp/zookeeper
```

And create the following identification file

```
echo "1" > /tmp/zookeeper/myid
```

This one (the value) should be adjusted for the other two nodes (it should be 2 and 3 respectively)

Open the main configuration file for the broker

```
vi config/server.properties
```

Adjust at least the following three parameters

```
broker.id=1
advertised.listeners=PLAINTEXT://kafka-1:9092
zookeeper.connect=kafka-1:2181,kafka-2:2181,kafka-3:2181
```

Save and close the file

*Again, the **broker.id** and **advertised.listeners** should be adjusted for the other two nodes*

Now, repeat the above steps on the other two nodes as well

Cluster Creation

Finally, execute on every machine the following to start the **Zoo Keeper** component

```
bin/zookeeper-server-start.sh -daemon config/zookeeper.properties
```

Check that all three instances started and connected successfully

First, check that process started

```
ps x
```

And then the logs

```
cat logs/zookeeper.out
```

*Look for some **INFO LEADING** and **INFO FOLLOWING** messages*

Then, start the brokers be executing on every machine the following

```
bin/kafka-server-start.sh -daemon config/server.properties
```

Check that all three instances started and connected successfully

First, check that process started

```
ps x
```

And then the logs

```
cat logs/server.log
```

Test the Setup #1

Let's return on the first machine (**kafka-1**), we will use it as a workstation for now

Create a topic by executing the following command

```
bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 3
--partitions 1 --topic demo
```

*Note that the **localhost** portion can be changed with **kafka-1** in our case*

This time, our topic has replication factor of three (equals to the number of nodes)

Now, check what we have on every node, in terms of files, by executing the following command

```
ls -al /tmp/kafka-logs/demo-0/
```

As we know, we can use another approach to get details about a topic

```
bin/kafka-topics.sh --describe --topic demo --bootstrap-server localhost:9092
```

Okay, let's start a producer

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic demo
```

Enter a message or two

Switch to one of the other two machines (for example, **kafka-2**) and check the files

```
ls -al /tmp/kafka-logs/demo-0/
```

The log increases its size

If we dump it, we will see the contents as expected

```
bin/kafka-run-class.sh kafka.tools.DumpLogSegments --deep-iteration --print-data-log -
-files /tmp/kafka-logs/demo-0/00000000000000000000.log
```

Ask for details about a topic

```
bin/kafka-topics.sh --describe --topic demo --bootstrap-server localhost:9092
```

We can see that **all nodes are in sync** *(the **Isr** field)*

Let's stop one of the machines and see what will happen

Go to the **kafka-3** machine and stop the broker by executing the following

```
bin/kafka-server-stop.sh
```

Return on one of the other two machines (for example, **kafka-2**) and ask again for details about the topic

```
bin/kafka-topics.sh --describe --topic demo --bootstrap-server localhost:9092
```

We can see that the third one is not listed any more under the **Isr** list

Go to the machine where the producer is running (**kafka-1**) and add two more messages

Return on the **kafka-2** machine and ask again for details about the topic

```
bin/kafka-topics.sh --describe --topic demo --bootstrap-server localhost:9092
```

And check the size of the log

```
ls -al /tmp/kafka-logs/demo-0/
```

Now, go to the third (**kafka-3**) machine and check the size

```
ls -al /tmp/kafka-logs/demo-0/
```

It is smaller, as it contains fewer messages compared to the other two nodes

Now, start the broker (on **kafka-3**) again with

```
bin/kafka-server-start.sh -daemon config/server.properties
```

And ask for details about the topic

```
bin/kafka-topics.sh --describe --topic demo --bootstrap-server localhost:9092
```

Now the third machine (**kafka-3**) is present in the **Isr** list, so it should be in sync

And check the size of the log

```
ls -al /tmp/kafka-logs/demo-0/
```

It is the same as the one on second machine (**kafka-2**)

All nodes synched

Test the Setup #2

Return on the first machine (**kafka-1**) and stop the producer

Create a new topic but this time with multiple partitions

```
bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 3
--partitions 5 --topic test
```

While still on the same machine, check the topic details

```
bin/kafka-topics.sh --describe --topic test --bootstrap-server localhost:9092
```

We can see who is responsible for which partition and where the replicas are going

And then the local files

```
ls -al /tmp/kafka-logs/
```

All the partitions are present here not only those that this node is leader for

The reason here is the fewer nodes and the great number of partitions combined with the replication factor

Start a producer

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
```

And enter at least six messages

Then switch to the second session and machine (**kafka-2**) and ask for details about the topic

```
bin/kafka-topics.sh --describe --topic test --bootstrap-server localhost:9092
```

Then check the files

```
ls -al /tmp/kafka-logs/test-?/*.log
```

Some of them may be empty while the others will have data

Return on the first machine (**kafka-1**) and enter at least one more message

Go back to the second machine (**kafka-2**) and check again the files

```
ls -al /tmp/kafka-logs/test-?/*.log
```

Their size changes

You can repeat this a few times if you like

Finally, return on the first machine (**kafka-1**) and stop the producer

Delete the two topics

```
bin/kafka-topics.sh --delete --bootstrap-server localhost:9092 --topic test
bin/kafka-topics.sh --delete --bootstrap-server localhost:9092 --topic demo
```

If we check the files they should be gone as well

```
ls -al /tmp/kafka-logs/
```

Not quite, there are still there but not forever

If we wait a while and ask again, we will see that they are gone

Leave the machines running

Part 2

Now, let's explore a bit the process of creating our own producer and consumer

Log on to one of the machines. For example, on the third one (**kafka-3**)

Install **Python** if not already installed

```
sudo dnf install python3 python3-pip
```

And add the **Kafka Python** library

```
sudo pip3 install kafka-python
```

Then, create a topic with one partition and replication factor of three

Let's use the following command

```
bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 3
--partitions 1 --topic demo1
```

Own Producer

Now, create a working folder, for example **/vagrant/code** and navigate to it

```
mkdir /vagrant/code
```

```
cd /vagrant/code
```

In the folder, create an empty file named **producer.py**

```
vi producer.py
```

Enter the following code

```
from kafka import KafkaProducer
from time import sleep
from random import randrange

tpk = 'demo1'
idx = 1
print('Producer started. Press Ctrl+C to stop. Working on topic=' + tpk)

try:
    producer = KafkaProducer(bootstrap_servers=['kafka-1:9092','kafka-2:9092','kafka-3:9092'])
    while True:
        st = 'My number is ' + str(idx)
        print('message: ' + st)
        producer.send(tpk, bytes(st, encoding='utf-8'))
        producer.flush()
        slp = randrange(1,10)
        print('Sleep for ' + str(slp) + ' second(s).')
        sleep(slp)
        idx = idx + 1
except Exception as ex:
    print(str(ex))
except KeyboardInterrupt:
    pass
finally:
    if producer is not None:
        producer.close()
```

```
print('... closed.')
```

Save and close the file

Start it with

```
python3 producer.py
```

Now, switch to another machine. For example, on the first machine (**kafka-1**)

And start a consumer

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic demo1
```

Wait for a few messages to appear

All arrive in the order in which they were generated

Now, stop the consumer and the producer

Let's add a few partitions to the topic using the following command

```
bin/kafka-topics.sh --alter --bootstrap-server localhost:9092 --partitions 3 --topic demo1
```

Check that the changes are applied

```
bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --topic demo1
```

Okay, now start the consumer with

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic demo1
```

Return on the third machine (**kafka-3**) and start the producer

```
python3 producer.py
```

Then return on the first machine (**kafka-1**) and watch the order in which the messages will appear

It appears that their order is the same. But is it?

Let's stop the consumer and start it again but ask it to reread everything since the beginning

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic demo1 --from-beginning
```

Now, we can see that the records appear in different order, and they seem to be somewhat spread across partitions

Okay, stop again both the consumer and producer

Own Consumer (Subscribe)

Continue on the first machine (**kafka-1**)

Let's create two new topics each with three partitions and three replicas

```
bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 3 --partitions 3 --topic demo2
```

```
bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 3 --partitions 3 --topic demo3
```

Check the setup

```
bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --topic demo2
```

```
bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --topic demo3
```

Then, make sure that you have **Python3** and the appropriate **Kafka** module installed

```
sudo dnf install python3 python3-pip
sudo pip3 install kafka-python
```

Now, change the producer to send messages to the two new topics

```
vi /vagrant/code/producer.py
```

Make sure it looks like this

```
from kafka import KafkaProducer
from time import sleep
from random import randrange

tpk = ['demo2', 'demo3']
idx = 1
print('Producer started. Press Ctrl+C to stop. Working on topic=' + str(tpk))

try:
    producer = KafkaProducer(bootstrap_servers=['kafka-1:9092','kafka-2:9092','kafka-3:9092'])
    while True:
        st = 'My number is ' + str(idx)
        print('message: ' + st)
        for t in tpk:
            producer.send(t, bytes(st, encoding='utf-8'))
        producer.flush()
        slp = randrange(1,10)
        print('Sleep for ' + str(slp) + ' second(s).')
        sleep(slp)
        idx = idx + 1
except Exception as ex:
    print(str(ex))
except KeyboardInterrupt:
    pass
finally:
    if producer is not None:
        producer.close()

print('... closed.')
```

And create a new file in the same folder (**/vagrant/code/**) named **consumer-subscribe.py**

```
vi /vagrant/code/consumer-subscribe.py
```

With the following content

```
from kafka import KafkaConsumer


print('Consumer started. Press Ctrl+C to stop.')


try:
    consumer = KafkaConsumer(bootstrap_servers=['kafka-1:9092','kafka-2:9092','kafka-3:9092'])
    consumer.subscribe(['demo2', 'demo3'])
    for message in consumer:
        print(message)
except Exception as ex:
    print(str(ex))
except KeyboardInterrupt:
    pass
finally:
    if consumer is not None:
        consumer.close()


print('... closed.')
```

Save and close the file

Start the consumer on the first machine (**kafka-1**)

```
python3 /vagrant/code/consumer-subscribe.py
```

On the same machine but in another session start the producer

```
python3 /vagrant/code/producer.py
```

Wait a while and explore the generated and consumed messages

The consumer gets all as it is subscribed to both topics

Stop both the producer and consumer

Own Consumer (Assign)

Continue on the first machine (**kafka-1**)

Create a new consumer application named **consumer-assign.py**

```
vi /vagrant/code/consumer-assign.py
```

With the following content

```
from kafka import KafkaConsumer, TopicPartition

print('Consumer started. Press Ctrl+C to stop.')

try:
    consumer = KafkaConsumer(bootstrap_servers=['kafka-1:9092','kafka-2:9092','kafka-3:9092'])
    consumer.assign([TopicPartition('demo2', 1), TopicPartition('demo3', 2)])
    for message in consumer:
        print(message)
except Exception as ex:
    print(str(ex))
except KeyboardInterrupt:
    pass
finally:
    if consumer is not None:
        consumer.close()

print('... closed.')
```

Save and close the file

Start the consumer on the first machine (**kafka-1**)

```
python3 /vagrant/code/consumer-assign.py
```

On the same machine but in another session start the producer

```
python3 /vagrant/code/producer.py
```

Wait a while and explore the generated and consumed messages

The consumer gets just those messages that match the desired topics and corresponding partitions

Stop both the producer and consumer

Consumer Groups

Continue on the first machine (**kafka-1**)

Create a new topic

```
bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 3 --topic demo4
```

Check the result

```
bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --topic demo4
```

Modify the **/vagrant/code/producer.py** application to work with just one topic (**demo4**)

Create a new consumer application and name it **consumer-group.py**

```
vi /vagrant/code/consumer-group.py
```

Make sure it looks like

```
from kafka import KafkaConsumer


print('Consumer started. Press Ctrl+C to stop.')


try:
    consumer = KafkaConsumer(bootstrap_servers=['kafka-1:9092','kafka-2:9092','kafka-3:9092'], group_id='mygroup')
    consumer.subscribe(['demo4'])
    for message in consumer:
        print(message)
except Exception as ex:
    print(str(ex))
except KeyboardInterrupt:
    pass
finally:
    if consumer is not None:
        consumer.close()


print('... closed.')
```

Open three sessions to the first machine (**kafka-1**)

In each session start a copy of the consumer

```
python3 /vagrant/code/consumer-group.py
```

Again, on the same machine (**kafka-1**) but in another session (the first one), start a copy of the producer

```
python3 /vagrant/code/producer.py
```

Explore consumer sessions

You will notice that each consumer is assigned to a specific partition and captures only those messages

If we add a new consumer, it will stay without showing anything as there is no partition for it (do not add one)

Now, stop the producer

Add a new partition

```
bin/kafka-topics.sh --alter --bootstrap-server localhost:9092 --partitions 4 --topic demo4
```

And start the producer again

```
python3 /vagrant/code/producer.py
```

Explore the sessions of the consumers

After a while, one of them will show data for two partitions

Add another consumer in an additional session to the same machine (**kafka-1**)

Explore the sessions of the consumers

After a while, the new one will start showing data for a partition

The same applies in reverse

Stop the producer and all consumers

Leave the machines running

Part 3

Now, let's explore one possible way to monitor an **Apache Kafka** setup

For this, we will use a combination of **Prometheus** (together with extractors) and **Grafana**

First, should you have a firewall running, disable it or adjust its rules accordingly

Let's assume that we decided to disable it and it is the **FirewallD** solution in place

```
sudo systemctl disable --now firewalld
```

We will start with the extractors but before that, let's check the **JConsole**

JConsole

Make sure that you have it on your host (it is part of the JDK package)

Next, go to one of the machines, for example the first machine (**kafka-1**)

Navigate to the folder where the **Kafka** files are

Execute the following to set an environment variable

```
export JMX_PORT=48888
```

You can use another port if you like

Now, stop the broker

```
bin/kafka-server-stop.sh
```

And start it again

```
bin/kafka-server-start.sh -daemon config/server.properties
```

Go to the host and start the **JConsole** application *(on Windows, it is in **C:\Program Files\Java\jdk-<version>\bin**)*

Select **Remote Process** and enter the following (for the first machine)

```
192.168.99.101:48888
```

Then hit **Connect**

When asked, click **Insecure connection**

After a few moments an overview dashboard will appear

Explore the other tabs as well

When on the **MBeans** tab, explore the tree's content

Open for example, the following

kafka.server > ReplicaManager > LeaderCount > Attributes

Try to match the value with the output of this command (executed on one of the other nodes)

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --describe | grep "Leader: 1" | wc -l
```

Should you want, continue the exploration

For example, check the value of **kafka.server > ReplicaManager > PartitionCount > Attributes**

And try to find if it is correct. You can use this command (executed on one of the other nodes)

```
bin/kafka-topics.sh --bootstrap-server localhost:9092 --describe
```

A hint – it shows not only of how many partitions the node is a leader, but also how many other partitions (replicas) are stored on it (so, it is a sum)

Once done, close the **JConsole** tool

Install Extractor (OS)

Open a session to the first machine (**kafka-1**)

Make sure you are in the home folder of the vagrant user

Download the node exporter by executing the following

```
wget https://github.com/prometheus/node_exporter/releases/download/v1.5.0/node_exporter-1.5.0.linux-amd64.tar.gz
```

Extract it

```
tar xzvf node_exporter-1.5.0.linux-amd64.tar.gz
```

Navigate to the folder

```
cd node_exporter-1.5.0.linux-amd64/
```

Start the exporter with

```
./node_exporter &> /tmp/node-exporter.log &
```

Repeat the procedure on the other two machines (**kafka-2** and **kafka-3**) as well

Check that the exporter is working

Do it the first machine (**kafka-1**) by visiting the following address in a browser **http://192.168.99.101:9100/metrics**

Install Extractor (JMX)

Open a session to the first machine (**kafka-1**)

For this we will need a special exporter - **jmx_prometheus_javaagent**

It may be found here: https://github.com/prometheus/jmx_exporter

Make sure that you are in the folder where the **Apache Kafka** files are

Then, create a folder to store the **JMX Agent**

```
mkdir jmxagent
```

And navigate to it

```
cd jmxagent
```

Download the latest version of the agent

```
wget
https://repo1.maven.org/maven2/io/prometheus/jmx/jmx_prometheus_javaagent/0.17.2/jmx_p
rometheus_javaagent-0.17.2.jar
```

Download the example **YAML** configuration file as well

```
wget
https://raw.githubusercontent.com/prometheus/jmx_exporter/master/example_configs/kafka
-2_0_0.yml
```

Return to the **Apache Kafka** folder

Export a special environment variable

```
export KAFKA_OPTS="$KAFKA_OPTS -javaagent:jmxagent/jmx_prometheus_javaagent-
0.17.2.jar=9101:jmxagent/kafka-2_0_0.yml"
```

Just for the first machine (**kafka-1**) unset the **JMX_PORT** variable we created earlier

```
unset JMX_PORT
```

Stop the broker process

```
bin/kafka-server-stop.sh
```

And start it again

```
bin/kafka-server-start.sh -daemon config/server.properties
```

Repeat the procedure on the other two machines (**kafka-2** and **kafka-3**) as well

Check on the host if metrics are being generated

Do it for the first machine (**kafka-1**) by visiting the following address in a browser
http://192.168.99.101:9101/metrics

Install Prometheus

Open a session to the fourth machine (**monitoring**)

Download the latest **Prometheus** package

```
wget https://github.com/prometheus/prometheus/releases/download/v2.41.0/prometheus-
2.41.0.linux-amd64.tar.gz
```

Then extract it

```
tar xzvf prometheus-2.41.0.linux-amd64.tar.gz
```

Enter the folder

```
cd prometheus-2.41.0.linux-amd64
```

Basic Configuration

Open the sample configuration file

```
vi prometheus.yml
```

And make sure that it contains the following

```
global:
  scrape_interval: 30s
  evaluation_interval: 30s


scrape_configs:
  - job_name: 'prometheus'
    static_configs:
    - targets: ['monitoring:9090']
  - job_name: 'kafka-hosts'
    static_configs:
    - targets: ['kafka-1:9100']
    - targets: ['kafka-2:9100']
    - targets: ['kafka-3:9100']
  - job_name: 'kafka-jmx'
    static_configs:
    - targets: ['kafka-1:9101']
    - targets: ['kafka-2:9101']
    - targets: ['kafka-3:9101']
```

Save and close the file

Test the configuration for errors with

```
./promtool check config prometheus.yml
```

Start Prometheus

Start **Prometheus** with

```
./prometheus --config.file prometheus.yml --web.enable-lifecycle &>
/tmp/prometheus.log &
```

Check that it is working and reachable by visiting the following URL on your host

http://192.168.99.104:9090

Explore a bit

Install and Run Grafana

Information on how to install **Grafana** can be found here: **https://grafana.com/docs/grafana/latest/installation/**

Now, return on the fourth machine (**monitoring**) and make sure that you are in the home folder of the vagrant user

Installation

There are multiple ways. Here we will download and install a package

CentOS

Download the package

```
wget https://dl.grafana.com/oss/release/grafana-9.3.2-1.x86_64.rpm
```

And install it

```
sudo yum install grafana-9.3.2-1.x86_64.rpm
```

Debian/Ubuntu

Add a few prerequisites

```
sudo apt-get install -y adduser libfontconfig1
```

Download the package

```
wget https://dl.grafana.com/oss/release/grafana_9.3.2_amd64.deb
```

And install it

```
sudo dpkg -i grafana_9.3.2_amd64.deb
```

Running

Update services information

```
sudo systemctl daemon-reload
```

Start and enable the service

```
sudo systemctl enable --now grafana-server
```

Check its status

```
sudo systemctl status grafana-server
```

Open a browser tab and navigate to http://192.168.99.104:3000/login

The default credentials are **admin** / **admin**

We will be asked if we want to change the password

Should you want, then do it. Set a password you like and hit **Submit**

Connect to Prometheus

Once in, we must add a data source (in our case **Prometheus**)

Let's go to **Configuration** > **Data sources**

Click the **Add data source** button

Select **Prometheus**

Leave the name as it is (or change it if you like)

Set the **URL** to **http://192.168.99.104:9090/**

Hit the **Save & test** button

Now, we have our data source

Explore the Data Source

Click the **Explore** option in the left menu

Switch to the data source we registered earlier (for example, **Prometheus**)

Make sure in the **Code** view (can be switched from the two buttons that are on the far right)

Click the **Metrics browser** link (on the left of the text box)

We can see many familiar and unfamiliar metrics here

Select **kafka_server_replicamanager_partitioncount**

We can apply some filtering

For example, on the instances

Then select **instance(3)** in the labels section (on right)

Then select the individual node values

Or we can leave it as it is and include everything

While we make selections, we can see the query that was built

Once done, click the **Use query** button

And bam, we have a chart 😊

We can experiment with the graph types

We can then click **Inspector** to explore what is happening

Then, we can click the **Query history** button to explore this as well

Return to the **Home** screen

Basic Visualization

Preparation

In a new or existing session to one of the Kafka nodes check how many active topics we have with

```
bin/kafka-topics.sh --describe --bootstrap-server localhost:9092
```

Now, change the producer application to emit messages to at least two of them

Open the file

```
vi /vagrant/code/producer.py
```

And change the fifth line to match the following (for topics **demo2** and **demo3**)

```
tpk = ['demo2', 'demo3']
```

Save and close the file

Finally, start the producer

```
python3 /vagrant/code/producer.py
```

Actual Work

Go to **Dashboards** > **New dashboard** in the left menu

Then click **Add a new panel**

Switch the visualization from **Timeseries** to **Stat** (top right corner)

Click the **Metrics browser**

Select **kafka_server_brokertopicmetrics_messagesin_total** item

Click **Use query**

A visualization will appear

Click the **Save** button (top right corner)

Enter **Messages Per Topic** in the **Dashboard name** field

Click the **Save** button

Now, click the **Dashboard settings** button (top right corner)

Click on **Variables**

Then click the **Add variable** button

Enter **instancename** in the **Name** field

Enter **label_values({job="kafka-jmx"}, instance)** in **Query**

Select the **Include All option**

A preview of the values will appear at the bottom

Click the **Apply** button

Click the **Save dashboard** button

Enter **A variable added** in the comment box of the **Save dashboard** dialog and click **Save**

Now, click the **Back** arrow (top left corner) to return to the dashboard

You will see the variable as a drop-down list just above the panel

Select the **All** option in it

Click the **Panel Title** and select **Edit**

Now, in the right panel, go to the **Panel options** section

Change the **Title** to **Messages In**

Scroll down to the **Repeat options** section

In the dropdown list select **instancename** (our variable)

Change the **Max per row** value to **2**

Now, change the expression to

```
sum (kafka_server_brokertopicmetrics_messagesin_total{instance="$instancename"})
```

Change the time period to Last 15 minutes (top part of the screen)

Click **Apply** (top right corner)

Change the value in the drop-down **instancename** list to one of the instances and then to **All**

Three visualizations will appear

Click the **Save dashboard** button

Select the **Save current time range as dashboard default** option and click **Save**

Go to dashboard settings

Go to **Variables**

Select our variable

Change the **Show on dashboard** option to **Nothing**

Click the **Apply** button

Click **Save dashboard** and select the option in the save dialog

Click again **Save**

Click the **Back** arrow

Now, everything seems to be just fine

We've made it 😊

Click on the **Grafana** icon (top left) to return to the **Home** page

Select **Dashboards** > **Browse**

Click on our dashboard (you may have to expand the **General** node)

It opens and looks perfect

Now, we can add it to a playlist if we want