Inhoud

[Getting started with Kafka 2](#_Toc474153786)

[Install the VM 2](#_Toc474153787)

[Getting to know the VM 3](#_Toc474153788)

[Optional: Set Networking in VM 6](#_Toc474153789)

[Getting started with Kafka 8](#_Toc474153790)

[Create a topic 8](#_Toc474153791)

[Produce a message 9](#_Toc474153792)

[Consume a message 12](#_Toc474153793)

[Availability 13](#_Toc474153794)

[Bonus 14](#_Toc474153795)

[Questions 14](#_Toc474153796)

[Java producer and client 15](#_Toc474153797)

[Produce a message 15](#_Toc474153798)

[Consume a message 16](#_Toc474153799)

[Questions 18](#_Toc474153800)

[Run Node.js Kafka Client on your host laptop 19](#_Toc474153801)

[Consume Message from Kafka in Node.JS 19](#_Toc474153802)

[Produce messages to Kafka from Node.JS 20](#_Toc474153803)

[Kafka REST 22](#_Toc474153804)

[Produce a message 22](#_Toc474153805)

[Create a consumer 23](#_Toc474153806)

[Bonus 25](#_Toc474153807)

[References 26](#_Toc474153808)

# Getting started with Kafka

## Install the VM

Running the VM requires 2Gb of RAM and approximately 7Gb of free HD space

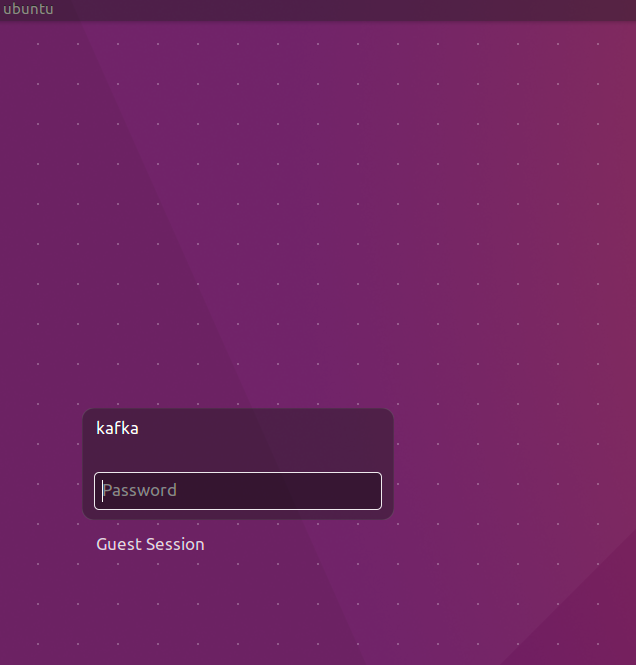
Download VirtualBox from <https://www.virtualbox.org>.

Start VirtualBox

Import the VM image by going to File, Import Virtual Appliance and browse to the image file. The VM requires 2Gb of RAM to run.

Start the VM

The VM will boot to a graphical login screen



The VM has two users: root and kafka. Both have password Welcome01.

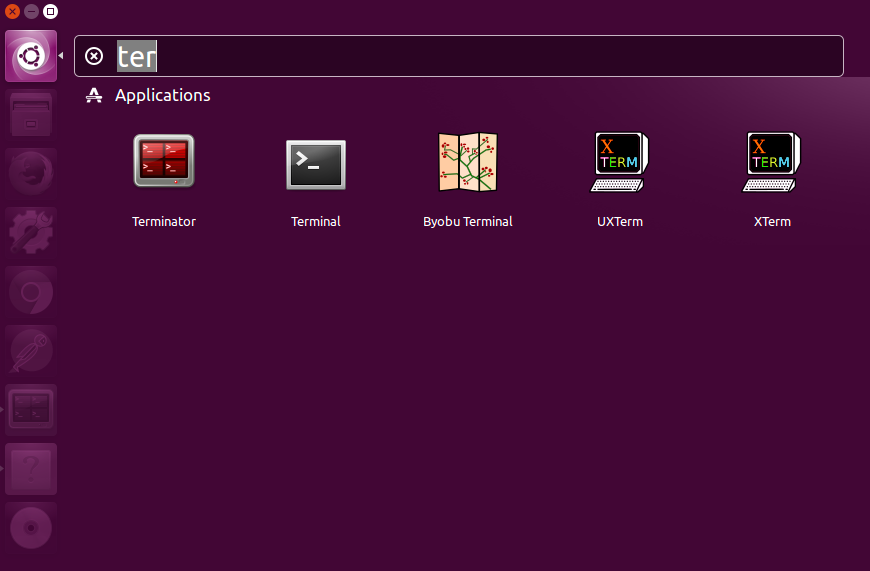
The VM contains:

* Ubuntu 16.04 LTS
* Oracle JDK 8
* Eclipse Neon.2
* Node 7.4
* Zookeeper
* Firefox
* Chrome
* Postman
* confluent-platform-oss-2.11
* kafka-manager 1.3.2.1. When started runs on <http://localhost:9000>
* kafkatool 1.0.1

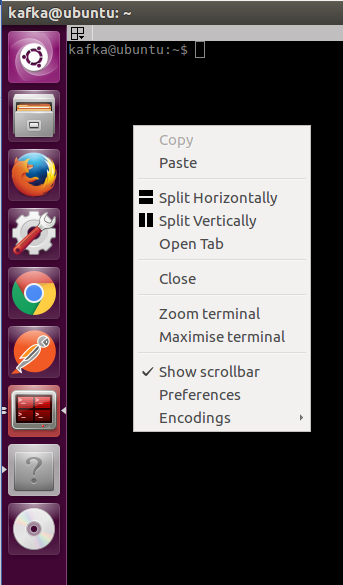
## Getting to know the VM

Login to the VM: kafka/Welcome01

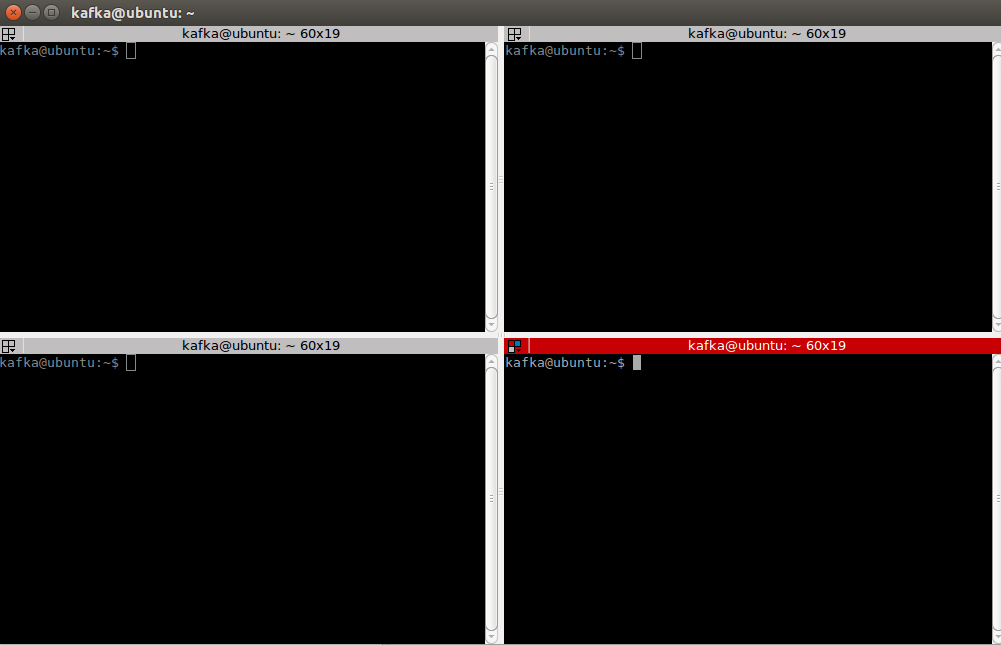
Start Terminator from the Unity menu



Split the screen horizontally – as shown in the next figure - and both screens vertically by right clicking in the window.



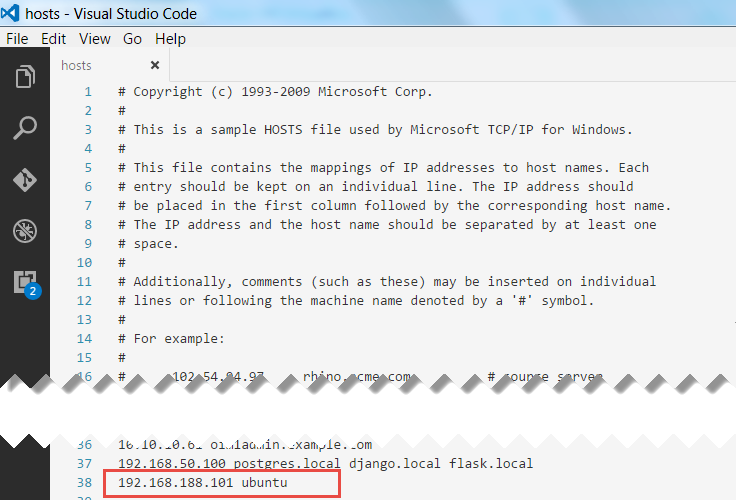
You will end up with four terminal windows.



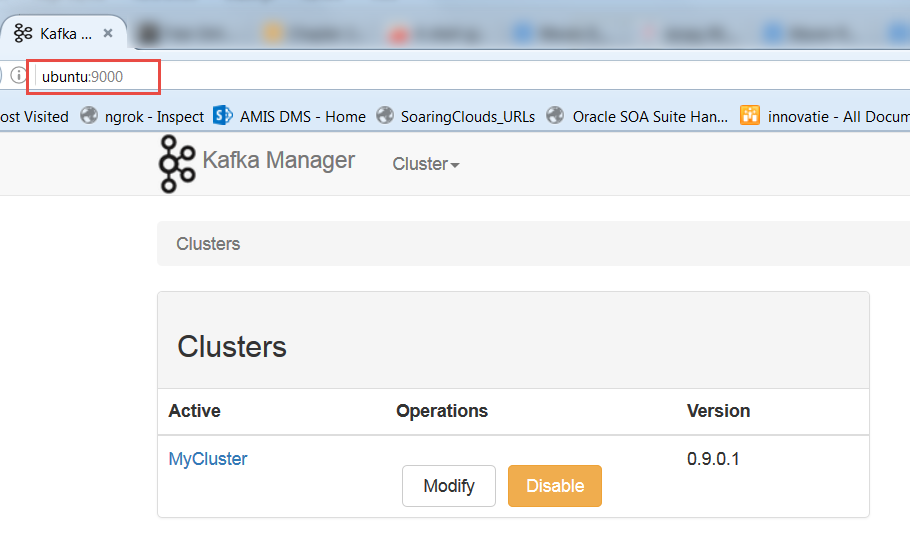
# Optional: Set Networking in VM

If and when you want to access the Kafka instance running inside the VM from your laptop, you need to ensure that you enable network access to the VM. The instructions in this article (AMIS Technology Blog: <https://technology.amis.nl/2017/01/29/network-access-to-ubuntu-virtual-box-vm-from-host-laptop/> ) show what you have to configure in order to be able to access the VM on its own IP address. Note that for this first series of labs, you do not need this. So you can skip this step for now, and optionally return to it later.

After arranging access to the VM, you may want to define a logical host name for the Virtual Machine – by adding an entry to the hosts file: C:\Windows\System32\drivers\etc\hosts, as shows below:



Note: ubuntu is the host name of the VM; to play nice with Zookeeper it seems convenient to use that same name as the host name in our hosts file.



# Getting started with Kafka

In the first terminal window start a Kafka broker:

kafka-server-start /etc/kafka/server.properties

Start in the second terminal window start the kafka manager:

kafka-manager

Start kafkatool in the third terminal window:

kafkatool

Minimize the tool. You will use it at a later time.

Continue in the fourth terminal window

### Create a topic

Create a topic:

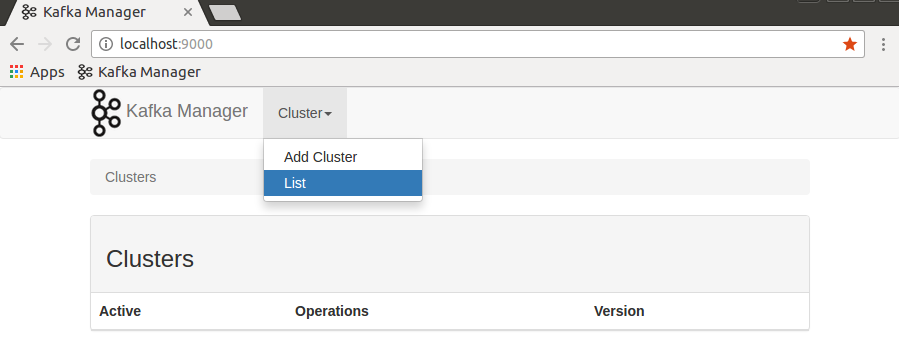
kafka-topics --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test

Confirm the topic is created:

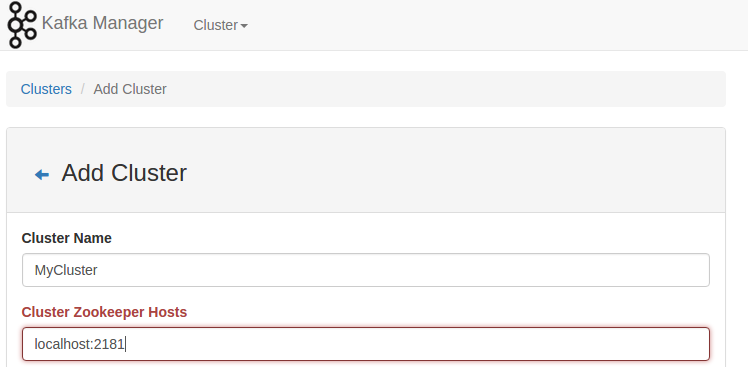
kafka-topics --list --zookeeper localhost:2181

Open a browser and go to <http://localhost:9000> and confirm the topic is created using a webinterface.

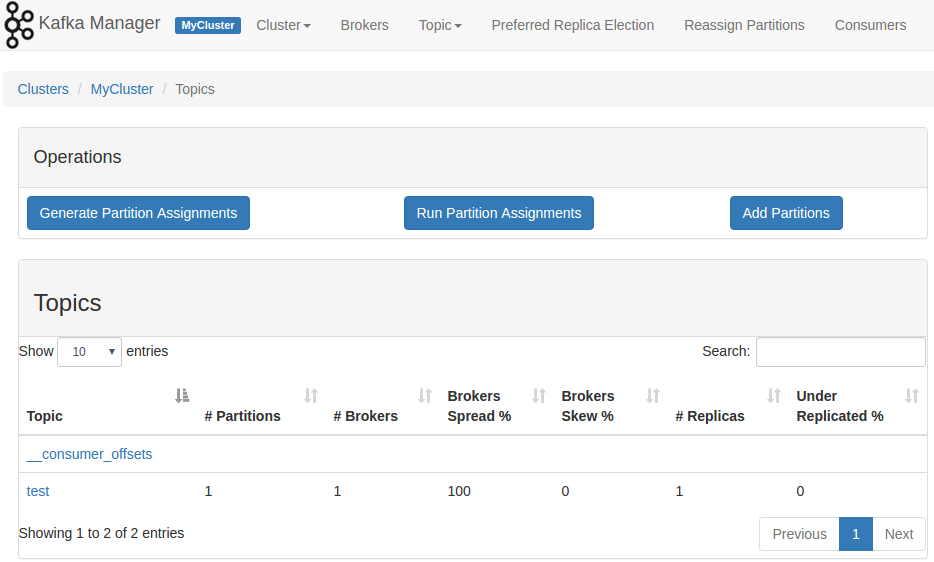
First create a cluster



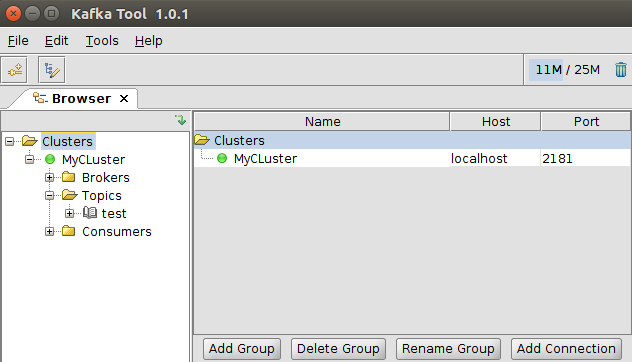
Name the cluster MyCluster and enter localhost:2181 as Zookeeper cluster. Scroll down and click save



Go to the cluster view and determine the topic has been created



Open kafkatool and confirm the topic is created



### Produce a message

Focus on the fourth terminal window again and type:

kafka-console-producer --broker-list localhost:9092 --topic test

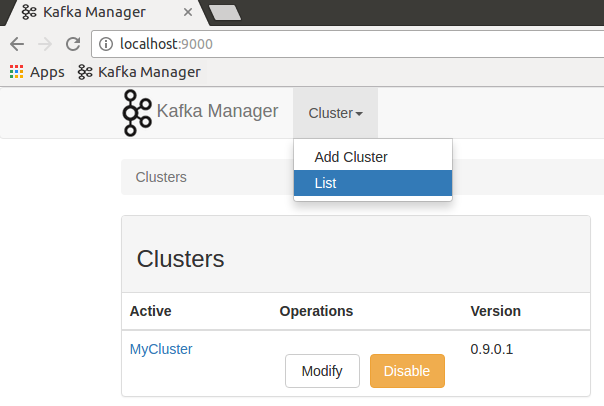
Press enter. No feedback appears; the cursor sits blinking and waiting for your input.

Type a message and press enter:

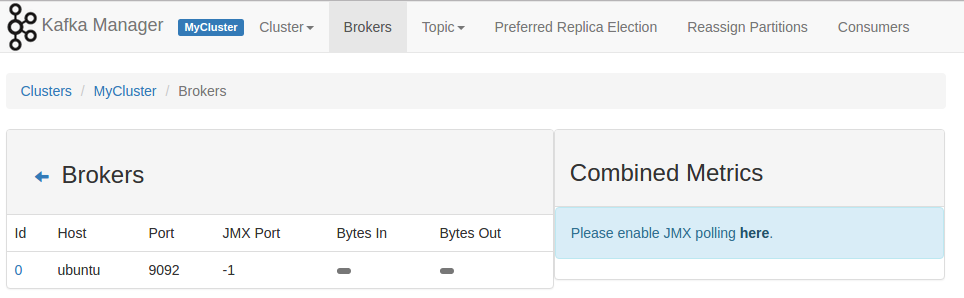
Hello World!

Check the message is created using kafka-manager and kafkatool.

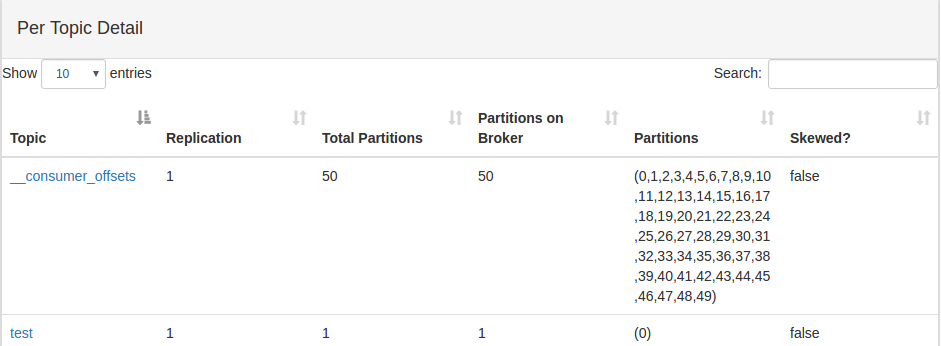
In the kafka-manager, go to Cluster, List and click the cluster you previously created



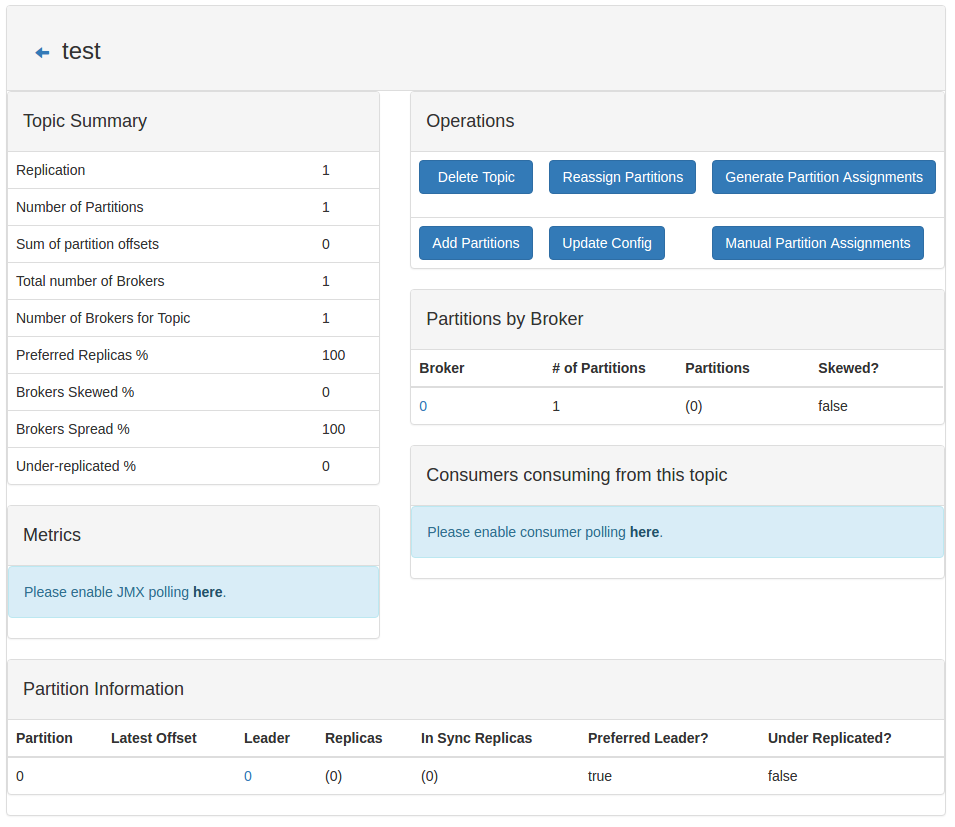
Click brokers, click 0



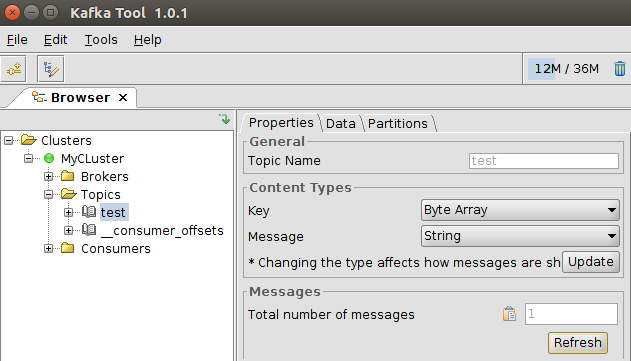
Scroll down and click the test topic



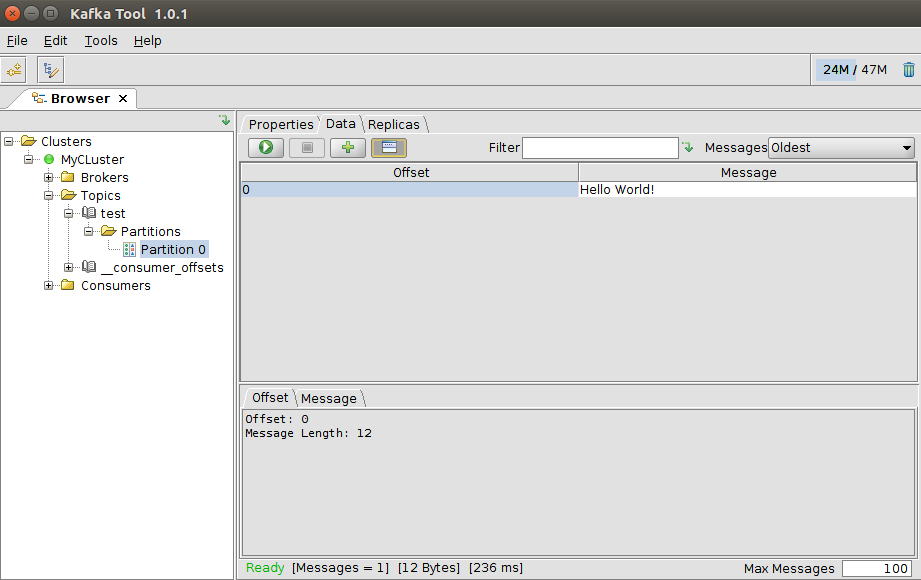
Scroll down again and confirm the latest offset is empty. No consumer has picked up messages yet.



Open kafkatool. Indicate the messages on the topic are string. Click the update button. Click the refresh button to view how many messages are present on the topic. Confirm the number is 1.



Open the partition under the test topic. Click the green play button. Confirm the Hello World! message has arrived.



### Consume a message

Consume a message

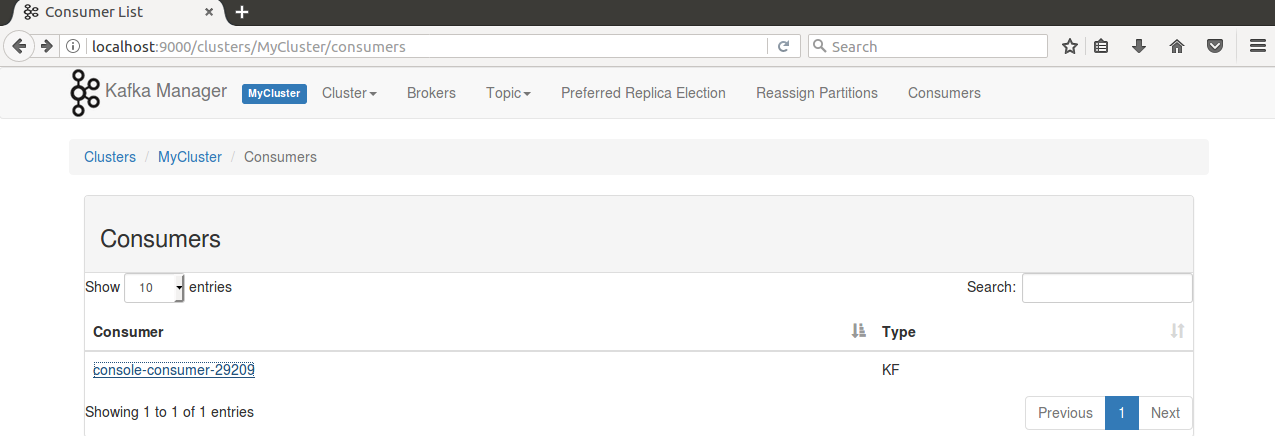
kafka-console-consumer --bootstrap-server localhost:9092 --topic test --from-beginning



Confirm that the previously posted message has been consumed.

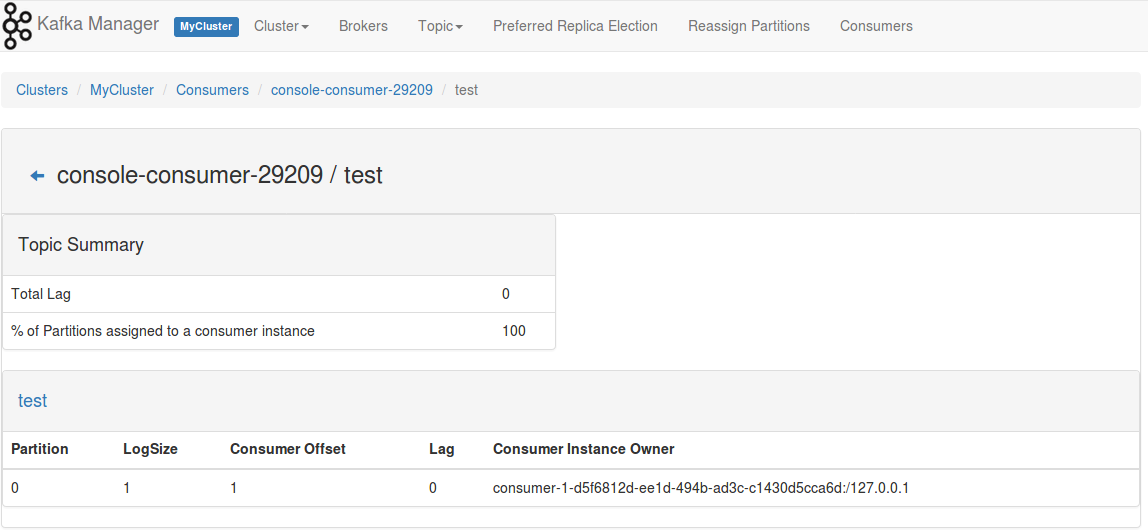
Look at kafka-manager: <http://localhost:9000>

Click on MyCluster, Consumers



Confirm the console consumer is visible

Click the consumer name. Next click the topic test.



Confirm that this consumer has an offset of 1.

End the console consumer by pressing ctrl-C in the terminal window where the consumer is running. End the console producer by pressing ctrl-C in the terminal window where the producer is running.

### Availability

Start a consumer

kafka-console-consumer --consumer-property group.id=TestConsumer --bootstrap-server localhost:9092 --topic test

Start a producer

kafka-console-producer --broker-list localhost:9092 --topic test

Send a message from the producer

Hello World!

Confirm the consumer receives the message.

End the consumer by pressing ctrl-C in the consumer console window

Produce a new message

Hello World!

Start the consumer

Confirm the consumer receives the message it had missed while being down.

### Bonus

Remove the consumer offset registrations.

kafka-consumer-groups --delete --zookeeper localhost:2181 --topic test

Remove the topic

kafka-topics --delete --zookeeper localhost:2181 --topic test

### Questions

* Why does the consumer not pickup missed messages when the group.id is not specified?
* Where is the offset for the consumers stored? Zookeeper, the consumer, the producer or the broker?

# Java producer and client

Start the Kafka broker if it is not already running

kafka-server-start /etc/kafka/server.properties

In a second terminal window start eclipse

cd ~/eclipse

./eclipse

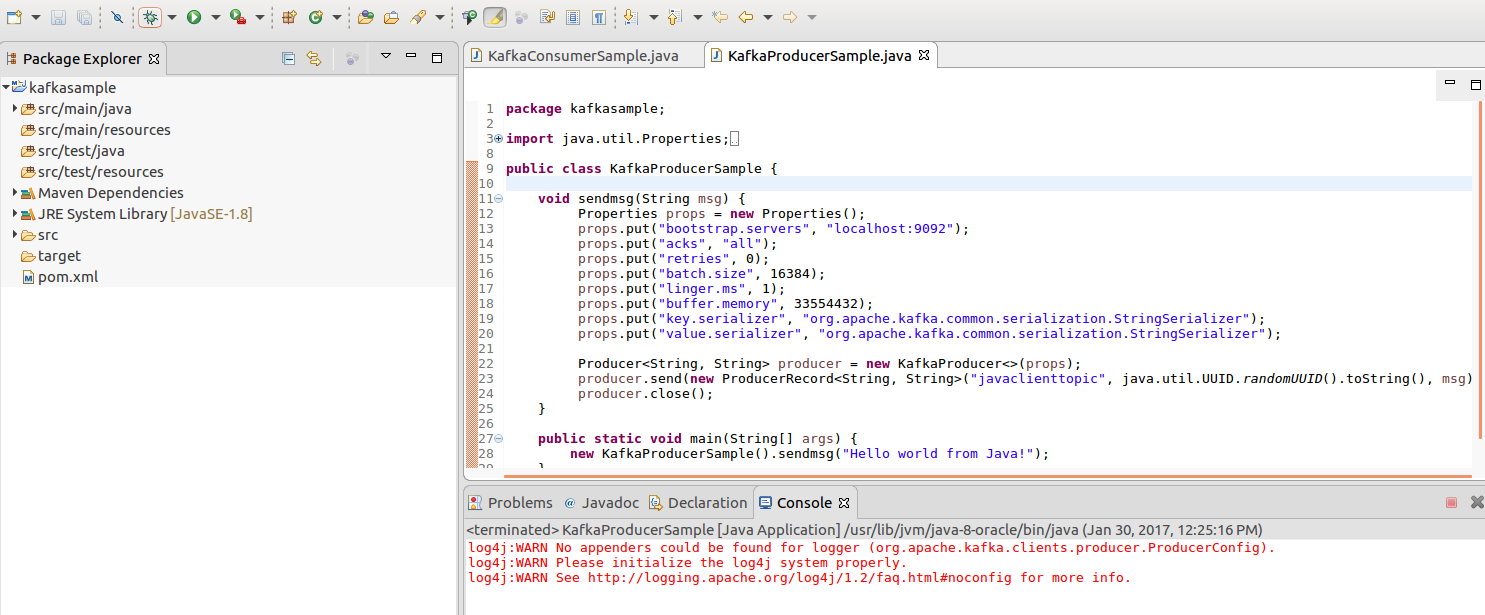
In a third terminal start kafkatool

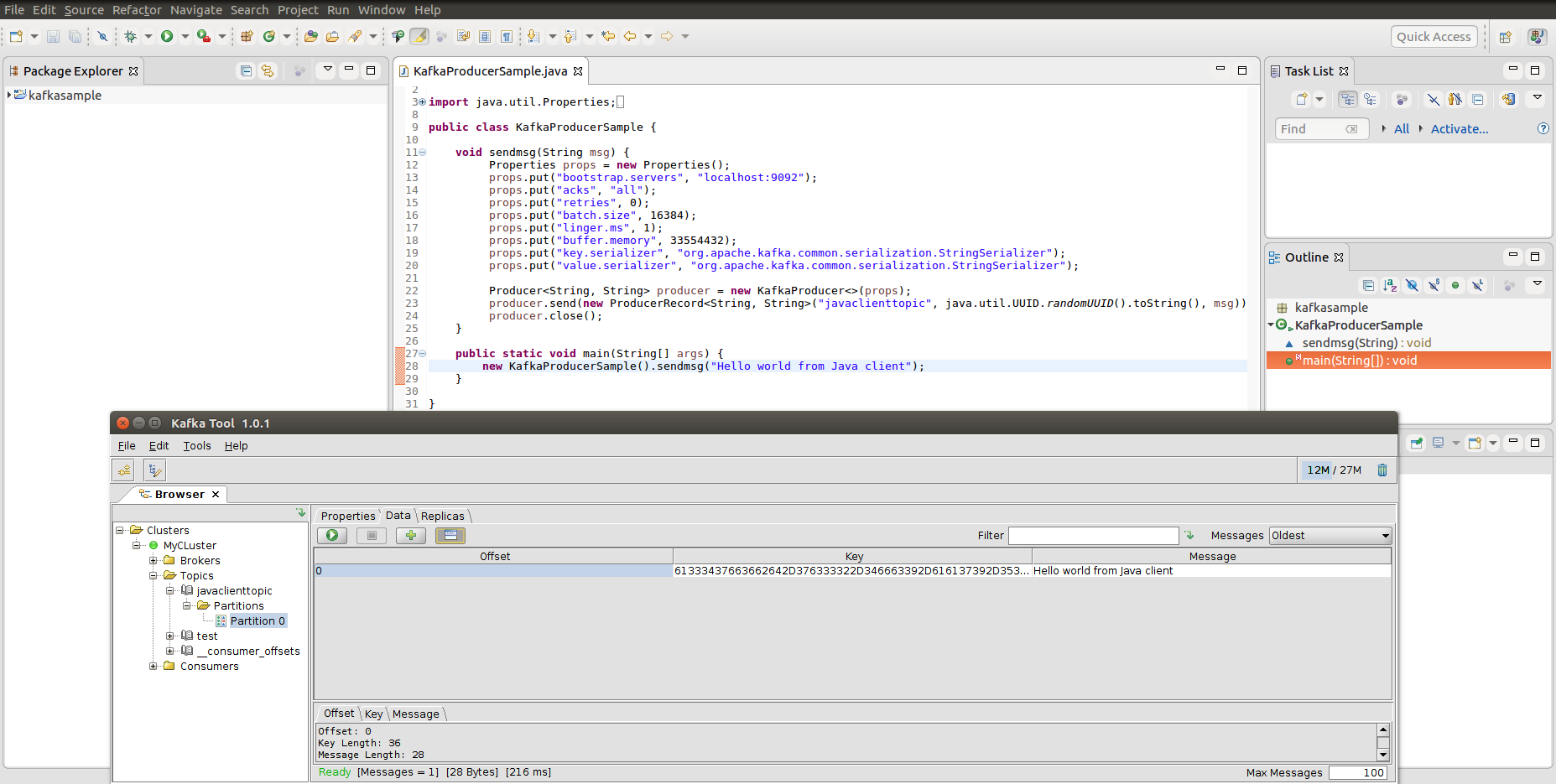
kafkatool

You can run the below code in the VM or outside within your own IDE. The sources can be downloaded from: https://github.com/MaartenSmeets/kafka-workshop

## Produce a message

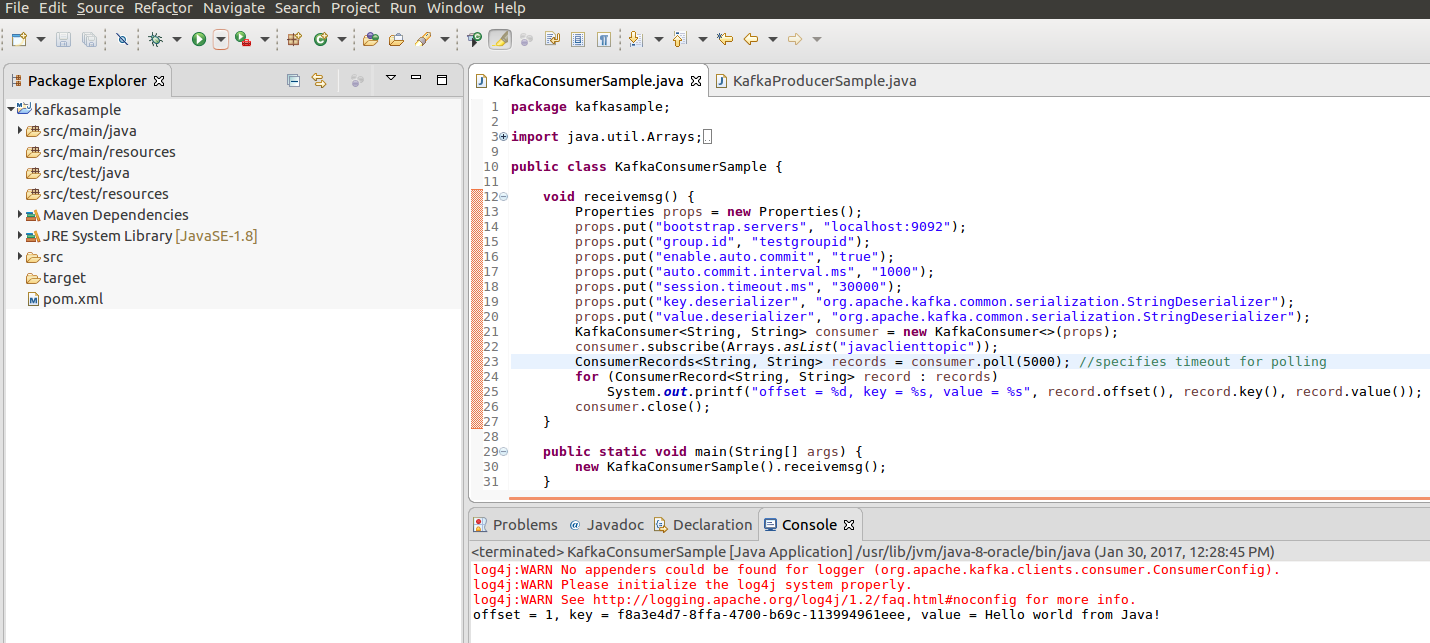
Run the Java producer (green play icon, run as Java application) and confirm message arrives on the topic specified with kafkatool. When running the producer, no output is produced.



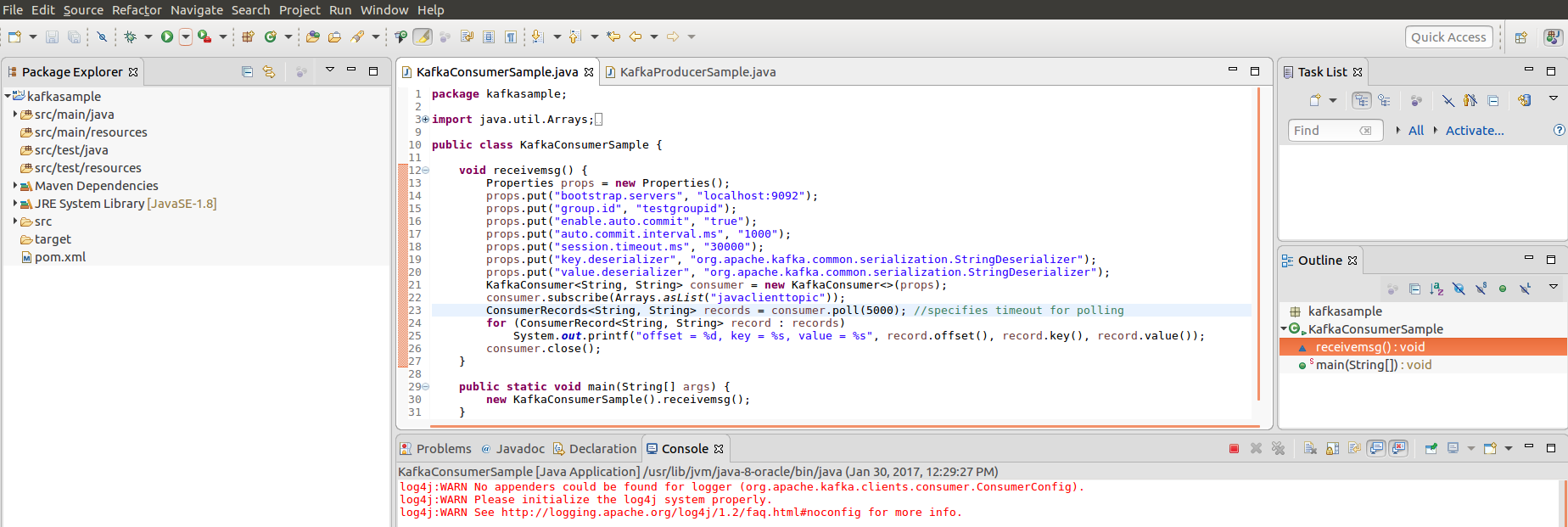


## Consume a message

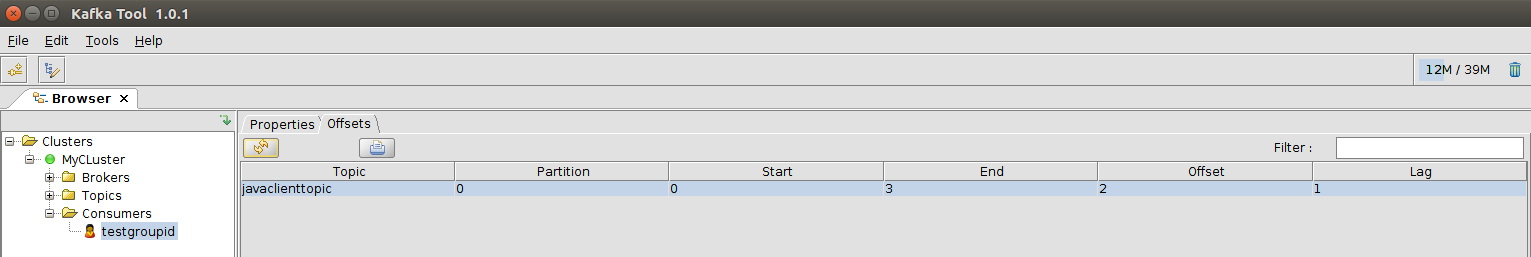
Run the Java consumer. Confirm in the console the message is consumed.



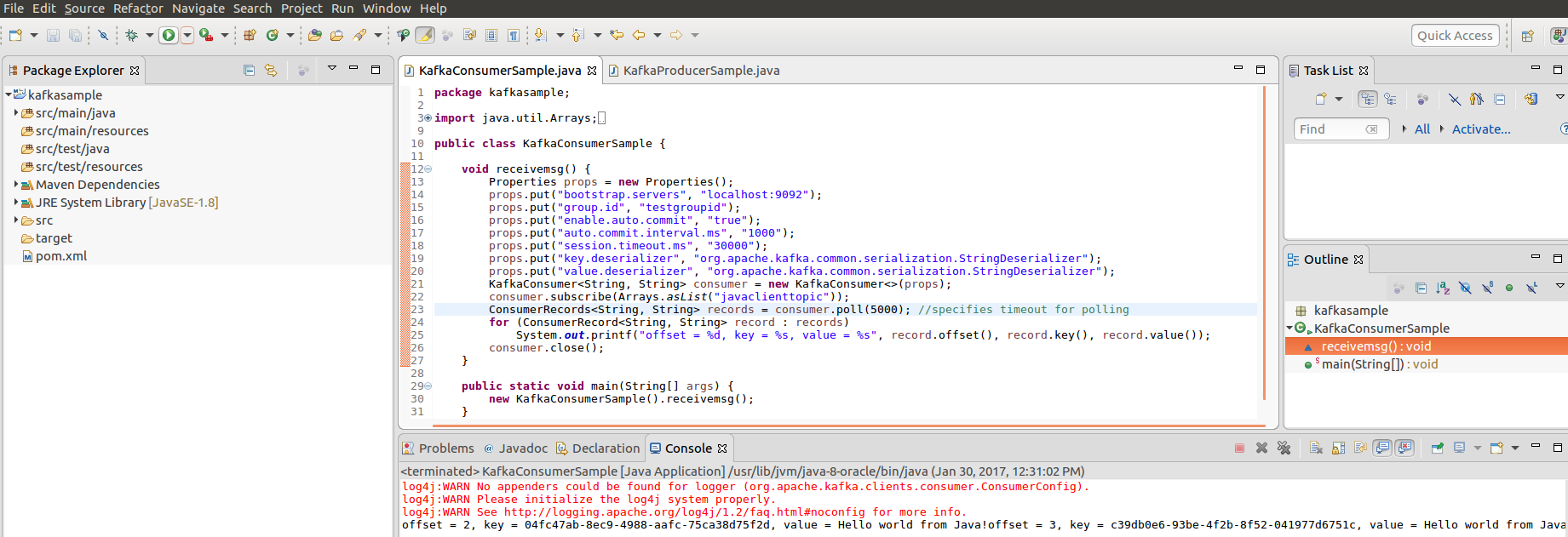
Run the consumer again. This time no messages are consumed. Why? How does the consumer know which offset to use?



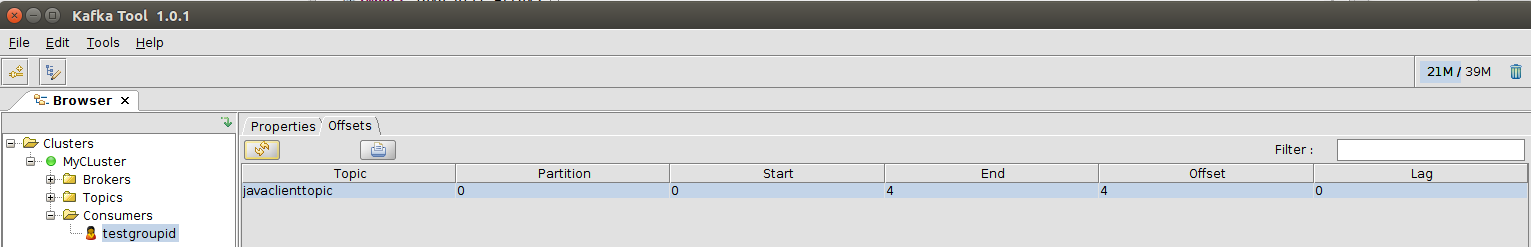
Produce a new message and check the offset and lag for the consumer in kafkatool



Produce a new message and run the consumer once. Confirm two messages are picked up



Confirm the lag has disappeared (is 0) for this consumer in kafkatool



Confirm the offset with a command-line

kafka-run-class kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic javaclienttopic --time -1

## Questions

What do the different timeout values in the consumer mean? Play with them and check the behavior. For example make the poll timeout in the consumer really small (1). What happens? Why?

# Run Node.js Kafka Client on your host laptop

As an example of how easy communicating with Kafka can be – even from other hosts than those running a node in the Kafka cluster – this section discusses running a Kafka client in Node.js. This will show how a Node.js application can easily produce messages to Kafka and consume messages from Kafka.

Prerequisites:

you need to have Node.JS and npm installed, either on your laptop host or inside the VM (where it is already installed) – and at least Node.JS release 4.4.7 at the time of writing. To install or upgrade Node.JS, go to <https://nodejs.org/en/download/> . The latest stable release at the time of writing is v6.9.4

Create a directory where you want to create new the Node.JS application to interact with Kafka. Optionally, create a new Node application with:

npm init

Open the command line window (a terminal window) for this directory. Install kafka-node, the Node.JS library for Kafka:

npm install kafka-node --save



## Consume Message from Kafka in Node.JS

Next, create a new file: KafkaConsumer.js. Add the following code to the file:

var kafka = require('kafka-node')

var Consumer = kafka.Consumer

var client = new kafka.Client("ubuntu:2181/")

var consumer = new Consumer(

client,

[],

{fromOffset: true}

);

consumer.on('message', function (message) {

console.log("received message", message);

});

consumer.addTopics([

{ topic: 'test', partition: 0, offset: 0}

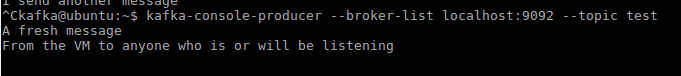
], () => console.log("topic added"));

Check if the topic name is correct and if the host (*ubuntu* – the logical name for the VM running the ZooKeeper for the Kafka cluster) and the port are correct. They should be if you have followed the steps in this tutorial.

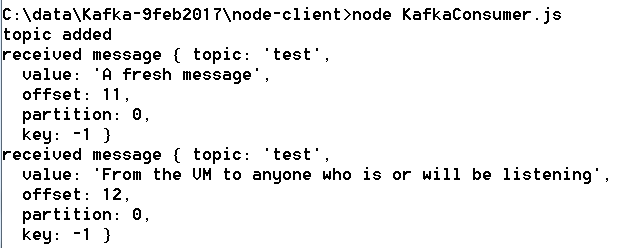
Run this simple Node application; from the command line:

node KafkaConsumer.js

The messages you have send to the topic earlier on in this tutorial are shown – all of them. If you end the program and then start it again – none of the messages are shown. This consumer is now up to date. If you were to add a new message to the topic, for example from inside the VM,



this new message will be shown:



## Produce messages to Kafka from Node.JS

Create a new file in the same directory, called KafkaProducer.js. Add the following content:

var kafka = require('kafka-node')

var Producer = kafka.Producer

var client = new kafka.Client("ubuntu:2181/")

KeyedMessage = kafka.KeyedMessage,

producer = new Producer(client),

km = new KeyedMessage('key', 'message'),

payloads = [

{ topic: 'test', messages: 'hi from Windows Host', partition: 0 },

{ topic: 'test', messages: 'hi from node producer', partition: 0 },

{ topic: 'test2', messages: ['hello', 'world', km] }

];

producer.on('ready', function () {

producer.send(payloads, function (err, data) {

console.log(data);

});

});

producer.on('error', function (err) {})

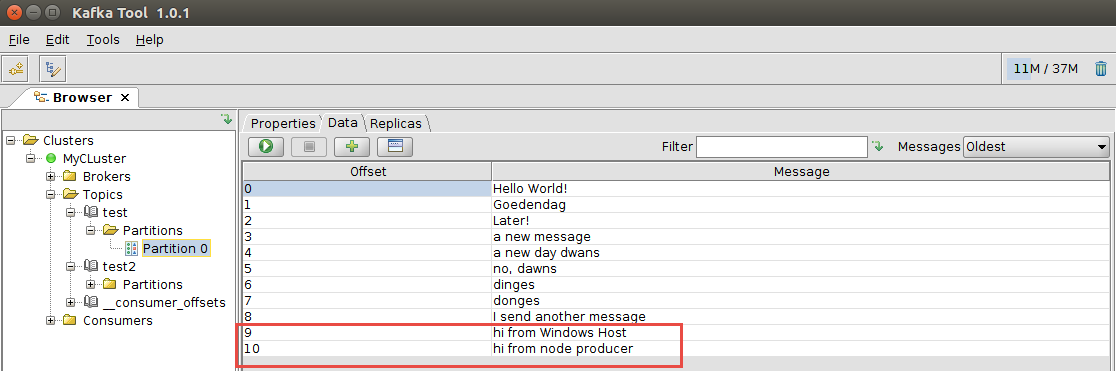
Again, verify the names of the topics (test and test1) and the name of the host running ZooKeeper and the port it is running on.

Now run the program:

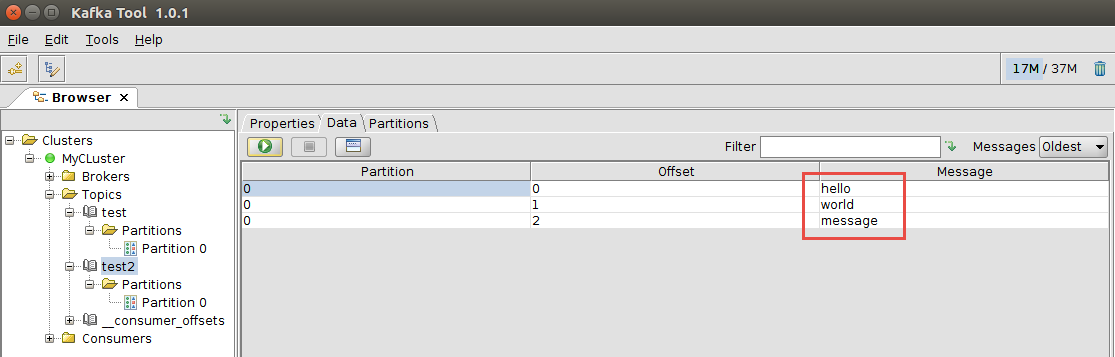
node KafkaProducer.js

Several messages are published to the two topics test and test2, as can be seen for example in Kafka Tool.

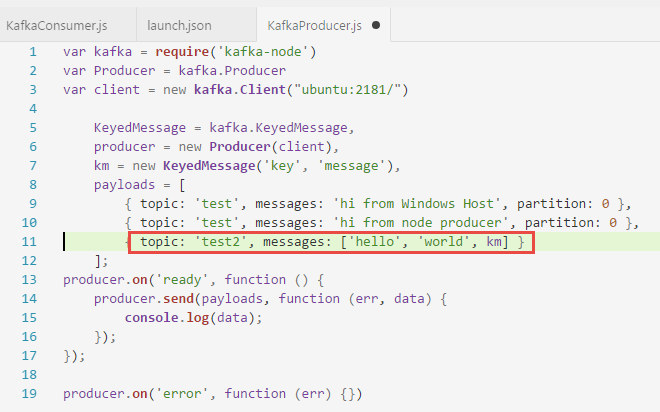
The first screenshot shows the messages in partition 0 of topic test:



and this figure the three messages on topic test2:



Corresponding with this line in the KafkaProducer.js program:



For more details on kafka-node: <https://www.npmjs.com/package/kafka-node> .

# Kafka REST

Start Kafka Server and Kafka REST

kafka-server-start /etc/kafka/server.properties

kafka-rest-start

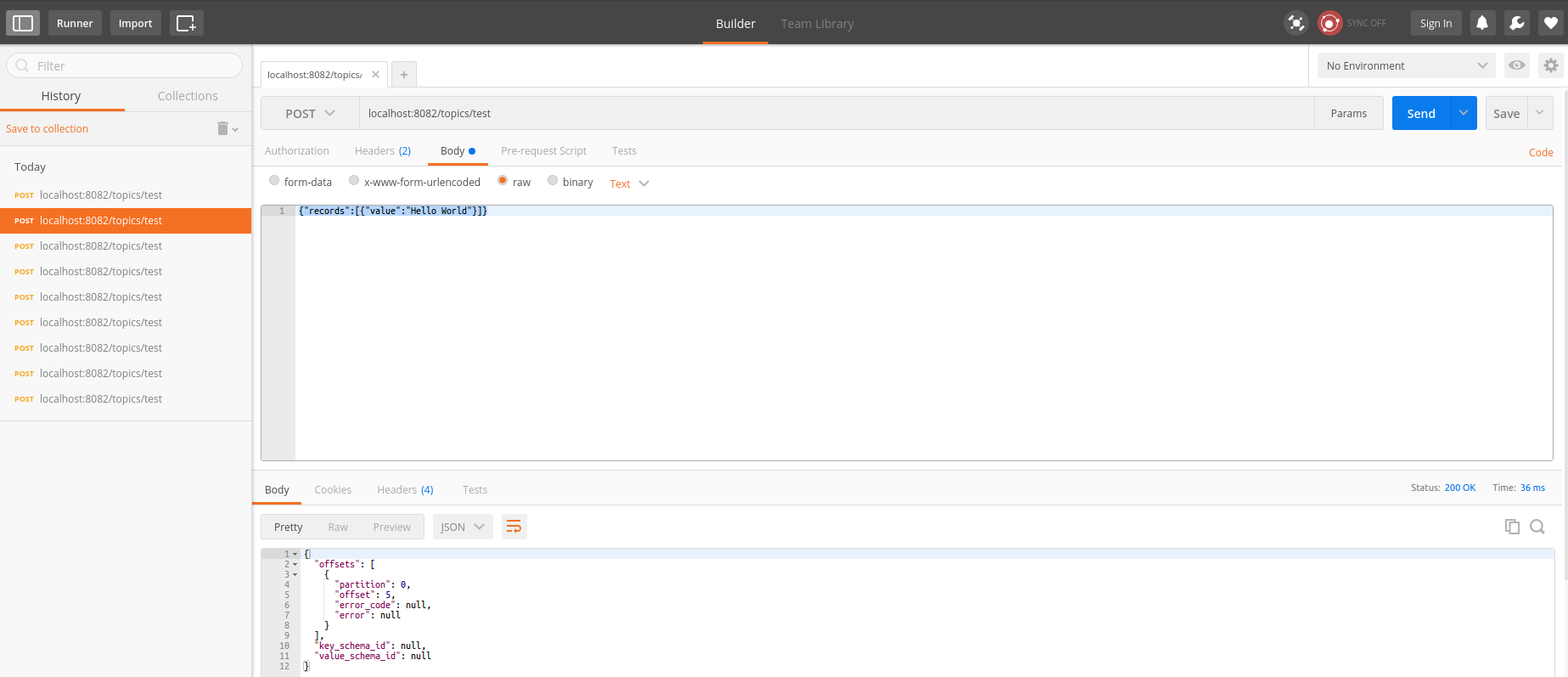
## Produce a message

Start Postman. Select POST and input localhost:8082/topics/test as URL

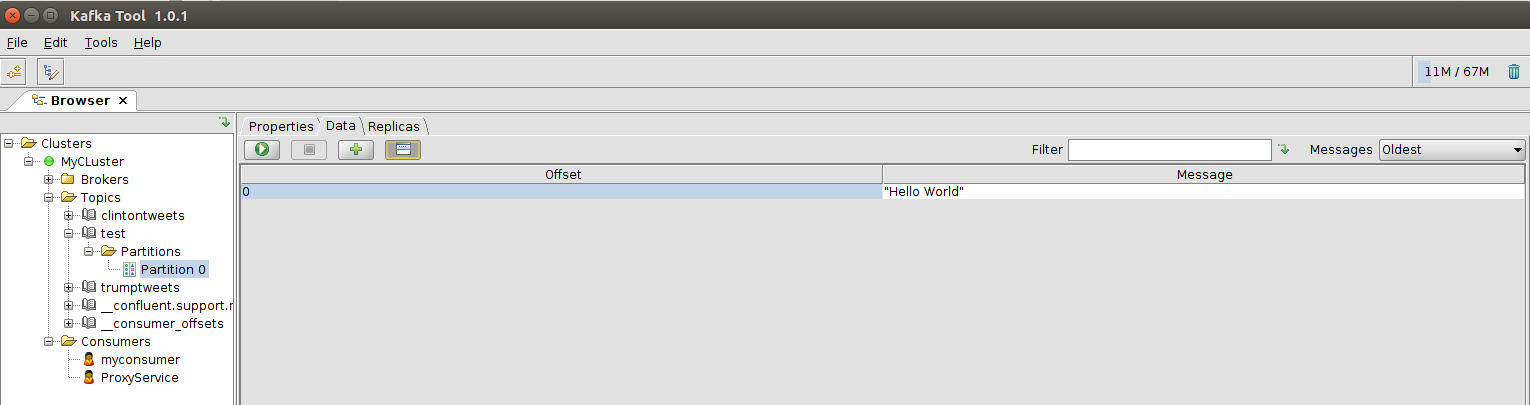
Add a Header: Content-Type application/vnd.kafka.json.v1+json

Use the body: {"records":[{"value":"Hello World"}]}

Click send



Start Kafkatool and confirm the message has arrived.



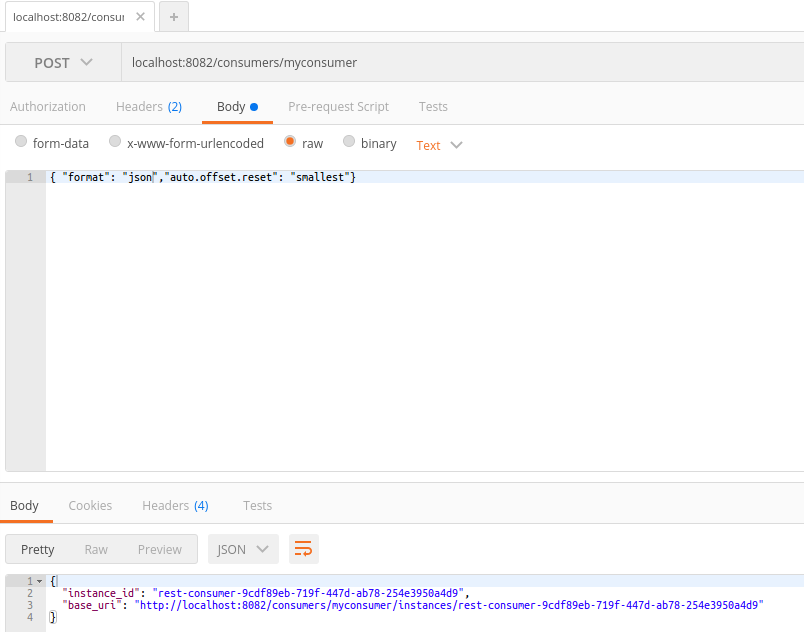
## Create a consumer

Go back to Postman

Do a POST request to localhost:8082/consumers/myconsumer

Use the following body

{ "format": "json","auto.offset.reset": "smallest"}



The response will be something like:

{

"instance\_id": "rest-consumer-a3f46a50-6fbf-4c5d-b48a-c576af53ecc6",

"base\_uri": "http://localhost:8082/consumers/myconsumer/instances/rest-consumer-a3f46a50-6fbf-4c5d-b48a-c576af53ecc6"

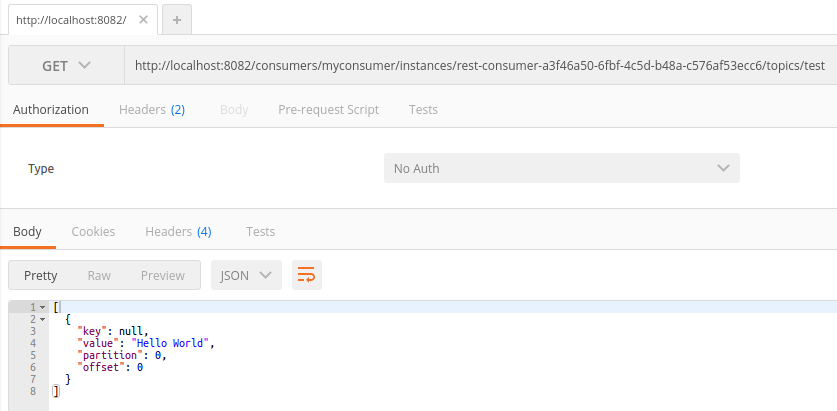
}

Do a GET request on (rest-consumer part is copied from the response below):

http://localhost:8082/consumers/myconsumer/instances/rest-consumer-a3f46a50-6fbf-4c5d-b48a-c576af53ecc6/topics/test

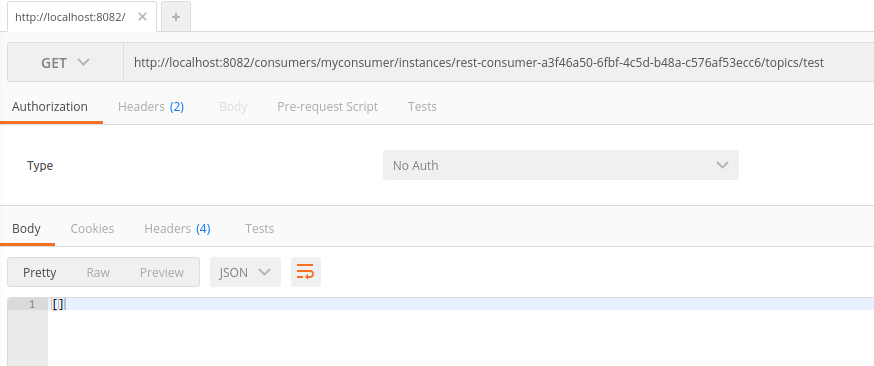
Specify Accept header application/vnd.kafka.json.v1+json

Confirm you can receive the response:



Repeat the request

Confirm you are not receiving new messages



## Bonus

Repeat this exercise with binary messages. See <https://github.com/confluentinc/kafka-rest> for inspiration.

Repeat the exercise using Node.js and kafka-rest-node. See <https://github.com/confluentinc/kafka-rest-node>.

# References

Kafka OSA:

<https://www.rittmanmead.com/blog/2016/07/stream-analytics-processing-kafka-oracle-stream-analytics/>

Network access

<https://technology.amis.nl/2017/01/29/network-access-to-ubuntu-virtual-box-vm-from-host-laptop/>