pm

Maarten Smeets

Maarten.smeets@amis.nl

Kafka workshop

With Docker, Node.js

Contents

[Getting started with the VM 2](#_Toc525470178)

[Prerequisites 2](#_Toc525470179)

[Introducing the VM 3](#_Toc525470180)

[Getting started with Kafka 6](#_Toc525470181)

[Creating topics, producing and consuming messages 6](#_Toc525470182)

[Using Kafkatool 6](#_Toc525470183)

[Kafka clients 10](#_Toc525470184)

[Kafka and Node 10](#_Toc525470185)

[Setting up 10](#_Toc525470186)

[Kafka consumer 11](#_Toc525470187)

[Kafka producer 12](#_Toc525470188)

[Postman as Kafka client 14](#_Toc525470189)

[Produce a message 14](#_Toc525470190)

[Create a consumer 15](#_Toc525470191)

[Schema registry 18](#_Toc525470192)

[Kafka Connect 19](#_Toc525470193)

[Kafka Streams / KSQL 20](#_Toc525470194)

# Getting started with the VM

## Prerequisites

Download and install VirtualBox from [www.virtualbox.org](http://www.virtualbox.org).

The machine you’re running the VM on requires at least 5Gb of available RAM and around 10Gb (preferably 30Gb) of free disk space.

Install Vagrant. You can download it from <https://www.vagrantup.com/>

Download the Vagrantfile and provision.sh from <https://github.com/MaartenSmeets/provisioning/tree/master/ubuntudev> and put them in a directory of choice. Start a command prompt inside this directory (shift right click, open powershell window here or open command prompt here).

Install required plugins. These allow automatic install of the correct version of the guest additions and easy resizing of the disk.

vagrant plugin install vagrant-vbguest

vagrant plugin install vagrant-disksize

Start building the VM:

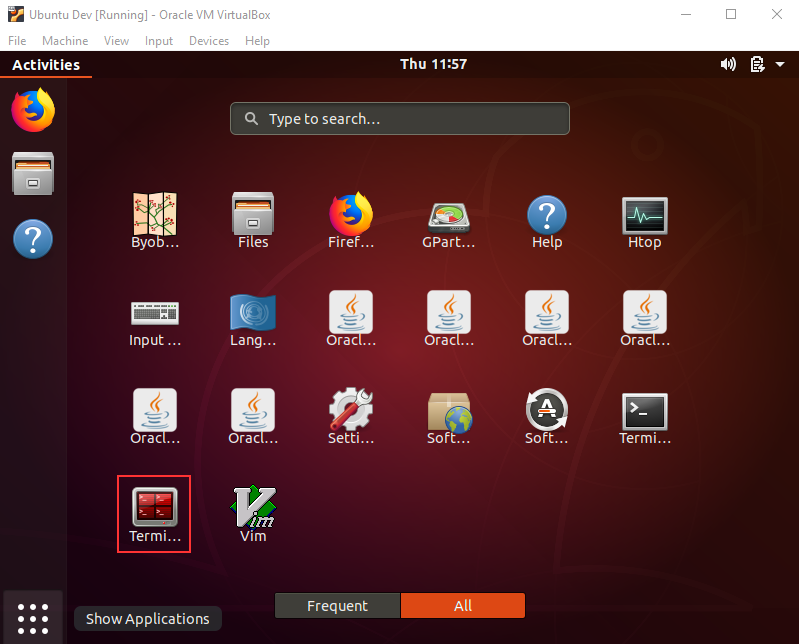
vagrant up

Wait until you have a running VM.

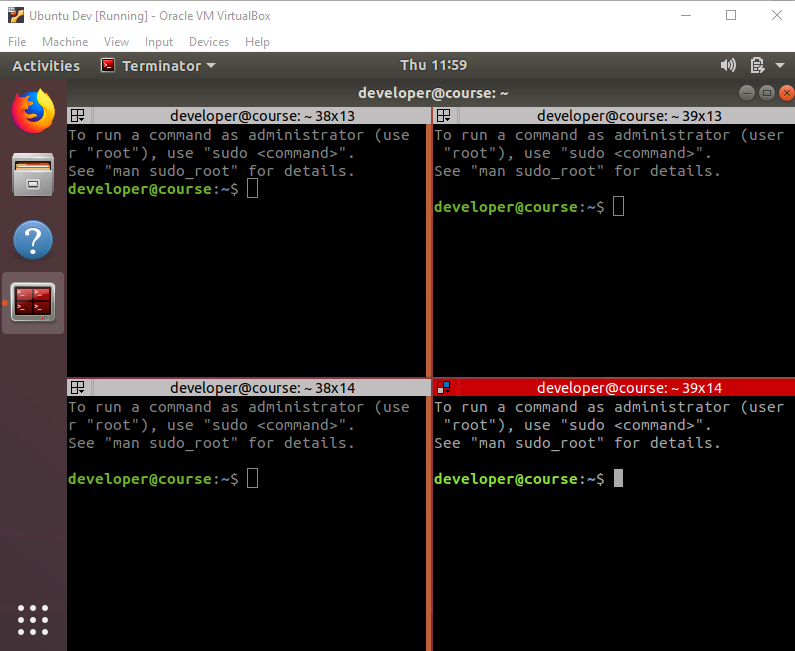
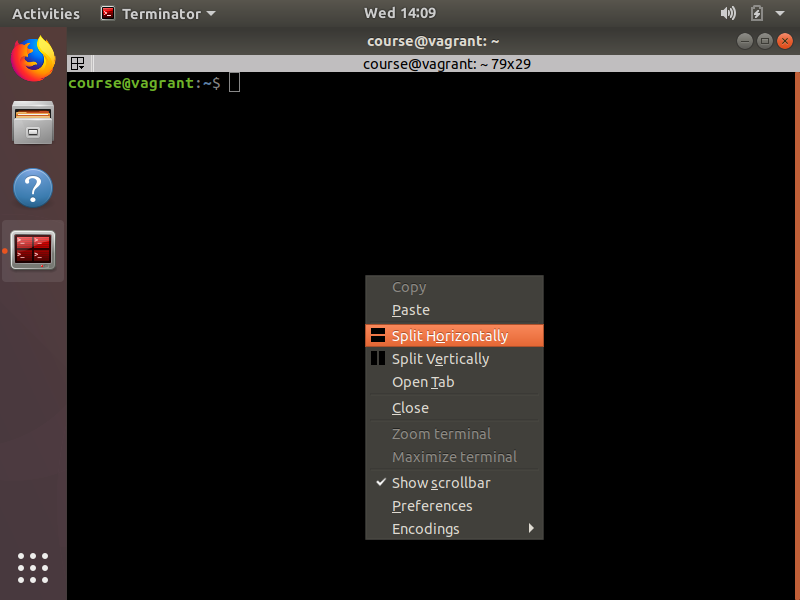
## Introducing the VM

Login to the VM with user developer password Welcome01. If at any time there is a request for a password, Welcome01 is the answer.

Start Terminator by clicking at the bottom left icon and clicking the Terminator icon

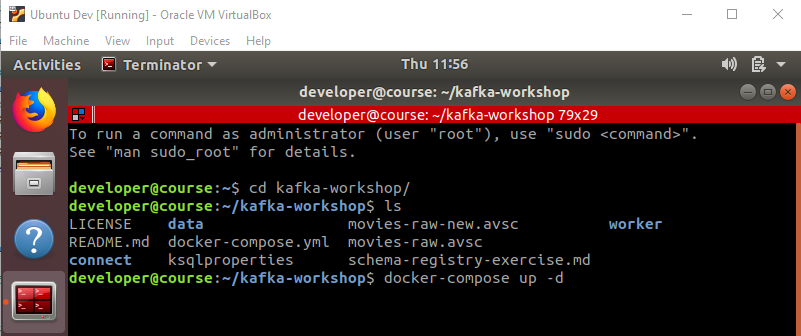


Terminator can easily split the terminal screens horizontally and vertically. Try this by rightclicking in the terminal window and split horizontally and then vertically.



Next go to the kafka-workshop folder (inspired on https://github.com/confluentinc/kafka-workshop) and start the download process:

docker-compose up -d



What is happening?

docker ps

What are these?

* kafkaworkshop\_connect\_1
* kafkaworkshop\_restproxy\_1
* kafkaworkshop\_ksql\_1
* kafkaworkshop\_kafka1\_1
* kafkaworkshop\_schemaregistry\_1
* kafkaworkshop\_database\_1
* kafkaworkshop\_zookeeper\_1

You can stop the containers by:

docker-compose stop

And start them again with

docker-compose start

# Getting started with Kafka

## Creating topics, producing and consuming messages

Enter the Kafka Docker container. If you want to exit the container, type exit followed by enter.

docker exec -it kafkaworkshop\_kafka1\_1 /bin/bash

cd /usr/bin

Suggested read: topic naming convention from: <https://medium.com/@criccomini/how-to-paint-a-bike-shed-kafka-topic-naming-conventions-1b7259790073>

kafka-topics --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic queuing.course.messages

Confirm the topic has been created

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

Send a message

kafka-console-producer --broker-list localhost:9092 --topic queuing.course.messages

Any sentence you type here followed by an enter will be send as a message to the specified topic.

Consume messages. Preferably in another console window so you can add messages in one terminal and see they are being consumed in another terminal window.

kafka-console-consumer --bootstrap-server localhost:9092 --topic queuing.course.messages --from-beginning

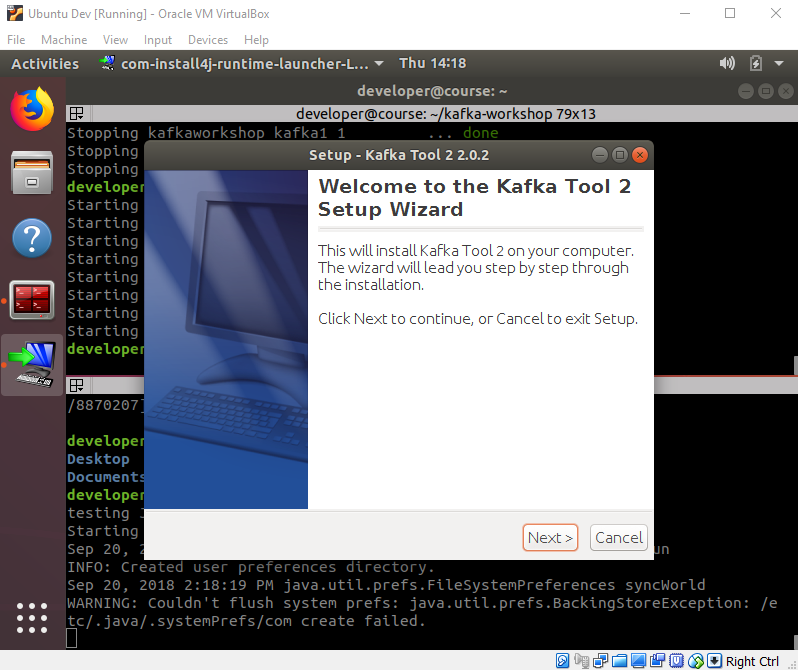
To stop a producers or consumer, do Ctrl-C.

## Using Kafkatool

Install kafkatool:

cd ~

sh ./kafkatool.sh

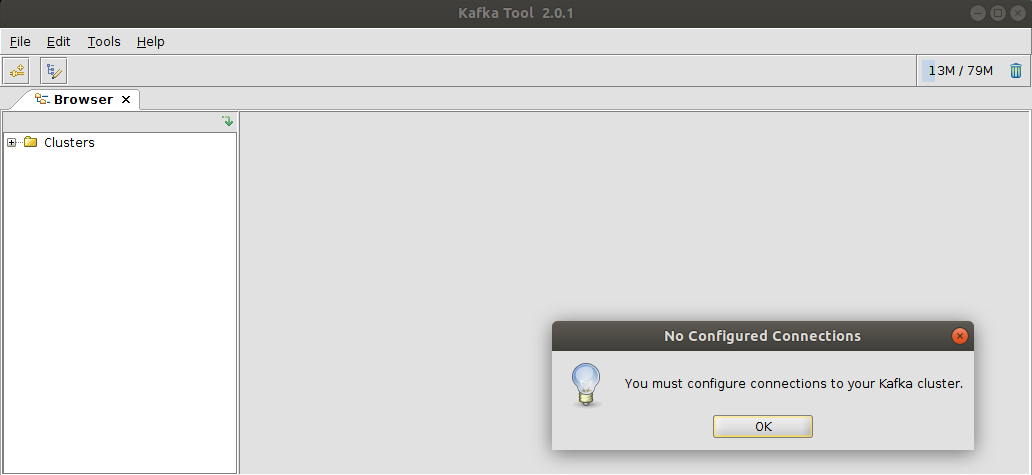


Next, Accept, Next, Next, Next, Next, Finish

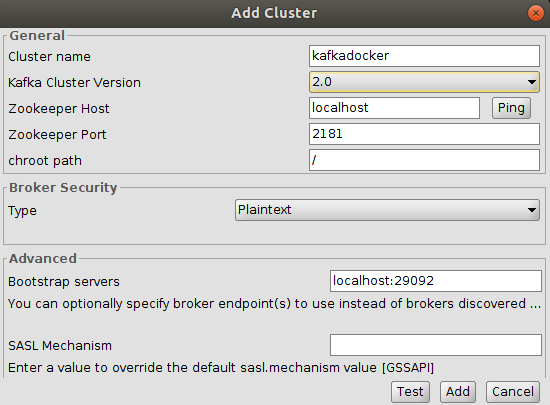
Start kafkatool

cd ~/kafkatool2

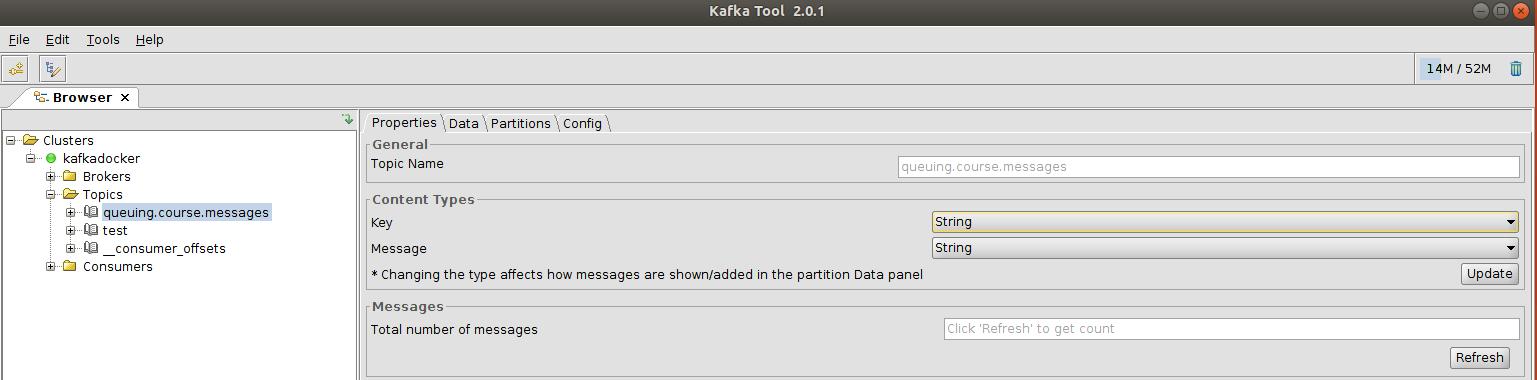
./kafkatool



Add a new cluster using below settings and Click the Ping button to confirm connectivity

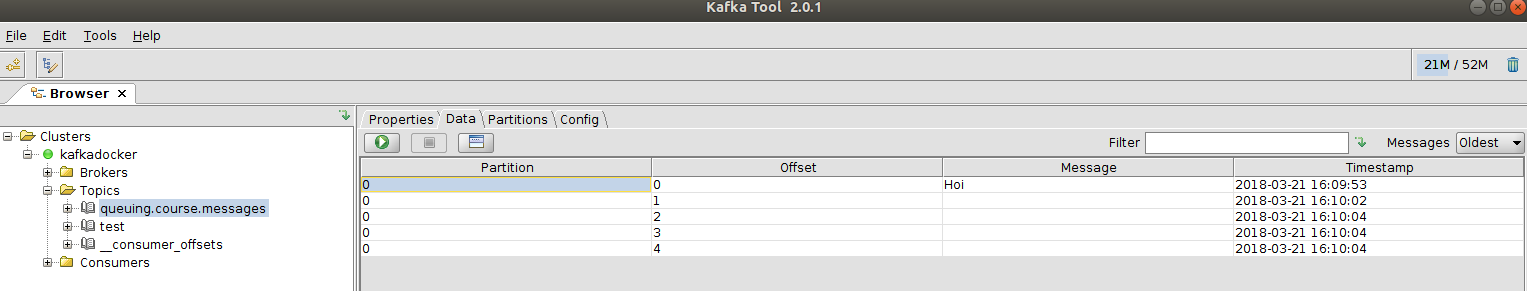


Set the type of the message and key of the queuing.course.messages to String, String.



Click Update, Refresh

Click Data, Click the green play icon and confirm your previously send messages are there



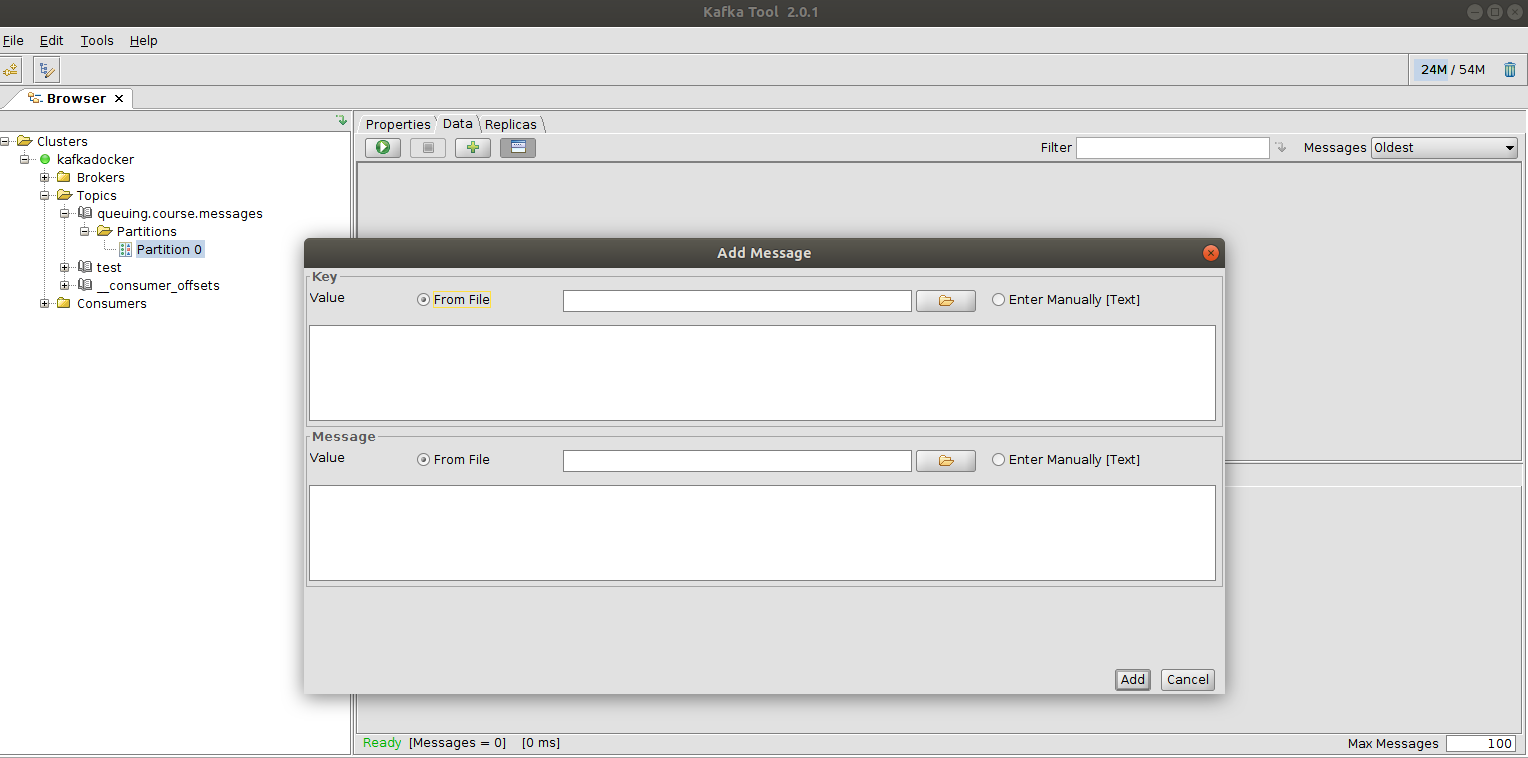
Start a consumer in a console window if you previously closed it.

docker exec -it kafkaworkshop\_kafka1\_1 /bin/bash

cd /usr/bin

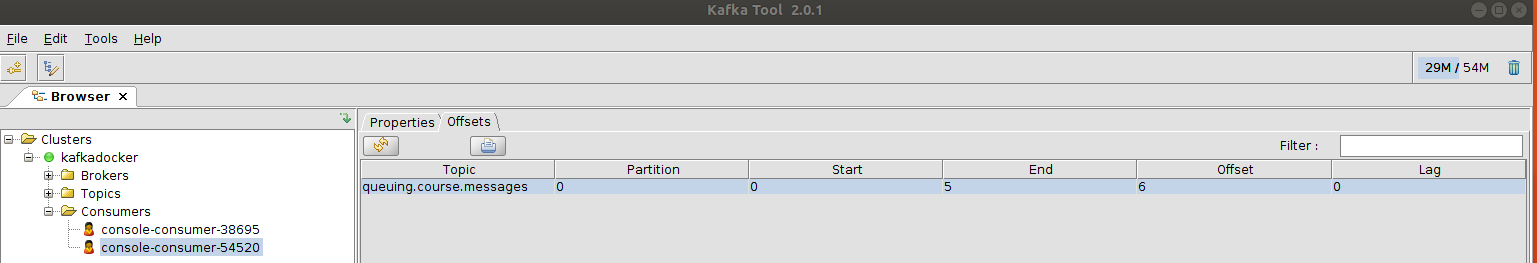
kafka-console-consumer --bootstrap-server kafka1:9092 --topic queuing.course.messages

Send a new message by opening a partition and clicking the + icon.



Notice the message is being picked up by the consumer

View the consumer offsets.



# Kafka clients

## Kafka and Node

### Setting up

Create a directorie for your application

mkdir ~/kafkaapp

cd ~/kafkaapp

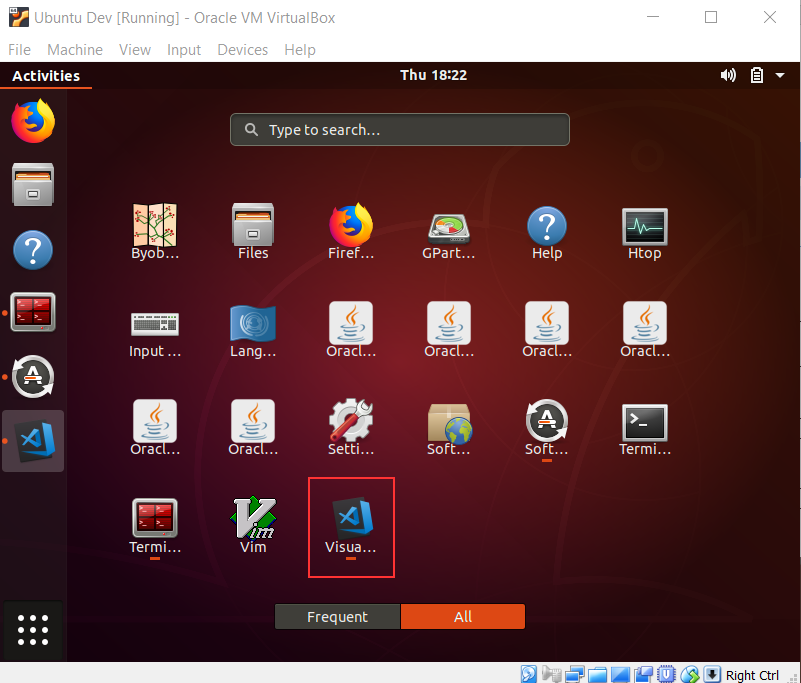
Create a default package.json

npm init -y

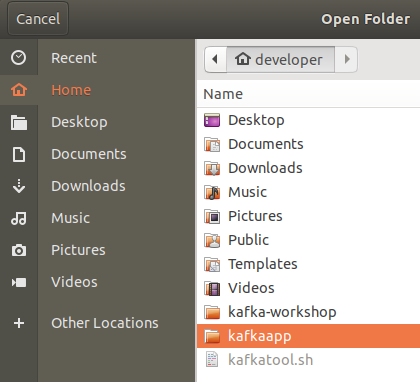
Add Kafka client libraries. –save adds them to package,json

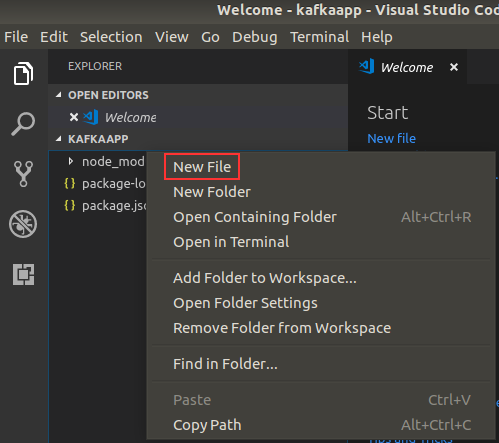
npm install kafka-node --save

Open Visual Studio Code



Open the folder you just created and create a new file app.js.





### Kafka consumer

Copy the below code in the created app.js file

var kafkaNode = require("kafka-node");

var options = {

kafkaHost: 'localhost:29092', // connect directly to kafka broker (instantiates a KafkaClient)

groupId: 'NodeConsumerGroup',

fromOffset: 'latest', // default

};

var consumerGroup = new kafkaNode.ConsumerGroup(options, 'queuing.course.messages');

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

console.log(message);

});

Run the code.

node app.js

When producing a message from another source (you can choose Kafkatool or the console producer), confirm it arrives in the Node application.

MyKey, MyValue from kafkatool will yield the following output:

{ topic: 'queuing.course.messages',

value: 'MyValue',

offset: 0,

partition: 0,

highWaterOffset: 1,

key: 'MyKey' }

### Kafka producer

Leave the consumer running. Create a new file app2.js and add the following code:

var kafkaNode = require("kafka-node")

var options = {

kafkaHost: 'localhost:29092' // connect directly to kafka broker (instantiates a KafkaClient)

};

HighLevelProducer = kafkaNode.HighLevelProducer;

client = new kafkaNode.KafkaClient(options);

producer = new HighLevelProducer(client);

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

//Creating the topic will fail if it is already there

producer.createTopics(['queuing.course.messages'], false, function (err, data) { console.log(JSON.stringify(err))});

});

payloads = [{

topic: 'queuing.course.messages',

key: 'key',

messages: 'value'

}];

producer.send(payloads, function (err, data) { console.log(JSON.stringify(err)) });

Run the producer and confirm the message is received by the consumer (additionally you can confirm it is visible in the console consumer and kafkatool)

Update the consumer in app.js with the following code.

var kafkaNode = require("kafka-node")

var options = {

kafkaHost: 'localhost:29092', // connect directly to kafka broker (instantiates a KafkaClient)

groupId: 'NodeConsumerGroup',

fromOffset: 'latest', // default

};

HighLevelProducer = kafkaNode.HighLevelProducer;

client = new kafkaNode.KafkaClient(options);

producer = new HighLevelProducer(client);

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

//Creating the topic will fail if it is already there

producer.createTopics(['queuing.course.messages.processed'], false, function (err, data) {});

});

var consumerGroup = new kafkaNode.ConsumerGroup(options, 'queuing.course.messages');

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

console.log('Received: '+JSON.stringify(message));

payloads = [{ topic: 'queuing.course.messages.processed',

key: message.key,

messages: message.value.replace('Bob','Jane') }];

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

console.log('Send: '+JSON.stringify(payloads)+' Result: '+JSON.stringify(data));

});

});

What does this do?

Run it and send in a message to topic ‘queuing.course.messages’ with message ‘Hi Bob’

The result can be something like:

Received: {"topic":"queuing.course.messages","value":"Hi Bob","offset":8,"partition":0,"highWaterOffset":9,"key":"Greeting"}

Send: [{"topic":"queuing.course.messages.processed","key":"Greeting","messages":"Hi Jane","partition":0,"attributes":0}] Result: {"queuing.course.messages.processed":{"0":2}}

What does the 2 in {"0":2} stand for?

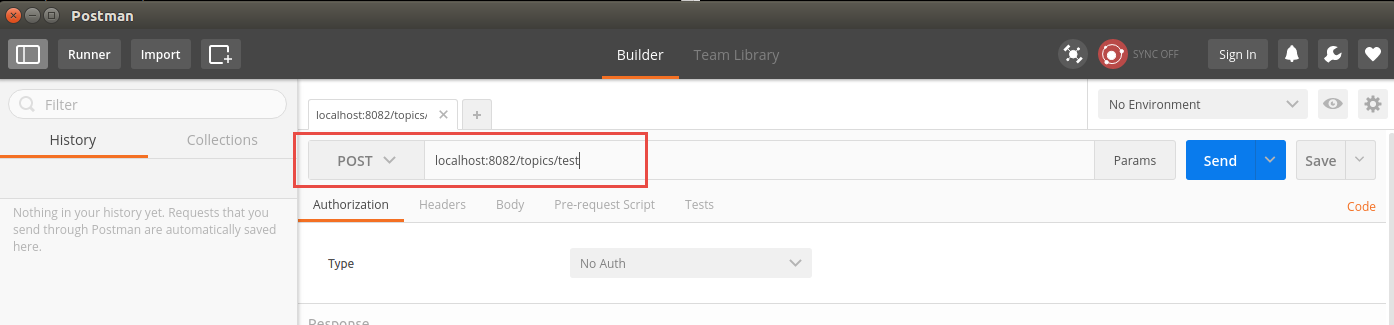
## Postman as Kafka client

### Produce a message

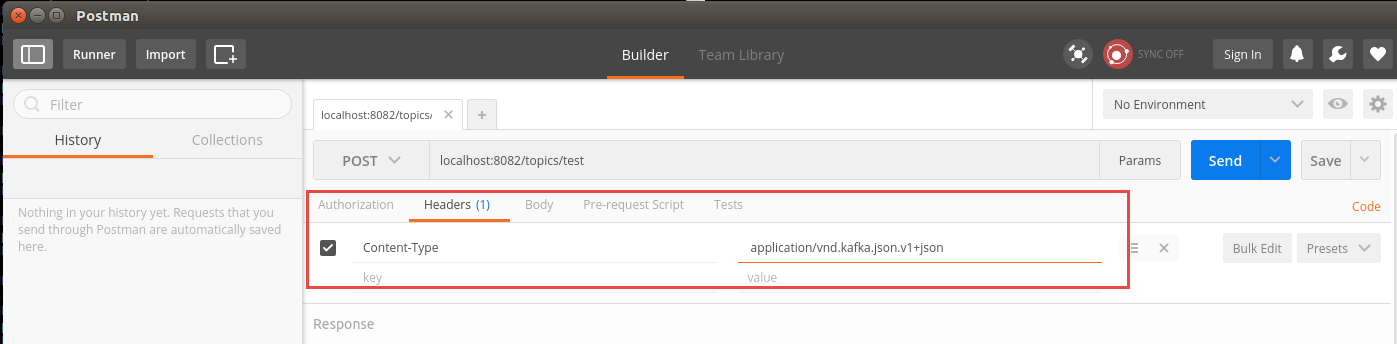
Start Postman.

postman

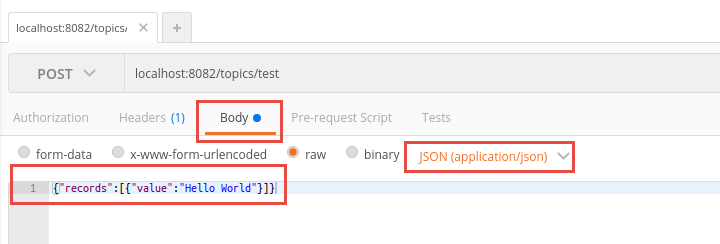
Select POST and input localhost:8082/topics/test as URL



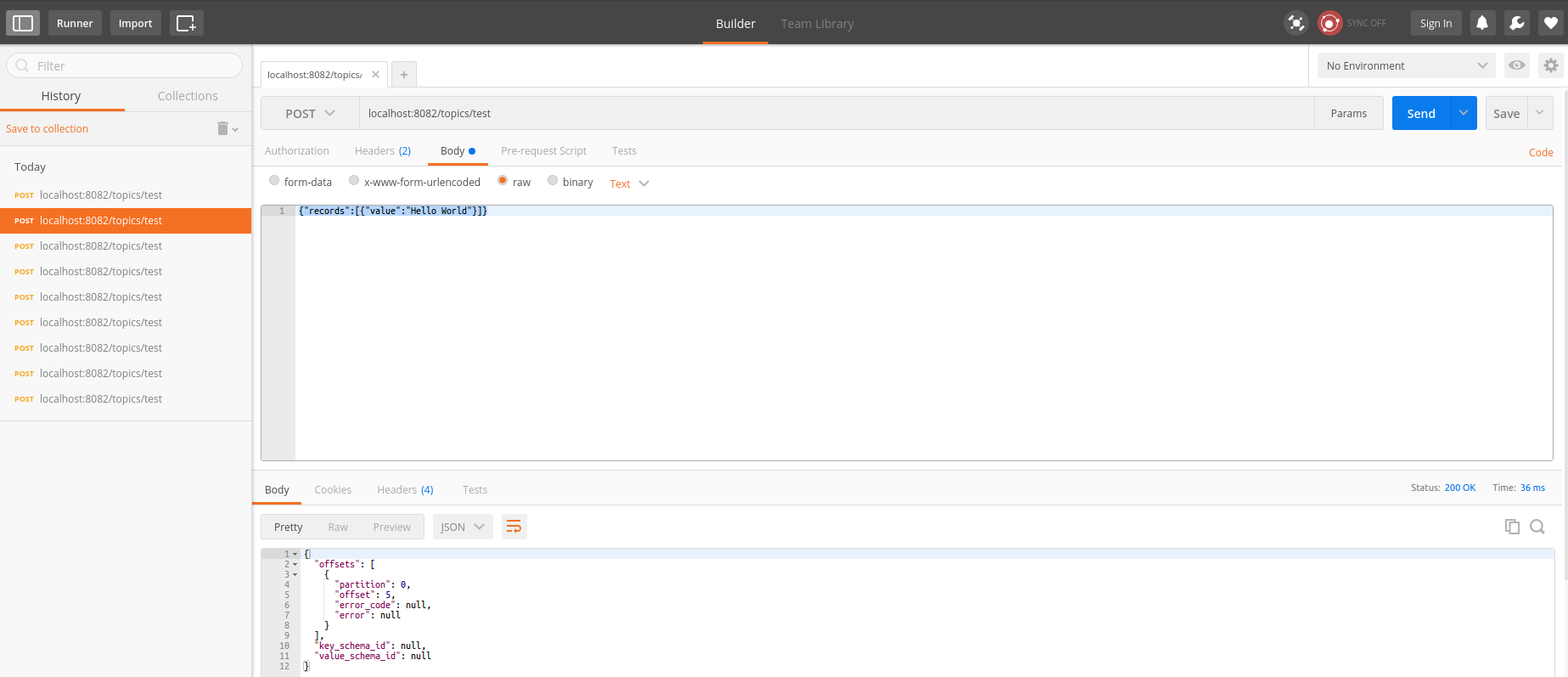
Add a Header: Content-Type application/vnd.kafka.json.v1+json



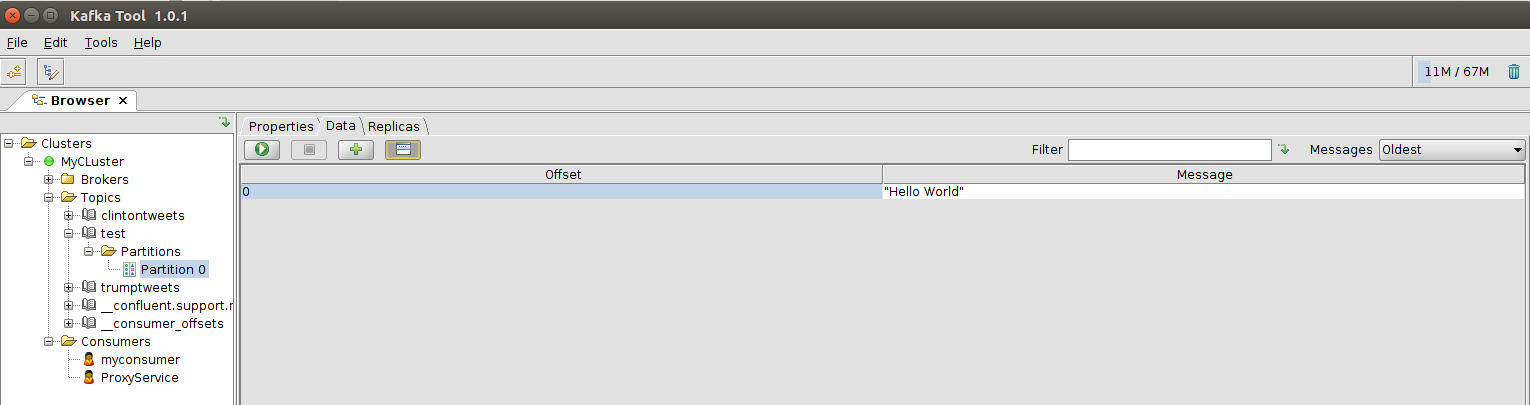
Use the body: {"records":[{"value":"Hello World"}]}



Click send



Start Kafkatool and confirm the message has arrived.



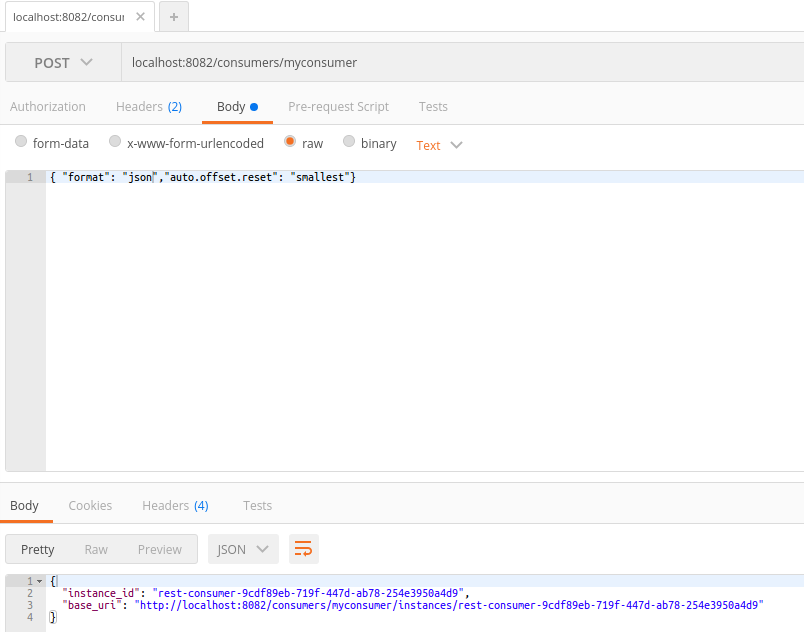
### Create a consumer

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"

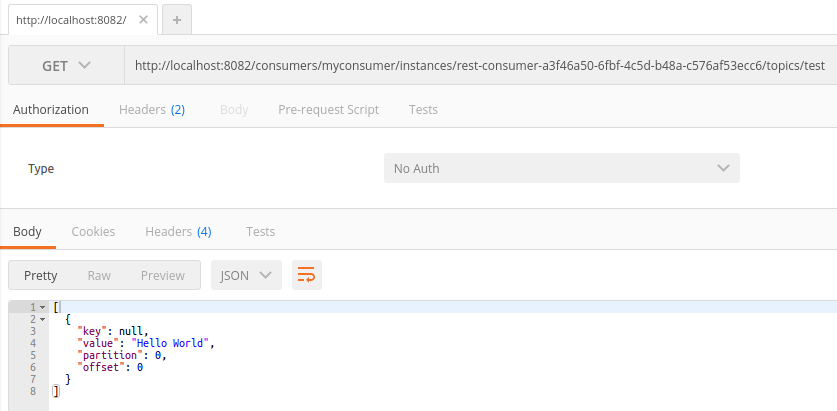
}

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

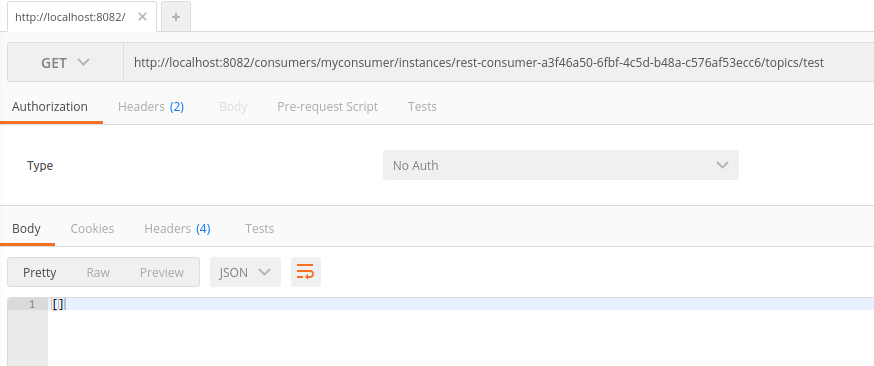
Specify Accept header application/vnd.kafka.json.v1+json

Confirm you can receive the response:



Repeat the request

Confirm you are not receiving new messages



# Schema registry

Go to the kafka-workshop folder and make sure Kafka is started

cd ~/kafka-workshop

docker ps

Enter the Kafka Schemaregistry Docker container.

docker exec -it kafkaworkshop\_schemaregistry\_1 /bin/bash

cd /usr/bin

Start an Avro console producer

kafka-avro-console-producer \

--broker-list kafka1:9092 --topic avrotest \

--property value.schema='{"type":"record","name":"myrecord","fields":[{"name":"f1","type":"string"}]}'

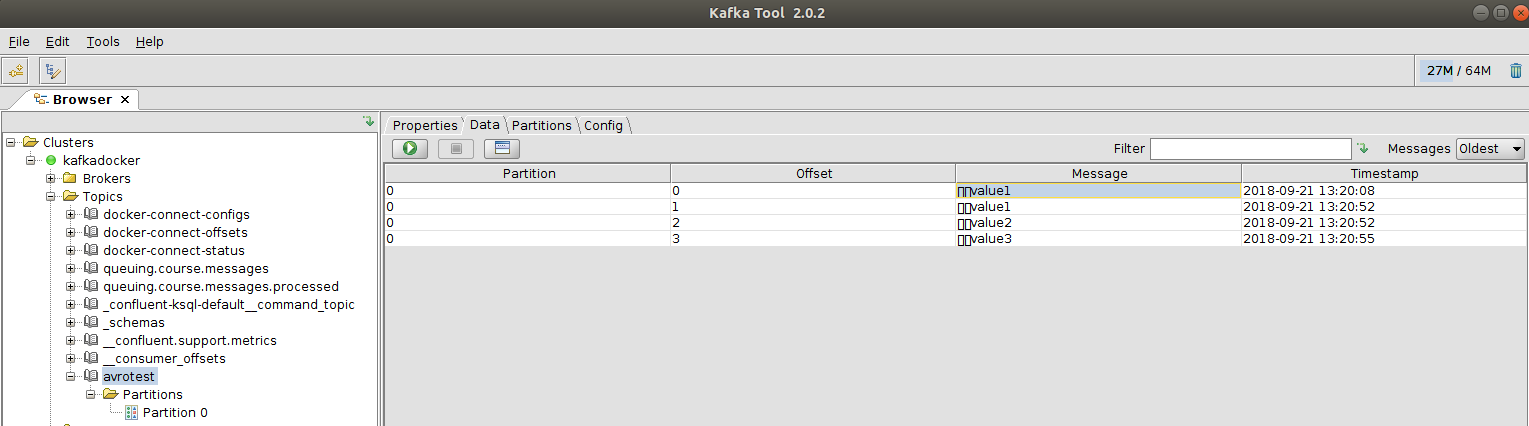
Enter the following messages

{"f1": "value1"}

{"f1": "value2"}

{"f1": "value3"}

Check how the messages look with Kafkatool



What are the strange characters in front of the string message?

Try to send a message which does not conform to the schema

{"f2": "value1"}

The message is rejected because it does not conform to the schema.

Consume the messages

kafka-avro-console-consumer --topic avrotest --zookeeper zookeeper:2181 --from-beginning

Produce a message with a new schema:

kafka-avro-console-producer \

--broker-list kafka1:9092 --topic avrotest \

--property value.schema='{"type":"record","name":"myrecord","fields":[{"name":"f1","type":"string"},{"name":"f2","type":"string"}]}'

Send a message like

{"f1":"value1","f2": "value1"}

This message is rejected because:

Caused by: io.confluent.kafka.schemaregistry.client.rest.exceptions.RestClientException: Schema being registered is incompatible with an earlier schema; error code: 409

Start a new producer but now which a slightly changed schema:

kafka-avro-console-producer \

--broker-list kafka1:9092 --topic avrotest \

--property value.schema='{"type":"record","name":"myrecord","fields":[{"name":"f1","type":"string"},{"name":"f2","type":"string", "default:"whatever"}]}'

Produce a message like:

{"f1":"value1","f2": "value1"}

Why does this not give any errors?

Run the consumer again and check the messages

kafka-avro-console-consumer --topic avrotest --zookeeper zookeeper:2181 --from-beginning

Browse the following URL’s. What do they show?

<http://localhost:8081/subjects>

<http://localhost:8081/subjects/avrotest-value/versions>

<http://localhost:8081/subjects/avrotest-value/versions/2>

<http://localhost:8081/subjects/avrotest-value/versions/1>

# Kafka Connect

(Source: below is based on exercise 2 from https://github.com/confluentinc/kafka-workshop)

Go to the kafka-workshop folder

cd ~/kafka-workshop

Check Kafka Connect has been started:

docker-compose logs -f connect|grep "Kafka Connect started"



Create a JDBC connector:

curl -i -X POST -H "Accept:application/json" \

-H "Content-Type:application/json" http://localhost:8083/connectors/ \

-d @connect/postgres-source.json

Check the status of the connector

curl -s "http://localhost:8083/connectors"| jq '.[]'| xargs -I{connector\_name} curl -s "http://localhost:8083/connectors/"{connector\_name}"/status"| jq -c -M '[.name,.connector.state,.tasks[].state]|join(":|:")'| column -s : -t| sed 's/\"//g'| sort

This should give as output: jdbc\_source\_postgres\_movies | RUNNING | RUNNING

List the Kafka topics

docker-compose exec kafka1 bash -c 'kafka-topics --zookeeper zookeeper:2181 --list'

It should list postgres-movies

List the topic contents:

docker-compose exec connect \

kafka-avro-console-consumer \

--bootstrap-server kafka1:9092 \

--property schema.registry.url=http://schemaregistry:8081 \

--topic postgres-movies --from-beginning

Leave the above running. In a new console Window enter the database container:

docker-compose exec database bash -c 'psql --username postgres --d WORKSHOP'

Insert a record

INSERT INTO movies(id,title,release\_year) VALUES (937,'Top Gun',1986);

Confirm it appears on the topic

# Kafka Streams / KSQL

<https://docs.confluent.io/current/ksql/docs/tutorials/basics-docker.html#ksql-quickstart-docker>