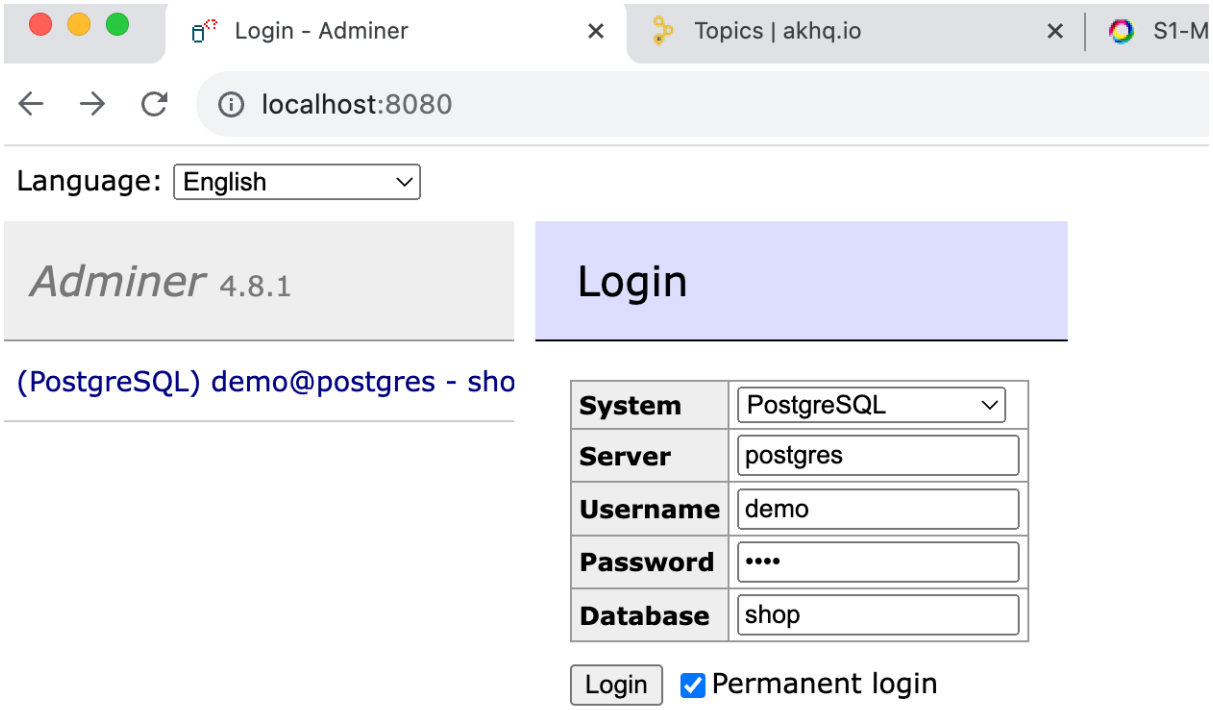
Laboratory 7

Lab1:

1. Login into admin application http://localhost:8080

2. Login as

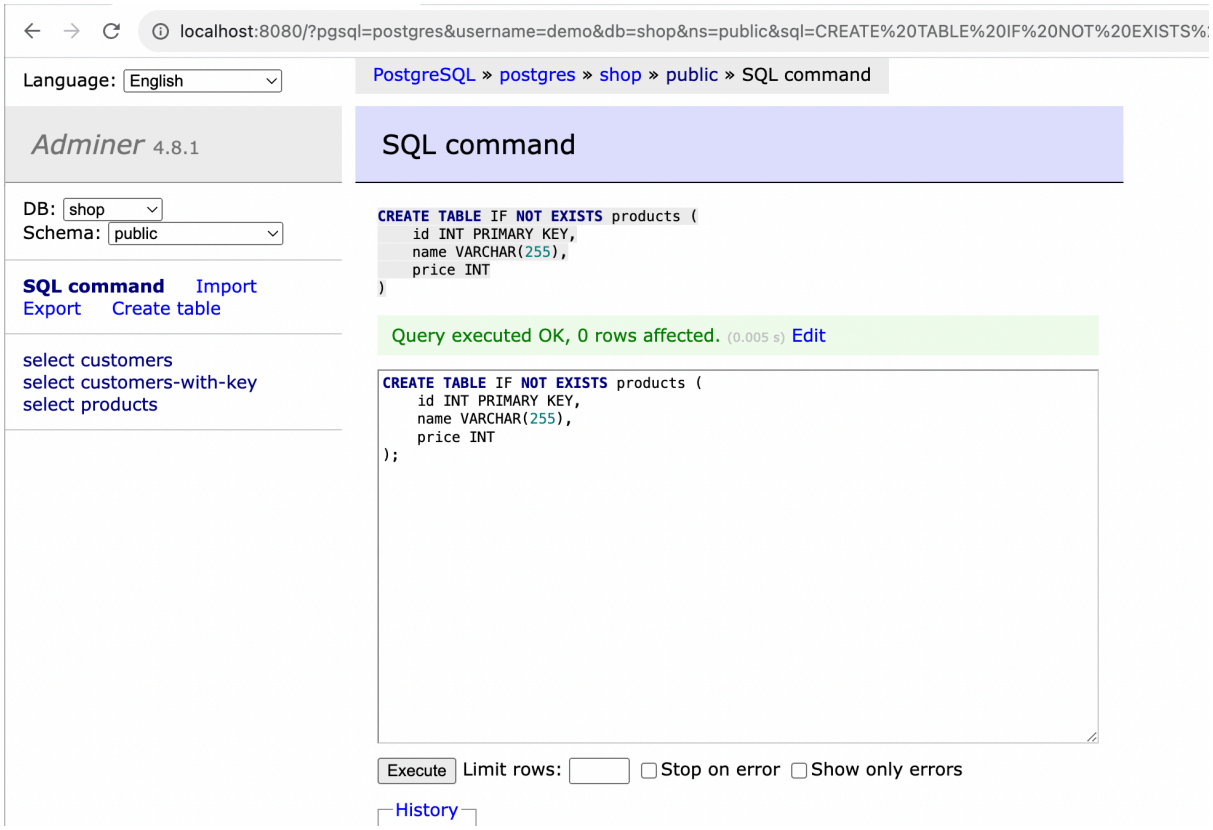
System: PostgreSQL
Server: postgres
Username: demo
Password: demo
Database: shop



2. Create a database (SQL command)

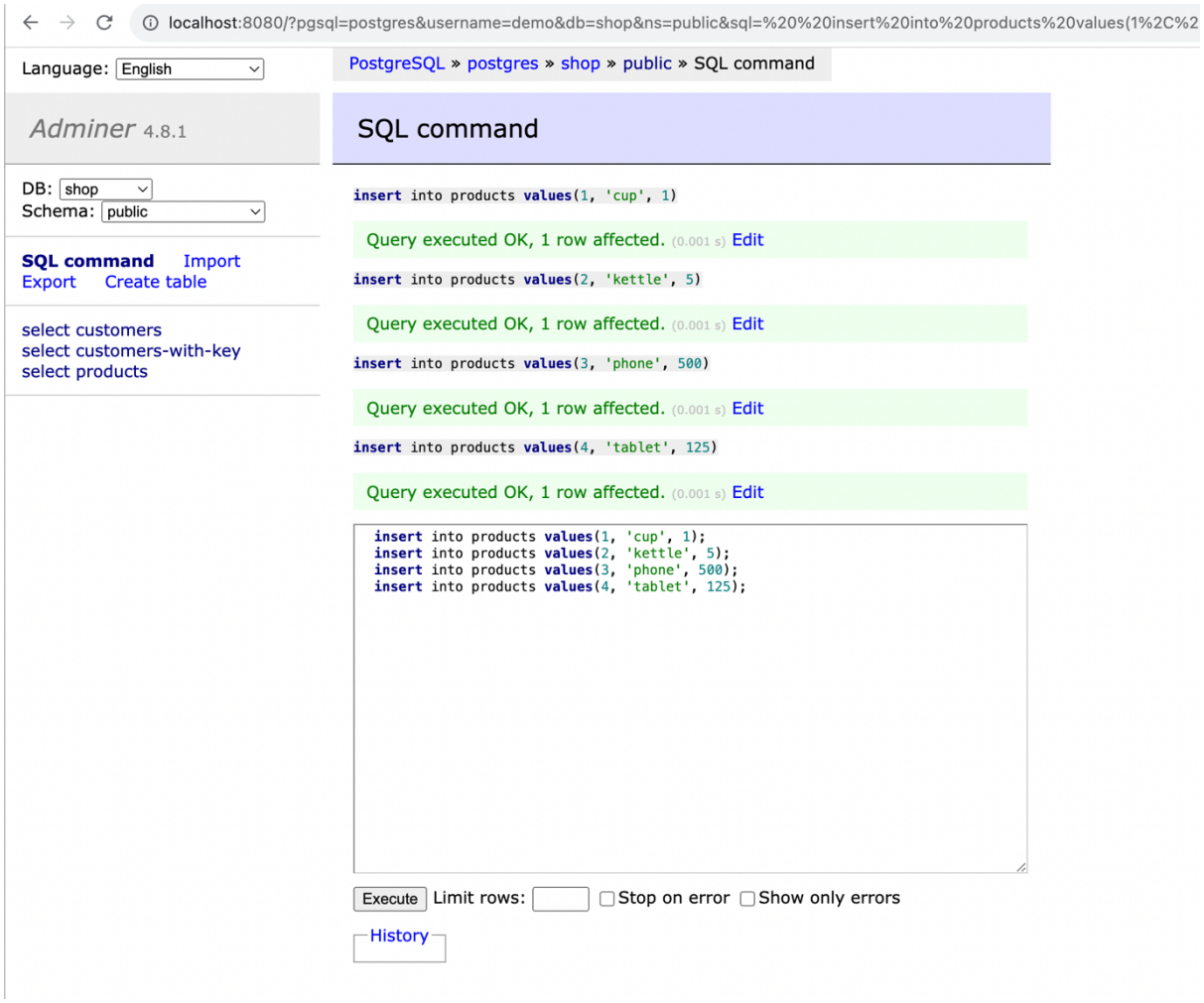
Press "SQL command" button from left side.

```
CREATE TABLE IF NOT EXISTS products (
   id INT PRIMARY KEY,
   name VARCHAR(255),
   price INT
);
```



3. Enter SQL Command

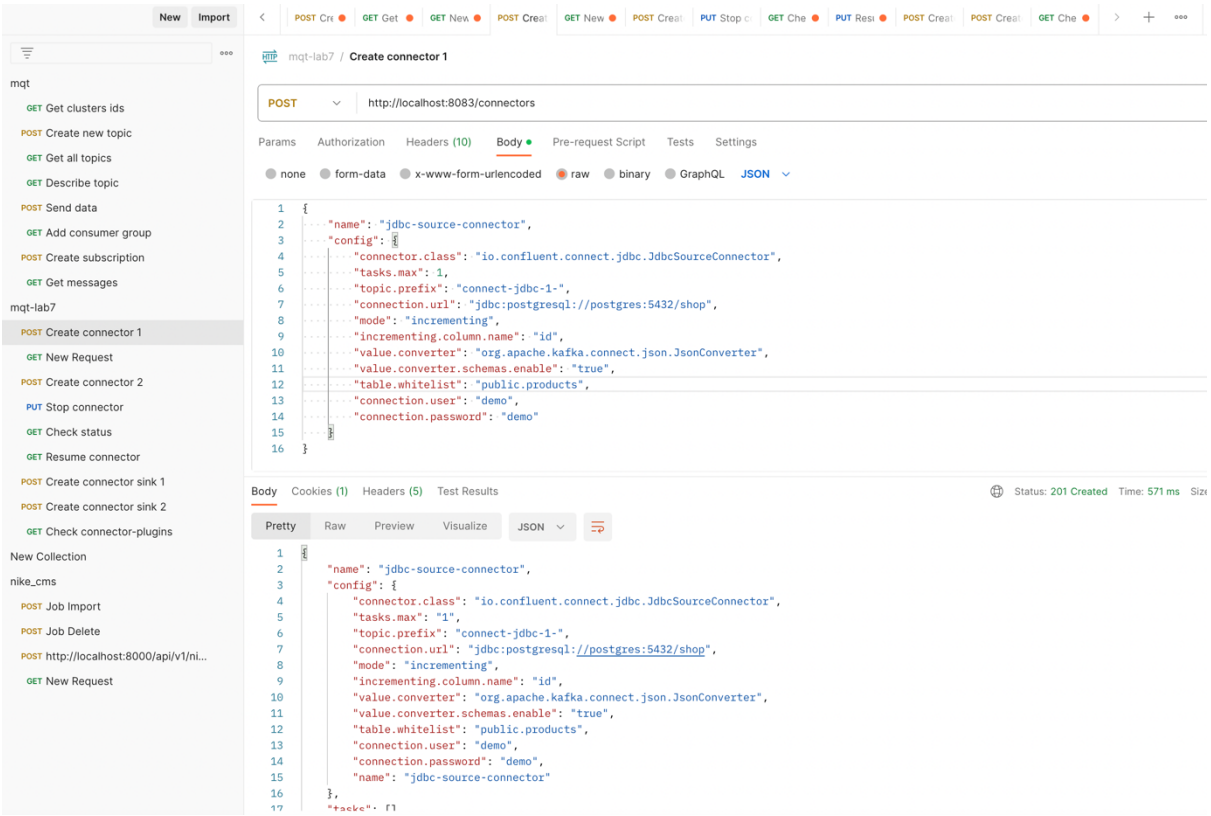
```
insert into products values(1, 'cup', 1);
insert into products values(2, 'kettle', 5);
insert into products values(3, 'phone', 500);
insert into products values(4, 'tablet', 125);
```



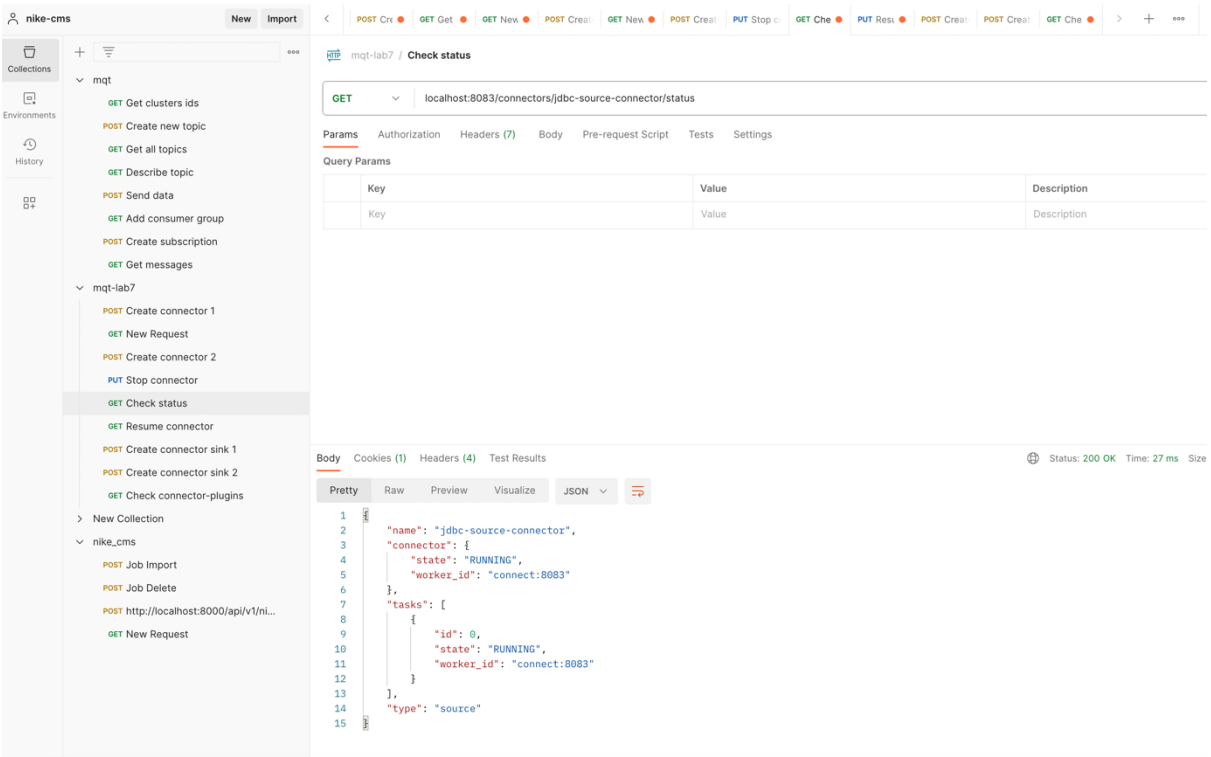
4. Check connector-plugins through control center or through

curl http://localhost:8083/connector-plugins from Windows PowerShell or MacOS shell or @GET http://localhost:8083/connector-plugins from Postman

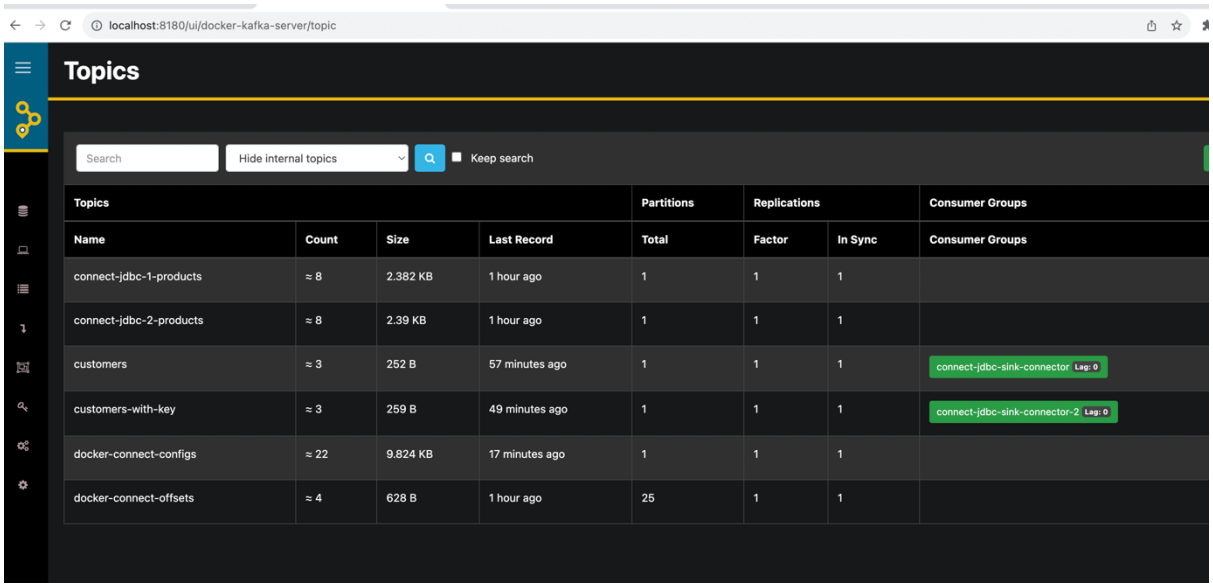
5. Add JDBC source connector



6. Check connectors through Control center or through REST



7. Check messages



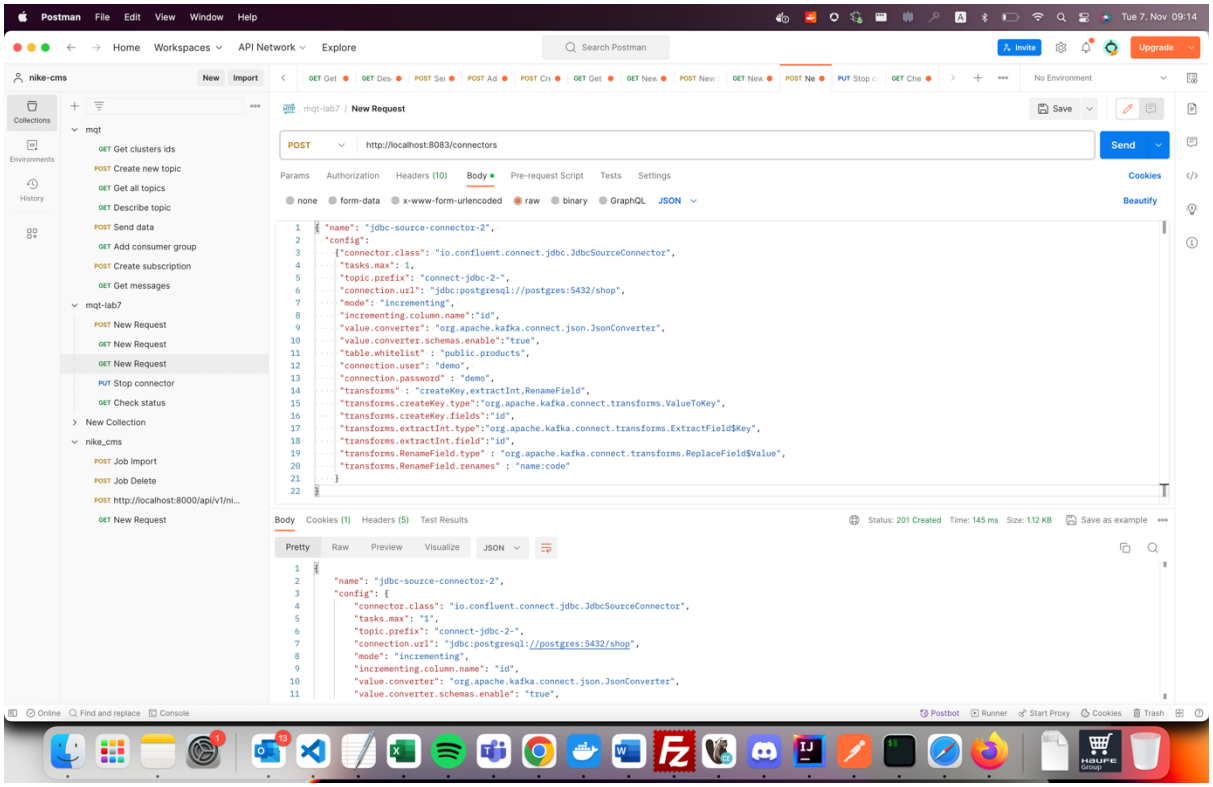
8. Add more data

insert into products values(5, 'toaster', 15);
insert into products values(6, 'coffee maker', 25);

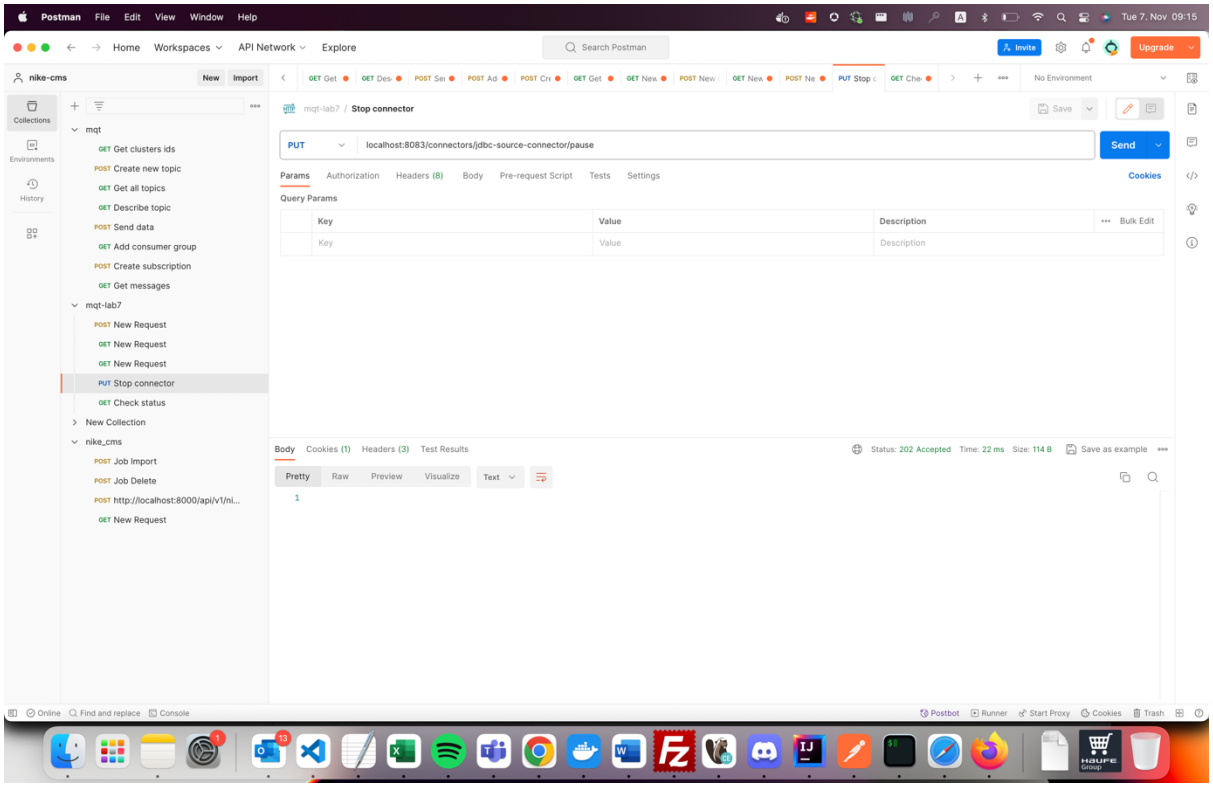
9. Check messages

Lab2:

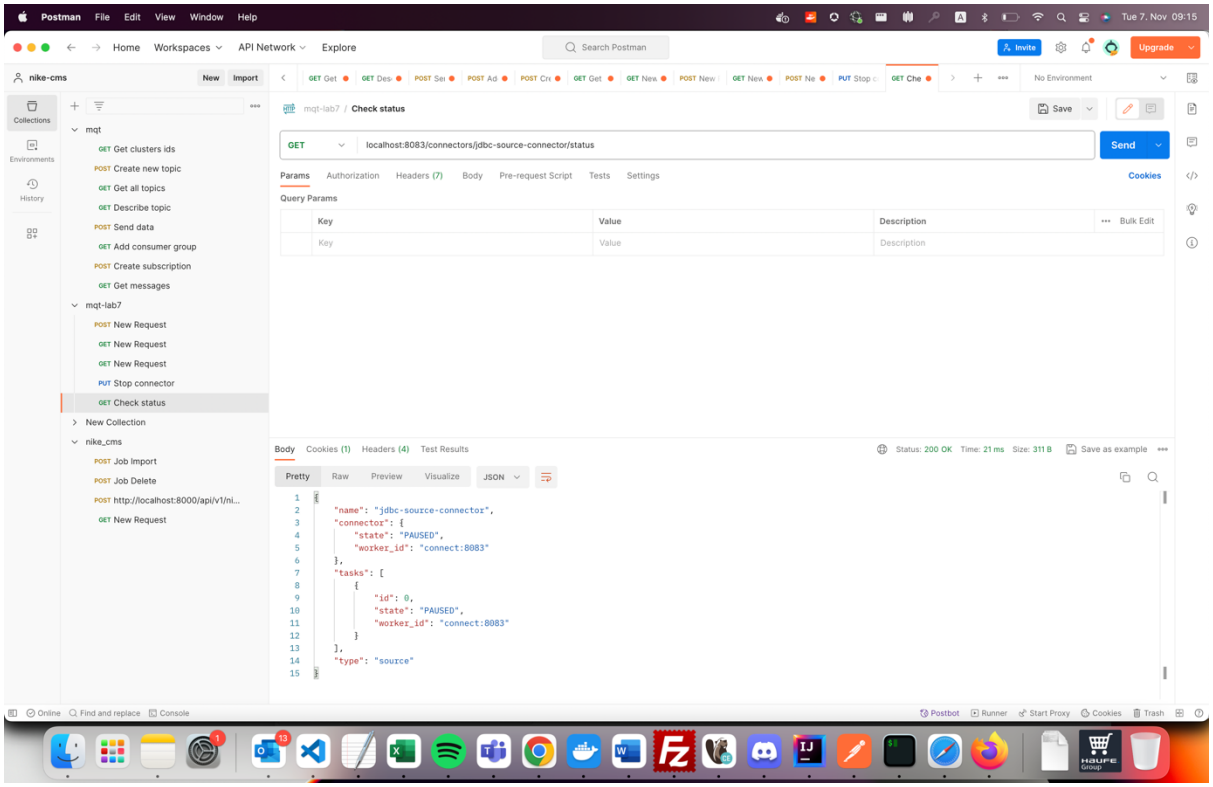
1. Add JDBC source connector



2. Stop connector



3. Check status



4. Add new data (http://localhost:8080 login as demo|demo, use SQL Command)

insert into products values(7, 'saucepan', 11);
insert into products values(8, 'iron', 7);

```
insert into products values(7, 'saucepan', 11)
```

Query executed OK, 1 row affected. (0.001 s) Edit, Warnings

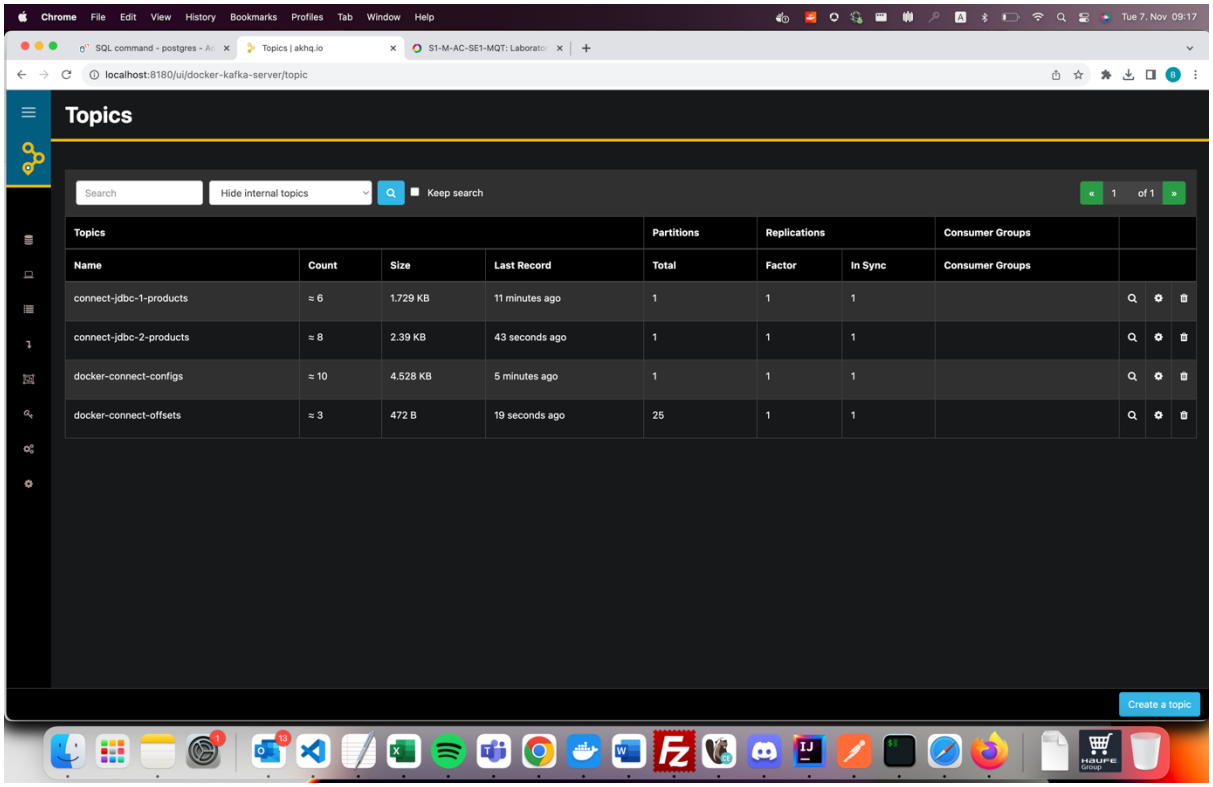
```
insert into products values(8, 'iron', 7)
```

Query executed OK, 1 row affected. (0.001 s) Edit, Warnings

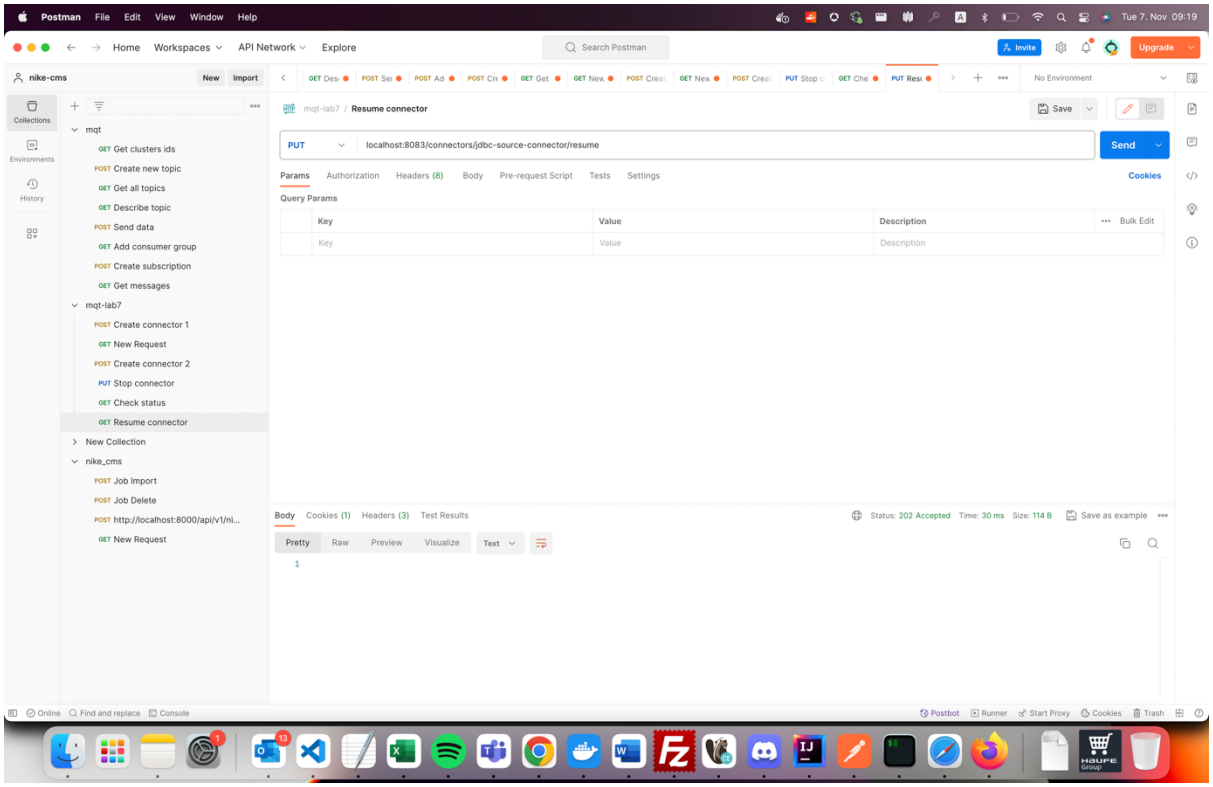
Error in query: 2 3 4 5 6 7

```
CREATE TABLE IF NOT EXISTS products (
    id INT PRIMARY KEY,
    name VARCHAR(255),
    price INT
);

  insert into products values(1, 'cup', 1);
  insert into products values(2, 'kettle', 5);
  insert into products values(3, 'phone', 500);
  insert into products values(4, 'tablet', 125);

  insert into products values(5, 'toaster', 15);
  insert into products values(6, 'coffee maker', 25);

  insert into products values(7, 'saucepan', 11);
  insert into products values(8, 'iron', 7);
```

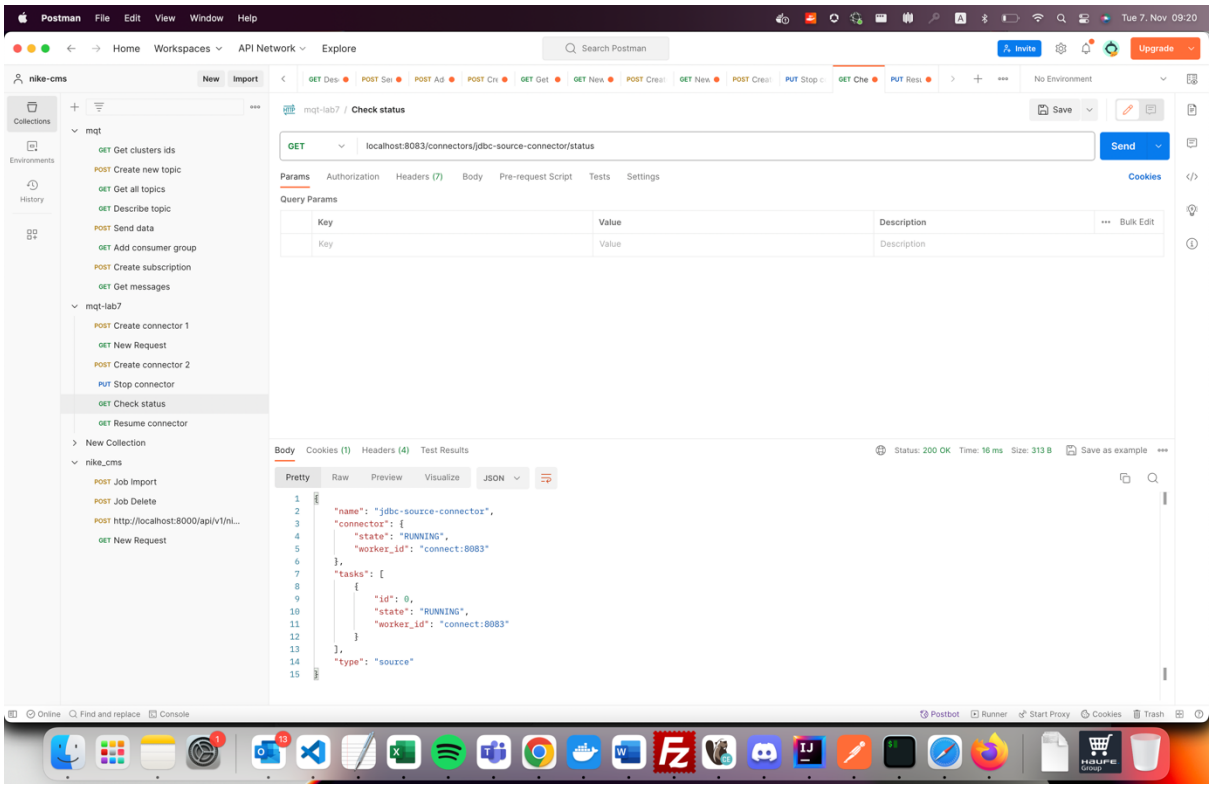
5. Check messages



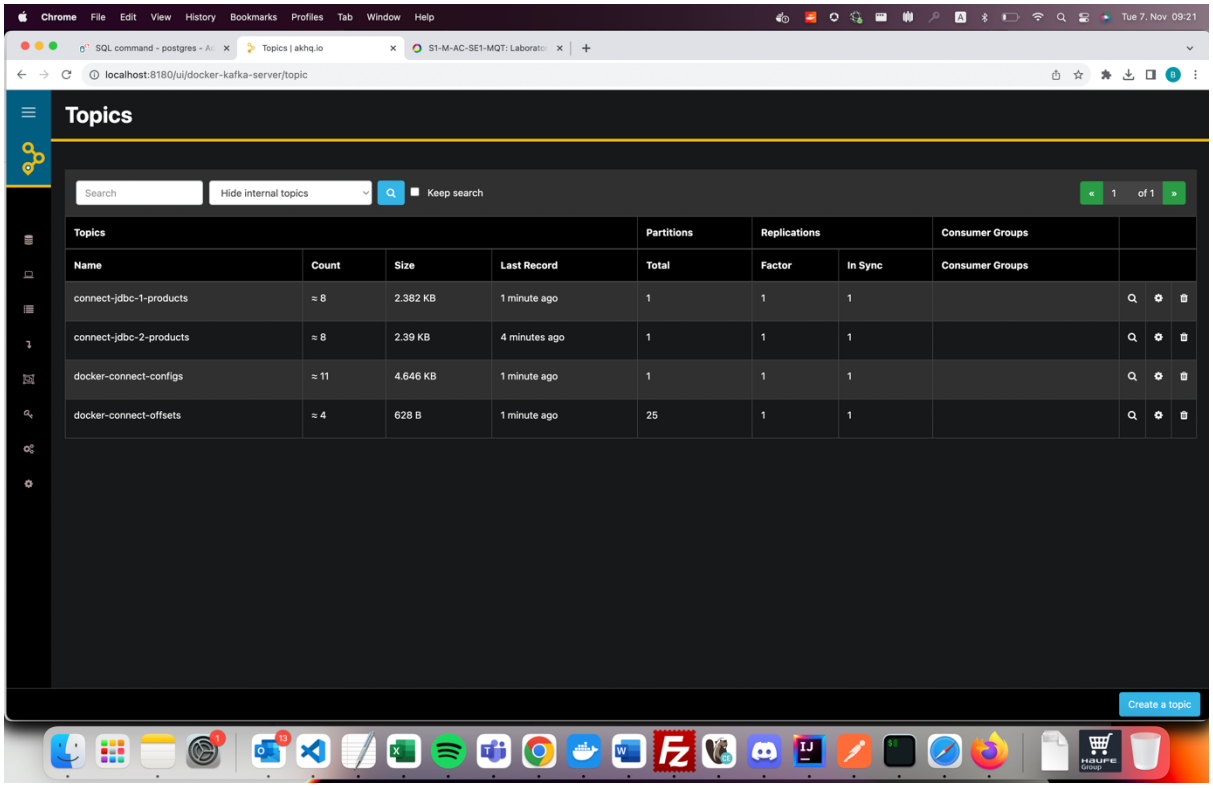
6. Resume connector



7. Check status

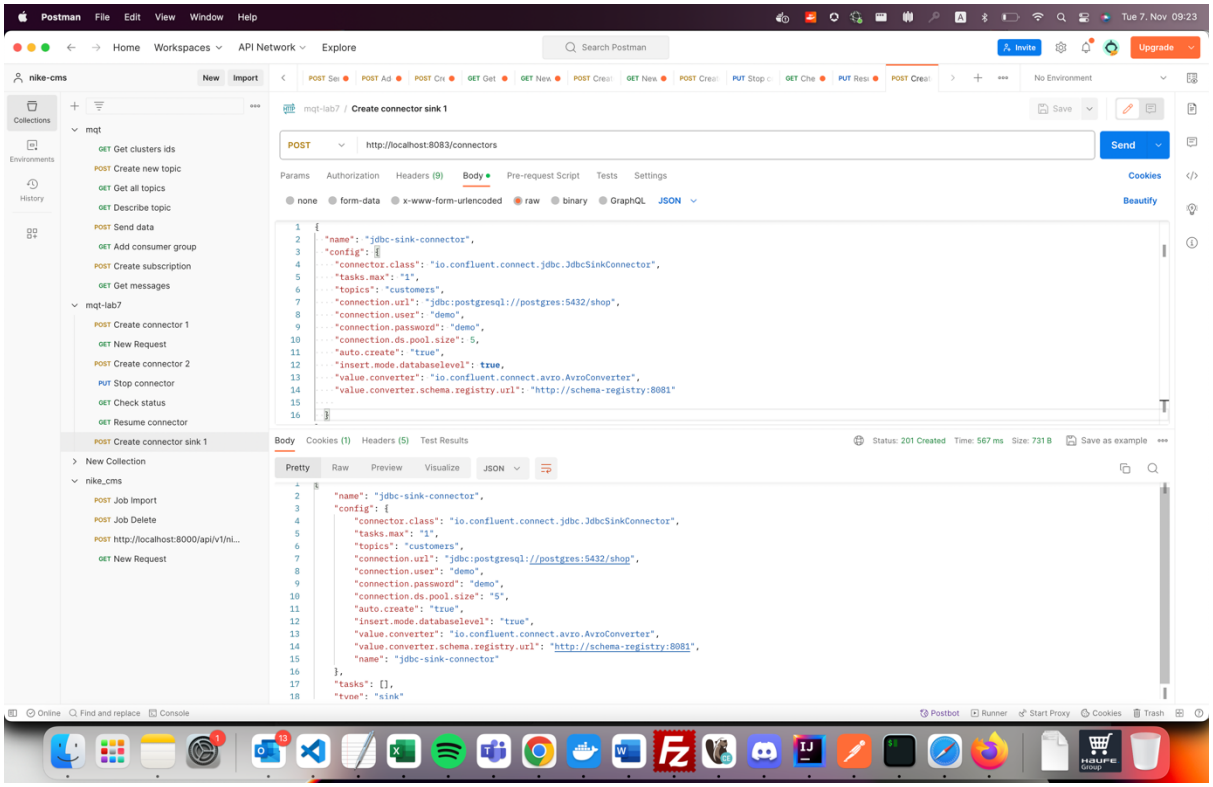


8. Check messages



Lab3:

1. In Control Center create JDBC sink connector. The configuration is in file postgres-sink-connector-1.json



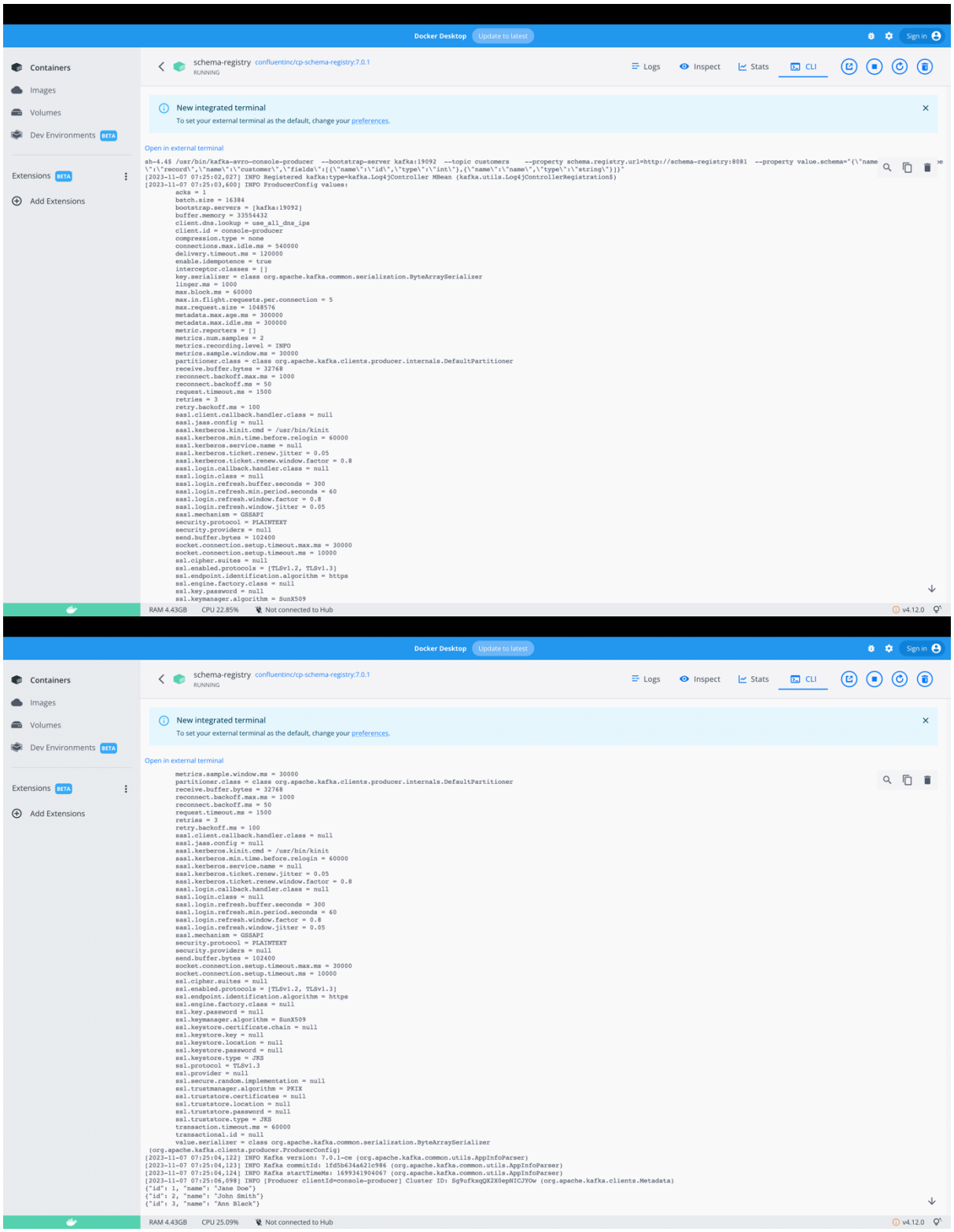
2. In schema registry container produce avro messages

docker exec -ti schema-registry /usr/bin/kafka-avro-console-producer --bootstrap-server kafka:19092 --topic customers --property schema.registry.url=http://schema-registry:8081 --property value.schema="{\"name\":\"lau\",\"type\":\"record\",\"name\":\"customer\",\"fields\":[{\"name\":\"id\",\"type\":\"int\"},{\"name\":\"name\",\"type\":\"string\"}]}"

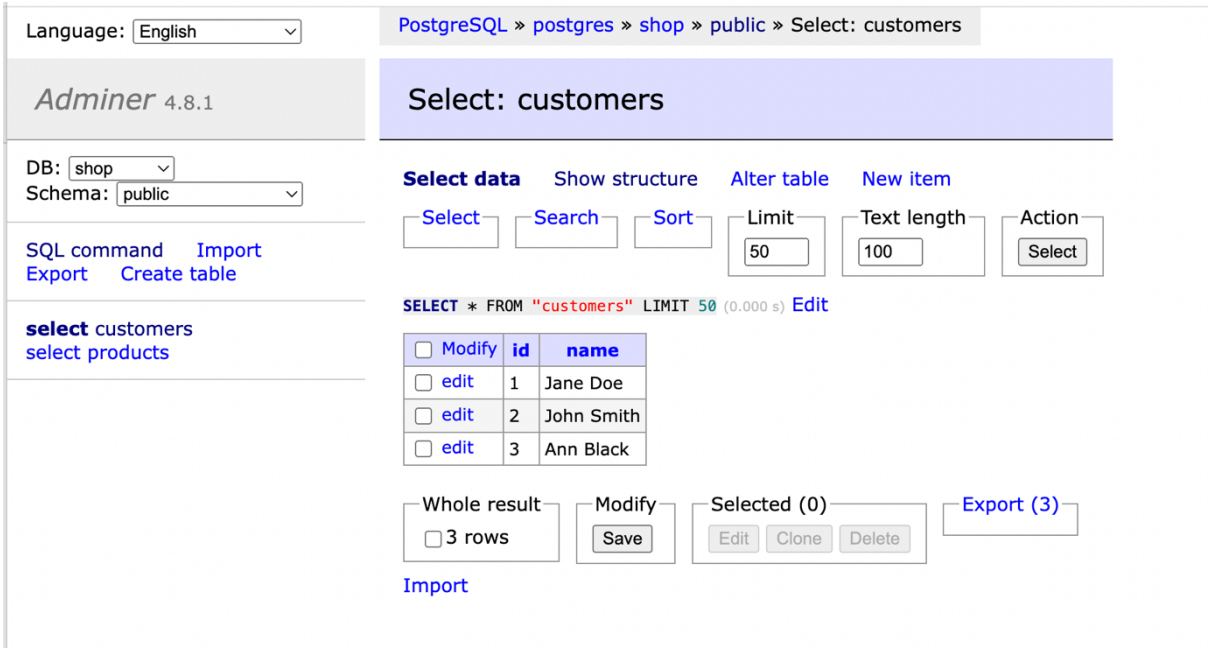
the command can also be executed from Docker Desktop, schema-registry terminal:
/usr/bin/kafka-avro-console-producer --bootstrap-server kafka:19092 --topic customers2 --property schema.registry.url=http://schema-registry:8081 --property value.schema="{\"name\":\"lau\",\"type\":\"record\",\"name\":\"customer\",\"fields\":[{\"name\":\"id\",\"type\":\"int\"},{\"name\":\"name\",\"type\":\"string\"}]}"

add in the producer console next lines:

{"id": 1, "name": "Jane Doe"}
{"id": 2, "name": "John Smith"}
{"id": 3, "name": "Ann Black"}

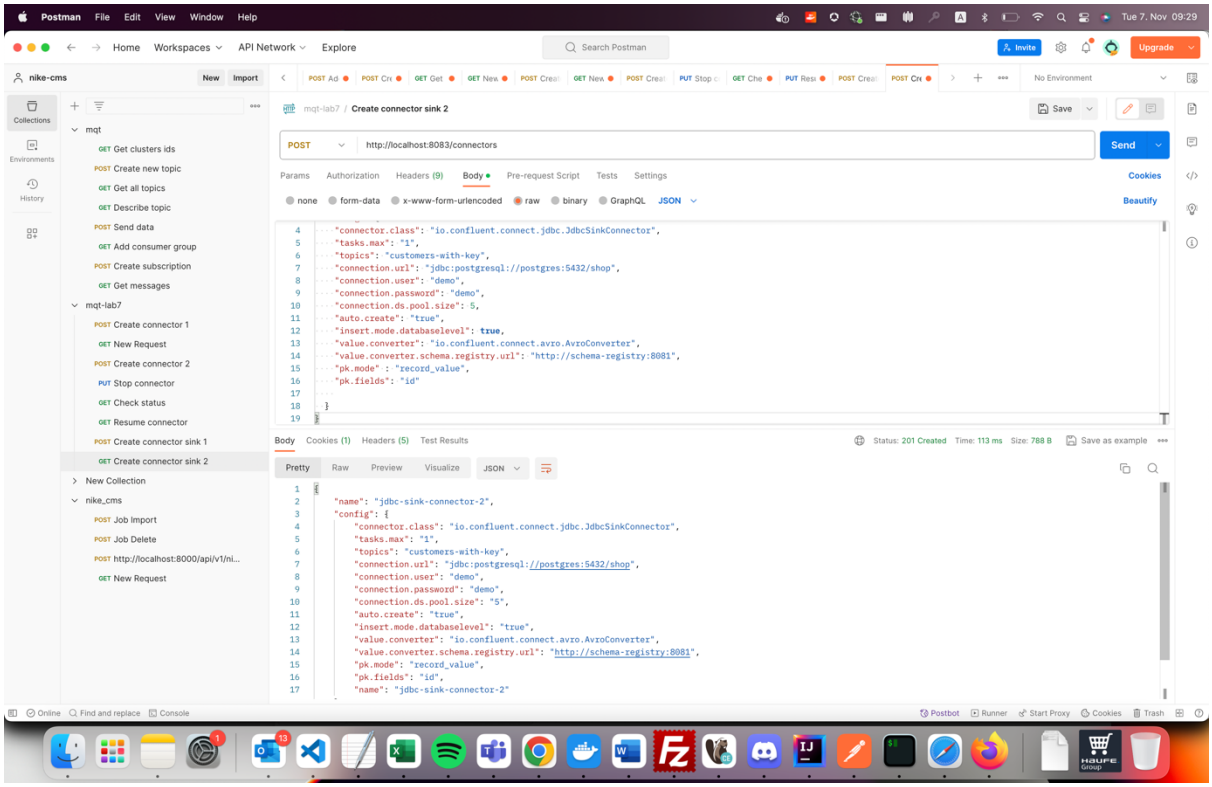


3. Check the table in http://localhost:8080



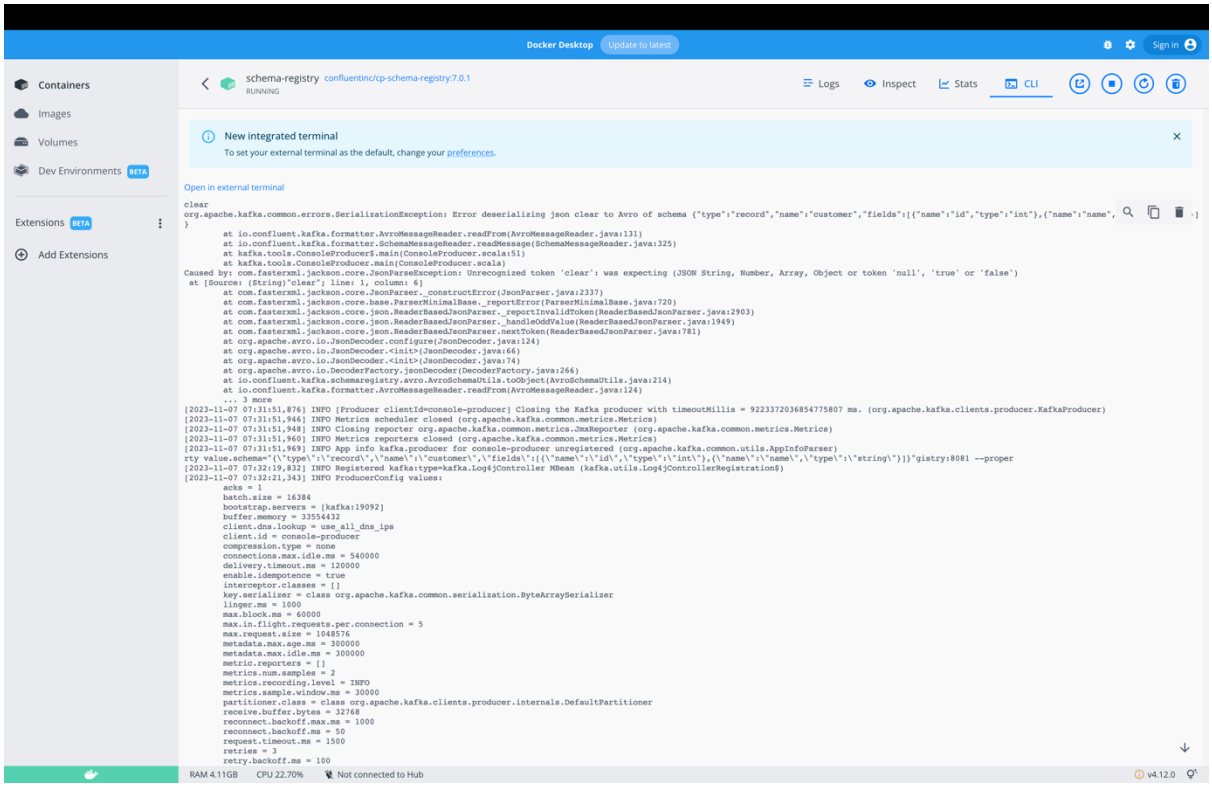
Lab4:

1. In Control Center create JDBC sink connector. The configuration is in file postgres-sink-connector-2.json

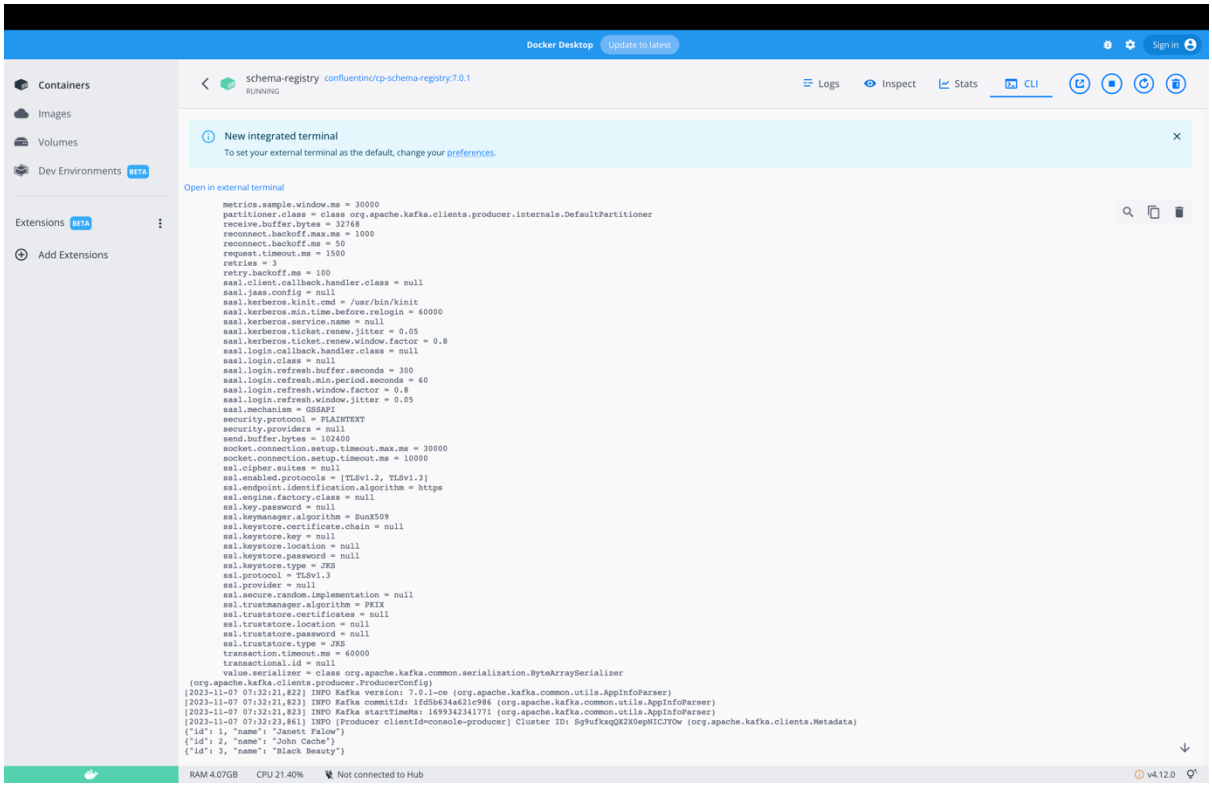


2. In schema registry container produce avro messages

MERGE:
docker exec -ti schema-registry /usr/bin/kafka-avro-console-producer --bootstrap-server kafka:19092 --topic customers-with-key --property schema.registry.url=http://schema-registry:8081 --property value.schema="{\"type\":\"record\",\"name\":\"customer\",\"fields\":[{\"name\":\"id\",\"type\":\"int\"},{\"name\":\"name\",\"type\":\"string\"}]}"



{"id": 1, "name": "Janett Falow"}
{"id": 2, "name": "John Cache"}
{"id": 3, "name": "Black Beauty"}



3. Check the table in http://localhost:8080