# [Docker microservice for ecommerce](https://github.com/kamlesh1984/micro-ecommerce)

# <https://github.com/kamlesh1984/micro-ecommerce.git>

# This project is intended to demonstrate end-to-end best practices for building a cloud native, microservice architecture using Spring Boot&Cloud.

## So first lets create a VM on Google cloud and do the git clone of repository

First install the Docker jdk maven and git on new VM

sudo yum install -y yum-utils device-mapper-persistent-data lvm2

sudo yum-config-manager --add-repo <https://download.docker.com/linux/centos/docker-ce.repo>

sudo yum install docker-ce -y

sudo usermod -aG docker $(whoami)

sudo systemctl enable docker.service

sudo systemctl start docker.service

sudo yum install epel-release -y

sudo yum install -y python-pip -y

sudo pip install docker-compose

sudo yum upgrade python\* -y

docker-compose version

sudo yum install java-1.8.0-openjdk -y

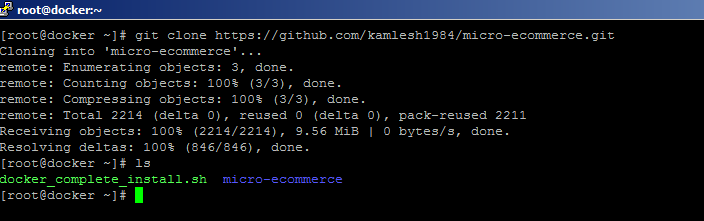
sudo yum install maven -y

mvn -version

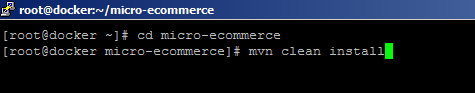
java -version

Now clone the java repository

<https://github.com/kamlesh1984/micro-ecommerce.git>

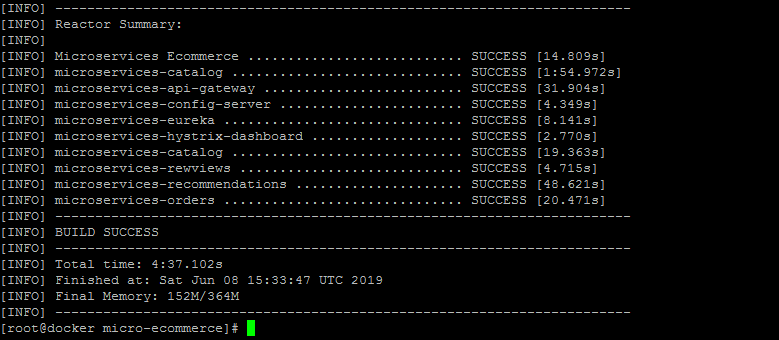


Now go to the directory and create the jar file

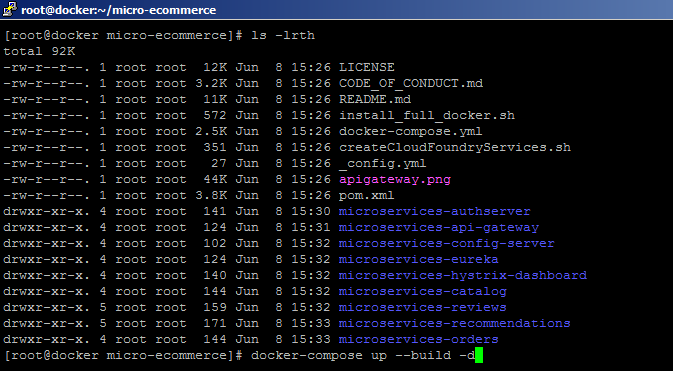


It will download all the maven dependency

So finally we cann see build is success and its created below jar file

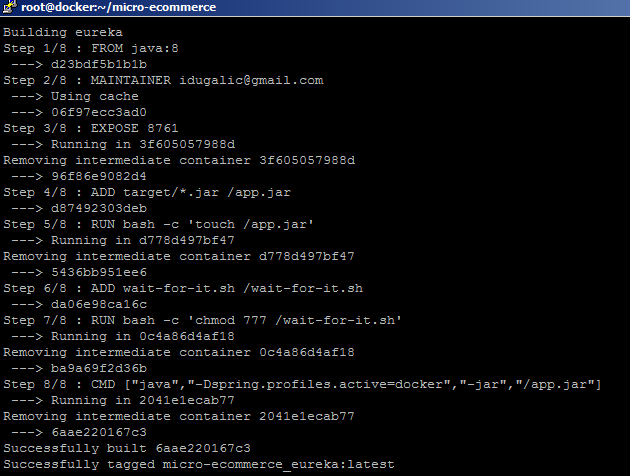


Now the jar file has been generated and placed in target folder ,so wee will run docker compose to run all docker container of microservice

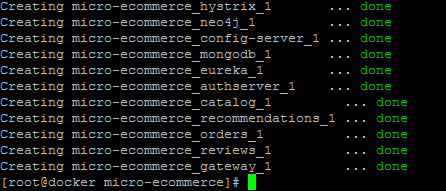


So it will read the compose file and create a docker container one by one

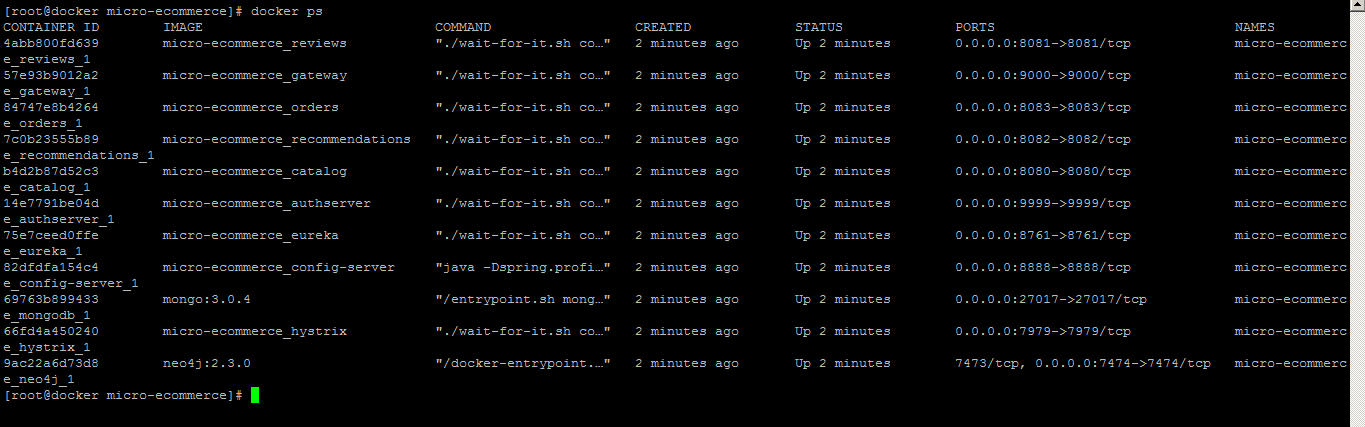
So we can see here it is createing a docker container of eureka microservice from deocker file using docker compose



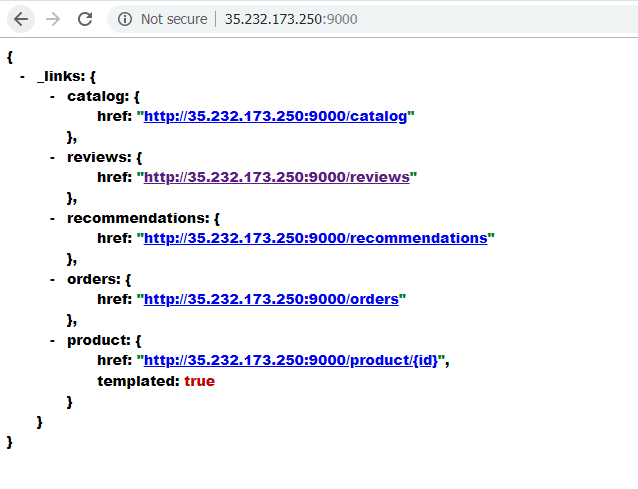
So we can see that docker compose created all the container



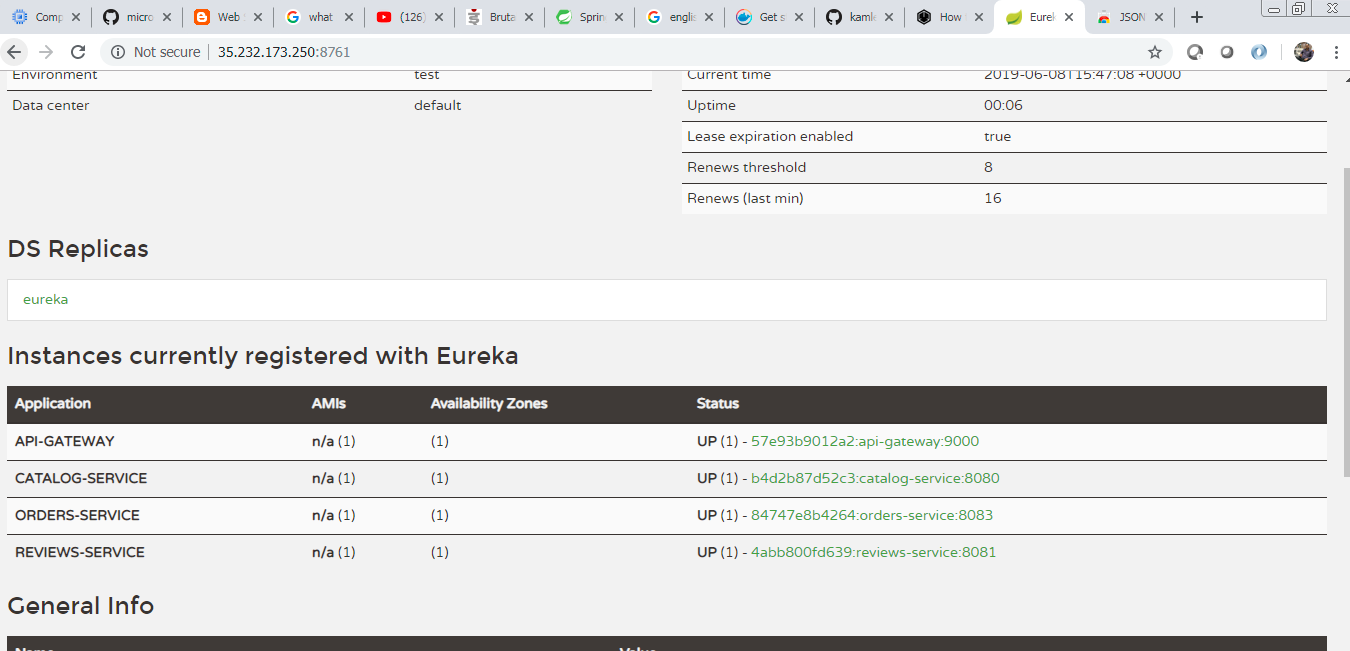
And if we do the docker ps command then we can see all the running caintainer

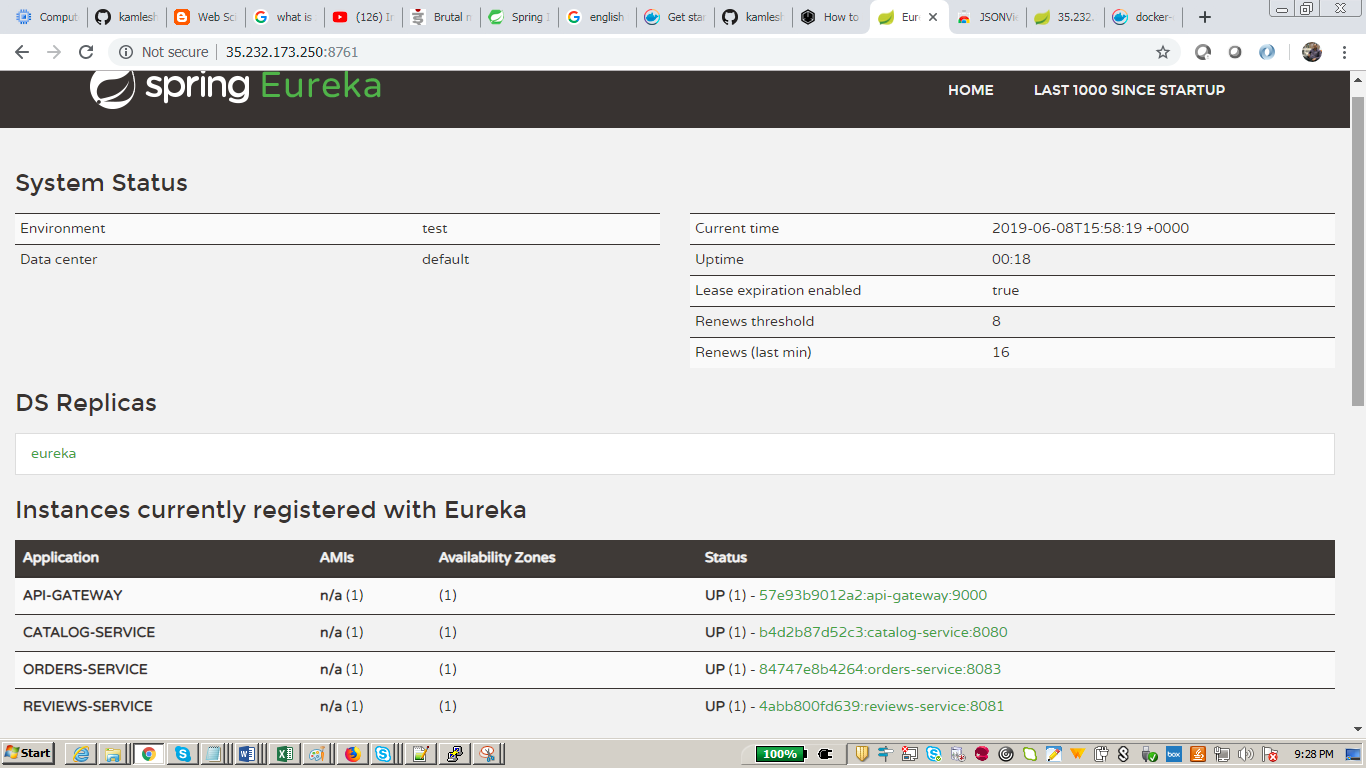


So if we want to see APi server the it will show all the endpoint

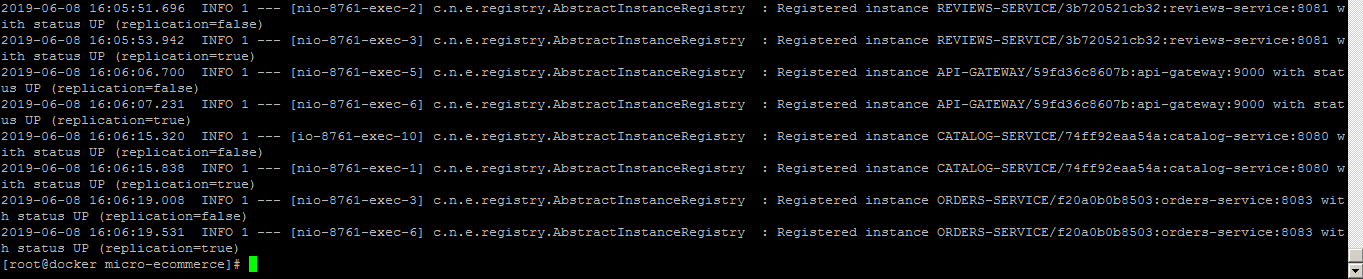


If i ill login on Eureka discovery then it will show all the server which is register in discoversy

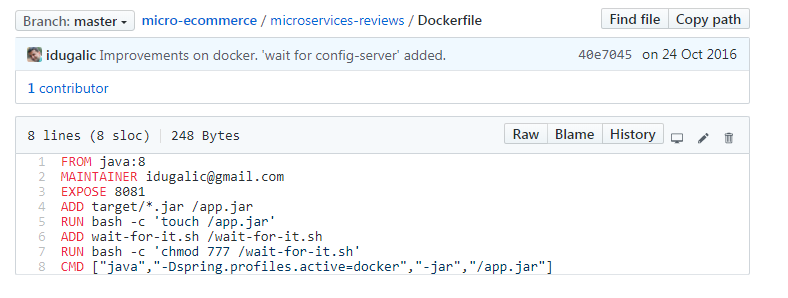




Now we can also see the log of discovery



Sample docker file for review service



## Table of Contents

* [Application 'Micro Ecommerce'](https://github.com/kamlesh1984/micro-ecommerce#application-micro-ecommerce)
  + [What is cloud native](https://github.com/kamlesh1984/micro-ecommerce#what-is-cloud-native)
  + [Architecture](https://github.com/kamlesh1984/micro-ecommerce#architecture)
  + [Backing services](https://github.com/kamlesh1984/micro-ecommerce#backing-services)
    - [API Gateway](https://github.com/kamlesh1984/micro-ecommerce#api-gateway)
    - [Config server](https://github.com/kamlesh1984/micro-ecommerce#config-server)
    - [Service registry (Eureka)](https://github.com/kamlesh1984/micro-ecommerce#service-registry-eureka)
    - [Authorization (Oauth2) server](https://github.com/kamlesh1984/micro-ecommerce#authorization-oauth2-server)
  + [Backend Microservices](https://github.com/kamlesh1984/micro-ecommerce#backend-microservices)
    - [Catalog](https://github.com/kamlesh1984/micro-ecommerce#catalog)
    - [Reviews](https://github.com/kamlesh1984/micro-ecommerce#reviews)
    - [Recommendations](https://github.com/kamlesh1984/micro-ecommerce#recommendations)
    - [Orders](https://github.com/kamlesh1984/micro-ecommerce#orders)
  + [Running Instructions](https://github.com/kamlesh1984/micro-ecommerce#running-instructions)
    - [Via maven (spring boot)](https://github.com/kamlesh1984/micro-ecommerce#via-maven-spring-boot)
    - [Via docker](https://github.com/kamlesh1984/micro-ecommerce#via-docker)
      * [Usage](https://github.com/kamlesh1984/micro-ecommerce#usage)
        + [Get a token:](https://github.com/kamlesh1984/micro-ecommerce#get-a-token)
        + [Catalog service:](https://github.com/kamlesh1984/micro-ecommerce#catalog-service)
        + [Reviews service:](https://github.com/kamlesh1984/micro-ecommerce#reviews-service)
        + [Recommendations service:](https://github.com/kamlesh1984/micro-ecommerce#recommendations-service)
        + [Orders service:](https://github.com/kamlesh1984/micro-ecommerce#orders-service)
        + [Catalog service(proxy) :](https://github.com/kamlesh1984/micro-ecommerce#catalog-serviceproxy-)
        + [Reviews service(proxy):](https://github.com/kamlesh1984/micro-ecommerce#reviews-serviceproxy)
        + [Recommendations service(proxy):](https://github.com/kamlesh1984/micro-ecommerce#recommendations-serviceproxy)
        + [Orders service(proxy):](https://github.com/kamlesh1984/micro-ecommerce#orders-serviceproxy)
        + [Mobile service (aggregate):](https://github.com/kamlesh1984/micro-ecommerce#mobile-service-aggregate)
  + [References and further reading](https://github.com/kamlesh1984/micro-ecommerce#references-and-further-reading)

# 

## What is cloud native

To understand “cloud native,” we must first understand “cloud.” In the context of this application, cloud refers to Platform as a Service. PaaS providers expose a platform that hides infrastructure details from the application developer, where that platform resides on top of Infrastructure as a Service (IaaS).

A cloud-native application is an application that has been designed and implemented to run on a Platform-as-a-Service installation and to embrace horizontal elastic scaling.

## Architecture

# The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP

# 

## Backing services

# The premise is that there are third-party service dependencies that should be treated as attached resources to your cloud native applications. The key trait of backing services are that they are provided as bindings to an application in its deployment environment by a cloud platform. Each of the backing services must be located using a statically defined route

### API Gateway

Implementation of an API gateway that is the single entry point for all clients. The API gateway handles requests in one of two ways. Some requests are simply proxied/routed to the appropriate service. It handles other requests by fanning out to multiple services.

### Config server

he configuration service is a vital component of any microservices architecture. Based on the twelve-factor app methodology, configurations for your microservice applications should be stored in the environment and not in the project.

The configuration service is essential because it handles the configurations for all of the services through a simple point-to-point service call to retrieve those configurations. The advantages of this are multi-purpose.

Let's assume that we have multiple deployment environments. If we have a staging environment and a production environment, configurations for those environments will be different. A configuration service might have a dedicated Git repository for the configurations of that environment. None of the other environments will be able to access this configuration, it is available only to the configuration service running in that environment.

When the configuration service starts up, it will reference the path to those configuration files and begin to serve them up to the microservices that request those configurations. Each microservice can have their configuration file configured to the specifics of the environment that it is running in. In doing this, the configuration is both externalized and centralized in one place that can be version-controlled and revised without having to restart a service to change a configuration.

With management endpoints available from Spring Cloud, you can make a configuration change in the environment and signal a refresh to the discovery service that will force all consumers to fetch the new configurations.

### Service registry (Eureka)

Netflix Eureka is a service registry. It provides a REST API for service instance registration management and for querying available instances. Netflix Ribbon is an IPC client that works with Eureka to load balance requests across the available service instances.

When using client-side discovery, the client is responsible for determining the network locations of available service instances and load balancing requests across them. The client queries a service registry, which is a database of available service instances. The client then uses a load balancing algorithm to select one of the available service instances and makes a request.

The client-side discovery pattern has a variety of benefits and drawbacks. This pattern is relatively straightforward and, except for the service registry, there are no other moving parts. Also, since the client knows about the available services instances it can make intelligent, application-specific load balancing decisions such as using hashing consistently. One significant drawback of this pattern is that it couples the client to the service registry. You must implement client-side service discovery logic for each programming language and framework used by your service clients

### Authorization (Oauth2) server

For issuing tokens and authorize requests.

## Backend Microservices

While the backing services in the middle layer are still considered to be microservices, they solve a set of concerns that are purely operational and security-related. The business logic of this application sits almost entirely in our bottom layer.

### atalog

The Catalog consists of categorized products. Products can be in one ore more categories, and category can contain one ore more products. Products and Categories are exposed as REST resources using Spring Data RESTs capability to automatically expose Spring Data JPA repositories contained in the application

### Reviews

Review is entity(document) related to product by productId and to customer(user) by userName. The repository under this service is MongoDb Reviews are exposed as REST resources using Spring Data RESTs capability to automatically expose Spring Data Mongo repositories contained in the application.

### Recommendations

This service consists of Person and Product entities and Like relation entity that links them. Recommendations are exposed as REST resources using Spring Data RESTs capability to automatically expose Spring Data Neo4j(Graph) repositories contained in the application.

### Orders

The implementation consists of mainly two parts, the order and the payment part. The Orders are exposed as REST resources using Spring Data RESTs capability to automatically expose Spring Data JPA repositories contained in the application. The Payment process are implemented manually using a Spring MVC controller (PaymentController).

## Running Instructions

### Via maven (spring boot)

Make sure you have Neo4J and MongoDB running on localhost (on default ports). $ cd micro-ecommerce/microservices-config-server

$ mvn spring-boot:run

$ cd micro-ecommerce/microservices-eureka

$ mvn spring-boot:run

$ cd micro-ecommerce/microservices-authserver

$ mvn spring-boot:run

Repeat this for all other services that you want to run. Please note that the order is important (config-server, erureka, authserver)

* After you run services, trigger shell scripts under script folder of each service to create sample data.

### Via docker

$ cd micro-ecommerce

$ mvn clean install

$ docker-compose up --build -d

#### Usage

##### Get a token:

$ curl -X POST -vu acme:acmesecret http://localhost:9999/uaa/oauth/token -H "Accept: application/json" -d "password=idugalic&username=idugalic&grant\_type=password&client\_secret=acmesecret&client\_id=acme"

##### Catalog service:

$ curl http://localhost:8080/ -H "Authorization: Bearer <YOUR TOKEN>"

##### Reviews service:

$ curl http://localhost:8081/ -H "Authorization: Bearer <YOUR TOKEN>"

##### Recommendations service:

$ curl http://localhost:8082/ -H "Authorization: Bearer <YOUR TOKEN>"

##### Orders service:

$ curl http://localhost:8083/ -H "Authorization: Bearer <YOUR TOKEN>"

##### Catalog service(proxy) :

$ curl http://localhost:9000/catalog -H "Authorization: Bearer <YOUR TOKEN>"

##### Reviews service(proxy):

$ curl http://localhost:9000/reviews -H "Authorization: Bearer <YOUR TOKEN>"

##### Recommendations service(proxy):

$ curl http://localhost:9000/recommendations -H "Authorization: Bearer <YOUR TOKEN>"

##### Orders service(proxy):

$ curl http://localhost:9000/orders -H "Authorization: Bearer <YOUR TOKEN>"