

**Description**

Spring Microservice - Spring Microservice - Discovery Server Security + API Gateway Security + Circuit Breaker

*Discovery Server or Discovery Service implements communication between API's.*

*API Gateway Service implements Load Balance and One-Single-Endpoint for multiple API endpoints.*

**What is Distributed Tracing**

Distributed Tracing helps to track and monitor the request from start to finish.

What is Span ID: it shows the number of trips / jump the request does inside our system (it has a unique ID for the whole request).

*E.g.: API Gateway (Span 1) -> Order Service (Span 2) -> Inventory Service (Span 3)*

It helps, for example, detect if we have a performance issue.

**What is Circuit Break**

WebClient has Asynchronus communication, sometimes the ordering service may try to communicate with the inventory service and not get a response (slowness, unavailability).

For example, problems that affect database performance can affect communication between APIs (resilience issue).

To fix this, Circuit Break saves states to troubleshoot WebClient Async by checking whether the inventory service is available.

*Resilience4j library.*

**Circuit Break - Test Case**

Health Check: <http://localhost:8081/actuator/health>

Main circuit break url: <http://localhost:8081/actuator>

*Check Inventory and Order console exceptions.*

**Docker**

**Network**

*# Containers Docker Network Creation*

docker network create my-network

**Containers**

*# Directory: docker-development-environment*

docker compose up -d

**Eureka Server / Service**

Endpoint: <http://localhost:8761/>

username: eureka

password: eureka

**Zipkin**

Zipkin: <http://localhost:9411/>

**Kafdrop**

Kafdrop: <http://localhost:9091/>

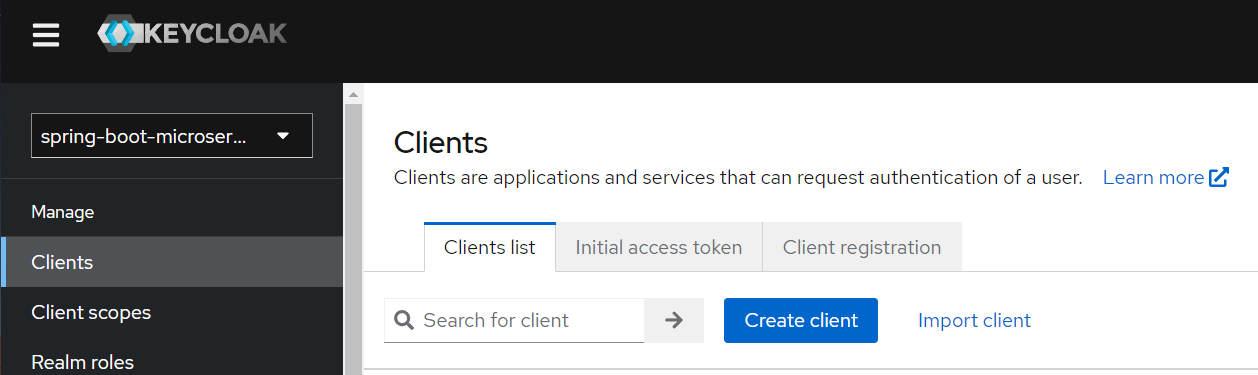
**Keycloak**

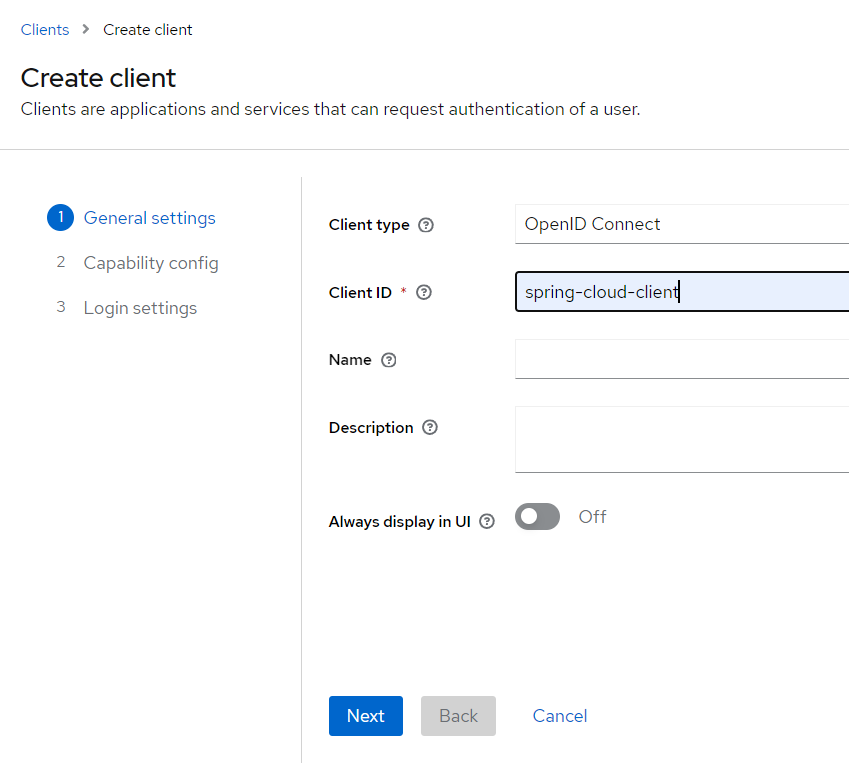
Keycloak: <http://localhost:8090/>

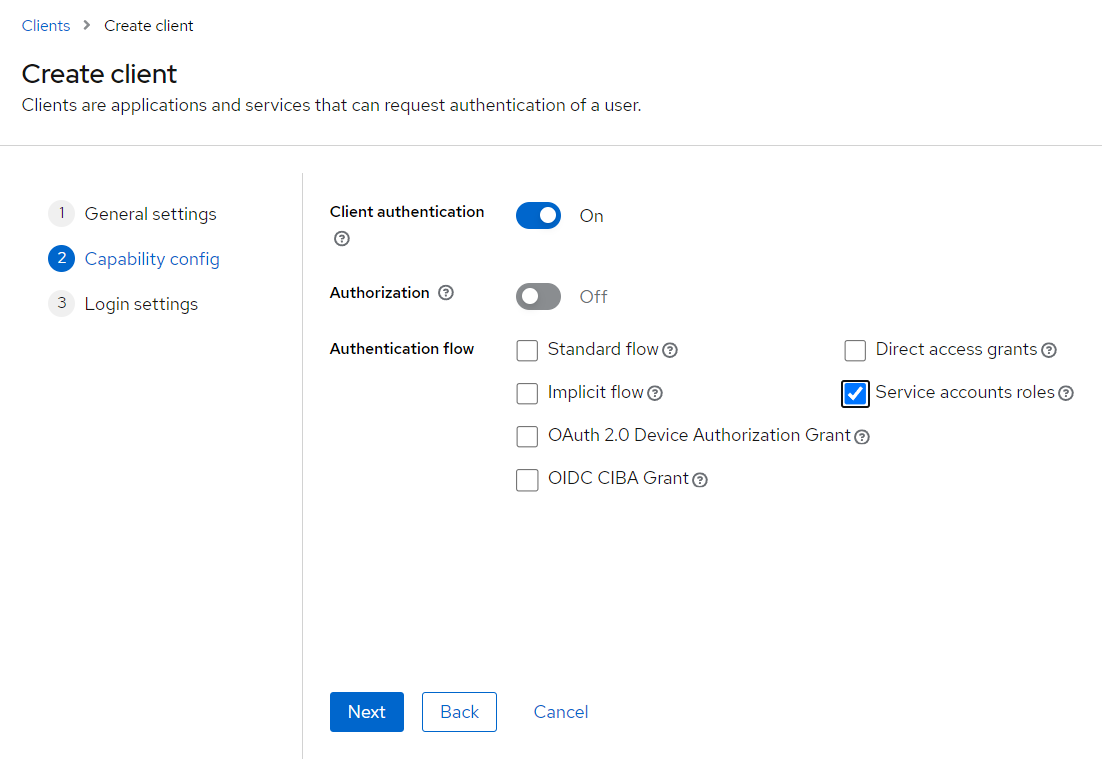
username: admin

password: admin

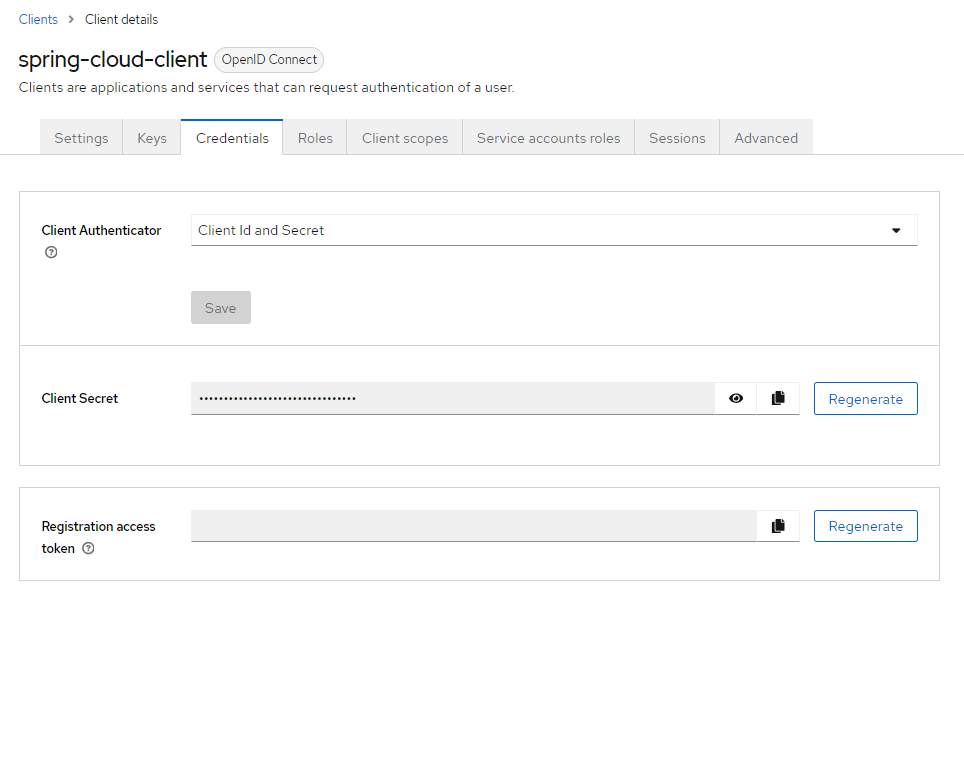
**Create a new realm**



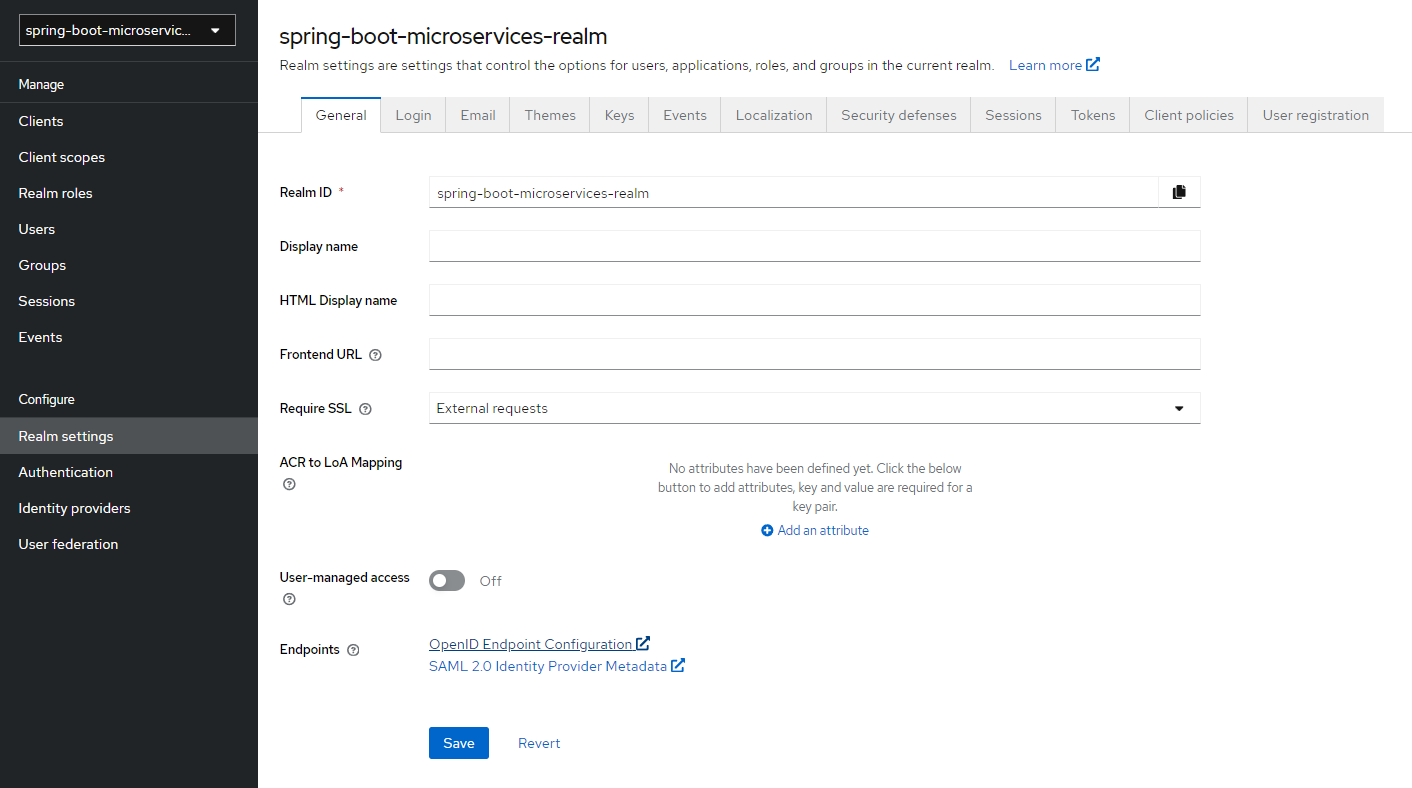




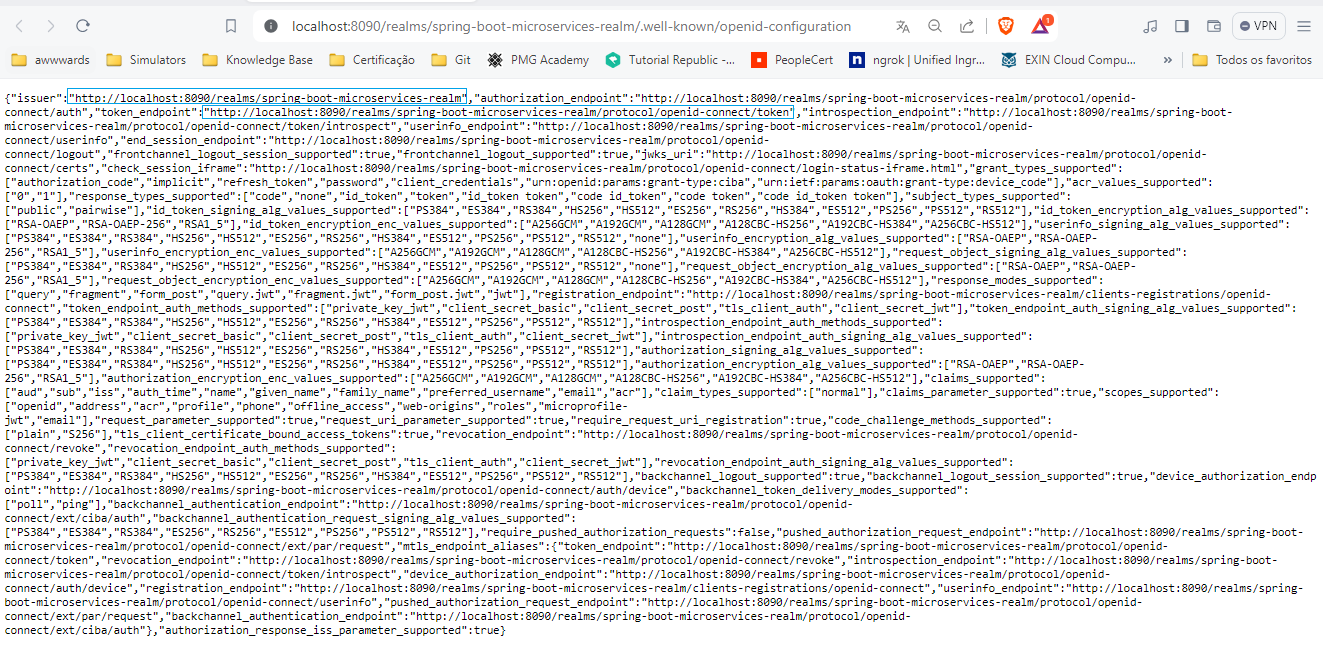
**Copy and save client secret**



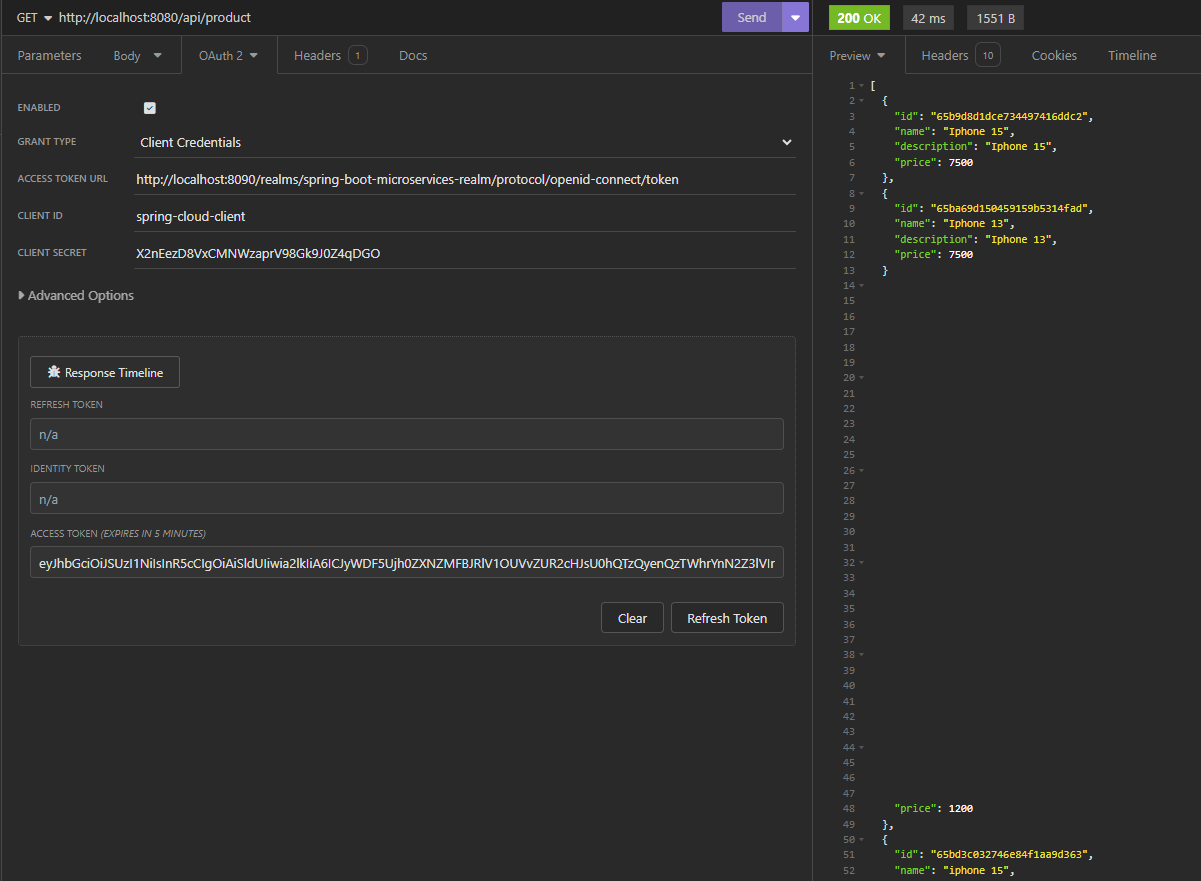
**Open OpenID Endpoint Configuration link**

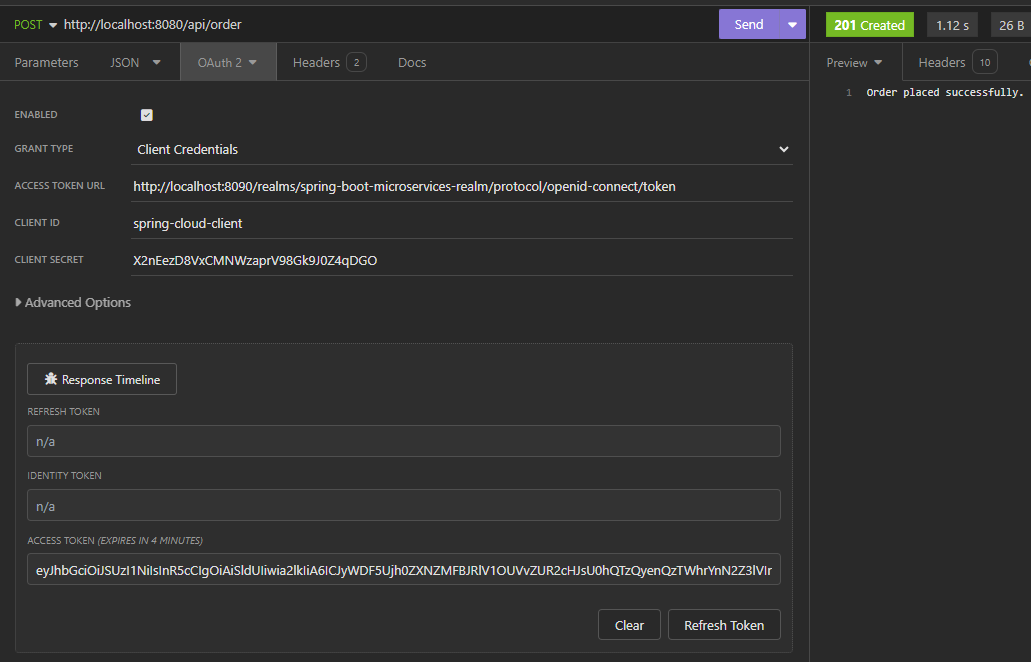


**Copy and save issuer and token endpoint**



**Insomnia**





**API Gateway**

Now we can access API's via API Gateway url:

api-service-endpoint:

[http://localhost:8080/{api-service-endpoint}](http://localhost:8080/%7Bapi-service-endpoint%7D)

**Product**: /api/product

**Order**: /api/order

**Inventory**: No implemented via API Gateway

**Endpoints**

*# - product-service*

GET http://localhost:8080/api/product

POST http://localhost:8080/api/product

{

"name":"iphone 15",

"description":"iphone 15",

"price":1200

}

*# - order-service (depends\_on: inventory-service)*

P.S.: iphone\_15\_black drops java.lang.IllegalArgumentException message:

"Product is not in stock, please try again later"

POST http://localhost:8080/api/order

{

"orderLineItemsDtoList": [

{

"skucode":"iphone\_15",

"price": 1200,

"quantity": 1

}

]

}

or

{

"orderLineItemsDtoList": [

{

"skucode":"iphone\_15",

"price": 1200,

"quantity": 1

},

{

"skucode":"iphone\_15\_black",

"price": 1200,

"quantity": 1

}

]

}

*# - inventory-service*

P.S.: Not implemented via API Gateway (access via API Endpoint, only)

GET http://localhost:{random-port}/api/inventory

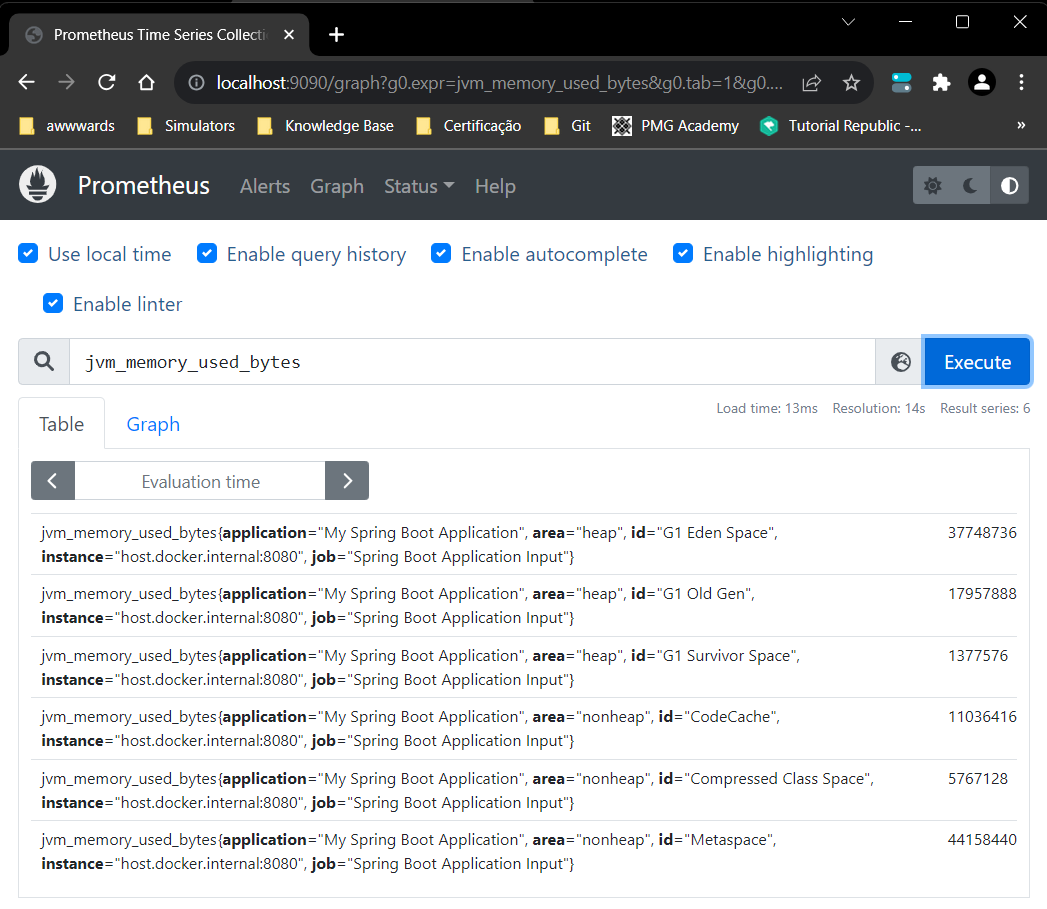
http://localhost:{random-port}/api/inventory?skucode=iphone\_15

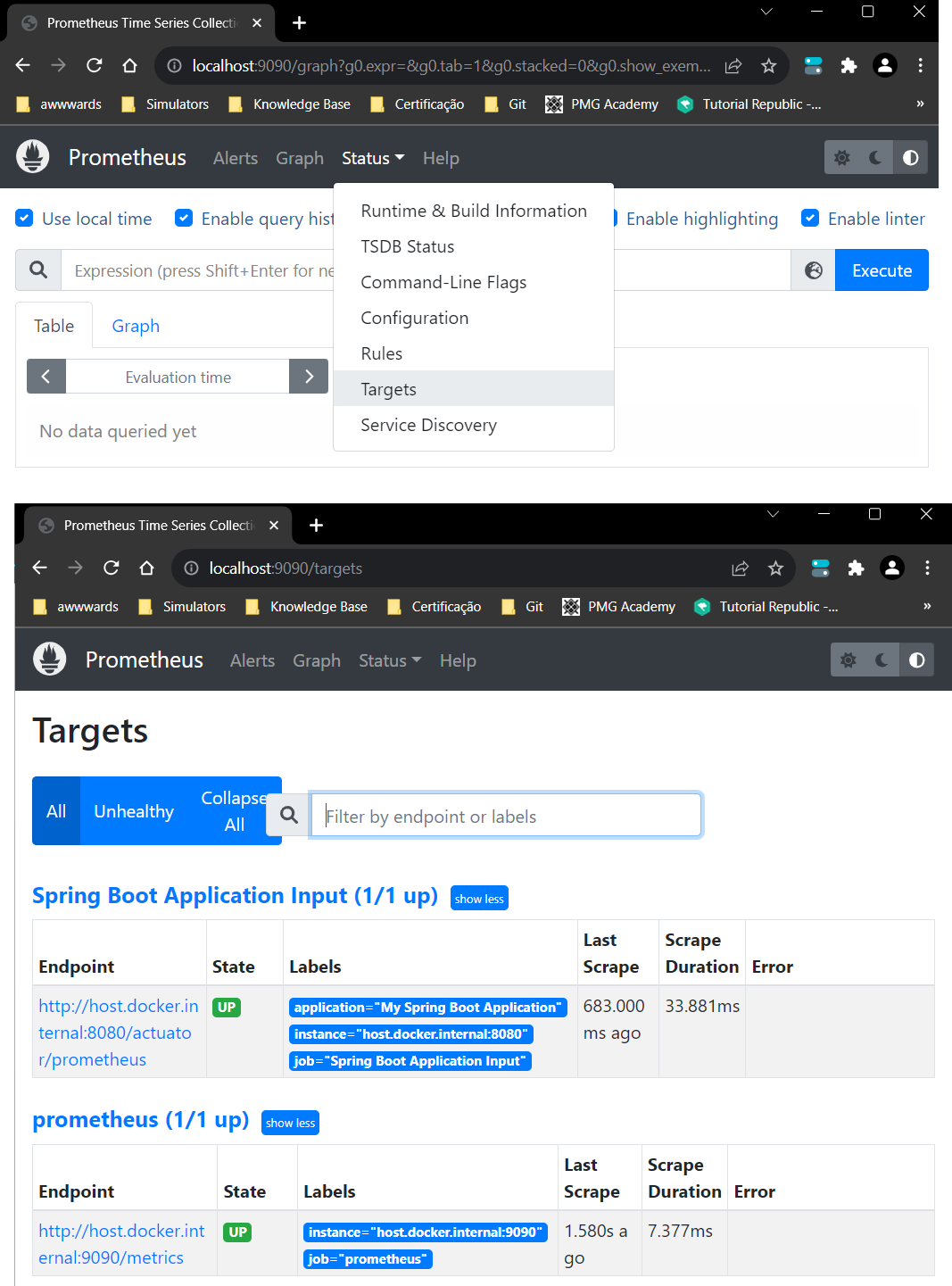
http://localhost:{random-port}/api/inventory?skucode=iphone\_15&skucode=iphone\_15\_black

**Prometheus**

Prometheus: <http://localhost:9090/>

Main endpoint: <http://localhost:8080/actuator>





**Grafana**

Grafana: <http://localhost:3000/>

username: grafana

password: grafana

**Create Dashboard**

* Add Visualization
* Add Prometheus Data Source
* Add Prometheus Server URL: [http://host.docker.internal:9090](http://host.docker.internal:9090/)
* Save and Test (**Explore view** link)

**Access application endpoint to produce Grafana data view**

P.S.: You also can import "Grafana\_Dashboard.json" dashboard.

