Table of Contents

[1. Prerequisites 2](#_Toc29243901)

[2. Application Information: 2](#_Toc29243902)

[a. Registry 2](#_Toc29243903)

[i. Spring boot cloud config server 2](#_Toc29243904)

[ii. Eureka discovery server 2](#_Toc29243905)

[iii. Spring boot cloud admin server 2](#_Toc29243906)

[iv. Screenshots 2](#_Toc29243907)

[b. Gateway 4](#_Toc29243908)

[i. Spring boot config client 4](#_Toc29243909)

[ii. Eureka discovery client 4](#_Toc29243910)

[iii. Zuul Proxy Gateway 4](#_Toc29243911)

[iv. Swagger UI 4](#_Toc29243912)

[v. Spring cloud sleuth 5](#_Toc29243913)

[c. Greeting-service 5](#_Toc29243914)

[i. Spring boot config client 5](#_Toc29243915)

[ii. Eureka discovery client 5](#_Toc29243916)

[iii. Swagger UI 5](#_Toc29243917)

[iv. Spring cloud sleuth 6](#_Toc29243918)

[3. Product usage guide 6](#_Toc29243919)

[a. Starting the applications 6](#_Toc29243920)

[b. Stopping the applications 6](#_Toc29243921)

[c. Deleting the applications 7](#_Toc29243922)

[d. Starting all applications with docker 7](#_Toc29243923)

[e. Stopping all applications with docker 7](#_Toc29243924)

[4. API Information 8](#_Toc29243925)

[a. Gateway 8](#_Toc29243926)

[i. Swagger URL 8](#_Toc29243927)

[ii. Authentication API 8](#_Toc29243928)

[iii. Swagger Authorization 8](#_Toc29243929)

[iv. Greet API: 9](#_Toc29243930)

[v. Refresh Token API 9](#_Toc29243931)

[b. Greeting-Service 10](#_Toc29243932)

[i. Greet API 10](#_Toc29243933)

[5. Postman Setup 11](#_Toc29243934)

# Prerequisites

* 1. Java 11 – should be installed in the system.
  2. docker & docker-compose – should be installed in the system (optional to run containers)
  3. Mongo DB instance should be up and running in the local machine on port 27017.
  4. Altimetrik.sh - copy this file to an empty folder. **(Important)**

# Application Information:

* 1. Registry – This application includes the following components
     1. Spring boot cloud config server – Spring Cloud Config provides server-side and client-side support for externalized configuration in a distributed system. With the Config Server, you have a central place to manage external properties for applications across all environments. This component is used to propagate the configurations to other applications without restarting them. This component has been connected and fetches the configurations from online git repository <https://github.com/sonirahul/git-config-server.git>.

Reference Link: <https://cloud.spring.io/spring-cloud-config/reference/html/>

* + 1. Eureka discovery server – Service Discovery is one of the key tenets of a microservice-based architecture. Trying to hand-configure each client or some form of convention can be difficult to do and can be brittle. Eureka is the Netflix Service Discovery Server and Client. The server can be configured and deployed to be highly available, with each server replicating state about the registered services to the others.

Once the “Registry” application is started Eureka discovery can be access on the following URL : <http://localhost:8761/>

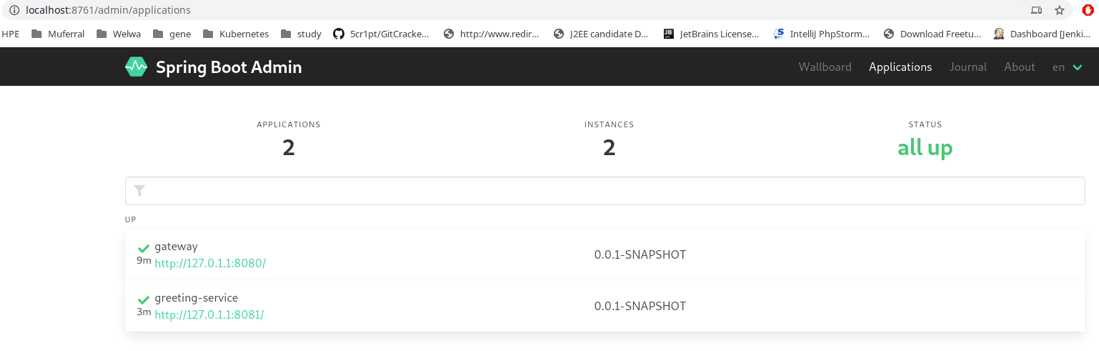
Reference Link: <https://cloud.spring.io/spring-cloud-netflix/multi/multi__service_discovery_eureka_clients.html>

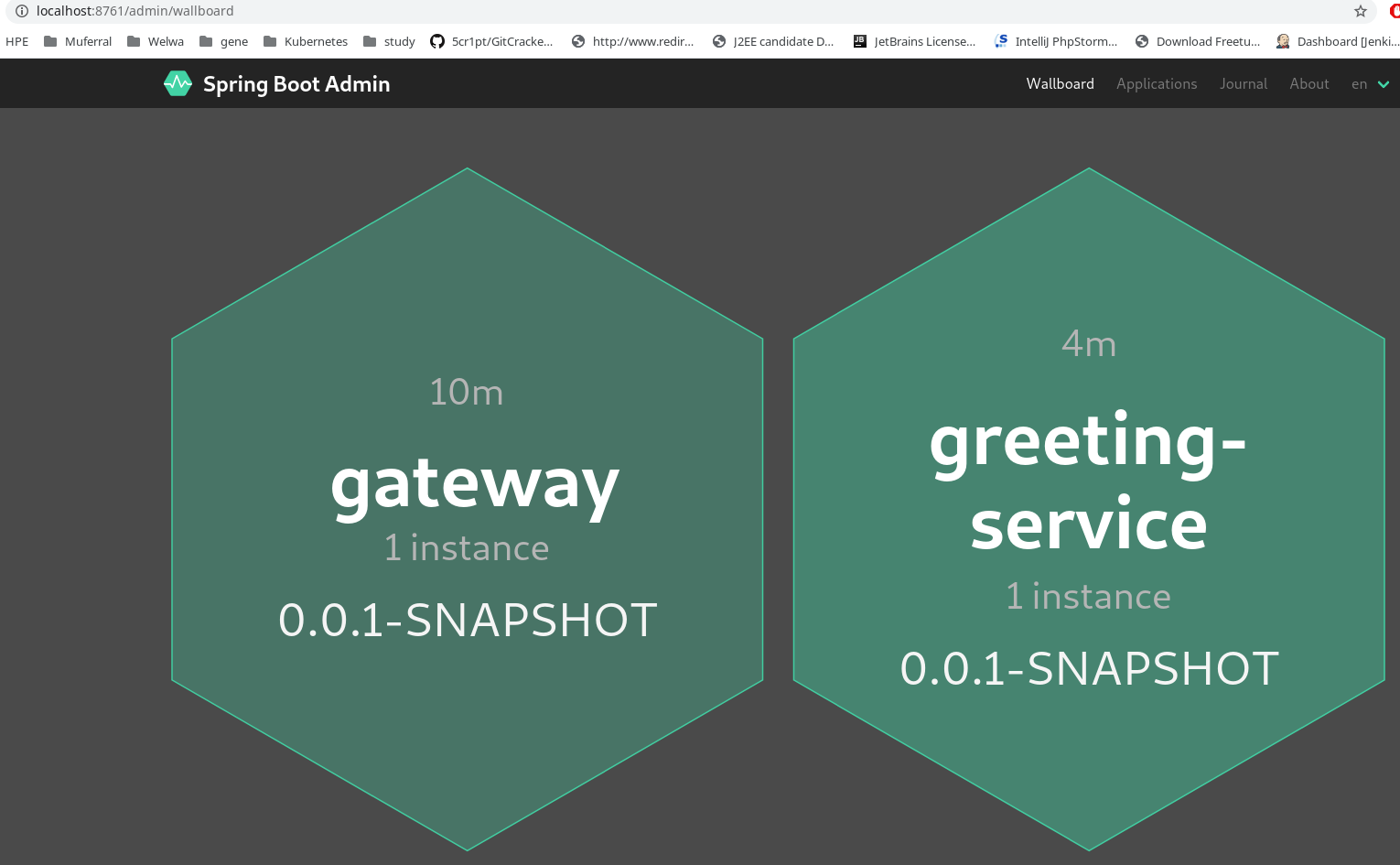
* + 1. Spring boot cloud admin server – This is a component to manager and monitor the spring boot applications. Spring boot actuator links & metrices for all the applications can be accessed from admin server.

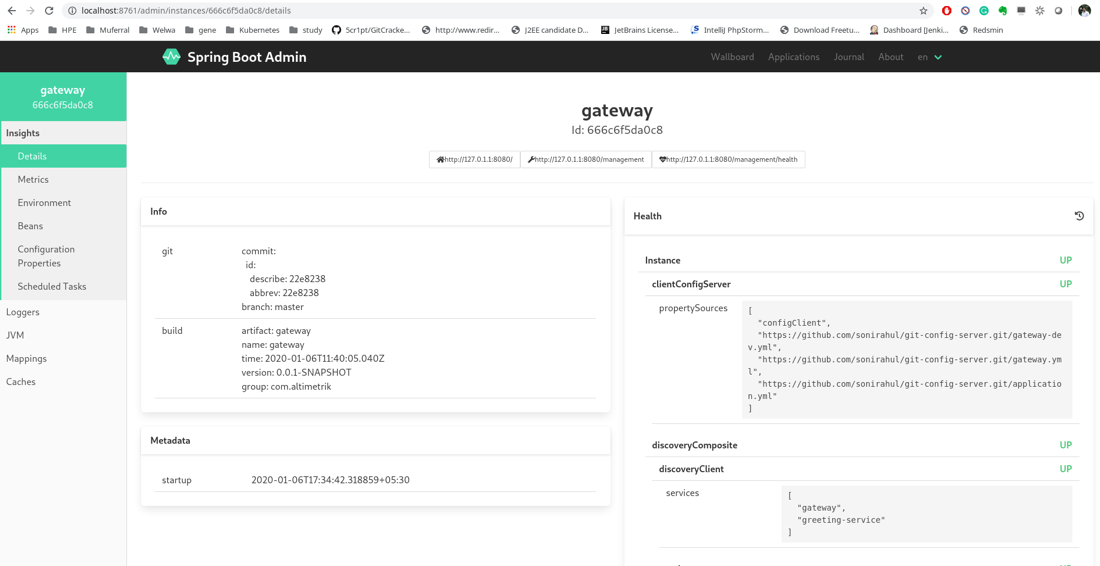
Once the “Registry” application is started spring boot admin server can be accessed on the following URL: <http://localhost:8761/admin>

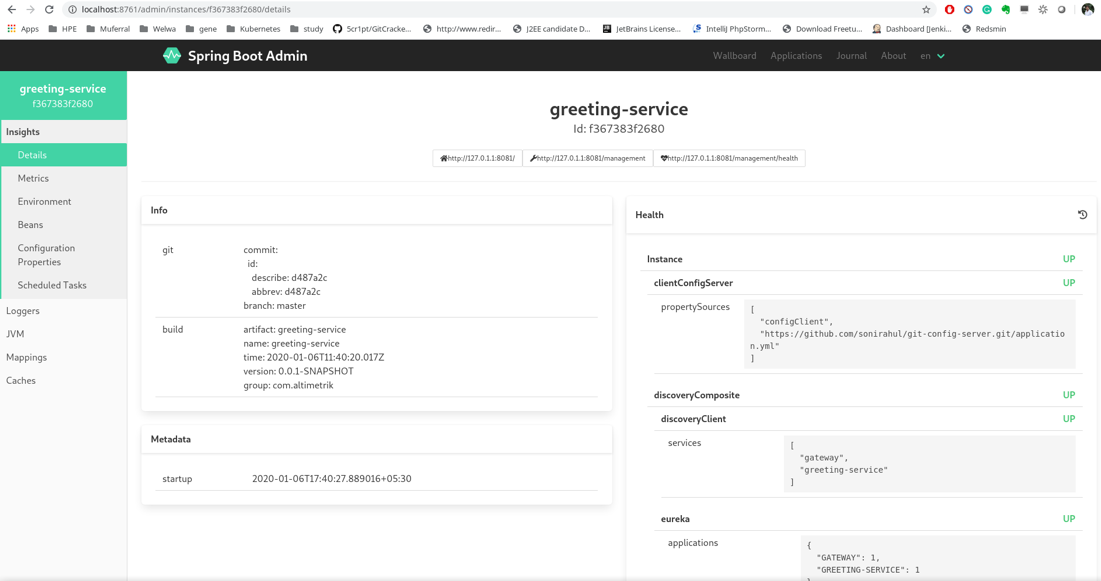
Reference Link: <https://codecentric.github.io/spring-boot-admin/current/>

* + 1. Screenshots:



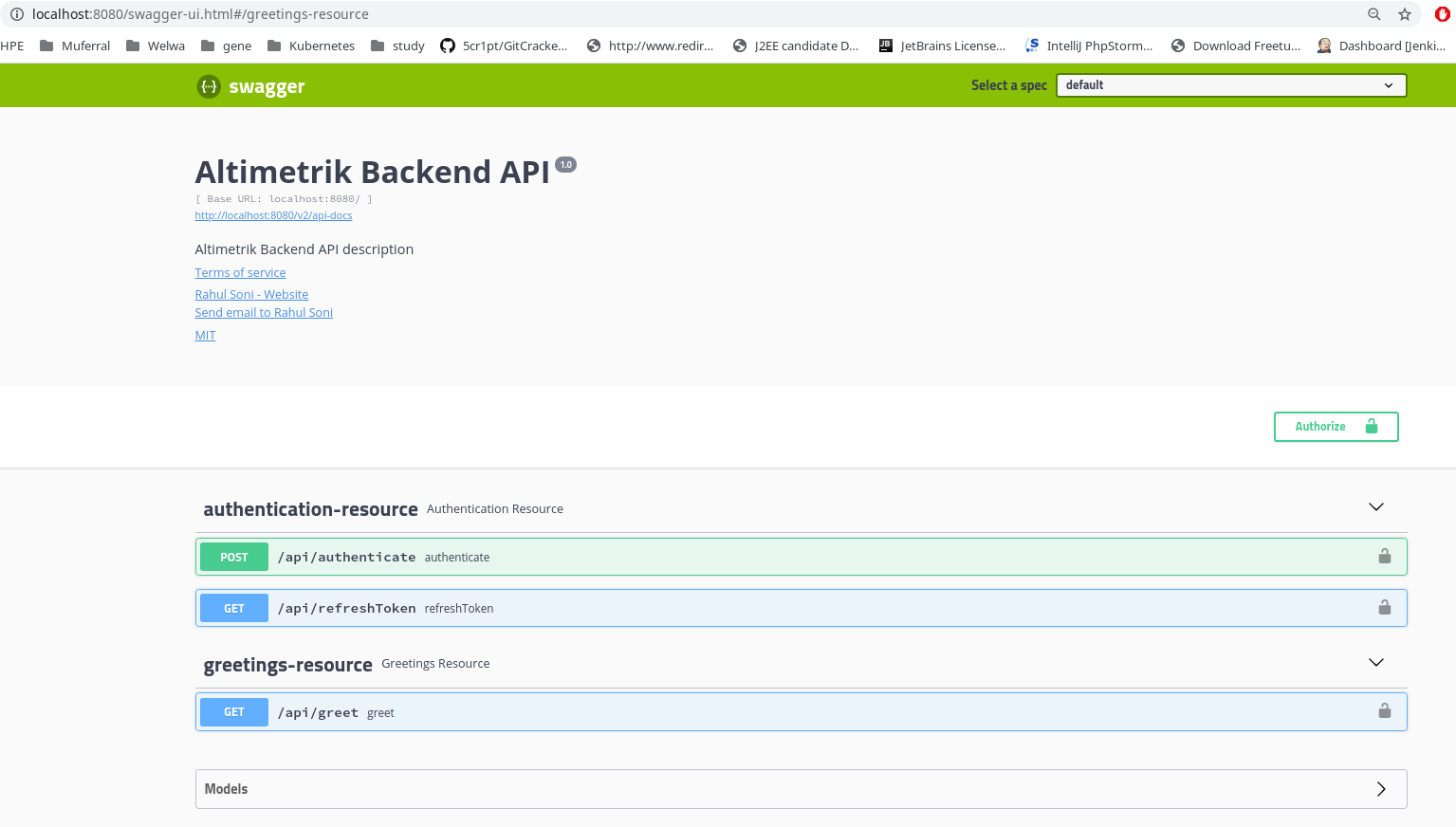






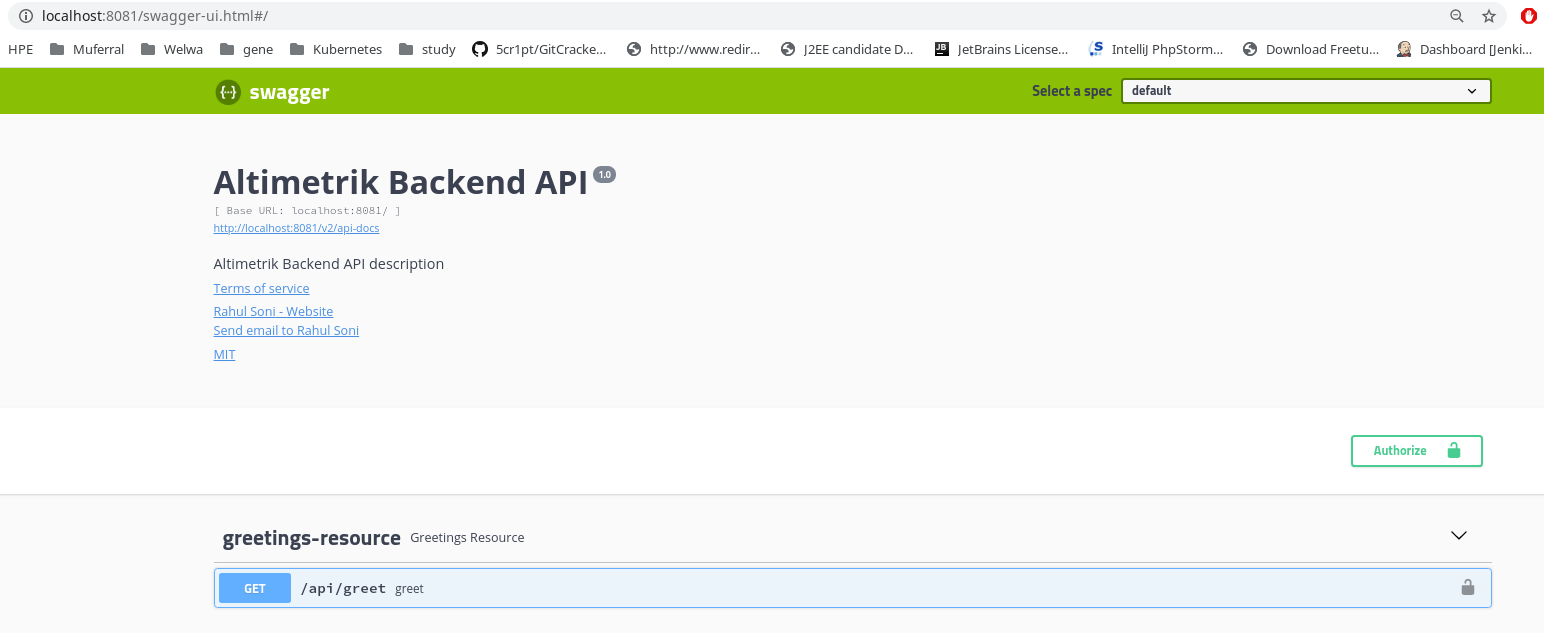
* 1. Gateway – This application includes the following components:
     1. Spring boot config client – This component acts as a client to Spring boot config server configured in the “Registry” application. The benefit we get by configuring this component in our application is to propagate the configuration changes dynamically at runtime with ZERO downtime.
     2. Eureka discovery client – This component acts as a client to Eureka discovery server configured in the “Registry” application. The benefit we get by configuring this component in our application is that on a single portal of Eureka we can see all the applications/services which are UP/Down, port information & # of instances for each application/service.
     3. Zuul Proxy Gateway – Zuul is a JVM-based router and server-side load balancer from Netflix. Zuul can be used for “Authentication”, “Rate Limiting”, “Traffic Management” and “Dynamic Routing”.
     4. Swagger UI – This component is used to document the Rest API’s.

Once the “Gateway” application is started it can be accessed on the following URL: <http://localhost:8080/swagger-ui.html>



* + 1. Spring cloud sleuth – Spring Cloud Sleuth implements a distributed tracing solution for Spring Cloud applications. This enables us to trace a particular request across multiple services using a unique identifier.
  1. Greeting-service – This application includes the following components:
     1. Spring boot config client – This component acts as a client to Spring boot config server configured in the “Greeting-service” application. The benefit we get by configuring this component in our application is to propagate the configuration changes dynamically at runtime with ZERO downtime.
     2. Eureka discovery client – This component acts as a client to Eureka discovery server configured in the “Registry” application. The benefit we get by configuring this component in our application is that on a single portal of Eureka we can see all the applications/services which are UP/Down, port information & # of instances for each application/service.
     3. Swagger UI – This component is used to document the Rest API’s.

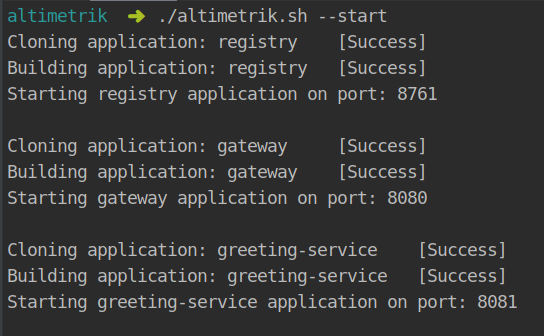
Once the “Gateway” application is started it can be accessed on the following URL: <http://localhost:8081/swagger-ui.html>



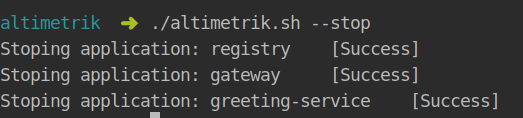
* + 1. Spring cloud sleuth – Spring Cloud Sleuth implements a distributed tracing solution for Spring Cloud applications. This enables us to trace a particular request across multiple services using a unique identifier.

# Product usage guide

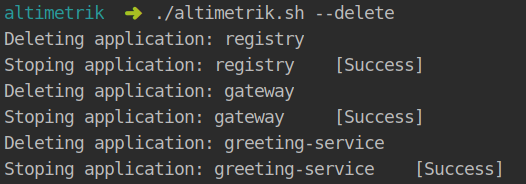
* 1. Starting the applications: Execute the following command. This command clones the git repositories for all the applications, builds them all and finally starts the applications on specific ports.
     1. ./altimetrik.sh --start



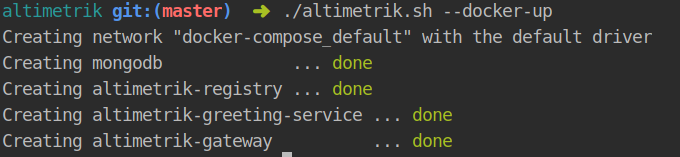
* 1. Stopping the applications: Execute the following command. This command stops all the running applications.
     1. ./altimetrik.sh --stop



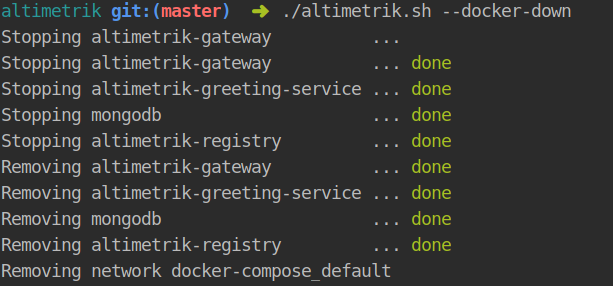
* 1. Deleting the applications: Execute the following command. This command stops all the running applications and deletes their respective git repo’s as well.
     1. ./altimetrik.sh --delete



* 1. Starting all applications with docker: Execute the following command. This command starts all the applications using docker.
     1. ./altimetrik.sh --docker-up



* 1. Stopping all applications with docker: Execute the following command. This command starts all the applications using docker.
     1. ./altimetrik.sh --docker-down



# API Information

* 1. Gateway:
     1. Swagger URL: <http://localhost:8080/swagger-ui.html>
     2. Authentication API: Execute the API “[/api/authenticate](http://localhost:8080/swagger-ui.html#/operations/authentication-resource/authenticateUsingPOST)" with the following request payload.

Request Payload:

{

"password": "password",

"username": "admin"

}

Response Payload:

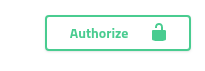
{

"jwt": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImF1dGgiOiIiLCJ1c2VySWQiOiJhZG1pbiIsImV4cCI6MTU3ODMxMzc2MH0.xLayNMDSwY4IrpoH9\_1PTMzwWUwXfto2uv6DJ9YBESKd3ccpxTLxUyaDK9kIImg29ip24q\_M1n1Ei5aqCI\_-cA",

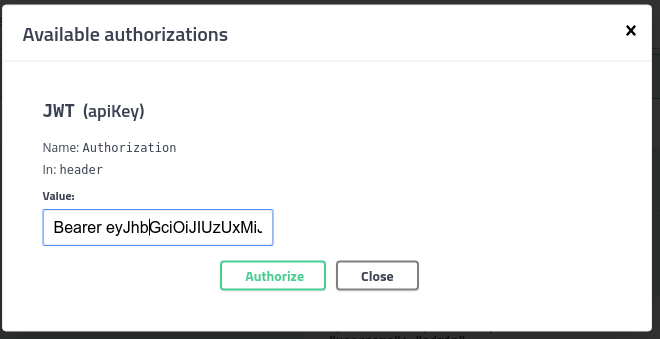
"validity": "2020-01-06T12:29:20.532+0000"

}

* + 1. Swagger Authorization: Then you will need to provide the JWT token received to authorize the swagger to use it for further API invocations.
       1. Click on the Authorize button



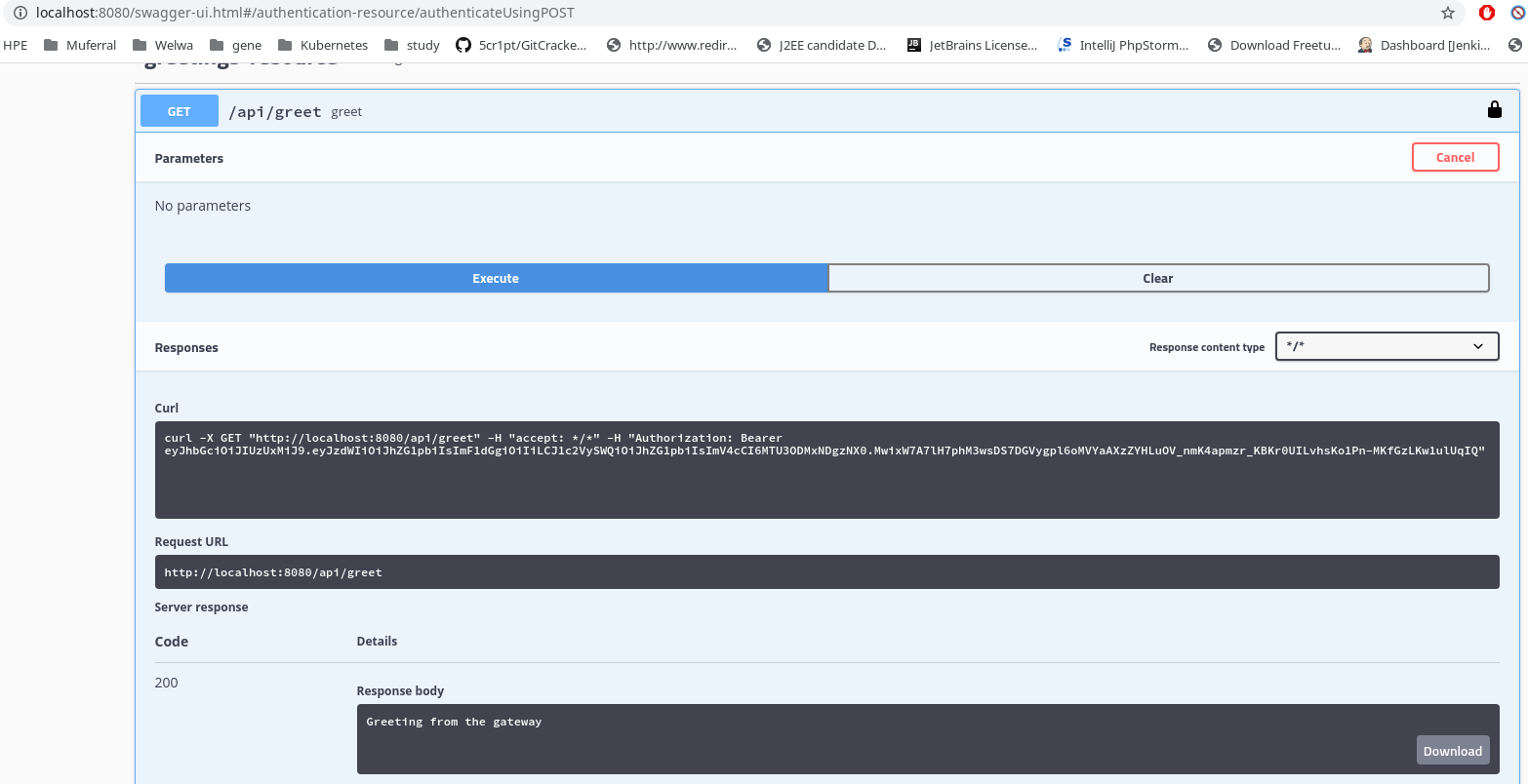
* + - 1. Then add the JWT token on the pop in the values section as shown below in a format. E.g. “Bearer <JWT Token>”



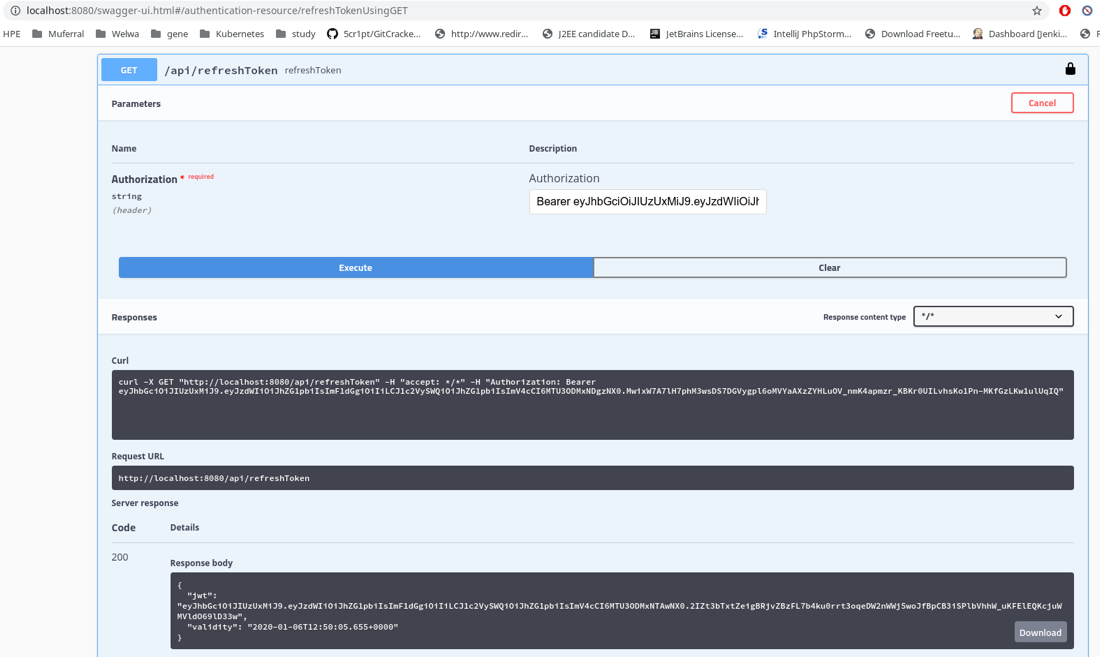
* + - 1. Finally click on the Authorize button and it should become locked as shown below:



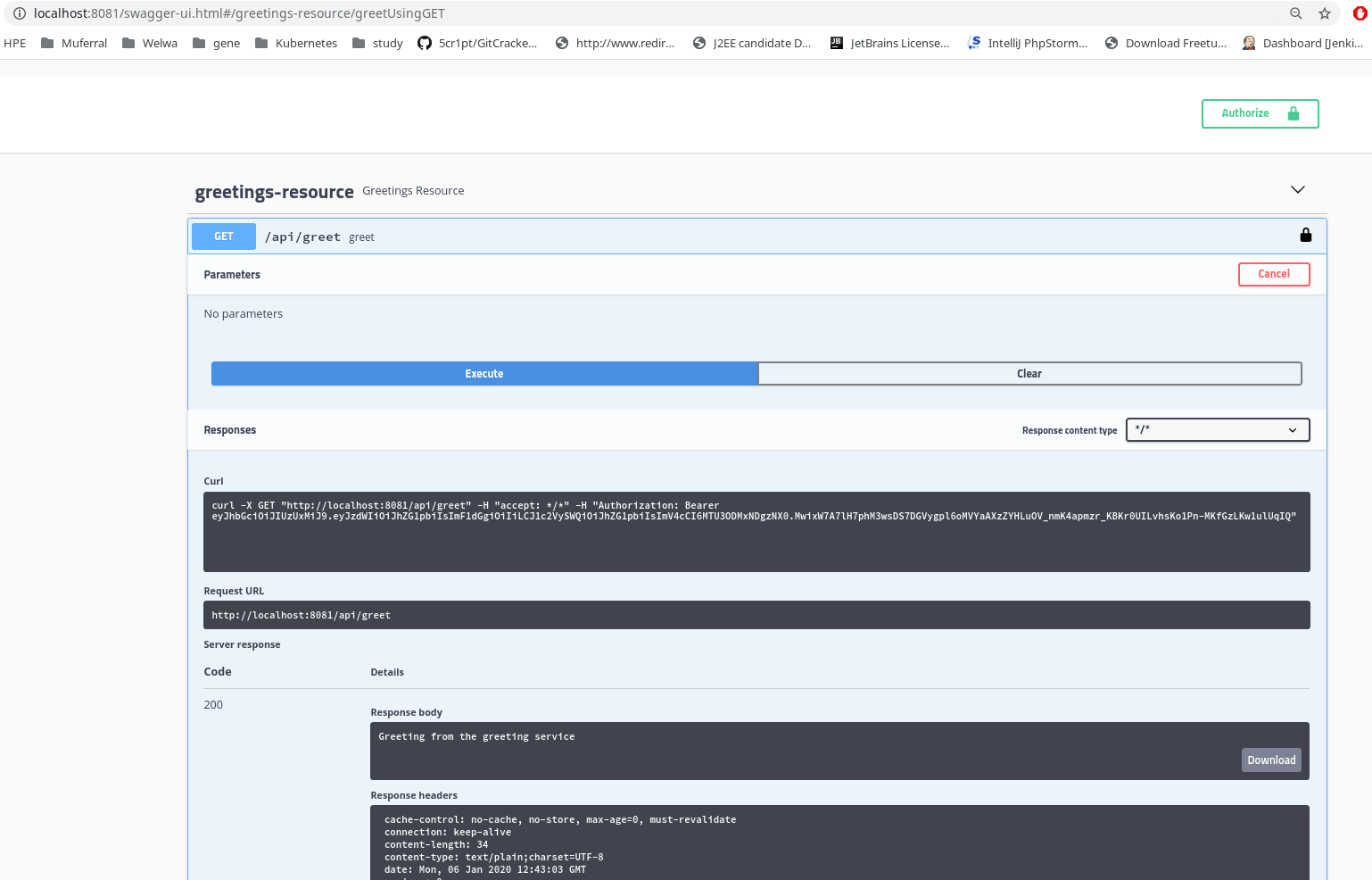
* + 1. Greet API: Now you can execute the greet API “/api/greet”. The response will be “**Greeting from the gateway”**



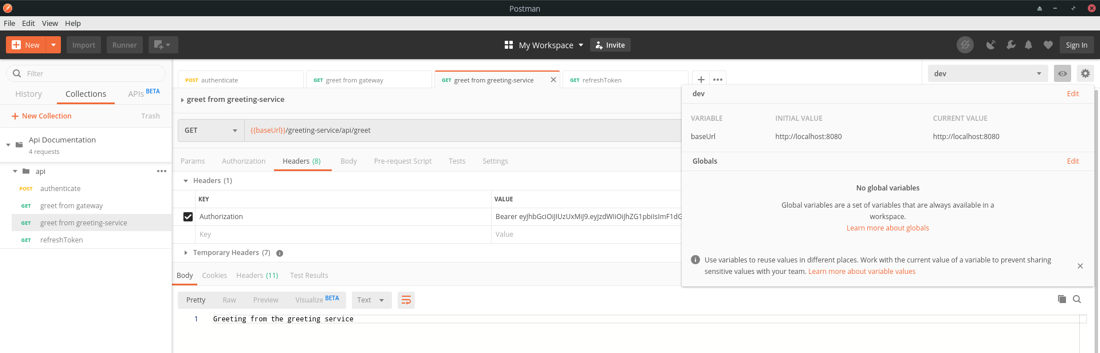
* + 1. Refresh Token API: Now you can execute the refresh token API “/api/refreshToken”.



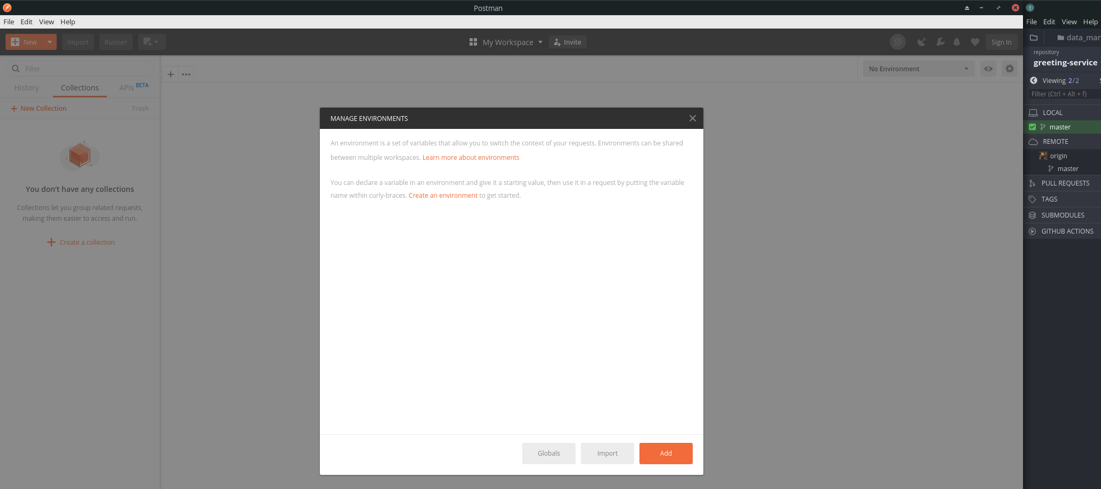
* 1. Greeting-Service:
     1. Greet API – now you can execute the greet API “/api/greet” within “greeting-service” using the same token generated in the “Gateway Application.” Although please make sure to “Authorize” using the valid token created by “Gateway” application.



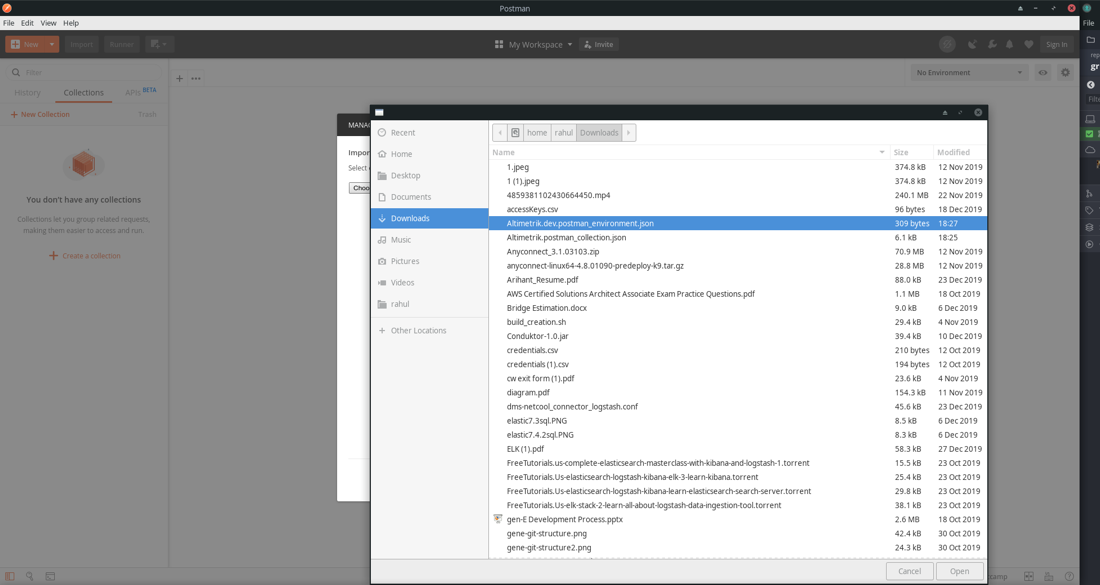
* + 1. This API can also be executed from the “Gateway” application using the ZUUL API Gateway. Although you will need to use postman for it.

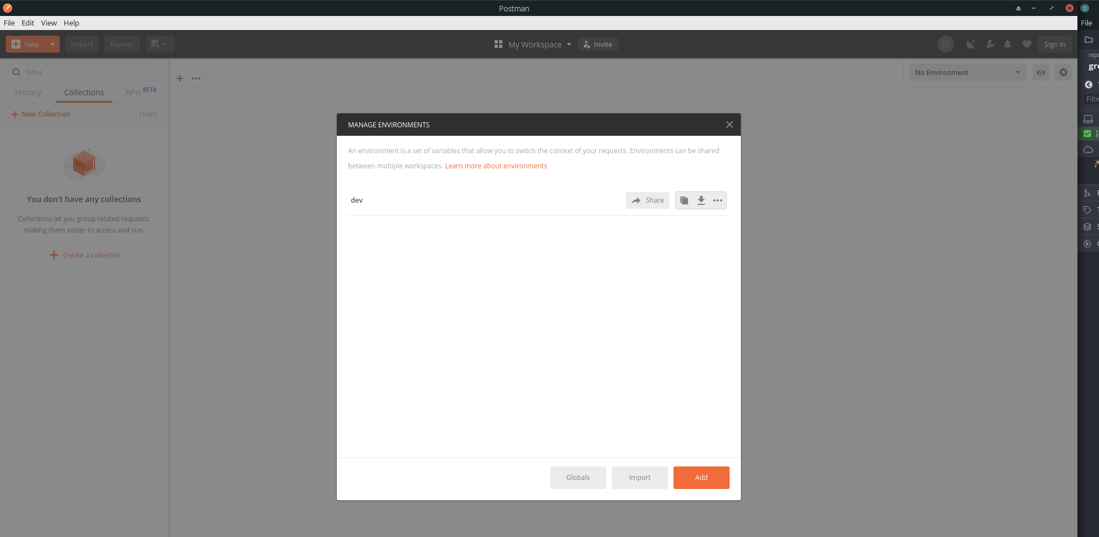


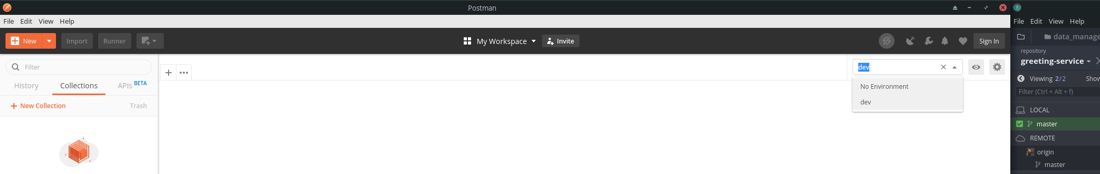
1. Postman Setup
   1. Import the environment by clicking the “Manage Environments” and import the “Altimetrik.dev.postman\_environment.json” file.



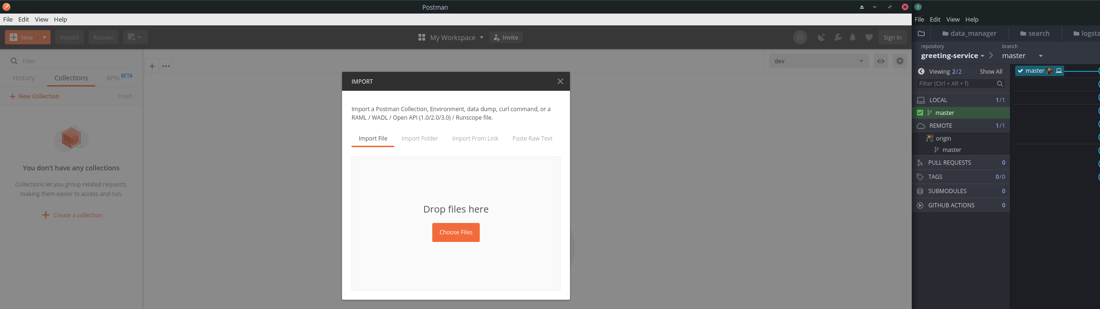
* 1. Import the “Altimetrik.dev.postman\_environment.json” file.

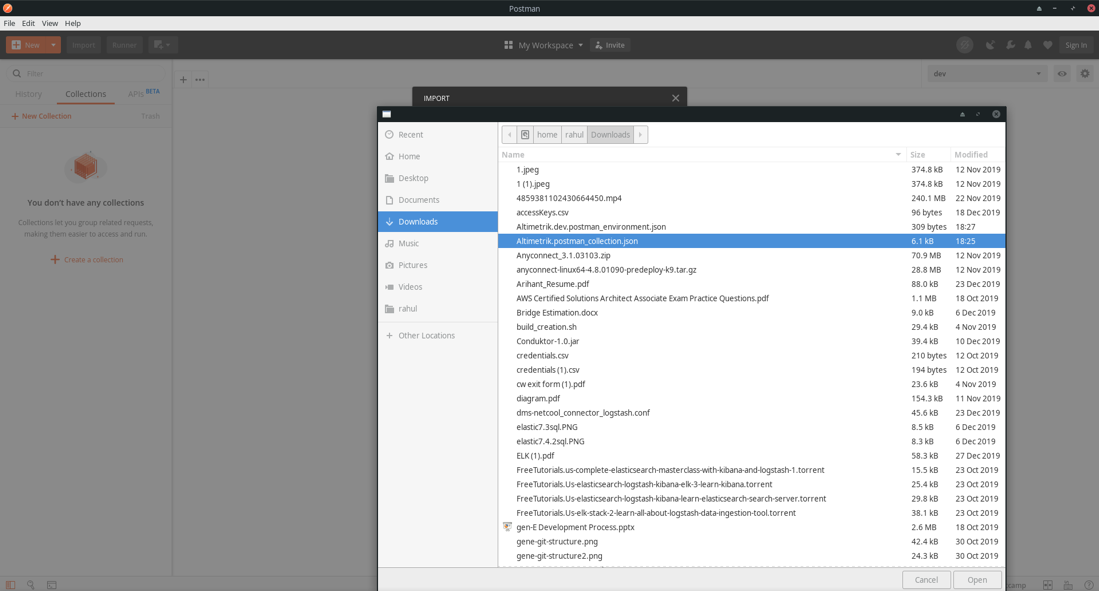


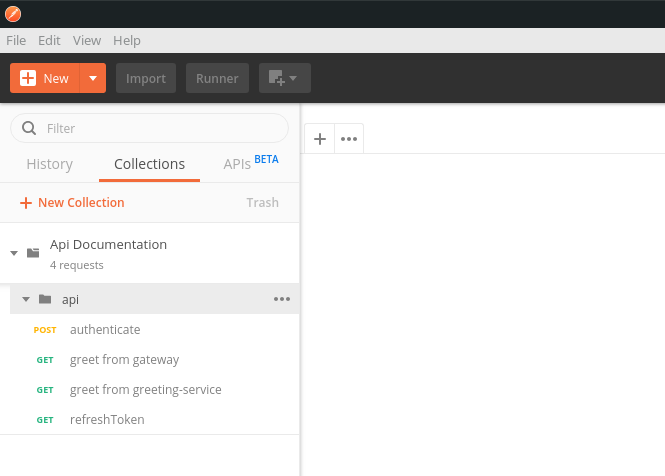




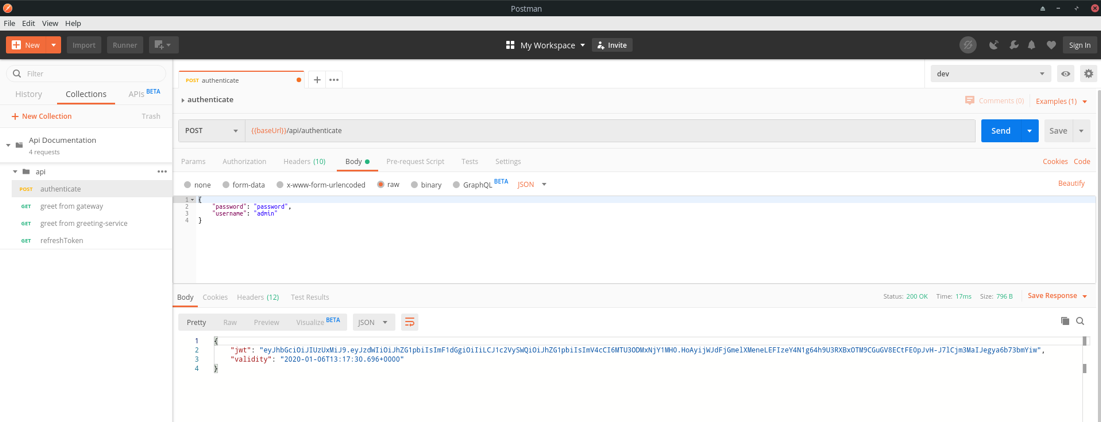
* 1. Import the Postman API collection by importing the “Altimetrik.postman\_collection.json” file.







* 1. Execute the “Authenticate” API. Make sure to select the environment as “dev” and pass the Request Body Payload as shown in the screenshot.



* 1. Execute the “Greet” API from the greeting-service.

