

# Microservices Architectures - Practices with JHipster

Henrique Figueiredo Conte, Teodor Neagu, Tetiana Yakovenko

January 2023

## 1 Introduction

The goal of the project is to practice microservices architecture using JHipster, Google Cloud Platform (GCP), Prometheus, Grafana and Gatling. The project was made by the following group:

- Henrique Figueiredo Conte - [Github](#) - [henrique.figueiredo-conte@grenoble-inp.org](mailto:henrique.figueiredo-conte@grenoble-inp.org)
- Tetiana Yakovenko - [Github](#) - [tetiana.yakovenko@grenoble-inp.org](mailto:tetiana.yakovenko@grenoble-inp.org)
- Teodor Neagu - [Github](#) - [teodor.neagu@grenoble-inp.org](mailto:teodor.neagu@grenoble-inp.org)

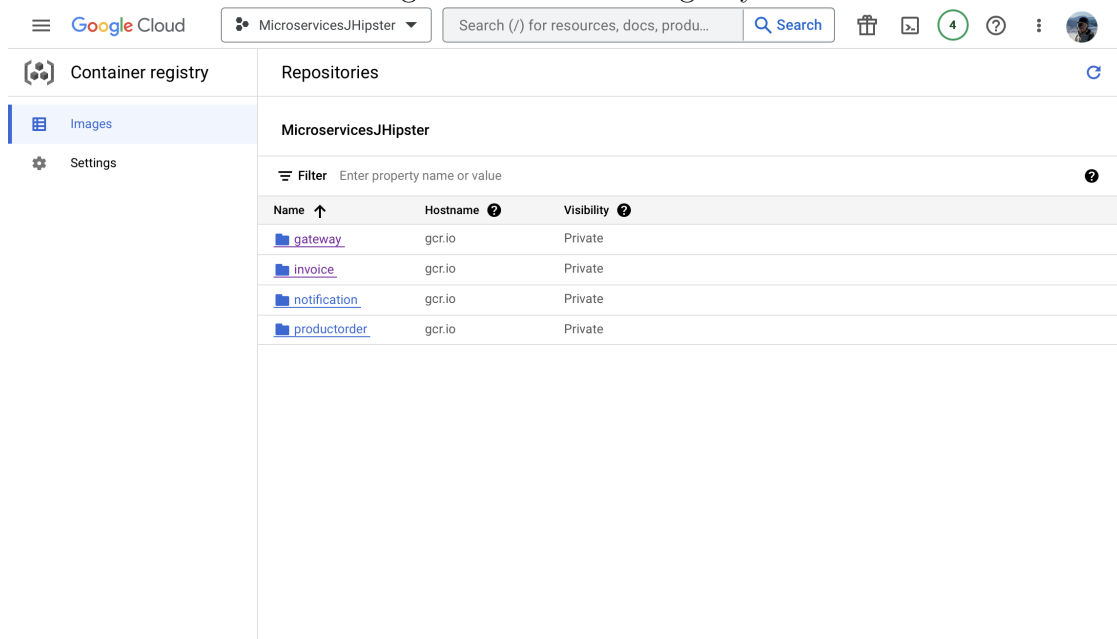
The source code is available on a [Github repository](#).

## 2 Tasks

### 2.1 Deployment of Microservices with JHipster on GCP

The first task was to deploy microservices with JHipster on GCP. In order to do that, we followed the [tutorial given by the teacher](#), and after following all the steps from the *Generate the microservices architecture*, we deployed the services to GCP:

Figure 1: Container registry



This is our container registry, that manages Docker images. There, we deployed the images from the *gateway*, *invoice*, *notification* and *productorder*.

## 2.2 Enabling scalability on GCP for one microservice

In order to enable scalability for at least one microservice, we followed the tutorial from Google, [Scaling an application](#). The tutorial taught us how to enable auto scaling for a Kubernetes cluster.

Figure 2: Kubernetes cluster auto scaling

The screenshot displays the Google Cloud Kubernetes Engine console. The top navigation bar shows the Google Cloud logo, the project name 'MicroservicesJHipster', and the cluster name 'gke'. The left sidebar contains various navigation options, with 'Clusters' selected. The main content area is titled 'Clusters' and includes tabs for 'DETAILS', 'STORAGE', 'OBSERVABILITY', and 'LOGS'. The 'LOGS' tab is active, showing a message about finding cluster and autoscaler logs. Below this, there are 'CLUSTER LOGS' and 'AUTO-SCALER LOGS' sections. A 'Suggested filters (0)' section is present, followed by a 'Severity' dropdown set to 'Default' and a 'Filter' button. A table of logs is displayed with columns for 'SEVERITY', 'TIMESTAMP', and 'SUMMARY'. The first log entry is expanded, revealing a JSON payload with the following structure:

```

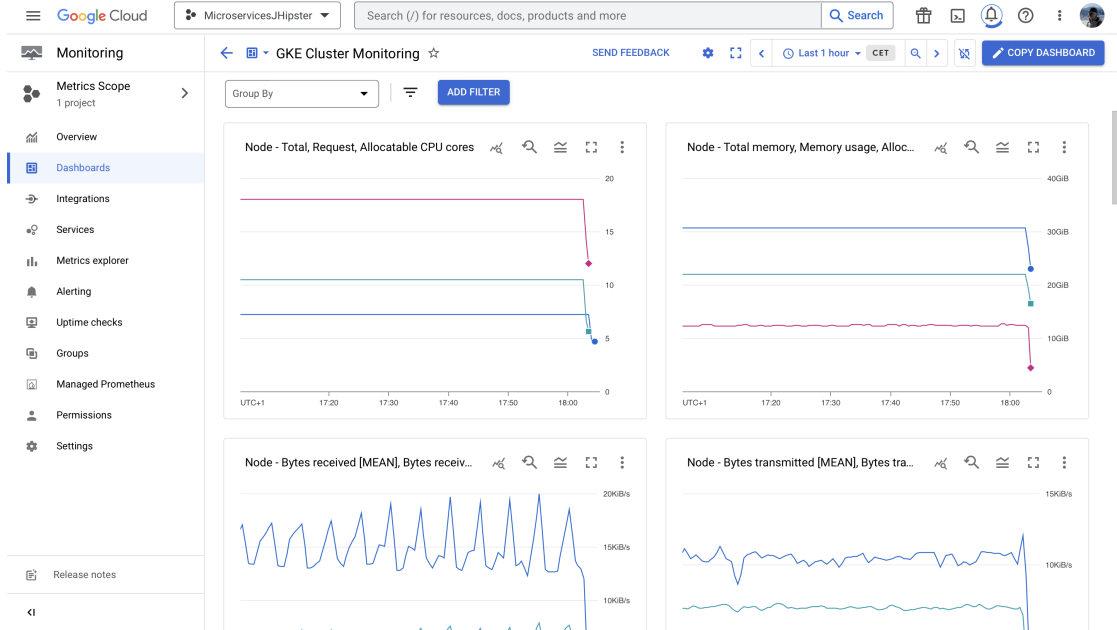
{
  insertId: "eb8c7776-563e-4363-b5b1-4a2d8e923914@a1",
  jsonPayload: {
    status: {
      measureTime: "1674479181",
      autoscaledNodesTarget: 5,
      autoscaledNodesCount: 5
    }
  },
  logName: "projects/wide-plating-329614/logs/container.googleapis.com%2Fcluster-autoscaler-visibility",
  receiveTimestamp: "2023-01-23T13:06:22.371064633Z",
  resource: {
    labels: {
      location: "europe-west1",
      cluster_name: "masteringmicroservice",
      project_id: "wide-plating-329614"
    },
    type: "k8s_cluster"
  },
  timestamp: "2023-01-23T13:06:22.298642593Z"
}

```

## 2.3 Monitoring dashboard

The Kubernetes Engine from Google Cloud already provides a monitoring dashboard, showing metrics such as allocatable CPU cores and total memory. Here is a screenshot of our monitoring dashboard:

Figure 3: GKE monitoring dashboard



## 2.4 Load injection with Gatling for demonstrating scalability

Unfortunately, we didn't understand how Gatling works in order to demonstrate scalability. According to the settings we defined, we believe our cluster will autoscale, but we couldn't see it happening in practice.