



Grenoble INP – ENSIMAG
École Nationale Supérieure d'Informatique et de Mathématiques Appliquées

Jhipster Project

Microservices Architectures - Practices with JHipster

Realized by:

Ismail BARKANI

Mehdi SAOUDI

Youssef TENNIA

Supervised by:

Pr. Didier DONSEZ

Contents

1	Deployment of online-store using Docker Compose	4
2	Deployment of online-store with Kubernetes On GCP	7
3	Load Injection tests with Gatling	10

Repository informations

Repository Link:

▷ *microservice-online-store*

Contributors:

- ▷ Ismail Barkani (Github Id: *IsmailBarkani*, Email: ismail.barkani@grenoble-inp.org)
- ▷ Saoudi Mehdi (Github Id: *SaoudiMehdi*, Email: mehdi.saoudi@grenoble-inp.org)
- ▷ Youssef Tennia (Github Id: *TenniaYoussef*, Email: youssef.tennia@grenoble-inp.org)

CHAPTER 1 Deployment of online-store using Docker Compose

In this Section, we tried to deploy our microservices application using Docker Compose, we started by containerizing each microservice artifact into a docker image.

Notice :

Working with the last version of Jhipster 7.4.1, we noticed that during the generation of a microservice using gradle, the only available building task is *bootJar* (screenshot below), also it generates a *JIB* configuration under *gradle/docker.gradle* which helps us to dockerize a microservice without using a Docker daemon : *./gradlew bootJar -Pprod jibDockerBuild*

```
Tasks runnable from root project 'gateway'

Default tasks: bootRun

Application tasks
bootRun - Runs this project as a Spring Boot application.

Build tasks
assemble - Assembles the outputs of this project.
bootBuildImage - Builds an OCI image of the application using the output of the bootJar task
bootBuildInfo - Generates a META-INF/build-info.properties file.
bootJar - Assembles an executable jar archive containing the main classes and their dependencies.
bootJarMainClassName - Resolves the name of the application's main class for the bootJar task.
bootRunMainClassName - Resolves the name of the application's main class for the bootRun task.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depend on it.
buildNeeded - Assembles and tests this project and all projects it depends on.
classes - Assembles main classes.
clean - Deletes the build directory.
generateGitProperties - Generate a git.properties file.
jar - Assembles a jar archive containing the main classes.
testClasses - Assembles test classes.
```

Afterwards, using Jhipster, we generated the Docker Compose manifest for all microservices and under the same root directory:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
34067f82de39	gateway	"bash -c /entrypoint..."	About a minute ago	Up About a minute	0.0.0.0:8080->8080/tcp	docker-compose_gateway_1
ebe291012782	mysql:8.0.27	"docker-entrypoint.s..."	About a minute ago	Up About a minute	3306/tcp, 33060/tcp	docker-compose_productorder-mysql_1
8a3e054db216	prom/alertmanager:v0.23.0	"/bin/alertmanager -..."	About a minute ago	Up About a minute	0.0.0.0:9093->9093/tcp	docker-compose_alertmanager_1
965da4ef83cb	invoice	"bash -c /entrypoint..."	About a minute ago	Up About a minute	5701/udp, 8082/tcp	docker-compose_invoice_1
4d6c46c9da00	prom/prometheus:v2.31.1	"/bin/prometheus --c..."	About a minute ago	Up About a minute	0.0.0.0:9090->9090/tcp	docker-compose_prometheus_1
c54cce215d8d	jhipster/jhipster-registry:v1.1.0	"bash -c /entrypoint..."	About a minute ago	Up About a minute	0.0.0.0:8761->8761/tcp	docker-compose_jhipster-registry_1
fbfe794c3773	productorder	"bash -c /entrypoint..."	About a minute ago	Up About a minute	5701/udp, 8081/tcp	docker-compose_productorder_1
1f56d24442cd	postgres:13.5	"docker-entrypoint.s..."	About a minute ago	Up About a minute	5432/tcp	docker-compose_invoice-postgresql_1
f86943de4d78	docker-compose_notification-mongodb-node	"docker-entrypoint.s..."	About a minute ago	Up About a minute	27017/tcp	docker-compose_notification-mongodb-node_1
529cb417117b	mysql:8.0.27	"docker-entrypoint.s..."	About a minute ago	Up About a minute	3306/tcp, 33060/tcp	docker-compose_gateway-mysql_1
aa8bc672c4ec	mongo:4.4.10	"docker-entrypoint.s..."	About a minute ago	Up About a minute	27017/tcp	docker-compose_notification-mongodb_1
7ab1ca124522	mongo:4.4.10	"docker-entrypoint.s..."	About a minute ago	Up About a minute	27017/tcp	notification-mongodb-config
0c09ada00ab1	grafana/grafana:8.2.4	"/run.sh"	About a minute ago	Up About a minute	0.0.0.0:3000->3000/tcp	docker-compose_grafana_1
750dc793454	notification	"bash -c /entrypoint..."	About a minute ago	Up About a minute	5701/udp, 8083/tcp	docker-compose_notification_1
7d017ed8b200	mongo:4.4.10	"docker-entrypoint.s..."	47 minutes ago	Up 47 minutes	127.0.0.1:27017->27017/tcp	docker_notification-mongodb_1

In order to meet changin demand, we scaled the invoice microservice up to 2 replicas, looking to jhipster registry on *localhost:8761* we can list each registered microservice with its replica:

The screenshot shows the JHipster Registry v7.1.0 interface. It has a dark header with navigation links: Home, Eureka, Configuration, and Administration. The main content area is titled 'JHipster Registry v7.1.0' and includes a 'Refresh now' button and a '60 sec.' timer.

System Status

Environment	JHipster-DataCenter
Data Center	JHipster-Environment
Current Time	2021-12-18T19:58:15 +0000
System Uptime	00:24
Below Renew Threshold	false

General Info

Total Available Memory	256mb
Current Memory Usage	124mb (48%)
Number of CPU	4
Instance Ip Address	172.19.0.13
Instance Status	UP

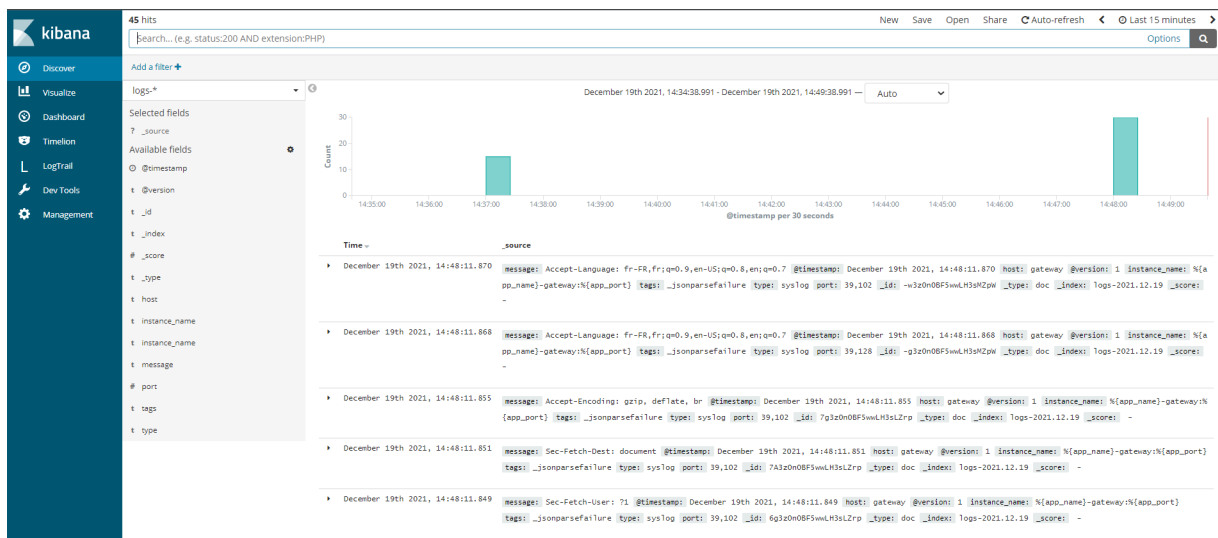
Instances Registered

App	Instance ID	Status
GATEWAY	gateway:ccb38916a81fbcc65d9211691781bc1	UP
INVOICE	invoice:a1e2b95b7287d34794c2af8c19b7087	UP
INVOICE	invoice:63c01029d767f748836a572f694ebf67	UP
JHIPSTER-REGISTRY	jhipsterRegistry:b26034f3ac263c206ffe88f9993498b	UP
PRODUCTORDER	productorder:c4aab4f9cb343a24f99d5906403e53d	UP

Health

ClientConfigServer	UP
ConfigServer	UP
DiscoveryComposite	UP
DiskSpace	UP
LivenessState	UP

In addition, for streaming logs and metrics we used the open-source application *Jhipster Console*. At first, we changed some settings on *application-dev.yml* (as we use the spring dev profile) by enabling *metrics.logs* and *logstash.logs* for all microservices including the gateway service. This modification will enable microservices to transfer logs and metrics to Jhipster Console running on port 5601. After re-deploying all microservices, we get the following metrics visualization:



Also logs of all deployed services:

kibana

- Discover
- Visualize
- Dashboard
- Timeline
- LogTrail**
- Dev Tools
- Management

Oldest event reached.

```

Dec 19 14:37:05 $(app_name)-gateway/$(app_port) - : GET / HTTP/1.1
Dec 19 14:37:06 $(app_name)-gateway/$(app_port) - : Host: localhost:5000
Dec 19 14:37:06 $(app_name)-gateway/$(app_port) - : Connection: keep-alive
Dec 19 14:37:06 $(app_name)-gateway/$(app_port) - : sec-ch-ua: " Not A Brand";v="99", "Chromium";v="96", "Google Chrome";v="96"
Dec 19 14:37:06 $(app_name)-gateway/$(app_port) - : sec-ch-ua-mobile: ?0
Dec 19 14:37:06 $(app_name)-gateway/$(app_port) - : sec-ch-ua-platform: "Windows"
Dec 19 14:37:06 $(app_name)-gateway/$(app_port) - : Upgrade-Insecure-Requests: 1
Dec 19 14:37:06 $(app_name)-gateway/$(app_port) - : User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36
Dec 19 14:37:06 $(app_name)-gateway/$(app_port) - : Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Dec 19 14:37:06 $(app_name)-gateway/$(app_port) - : Sec-Fetch-Site: none
Dec 19 14:37:06 $(app_name)-gateway/$(app_port) - : Sec-Fetch-Mode: navigate
Dec 19 14:37:06 $(app_name)-gateway/$(app_port) - : Sec-Fetch-User: ?1
Dec 19 14:37:06 $(app_name)-gateway/$(app_port) - : Sec-Fetch-Dest: document
Dec 19 14:37:06 $(app_name)-gateway/$(app_port) - : Accept-Encoding: gzip, deflate, br
Dec 19 14:37:06 $(app_name)-gateway/$(app_port) - : Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : GET / HTTP/1.1
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Host: localhost:5000
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Host: localhost:5000
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Connection: keep-alive
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : sec-ch-ua: " Not A Brand";v="99", "Chromium";v="96", "Google Chrome";v="96"
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : sec-ch-ua-mobile: ?0
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : sec-ch-ua-platform: "Windows"
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Upgrade-Insecure-Requests: 1
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : sec-ch-ua: " Not A Brand";v="99", "Chromium";v="96", "Google Chrome";v="96"
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : sec-ch-ua-mobile: ?0
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Sec-Fetch-Site: none
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Sec-Fetch-Mode: navigate
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Upgrade-Insecure-Requests: 1
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Sec-Fetch-User: ?1
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Sec-Fetch-Dest: document
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Accept-Encoding: gzip, deflate, br
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Sec-Fetch-Site: none
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Sec-Fetch-Mode: navigate
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Sec-Fetch-User: ?1
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Sec-Fetch-Dest: document
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Accept-Encoding: gzip, deflate, br
Dec 19 14:48:11 $(app_name)-gateway/$(app_port) - : Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7

```

CHAPTER 2

Deployment of online-store with Kubernetes On GCP

In this section, we deployed our microservices on GCP using Kubernetes as an orchestrator. The process of deploying with kubernetes is one of the hardest option, but Jhipster simplifies this process by generating the Kubernetes manifests for each microservices, which are basically files that describe how Kubernetes will deploy and manage our services. These files point to public docker images, so we pushed each microservice image into our common docker registry named *oncobe*:

oncobe / productorder Updated 6 days ago	Not Scanned	☆ 0	↓ 5	Public
oncobe / notification Updated 6 days ago	Not Scanned	☆ 0	↓ 10	Public
oncobe / invoice Updated 6 days ago	Not Scanned	☆ 0	↓ 7	Public
oncobe / gateway Updated 6 days ago	Not Scanned	☆ 0	↓ 10	Public

After generating all manifests in order to run our microservices-based application, we created a Kubernetes cluster in GCP using the installed SDK for gcp.

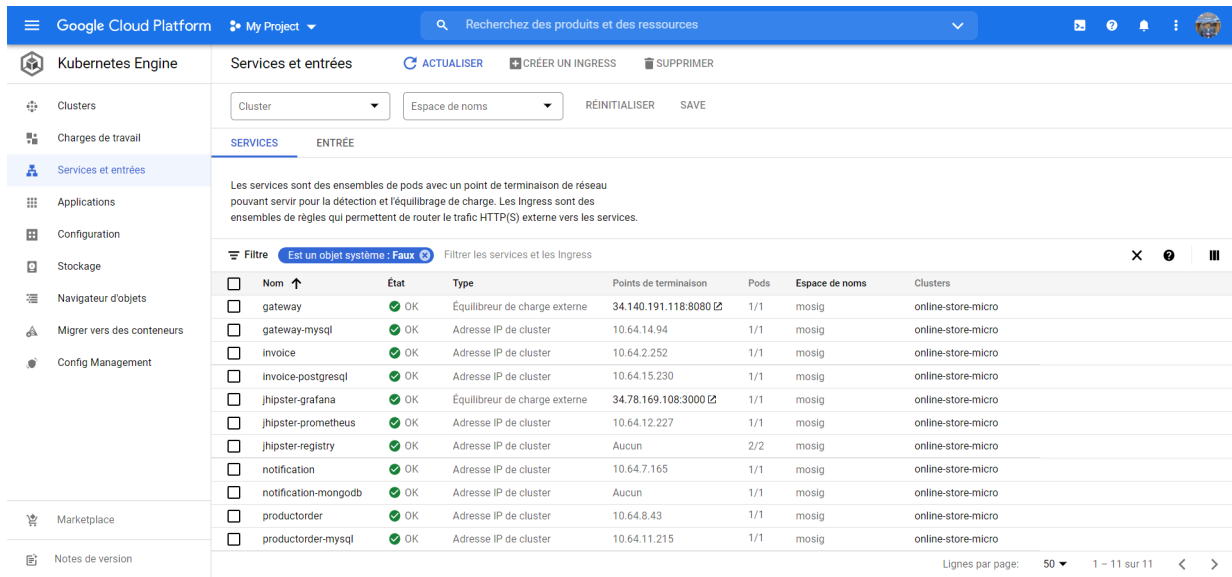
Also we used the medium machine type *-machine-type n1-standard-2*. After running this command:

gcloud container clusters create online-store-micro --machine-type n1-standard-2

We got the following cluster up and running:

NAME	LOCATION	MASTER_VERSION	MASTER_IP	MACHINE_TYPE	NODE_VERSION	NUM_NODES	STATUS
online-store-micro	europe-west1-b	1.21.5-gke.1302	35.241.207.215	n1-standard-2	1.21.5-gke.1302	3	RUNNING

We can check now all running status of pods after deploying microservices on google cloud platform :



Google Cloud Platform My Project Recherchez des produits et des ressources

Kubernetes Engine Services et entrées ACTUALISER CRÉER UN INGRESS SUPPRIMER

Cluster Espace de noms RÉINITIALISER SAVE

SERVICES ENTRÉE

Les services sont des ensembles de pods avec un point de terminaison de réseau pouvant servir pour la détection et l'équilibrage de charge. Les Ingress sont des ensembles de règles qui permettent de router le trafic HTTP(S) externe vers les services.

Filtre Est un objet système : Faux Filtrer les services et les Ingress

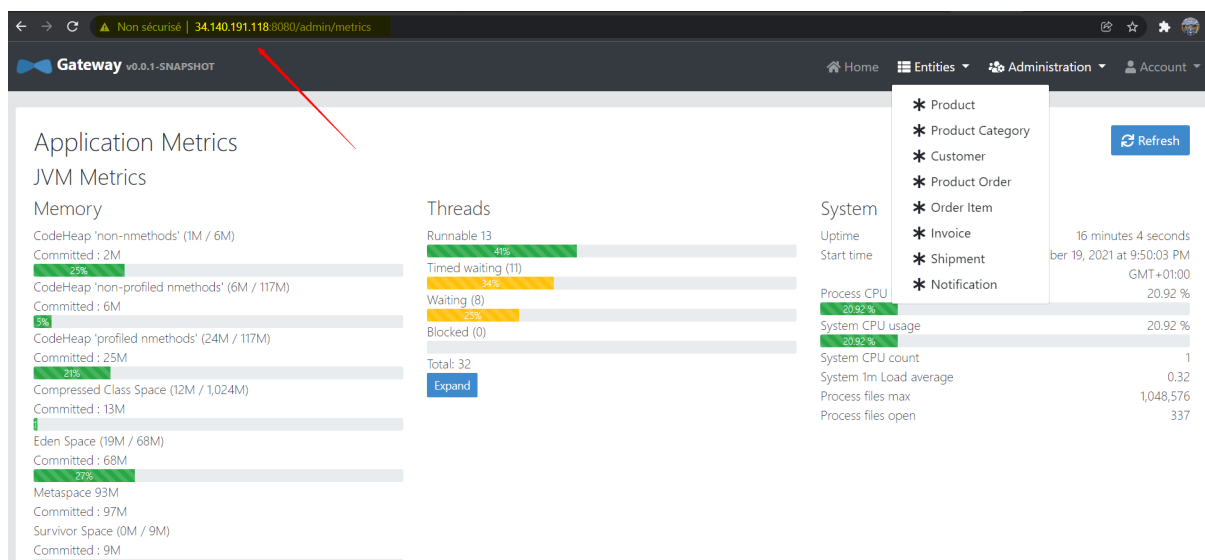
Nom	État	Type	Points de terminaison	Pods	Espace de noms	Clusters
gateway	OK	Équilibreur de charge externe	34.140.191.118:8080	1/1	mosig	online-store-micro
gateway-mysql	OK	Adresse IP de cluster	10.64.14.94	1/1	mosig	online-store-micro
invoice	OK	Adresse IP de cluster	10.64.2.252	1/1	mosig	online-store-micro
invoice-postgresql	OK	Adresse IP de cluster	10.64.15.230	1/1	mosig	online-store-micro
jhipster-grafana	OK	Équilibreur de charge externe	34.78.169.108:3000	1/1	mosig	online-store-micro
jhipster-prometheus	OK	Adresse IP de cluster	10.64.12.227	1/1	mosig	online-store-micro
jhipster-registry	OK	Adresse IP de cluster	Aucun	2/2	mosig	online-store-micro
notification	OK	Adresse IP de cluster	10.64.7.165	1/1	mosig	online-store-micro
notification-mongodb	OK	Adresse IP de cluster	Aucun	1/1	mosig	online-store-micro
productorder	OK	Adresse IP de cluster	10.64.8.43	1/1	mosig	online-store-micro
productorder-mysql	OK	Adresse IP de cluster	10.64.11.215	1/1	mosig	online-store-micro

Lignes par page: 50 1 - 11 sur 11

In addition, we can get the accessible external IP of our application store:

```
$ kubectl get svc gateway -n mosig
NAME      TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
gateway   LoadBalancer  10.64.4.239   34.140.191.118  8080:30088/TCP   15m
```

Also we can use this IP in order to access to the application through the gateway service on 8080, and display some metrics with a nice dashboard



Moreover, one of the most interesting advantages with microservices-based application is the scalability of microservices (horizontal and vertical), now we add the scalability option to a specific microservices instead of doing it to the whole application (case of monolithic). For instance lets scale up the service of notification up to two replicas using the following commande :

```
$ kubectl scale deployment/notification --replicas=2 -n mosig
deployment.apps/notification scaled
```


As we see under the *Services & Ingress* we got two pods for notifications microservices:

<input type="checkbox"/>	Nom ↑	État	Type	Points de terminaison	Pods	Espace de noms	Clusters
<input type="checkbox"/>	gateway	✓ OK	Équilibreur de charge externe	34.140.191.118:8080 ↗	1/1	mosig	online-store-micro
<input type="checkbox"/>	gateway-mysql	✓ OK	Adresse IP de cluster	10.64.14.94	1/1	mosig	online-store-micro
<input type="checkbox"/>	invoice	✓ OK	Adresse IP de cluster	10.64.2.252	1/1	mosig	online-store-micro
<input type="checkbox"/>	invoice-postgresql	✓ OK	Adresse IP de cluster	10.64.15.230	1/1	mosig	online-store-micro
<input type="checkbox"/>	jhipster-grafana	✓ OK	Équilibreur de charge externe	34.78.169.108:3000 ↗	1/1	mosig	online-store-micro
<input type="checkbox"/>	jhipster-prometheus	✓ OK	Adresse IP de cluster	10.64.12.227	1/1	mosig	online-store-micro
<input type="checkbox"/>	jhipster-registry	✓ OK	Adresse IP de cluster	Aucun	2/2	mosig	online-store-micro
<input type="checkbox"/>	notification	✓ OK	Adresse IP de cluster	10.64.7.165	1/2	mosig	online-store-micro
<input type="checkbox"/>	notification-mongodb	✓ OK	Adresse IP de cluster	Aucun	1/1	mosig	online-store-micro

And by accessing to the service registry on 32.240.83.123:8761 we can see that we have two services of notification which are registered:

The screenshot shows the Eureka Registry v7.1.0 web interface. The browser address bar shows the URL `35.240.83.123:8761/registry/applications`. The page title is "Application Instances". On the right, there are buttons for "Refresh now" and "disabled". The main content area lists four application instances: GATEWAY (1/1), INVOICE (1/1), JHIPSTER-REGISTRY (2/2), and NOTIFICATION (2/2). Below this, there is a section titled "Instances" which displays a table with two columns: "ID" and "Status". The "ID" column shows the instance ID `gateway:8b33ac37fb43ee6b369d3f0a327d8b62`. The "Status" column shows the instance is "UP" and lists various metadata tags: `git-commit: 9ba8152`, `version: 0.0.1-SNAPSHOT`, `management.port: 8080`, `profile: prod`, `zone: primary`, and `git-branch: master`.

CHAPTER 3 Load Injection tests with Gatling

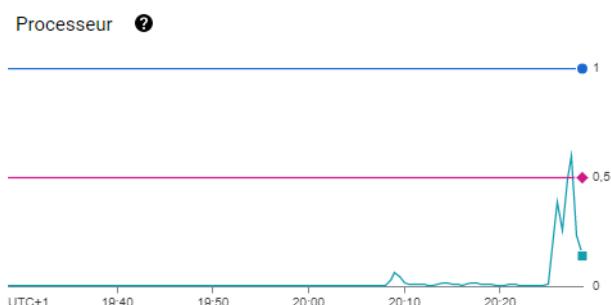
In this section we will test the scalability property of our microservice *notification*, first lets enable the autoscaling policy using the following commande:

```
C:\Users\Ismael\Desktop\trainee\dev\jhipster\tp_cp_online_store\microservices_app>kubectl autoscale deployment notification --cpu-percent=50 --min=1 --max=3 -n mosig
horizontalpodautoscaler.autoscaling/notification autoscaled
```

Here we have specified the threshold limit of CPU percentage in which the pod will be replicated and handle the high load of requests on notification microservices. We have specified also the maximum replicas allowed which is 3.

As you see in the following screenshot, when we launched *Gatling*, the notification microservices starts receiving a lot of requests, which increased the CPU consumption.

Notice that when the CPU consumption was around 50% it starts decreasing gradually, which is a result of scaling up the pod:



Here we got the maximum replica (max=3) after launching *Gatling*:

Pods gérés

Révision	Nom	État	Redémarrages	Créées le ↑
1	notification-b5bff9659-6m9hp	Running	1	25 déc. 2021, 15:58:48
1	notification-b5bff9659-vnd8f	Unschedulable	0	25 déc. 2021, 20:26:19
1	notification-b5bff9659-4wgkg	Unschedulable	0	25 déc. 2021, 20:26:19

Here is a summary of Gatling report (you can find the full report in our git repository), we can notice that after a while, responses time start to be relatively slow (> 1200 ms), and that because of the number of users (200 users at a time), so maybe we have to increase the maximum replica of our autoscaling policy.

