Cloud Computing lab

Microservices, Kubernetes and Google Cloud

Dorian Goepp

2022-12-16

Introduction

This lab was supervised by Didier Donsez. The related instructions are in tutorial_en, found from the class's website.

Monolithic application (`./monolith/README.md`)

Creating a monolith (`monolith.md`)

Creating the app

For the sake of brevity, the outputs of the `cloc` command bellow are summarised.

```
~/Code/mastering-microservices/online-store$ cloc src/ webpack/
```

files	blank	comment	code
471	3034	1645	3232

```
~/Code/mastering-microservices/online-store$ cloc src/ webpack/ *.json *.xml
```

Formatted output of this command:

files	blank	comment	code
479	3035	1653	48497

Estimate the cost and the time for coding the basic application:

Cocomo calculator: Based on 48497 being the total number of lines of code, and on a total employer cost of 4800 €/month/employee, the total cost for this piece of software could be estimated to be **1 007 788 €, for 21 months of time**. Stackoverflow jobs was shut down on 2022, April 1st so I could not use it for comparison.

We are very fortunate not to have to write this on our own.

Generate the entities and the relationships

```
~/Code/mastering-microservices/online-store$ cloc src/ webpack/
```

Formatted output of this command:

files	blank	comment	code
684	5696	3093	37915

```
~/Code/mastering-microservices/online-store$ cloc src/ webpack/ *.json *.xml
```

Formatted output of this command:

files	blank	comment	code
692	5697	3101	63181

Estimate the new cost and the new time for coding the basic application : **1 348 011 € and 22 months**.

to explore the API. We see many more operations, related to the new database tables that were introduced.

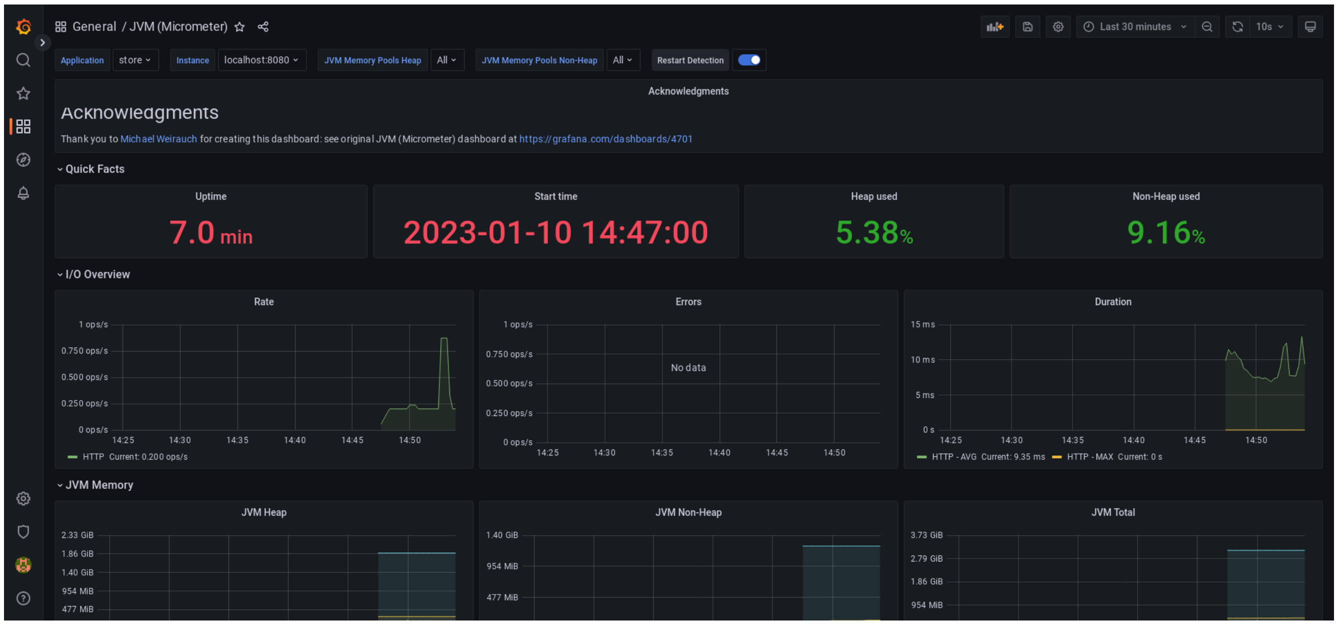
Deploy locally with Docker (`deploy_docker.md`)

Deploy locally the application with Docker

When we stop the PostgreSQL sever, the application still responds but we are logged out and it is impossible to login again. As soon as the PostgreSQL server is restarted, we can resume using the web application (provided we reload the page).

Monitor the application (`monitor.md`)

Done. It required to comment the `ports` sections of the docker-compose file `monitoring.yml`. Outcome:


Acknowledgments

Thank you to Michael Weirauch for creating this dashboard: see original JVM (Micrometer) dashboard at https://grafana.com/dashboards/4701

Microservices-based application (`./miscroservices/README.md`)

Generate the microservices architecture (`microservices.md`)

Browse the application and service registry

```
open http://localhost:8080
open http://localhost:8761
```

What can you see ?

In the API browser, only the registry's API shows. However, the gateway administration page shows all services. They are online but I do not know how to interact with them. We probably need to configure the gateway to interface with the services' APIs.

The registry also lists all services.

What happens if one microservice is stopped (failure simulation . . .)?

I stopped the ‘invoice’ service. The related entries in the gateway and registry webpages is removed upon refresh. There is nothing yet ensuring resilience to failures.

Generation of the code of the microservice `notification`

Thanks to the modifications here, the gateway’s API interface allows to browse all services’s API. Nothing new in the registry.

Build the Docker images

The new syntax for Gradle 8.0 to generate docker images (OCI) is:

```
./gradlew -Pprod bootJar assemble -x test
```

Generation of the docker-compose files

The version of JHipster on my machine (v7.9.3) does not have the expected gateway

```
? Which *type* of gateway would you like to use? JHipster gateway based on Netflix Zuul
```

but instead offers `JHipster gateway ased on Spring Cloud Gateway` (only option). Likewise

```
? Do you want to setup monitoring for your applications ? Yes, for logs and metrics
    with the JHipster Console (based on ELK and Zipkin)
```

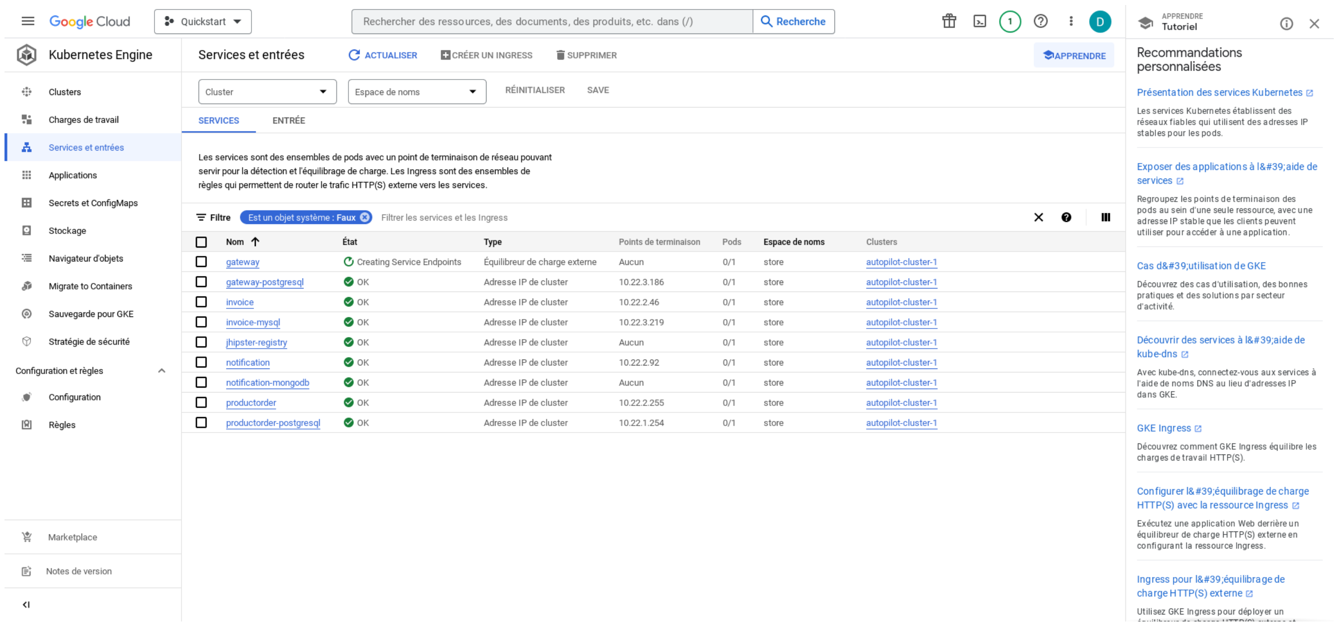
is replaced with `Yes, for metrics only with Prometheus`.

Deploy in production the microservices on GCP with Kubernetes (`deploy_k8n.md`)

Diverging answer to the questions in `jhipster kubernetes`:

```
? What should we use for the base Docker repository name? dogoepp
```

Here is the deployed software in Kubernetes Engine:



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.

Nom	État	Type	Points de terminaison	Pods	Espace de noms	Clusters
gateway	Creating Service Endpoints	Équilibreur de charge externe	Aucun	0/1	store	autopilot-cluster-1
gateway-postgresql	OK	Adresse IP de cluster	10.22.3.186	0/1	store	autopilot-cluster-1
invoice	OK	Adresse IP de cluster	10.22.2.46	0/1	store	autopilot-cluster-1
invoice-mysql	OK	Adresse IP de cluster	10.22.3.219	0/1	store	autopilot-cluster-1
jhipster-registry	OK	Adresse IP de cluster	Aucun	0/1	store	autopilot-cluster-1
notification	OK	Adresse IP de cluster	10.22.2.92	0/1	store	autopilot-cluster-1
notification-mongodb	OK	Adresse IP de cluster	Aucun	0/1	store	autopilot-cluster-1
productorder	OK	Adresse IP de cluster	10.22.2.255	0/1	store	autopilot-cluster-1
productorder-postgresql	OK	Adresse IP de cluster	10.22.1.254	0/1	store	autopilot-cluster-1

Recommandations personnalisées

Présentation des services Kubernetes

Les services Kubernetes établissent des réseaux fiables qui utilisent des adresses IP stables pour les pods.

Exposer des applications à l&#39;aide de services

Regroupez les points de terminaison des pods au sein d'une seule ressource, avec une adresse IP stable que les clients peuvent utiliser pour accéder à une application.

Cas d&#39;utilisation de GKE

Découvrez des cas d'utilisation, des bonnes pratiques et des solutions par secteur d'activité.

Découvrir des services à l&#39;aide de kube-dns

Avec kube-dns, connectez-vous aux services à l'aide de noms DNS au lieu d'adresses IP dans GKE.

GKE Ingress

Découvrez comment GKE Ingress équilibre les charges de travail HTTP(S).

Configurer l&#39;équilibrage de charge HTTP(S) avec la ressource Ingress

Exécutez une application Web derrière un équilibreur de charge HTTP(S) externe en configurant la ressource Ingress.

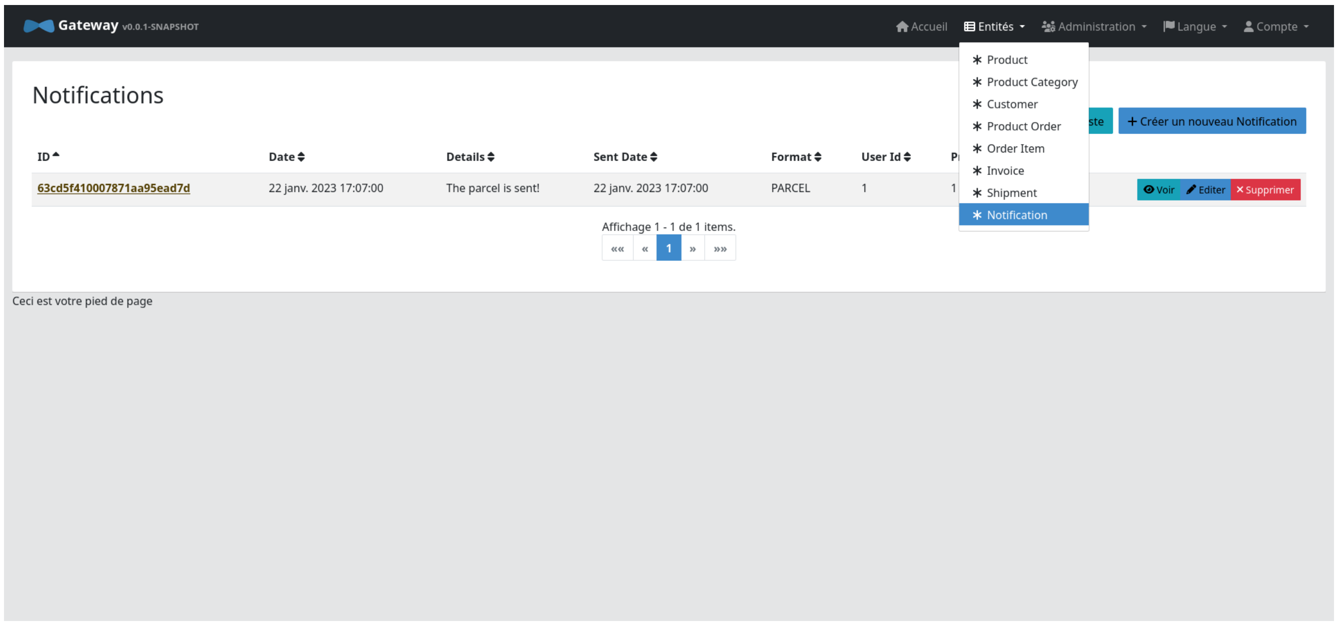
Ingress pour l&#39;équilibrage de charge HTTP(S) externe

Utilisez GKE Ingress pour déployer un

I took the care to change the default password for the gateway, as the application has been made available to the whole internet with a default password that is obvious to guess.

```
$ kubectl get svc gateway -n store
NAME      TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
gateway   LoadBalancer   10.22.3.158   34.159.47.62   8080:31058/TCP   101m
```

This done, I could access the gateway at the above IP address and port 8080. The screenshot below shows the data entities that could be created, read, modified and deleted from the gateway, although they were hosted and made available through separate micro-services.



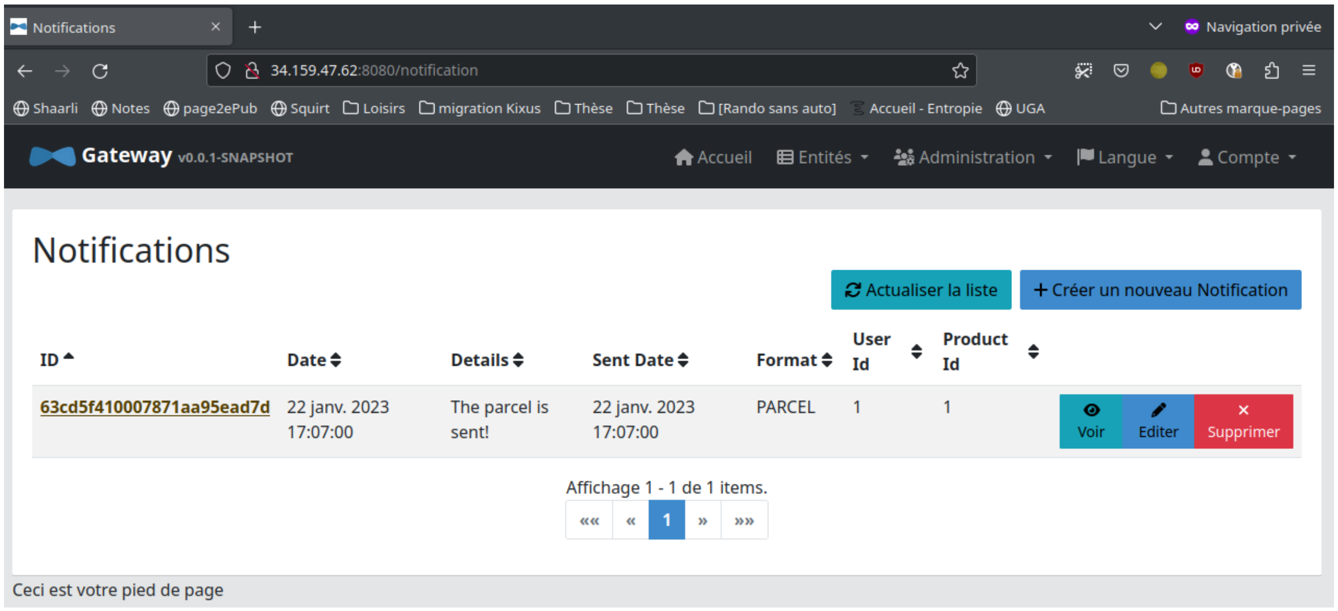
Notifications

ID	Date	Details	Sent Date	Format	User Id	P
63cd5f410007871aa95ead7d	22 janv. 2023 17:07:00	The parcel is sent!	22 janv. 2023 17:07:00	PARCEL	1	1

Affichage 1 - 1 de 1 items.

Ceci est votre pied de page

Here is a detailed view of the entry to the `notifications` table I added for testing, also showing the **IP address of the gateway** in the address bar.



Notifications

ID	Date	Details	Sent Date	Format	User Id	Product Id
63cd5f410007871aa95ead7d	22 janv. 2023 17:07:00	The parcel is sent!	22 janv. 2023 17:07:00	PARCEL	1	1

Affichage 1 - 1 de 1 items.

Ceci est votre pied de page

We are asked to get the IP address of the console, but I do not have the related service running. These are all the services I have:

```
$ kubectl get svc -n store
NAME                       TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)          AGE
gateway                    LoadBalancer   10.22.3.158    34.159.47.62   8080:31058/TCP   102m
gateway-postgresql         ClusterIP      10.22.3.186    <none>         5432/TCP         102m
invoice                    ClusterIP      10.22.2.46     <none>         8082/TCP         102m
invoice-mysql              ClusterIP      10.22.3.219    <none>         3306/TCP         102m
jhipster-registry          ClusterIP      None           <none>         8761/TCP         103m
notification               ClusterIP      10.22.2.92     <none>         8083/TCP         102m
```

```
notification-mongodb      ClusterIP     None          <none>        27017/TCP       102m
productorder              ClusterIP     10.22.2.255   <none>        8081/TCP        102m
productorder-postgresql   ClusterIP     10.22.1.254   <none>        5432/TCP        102m
```

It is probably because my Jhipster did not offer the ELK option on deployment's monitoring. It instead offered Prometheus and Grafana. But the latter never got started. It seems that the `./kubectl-apply.sh` script did not manage to start it. It would display `Waiting for the custom resource prometheus operator to get initialised` in a loop. Here is a more detailed error message displayed as part of a long stream of messages:

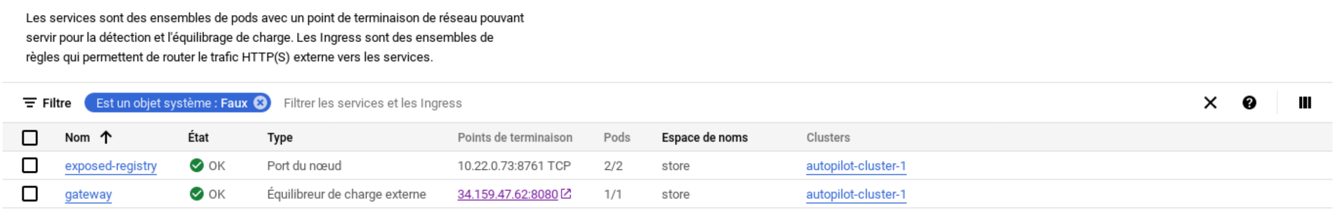
```
error: resource mapping not found for name: "gateway-app" namespace: "store" from "gateway-k8s/gateway-pro
ensure CRDs are installed first
```

Exposing the service registry:

```
$ kubectl get svc exposed-registry -n $APPID
NAME               TYPE       CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
exposed-registry   NodePort   10.22.0.73   <none>        8761:30790/TCP   8s
```

Once this was done, I took good care to stop or destroy all relevant resource in the Google Cloud console. I found that the most reliable way to ensure nothing will incur later costs was through the billing panel ("Facturation" in French). There, I disallowed billing of all Google Cloud projects. This prevents anything billable to run.

Creating more replicas is illustrated with the screenshot below:



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
exposed-registry	✓ OK	Port du nœud	10.22.0.73:8761 TCP	2/2	store	autopilot-cluster-1
gateway	✓ OK	Équilibreur de charge externe	34.159.47.62:8080	1/1	store	autopilot-cluster-1

Note that only two services are listed here. It is just because I forgot to take the screenshot before closing all the other services. I took this one in the nick of time, as I was already closing everything down.

Conclusion

This has been an epic journey discovering how modern web-development tools are used. It has been rewarding, and challenging.

Beyond what was taught in the lectures, this lab session taught me about the complexity of the tools related to micro-services and general cloud development. I also come with the impression that there is a lot of overhead, computational and memory cost, coming with these tools, probably as a cost of their powers. I hope and assume that these are offset by the features and developer support they provide when developing truly complex software, unlike the toy code we were using here.

As an illustration to the above, the whole folder for the tutorial and both monolithic and microservice versions of the application takes 5.1 GB of space on my computer.

As a personal note of improvement, I wish I had done all the development of the app within a container. I would have been more confident in the environment in which everything was being built. Besides, it might have spared me some of the issues faced above, especially those I could not fix.