POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Computer Engineering

Master degree's Thesis

Implementation of a docker containers orchestration solution

Company: KelkooGroup (France)

Supervisor

prof. Morisio Maurizio

Candidate

Nguetsop Roy Paulin

Academic year 2020-2021

Contents

1 Introduction

The final project represents the opportunity to use all the notions learned in a practical context. I started mine in march 2020 in a field more and more important: microservices. A field which get the interest of many people due to all its applications.

2 Report

2.1 Problematic

For a long time, virtual machines have been considered by companies as the best way to deploy their different services . However, given that they require a lot of resources and maintenance, a more efficient solution has been considered and today several companies have migrated their infrastructures in the cloud. Technologies such as Docker and Kubernetes are more and more used to host their services.

That is why KelkooGroup has decided to hold this internship. They want to explore how they could benefit from these technologies to optimize their deployment procedures.

2.2 Internship's environment

The internship was supposed to be held in the KelkooGroup offices in Echirolles but due to the Covid-19 crisis, I did it remotely from home. I was not part of a specific team. All the people working on the project were from different teams.

2.3 Problematic

2.3.1 Context of the project

The current infrastructure pf KelkooGroup uses 2 data centers with a total of 300 servers and 1000 virtual machines. KelkooGroup would like to set up Docker containers on development, pre-production and production environments.. Kubernetes is the core, the orchestrator who helps us to manage all those Docker containers.

2.3.2 Problems and Objectives

It was very important to understand the technologies currently in use, to study those used to build the new one in order to decide if, when and how this change will impact KelkooGroup. KelkooGroup would like to increase its global efficiency thanks to microservices and containers' orchestration.

The goals are:

- a better efficiency through an easier evolution and maintenance.
- a better operability through scalability, isolation and fault tolerance.
- Infrastructure costs lower thanks to an optimized hardware and an hybrid infrastructure.

2.4 Methodology

For this project, an Agile methodology has been adopted. A team gathering people from different background has been built. a 2 weeks workload was decided followed by a demonstration. Everyone gave his/her view and according to

that, the next 2 weeks goals were defined. Moreover, a stand up was scheduled everyday to check the progression of the student.

3 Solution

3.1 Application used

We chose ImageflyResizer as test application. It is an application built by KelkooGroup, coded in php which downloads an image and resize it. It is used in ecommerce for products images.

3.2 Existing solution

For its current deployment, KelkooGroup uses a few proprietary technologies.

The procedure is as described:

- The application L'application est packagée sous forme de rpm + rôle Puppet pour la configuration.
- Geppetto allows to manage the configuration of nodes.
- Release Automation allows to automate configuration changes linked to releases.
- Puppet allows to deploy the application with the desired configuration on a server.
- Ansible allows to orchestrate the deployment.

3.3 New solution

The current solution is taking too much resources and is asking too much maintenance due to all the virtual machines used. The new one is based on docker containers which are lighter. We want to "dockerize" the application (which means create its docker image). To use that solution on a higher scale we used Kubernetes which is an orchestrator. This way, we could deploy our applications in a Kubernetes cluster which offers a lot of tools, like configmap, for the configuration. This solution should also allow to view our application logs using an ElasticSeartch server and Kibana as front end.

3.4 Software stack

3.4.1 Docker

Docker is a containerization tool to automate the deployment of programs through OS level virtualization. A container is like a virtual machine but it is lighter and fast because it does not go through the virtualization of all the software stack.

With Docker the distribution of our program in several machines as well as the updates are easier.

3.4.2 Kubernetes

Most of the time, we must manage a great number of containers, their creation, interaction, destruction and the exposition of our applications to the outside.

Doing it manually would be very challenging and that is where we need Kubernetes. In our project, we used Kubernetes to manage different nodes where our applications were running.

3.4.3 Rancher

Rancher is a software based on Kubernetes. It allows to resolve in an effective way operational and security problems caused by the management of several Kubernetes clusters.It provides tools to manage and monitor our different workloads.

4 Roadmap

4.1 Dockerisation

In order to run our application in containers, it is necessary to create first a Docker image of our application and then, create as many containers as we want all following the model defined by the image. So the dockerisation process consists in taking as input an application and after a certain number of operations, obtaining as output a docker image. From that image, any container created will host our application ensuring the same functionalities.

The "dockerisation" can be summarized in 2 steps :

- Building the Dockerfile : The Dockerfile is a text document where we put a set of necessary commands to set up our image. It is like a recipe for image.
- saving the image in a registry.

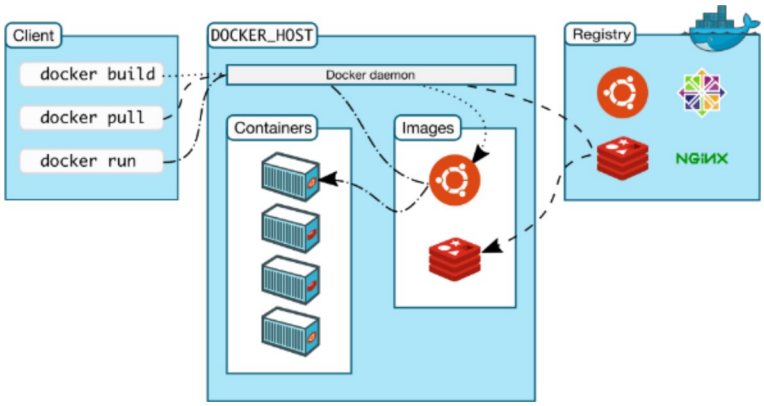

Figure 1: docker workflow

4.2 Deployment in a rancher cluster

Once the application is available in a registry, we need to deploy it in a cluster and access it from the outside.

For that we needed to:

- Save our register in order that we can access it from everywhere.
- From yaml files, create deployment, service and ingress which are the main resources used to set up our application.

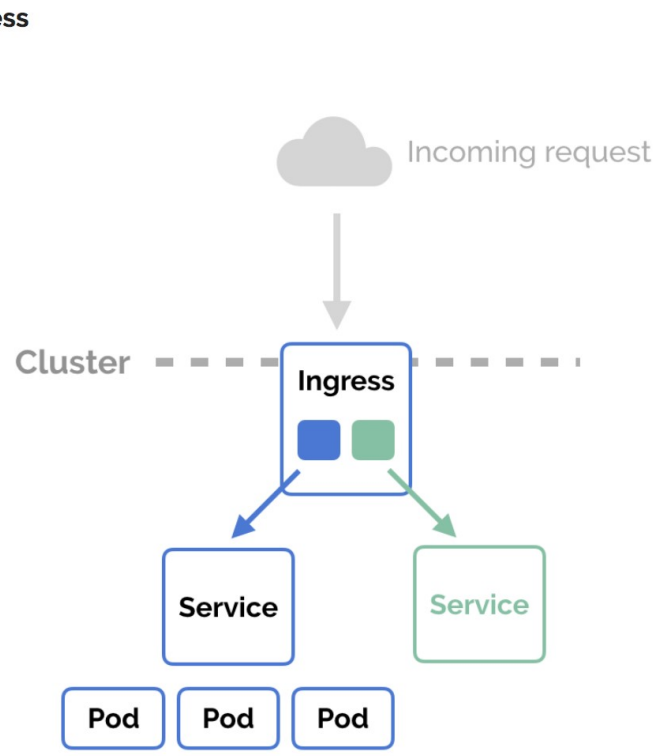

Figure 2: Cluster Components

- deployment: It is a resource used to describe our application by precising images, number of pods, port used etc...

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: imageflyresizer-deployment
  labels:
    app: imageresizer
spec:
  replicas: 3
  selector:
    matchLabels:
      app: imageresizer
  template:
    metadata:
      labels:
        app: imageresizer
    spec:
      containers:
      - name: imageflyresizer
        image: kelkoo20/imageflyresizer
        ports:
        - containerPort: 80
```

- Service: It is an abstraction which defines a logical set of pods and rules on how to access them. The motivation of its usage come from the fact the pods present at a certain moment could be different from those present at another one. That create a persistence issue due to the changing of the ip address.

```
apiVersion: v1
kind: Service
metadata:
  name: imageflyresizer-service
  labels:
    app: imageresizer
spec:
  selector:
    app: imageresizer
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: NodePort
```

This way we have an unique ip address to refer to when we want to access the application.

- Ingress: It an object used to access our services from outside the cluster.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-imageresizer
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: imageflyresizer-service
          servicePort: 80
    host: imageresizer.name.domain
```

All the requests to our application will be sent using the host name specified on the yaml above, and those requests will be redirected to the specified service.

4.3 Logging

Logs are important parts of applications. They let us know about the events occurring and which, very often, help us to determine the application's state.

Collecting logs in rancher is done through Fluentd which extracts stdout/stderr logs from from each container in files located at /var/log/containers.

this figure summarizes the procedure:

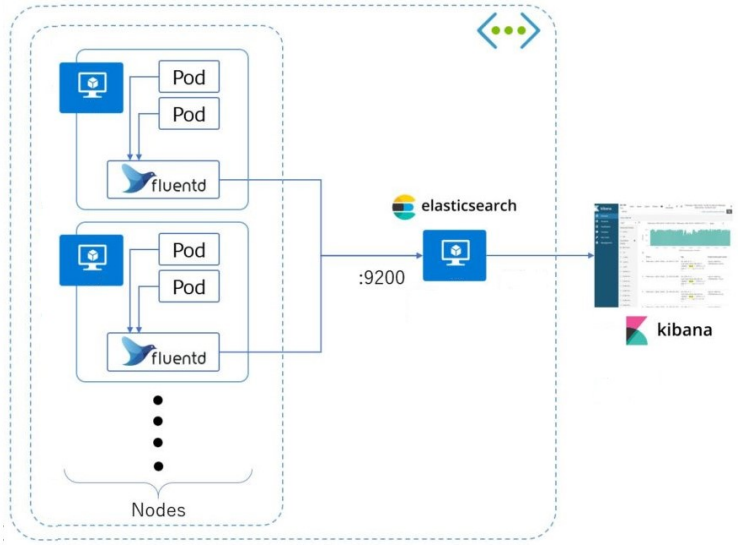

Figure 3: Rancher Logging

The final goal is accessing these logs from kibana which is used as front end. an ElasticSearch server has been configured to receive those logs and send them to Kibana. Logs are differentiated from each other using indexes.

4.3.1 Sidecar

A sidecar container is a container which runs in the same pod as the application, sharing the same resources, and improving how the application works by adding other functionalities. In this case, it has been used to manage the logging.

We have created a sidecar container, from the busybox image, which shares a file (with our application) which it will read logs from and will export them in other to be used by rancher. The application will write the logs in the shared file.

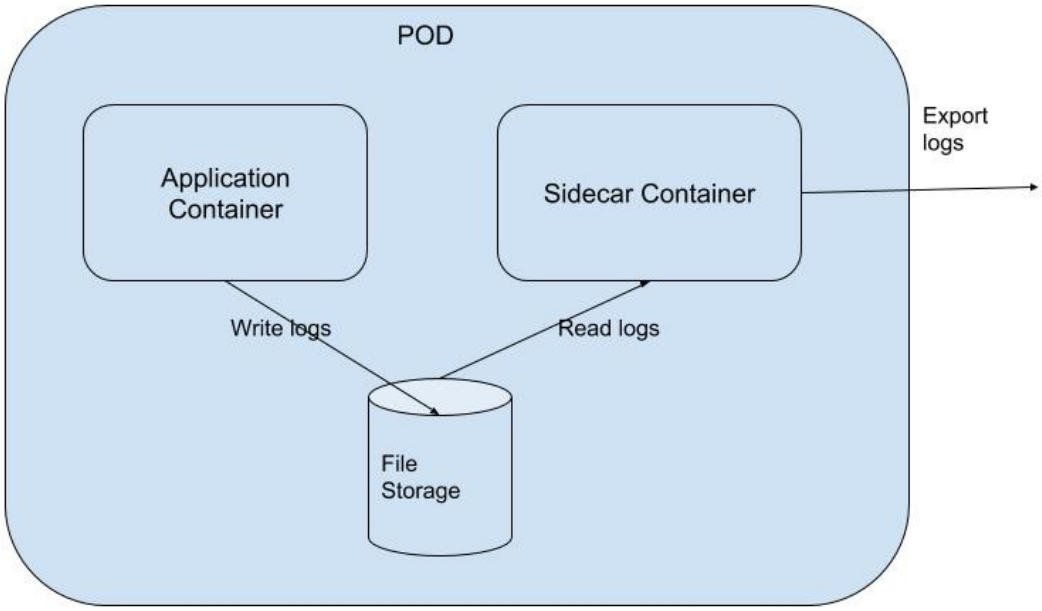

Figure 4: logging with sidecar

4.4 Helm

Deploy applications in kubernetes can by complex. Setting up an application can imply creating several pods, services, deployments, and so writing yaml files for each one of them. Helm is a package manager providing the same functionalities as Debian's apt and Python's pip.

It is very important first of all to understand 3 key concepts of helm:

- CHART: It is a package helm. It contains all resources' definitions (deployment, services, ingress, etc...) necessary to execute an application or service inside the cluster.

- REPOSITORY: It is a simple place where charts can be collected and shared.

- RELEASE: It is a chart instance running in the cluster. A chart can be installed several times in the same cluster and at each installation, a new release is created.

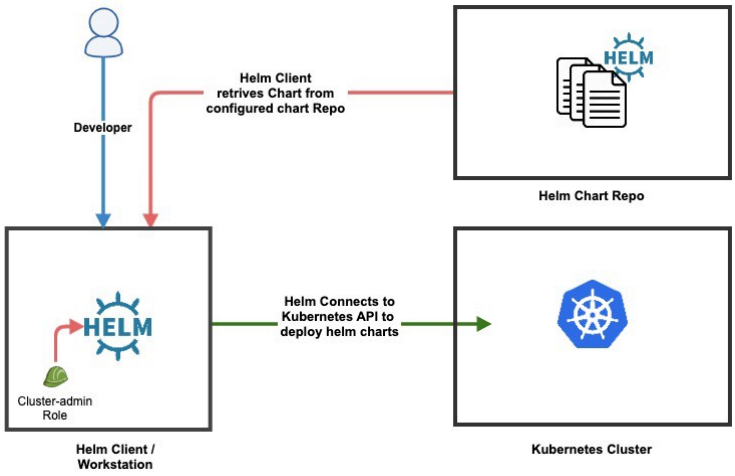

Figure 5: Helm's general structure

4.5 ConfigMap

A configmap lets you separate your application code from the configuration. That way, you can easily change configuration depending on the environment.

Firstly, create your yaml file to define your configurations:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
  namespace: default
data:
   MAX_IMAGE_SIZE_IN_PIXELS: '6'
```

Precise the configmap name as well as a set of key - value pairs indicating the configuration. You will just need to refer to your configuration file, in the deployment yaml, in order that newly created pods take into account the new configuration.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: imageflyresizer-deployment2
  labels:
    app: imageflyresizer
spec:
  replicas: 3
  selector:
    matchLabels:
      app: imageflyresizer
  template:
    metadata:
      labels:
        app: imageflyresizer
    spec:
      containers:
      - name: imageflyresizer
        image: kelkooprivate/kelkoo20:imageflyresizer
        envFrom:
        - configMapRef:
            name: app-config
        ports:
        - containerPort: 80
```

This way, when the container is created, it will use the key-value pairs from the configuration files as environment variables.

4.6 Security and roles

4.6.1 Authentication

Kubernetes does not know the concept of normal user. So, normal users should be managed by external services.

Rancher allows to go beyond that limit by offering the capability to execute a centralized user authentication. This feature let the users authenticate themselves on each of their kubernetes clusters with the same credentials.

Rancher provide us 2 types of authentication: local and external.

- Local authentication: It is the default one where Rancher itself save (local) users.

- External authentication : The one we used. KelkooGroup already has a server where all the employees data are saved. The goal is, when an employee wants to log in, Rancher will go to check his/her credentials on the company ldap server.

Rancher include several external authentication services such as openLDAP, Github, Shibboleth...and in general external authentication procedure follows these steps:

Figure 6: rancher log in as local admin and external authentication configuration

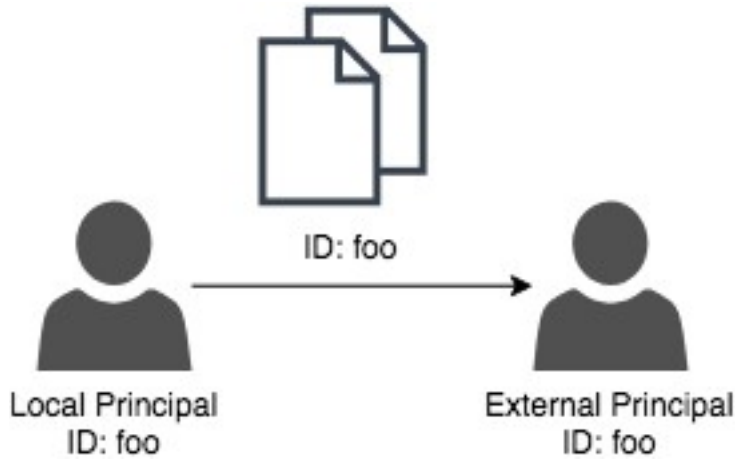

Figure 7: association of external and local users both sharing the same ID

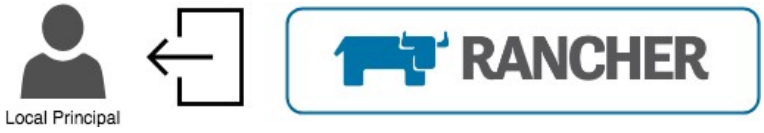

Figure 8: Automatic log out of the local user after configuration

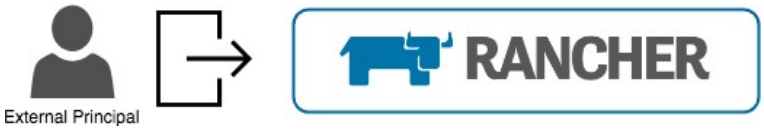

Figure 9: Automatic log in of the external user

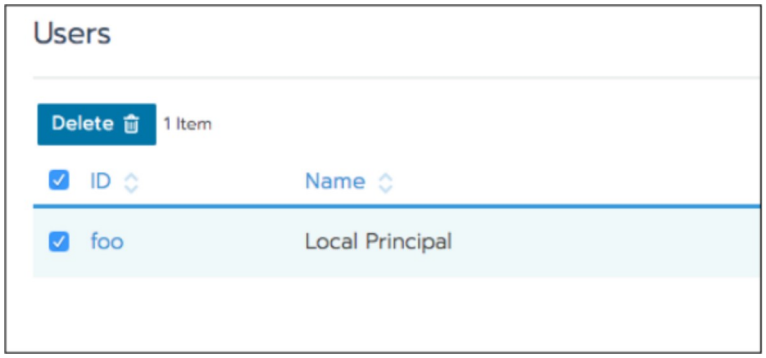

Figure 10: At the end only the external one remains

4.6.2 Autorisation

Once a user has been authenticated, it is very important to control what he/she can do. To do that we define and provide a role which is a set of actions he/she is allowed to take. There are three rancher default roles: Administrator, Standard User and User-base.

- Administrator: users have the complete control of the system and its clusters.

- Standard User: users can create new clusters and use them. The Standard User can also assigned to other users access rights to his/her clusters.

- User-base: these users can only log in.

It is also possible to create your own roles in order to define yourself permissions assigned to authorized users. That is what we have done. We have created 2 roles from the Rancher UI.

Thanks to the external authentication, for each user we are able to access the group he/she belongs to. We assigned permissions to a group and this way all its members will inherit them. The 2 created roles were:

- Admin: assigned to KelkooGroup members part of the sysadmins group. The have a total control on clusters. - New User: assigned to all users but sysadmin members. Possible actions are limited.

4.6.3 HTTPS

Now we have a working and accessible cluster, and our application running in http. For security purpose, we decide to set up https.

Once we have tls certificate (key + certificate), we need to save it on the cluster using a secret. A secret let us stock and manage sensible information such as passwords, tokens, keys... But it is safer because it is completely separated from the pod. We can create one from the rancher UI or using a yaml file as below:

```
apiVersion: v1
data:
  tls.crt: <BASE64 ENCODE .crt>
  tls.key: <BASE64 ENCODE .key>
kind: Secret
metadata:
  créationTimestamp: null
  name: rancher-dev-kelkoo-net-tls
  namespace: <NAMESPACE>
type: kubernetes.io/tls
```

The secret is then deployed in the cluster and each time we will need it, we will just call it from a yaml file. No changes are needed in our application or deployment.

```
apiVersion: extensions/v1beta1
```

```
kind: Ingress
metadata:
  name: imageresizer-secure
spec:
  rules:
  - host: whateveryouwant.rancher.dev.kelkoo.net
    http:
      paths:
      - backend:
          serviceName: imageflyresizer-deployment
          servicePort: 80
        path: /
  tls:
    - hosts:
        - whateveryouwant.rancher.dev.kelkoo.net
      secretName: rancher-dev-kelkoo-net-tls
```

As we can see we just need to add the key word tls followed by the hostname and the secret name used to save the certificate.

4.7 Autoscaling

One of the main reasons why we chose this solution was its flexibility when the traffic is increasing. With Kubernetes, new pods can be created automatically when the cluster is in a particular state. That way, our applications can function efficiently even when the traffic is huge, and they use only the needed resources.that gives an advantage in managing available resources.

4.7.1 Horizontal Pod Autoscaling

There are a lot of types of autoscaling but the one we are interested in in the Horizontal Pod Autoscaling. It consists in increasing the number of pods according to the cpu and/or memory usage (or other custom metrics).

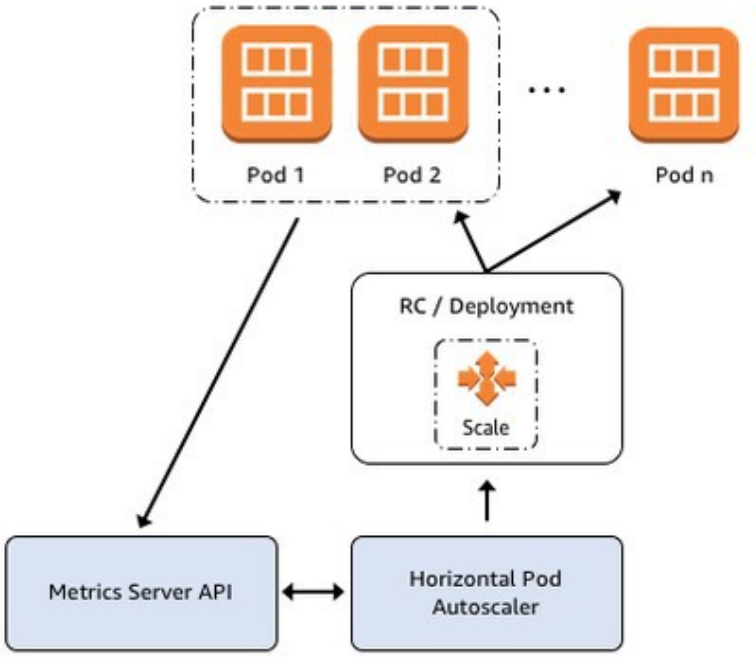

Figure 11: Horizontal pod autoscaler

First of all we need to install a metrics server which will collect data on the resources usage of pods. The hpa, from collected data, will check if our pods are in a certain state (ex: cpu usage greater than 50%) defined at its creation , and will increase the number of pods.

4.7.2 Creation of an horizontal pod autoscaler (hpa)

An hpa can be created from the rancher UI but it is better to use a yaml file because this way you could specify advanced options. An example of such a file could be:

```
    apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: hpa1
  namespace: default
spec:
  maxReplicas: 10
  metrics:
  - resource:
      name: memory
      target:
        averageUtilization: 4
        type: Utilization
    type: Resource
  - resource:
      name: cpu
      target:
        averageUtilization: 25
        type: Utilization
    type: Resource
  minReplicas: 1
  scaleTargêtref:
    apiVersion: apps/v1beta2
    kind: Deployment
    name: imageflyresizer-deployment
```

Here we are interested by the cpu/memory usage. We also specify the deployment we are interested in. When the cpu average usage will go above 25% and/or the memory average usage will go above 4%, new pods will be created until both the usages reach a value lower than their respective threshold.

4.7.3 Gatling

Gatling is a test tool easy to use and that let us emulate the behavior of one or several clients sending http requests to a specified server . It also provides results about the test which lets us know if the server is working correctly.

It shows us the numbers of requests sent to the server as well as their results.

In this case, gatling is used to send several requests to the host defined in the ingress and, due to the increase of the resources usage, new pods will be created when the condition specified in the hpa will be verified.

Figure 12: Gatling test report

5 Societal and environmental impact

5.1 Societal and environmental personal impact

Due to the covid-19 crisis, I had to stay home so I didn't do any professional activity in presence

The computer I received to work at home was a DELL latitude 6440 with a total carbon print of 348 kg eq. CO2, which is like refueling 3 times a VW golf. The usage contributes up to 23% of the total. Moreover, this model has an annual energy consumption of 26 KWh; and if we consider the internship duration of 5 months, we obtain 11KWh.

5.2 KelkooGroup politics

KelkooGroup strictly uses hardware with low carbon prints. Moreover they have a good work scheduling with flexible hours which allows a limited number of people to access the structure at the time.

5.3 Global impact of the project

5.3.1 Impact environnemental

In this project, a limited environmental impact has always been the goal.Choice of low impact hardware and increase of resources only if strictly needed, are few of the measures taken to limit the impact. However the the energy consumption will increase due to the number of applications working simultaneously which will grow bigger.

6 Conclusion

6.1 Personal Experience

One of the greatest difficulties was understanding how Kubernetes works. Although it is a tool which make management and deployment in cloud easier, it is so vast and complex. It is very important to understand its key concepts. The security aspect is also important because it will be useless to implement this solution if we can guarantee the safety of our applications. It was also a first experience of the Agile method and, it really optimizes team work and allows to focus on a set of tasks at a time. I have also learnt a lot about microservices, their impact and their integration to modern technologies.

6.2 Progression

KelkooGroup thanks to this internship explored a lot of aspects in order to use them in production. However there is still a long way to go because they still have to study monitoring, build and integration with existing tools

Figure 13: avancement

7 Glossary

docker: is an open-source project that automates the deployment of applications within software containers, providing an additional abstraction through Linux OS-level virtualization.

kubernetes: is an open source platform that automates Linux container operations . It eliminates many of the manual processes involved in deploying and scaling containerized applications. In other words, Kubernetes allows you to easily and efficiently manage host clusters running Linux containers.

jenkins: an open source automation server which enables developers around the world to reliably build, test, and deploy their software.

Helm: is a package manager for Kubernetes that allows developers and operators to more easily package, configure, and deploy applications and services onto Kubernetes clusters.

8 Bibliography

References

[1] Kubernetes-sigs, kubernetes.
`https://github.com/kubernetes-sigs/metrics-server`

[2] Rancher Docs: Authentication, by Rancher
`https://rancher.com/docs/rancher/v2.x/en/admin-settings/authentication/`

[3] Daniel Weibel, *Autoscaling apps on Kubernetes with the horizontal pod autoscaler*,

https://itnext.io/autoscaling-apps-on-kubernetes-with-the-horizontal-pod-autoscaler-798750ab7847#f8ca

[4] Docker overview, by Docker
`https://docs.docker.com/get-started/overview/`

[5] Matthew Palmer, *Ultimate Guide to ConfigMaps in Kubernetes*,
`https://matthewpalmer.net/kubernetes-app-developer/articles/ultimate-configmap-guide-kubernetes.html`

[6] Helm package
`https://helm.sh/`

[7] Gatling test analysis
`https://gatling.io/open-source/`

[8] horizontal pod autoscaling
`https://aws.amazon.com/blogs/opensource/horizontal-pod-autoscaling-eks/`

[9] kubernetes basics
`https://kubernetes.io/`