Poppit Project Report

CI-CD process and kubernetes management of a an application based on NodeJS

Albert Vidal González

May, 2023

p·it

Contents

1 Introduction

Poppit is an application based on NodeJS which seeks to offer a solution to different problems in the sale of tickets to events. In this short memory we will not focus on the benefits of the application itself but will explain in detail the process of CI-CD and the application's containerization.

For this project a private server was used with a hypervisor installed. thanks to this hypervisor a kuberente cluster was installed to host the application.

This project has been the first contact with kubernetes for the autor which has allowed him to learn about the technology and to become interested in the world of containerisation and DevOps practices.

2 Context

This project has been carried out in the PTI subject at the FIB-UPC (Facultad de Informática de Barcelona - Universidad Politécnica de Cataluña) achieving honours.

3 Docker

Docker is the most important integration of the project, as it is the linchpin of the entire integration and continuous deployment system that we will explain in the following sections. To understand how this automated system is organised, we need to understand the current architecture.

We have two dockers encapsulating the API and the database. The second one is kept static on the server without being redeployed because we are interested in keeping the data between different API releases. However, the API is not kept constant and the Docker needs to be regenerated with some frequency. Dockerhub help with this last task.

Encapsulating the application in containers gives a number of benefits that can significantly improve the management and deployment of the system. Firstly, Docker allows to create lightweight and portable containers that can be easily deployed across a wide range of platforms and production environments. This means that the application will be easier to distribute and migrate.

Another important benefit of Dockerisation is scalability and efficiency. Compared to other virtualisation solutions, Docker offers very fast deployment times and lower resource consumption, which means that Docker containers are easy to scale (as we will see in the following sections).

Finally, one of the most important aspects of Docker is security, since we isolate the content of the other elements of the system, leaving only the strictly necessary connections (between API and DB)

4 Docker Compose

Docker Compose is a very useful tool for organising and easily linking Docker containers. I used it in an early stage of the project. Even if I didn't use it now, it would be a perfectly good solution, although it wouldn't be as scalable as the current one.

It allows the two main Dockers I explained in the previous section to be linked together, thanks to the docker-compose.yml configuration file. This file allows the administrator to explicitly detail how they want the Docker network to look, with tasks such as deployment dependencies, container IPs and open ports.

5 Jenkins

Jenkins is the engine I chose to power Continuous Integration, more commonly known as CI. At the start of the project I decided to use it over other competitors such as Gitlab CI for the following reasons.

Firstly, we didn't have CD (Continuous Deployment) at the start of the project and deployment was done using docker-compose and a Makefile. This required me to somehow access the server shell and redeploy the Docker containers. Due to the university's infrastructure, the machines on which I am deploying the Docker containers are under VPN restrictions. So ssh access to the machines could not be automated due to the two-factor authentication.

The practical solution to this problem was to set up a Jenkins on the server (inside the VPN) [1] that listens to a webhook on the repo [2], and whenever it receives a notification, it executes the commands [3] (git pull and make restart) that clone the latest version of the repo and redeploy it in a Docker as mentioned above.

As the project has progressed, the functionality of Jenkins has been adapted to verify its interconnectivity with other components that have become indispensable to the system [4], which are explained later in this report.

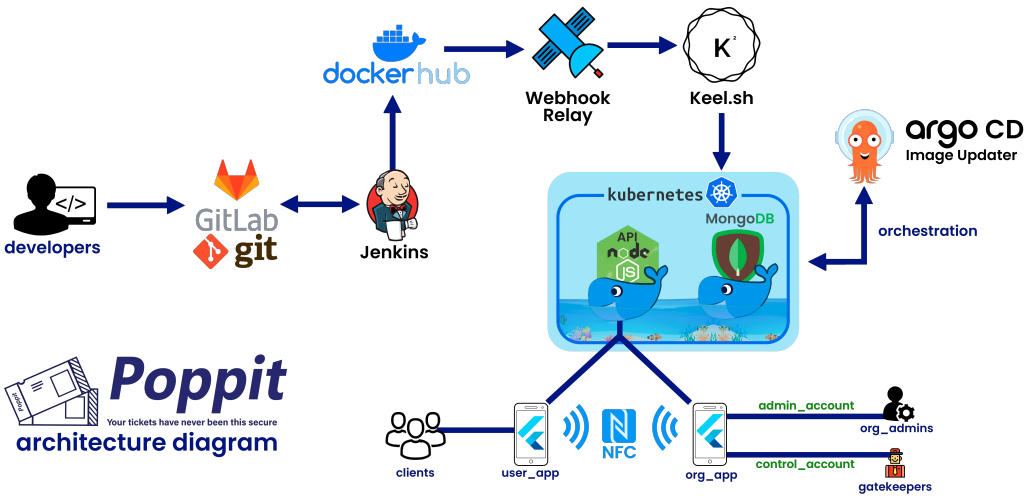

Figure 1: System architecture

As can be seen in the diagram above, Jenkins is currently responsible for grabbing the updated repository folders, building the Docker image and publishing it to the DockerHub repo to continue the CI/CD process. The Jenkins configuration includes writing a Jenkinsfile that indicates the various stages of pipeline development, which is triggered each time a merge to master is performed and a new release is published.

It also incorporates the Docker plugin and stores the credentials of the various places where the pipeline is accessed. Other features could be implemented in the future, such as code security checks or other powerful tools that Jenkins can integrate with.

6 Kubernetes cluster

Using the tools provided by the university (and without spending any money), Although it is a small project, I have decided to use kubernetes to learn about the technology. I deployed a simple cluster (3 nodes, 2 workers and 1 control plane) on the servers, trying to make it as realistic as possible and allowing me to work in a comfortable way. I have used k3s as a kubernetes distribution. [5]

To do this I have used some open source tools which we will explain in the following points.

In this section I will not go into very technical aspects, as it not the aim of this memory. I am going to focus on the CI-CD process instead of the cluster ittelf.

The following figure represents the life cycle from when a push is made to docker-hub (with a new image of the api) until it is deployed to our cluster.

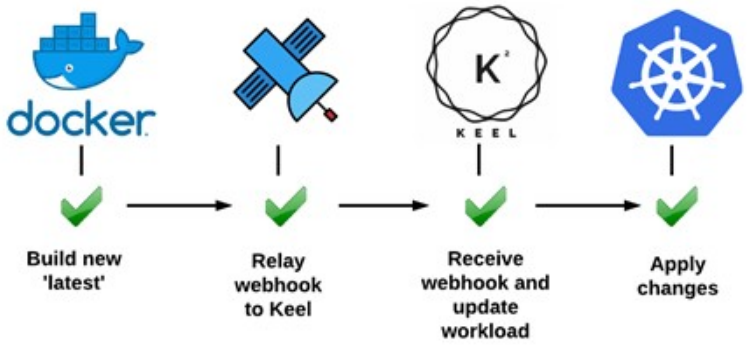

Figure 2: Life cyle of the docker image

6.1 Webhook relay

Webhook relay is the tool that allows us to communicate our docker-hub account with keel, which we will explain in the next point. [6]

This machine provides us with two access tokens, one with the tag "key" and the other with the tag "secret". I will use these tags to define the "tunnel" from the dockerhub to the keel service in the server.

The following command is the one I used to do this.

```
helm upgrade --install webhookrelay-operator --namespace=push-
   ↪ workflow webhookrelay/webhookrelay-operator \
  --set credentials.key=$RELAY_KEY --set credentials.secret=
     ↪ $RELAY_SECRET
```

Listing 1: Command to enter tags on the server

6.2 Keel

Keel allows our Kubernetes cluster to update the docker images. In our case, it updates the api image for the new one that has been uploaded to the docker hub and arrived at keel thanks to the webhook relay. Using helm, I was able to deploy this application on our cluster in a simple way [7].

To make keel work, we had to write some rules in the manifest of our node application. Keel has many combinations, but we used this one because we felt it was the most appropriate.

```
annotations:
    keel.sh/policy: force
    keel.sh/trigger: poll
    keel.sh/pollSchedule: "@every 3m"
```

Listing 2: YAML code fragment

These rules force a pull of the image that its on the docker-hub every 3 minutes and compare it to the image I have on our deployed server. Keel allows to do pulls every minute, but with the default docker hub I was not allowed to do more than 30 pulls per hour.

So from the time an image is pushed to the docker hub until we have it running on our cluster, it can take anywhere from 10 seconds (best case) to 3:10 minutes (worst case).

6.3 Services

There are three services deployed inside the cluster. I have defined two YAML files for each service. [8]

→ API service, exposes the API to port 8081 of the machine and it will be accessible from the outside via `https://poppit.ddns.net:40341/` (explained at the following point)

→ MongoDB service, this service is not exposed. It is a database that is only accessible from the private network. In the deployment YAML file of the API, I have defined the container name of the DB so that it can connect.

→ ArgoCD service, exposes the Argo server (explained below) on port 8080 of the machine, which is accessible from the outside at `http://nattech.fib.upc.edu:40340/`

6.4 Domain and TLS

I have created a free domain so I can access from outside the private network via https. This domain has been created with the no-ip tool [9].

I have defined two new manifests on our server. One is needed to link the server to the domain I have created and the other one is used to define the self-signed certificate.

In this way, to access our server we will use → `https://poppit.ddns.net:40341/`

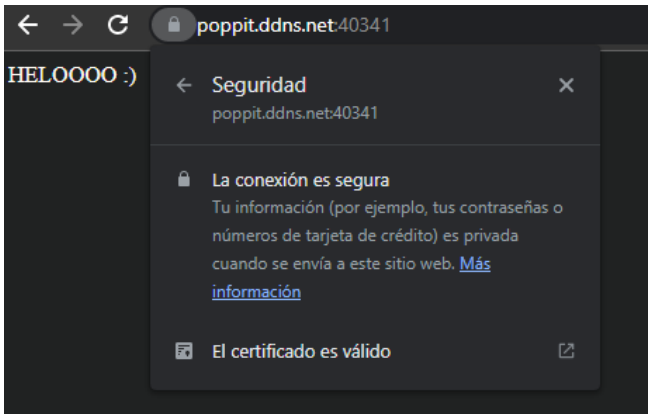

Figure 3: Domini i TLS

6.5 Persistence of images

In order not to lose the images every time the api image is updated and the new pod is created, I have had to create two persistent volumes to save the api images as they cannot be saved in the database. [10]

We did this by defining classic Kubernetes persistent volumes (pv and pvc), which store the images on the server and store them between sessions.

6.6 ArgoCD orquestration

I used ArgoCD to orchestrate our cluster in an easy way [11]. ArgoCD shows us in a very visual way what's going on in our cluster, so we don't have to run console commands. To install ArgoCD on the server we used the homebrew tool [12].

In this image we can see our cluster from ArgoCD. We can see the active pods and the services that can be exposed, in both cases the api and the MongoDB database. ArgoCD also shows the logs of our pods, so that we can see what is being done at any time, and in the event of an error, we can easily and visually detect it.

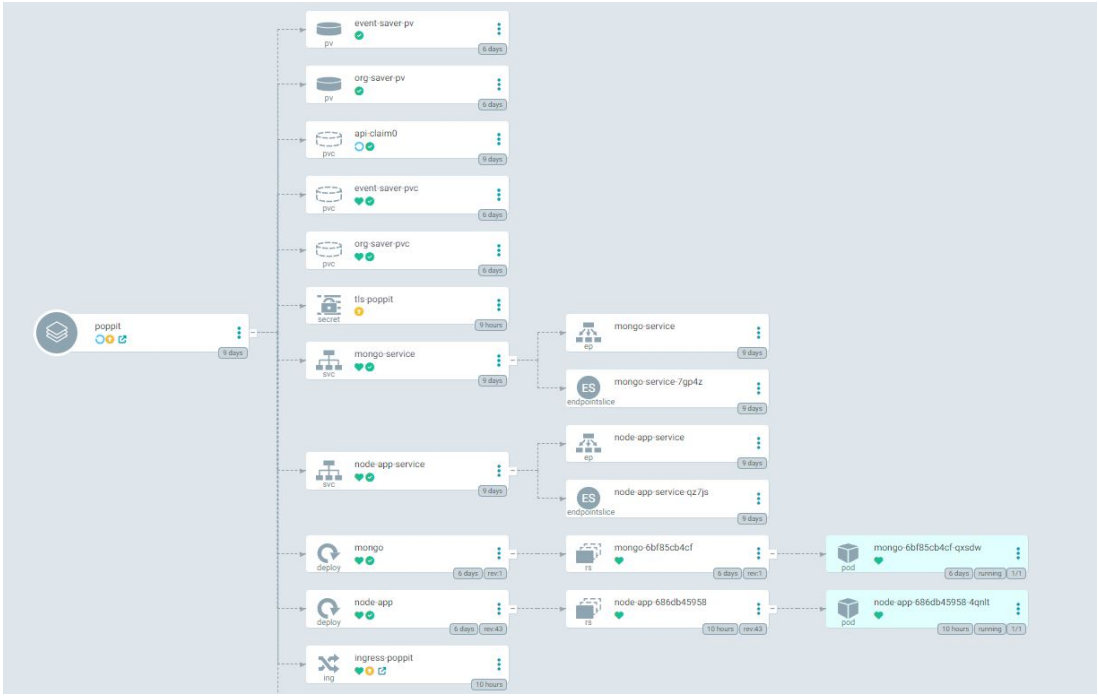

Figure 4: Cluster seen form ArgoCD

7 Possible improvements

7.1 Dockerhub premium and server reliability

Currently, dockerhub only allows me to pull an image every 3 minutes. By paying the dockerhub premium subscription would allow me to do unlimited image pulls, which would result in almost instantaneous deployment of the service.

On the other hand. The reliability of the university servers is quite low. Throughout the development of the practice I have had various problems. The virtual machines have been deleted twice and I have found that the servers go into 'sleep' mode in the morning, which does not allow us to work at night. That's why I think it's important to have a good service provider in a real system, such as AWS, Azure, Google Cloud...

7.2 Own domain name

For the development of the project I have used a standard and free domain name provided by the no_ip tool explained in the previous sections. I believe that in a real situation it would be necessary to buy a personalised domain name.

References

[1] *Jenkins Installation: Informació de suport per instal·lar i configurar Jenkins a un dels servidors.* URL: https://www.makeuseof.com/install-jenkins-on-ubuntu/.

[2] *Jenkins Webhook detection: Informació relacionada amb l'obtenció de dades per part de Jenkins.* URL: https://santoshk.dev/posts/2022/how-to-setup-a-github-to-jenkins-pipeline-with-webhook/.

[3] *Jenkins pipeline trigger: Informació utlitzada per poder fer trigger de la pipeline de Jenkins.* URL: https://www.youtube.com/watch?v=r5zhTu694Kc.

[4] *Jenkins/ArgoCD connection: Informació utlitzada per connectar Kubernetes amb Jenkins.* URL: https://www.opsmx.com/blog/how-to-enable-ci-cd-with-argo-cd-and-jenkins/.

[5] *k3s Install: Instal·lació de k3s a un servidor.* URL: https://computingforgeeks.com/install-kubernetes-on-ubuntu-using-k3s/.

[6] *Webhook relay usage: Pàgina web usada com a base per fer servir webhook relay.* URL: https://webhookrelay.com/.

[7] *Keel usage: Pàgina web usada com a base per fer servir keel.sh.* URL: https://keel.sh/docs/#kubernetes.

[8] *Definition of deployments and services: Documentació per crear serveis i deployments a kubernetes.* URL: https://sweetcode.io/how-to-create-kubernetes-deployments-and-services-using-yaml-files/.

[9] *no-IP free domain: Domini gratuït per expossar la API amb TLS.* URL: https://www.noip.com/es-MX.

[10] *Persistent Volume Documentation: Documentació per crear volums persistents a k3s.* URL: https://kubernetes.io/docs/concepts/storage/persistent-volumes/.

[11] *ArgoCD init: Instal·lació i inització d'ArgoCD a un servidor.* URL: https://argo-cd.readthedocs.io/en/stable/getting_started/.

[12] *HomeBrew Install: Instal·lació de HomeBrew a un servidor.* URL: https://phoenixnap.com/kb/homebrew-for-linux.