OUEST SOLUTIONS INFORMATIQUES

IUT La Roche-Sur-Yon

GREPILLOUX Antonin
LP A2SR

2022-2023

OUEST SOLUTIONS INFORMATIQUES

IUT La Roche-sur-Yon
Pôle Sciences et technologie

Nantes Université

SOMMAIRE

Table des matières

Course 1

Mission 1

Découverte du module, Communication en langue anglaise :

The discovery of the *" Communication en langue anglaise "* module began with a personal introduction from the teacher/trainer, sharing their profile and background.
After this brief introduction, all students introduced themselves, one by one, including a description of themselves and their company.

Découverte et utilisation de Docker et Git :

During this first session, we practiced Docker and Git through english communication between students and the teacher/trainer.
It started with a presentation of Docker, including its workings and features.
We then practiced Docker, with the installation of Docker Desktop on the Linux Debian environment.

Secondly, using the Git tool, we retrieved a Github project to build a Docker container.
We followed the official Docker documentation, using the commands provided.
We edited a file named Dockerfile, containing various configuration elements for the container.
We defined several elements, such as "node" and the IUT proxies.

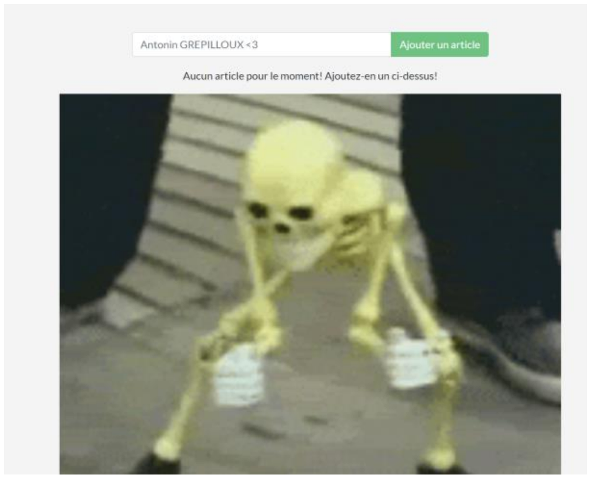
Finally, After defining our configuration file, we created the container.
We then launched the container to use it.
The container indicated it was listening on port 3000, so we opened a web browser and entered our IP and listening port.

Mission 2

Modification de l'application :

We had to modify our application to see the change with the old container.

For that, I modify the src/static/app.js file to translate the sentences in French. Then I also modify the HTML page to add a GIF image. Then we rebuild our application.

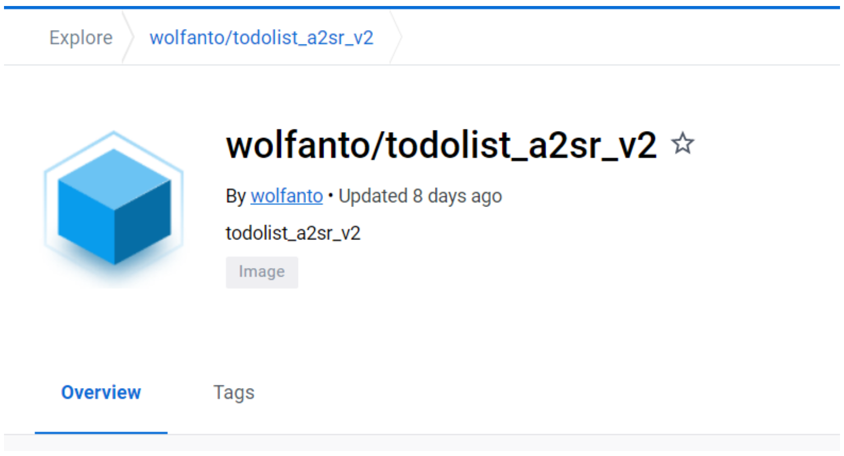


Aucun article pour le moment! Ajoutez-en un ci-dessus!

Partage et installation de l'image Docker :

After rebuilding our application, we try it out to see our changes.

We create an account on the hub docker site that will allow us to share images with our classmates so they can see our changes. We use various commands to send and retrieve the images from the Docker site.



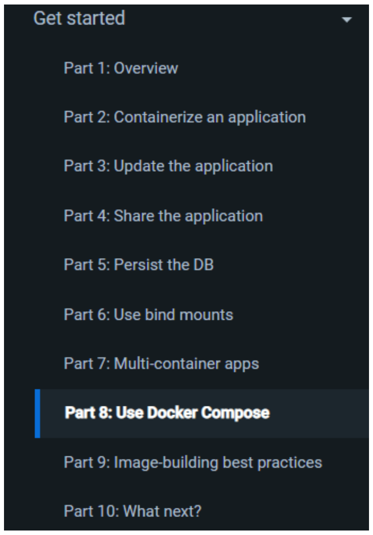
wolfanto/todolist_a2sr_v2

By wolfanto • Updated 8 days ago

todolist_a2sr_v2

Mission 3

Suivi du tutoriel Docker :
We follow the tutorial proposed by Docker which will allow us to understand how to use Docker. We follow several steps explaining us step by step how it works. We could understand how to create a container, update it, share it, set several options and configurate the configuration files.


Get started
Part 1: Overview
Part 2: Containerize an application
Part 3: Update the application
Part 4: Share the application
Part 5: Persist the DB
Part 6: Use bind mounts
Part 7: Multi-container apps
Part 8: Use Docker Compose
Part 9: Image-building best practices
Part 10: What next?

Mise en place DB Persist :
After all the steps of the tutorial, we had to set up a persistent database to save the todolist items, even if the container is turned off and then on again.

For this we create a persistent volume for the container.

Découverte et utilisation de Docker Compose :
Docker Compose is a tool that was developed to help define and share multi-container applications. With Compose, we can create a YAML file to define the services and with a single command, can spin everything up or tea it all down.

We had to create a file named docker-compose.yml with a set of information allowing the application to work properly. For this we define the service (Node), and an installation command. We also define the port on which to connect.

We indicate the directory folder and the volume. We also specify the MySQL environment.
Then, via the Docker Compose file, we run it, and check if the container has been created.

Conclusion :

To summarize, in the first session, Docker and Git were practiced through English communication between students and teacher. There was a presentation of Docker and its workings, followed by the installation of Docker Desktop on Linux Debian.

Using Git, a Github project was retrieved to build a Docker container, following the official documentation. A Dockerfile was edited with configuration elements for the container, created and launched.

The container was listening on port 3000 and was used by entering the IP and port in a web browser.

We modified our application and shared it with our classmates. We also followed the Docker tutorial with the implementation of a persistent database.

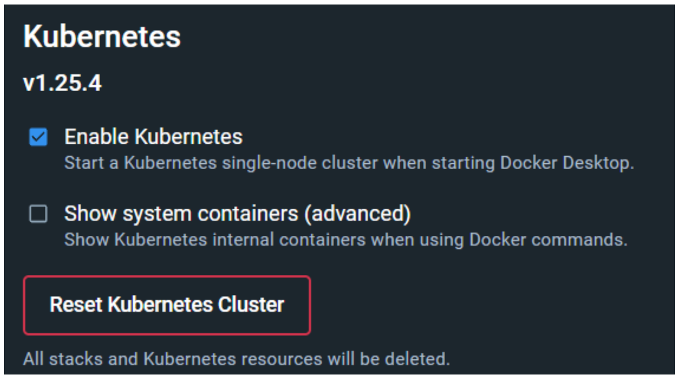
Then we saw how to use Docker Compose. We also created a GitHub repository and commited our repository.

Course 2

Mission 1

We have installed Docker Desktop for Windows to use Docker and Kubernetes.

For this we took the Docker Desktop installer and installed it. Then, in the settings of Docker Desktop we went to the Kubernetes tab and checked the box to enable Kubernetes.


Kubernetes

v1.25.4

[✓] Enable Kubernetes
Start a Kubernetes single-node cluster when starting Docker Desktop.

[ ] Show system containers (advanced)
Show Kubernetes internal containers when using Docker commands.

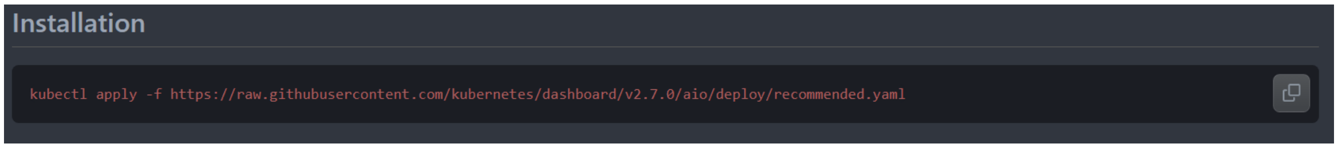
Reset Kubernetes Cluster

All stacks and Kubernetes resources will be deleted.

Then we installed Kubernetes dashboard. For this we installed via Powershell the Kubectl utility.

winget install -e --id Kubernetes.kubectl

Using kubectl, we have deployed Kubernetes Dashboard via a YAML file on the official Kubernetes github.

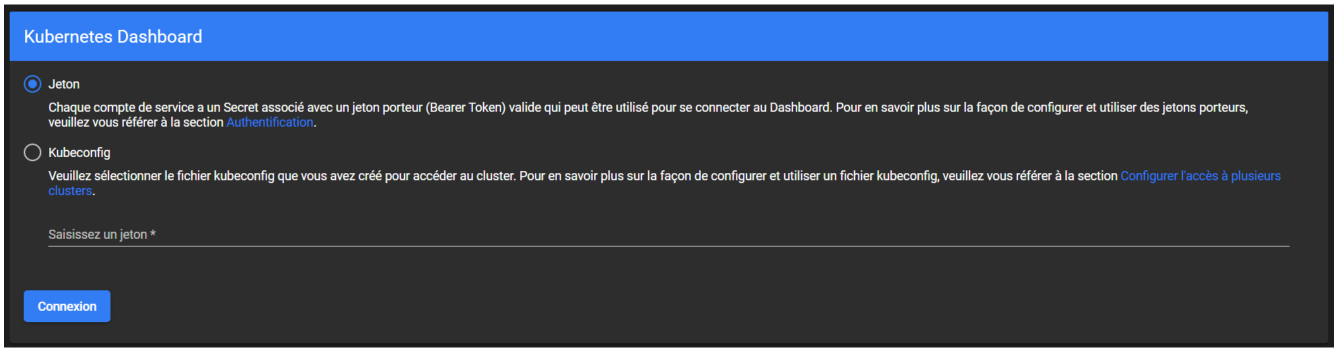

Installation

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml
```

After installing Kubernetes, we started it. After that, Kubernetes Dashboard asked us for a token to authenticate. For this we created a "Service Account" and a "ClusterRoleBinding". After these steps, we generated the Token to authenticate. After authentication on Kubernetes Dashboard we now have access to the Dashboard.

kubectl proxy

http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/



Kubernetes Dashboard

Jeton

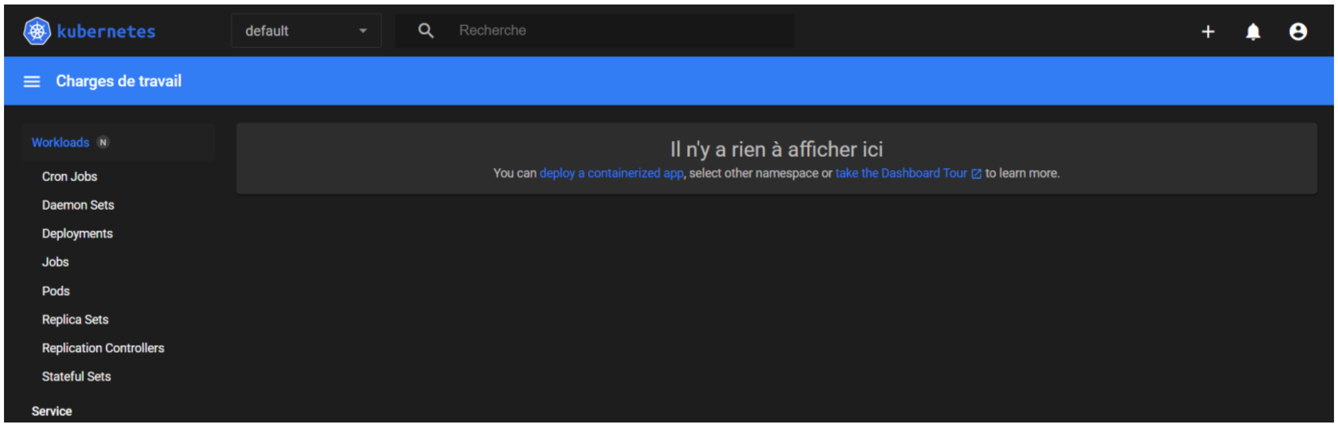
Chaque compte de service a un Secret associé avec un jeton porteur (Bearer Token) valide qui peut être utilisé pour se connecter au Dashboard. Pour en savoir plus sur la façon de configurer et utiliser des jetons porteurs, veuillez vous référer à la section Authentification.

Kubeconfig

Veuillez sélectionner le fichier kubeconfig que vous avez créé pour accéder au cluster. Pour en savoir plus sur la façon de configurer et utiliser un fichier kubeconfig, veuillez vous référer à la section Configurer l'accès à plusieurs clusters.

Saisissez un jeton *

Connexion

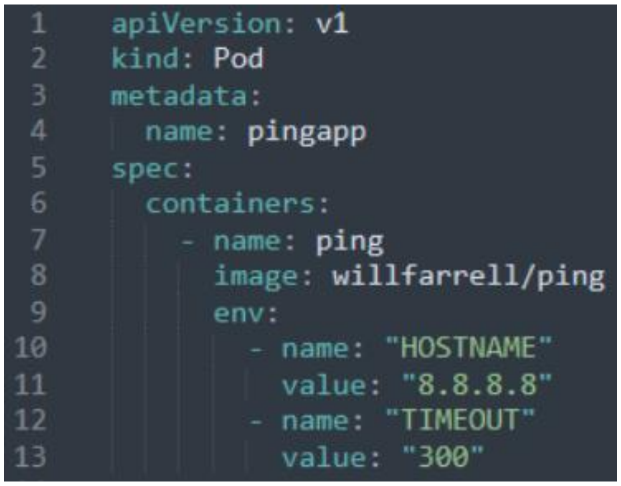


Il n'y a rien à afficher ici

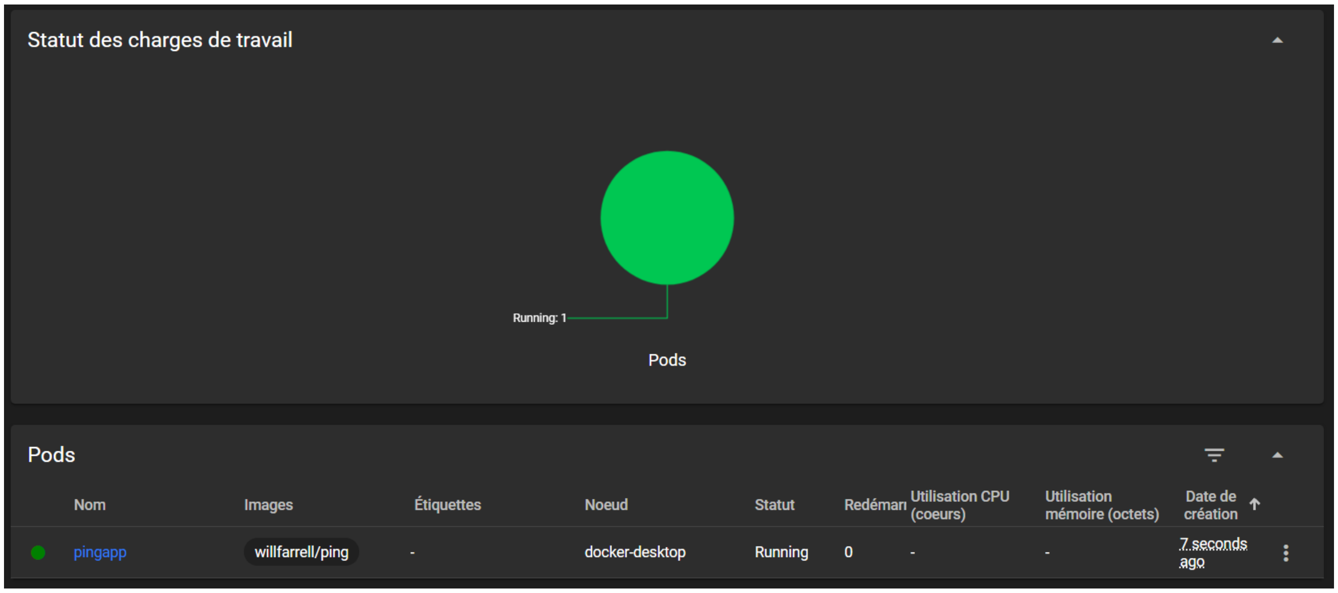
You can deploy a containerized app, select other namespace or take the Dashboard Tour to learn more.

Mission 2

We updated the A2SR github project with the "git pull" command and deployed a YAML file on our Kubernetes cluster.
After installing Kubernetes Dashboard, we created a Pod.
The default YAML file needs to be modified to work. When we install it an error appears, it comes from a syntax error.
Here is the line to correct.



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pingapp
spec:
  containers:
    - name: ping
      image: willfarrell/ping
      env:
        - name: "HOSTNAME"
          value: "8.8.8.8"
        - name: "TIMEOUT"
          value: "300"
```

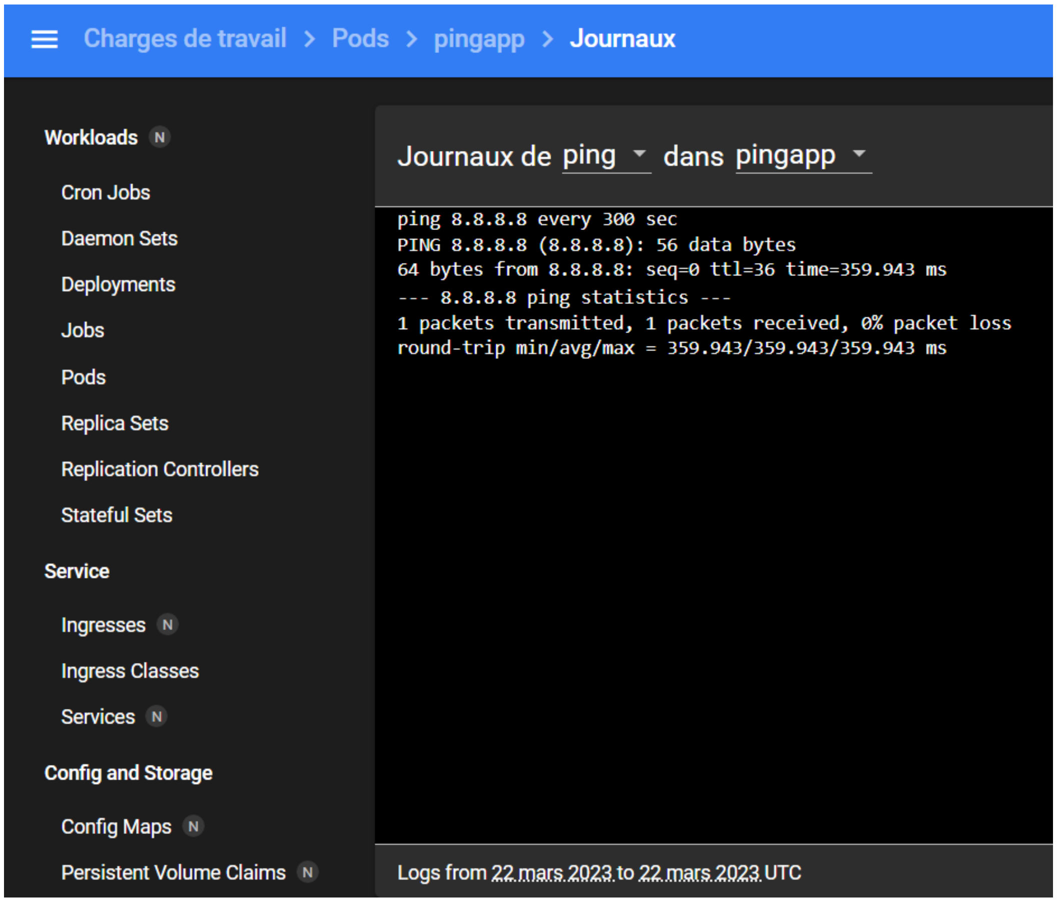


Statut des charges de travail

Pods

	Nom	Images	Étiquettes	Noeud	Statut	Redéman	Utilisation CPU (coeurs)	Utilisation mémoire (octets)	Date de création
●	pingapp	willfarrell/ping	-	docker-desktop	Running	0	-	-	7 seconds ago

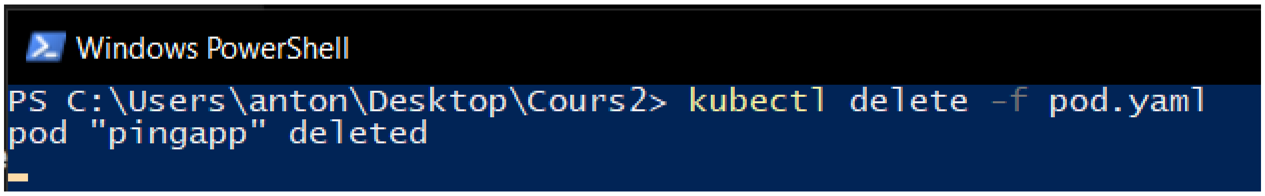
We also needed to check our pod logs to notice a ping.



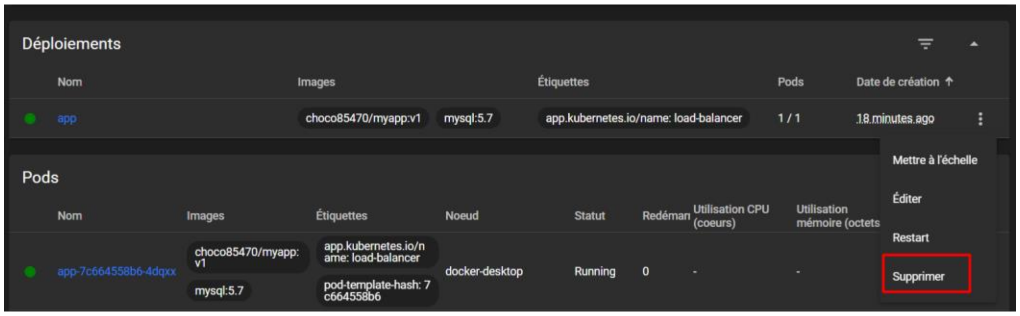
```
ping 8.8.8.8 every 300 sec
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=36 time=359.943 ms
--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 359.943/359.943/359.943 ms
```

Logs from 22 mars 2023 to 22 mars 2023 UTC

After installing and verifying our Pod, we need to delete it.
To do this, we can delete it via command line or directly from the Kubernetes dashboard.



```
PS C:\Users\anton\Desktop\Cours2> kubectl delete -f pod.yaml
pod "pingapp" deleted
```



Déploiements

	Nom	Images	Étiquettes	Pods	Date de création
●	app	choco85470/myapp:v1 mysql:5.7	app.kubernetes.io/name: load-balancer	1 / 1	18 minutes ago

Pods

	Nom	Images	Étiquettes	Noeud	Statut	Redéman	Utilisation CPU (coeurs)	Utilisation mémoire (octets
●	app-7c664558b6-4dqxx	choco85470/myapp:v1 mysql:5.7	app.kubernetes.io/name: load-balancer pod-template-hash: 7c664558b6	docker-desktop	Running	0	-	-

After all the assignments were done, we uploaded our work to our Github repository.

```
git config --global user.name "username"
git config --global user.email username@mail.com
git init
git add . (to add all the files in the current directory in the .git)
git commit -m "Commit English Courses" (to commit modification)
git remote add origin https://github.com/wolfanto/A2SR.git (to specify where is
my github repository)
git push -u origin main (to push the main branch)
```

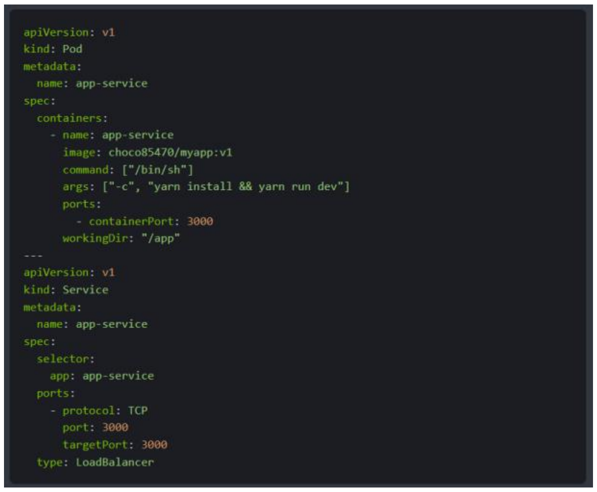
To conclude course n°2, we installed several services such as Docker Desktop and Kubernetes, and understand how they work. We also installed a dashboard for Kubernetes. We learned how to use Kubernetes and its dashboard to create and delete a Pod.

Finally, we published our code on our Github repository.

Course 3

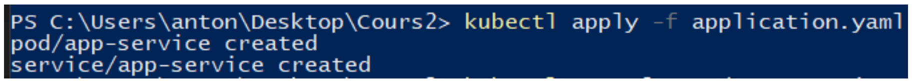
Mission 1

We create a Pod and its service in a .YAML file, then we apply it via the Kubectl utility in command line.



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-service
spec:
  containers:
    - name: app-service
      image: choco85470/myapp:v1
      command: ["/bin/sh"]
      args: ["-c", "yarn install && yarn run dev"]
      ports:
        - containerPort: 3000
      workingDir: "/app"
---
apiVersion: v1
kind: Service
metadata:
  name: app-service
spec:
  selector:
    app: app-service
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
  type: LoadBalancer
```

kubectl apply -f application.yaml



```
PS C:\Users\anton\Desktop\Cours2> kubectl apply -f application.yaml
pod/app-service created
service/app-service created
```

This file contains two Kubernetes objects: a Pod and a Service.

The Pod object describes the containerized application that will run inside the Kubernetes cluster. The parameters of this object are:

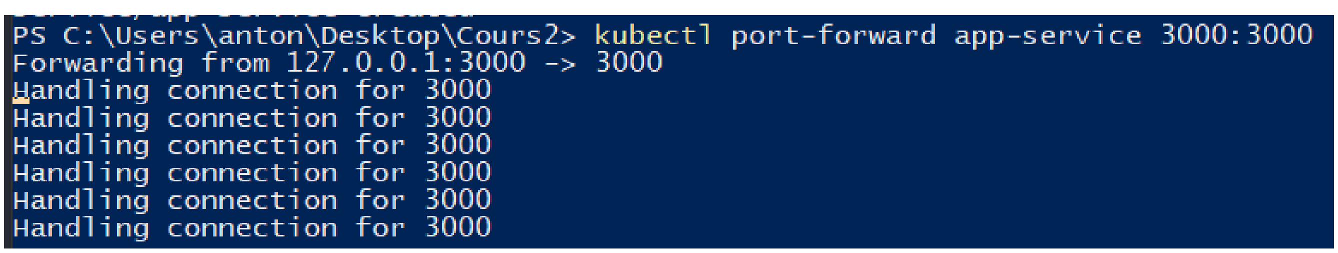
apiVersion: The version of the Kubernetes API to use (in this case, version 1).
kind: The type of object being defined (in this case, a Pod).
metadata: Information about the Pod, such as its name.
spec: The specifications for the Pod, including which container image to use, any environment variables, and which ports to expose.

The Service object describes how traffic will be routed to the Pod. The parameters of this object are:

apiVersion: The version of the Kubernetes API to use (in this case, version 1).
kind: The type of object being defined (in this case, a Service).
metadata: Information about the Service, such as its name.
spec: The specifications for the Service, including which Pod(s) to target, which ports to listen on, and which type of Service to use (in this case, a LoadBalancer).

After creating and applying our Pod, we open its ports with the Kubectl command line utility.

kubectl port-forward app-service 3000:3000



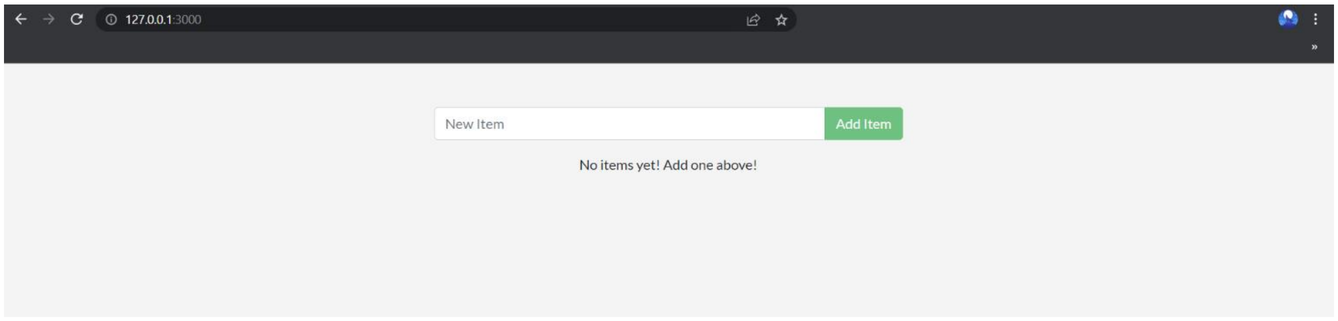
```
PS C:\Users\anton\Desktop\Cours2> kubectl port-forward app-service 3000:3000
Forwarding from 127.0.0.1:3000 -> 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
```

The goal of the command (*kubectl port-forward app-service 3000:3000*) is to forward traffic from port 3000 on the local machine to port 3000 of the Kubernetes service named "app-service". This is useful when the application running inside the Kubernetes cluster is not accessible from outside the cluster.

kubectl: The Kubernetes command-line tool used to interact with the Kubernetes API.
port-forward: The sub-command used to forward traffic from a local port to a port on a Kubernetes Service or Pod.
service/app-service: The name of the Kubernetes Service to forward traffic to.
3000:3000: The local port (3000) to forward traffic from and the target port (also 3000) on the Kubernetes Service to forward traffic to. This specifies that traffic received on the local machine at port 3000 should be forwarded to port 3000 of the Kubernetes Service named "app-service".

To check that our manipulation works, we open a web browser and go to the following address:

http://127.0.0.1:3000



No items yet! Add one above!

To conclude course n°3, we have created a Pod and its service, and exposed its port on a local machine port. Then we check if our manipulation is correct.

Finally, we published our code on our Github repository.

LEXIQUE

Debian 11 : Linux distribution

Linux : Linux is an open-source operating system. An operating system is software that directly manages physical system components and resources, such as the processor, memory, and storage. It acts as the interface between applications and hardware.

Docker : Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Git : Git is a distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Docker Desktop : Docker Desktop is an application for MacOS, Linux, and Windows machines for the building and sharing of containerized applications and microservices.

Kubernetes : ubernetes is an open source orchestration system for automating the management, placement, scaling and routing of containers that has become popular with developers and IT operations teams in recent years.

kubectl : kubectl is the Kubernetes-specific command line tool that lets you communicate and control Kubernetes clusters.

YAML : YAML is a data serialization language that is often used for writing configuration files.

Github : GitHub is a for-profit company that offers a cloud-based Git repository hosting service. Essentially, it makes it a lot easier for individuals and teams to use Git for version control and collaboration.

Kubernetes Dashboard : The Kubernetes Dashboard is a web-based management interface that enables you to: deploy and edit containerized applications. assess the status of containerized applications. troubleshoot containerized applications.

BIBLIOGRAPHIE

Docker. (n.d.). https://hub.docker.com/

Install on Linux. (2023, January 30). Docker Documentation. https://docs.docker.com/desktop/install/linux-install/

Perochon, M. (n.d.). *GitHub - mperochon/A2SR*. GitHub. https://github.com/mperochon/A2SR

Wikipedia contributors. (2023, January 16). *Linux*. https://fr.wikipedia.org/wiki/Linux

Kubernetes. (n.d.). https://kubernetes.io/