

Docker

Si vous souhaitez faire des modifications ou ajouter des trucs, n'hésitez pas à les proposer !

Documentation

Le cours de Dirane :

https://drive.google.com/file/d/1q_PaPjBBVS3a70LSAnXKN3hTIJO08I7M/view

Documentation docker

<https://docs.docker.com/>

script pour installer docker

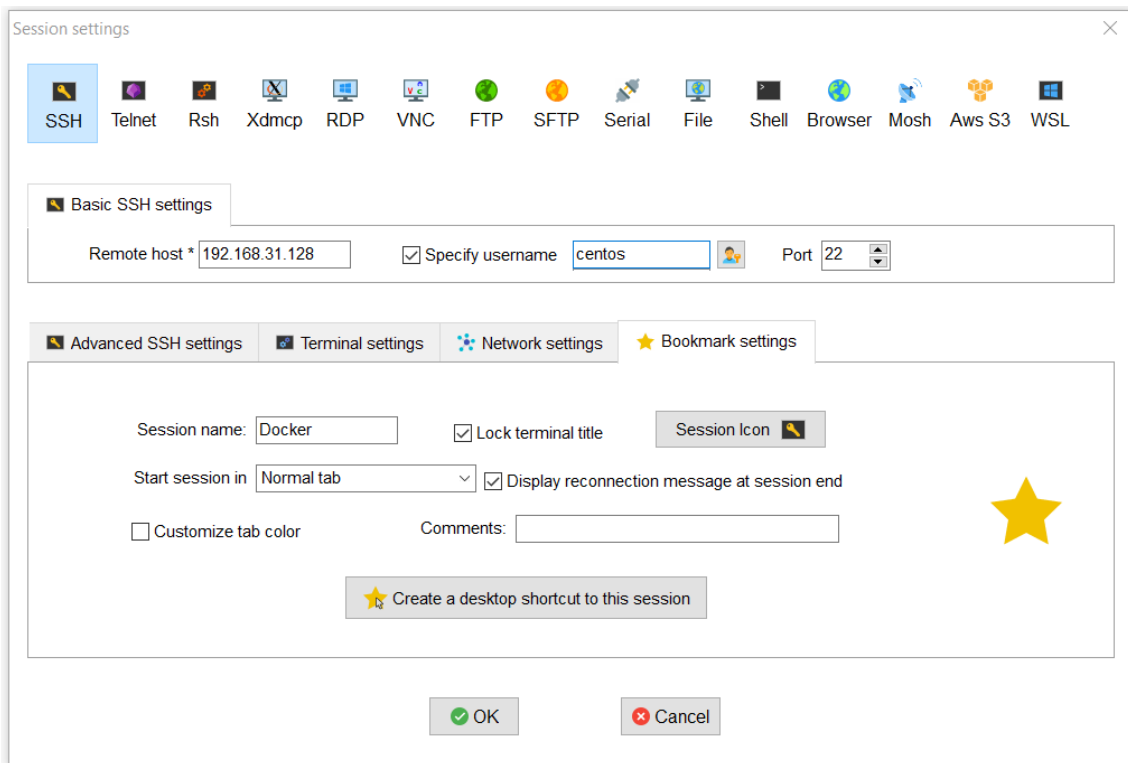
<https://get.docker.com/>

Cheatsheet Docker : <https://dockerlabs.collabnix.com/docker/cheatsheet/>

Mise en place de l'environnement de travail

sudo dhclient ens33 pour définir une adresse ip

on mobaxstream



j'ai été obligé de reset vmware

méthode de connexion login mot de passe
ou mot de passe avec des clefs

remettre à l'heure sa machine : `systemctl start ntpd`
`sudo ntpd -gq`

<https://www.thegeekdiary.com/centos-rhel-6-how-to-force-a-ntp-sync-with-the-ntp-servers/>

avec la mécanique d'installation décrite au début

puis ajouter l'utilisateur au groupe docker (qu'on soit pas obligé de passer par sudo)
`sudo usermod -aG docker centos`

commande docker

Déploiement de micro services

Lancer un micro service

```
^C[centos@localhost ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS    PORTS    NAMES
[centos@localhost ~]$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS    PORTS    NAMES
f0d9c8db5f0f   nginx    "/docker-entrypoint..." 4 minutes ago   Exited (0) 11 seconds ago   musing_merkle
bb9536f48ebc   hello-world "/hello"                15 minutes ago   Exited (0) 15 minutes ago   awesome_panini
[centos@localhost ~]$ docker rm awesome_panini
awesome_panini
[centos@localhost ~]$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS    PORTS    NAMES
f0d9c8db5f0f   nginx    "/docker-entrypoint..." 5 minutes ago   Exited (0) 31 seconds ago   musing_merkle
[centos@localhost ~]$ docker rm musing_merkle
musing_merkle
[centos@localhost ~]$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS    PORTS    NAMES
[centos@localhost ~]$ docker run --name webserver -p 80:80 -d nginx
```

-p pour pouvoir communiquer avec l'ext (fixe les ports) celui de gauche est arbitraire(mais il faut pas que deux micro services ait le même) pas celui de droite
celui de gauche c'est le port externe et le droit le port interne
--name pour changer le nom
nginx : un micro service représentant un serveur web
-d permet de continuer à utiliser la console
-e env var

installation d'un conteneur contenant wordpress

```
docker run --name wordpress -p 8080:80 -d wordpress
```

192.168.31.128:8080/wp-admin/setup-config.php : pour y accéder

Déployer le micro service kodekloud/webapp-color

avec les infos suivantes name : webserver port interne 8080 port externe 80 variable d'env APP_COLOR ait la valeur red

```
docker run --name webserver-red -p 80:8080 -e APP_COLOR='red' -d kodekloud/webapp-color
```

bien penser à vider le cache avec maj+F5

déployer le même ms : name webserver-blue interne 8080 et ext 8080

```
docker run --name webserver-blue -p 8080:8080 -e APP_COLOR='blue' -d kodekloud/webapp-color
```

la on a fait que déployer des choses existantes mais comment les construire

Créer nos propres images/micro-services



```
Dockerfile 455 Bytes
1 FROM python:2.7-stretch
2 #Maintainer of image
3 LABEL maintainer="175777@supinfo.com"
4 #Install of dependencies
5 RUN apt-get update -y && apt-get install python-dev python3-dev libssl-dev python-dev libldap2-dev libssl
6 RUN pip install flask flask_httpauth flask_simpleldap python-dotenv
7 #COPY SOURCE CODE IN THE IMAGE
8 COPY student_age.py /
9 #CREATE DATA FOLDER
10 VOLUME [ "/data" ]
11 #EXPOSE PORT
12 EXPOSE 5000
13 #RUN CODE
14 CMD [ "python", "./student_age.py" ]
```

1ere ligne système d'ex sous jacent

3l fournit les info du maintainer de l'img (un peu comme une signature)

5l install python et met à jour la liste des paquets

6l install dep system (de python)

8l copy le code de student_age

10l

12l port interne 5000 (celui a droite)

14l cmd que le conteneur va lancer à son démarrage

créer un dockerfile (il faut créer un fichier qui s'appelle Dockerfile

1 ubuntu image de base

2 préciser le maintainer

3 maj de la liste des package

4 nginx comme serveur web de l app

5 rajouter le code dans le rep qui va bien (nginx stock le code dans /var/www/html

6 port 80

7 cmd sera la commande qui permet de lancer nginx en mode daemon off

ADD garde l'arborescence COPY prend les fichiers sans l'arbo

```
FROM ubuntu
#Maintainer of image
LABEL maintainer="antoine"

RUN apt-get update
RUN apt install -y nginx

ADD static-website-example /var/www/html

EXPOSE 80
```

```
CMD ["nginx", "-g", "daemon off;"]
```

-y permet de faire les choses manière algo. apt plutôt que apt-get

```
docker build -t webapp:v1 .
```

le point à la fin est le contexte (l'endroit où sont présents les fichiers dont a besoin dockerfile, ici c'est le doc courant donc .)

docker images pour voir les images

```
docker run --name webapp-v1 -p 80:80 -d webapp:v1
```

faire une v2

```
docker build -t webapp:v2 .
```

 ATTENTION j'avais oublié le contexte !!!!

```
docker run --name webapp-v2 -p 80:80 -d webapp:v2
```

le client juge que l'on a pas la dernière version

objectif c'est que au moment du build récupérer la dernière version du site

supprimer fichier web app et mettre un git clone dans le dockerfile

```
FROM ubuntu
#Maintainer of image
LABEL maintainer="antoine"

RUN apt-get update
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y nginx git
RUN rm -Rf /var/www/html/*

RUN git clone https://github.com/diranetafen/static-website-example.git
/var/www/html/

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

le client veut qu'on lui envoie notre dockerfile via git

créer un repo webapp-ib et pousser le code

```

53 git init
54 ls
55 git add Dockerfile
56 git commit -m "first commit of the Dockerfile"
57 git remote add origin https://github.com/antoine-bouquet/webapp-ib.git
58 git branch
59 git push -u origin master

```

attention il faut renommer la branche en main (moi j'ai gardé master mais prochaine fois faut faire gaffe)

pseudo dockerhub : antoinebouquet1010

mdp 4*****=E*****

mettre en ligne ce qu'on a fait (push sur dockerhub)

```

69 docker login --username=antoinebouquet1010
70 docker images
71 docker push webapp:v1
72 docker tag a58fc9a9c683 antoinebouquet1010/webapp
73 docker images
74 docker tag a58fc9a9c683 antoinebouquet1010/webapp:v1
75 docker images
76 docker push antoinebouquet1010/webapp:v1
77 docker images
78 docker tag 12fee9919192 antoinebouquet1010/webapp:v2
79 docker images
80 docker push antoinebouquet1010/webapp:v2
81 docker tag 5db07b10f9a3 antoinebouquet1010/webapp:v3
82 docker images
83 docker push antoinebouquet1010/webapp:v3

```

mettre en place le build auto via github

sur quelle branch on se met pour faire le build auto

Source Type	Source	Docker Tag	Dockerfile location	Build Context	Autobuild	Build Caching
Branch	master	latest	Dockerfile	/	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

View example build rules

Deployer une app <https://hub.docker.com/r/pengbai/docker-supermario>

Réseaux Docker

Créer un réseau sur docker de type bridge, s'appelant wordpress et sur ce réseau déployer un conteneur mysql devant contenir les infos suivant : root pwd : root user: toto pwd user toto nom bdd = wordpress

création d'un réseau et affichage de celui ci

```
86 docker network create --driver bridge wordpress
87 docker network inspect wordpress
88 docker network ls
104 docker run --name mysql --network wordpress -d -e MYSQL_ROOT_PASSWORD=root -e MYSQL_USER=toto -e MYSQL_PASSWORD=toto -e MYSQL_DATABASE=wordpress mysql:5
docker run --name mysql -e MYSQL_ROOT_PASSWORD=root -e MYSQL_USER=toto -e MYSQL_PASSWORD=toto -e MYSQL_DATABASE=wordpress --network wordpress -d
mysql:5.7 (commande de dirane)
```

déploiement on ne met pas le port car c'est notre backend

déployer cette fois ci le front end wordpress et il doit connaître les infos pour se connecter au backend

```
[centos@localhost webapp]$ docker run --name wordpress --network wordpress -d -p 8080:80 -e WORDPRESS_DB_USER=toto -e WORDPRESS_DB_HOST=mysql -e WORDPRESS_DB_PASSWORD=toto
-e WORDPRESS_DB_NAME=wordpress wordpress
docker run --name wordpress --network wordpress -d -p 8080:80 -e WORDPRESS_DB_USER=toto -e WORDPRESS_DB_HOST=mysql -e WORDPRESS_DB_PASSWORD=toto -e WORDPRESS_DB_NAME=wordpress wordpress
```

déployer odoo version latest--

conseil : déployer sous la même architecture que précédemment (réseau odoo backend postgres et front end odoo)

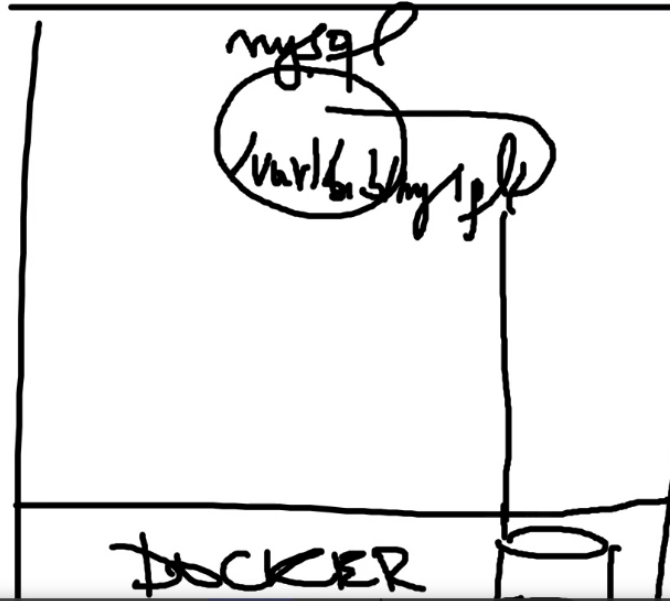
```
10 docker run -d -e POSTGRES_USER=odoo -e POSTGRES_PASSWORD=odoo -e POSTGRES_DB=postgres --name postgres --network odoo postgres:10
11 docker run -p 8069:8069 --name odoo --network odoo -d -e HOST=postgres odoo
```

Entrypoint plus cmd il concatene donc ici entrypoint.sh/odoo (voir Dockerfile)

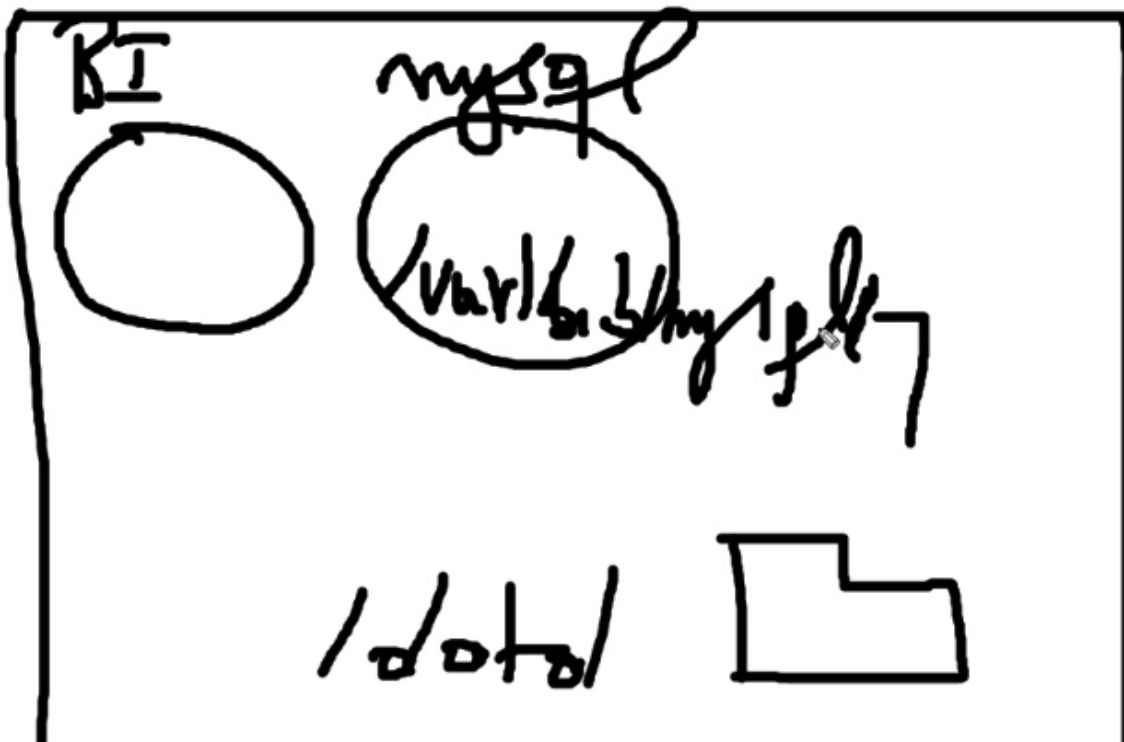
Les Volumes

mysql stocke ces données dans var/lib/mysql

1-volumes



1 - donnée stocke dans un volume (externalisation) stocker dans var/lib/docker/volu (interne a docker)



donnée stockée directement sur la machine dans /data par ex volume de type bind mount

donnée traitée dans mysql de façon temporaire mais pas sur le disque de la machine (type tmpfs)

déployer ubuntu

```
docker run --name ubuntu -d -i ubuntu
```

-i pour interactive lui dis de rester ouvert


```
docker exec ubuntu ls
```

pour exec des truc sur ubuntu

```
docker exec ubuntu echo a && echo b
```

```
docker exec -it ubuntu /bin/bash
```

-t permet de travailler avec le terminal

-i pour qu'il continue de bosser avec toi

créer un volume ubuntuvolume

créer ubuntu 1 et ubuntu 2

attache /tmp sur ubuntu 1 et 2

dans ubuntu 1 créer toto.txt avec le contenu docker

vérifier que toto est bien dans ubuntu 2

```
docker volume create ubuntuvolume
docker run --help
docker run -i -d --name ubuntu1 -v ubuntuvolume:/tmp ubuntu
docker run -i -d --name ubuntu2 -v ubuntuvolume:/tmp ubuntu
docker ps -a
docker volume inspect ubuntuvolume
docker exec -it ubuntu1 /bin/bash
docker exec -it ubuntu2 /bin/bash
```

si le volume n'existe pas il le créer quand on déploie

```
docker volume prune
```

détruit tous les volumes

recup static

créer un conteneur apache (ici httpd) qui va monter le dossier qu'on va avoir sur apache port (80:voir doc)

```
docker run -d --name webapp -p 80:80 -v
/home/centos/static-website-example/:/usr/local/apache2/htdocs/ httpd
```

il sait que c'est un bind mount pcq on lui donne un chemin

pas besoin de rebuild à chaque fois

même commande avec chemin relatif

```
docker run -d --name webapp -p 80:80 -v
${PWD}/static-website-example/:/usr/local/apache2/htdocs/ httpd
```

Redéployer odoo (réseau odoo, la bdd, le front end odoo)

conserver les données de la bdd

```
docker run -d -e POSTGRES_USER=toto -e POSTGRES_PASSWORD=toto -e
POSTGRES_DB=postgres --network odoo -v pgdata:/var/lib/postgresql/data
--name db postgres
```

```
docker run -d -p 8069:8069 --name odoo --network odoo -e USER=toto -e
PASSWORD=toto -t odoo
```

mais ca ne suffit pas de faire des volumes car on peut perdre la machine qui héberge l'appli pour garantir la résilience il faut backuper l'appli

```
docker exec -t db pg_dumpall -c -U toto > dump_`date
+%d-%m-%Y`_"%H_%M_%S`.sql
```

commande qui fait utiliser pgdump (une commande de postgres qui permet de backuper la bdd) on récupère une sortie datée

Même travail mais pour wordpress

réseau

conteneur backend (mysql) y faire un volume

conteneur front end wordpress

jouer avec le wordpress

delete la bdd

recréer la bdd

et voir si tout est la

quand on supprime des conteneurs on peut supprimer les volumes orphelin avec -v

```
docker run -d -v mysqlldb:/var/lib/mysql -e MYSQL_DATABASE=toto -e
MYSQL_USER=toto -e MYSQL_PASSWORD=toto -e MYSQL_RANDOM_ROOT_PASSWORD=1
--name mysql --network wordpress mysql:5.7
```

```
docker run -d -p 80:80 -e WORDPRESS_DB_HOST=mysql -e WORDPRESS_DB_USER=toto
-e WORDPRESS_DB_PASSWORD=toto -e WORDPRESS_DB_NAME=toto --network wordpress
--name wordpress wordpress
```

Docker Compose

Installer docker-compose

<https://docs.docker.com/compose/install/>

Version dockerisé de docker-compose :

<https://hub.docker.com/r/linuxserver/docker-compose>

ecire un fichier docker-compose.yml permettant de déployer odoo comme tout à l' heure avec un réseau spécifique et un stockage persistant

Très important que le fichier s'appelle **docker-compose.yml**

Mettre le fichier dans un dossier odoo

```
version: '2'
services:
  odoo:
    container_name: odoo
    image: odoo
    depends_on:
      - db
    ports:
      - "8069:8069"
    environment:
      - USER=toto
      - PASSWORD=toto
    networks:
      - odoo-network
  db:
    container_name: postgres
    image: postgres:10
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_PASSWORD=toto
      - POSTGRES_USER=toto
    networks:
      - odoo-network
    volumes:
      - pgdata:/var/lib/postgresql/data

networks:
  odoo-network:
    driver: bridge

volumes:
  pgdata:
    name: pgdata
```

sur docker-compose -v supprime TOUS les volumes

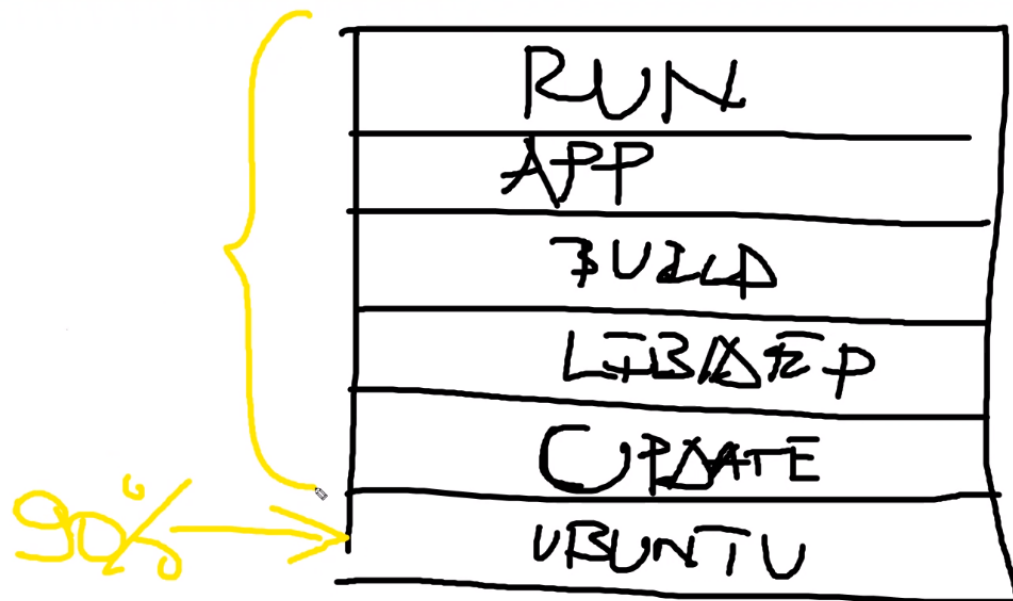
`docker-compose down -v` supprime tout

`docker-compose up -d` pour lancer le docker compose

site où on peut mettre le docker run et qui nous donne le docker-compose :

<https://www.composerize.com/>

Aspect sécurité



90% des problèmes qui viennent de la couche la plus basse

<https://github.com/quay/clair>

liste des image docker et de leur faille (version open source)

<https://snyk.io/>

idem mais version entreprise

refaire notre dockerfile en externalisant les infos sensibles

créer un fichier `.env` (fichier caché car souvent les fichiers cachés sont dans le gitignore)

```
DB_HOST=postgres
DB_NAME=postgres
DB_USER=toto
DB_PW=toto
```

: le fichier `.env`

voir aussi fichier `env_file` : <https://docs.docker.com/compose/environment-variables/>

le docker-compose :

```

Version: '2'
services:
  odoo:
    container_name: odoo
    image: odoo
    depends_on:
      - db
    ports:
      - "8069:8069"
    environment:
      - USER=${DB_USER}
      - PASSWORD=${DB_PW}
    networks:
      - odoo-network
  db:
    container_name: db
    image: postgres:10
    environment:
      - POSTGRES_DB=${DB_NAME}
      - POSTGRES_PASSWORD=${DB_PW}
      - POSTGRES_USER=${DB_USER}
    networks:
      - odoo-network
    volumes:
      - pgdata:/var/lib/postgresql/data

networks:
  odoo-network:
    driver: bridge

volumes:
  pgdata:
    name: pgdata

```

créer un dossier mario et faire un docker compose pour déployer mario

```

version: '2'
services:
  docker-supermario:
    container_name: mario
    ports:
      - '8600:8080'
    image: pengbai/docker-supermario

```

Kubernetes

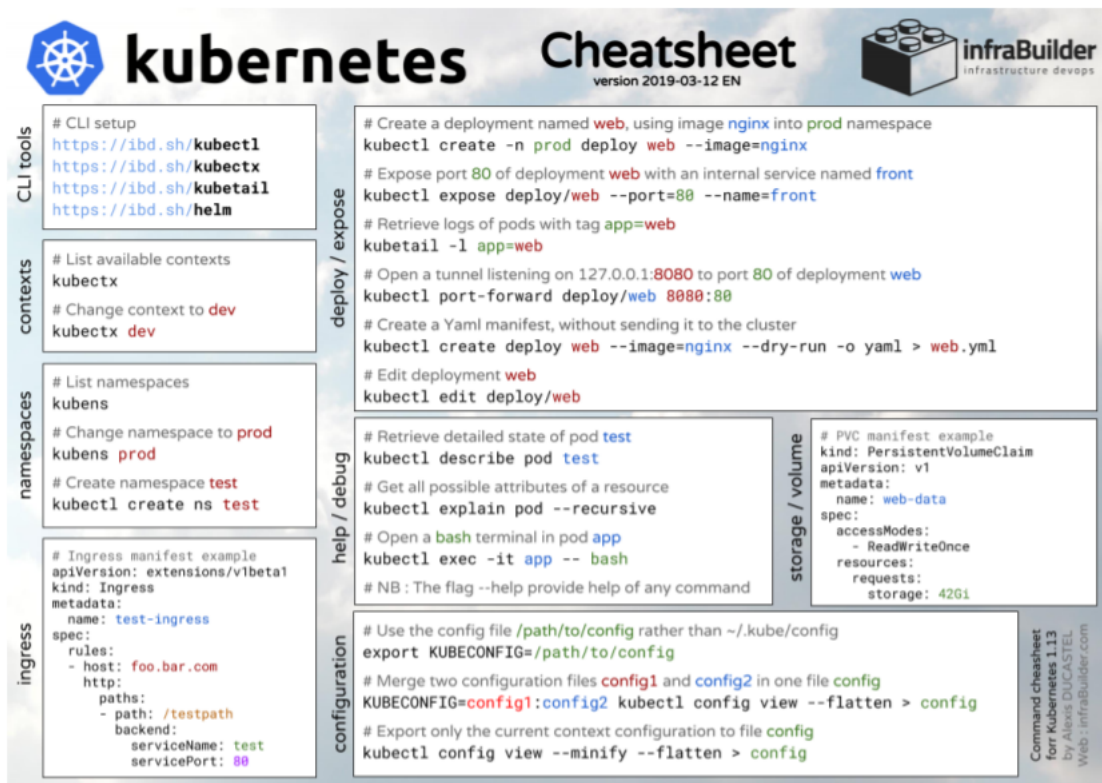
Documentation

<https://kubernetes.io/docs/home/>

doc kubernetes (il existe des exemples dans la doc, ne pas hésiter à copier coller les exemples pour aller plus vite)

Kubernetes fait partie de la famille des orchestrateurs permettant de gérer des cluster de conteneurs (il existe aussi docker swarm)

<https://landscape.cncf.io/>



The image is a 'Kubernetes Cheatsheet' by infraBuilder, version 2019-03-12 EN. It is organized into a grid of categories on the left: CLI tools, contexts, namespaces, ingress, deploy / expose, help / debug, configuration, and storage / volume. Each category contains a box with relevant commands and examples. For example, under 'CLI tools', it lists commands for setting up the CLI, installing Helm, and installing Kubernetes. Under 'deploy / expose', it shows how to create a deployment, expose it, and retrieve logs. The 'ingress' section includes an example manifest for an Ingress resource. The 'storage / volume' section shows a PVC manifest example. The bottom right corner includes a command to download the cheatsheet and credits to Alexis DUCASTEL.

```
kubectl explain pods
```

en gros c'est le man de kubernetes

Mise en place de Minikube

Minikube permet de mettre le master et les worker sur un même noeud (parfait pour formation ou usage perso mais pas du tout pour la prod)

commande qui permet de déployer kubernetes (avec minikube)

```
minikube start --driver=docker
```

```
docker exec minikube docker ps
```

permet de voir les conteneurs déployés dans le conteneur minikube

(d'abord installer docker puis minikube)

<https://github.com/diranetafen/cursus-devops/blob/master/stack/k8s-minikube-stack.yml#L60>

(ligne 60 a 78 installation de kubernetes)

```
kubectl get nodes
```

permet de voir les noeuds (ici on a Ready donc tout va bien)

Element min manipulable par kub : Pod (pouvant contenir plusieurs app donc 1 a plusieurs conteneurs) proscrit d'avoir plusieurs conteneur pour un pod (même si des fois pas le choix) plutôt conseillé est de faire 3 pod pour 3 conteneurs (mieux pour scalability) on peut avoir des volumes dans les pod mais aussi hors des pods

Replicaset : objet kub qui permet de définir le nb d'instance d'un pod, s'assure qu'on a tjrs le nombre de pod indiqué. On peut modifier les paramètres pour qu'il déploie auto plus de pod si un tombe il le redéploie

Deployment : objet de haut niveau. gère les réplicats pour les montés en version.

Conseillé de travailler directement sur le deployment pour gérer le reste

les noeuds sont des serveurs où la charge est distribué

Création et accès à un pod

Mode déclaratif

sur la vm kub, dans le /home/centos/labs/dossier pour chaque tp la on est sur tp-2 et c'est à l'intérieur

↳ Ecrivez un manifest pod.yml pour déployer un pod avec l'image mmumshad/simple-webapp-color en précisant que la color souhaitée est la rouge

name: simple-webapp-color

app: web

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: web
    name: simple-webapp-color
spec:
  containers:
    - name: simple-webapp-color
```

```

image: mmumshad/simple-webapp-color
ports:
  - containerPort: 8080

env:
  - name: APP_COLOR
    value: red

```

commande pour lancer mon pod

```
kubectl apply -f pod.yml
```

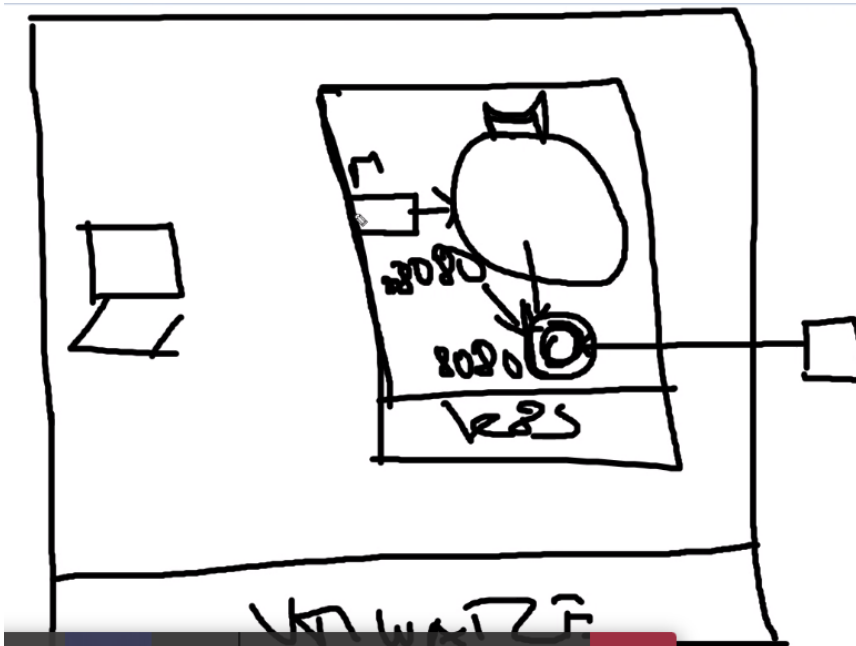
affiche les pods

```
kubectl get pod
```

permet d'avoir la description de l'objet souhaité

```
kubectl describe pod simple-webapp-color
```

Méthode du port forward:



```
kubectl port-forward simple-webapp-color 8080:8080 --address 0.0.0.0
```

→ tout trafic qui arrive sur le 8080 tu le renvoie sur le 8080 du pod

```
curl 127.0.0.1:8080
```

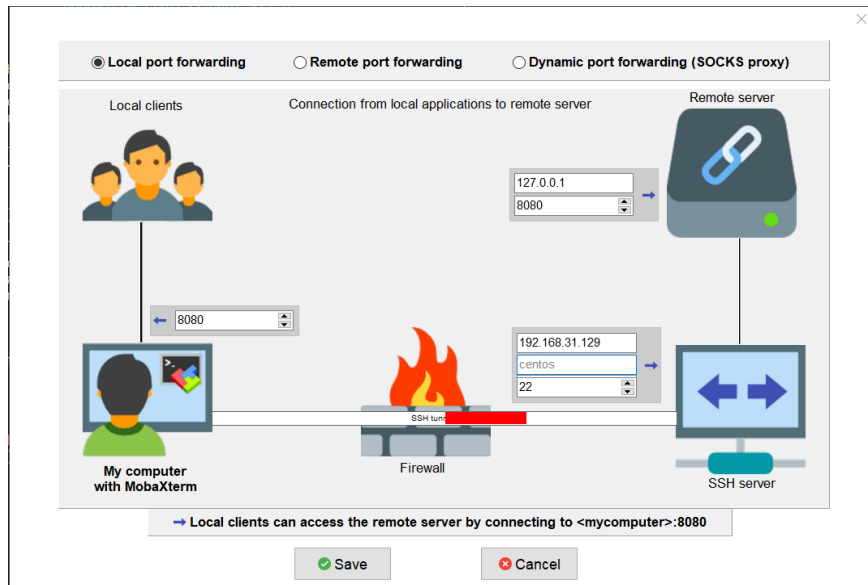
requête qui permet de requêter notre page web

pour y accéder via un navigateur on a besoin d'un tunnel pour tromper

Mise en place du tunnel (tunneling sur mobaxtream)

la requête sur le navigateur sera après 127.0.0.1:8080

bien penser à start le tunnel



la on lui dit que la requête vient de lui même

Tout ça est pour contourner une limite de minikub normalement il n'y aura pas ce pb sur kubernetes en entreprise
Manip importante à connaître quand même

Mode impératif

```
kubectl run --image=mmumshad/simple-webapp-color --env="APP_COLOR=red"
--restart=Never simple-webapp-color
```

Création et gestion d'un deployment

Mode déclaratif

Ecrivez un manifest nginx-deployment.yml pour déployer 2 replicas d'un pod nginx (en version 1.18.0)
nom de l'objet = nom de la racine du fichier
labels:

app: nginx

nginx-deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
```

```

replicas: 2
selector:
  matchLabels:
    app: nginx
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
    - name: nginx
      image: nginx:1.18.0
      ports:
      - containerPort: 80

```

```

kubectl get deployment
kubectl get replicaset

```

affiche la liste des deployment et replicaset

pour supprimer un déploiement fait avec un fichier .yaml:

```

kubectl delete -f pod.yaml

```

Modifiez le fichier nginx-deployment.yml afin d'utiliser l'image nginx en version latest, appliquer la modification (kubectl apply)
Que se passe-t'il ? Combien de replicaset avez-vous ? Quelle est l'image utilisée par le replicaset en cours d'utilisation ?

résultat après :

```

[centos@localhost tp-2]$ kubectl get replicaset
NAME                                DESIRED   CURRENT   READY   AGE
nginx-deployment-67dfd6c8f9         2         2         2       8m24s
nginx-deployment-845d4d9dff         1         1         0       24s
[centos@localhost tp-2]$ kubectl get po
NAME                                READY     STATUS    RESTARTS   AGE
nginx-deployment-845d4d9dff-8dvmk   1/1       Running   0          43s
nginx-deployment-845d4d9dff-cdmj2   1/1       Running   0          12s
[centos@localhost tp-2]$ kubectl get replicaset
NAME                                DESIRED   CURRENT   READY   AGE
nginx-deployment-67dfd6c8f9         0         0         0       8m48s
nginx-deployment-845d4d9dff         2         2         2       48s
[centos@localhost tp-2]$ kubectl get deployment
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment 2/2     2            2           8m54s

```

l'ancien replicaset n'a plus rien et un nouveau replica a été créé pour la montée en version
pour voir l'évolution des pod en temps réel

```

kubectl get po -w

```

permet de voir plus d'info sur les replicaset

```
kubectl get replicaset -o wide
```

supprimer un pod

```
kubectl delete po <nom du pod>
```

permet de voir l'historique des versions

```
kubectl rollout history deployment/nginx-deployment
```

permet de voir les détails de la révision 1

```
kubectl rollout history deployment/nginx-deployment --revision=1
```

basculement sur la version précédente (ici version 1)

```
kubectl rollout undo deployment/nginx-deployment --to-revision=1
```

si on n'indique pas --to-revision il remet la version précédente

le rollout permet de définir la release que l'on veut utiliser

Mode impératif

création du deployment (par défaut un seul replicaset)

```
kubectl create deployment --image=nginx:1.18.0 nginx-deployment
```

changement du nombre de replicaset

```
kubectl scale --replicas=2 deployment/nginx-deployment
```

pour changer la version

```
kubectl set image deployment/nginx-deployment nginx=nginx:latest
```

idem mais de façon plus générique

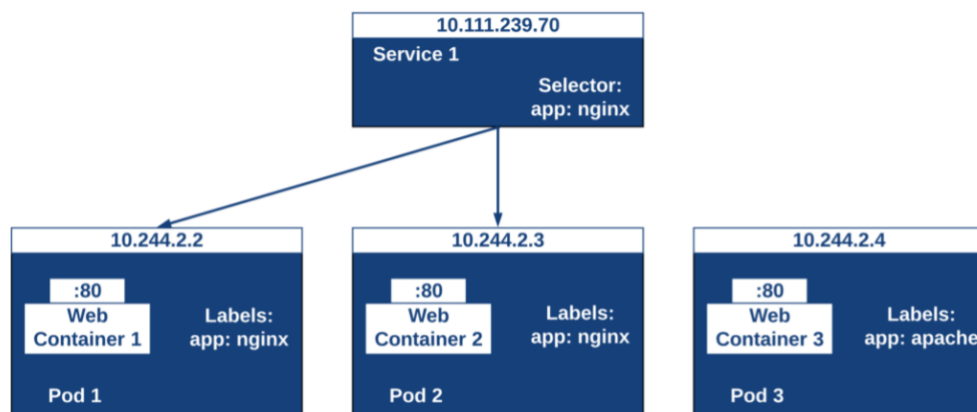
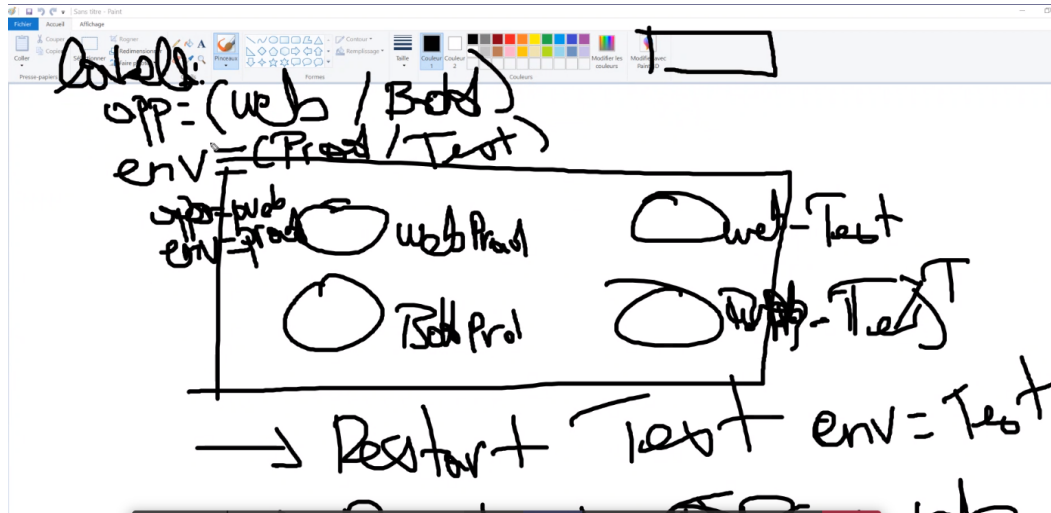
```
kubectl set image deployment/nginx-deployment <nom du  
conteneur>=nginx:latest
```

On obtient le nom du conteneur en faisant

```
kubectl describe deployment/nginx-deployment
```

Gestion du réseau

Label et selector



Une ressource parent peut récupérer ces enfants grâce aux labels et au selector

```
kubectl get pods -l app=nginx
```

je veux les pods dont le label app est nginx

```
kubectl get pods -l 'env in (production)'
```

je veux les pods dont le label env est dans la liste production

Les services

<https://kubernetes.io/docs/concepts/services-networking/service/>

<https://kubernetes.io/docs/concepts/services-networking/connect-applications-service/>

Ils sont chargés d'exposer les pods à l'extérieur ou l'intérieur (les services se moquent du replicaset et deployment)

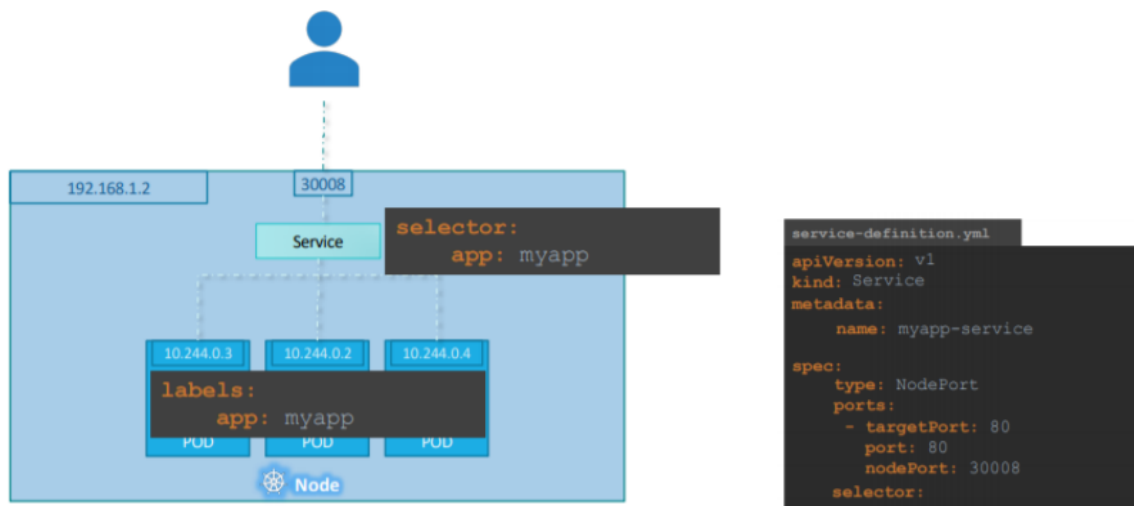
Type values and their behaviors are:

- **ClusterIP**: Exposes the Service on a cluster-internal IP. Choosing this value makes the Service only reachable from within the cluster. This is the default `ServiceType`.
- **NodePort**: Exposes the Service on each Node's IP at a static port (the `NodePort`). A `ClusterIP` Service, to which the `NodePort` Service routes, is automatically created. You'll be able to contact the `NodePort` Service, from outside the cluster, by requesting `<NodeIP>:<NodePort>`.
- **LoadBalancer**: Exposes the Service externally using a cloud provider's load balancer. `NodePort` and `ClusterIP` Services, to which the external load balancer routes, are automatically created.
- **ExternalName**: Maps the Service to the contents of the `externalName` field (e.g. `foo.bar.example.com`), by returning a `CNAME` record with its value. No proxying of any kind is set up.

type nodeport

<https://kubernetes.io/docs/concepts/services-networking/service/#nodeport>

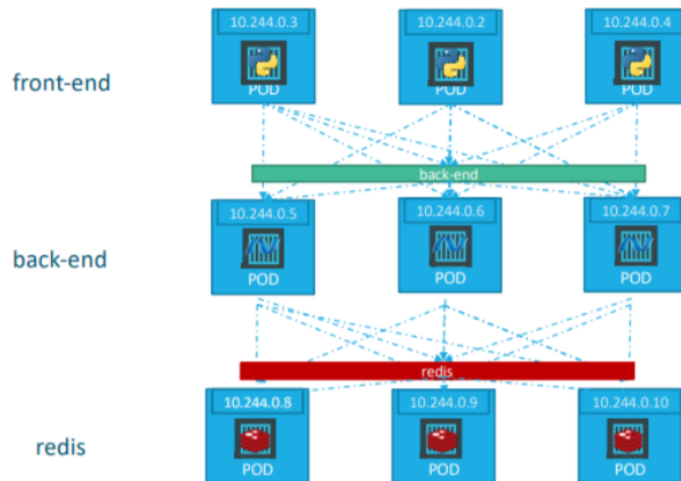
ils permettent de donner accès directement depuis l'ext à nos app (il affiche son port directement à l'ext et load balance sur les pods) il connait ses pods à gérer grâce à son selector



Utilisé surtout en phase de test (port externe compris entre 30000 et 32767)

type ClusterIP:

permet de faire communiquer entre pods en interne (par exemple le front end avec le back end)



exemple

```

service-definition.yml
apiVersion: v1
kind: Service
metadata:
  name: back-end
spec:
  type: ClusterIP
  ports:
    - targetPort: 80
      port: 80
  selector:

```

```

pod-definition.yml
> kubectl create -f service-definition.yml
service "back-end" created

> kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP    16d
back-end     ClusterIP   10.106.127.123 <none>        80/TCP     2m

```

```

app: myapp
type: back-end
spec:
  containers:
    - name: nginx-container
      image: nginx

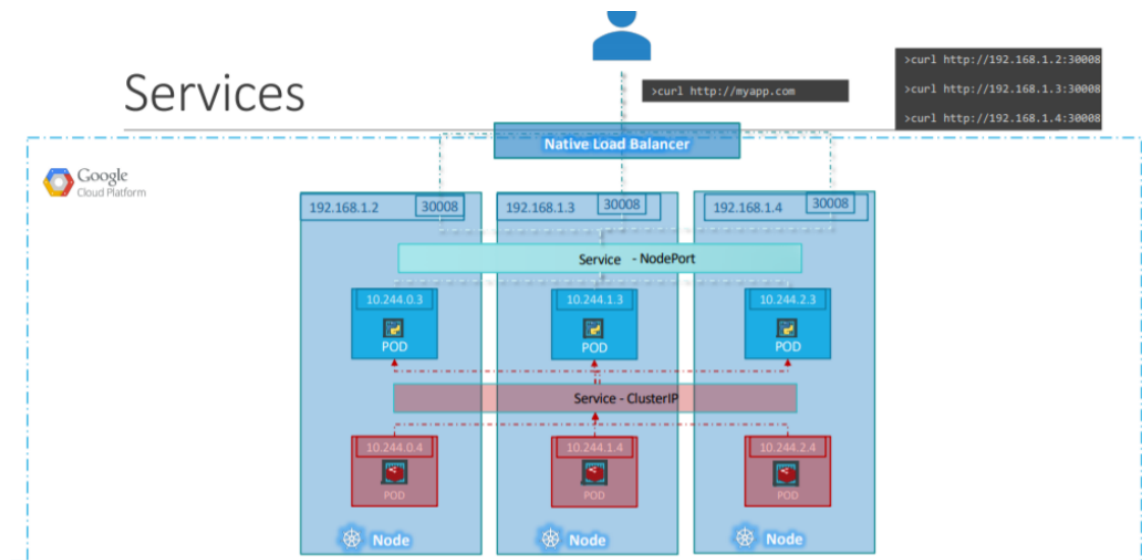
```

exemple de fichier de définitions

type loadbalancer

<https://kubernetes.io/fr/docs/concepts/services-networking/service/#loadbalancer>

Le trafic provenant du load balancer externe est dirigé vers les Pods backend. Le fournisseur de cloud décide de la répartition de la charge.



Création d'un load balancer externe :

<https://kubernetes.io/docs/tasks/access-application-cluster/create-external-load-balancer/>

```
service-definition.yml
apiVersion: v1
kind: Service
metadata:
  name: front-end
spec:
  type: NodeBalancer
  ports:
    - targetPort: 80
      port: 80
  selector:
    app: myapp
    type: front-end
```

```
> kubectl create -f service-definition.yml
service "front-end" created
```

```
> kubectl get services
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	16d
front-end	LoadBalancer	10.106.127.123	<Pending>	80/TCP	2m

Namespace

Cas particulier pour des équipes différentes

Env isoler ou les ressources de chaque équipe seront créées. On peut également y appliquer des droits d'accès.

Cas particulier d'un projet

séparer dev prod et recette par ex pour que ces différents éléments n'empiète pas les uns sur les autres

TP

- Ecrivez un manifest namespace.yml qui crée un namespace nommé production et lancez la création de ce namespace à partir du manifest

```
apiVersion: v1
kind: Namespace
metadata:
  name: production
```

default	Active	22h
kube-node-lease	Active	22h
kube-public	Active	22h
kube-system	Active	22h
production	Active	21s

default : le namespace de base (quand on fait un get node on va voir dans ce namespace)

kube-node-lease: This namespace for the lease objects associated with each node which improves the performance of the node heartbeats as the cluster scales.

kube public tout le monde peut voir mais seul le proprio peut supprimer

kube-system: là où se trouvent les composants nécessaires au fonctionnement de kube

Pour voir tout ce qui est déployé dans un namespace

```
kubectl get all -n <namespace>
```

Ecrivez un manifest pod-red.yml pour déployer un pod avec l'image mmumshad/simple-webapp-color en précisant que la couleur souhaitée est la rouge (red), le pod doit posséder le label « app: web »

Ecrivez un manifest pod-blue.yml pour déployer un pod avec l'image mmumshad/simple-webapp-color en précisant que la couleur souhaitée est la bleue (blue) le pod doit posséder le label « app: web »

pod-red.yml (avec le namespace)

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: web
  name: pod-red
  namespace: production
spec:
  containers:
  - name: simple-webapp-color
    image: mmumshad/simple-webapp-color
    ports:
      - containerPort: 8080
    env:
      - name: APP_COLOR
        value: red
```

pod-blue.yml (avec le namespace)


```

apiVersion: v1
kind: Pod
metadata:
  labels:
    app: web
    name: pod-blue
    namespace: production
spec:
  containers:
  - name: simple-webapp-color
    image: mmumshad/simple-webapp-color
    ports:
      - containerPort: 8080
    env:
      - name: APP_COLOR
        value: blue

```

lancer les pods si on a pas précisé le namespace dans le manifeste

```

kubectl apply -f pod-red.yml --namespace production
kubectl apply -f pod-blue.yml --namespace production

```

Visualiser les pods dans le namespace prod

```

kubectl get pod --namespace=production
kubectl get po -n production

```

Ecrivez un manifest service-nodeport-web.yml qui permettra d'exposer les pods via un service de type node port, le nodeport devra être le 30008 et les target les ports 8080 de nos pods dont le label est « app: web »

```

apiVersion: v1
kind: Service
metadata:
  name: service-nodeport-web
  namespace: production
spec:
  type: NodePort
  selector:
    app: web
  ports:
    - port: 8080 ## le port sur lequel les containers écoutent
      targetPort: 8080 ## port exposé en interne dans le cluster
      nodePort: 30008 ## Port d'accès (limité 30000-32767 pour nodePort)

```

manifeste service-nodeport-web.yml

permet de voir les node relia au service

```

kubectl describe svc service-nodeport-web -n production

```

s'il y a une ou plusieurs adresse IP (en fonction du nombre de pod écouté par le service)
sur la ligne EndPoint c'est good normalement

Une solution pour déployer un wordpress

architecture souhaite :

backend : un deployment avec un replicas pod mysql

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
  labels:
    app: mysql
  namespace: wordpress
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:5.7
          ports:
            - containerPort: 3306    ### voir la doc
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: root
            - name: MYSQL_USER
              value: toto
            - name: MYSQL_PASSWORD
              value: toto
            - name: MYSQL_DATABASE
              value: wordpress
```

front end : un deployment avec un replicas pod wordpress

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-deployment
```

```

labels:
  app: wordpress
  namespace: wordpress
spec:
  replicas: 1
  selector:
    matchLabels:
      app: wordpress
  template:
    metadata:
      labels:
        app: wordpress
    spec:
      containers:
        - name: wordpress
          image: wordpress
          ports:
            - containerPort: 80
          env:
            - name: WORDPRESS_DB_USER
              value: toto
            - name: WORDPRESS_DB_HOST
              value: mysql-service
            - name: WORDPRESS_DB_PASSWORD
              value: toto
            - name: WORDPRESS_DB_NAME
              value: wordpress

```

mettre en place un clusterIP pour joindre le backend depuis le front

```

apiVersion: v1
kind: Service
metadata:
  name: mysql-service
  namespace: wordpress
spec:
  type: ClusterIP
  ports:
    - targetPort: 3306
      port: 3306
  selector:
    app: mysql

```

un service nodeport pour pouvoir exposer a l'ext sur le port 30080

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress-service
  namespace: wordpress
spec:
  type: NodePort
  selector:
    app: wordpress
  ports:
    - port: 80 ## le port sur lequel les containers écoutent
      targetPort: 80 ## port exposé en interne dans le cluster
      nodePort: 30080 ## Port d'accès (limité 30000-32767 pour nodePort)
```

namespace: wordpress

```
apiVersion: v1
kind: Namespace
metadata:
  name: wordpress
```

on va devoir respecter ce qu'on a fait avec docker au niveau des variables d'env

A chaque fois que l'on crée un service, kubernetes nous crée un DNS :

<nom du service>.<nom namespace>.svc.cluster.local

si on est dans le même namespace on peut se contenter de <nom du service>

Secret

problème de sécurité les passwords sont en clair

<https://kubernetes.io/docs/concepts/configuration/secret/>

<https://chrisshort.net/the-secret-to-kubernetes-secrets/>

supprime tout ce qui est déployé dans le fichier courant

```
kubectl delete -f .
```

mysql-secret.yml

```
apiVersion: v1
kind: Secret
metadata:
```

```
name: mysql-secret
namespace: wordpress
type: Opaque
stringData: ### avec stringData les infos sont en clair et avec data les
infos sont encodés en base64
  MYSQL_ROOT_PASSWORD: root
  MYSQL_USER: toto
  MYSQL_PASSWORD: toto
  MYSQL_DATABASE: wordpress
```

Utilisation des secrets dans des variables d'environnements :

<https://kubernetes.io/docs/concepts/configuration/secret/#using-secrets-as-environment-variables>

mysql-deployment avec secret

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
  labels:
    app: mysql
  namespace: wordpress
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:5.7
          ports:
            - containerPort: 3306    ### voir la doc
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-secret
                  key: MYSQL_ROOT_PASSWORD
            - name: MYSQL_USER
              valueFrom:
                secretKeyRef:
```

```
      name: mysql-secret
      key: MYSQL_USER
-   name: MYSQL_PASSWORD
    valueFrom:
      secretKeyRef:
        name: mysql-secret
        key: MYSQL_PASSWORD
-   name: MYSQL_DATABASE
    valueFrom:
      secretKeyRef:
        name: mysql-secret
        key: MYSQL_DATABASE
```

wordpress-deployment avec secret (on réutilise les secrets de mysql-secret car c'est les mêmes que pour mysql-deployment)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-deployment
  labels:
    app: wordpress
  namespace: wordpress
spec:
  replicas: 1
  selector:
    matchLabels:
      app: wordpress
  template:
    metadata:
      labels:
        app: wordpress
    spec:
      containers:
        - name: wordpress
          image: wordpress
          ports:
            - containerPort: 80
          env:
            - name: WORDPRESS_DB_USER
              valueFrom:
                secretKeyRef:
                  name: mysql-secret
                  key: MYSQL_USER
```

```

- name: WORDPRESS_DB_HOST
  value: mysql-service
- name: WORDPRESS_DB_PASSWORD
  valueFrom:
    secretKeyRef:
      name: mysql-secret
      key: MYSQL_PASSWORD
- name: WORDPRESS_DB_NAME
  valueFrom:
    secretKeyRef:
      name: mysql-secret
      key: MYSQL_DATABASE

```

Plus d'info sur les secrets:

<https://engineering.bitnami.com/articles/sealed-secrets.html>

Gestion du stockage : les volumes

dans un fichier de deployment le volume est au même niveau que containers

```

apiVersion: v1
kind: Pod
metadata:
  name: random-number-generator
spec:
  containers:
  - image: alpine
    name: alpine
    command: ["/bin/sh", "-c"]
    args: ["shuf -i 0-100 -n 1 >> /opt/number.out;"]
    volumeMounts:
    - mountPath: /opt
      name: data-volume

  volumes:
  - name: data-volume
    hostPath:
      path: /data
      type: Directory

```

hostpath : type de volume qui se crée dans notre machine

il y a des limites si on perd le pod et qu'il y a plusieurs node on perd les données
utilisé seulement en single node. Possibilité de le retrouver si les données sont en réseaux
(jonglerie des entreprises vis à vis de ça)

D'autres solution comme NFS ou d'autres outils de partage réseaux (voir diapo).

Utilisation de volume persistant : (pour faciliter le boulot de tout le monde et faire plus d'automatisation) ops mettent en place le stockage

<https://kubernetes.io/docs/concepts/storage/persistent-volumes/>

pv-definition.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-vol1
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 1Gi
  awsElasticBlockStore:
    volumeID: <volume-id>
    fsType: ext4
```

Persistent volume claim : demande de place de stockage :

<https://kubernetes.io/docs/concepts/storage/volumes/>

pvc-definition.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

Sur de petites infra on va plutôt aller sur du stockage de type volume et quand on va augmenter d'échelle on va passer sur du persistentVolume

Faire un stockage persistant de type hostpath sur mysql
mettre l'option qui fait que si le dossier de stockage n'existe pas on le crée

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
  labels:
    app: mysql
  namespace: wordpress
spec:
  replicas: 1
```



```

selector:
  matchLabels:
    app: mysql
template:
  metadata:
    labels:
      app: mysql
  spec:
    containers:
      - name: mysql
        image: mysql:5.7
        ports:
          - containerPort: 3306    ### voir la doc
        env:
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysql-secret
                key: MYSQL_ROOT_PASSWORD
          - name: MYSQL_USER
            valueFrom:
              secretKeyRef:
                name: mysql-secret
                key: MYSQL_USER
          - name: MYSQL_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysql-secret
                key: MYSQL_PASSWORD
          - name: MYSQL_DATABASE
            valueFrom:
              secretKeyRef:
                name: mysql-secret
                key: MYSQL_DATABASE
        volumeMounts:
          - mountPath: /var/lib/mysql
            name: data-volume
    volumes:
      - name: data-volume
        hostPath:
          path: /data
          type: DirectoryOrCreate

```

Permet de voir nos datas dans notre architecture un peu particulière

```
docker exec -it minikube ls /data
```

Le client veut passer à l'échelle
donc demande de création d'un PV et d'un PVC pour stocker les données
créer un PV mysql-pv.yml: storageClassName: manual taille=1G hostpath: /data-pv
objet de type cluster n'ont pas besoin de namespace (apparemment les persistent volume
n'en n'ont pas besoin)

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv
  namespace: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 1Gi
  storageClassName: manual
  hostPath:
    path: /data-pv
    type: DirectoryOrCreate
```

créer un pvc storageClassName: manual taille=100M même accessMode que sur le pv

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pvc
  namespace: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
  storageClassName: manual
```

changement fait pour passer en stockage via volume permanent

```
volumeMounts:
  - mountPath: /var/lib/mysql
    name: data-volume
volumes:
  - name: data-volume
    persistentVolumeClaim:
      claimName: mysql-pvc
```

Git (à compléter)

Premier push dans un nouveau repo

```
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/antoine-bouquet/test.git
git push -u origin main
```

<https://about.gitlab.com/images/press/git-cheat-sheet.pdf>