

REDS DOCKER ET KUBERNETES

Pratique des Architectures GPU à base de conteneurs

Thursday 19th November, 2020

Christophe Boudier

Pratique des Architectures GPU à base de conteneurs

Objectifs

Comprendre les conteneurs et infrastructures

- Docker et infrastructures
- Docker en pratique
- Usage avancé de docker

Environnement BigData et GPU

- Masse de données et configuration Nvidia

Les conteneurs et le cas Docker

Conteneurs et infrastructures

PLAN

- Les conteneurs : principes, objectifs et solutions
 - Les conteneurs et le cas Docker
 - Conteneurs et infrastructures
- Docker en pratique : les outils de base
 - Conteneurs et images
 - le client docker
- Exemples d'architectures et concepts avancés
 - Application multi-conteneurs
 - Le réseau Docker
 - Orchestration avec Docker Compose
 - CLOUD style amazon

Les conteneurs : principes, objectifs et solutions

- Les conteneurs et le cas Docker
 - le contenu du conteneur, c'est-à-dire le code et ses dépendances (jusqu'au niveau de l'OS), est de la responsabilité du développeur
 - la gestion du conteneur et les dépendances que celui-ci va entretenir avec l'infrastructure (soit le stockage, le réseau et la puissance de calcul) sont de la responsabilité de l'exploitant.

Les conteneurs et le cas Docker

différence avec VM : VM isolation élevée poids important Beaucoup d'éléments hardware et software Conteneurs isolation très léger nb de conteneurs > aux vm

- linux cgroups

- CGroups (pour Control Groups) permet de partitionner les ressources d'un hôte (processeur, mémoire, accès au réseau ou à d'autres terminaux).

- linux namespaces

- Les Namespaces permettent de faire en sorte que des processus ne voient pas les ressources utilisées par d'autres.

Les conteneurs et le cas Docker

- notion d'image
 - les images : blocs réutilisables et échangeables Ces images sont donc des archives qui peuvent être échangées entre plusieurs hôtes, mais aussi être réutilisées.
 - ufs overlay
 - docker hub registry
 - copy on write persistance et volume

Conteneurs et infrastructures

IaaS Infrastructure as a Service : les solutions IaaS offrent aux applications conditionnées sous la forme d'images de machines virtuelles des services d'infrastructure, comme :

- de la puissance de calcul, via un environnement d'exécution pour une machine virtuelle basée sur un hyperviseur : VMWare, Xen, KVM, etc. ;
- du stockage qui peut se présenter sous différentes formes selon les niveaux de performance et de résilience requis ;
- des services réseau : DHCP, pare-feu, fail-over, etc. ;
- des services de sécurité : authentification, gestion des mots de passe et autres secrets.

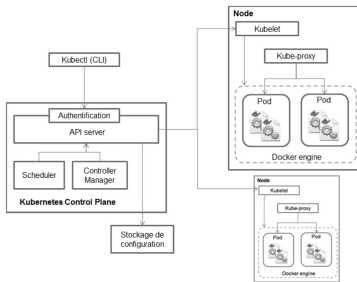
Conteneurs et infrastructures

CaaS Containeur as a Service :

- Les approches CaaS visent à aborder en premier lieu la problématique du déploiement d'applications. La problématique de la virtualisation des ressources matérielles est en fait secondaire
 - le conteneur logiciel est scellé.
 - L'exploitant joue le rôle de transporteur.
 - Il choisit la manière dont le conteneur va être exécuté sans l'altérer.
- Les solutions CaaS associent au déploiement des règles de gestion.

Conteneurs et infrastructures

- kubernetes : Kubernetes suit une architecture maître-esclave (master/slave) :
 - Le contrôleur central maître , kubernetes control plane sert à gérer notre cluster Il est composé de 3 processus : kube-apiserver, kube-controller-manager et kube-scheduler.
 - Les node (slave) Kubernetes sont les machines qui vont exécuter nos applications. Chaque node est composé de 2 processus : kubelet et kube-proxy.



Conteneurs et infrastructures

- le Kubernetes Control Plane est en fait constitué de plusieurs sous composants
 - un serveur d'API REST « API server », qui est notamment utilisé par Kubectl, la ligne de commande Kubernetes qui permet de piloter tous les composants de l'architecture ;
 - le scheduler, qui est utilisé pour provisionner sur différents nœuds (node) de nouveaux pools de conteneurs en fonction de la topologie, de l'usage des ressources ou de règles d'affinité plus ou moins complexes (sur le modèle des stratégies Swarm)
 - le controller manager, qui exécute les boucles de contrôle du système ou controllers (à une fréquence déterminée) pour vérifier que les règles de gestion programmées sont respectées (nombre d'instances de certains conteneurs, par exemple).

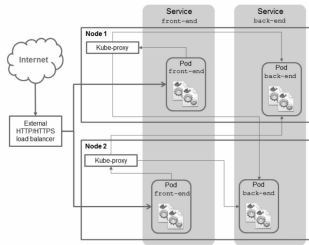
Conteneurs et infrastructures

- Le kube-proxy est un proxy répartiteur de charge (load balancer) qui gère les règles d'accès aux services
- Kubernetes organise les applications à base de conteneurs selon une structure et un modèle réseau propre auxquels il faut se soumettre
- Un pod est un groupe de conteneurs qui seront toujours co-localisés et provisionnés ensemble.
- Ce sont des conteneurs qui sont liés et qui offrent une fonction (une instance de micro-service)

Conteneurs et infrastructures

- tous les conteneurs d'un pod vont se trouver dans une sorte de même machine logique : ils partagent les mêmes volumes (stockage partagé) ; un volume est détruit en même temps que le pod auquel il appartient. Pour stocker des données de manière permanente, il est préférable d'utiliser un PersistentVolume ou des services de stockage externe tel que AWS S3. Kubernetes définit un service (ou micro-service) comme un ensemble de pods associés à des règles d'accès, de réplication et de répartition de charge spécifiques. la notion de service est interne au cluster Kubernetes.

Conteneurs et infrastructures



Docker en pratique:les outils de base

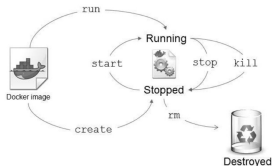
■ Conteneurs et images

- créer, démarrer, se connecter à un conteneur
- Le démon Docker écoute sur un socket Unix à l'adresse `/var/run/docker.sock`. Le client Docker utilise HTTP pour envoyer des commandes à ce socket qui sont ensuite transmises au démon, aussi via HTTP.



Docker en pratique:les outils de base

Cycle de vie d'un conteneur :



- `docker run -d -p 8000:80 name Webserver nginx`
 - `docker ps`
 - `docker ps -a`
 - `/var/lib/docker/containers/`
 - `docker start stop rm`
 - `docker exec -t -i Webserver /bin/bash`

Docker en pratique:les outils de base

■ Les volumes

- `docker run -p 8000:80 --name Webserver -d -v /usr/share/nginx/html nginx`
- `docker inspect Webserver |grep -A9 Mounts`
- il est nécessaire de disposer de privilèges étendus pour accéder aux données et pour les modifier
- la destruction/recréation du conteneur entraînera l'allocation d'un nouveau volume différent et les modifications opérées seront perdues.
- `docker run -p 8000:80 --name Webserver -d -v /home/USER/workdir/:/usr/share/nginx/html nginx`
- `docker inspect Webserver |grep -A9 Ports`

Docker en pratique:les outils de base

- gestion des images, dockerfile :
 - docker images
 - docker search
 - docker rmi nginx
 - docker **pull** nginx:1.7
- le propos des Dockerfile:
 - décrire la création d'images de manière formelle.
 - Dockerfile:
 - FROM nginx:1.7
 - COPY index.html /usr/share/nginx/html/index.html
 - contenu du fichier index.html:hello
 - docker build -t nginxhello .
 - docker history nginxhello

le client docker : les variables d'environnement

Nom	Définition
DOCKER_API_VERSION	La version de l'API Docker à utiliser.
DOCKER_CONFIG	L'emplacement des fichiers de configuration Docker.
DOCKER_CERT_PATH	L'emplacement des certificats liés à l'authentification.
DOCKER_DRIVER	Le pilote graphique à utiliser.
DOCKER_HOST	Le démon Docker à utiliser.
DOCKER_NOWARN_KERNEL_VERSION	Si le paramètre est activé, cela empêche l'affichage d'avertissements liés au fait que la version du noyau Linux n'est pas compatible avec Docker.
DOCKER_RAMDISK	Si le paramètre est activé, alors Docker fonctionne avec un utilisateur en mémoire RAM (l'utilisateur est sauvé en RAM et non sur le disque dur).
DOCKER_TLS_VERIFY	Si le paramètre est activé, alors Docker ne permet des connexions distantes qu'avec le protocole de sécurisation TLS.
DOCKER_CONTENT_TRUST	Si le paramètre est activé, alors Docker utilise Docker Content Trust pour signer et vérifier les images. Ce comportement est équivalent à l'option <code>--disable-content-trust=false</code> lors de l'utilisation des commandes <code>build</code> , <code>create</code> , <code>pull</code> , <code>push</code> et <code>run</code> . Nous le verrons en détail dans le chapitre 8.
DOCKER_CONTENT_TRUST_SERVER	L'URL du serveur Notary à utiliser lorsque la variable d'environnement <code>DOCKER_CONTENT_TRUST</code> est activée.
DOCKER_TMPDIR	L'emplacement des fichiers temporaires Docker.

le client docker

■ les options des commandes docker :

- `docker run -i` `docker run -detach`
- `docker run -m 5M` `docker run --memory 5M`
- `docker info` `docker stats` `docker events` `docker inspect`

■ Création et exécution

- `create` et `run` sont des commandes soeurs dans la mesure où un `run` correspond à un `create` suivi d'un `start`.
- `docker create -name=webserver nginx`, Crée un conteneur `nginx` prêt à être démarré
- `docker start Webserver`, Démarre le conteneur précédemment créé
- `docker -t -i`, la plupart du temps combinés, qui permettent d'ouvrir un pseudo terminal sur le conteneur. `docker logs`, `docker exec -t -i webserver /bin/bash`, `docker attach --sig-proxy=false webserver` `docker cp`

commandes relatives aux images :

- `docker build -t name`
- `FROM` qui vous permet de définir l'image source ;
- `RUN` qui vous permet d'exécuter des commandes dans votre conteneur ;
- `ADD` qui vous permet d'ajouter des fichiers dans votre conteneur ;
- `WORKDIR` qui vous permet de définir votre répertoire de travail ;
- `EXPOSE` qui permet de définir les ports d'écoute par défaut ;
- `VOLUME` qui permet de définir les volumes utilisables ;
- `CMD` qui permet de définir la commande par défaut lors de l'exécution de vos conteneurs Docker.

docker network et docker volumes

Commande	Objet
docker network create	Permet de créer un réseau Docker. Cette commande prend notamment en paramètre <code>--driver</code> qui permet de spécifier le type de réseau souhaité (par défaut bridge).
docker network connect	Cette commande permet de connecter un conteneur à un réseau.
docker network disconnect	Cette commande permet de déconnecter un conteneur d'un réseau.
docker network inspect	Une fonction d'inspection du réseau dont nous verrons l'utilité dans le chapitre 9.
docker network ls	Une commande qui liste les réseaux disponibles. Par défaut, trois réseaux sont systématiquement définis : none, host et bridge.
docker network rm	La commande qui permet de détruire des réseaux existants (sauf évidemment nos trois réseaux prédéfinis).

Commande	Objet
docker volume create	Une commande qui permet de créer un volume qu'il sera ensuite possible d'associer à un ou plusieurs conteneurs. Cette commande prend en paramètre <code>--driver</code> qui permet de spécifier le driver utilisé pour ce volume. Il existe aujourd'hui des plugins Docker permettant de s'appuyer sur des systèmes de stockage tiers (en lieu et place d'une simple persistance sur l'hôte).
docker volume inspect	Une commande pour visualiser des métadonnées relatives à un volume.
docker volume ls	La commande qui permet de lister les volumes disponibles.
docker volume rm	La commande qui permet d'effacer des volumes. Attention, une fois détruit, les données associées à un volume sont perdues définitivement. Il n'est cependant pas possible d'effacer un volume utilisé par un conteneur.

- Dockerfile : le descripteur d'une image
- une et une seule instruction FROM avec la version de l'image source
- CMD ENTRYPOINT : Si vous spécifiez uniquement CMD alors docker exécutera cette commande en utilisant le ENTRYPOINT par défaut, à savoir /bin/sh -c . Vous pouvez remplacer soit le point d'entrée, soit la commande lorsque vous démarrez l'image construite. Si vous spécifiez les deux, alors ENTRYPOINT spécifie l'exécutable de votre processus de conteneur et CMD sera fourni comme paramètre de cet exécutable.

Les instructions dockerfile

- EXPOSE décrit les ports du conteneur que ce dernier écoute
- COPY permet d'ajouter un fichier dans l'image. La source du fichier (d'où il provient) doit être disponible à travers la machine qui construira l'image (par exemple, un fichier local ou distant si une connexion distante est possible). Le fichier sera définitivement ajouté dans l'image.
- VOLUME permet de créer un point de montage dans l'image. Ce dernier se référera à un emplacement, appelé volume, dans l'hôte ou dans un autre conteneur.
- ENV permet de créer ou de mettre à jour une ou plusieurs variables d'environnement,
- LABEL permet d'ajouter des métadonnées à une image type nom version
- WORKDIR permet de changer le chemin courant
- ARG permet de définir des variables (appelées arguments) qui sont passées comme paramètres lors de la construction de l'image.

Les instructions dockerfile

```
FROM debian:latest
```

```
MAINTAINER Christophe Boudier "christophe.boudier@lip6.fr"
```

```
RUN apt-get update RUN apt-get install -y python python-pip wget
```

```
RUN pip install Flask
```

```
ADD hello.py /home/hello.py
```

```
WORKDIR /home
```

```
docker build -t "flask:dockerfile" .
```

Execution :

```
docker run -p 5050:5000 flask:dockerfile python hello.py
```

Les instructions dockerfile

```
FROM debian:latest
```

```
LABEL version="1.0" maintainer="Christophe Boudier  
<christophe.boudier@lip6.fr>"
```

```
RUN apt-get update apt-get install -y python python-pip wget pip  
install Flask
```

```
EXPOSE 5050
```

```
VOLUME /home
```

```
WORKDIR /home
```

```
CMD python
```

```
ENTRYPOINT /home/hello.py
```

Execution :

```
docker run -p 5050:5050 -v /data/docker/cours/1/home/:/home  
flask:dockerfile python hello.py
```

registry privé

```
docker search registry
```

```
docker run -d -p 5000:5000 --restart=always --name registry  
registry:latest
```

```
docker pull nginx
```

```
docker tag nginx localhost:5000/mynginx
```

```
docker push localhost:5000/mynginx
```

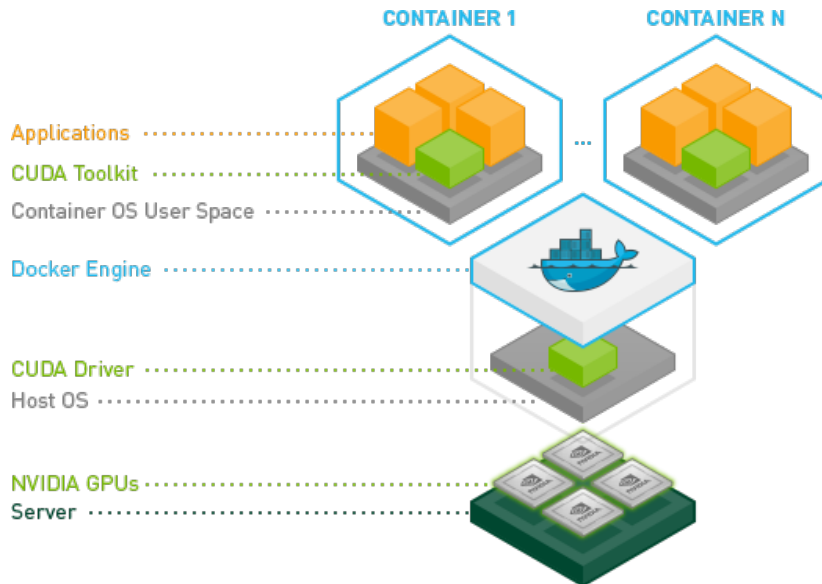
```
docker images |grep nginx
```

```
localhost:5000/mynginx latest 540a289bab6c 3 weeks ago 126MB  
nginx latest 540a289bab6c 3 weeks ago 126MB
```

```
registry tiers nvidia https://ngc.nvidia.com (gratuit)
```

```
docker pull nvidia/cuda
```

docker nvidia cuda



Docker nvidia cuda

```
docker run --runtime=nvidia --rm nvidia/cuda nvidia-smi
```

Définir nvidia comme runtime par défaut

modification dans le fichier /etc/docker/daemon.json

```
"runtimes": {"nvidia": {"path": "nvidia-container-runtime"},  
"runtimeArgs": [], "default-runtime": "nvidia"}
```

```
docker info
```

Runtimes: nvidia runc

Default Runtime: nvidia

```
docker pull nvcr.io/nvidia/pytorch:19.10-py3
```

```
docker run --gpus all -it --rm -v localdir:containerdir  
nvcr.io/nvidia/pytorch:19.10-py3
```

Docker nvidia cuda

nbody graphical sample provided with the CUDA toolkit.

```
docker build -t nbody .
```

```
xhost +si:localuser:root
```

```
docker run --runtime=nvidia -ti --rm -e DISPLAY -v  
/tmp/.X11-unix:/tmp/.X11-unix nbody
```

```
FROM nvidia/cudagl:9.0-base-ubuntu16.04
```

```
ENV NVIDIA_DRIVER_CAPABILITIES NVIDIA_DRIVER_CAPABILITIES,display
```

```
RUN apt-get update apt-get install -y --no-install-recommends  
cuda-samples-CUDA_PKG_VERSION rm -rf /var/lib/apt/lists/*
```

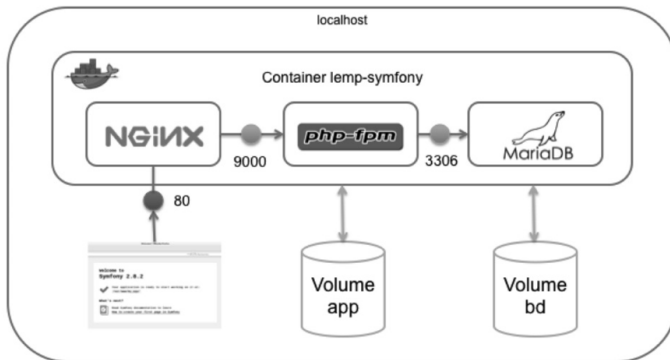
```
WORKDIR /usr/local/cuda/samples/5_Simulations/nbody
```

```
RUN make
```

```
CMD ./nbody
```


Exemples d'architectures et concepts avancés

Application multi-conteneurs :



Orchestration avec Docker Compose

Docker Compose : tous les conteneurs de votre application dans un fichier `docker-compose.yml`

un bloc `services` : qui contient la définition de nos conteneurs.

- pour chaque conteneur, nous avons un paramètre `build` qui permet de spécifier le chemin du répertoire à utiliser comme contexte de création. Docker Compose fabriquera une nouvelle version de notre image si nécessaire.
- nous spécifions aussi le nom de nos conteneurs avec le paramètre `container_name`.
- le paramètre `depends_on` permet de nous assurer que le conteneur X sera disponible lors du démarrage du conteneur Y. Si nous n'avons pas ce paramètre, le conteneur Y s'arrête s'il démarre avant le conteneur X ;
- le paramètre `links` établit un lien entre les conteneurs dans un autre service et instaure une dépendance entre les services comme `depends_on`.

`docker-compose down` `docker-compose up -d` `docker-compose logs X`

Orchestration avec Docker Compose

mysql:

container_name: mysql

image: mariadb:latest

volumes:

- /docker/monsite/bd:/var/lib/mysql

environment:

MYSQL_DATABASE: wpdb

MYSQL_USER: wpdb

MYSQL_PASSWORD: wpdbX2

MYSQL_ROOT_PASSWORD: superroot

restart: unless-stopped

monsite:

container_name: monsite

image: wordpress:latest

volumes:

- /docker/monsite/conf/uploads.ini:/usr/local/etc/php/conf.d/uploads.ini

- /docker/monsite/www/wp-content:/var/www/html/wp-content:rw

links:

- mysql:mysql

ports:

- 8089:80

links:

- mysql

environment:

WORDPRESS_DB_NAME: wpdb

WORDPRESS_DB_USER: wpdb

WORDPRESS_DB_PASSWORD: wpdbX2

WORDPRESS_DB_HOST: mysql:3306

restart: unless-stopped

Orchestration avec Docker Compose

phpmyadmin:

container_name: phpmyadmin

image: phpmyadmin/phpmyadmin

links:

- mysql:db

ports:

- 8189:80

environment:

MYSQL_ROOT_PASSWORD: superroot

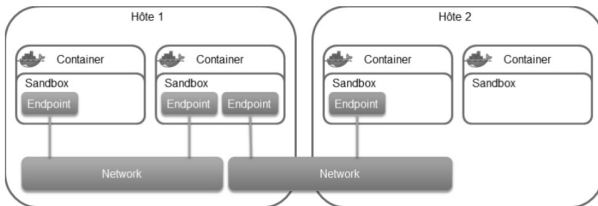
restart: unless-stopped

Le réseau Docker

Le réseau Docker

La communication entre conteneurs est un élément clé du succès de l'architecture micro-services et par conséquent de Docker. Elle repose sur la librairie libnetwork1 qui est née de la volonté d'extraire le système de gestion du réseau du moteur. Elle implémente le modèle CNM (Container Network Model)

Le modèle CNM est véritablement une abstraction pour la communication inter conteneurs (potentiellement déployés sur des hôtes différents).



Le réseau Docker

`docker network ls`

- `null` : c'est un driver un peu particulier qui signifie « pas de réseau ». Il est là par souci de rétrocompatibilité au niveau de Docker ;
- `overlay` : ce driver est pour l'instant le seul qui permette une communication entre plusieurs hôtes. Son utilisation rentre dans le cadre de Docker Swarm ;
- `host` : permet de rendre disponible la configuration de la machine hôte à notre conteneur ; le conteneur peut donc directement accéder à toutes les ressources de l'hôte ;
- `bridge` : ce driver se base sur un bridge Linux. Il n'est disponible qu'à l'intérieur d'un même hôte.

`docker network inspect bridge`

docker compose V3

```
version: "3.7"
volumes:
  officelog:
    driver: local
    name: officelog
    driver_opts:
      type: none
      device: /data/docker/nuage/office/log/
    o: bind
services:
  office:
    container_name: office
    hostname: office
    image: onlyoffice/documentserver:latest
    networks:
      - nuage
    stdin_open: true
    tty: true
    ports:
      - 8087:443
    volumes:
      - officelog:/var/log/onlyoffice/
    restart: unless-stopped
    networks:
      nuage:
        name: nuage
    ipam:
      config:
        - subnet: 172.22.0.0/16
```


Podman

DOCKER : moteur de conteneur

autres outils alternatifs:

PODMAN lxd cri-o rkt

```
podman pod create --name mypod
```

```
bf40da3347b292cb44cbc4fb23d128dab6923fabf350513fa2840f74d48f2e71
```

```
podman pod list
```

```
POD ID NAME STATUS CREATED OF CONTAINERS INFRA ID
```

```
bf40da3347b2 mypod Created 9 seconds ago 1 d9b4e641e381
```

```
podman run -d --pod mypod nginx
```

```
Getting image source signatures
```

```
Copying blob 166a2418f7e8 done
```

```
Copying config c39a868aad done
```

```
Writing manifest to image destination
```

```
Storing signatures
```

```
1ad4076acc8ece1e9df969b747fb4946768c20a44b708fba8c16e196956f0430
```

```
podman run -d --pod mypod nginx
```

```
2b6a4837ff2071f9b8efe9bf52ddb272d4b9ca79649d830a2034aba84b14dc58
```

```
podman ps -a --pod
```

```
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES POD ID PODNAME
```

```
1ad4076acc8e docker.io/library/nginx:latest nginx -g daemon o... 29 seconds ago Up 28 seconds ago
```

```
sharpmendelbf40da3347b2mypod
```

```
2b6a4837ff20 docker.io/library/nginx:latest nginx -g daemon o... 10 seconds ago Exited (1) 7 seconds
```

```
ago xenodochialjederbergbf40da3347b2mypod
```

```
d9b4e641e381 k8s.gcr.io/pause:3.2
```

Podman

Pour construire une image :

Podman build

buildah bud -f Dockerfile .

Kaniko de google doit être exécuté en tant qu'image.

CLOUD

CLOUD

Kubernetes Operations (kops)

kops est un système de provisionnement dont les principes sont : une installation totalement automatisée, l'auto-guérison, haute disponibilité, etc ... Il ne supporte officiellement qu'AWS (Amazon Web Service).

— Amazon Elastic Compute Cloud (EC2) qui est service le plus utilisé d'Amazon. EC2 fournit des serveurs virtuelles.

— Amazon Virtual Private Cloud (VPC) qui fournit des réseaux virtuels privés au sein d'AWS pour isoler nos serveurs virtuelles.

— Amazon Elastic Block Store (EBS) qui fournit un stockage temporaire par bloc pour nos instances EC2.

— Amazon Simple Storage Service (S3) est un service de stockage d'objet offrant une scalabilité, une disponibilité des données, une sécurité et des performances de pointe.

Helm : Gestionnaire de paquet Helm est un gestionnaire de paquet pour Kubernetes maintenu par la Cloud Native Computing Foundation (CNCF) en collaboration avec des grands groupes comme Microsoft ou Google.

Helm permet de gérer facilement des applications de la plus simple aux plus complexes. Pour cela, les applications sont décrits dans des répertoires que l'on appelle des Charts. Il existe un répertoire de Charts stable qui ne contient que des Charts considérés comme sûre et facilement paramétrables.

Pour fonctionner Helm a besoin que l'on installe Tiller qui est la partie serveur de Helm. Il faut tout d'abord créer un utilisateur sur Kubernetes, lui allouer des droits et enfin installer le conteneur contenant Tiller. Ensuite on pourra communiquer nos instructions via l'interface en ligne de commande helm.

CLOUD

Helm est axé autour de trois notions :

Charts : Collection de fichiers YAML variabilisés décrivant des ressources qui, mises bout à bout, donnent une application déployable sur Kubernetes.

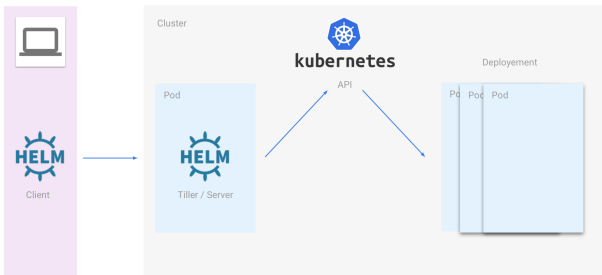
Helm : Client permettant de récupérer des Charts et de les appliquer sur un cluster Kubernetes.

Tiller : Partie serveur permettant à un client Helm de donner des ordres au cluster Kubernetes visé.

- `helm search hub pytorch`
- `helm repo add bitnami https://charts.bitnami.com/bitnami`
- `helm install bitnami/pytorch`

CLOUD

L'architecture de Helm :



- . Un package Kubernetes est appelé Charts dans Helm.
- . Un Chart contient un lot d'informations nécessaires pour créer une instance d'application Kubernetes.
- . La Config contient les informations dynamiques concernant la configuration d'un Chart
- . Une Release est une instance existante sur le cluster, combinée avec une Config spécifique.
- . Helm 3 n'a plus besoin de tiller

CLOUD

```
helm search repo nginx
```

```
NAME CHART VERSION APP VERSION DESCRIPTION
```

```
bitnami/nginx 5.0.0 1.16.1 Chart for the nginx server
```

```
bitnami/nginx-ingress-controller 5.2.0 0.26.1 Chart for the nginx Ingress controller
```

```
helm install reds bitnami/nginx
```

```
NAME: reds
```

```
LAST DEPLOYED: Tue Nov 26 09:35:09 2019
```

```
NAMESPACE: default
```

```
STATUS: deployed
```

```
REVISION: 1
```

```
TEST SUITE: None
```

```
NOTES:
```

```
Get the NGINX URL:
```

```
NOTE: It may take a few minutes for the LoadBalancer IP to be available.
```

```
Watch the status with: 'kubectl get svc --namespace default -w reds-nginx'
```

```
export SERVICE_IP=(kubectl get svc --namespace default reds-nginx --  
-template "range({index.status.loadBalancer.ingress0}.end)" )echo "NGINXURL : http : //SERVICE_IP/"
```



```
kubectl get svc -w reds-nginx
```

```
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
```

```
reds-nginx LoadBalancer 10.0.15.121 35.202.189.178 80:31529/TCP,443:30655/TCP 115m
```

Cloud Google Kubernetes Engine

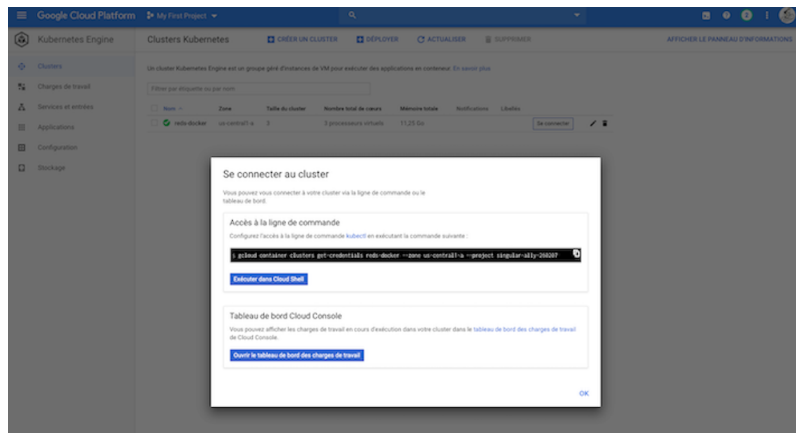
Pour emballer et déployer une application sur GKE, vous devez :

- emballer l'application dans une image Docker ;
- exécuter le conteneur localement sur votre ordinateur ;
- importer l'image dans un registre ;
- créer un cluster de conteneur ;
- déployer l'application sur le cluster ;
- exposer l'application sur Internet ;
- procéder au scaling du déploiement ;
- déployer une nouvelle version de l'application.

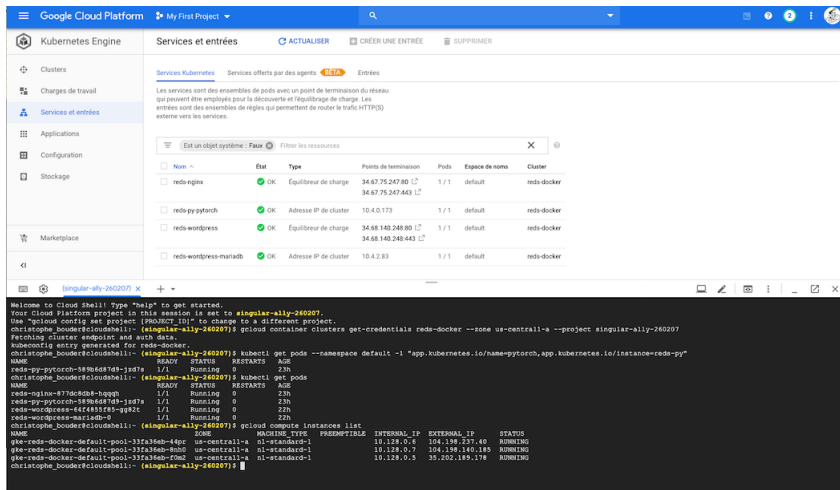
GKE

Utilisation de Google Cloud Shell ou Installation du SDK Google Cloud, qui inclut l'outil de ligne de commande gcloud.

gcloud components install **kubectl**



Utilisation de Google Cloud Shell



The screenshot shows the Google Cloud Platform console for a project named "My First Project". The left sidebar shows the navigation menu with "Services et entrées" selected. The main content area displays a table of services and their status.

Non	État	Type	Points de terminaison	Pods	Espace de noms	Cluster
<input type="checkbox"/>	OK	Équilibreur de charge	34.67.75.247:80	1/1	default	reds-docker
<input type="checkbox"/>	OK	Adresse IP de cluster	10.4.0.173	1/1	default	reds-docker
<input type="checkbox"/>	OK	Équilibreur de charge	34.68.140.248:80	1/1	default	reds-docker
<input type="checkbox"/>	OK	Adresse IP de cluster	10.4.2.83	1/1	default	reds-docker

Below the table, the Google Cloud Shell terminal is open, showing the following commands and output:

```

Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to singular-ally-260207.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
christophe_boude@cloudshell:~$ (singular-ally-260207) gcloud container clusters get-credentials reds-docker --zone us-central1-a --project singular-ally-260207
Fetching cluster endpoint and auth data.
kubeconfig entry generated for reds-docker.
christophe_boude@cloudshell:~$ (singular-ally-260207) kubectl get pods --namespace default -l "app.kubernetes.io/name=pytorch,app.kubernetes.io/instance=reds-py"
NAME          READY   STATUS    RESTARTS   AGE
reds-py-pytorch-38b6d87d8-3d7a  1/1     Running   0           23h
christophe_boude@cloudshell:~$ (singular-ally-260207) kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
reds-nginx-877dc8d88-hqggh  1/1     Running   0           23h
reds-py-pytorch-38b6d87d8-3d7a  1/1     Running   0           23h
reds-wordpress-64f4855f85-gu82t  1/1     Running   0           22h
reds-wordpress-mariadb-0  1/1     Running   0           22h
christophe_boude@cloudshell:~$ (singular-ally-260207) gcloud compute instances list
NAME          ZONE          MACHINE_TYPE  PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP  STATUS
gke-reds-docker-default-pool-33fa36eb-44pr  us-central1-a  n1-standard-1  10.128.0.6  104.198.237.40  RUNNING
gke-reds-docker-default-pool-33fa36eb-8m60  us-central1-a  n1-standard-1  10.128.0.7  104.198.140.185  RUNNING
gke-reds-docker-default-pool-33fa36eb-cm02  us-central1-a  n1-standard-1  10.128.0.5  35.202.189.176  RUNNING
christophe_boude@cloudshell:~$ (singular-ally-260207)

```

GKE

- . Construire l'image du conteneur

```
docker build -t gcr.io/PROJECT_ID/hello-app:v1 .
```

- . Importer l'image de conteneur

```
gcloud auth configure-docker
```

```
docker push gcr.io/PROJECT_ID/hello-app:v1
```

- . Exécuter le conteneur localement

Sous Cloud Shell, vous pouvez cliquer sur le bouton "Aperçu Web"

- . Déployer l'application

```
kubectl create deployment hello-web gcr.io/PROJECT_ID/hello-app:v1
```

- . Exposer l'application sur Internet

```
kubectl expose deployment hello-web --type=LoadBalancer --port 80 --target-port 8080
```

- . Procéder au scaling de l'application

```
kubectl scale deployment hello-web --replicas=3
```

- . Déployer une nouvelle version de l'application

refaire un build transférer l'image vers Google Container Registry gcr.io

```
kubectl set image deployment/hello-web hello-app=gcr.io/PROJECT_ID/hello-app:v2
```

Installation de docker et podman sous linux au besoin installer une VM sous linux si pb sous mac ou windows

Télécharger l'ensemble du dossier TP :

<https://nuage.lip6.fr/s/PRJZdwC564DPtQq>

Effectuer les exercices dans l'ordre

Créer un Dockerfile fonctionnel avec de multiples instructions

Créer un fichier docker-compose fonctionnel avec de multiples services en version V2 ou V3

Déployer un Dockerfile sur kubernetes

Merci de votre attention