Docker outil indispensable du Devops

Paquet Judicaël
DSI chez **batiwiz** *et*
Coach Agile & Devops
chez **Myagile Partner**

De plus en plus populaire ces dernières années, nous allons comprendre pourquoi docker est de plus en plus utilisé par les développeurs, et pourquoi il est devenu une arme indispensable dans le monde du Devops. Nous allons ensuite le prendre en main avec des exemples concrets afin d'apprendre à l'utiliser sur des cas concrets.

Docker est un logiciel libre qui permet d'automatiser le déploiement d'applications et de leurs dépendances dans différents conteneurs isolés sur n'importe quel serveur Linux. Docker a été développé par Solomon Hykes à la base comme projet interne de DotCloud en tant qu'évolution de solutions open-source déjà existantes au sein de la société. Docker a été distribuée en open-source la première fois le 13 mars 2013. Au moment où j'écris ces lignes, il y a 1600 contributeurs sur le Github de l'application.

Les conteneurs

Si la notion de conteneurs se démocratisa avec l'arrivée de Docker, il faut savoir que ce concept en réalité n'a pas été créé par Docker. Linux a en fait déjà un système de conteneurs qui s'appelle LXC (Linux Containers) sur lequel Docker s'est d'ailleurs initialement basé. Un conteneur est une boîte qui va être intégralement isolée du système d'exploitation dans laquelle on installera une application ainsi que toutes les librairies nécessaires au bon fonctionnement de celle-ci. Il sera alors facile de distribuer son application qui ne sera pas dépendante de votre système d'exploitation.

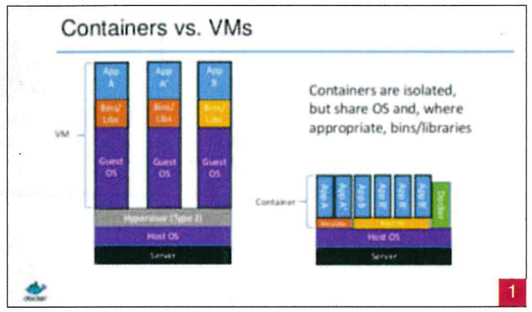
Les avantages de ce concept

Nos applications se complexifient avec le temps et nous sommes contraints d'installer de nombreuses librairies pour faire fonctionner correctement nos applications.

Pour prendre un exemple simple, mon futur site Internet aura plusieurs besoins pour fonctionner : mettre un Apache en serveur Web, un redis en serveur de cache, un MySQL pour avoir une base de données, un PHP 7 avec différentes librairies en interpréteur de code.

Sans conteneur, ceci va avoir des conséquences pour les administrateurs système qui vont devoir refaire la configuration des machines (éventuellement via un infogéreur externe) et tous les développeurs vont devoir installer l'environnement (ce qui se complexifie quand les développeurs ne travaillent pas sur le même système d'exploitation).

En effet il existe des outils comme Vagrant, Chef, Puppet qui aident les administrateurs système mais pas du tout au niveau d'automatisations qu'on aimerait avoir. Les conteneurs vont permettre d'éviter toute cette complexité en proposant une installation commune pour tout le monde. Un gain de temps phénoménal quand on maîtrise un minimum Docker.



Quelle différence avec des machines virtuelles ?

Au premier abord, nous pourrions ne pas comprendre ce qu'apportent les conteneurs par rapport aux machines virtuelles.

Voici un schéma proposé par Docker pour expliquer la différence entre des conteneurs et des machines virtuelles : [1]

Les machines virtuelles sont des systèmes d'exploitation qui vont fonctionner par-dessus le système d'exploitation du serveur. Pour faire simple, on a des ordinateurs dans un ordinateur.

Le souci de ce type d'architecture, c'est qu'on est obligé de faire tourner 4 systèmes d'exploitation différents pour mettre 3 machines virtuelles (3 applications différentes) et dédier de la ressource à celles-ci (ram, cpu, disque dur) ; ceci prend énormément de ressources et limite le nombre de machines virtuelles que nous pouvons mettre sur un serveur. Les conteneurs vont permettre de faire tourner les 3 applications directement sur 1 seul système d'exploitation et donc d'alléger considérablement le besoin en ressources.

Du coup au lieu de transférer des systèmes complets comme on est obligé de le faire avec des machines virtuelles pour porter notre application à différents endroits, les conteneurs seront très légers à transférer.

Les conteneurs permettent en conséquence d'être beaucoup plus scalables et peuvent beaucoup plus facilement être déployés.

Avec Docker, les conteneurs se popularisent

Si le système des conteneurs n'est pas nouveau, Docker facilite considérablement son utilisation en proposant de gérer vos conteneurs avec de simples fichiers plats contenant l'ensemble des installations complémentaires à réaliser et en vous offrant des lignes commandes très simples à utiliser pour vous aider dans la gestion de vos conteneurs.

De plus comme Github, Docker propose un espace de partage de conteneurs privés ou publics très utilisés par les différents éditeurs de logiciels. Ces derniers y exposent des conteneurs complets et configurés prêt à l'emploi. Une mine d'or pour travailler plus rapidement.

Fonctionnement de Docker

Docker, c'est un démon qui tournera en permanence pour gérer les conteneurs en cours d'exécution et un client qui propose de nombreuses lignes de commande que nous verrons à la suite de ce dossier.

Afin de faire profiter de cette technologie aux utilisateurs de Windows et macOS, Docker propose également le Boot2Docker (Docker Toolbox) qui se chargera de monter une machine virtuelle sur ces systèmes d'exploitation avec l'ensemble des outils docker nécessaires pour son bon fonctionnement. Docker travaille également sur la possibilité de faire des conteneurs sur Windows dans le futur ; cela qui apportera également un gain considérable pour tous les systèmes spécialisés sous Windows.

Installation

Pour nos premiers pas, je vais prendre une distribution Linux Ubuntu 16.04 qui est l'une des plus populaires dans le monde des développeurs. Si vous utilisez d'autres distributions, vous trouverez facilement les démarches pour installer votre Docker.

Commençons par installer Docker sur notre serveur Ubuntu :

```
sudo apt-get install -y docker.io
```

Sous Windows et macOS, il faudra installer le logiciel Docker Toolbox que vous pouvez trouver sur le site officiel de Docker. Dès que vous aurez lancé votre application Docker Toolbox, vous pourrez utiliser Docker et toutes les fonctions docker de cet article de la même façon que sous Linux

Lançons notre premier conteneur Docker

Sur Linux, il faudra exécuter le démon de Docker (s'il n'est pas déjà lancé) de cette façon :

```
/etc/init.d/docker start
```

Si le démon se lance correctement ce qui devrait être le cas, vous aurez le message suivant : *[OK] Starting docker (via systemctl): docker.service.*

Sous Windows, le démon est déjà démarré au lancement de l'application Docker Toolbox. Si vous avez besoin un jour de connaître la version de votre Docker, voici la commande à faire :

```
docker version
```

Attention : selon la configuration de votre Linux Ubuntu, il sera peut-être nécessaire de mettre sudo avant chaque commande docker ou de revoir la configuration de vos utilisateurs.

Nous commençons par lancer un premier conteneur qui sert uniquement à afficher "Hello world" :

```
docker run hello-world
```

La première fois, Docker vous prévient qu'il ne trouve pas l'image docker du nom de hello-world en local sur la machine ; il va donc la télécharger automatiquement, l'installer et lancer ce premier conteneur.

Si tout va bien, vous verrez un *« Hello from Docker »* apparaître. Ce conteneur est simple et ne sert qu'à faire ce petit test ; cependant vous venez de lancer votre premier conteneur. Afin d'avoir une vision des conteneurs qui sont en cours d'exécution, il suffira d'utiliser cette commande :

```
docker ps
```

Notre conteneur s'est donc arrêté car il n'est pas dans la liste. En ajoutant l'option -a, vous pourrez le retrouver et ainsi voir tous les conteneurs qui ont été lancés même ceux qui se sont arrêtés.

Les images Docker

Une image Docker est une configuration d'un conteneur que nous pouvons récupérer ou que nous pouvons partager. Cet échange de conteneur se fait par le site Docker Hub fortement inspiré de GitHub que nous avons vu brièvement lors de la présentation de Docker.

Vous pourrez y récupérer un grand nombre d'images dont certaines directement proposées par les éditeurs pour vous mâcher une partie du travail. Vous pourrez sans soucis aller chercher une image sur leur site : https://hub.docker.com/ ou directement en ligne de commande.

Nous allons utiliser une commande proposée par Docker pour chercher une image PHP dans le but de la télécharger :

```
docker search php
```

Le premier résultat retourné est parfait pour mon test (image avec comme nom « php »). Nous allons en profiter pour la télécharger afin de l'utiliser :

```
docker pull php
```

Si ce terme vous est familier c'est parce que Docker a copié Git pour la gestion de partage d'images (docker commit, docker pull, docker push…) afin de rendre l'utilisation de Docker Hub la plus intuitive possible. Si vous avez bien suivi ce tutoriel, j'ai normalement 2 images à présent sur mon ordinateur : hello-world et php. Nous allons vérifier cela grâce à la commande suivante qui me permet de connaître le nom de toutes les images disponibles en local :

```
docker images
```

Vous pourrez à présent lancer un conteneur de cette nouvelle image PHP.

Créer sa propre image

Docker nous propose de créer nos images nous-même en utilisant le concept du fichier Dockerfile. Nous allons commencer par créer notre propre image pour bien comprendre les principes de base.

Nous allons créer un dossier dans lequel on va aller créer le fichier Dockerfile ci-dessous :

```
FROM ubuntu:16.04
MAINTAINER Judicael paquet
```

Pour expliquer brièvement, je commence par définir le Linux sur lequel va se créer mon image et l'auteur de celle-ci (autant se faire de l'auto promo). A présent, nous allons builder (construire en mode pro) notre image. Le point en fin de ligne de ma commande n'est pas une erreur car il permet de définir le lieu où se situe mon fichier (soit dans le dossier courant dans notre cas).

```
docker build .
```

Il télécharge puis exécute les deux étapes que je lui ai définies dans le Dockerfile. Si tout va bien, le script se termine sur un Successfully Build XXX. Nous allons à présent rajouter l'installation d'apache 2 en mettant à jour la liste de packages et en installant le package apache2 pour créer une image plus intéressante qu'un Linux vide. Voici le Dockerfile mis à jour :

```
FROM ubuntu:16.04
MAINTAINER Judicael paquet

RUN apt-get update && \
      apt-get install -y apache2
```

La commande RUN nous permettra comme vous devez le deviner de lancer des commandes lors de la création de notre image et le caractère \ permet d'écrire notre commande sur plusieurs lignes.
Je vous laisse builder votre image comme tout à l'heure afin de la mettre à jour. Vous constaterez que Docker n'exécute que l'étape 3 car il a gardé les étapes précédentes en mémoire.
Un jour cela pourra cependant vous poser des soucis donc pensez qu'il existe un paramètre *--no-cache* pour refaire à 100% le build de votre image.
Vous pourrez également nommer votre image avec le paramètre *-t monapache* qui vous permettra quand vous listerez vos images d'avoir écrit "monapache" au lieu de "<none>" dans la colonne REPOSITORY.

```
docker build --no-cache -t monapache .
```

Maintenant que notre image est complète, nous allons pouvoir en faire un conteneur en lançant la commande suivante que nous expliquerons juste en-dessous :

```
sudo docker run -d -p 8000:80 monapache /usr/sbin/apache2ctl -D FOREGROUND
```

-d : le mode détaché permet de ne pas bloquer votre terminal. Dès que le conteneur est lancé, docker vous rend la main
-p : permet de définir le port à appeler pour qu'il atteigne le port 80 sur le conteneur. C'est très pratique car nous pourrons du coup avoir un conteneur par port.
-D FOREGROUND : permet de lancer le processus dans le conteneur et d'attacher la console au processus d'entrée classique.
Nous pouvons à présent tenter de mettre http://localhost:8000 dans notre navigateur. En effet, nous arrivons bien à afficher la page par défaut d'Apache sur Ubuntu. Voici un autre exemple intéressant où nous allons copier des fichiers dans notre Docker en créant un dossier docker/ dans lequel nous allons créer les fichiers suivants.
host-apache2.conf :

```
<VirtualHost *:80>
  ServerAdmin webmaster@localhost
  DocumentRoot /var/www/test
</VirtualHost>
```

Index.html :

```
Hello World
```

On termine en mettant à jour notre Dockerfile avant de builder notre application :

```
FROM ubuntu:16.04
MAINTAINER Judicael paquet

RUN apt-get update && \
      apt-get install -y apache2

ADD docker/host-apache2.conf /etc/apache2/sites-enabled/000-default.conf
ADD docker/index.html /var/www/test/index.html

ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

Vous constaterez que nous avons d'ailleurs utilisé la méthode Entrypoint qui permet de lancer une commande au lancement du conteneur. Cela nous évitera de le faire lors de notre docker run que nous ferons comme cela :

```
docker build -t monapache .
docker run -d -p 8002:80 monapache
```

Si vous avez bien suivi l'ensemble de la procédure, vous devriez avoir un joli « Hello World » qui apparaît sur votre navigateur quand vous y appelez http://localhost:8002.
Pour vous éviter de nombreuses recherches, si vous avez besoin de faire un mapping sur deux ports différents vous pourrez rajouter une deuxième fois un –p 3306 :3306 par exemple si vous y installez aussi un MySQL. Nous allons à présent créer un fichier index.html sur la machine « host » avec un mkdir –p /var/www/test2 en y mettant « Hello my friend ». Docker permet également de partager ce fichier de la machine « host » au conteneur en utilisant l'option –v [path_host]:[path_container] comme ceci :

```
Docker run –d –p 8003:80 –v /var/www/test2:/var/www/test monapache
```

Le fichier qu'on avait copié par le Dockerfile n'est plus pris en compte car on a demandé au lancement du conteneur de monter le dossier test2 en tant que dossier test.
Et en testant dans votre navigateur, vous aurez la bonne surprise de voir s'afficher « Hello my friend » et non plus le « Hello World » précédent. Si l'exemple ici est un peu bateau, cela peut-être fort utile pour les personnes qui désirent partager le code de la machine host à un conteneur qui en aurait besoin.

Ajout d'une image MySQL

Maintenant que vous êtes plus à l'aise avec Docker et les Dockerfile, nous allons voir comment faire communiquer les conteneurs entre eux.
Nous allons commencer par faire tourner un nouveau conteneur faisant fonctionner MySQL de façon isolée. Pour cela, nous allons chercher sur Docker Hub (https://hub.docker.com), un conteneur déjà tout prêt que nous lancerons comme ceci :

```
docker run --name monmysql -e MYSQL_ROOT_PASSWORD=passe -d mysql
/mysql-server:5.7
```

L'option –e permet de définir une variable d'environnement lors du lancement de votre conteneur.
Notre nouveau conteneur est à présent lancé mais on va devoir vérifier cela. Docker propose une commande qui permet de se connecter à son conteneur et de l'utiliser :

```
docker exec –it monmysql mysql –uroot –p
```

Si vous suivez bien, le mot de passe demandé est celui que nous avions indiqué dans la précédente commande soit « passe ». Vous pourrez à présent faire toutes les opérations désirées sur votre MySQL.
Vous pourrez exécuter toutes les commandes que vous désirez avec cette commande docker exec dont la plus importante qui permet de se connecter de la même façon qu'en SSH :

```
docker exec –it monmysql /bin/bash
```

Linker les conteneurs entre eux

Nous allons maintenant voir comment faire communiquer deux conteneurs ensemble car cela pourrait vous aider pour monter vos futures applications avec docker.
Nous allons commencer par supprimer notre fichier /var/www/test2/index.html et nous allons créer un fichier /var/www/test2/index.php qui contiendra ceci :

```
<?php

$dbh = new PDO('mysql:host='.getenv('MYSQL_PORT_3306_TCP_ADDR').';dbname=
mysql', 'root', 'passe');

foreach($dbh->query('SHOW DATABASES') as $row) {
  echo $row[0]. '<br/>';
}
```

Nous allons utiliser le PHP pour faire une simple connexion sur le MySQL de l'autre conteneur et afficher le nom des bases.
Nous allons devoir par contre faire évoluer notre Dockerfile afin de rajouter toutes les librairies nécessaires pour faire la connexion à MySQL :

```
FROM ubuntu:16.04
MAINTAINER judicael paquet

RUN apt-get update && \
  apt-get install -y apache2 php libapache2-mod-php mysql-client php7.0-mysql

RUN sed -i 's/;extension=php_mysqli.dll/extension=php_mysqli.dll/' /etc/php/7.0/
apache2/php.ini
RUN sed -i 's/;extension=php_pdo_mysql.dll/extension=php_pdo_mysql.dll/' /etc/
php/7.0/apache2/php.ini

ADD docker/host-apache2.conf /etc/apache2/sites-enabled/000-default.conf

ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

Il faudra par contre dans notre cas donner des droits à notre utilisateur root de MySQL pour pouvoir y accéder de l'extérieur avec la commande suivante :

```
docker exec -it monmysql mysql -uroot -p -e "CREATE USER 'root'@'%' IDENTIFIED
BY 'passe';GRANT ALL PRIVILEGES ON *.* TO 'root'@'%';FLUSH PRIVILEGES;"
```

Cet article n'a pas pour but de travailler sur la sécurité de vos applications mais il est évident qu'il faudra créer des utilisateurs plus sécurisés pour vos futures applications.
Il ne reste plus qu'à rebuilder l'image monapache pour pouvoir la lancer avec l'option –link qui permettra de lier nos deux conteneurs :

```
docker run -p 8004:80 --name monapachephp --link monmysql:mysql -v /var/
www/test2:/var/www/test -d monapache
```

Normalement vous verrez la liste des bases de données disponibles qui s'afficheront sur votre navigateur Internet quand vous mettrez http://localhost:8004. J'espère que j'ai pu vous permettre de bien comprendre comment fonctionne docker et les Dockerfile qui vous seront très utiles pour créer vos futurs environnements.

Des petites choses à savoir sur le Dockerfile

Voici d'autres instructions que vous pourrez utiliser au sein de vos Dockerfile pour créer vos images :
ENV abc=demo permet de créer une variable d'environnement abc qui aura la valeur demo ;
CMD echo 'Test' permet d'exécuter du code au moment où le conteneur se lance. Vous pourrez surcharger cette commande dans le docker run. Si vous mettez plusieurs CMD au sein de votre Dockerfile, seul le dernier sera pris en compte ;
EXPOSE 3306 permet de définir le ou les ports qui pourront communiquer avec l'extérieur ;
ENTRYPOINT cmd permet d'exécuter une commande au démarrage du conteneur. Comme pour CMD, seul le dernier ENTRYPOINT sera pris en compte. Par contre cette méthode n'est pas surchargeable par docker run ;
USER ubuntu permet de définir l'utilisateur qui sera utilisé pour lancer tous les prochains RUN et ENTRYPOINT du fichier. Par défaut docker utilise le root ;
WORKDIR path permet de définir le répertoire dans lequel s'exécuteront les commandes CMD et ENTRYPOINT ;
ONBUILD est une notion un peu plus avancée qui permet de réaliser des triggers que nous ne verrons pas dans cet article.

Quelques commandes Docker supplémentaires

Voici des commandes qui pourront vous être utiles dans docker :
docker ps –last 10 permet de voir les dix derniers conteneurs lancés ;
docker kill monconteneur permet d'arrêter un conteneur en cours d'exécution. C'est très pratique quand on commence à avoir de nombreux conteneurs de tests ;

docker logs monconteneur permet de revoir les logs du conteneur ;
docker restart monconteneur permet de redémarrer si besoin un conteneur ;
docker rm monconteneur permet de supprimer un conteneur définitivement ;
docker stop monconteneur et **docker start monconteneur** permettent d'arrêter ou de démarrer un conteneur ;
docker top monconteneur permet d'avoir un top du conteneur.
Il existe un grand nombre d'autres commandes mais je n'aurai pas la place de toutes les décrire car docker propose vraiment un grand nombre de choses pour gérer vos conteneurs.

Docker-compose permet de packager

Docker propose un outil très intéressant de gestion de package docker. Cet outil permet de définir et de faire tourner plusieurs conteneurs grâce à un seul fichier de configuration. C'est un peu le apt-get ou le composer de docker pour faire simple. Cet outil vient simplifier la vie aux utilisateurs car comme vous l'avez vu dans cet article, maîtriser Docker n'est pas si simple.
Voici l'installation qu'il faudra faire pour installer docker-compose qui ne fait pas partie du docker.io qu'on avait installé en début d'article :

```
sudo curl -o /usr/local/bin/docker-compose -L "https://github.com/docker/compose/releases/download/1.8.1/docker-compose-$(uname -s)-$(uname -m)"
sudo chmod +x /usr/local/bin/docker-compose
```

Nous allons vérifier que l'installation s'est bien déroulée en tentant de vérifier la version du docker-compose que nous venons d'installer :

```
docker-compose –v
```

Nous allons à présent légèrement modifier notre fichier /var/www/test2/index.php comme ceci afin d'utiliser docker-compose pour recréer l'environnement que nous avions dans cet article avant d'entamer docker-compose :

```
<?php

$dbh = new PDO('mysql:host=db;dbname=mysql', 'root', 'passe');

foreach($dbh->query('SHOW DATABASES') as $row) {
 echo $row[0]. '<br/>';
}
```

Docker compose se base sur un fichier docker-compose.yml qui sera la description de l'ensemble des besoins pour créer une application complète potentiellement constituée de plusieurs conteneurs ; ce fichier est écrit sous le format Yaml.

```
db:
  image: mysql/mysql-server:5.7
  ports:
    - "3306:3306"
  environment:
    - "MYSQL_ROOT_PASSWORD=passe"
    - "MYSQL_USER=your_user"
    - "MYSQL_PASSWORD=your_user_password"
    - "MYSQL_DATABASE=your_database_name"
superapachephp:
  build: ./
  ports:
    - "9010:80"
  volumes:
    - "/var/www/test2:/var/www/test:rw"
  links:
    - "db:db"
  working_dir: "/home/docker"
```

Nous devons lancer la commande suivante pour mettre notre docker compose en action et qu'il fasse lui-même tout le travail pour lancer les différents conteneurs :

```
docker-compose up -d
```

En testant http://localhost:9010 dans votre navigateur vous aurez la liste des bases existantes dans votre base de données MySQL. N'oubliez pas la manipulation à faire sur les utilisateurs sous MySQL si rien ne s'affiche.
Vous pourrez arrêter tous les conteneurs de votre docker-compose automatiquement en une seule commande ce qui est très pratique quand on travaille sur des projets assez gros :

```
sudo docker-compose stop
```

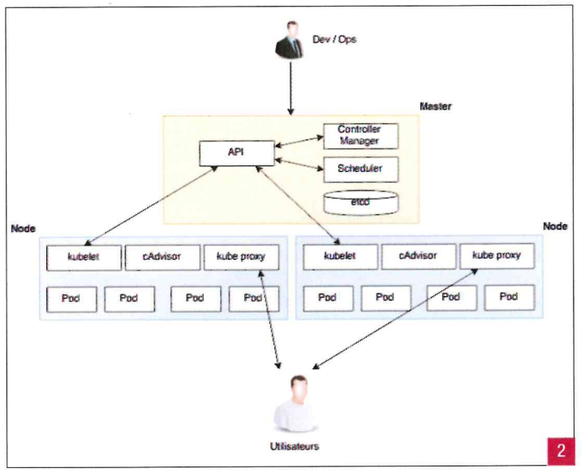
L'utilisation de docker-compose n'est pas toujours simple mais la communauté de Docker est tellement active que vous trouverez toujours une solution à vos différents problèmes.

Docker au service du déploiement

Docker et docker-compose permettent de préparer vos déploiements dans tous types d'environnements : dev, recette, pré-production, production.
Vous verrez de plus en plus dans les projets récents partagés sur Github des fichiers Dockerfile et docker-compose.yml à la racine des projets pour monter les environnements complets de l'application à la suite du git clone du projet. Cette pratique est très utile pour tester rapidement l'application proposée voire pour travailler dessus.

Orchestrer ses conteneurs avec Kubernetes

Kubernetes est une solution d'orchestration et de gestion de clusters de conteneurs open-source écrit en Go qui a été créé par Google.
Le projet est partagé sur Github et possède une communauté de plus de 1000 contributeurs.
Si l'utilisateur classique va simplement choisir un type d'instance, un OS et une région d'hébergement, l'hébergeur va, lui, avoir un véritable orchestrateur pour répondre à votre demande en provisionnant des VM sur des serveurs physiques ou en adaptant des VM déjà existantes.
SI Docker est un outil idéal dans la gestion de conteneurs, Kubernetes propose beaucoup plus que la gestion de conteneurs. [2]
L'idée de base de Kubernetes est de pouvoir déployer un ou plusieurs conteneurs dockers gérés par un master. On appellera des « Pods » (voir image sur l'architecture de Kubernetes), des unités de déploiement de Kubernetes qui peuvent être créées ou supprimées par l'intermédiaire de commandes ; c'est la notion la plus granulaire sur un cluster Kubernetes.



Les nodes (appelés aussi workers ou minions) sont des machines (ou machines virtuelles) où des conteneurs sont déployés.
Kubernetes fonctionne avec les différents hébergeurs Cloud comme AWS, Azure et Google Cloud.
Kubernetes n'est pas la seule solution d'orchestration mais est de loin la plus répandue. Vous pourrez également regarder du côté de docker-swam, Mezos ou Fleet.

Installation de kubernetes

Pour installer Kubernetes sur notre Ubuntu, nous allons appliquer cette liste de commandes :

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
sudo chmod 777 /etc/apt/sources.list.d
cat <<EOF > /etc/apt/sources.list.d/kubernetes.list
deb http://apt.kubernetes.io/ kubernetes-xenial main
EOF
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl kubernetes-cni
```

Kubernetes et Google Container Engine

Pour travailler avec Google Container Engine, nous devons installer l'outil proposé par Google comme ceci :

```
export CLOUD_SDK_REPO="cloud-sdk-$(lsb_release -c -s)"
echo "deb https://packages.cloud.google.com/apt $CLOUD_SDK_REPO main" |
sudo tee -a /etc/apt/sources.list.d/google-cloud-sdk.list
curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
sudo apt-get update && sudo apt-get install google-cloud-sdk
```

Il vous faudra également créer un compte gratuit sur Google Container Engine qui vous permettra de réaliser jusqu'à 300 de tests.
Dès que votre inscription est terminée, pensez à créer un nouveau projet vide sur l'interface. Ce projet va regrouper l'ensemble de vos clusters donc le cluster que nous allons installer.
Maintenant que gcloud est installé, nous allons le configurer comme ceci :

```
gcloud init
gcloud config set compute/zone europe-west1-b
gcloud container clusters create cluster-1
gcloud auth application-default login
gcloud container clusters get-credentials cluster-1 --zone us-central1-c --project
stable-synapse-156722
```

Le paramètre à mettre en projet sera l'id associée au projet que vous avez créé sur l'interface de Google. Vous le trouverez dans la page d'accueil de Google Cloud Platform au sein du bloc projet.
Nous pourrons vérifier l'état des clusters avec la méthode suivante :

```
Kubectl cluster-info
```

Nous pourrons regarder à tout moment l'état de nos pods et de nos services avec les commandes suivantes :

```
kubectl get pod
kubectl get service
```

On va ainsi créer des pods et le service associé d'une nouvelle image que nous allons récupérer :

```
docker pull eboraas/apache-php
kubectl run apache --image=eboraas/apache-php --replicas=2 --port=80 --
expose --service-overrides='{ "spec": { "type": "LoadBalancer" } }'
```

En affichant vos pods et vos services, vous aurez au bout de quelques minutes, une IP publique pour votre application :

```
NAME                  READY   STATUS    RESTARTS   AGE
apache-3670720257-1ll4t  1/1     Running   0          3m
apache-3670720257-1v7af  1/1     Running   0          3m

NAME         CLUSTER-IP     EXTERNAL-IP      PORT(S)      AGE
apache       10.55.254.17   104.198.150.151  80:32130/TCP 3m
kubernetes   10.55.240.1    <none>           443/TCP      14m
```

En vérifiant la nouvelle adresse dans le navigateur, vous pourrez voir la belle page d'accueil d'apache 2 qui nous confirme que nous venons de déployer notre premier projet qui contient un load balancer et deux enfants sur Google Cloud Platform.
On pourra supprimer notre déploiement avec les deux méthodes suivantes car il ne faut pas oublier qu'on a créé un service et des pods :

```
kubectl delete deployment my-nginx
kubectl delete svc my-nginx
```

CONCLUSION

Docker et Kubernetes sont devenus des outils indispensables dans l'univers devops mais aussi dans le monde de l'open source.
Si cet article vous permet de vous lancer avec ce type d'outils, sachez qu'ils ne sont pas évidents à maîtriser. N'hésitez pas à parcourir le net qui propose de nombreux articles sur les sujets qui viendront compléter le contenu de cet article.