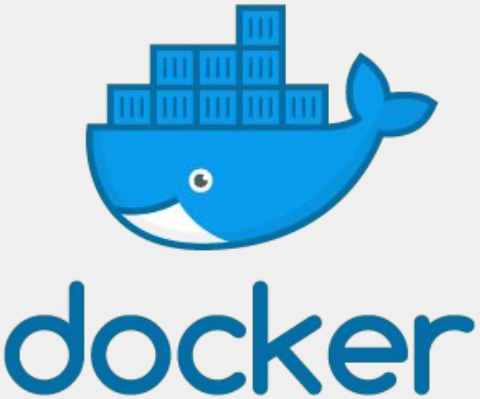
Tutoriel Docker et Kubernetes

On contient notre joie, s'il vous plaît!

#Fait sur Linux Server 16.03

Qu’est-ce que Docker?

Les machines virtuelles, cétacé!



Docker est un outil permettant de grandement faciliter la création, la publication et l’exécution d’applications dans des conteneurs.

Les conteneurs permettent de paqueter une application avec tout le nécessaire pour fonctionner: Librairies et autres dépendances.

Qu’est-ce que Kubernetes?

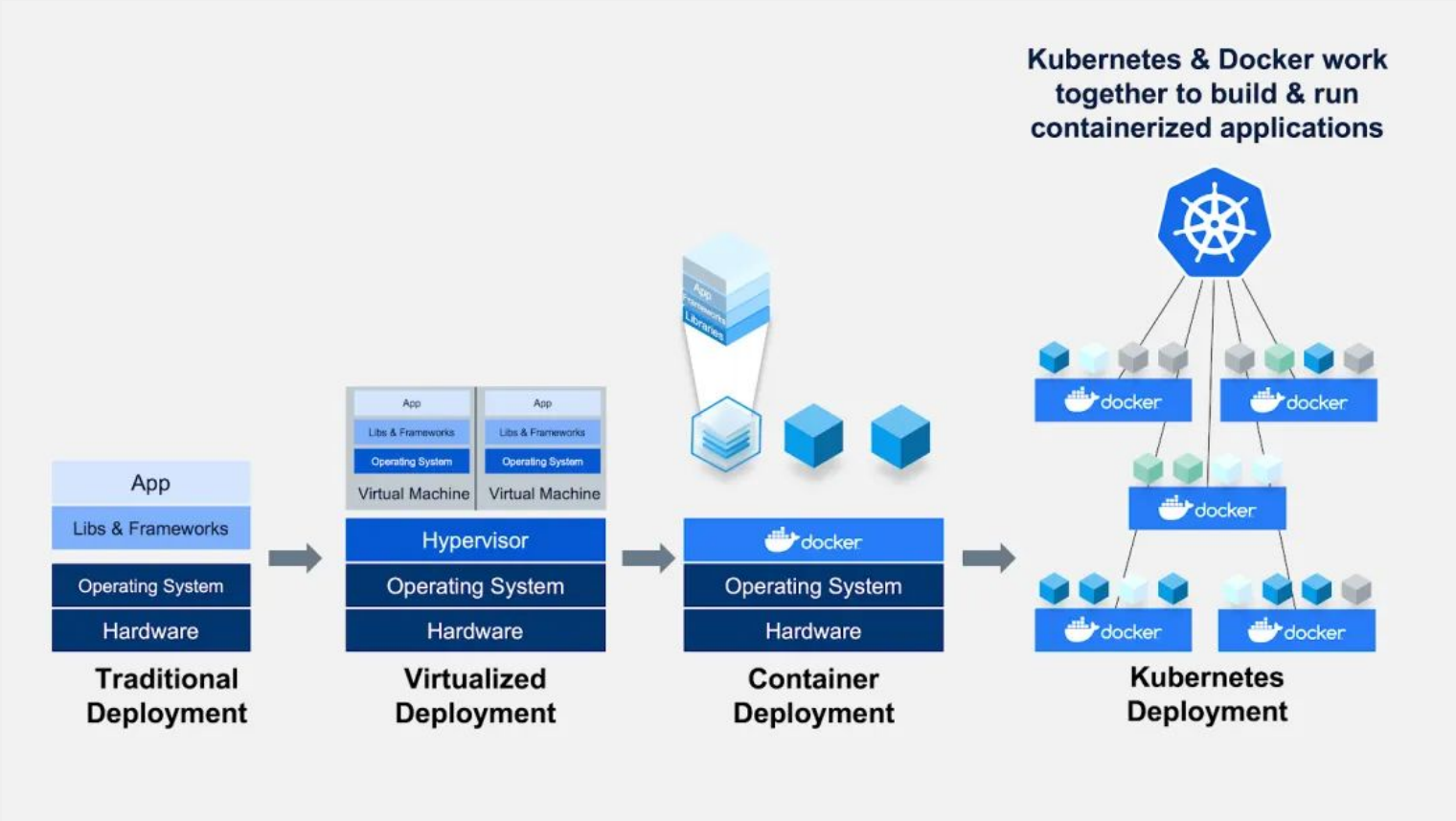
Kuberne-quoi? Ça se mange?

Kubernetes est un logiciel permettant l'orchestration complète de conteneurs.

Grâce à *Kubernetes*, il est facile d'augmenter la puissance qu'on veut octroyer à nos différentes applications (gestion de la scalability).

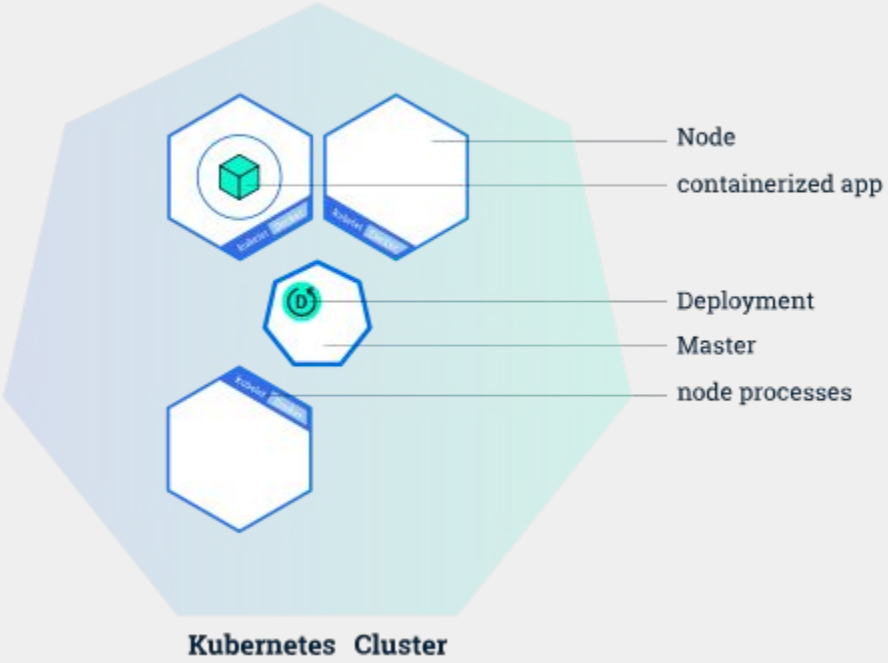
kubernetes

Architecture de Docker & Kubernetes (K8s)



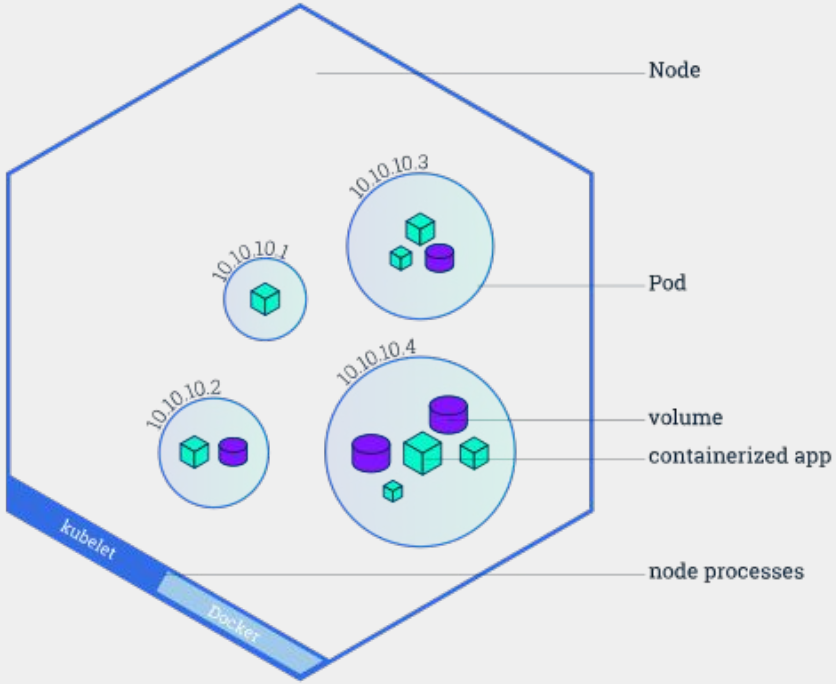
Les bases de Kubernetes (K8s)

Le cluster, l'environnement naturel de K8s



Les bases de Kubernetes (K8s)

Les nodes, la puissance est dans le nombre!



Les bases de Kubernetes (K8s)

Les pods, l'unité de K8s

Par où commencer?

Compliqué de naviguer sans carte...

La première étape est de se procurer un hyperviseur de machine virtuelle.

VirtualBox, Hyper-V, VMWare, etc.

Ils ont chacune leur propre méthode de configuration.

```
#Pour la démonstration, nous allons utiliser Hyper-V
```

Ensuite il faut installer microk8s, une distribution Kubernetes permettant d'utiliser facilement ce framework.

À faire dans la console du serveur Linux:

```
$> sudo apt install snapd | sudo snap install microk8s --classic
--channel=1.16/stable
```

Ensuite on installe Docker :

```
$> sudo apt install docker.io
```

Puis, pour éviter de toujours lancez les prochaines commandes avec sudo, on ajoute l'accès à l'utilisateur.

```
$> sudo usermod -a -G microk8s <utilisateur actuel>
$> sudo usermod -aG docker <utilisateur actuel>
```

Finalement on redémarre le serveur pour appliquer la sécurité.

```
$> sudo shutdown -r now
```

Préparer une image conteneurisé Docker

Allez hop! C'est dans la boîte!

Nous allons préparer un dossier pour notre application :

```
$> mkdir application
$> cd application
$> sudo bash
```

Ensuite, on écrit l'application :

```
$> nano index.html
```

Ensuite dans nano, entrez : (ctrl-o pour enregistrer, ctrl-x pour quitter)

```
<!DOCTYPE>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title></title>
</head>
<body>
    <p>Hello World!</p>
</body>
</html>
```

Ensuite, nous devons ajouter un fichier DockerFile, qui va indiquer à Docker comment construire l'image de l'application :

```
$> nano Dockerfile
```

Ensuite dans nano, entrez :

```
FROM nginx:alpine
COPY . /usr/share/nginx/html
RUN chmod -R +rwx /usr/share/nginx/html
```

Bien sûr, il existe BIEN d'autres commandes DockerFile, mais nous allons faire simple pour ce tutoriel.

Pause Docker Hub...

Mais, on vient à peine de partir...

Pour que Kubernetes puisse utiliser l'image Docker, il faut la mettre en ligne, soit par Docker Hub (Compte Requis), Google Cloud ou autre.

```
#Ici on va passer par Docker Hub, utilisons les outils fournis par eux
```

Préparer une image conteneurisé Docker

Puis, nous allons demander à Docker de construire l'image de l'application. (Veuillez d'abord vous rendre à la racine de l'application via une console) :

```
$> docker build  -t <nom de l'image>:<tag> .
     #Le point est important, il m'a causé quelques soucis...
```

Docker devrait finir le travail avec `Successfully tagged <nom de l'image>:<tag>`. Des messages d'avertissement de sécurité passeront également, mais ils sont en rapport avec les permissions de fichiers, qui sont normalement réglés rendu à cette étape.

Ensuite nous allons cloner l'image et le renommer pour l'envoyer sur Docker Hub:

```
$> docker tag <nom de l'image local>:<tag> <nom utilisateur Docker hub>/<nouveau
nom de l'image>:<tag>
#le tag est un indice de version au format quelconque, 'latest' par défaut
```

Envoyer l’image conteneurisé sur Docker Hub

L’image est dans le conteneur, le conteneur est dans le Hub et l’arbre est dans ses feuilles dabidom-dabidé!

Nous allons d’abord nous connecter sur le Docker Hub :

```
$> docker login
     #On connaît la routine, usager/mot de passe tout le tralala...
```

Puis on pousse simplement l’image clonée :

```
$> docker push <nom usager docker hub>/<nouveau nom image>:<tag>
```

Si on veut ajouter une nouvelle image ou mettre à jour celle-ci, on répète les étapes précédentes.

Et voilà! Nous avons une image conteneurisé Docker disponible à être utilisée par Kubernetes.

Utilisation de Kubernetes

Enfin! C'est pas trop tôt!

Commençons par créer un déploiement :

```
$> microk8s.kubectl create deployment <nom du déploiement> --image=<lien vers l'image>
```

On peut voir les déploiements avec `microk8s.kubectl get deployments`

Puis il faut exposer ce déploiement au reste du monde avec un service:

```
$> microk8s.kubectl create service nodeport <nom du déploiement> --tcp=80:80
```

On peut voir les services avec `microk8s.kubectl get services`

Pour voir notre application en ligne, on doit trouver une information dans le service : le port du noeud

```
$> microk8s.kubectl get service
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
k8stuto       NodePort    10.152.183.58   <none>        80:30618/TCP     3s
kubernetes    ClusterIP    10.152.183.1    <none>        443/TCP          3m20s
```

Puis dans un navigateur (ou via curl), on ouvre l'url suivant:

```
http://<ip du serveur>:<port du noeud>
```

Si vous vous heurtez à une erreur 403:Forbidden, voici une solution :

```
$> microk8s.kubectl exec <nom du pod> chmod +rwx /usr/share/nginx/html/index.html
```

La commande `exec` permet de lancer une commande sur le pod.
Pour voir la liste des pods :

```
$> microk8s.kubectl get pods
#cherchez les pods dont le nom commence par celui du déploiement
```

Et voilà la stricte base pour publier un service via Kubernetes

Mise à jour de l’application

Quoi? L’application est en train d’évoluer!?

Disons que nous avons publié une nouvelle version de l’application sur le docker hub, comment fait-on pour mettre notre déploiement Kubernetes à jour?

Cette commande fait tout le travail. Elle fait une mise à jour roulante sur les multiples pods du déploiements à mesure que ceux-ci ne sont plus utilisés :

```
$> microk8s.kubectl set image deployments/<nom du déploiement> <nom du déploiement>=<url de l’image 2.0>
```

```
node2@node2:~/docker$ microk8s.kubectl get pods
NAME                       READY   STATUS    RESTARTS   AGE
k8stuto-785ff848d7-625dl   1/1     Running   0          6m51s
node2@node2:~/docker$ microk8s.kubectl scale deployments/k8stuto --replicas=5
deployment.apps/k8stuto scaled
node2@node2:~/docker$ microk8s.kubectl get pods
NAME                       READY   STATUS              RESTARTS   AGE
k8stuto-785ff848d7-625dl   1/1     Running             0          7m31s
k8stuto-785ff848d7-bfsxs   0/1     ContainerCreating   0          2s
k8stuto-785ff848d7-dvj9f   1/1     Running             0          2s
k8stuto-785ff848d7-ffn7q   0/1     ContainerCreating   0          2s
k8stuto-785ff848d7-q7gfr   1/1     Running             0          2s
node2@node2:~/docker$ microk8s.kubectl get pods
NAME                       READY   STATUS    RESTARTS   AGE
k8stuto-785ff848d7-625dl   1/1     Running   0          7m34s
k8stuto-785ff848d7-bfsxs   1/1     Running   0          5s
k8stuto-785ff848d7-dvj9f   1/1     Running   0          5s
k8stuto-785ff848d7-ffn7q   1/1     Running   0          5s
k8stuto-785ff848d7-q7gfr   1/1     Running   0          5s
node2@node2:~/docker$
node2@node2:~/docker$ microk8s.kubectl set image deployments/k8stuto k8stuto=docker.io/duptom44/k8stuto:v2
deployment.apps/k8stuto image updated
node2@node2:~/docker$ microk8s.kubectl get pods
NAME                       READY   STATUS        RESTARTS   AGE
k8stuto-68bccd99d6-8djm8   1/1     Running       0          3s
k8stuto-68bccd99d6-kkmgs   0/1     Pending       0          0s
k8stuto-68bccd99d6-m26tm   1/1     Running       0          3s
k8stuto-68bccd99d6-mwhgs   0/1     Pending       0          0s
k8stuto-68bccd99d6-sfdgp   1/1     Running       0          3s
k8stuto-785ff848d7-625dl   1/1     Running       0          7m53s
k8stuto-785ff848d7-dvj9f   1/1     Terminating   0          24s
k8stuto-785ff848d7-ffn7q   1/1     Terminating   0          24s
k8stuto-785ff848d7-q7gfr   1/1     Terminating   0          24s
node2@node2:~/docker$ microk8s.kubectl get pods
NAME                       READY   STATUS              RESTARTS   AGE
k8stuto-68bccd99d6-8djm8   1/1     Running             0          6s
k8stuto-68bccd99d6-kkmgs   1/1     Running             0          3s
k8stuto-68bccd99d6-m26tm   1/1     Running             0          6s
k8stuto-68bccd99d6-mwhgs   0/1     ContainerCreating   0          3s
k8stuto-68bccd99d6-sfdgp   1/1     Running             0          6s
k8stuto-785ff848d7-625dl   1/1     Terminating         0          7m56s
k8stuto-785ff848d7-dvj9f   0/1     Terminating         0          27s
k8stuto-785ff848d7-ffn7q   0/1     Terminating         0          27s
k8stuto-785ff848d7-q7gfr   0/1     Terminating         0          27s
node2@node2:~/docker$ microk8s.kubectl get pods
NAME                       READY   STATUS    RESTARTS   AGE
k8stuto-68bccd99d6-8djm8   1/1     Running   0          24s
k8stuto-68bccd99d6-kkmgs   1/1     Running   0          21s
k8stuto-68bccd99d6-m26tm   1/1     Running   0          24s
k8stuto-68bccd99d6-mwhgs   1/1     Running   0          21s
k8stuto-68bccd99d6-sfdgp   1/1     Running   0          24s
```

Multiplication des conteneurs

Voilà ce qui arrive quand on les arrose les gremlins.

Une des grandes forces de kubernetes est la multiplication des conteneurs.
De plus, puisque notre service est de type nodeport, elle fait également office de balanceur de charge entre les conteneurs. Voici la commande :

```
$> microk8s.kubectl scale deployments/<nom du déploiement> --replicas=<nombre de conteneurs>
```

On peut également laisser kubernetes décider quand ajouter un nouveau conteneur automatiquement :

```
$> microk8s.kubectl autoscale deployments/<nom du déploiement> --min=<nombre de conteneurs> --max=<nombre de conteneurs> --cpu-percent=<taux d’utilisation du processeur qui active la multiplication>
```

Commandes supplémentaires

Voir les déploiements :

```
$> microk8s.kubectl get deployments
```

Lancer le dashboard :

```
$> microk8s.enable dashboard
$> microk8s.kubectl proxy
#Préférablement faire le proxy sur une autre fenêtre, car elle va verrouiller la console
```

Lien vers le dashboard :

http://127.0.0.1:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/

Fin du tutoriel

Ce fut court mais ce fut bon! (Enfin, pour vous... 😅)

Des questions?