

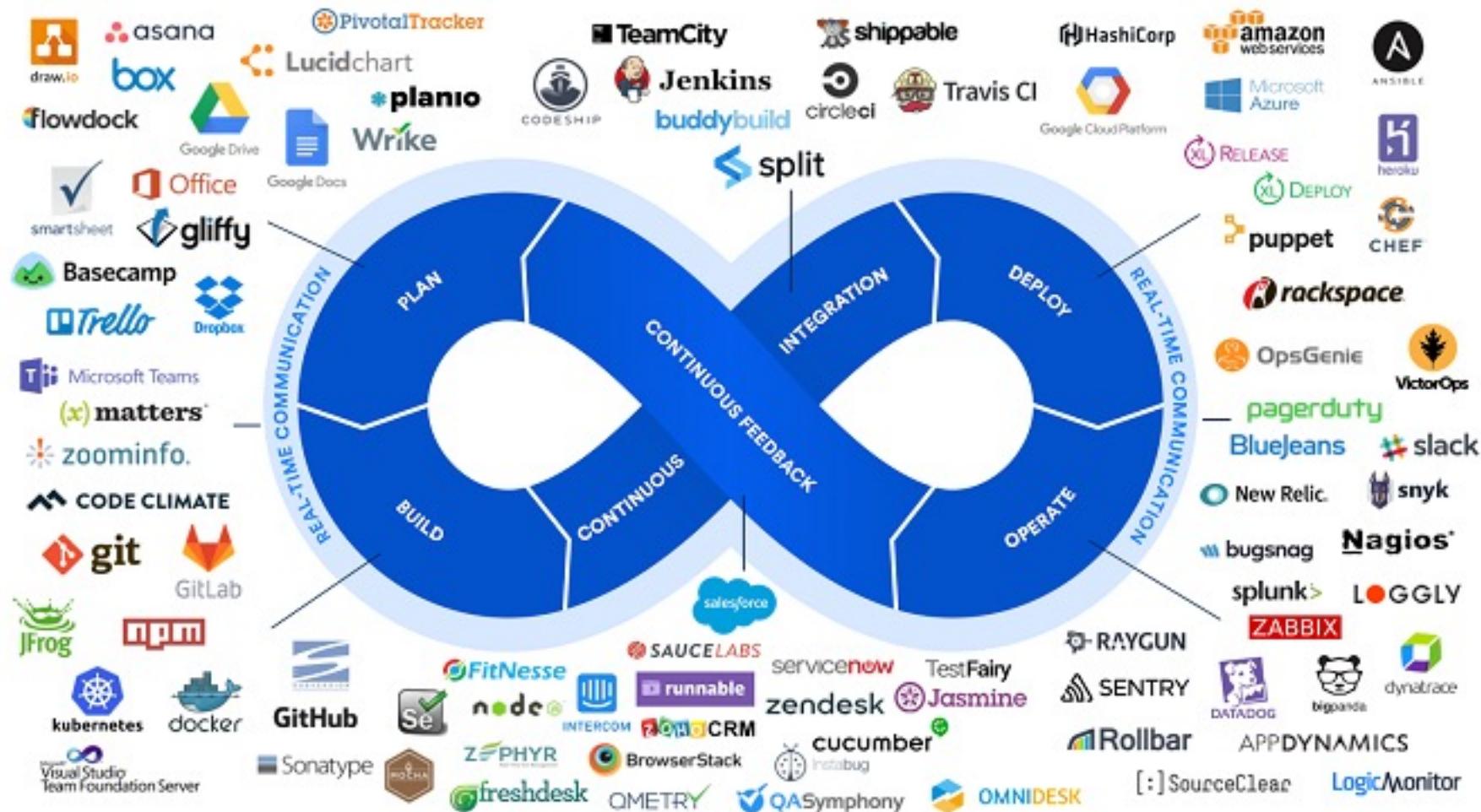
Formation DevSecOps-Docker- Ansible-Terraform

Ihab ABADI / UTOPIOS

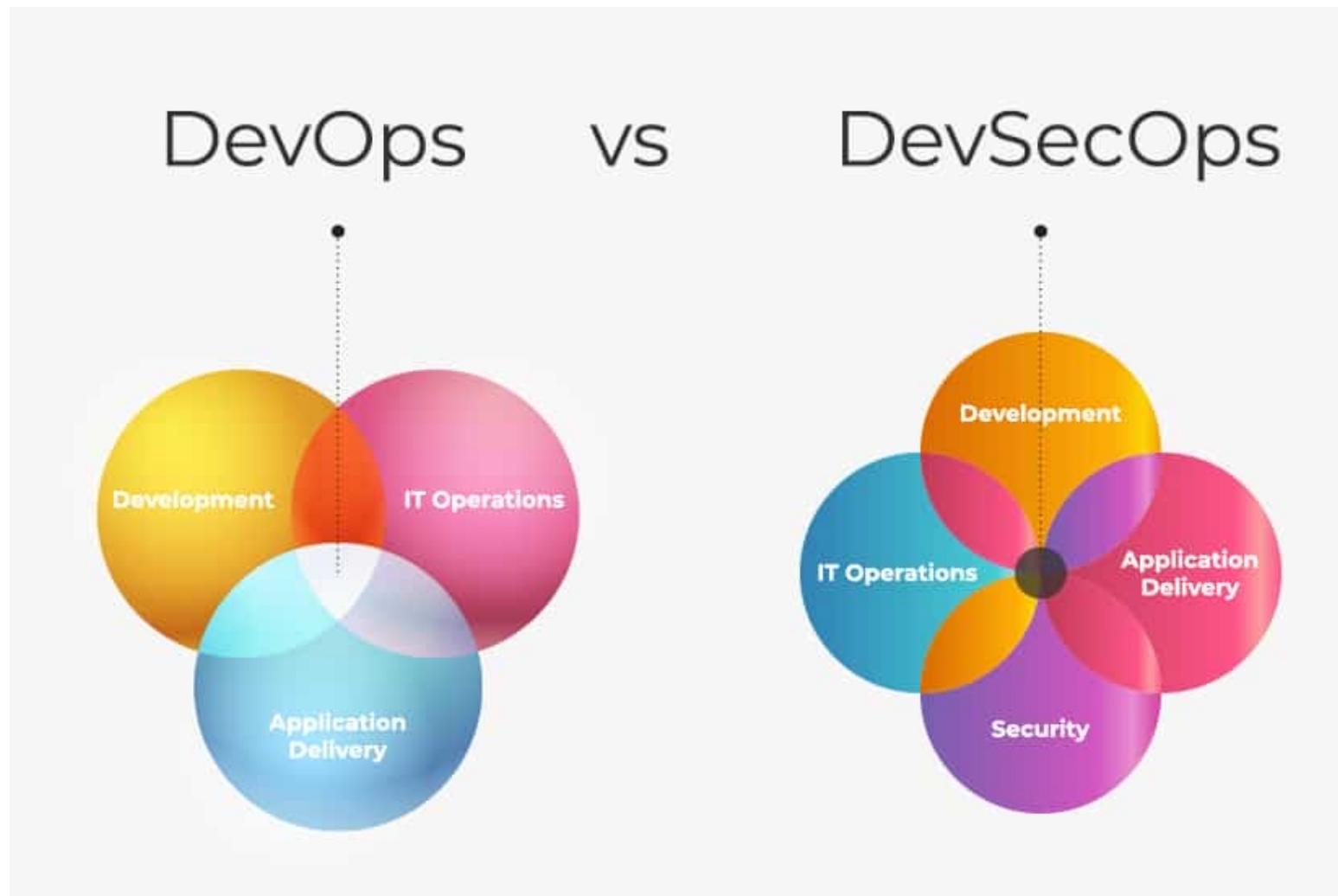
SOMMAIRE – Introduction DevSecOps

1. Rappel Philosophie DevOps.
2. Inclure la notion de sécurité dans le DevOps.
3. Pipeline DevSecOps
4. Sécurisation des applications

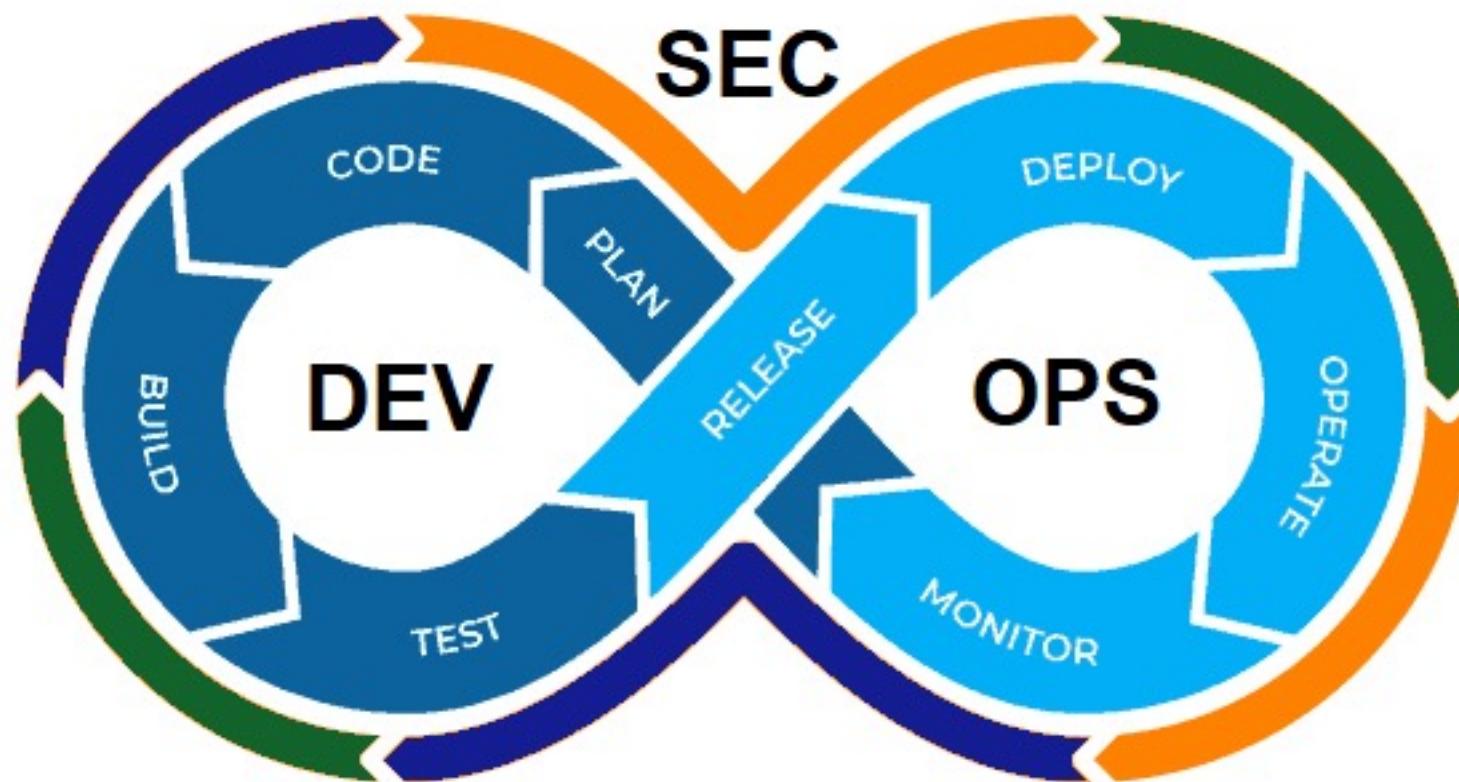
Rappel DevOps



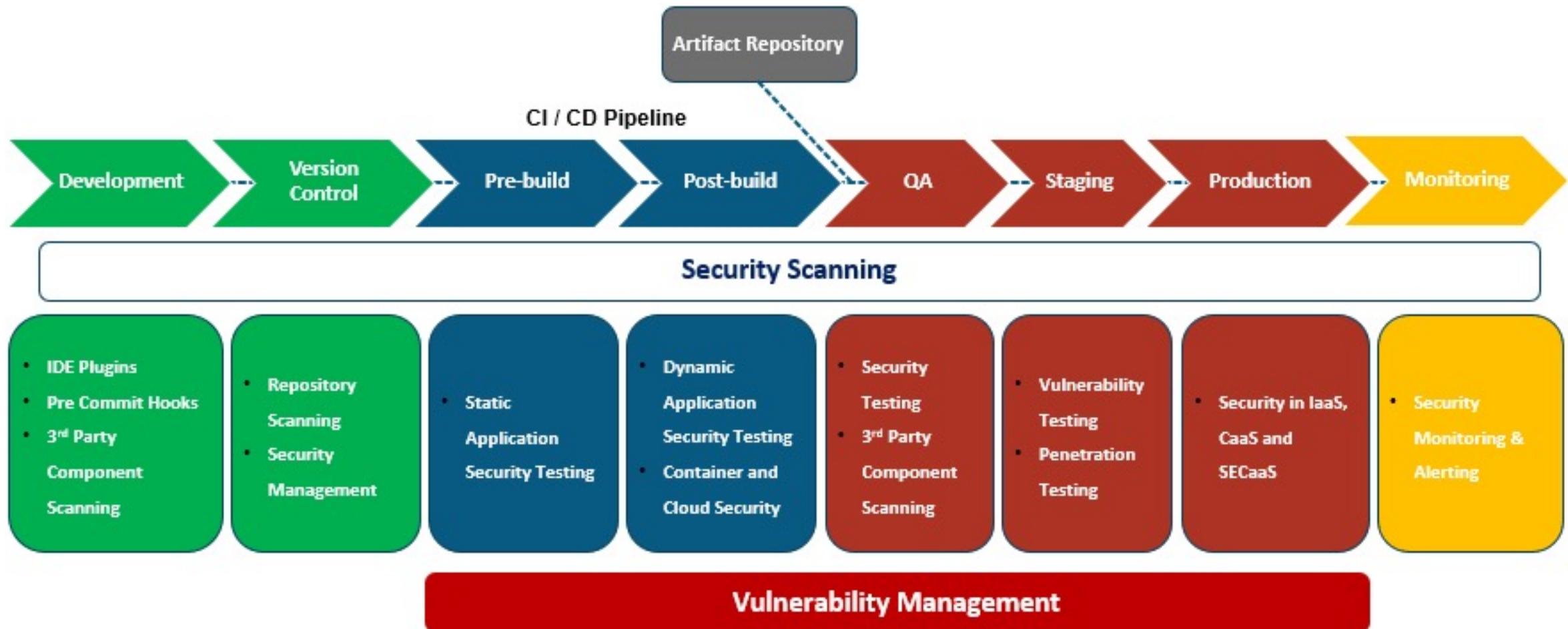
Introduction DevSecOps



Introduction DevSecOps



PipeLine DevSecOps



Sécurisation des Applications

- 1- SCA software composition analysis
- 2- DAST Dynamic Application testing
- 3- SAST Static Application security testing

SOMMAIRE - Docker

Partie 1 Rappel Docker

- 1- Introduction docker / virtualisation
- 2- Docker / Isolation
- 3- Avantages conteneurisation
- 4- Rappel linux et définition
- 5- Fonctionnement docker
- 6- Installation docker
- 7- Image docker
- 8- Conteneur Docker
- 9- Volume docker
- 10- Dockerfile
- 11- Docker compose
- 12- Nettoyage docker

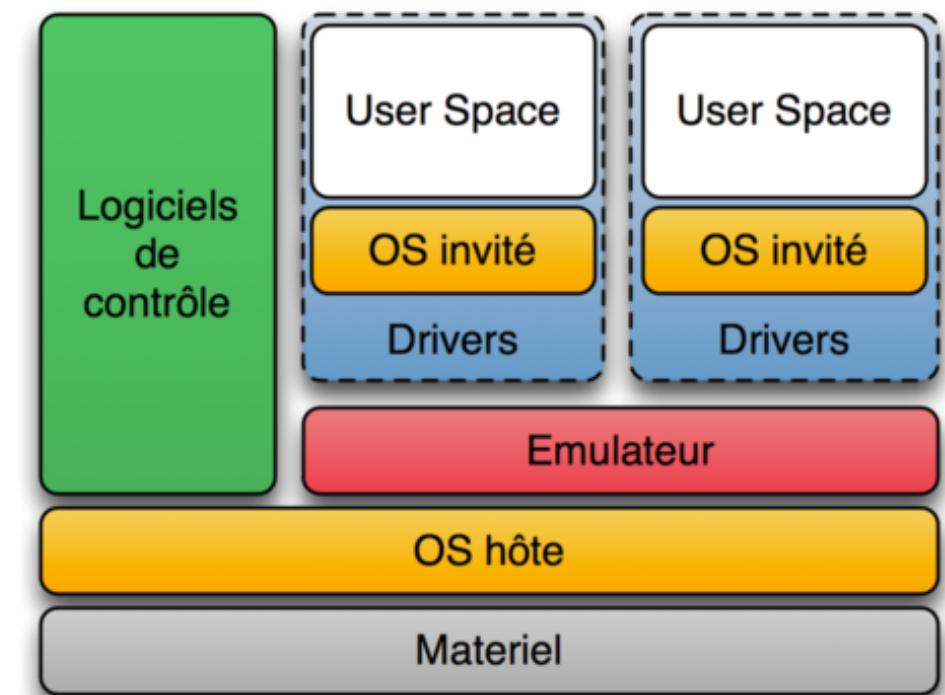
SOMMAIRE - Docker

Partie 2 -

- 1- Utilisation des UID, GID avec Docker
- 2- Utilisation de namespace remapping et Cgroup
- 3- Attack surface
- 4- L'utilisation des volumes pour Escalade de privilèges
- 5- L'utilisation des capacités
- 6- docker.sock
- 7- docker.sock – Evasion des conteneurs
- 8- docker API

INTRODUCTION DOCKER- Virtualisation

- La virtualisation est la capacité de faire tourner un, ou plusieurs serveur virtuel sur une seul machine physique grâce à un hyperviseur.
- L'hyperviseur permet d'émuler les différentes ressources matérielles d'un serveur physique et permet à des machines virtuelles de les partager.
- Une machine virtuelle possède ses propres ressources matérielles et son propre système d'exploitation



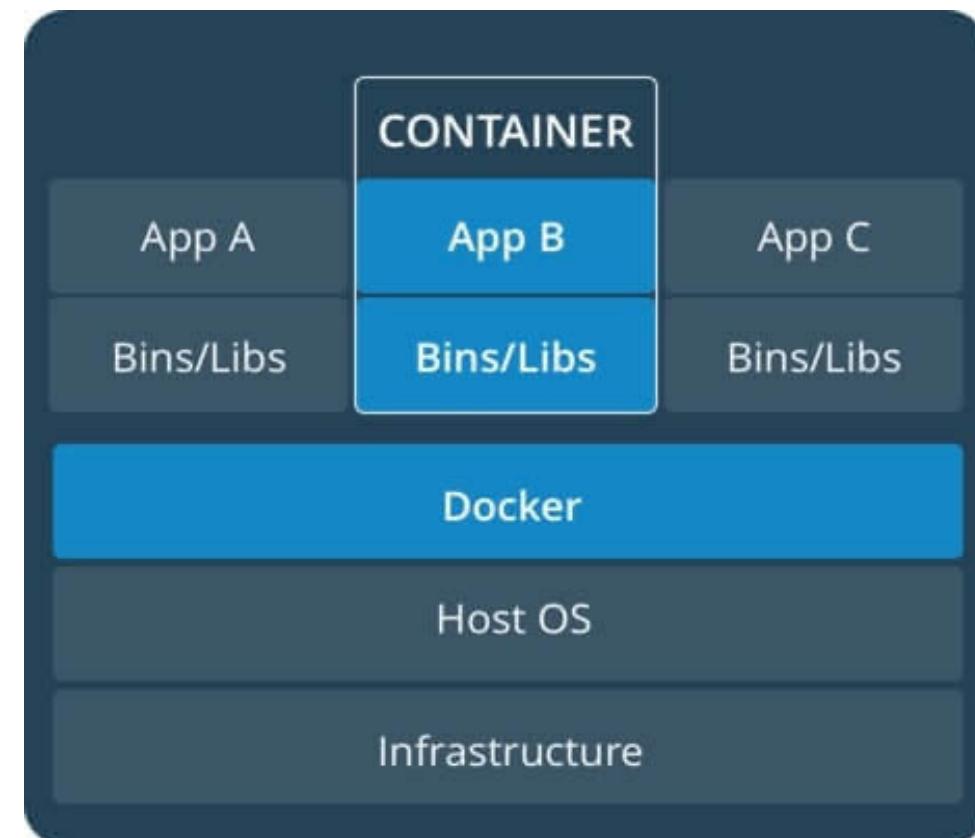
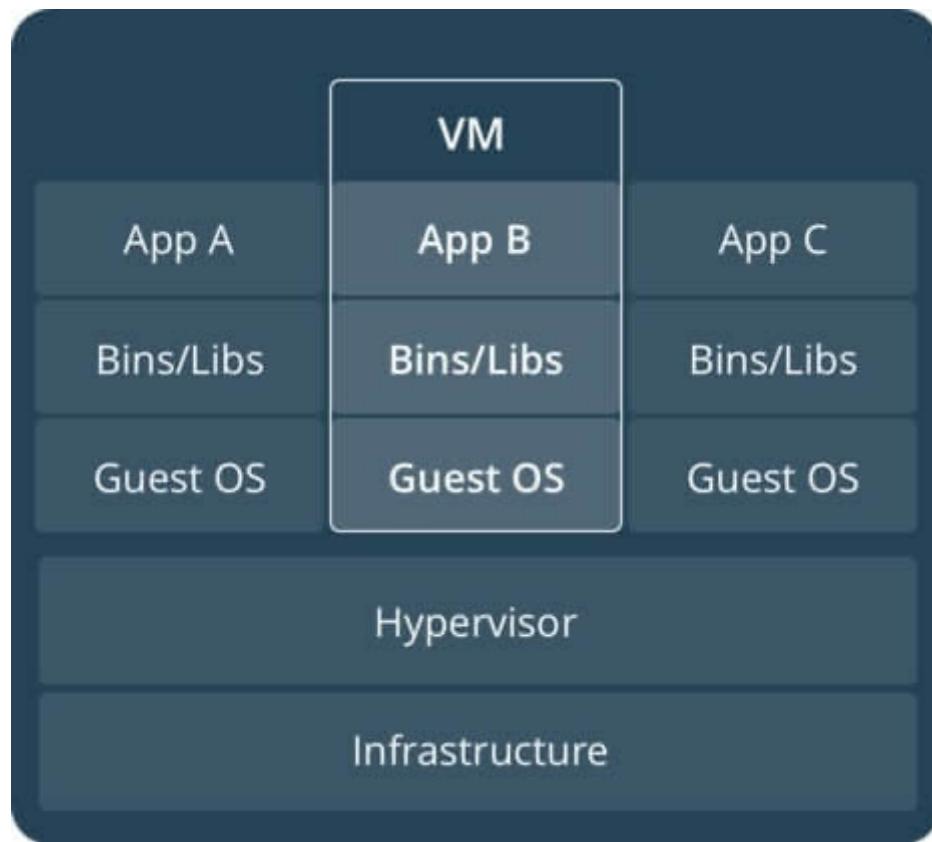
INTRODUCTION DOCKER- Virtualisation

- **La virtualisation apporte des avantages :**
 - Ressources adaptées aux besoins de l'application.
 - Faciliter de manipulation des VMs (Sauvegarde, bascule,...)
 - Réduction des dépenses et réduction d'équipements nécessaires
 - Faciliter dans l'administration
- **La virtualisation a des inconvénients également :**
 - Réduction des performances
 - Multiplication des couches OS
- **Conclusion**
 - La virtualisation est une technologie avec beaucoup d'avantages mais avec certain inconvénients bloquant, d'où la nécessité de pousser vers une nouvelle technologie qui répond à ces inconvénients

DOCKER – Isolation

- **Isolation en VM:**
 - Se fait au niveau matérielle
 - Accès virtuel aux ressources via l'hôte à l'aide de l'hyperviseur
- **Isolation Dans la conteneurisation:**
 - Se fait au niveau du système d'exploitation
 - Exécution native sur linux et partage du noyau hôte avec les conteneurs
- **Conclusion**
 - La virtualisation est une technologie avec beaucoup d'avantages mais avec certain inconvénients bloquant, d'où la nécessité de pousser vers une nouvelle technologie qui répond à ces inconvénients

DOCKER – Isolation



DOCKER – Avantages

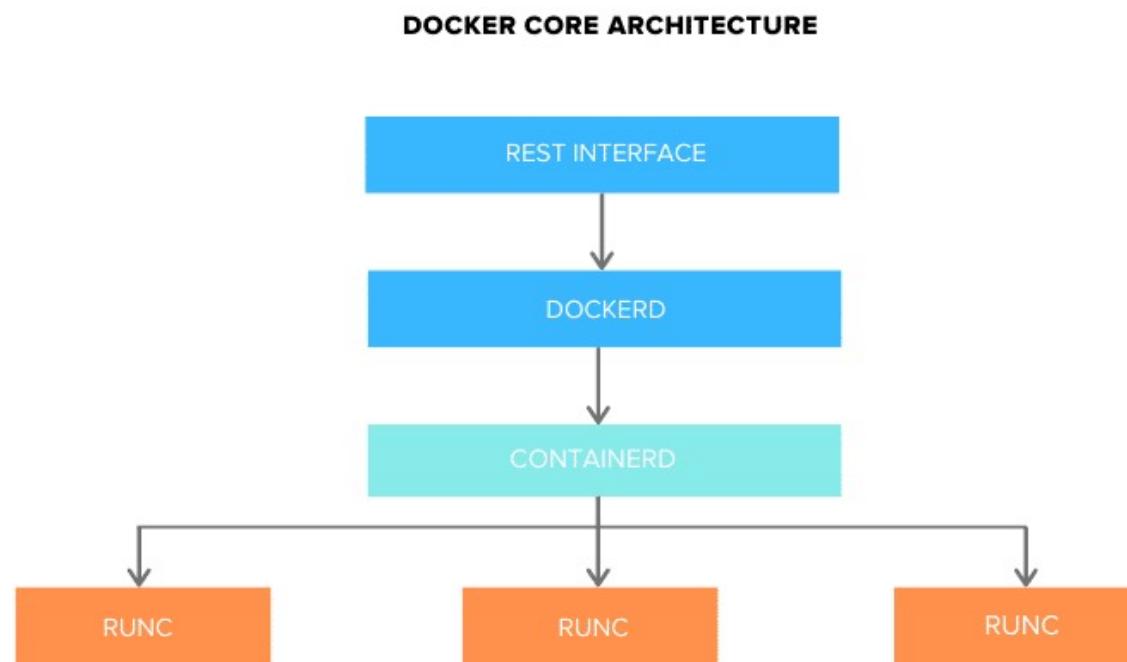
- **Avantages :**
 - Flexibilité quelque soit la complexité de l'application
 - Légereté grâce au partage du noyau du hôte
 - Scalabilité ou Extensibilité
 - ...

DOCKER – Rappel Linux et définition

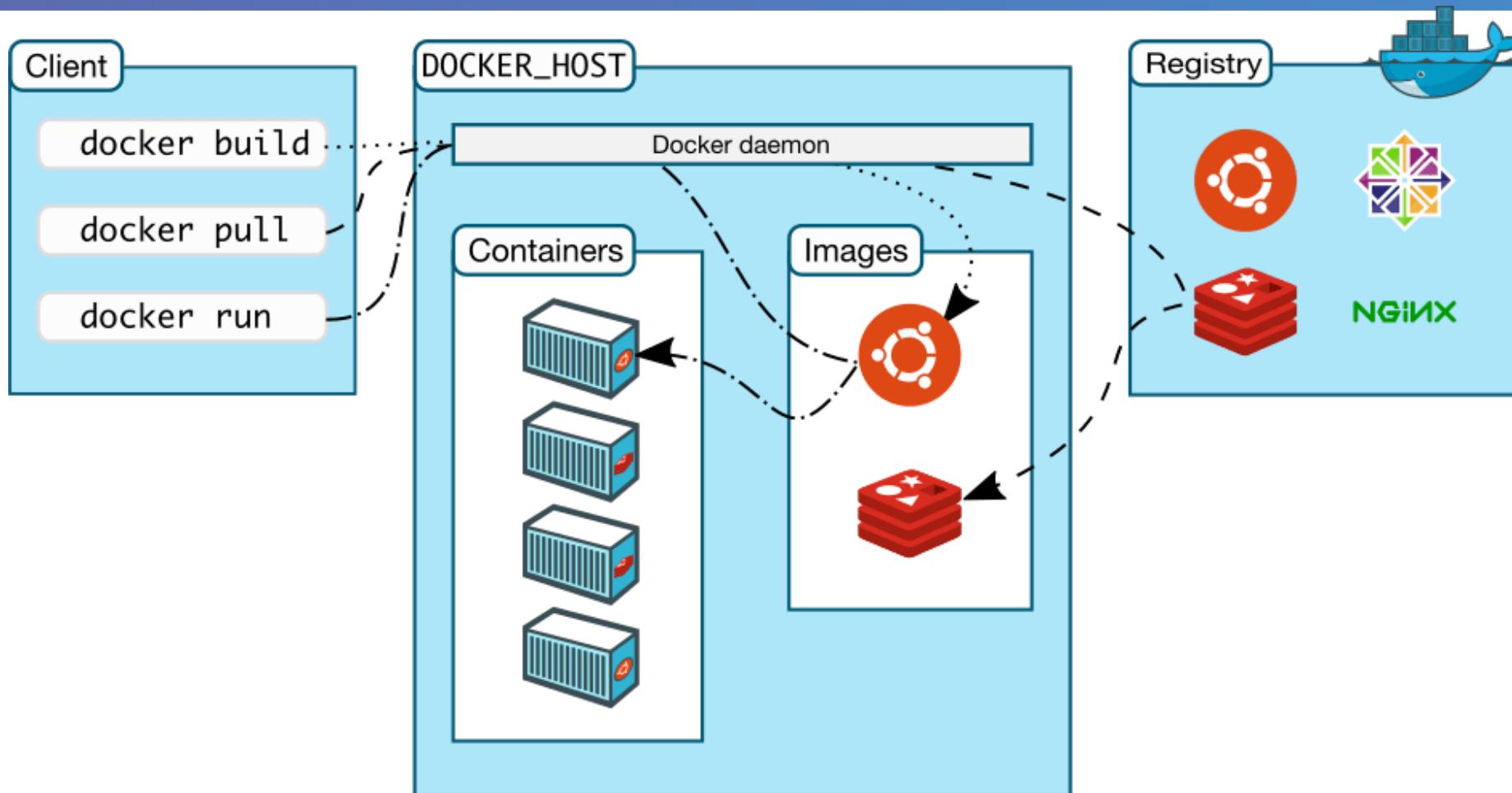
- Un conteneurs et un processus linux isolé à l'aide des namespaces et cgroups
- Les namespaces linux est un mécanisme qui permet de limiter l'accès d'un processus
- Les cgroups linux est un mécanisme qui permet de limiter l'accès au ressources d'un processus
- Quelques exemple de namespaces:
 - Namespace PID
 - Namespace IPC
 - Namespace USER
- Quelques exemple de cgroups
 - Cgroup cpuset
 - Cgroup cpu
 - Cgroup memory

DOCKER – définition

- Docker est une technologie qui permet la conteneurisation sur Linux
- Docker démocratise l'utilisation des conteneur
- Docker est composé de :
- Docker engine
- Docker-containerd
- Docker-runc



DOCKER – Fonctionnement



DOCKER – Installation

- Docker peut être installer sur toute type de distribution
- Windows
- Linux
- MacOS
- <https://docs.docker.com/get-docker/>

DOCKER – Image

- Sur Docker, un conteneur est lancé à partir d'une image
- Une image est un package qui inclut les fonctionnalités nécessaires à l'exécution de notre processus
- Le contenu de l'image est :
 - Le code
 - Les variables d'environnement
 - Les fichiers de configuration
 - Les bibliothèques
- On peut créer une image de plusieurs façons
 - A partir d'un conteneur
 - A partir d'un fichier DockerFile
- Docker fournit un CLI pour gérer les images sur notre machine

DOCKER – Commande pour Image

- Pour lister les images sur la machines : docker images
- Pour rechercher une image sur le hub docker : docker search
- Pour télécharger une image à partir du hub : docker pull
- Pour exécuter une image : docker run
 - Chaque image a un nom unique
 - Chaque image peut avoir une description
 - Chaque image a plusieurs tags
- Pour supprimer une image : docker rmi nom_image

DOCKER – Conteneur

- Un conteneur est la représentation en mémoire lors de l'exécution d'une image
- Pour créer un conteneur, on utilise la commande docker run [OPTIONS] <nom_image>
- Lors de la création d'un conteneur, on peut spécifier une ou plusieurs options
- Quelques options :
- -t : Allouer un pseudo TTY
- -i : Garder un STDIN ouvert
- -d: Exécuter le conteneur en arrière-plan
- -p :exposer un ou plusieurs ports
- --name : donner un nom au container

DOCKER – Conteneur

- Pour afficher la liste des conteneurs docker container ls ou docker ps
- On peut utiliser l'option -a pour afficher la totalité des conteneurs quelque soit le statut
- Chaque conteneur a :
 - Container ID : id du conteneur
 - IMAGE : l'image utilisée pour créer le conteneur
 - COMMAND : dernière commande lancée lors de l'exécution de l'image
 - CREATED : date de création du conteneur
 - STATUS : statut du conteneur
 - PORTS : les ports utilisés par le conteneur
 - NAME : nom du conteneur

DOCKER – Conteneur

- Un conteneur passe par plusieurs états
 - created : conteneur créé mais non démarré (cet état est possible avec la commande docker create)
 - restarting : conteneur en cours de redémarrage
 - running : conteneur en cours d'exécution
 - paused : conteneur stoppé manuellement (cet état est possible avec la commande docker pause)
 - exited : conteneur qui a été exécuté puis terminé
 - dead : conteneur que le service docker n'a pas réussi à arrêter correctement (généralement en raison d'un périphérique occupé ou d'une ressource utilisée par le conteneur)

DOCKER – Conteneur

- Pour supprimer un conteneur :
- Docker rm <container_id> ou <container_name>
- Pour exécuter une commande dans un conteneur
- Docker exec <OPTIONS> <container_id ou container_name> command
- Pour convertir un conteneur en Image
- Docker commit <container_id ou container_name> <image_name>

DOCKER – Volumes

- Les données dans un conteneur sont éphémères.
- Pour sauvegarder les données d'un conteneur, on peut le convertir à chaque fois en image (mauvaise pratique)
- La deuxième solution, consiste à utiliser les volumes

DOCKER – Volumes (Système de fichier docker)

- Docker utilise un système de fichier en calques
- Chaque image se compose de pile de calques en lecture seule
- A la création d'un conteneur un calque est ajouté dans le haut de pile en lecture-écriture
- Lors d'une modification de fichier, Docker crée une copie depuis les couches en lecture seule vers le layer en lecture-écriture.
- Lors d'une création de fichier, Docker crée le fichier que sur le layer en lecture-écriture, et ne touche pas au layer en lecture seule.
- Lors d'une suppression de fichier, Docker supprime le fichier que sur le layer en lecture-écriture, et s'il est déjà existant dans le layer en lecture seule alors il le garde.
- Les données dans le layer de base sont en lecture seule, elles sont donc protégées et intactes par toutes modifications, seul le layer en lecture-écriture est impacté lors de modifications de données.
- Lorsqu'un conteneur est supprimé, le layer en lecture-écriture est supprimé avec. Cela signifie que toutes les modifications apportées après le lancement du conteneur auront disparus avec.

DOCKER – Volumes (Volume non lié)

- Docker permet la création de volume en dehors de tout conteneur
- Pour créer un volume, on utilise la commande
- Docker volume create <nom_volume>
- Pour lister des volumes
- Docker volume ls
- Pour supprimer un volume
- Docker volume rm <nom_volume>
- Pour créer un conteneur avec volume, on utilise l'option v
- Docker run -v <nom_volume>:<dossier_conteneur>

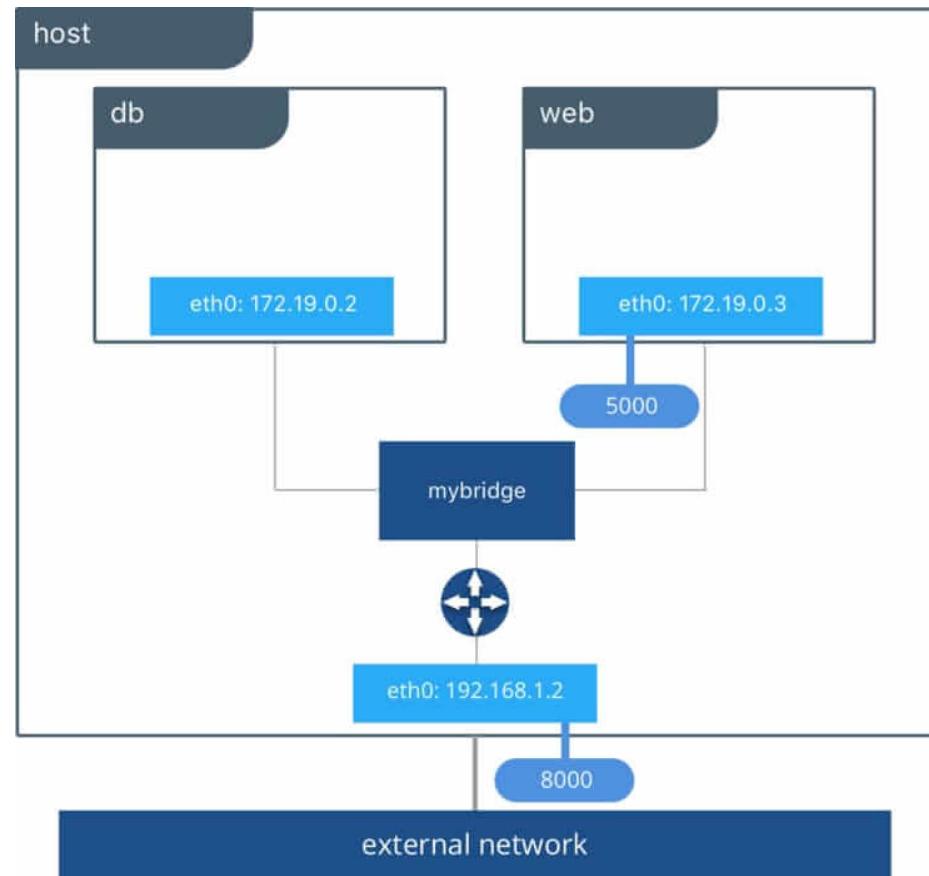
DOCKER – Volumes (Volume lié)

- On peut également monter un volume local sur un conteneur
- Docker run -v <volume_local>:<volume_conteneur>

DOCKER – Gestion de réseau

- Le réseau docker utilise des drivers réseau
- Par défaut docker crée un driver Bridge
- Chaque nouveau conteneur est automatiquement connecté à ce réseau
- Les conteneurs qui utilisent ce driver, ne peuvent communiquer qu'entre eux.
- Il ne sont pas accessibles de l'extérieur

DOCKER – Gestion de réseau



DOCKER – Gestion de réseau

- Il existe d'autre type de driver
- Driver none
- Driver host
- Driver overlay
- Driver macvlan

DOCKER – Gestion de réseau

- Pour créer un réseau docker
- docker network create --driver <DRIVER TYPE> <NETWORK NAME>
- Pour lister les réseaux docker
- Docker network ls
- Pour inspecter un réseau
- Docker network inspect <network_name>
- Un conteneur docker est par défaut connecté à un réseau de type Bridge
- Pour connecter un conteneur au moment de la création, on utilise l'option --network
- Pour connecter un conteneur déjà créé, on utilise la fonction connect de docker network
- Pour déconnecter un conteneur déjà créé, on utilise la fonction disconnect de docker network
- Des conteneurs dans le même réseau peuvent communiquer entre eux
- Les conteneurs dans un bridge ne peuvent pas communiquer avec l'extérieur qu'à travers des ports exposés vers l'hôte
- On peut exposer des ports uniquement au démarrage d'un conteneur avec l'option p
- Exemple
- Docker run -p <port hot>:<port container> --name=<nom_du_container> <image_container>

DOCKER – Dockerfile

- Dockerfile est fichier qui permet de créer une image docker à l'aide de Docker DSL
- DSL pour dockerfile contient une multitude d'instruction
- FROM : Définit l'image de base qui sera utilisée par les instructions suivantes.
- LABEL : Ajoute des métadonnées à l'image avec un système de clés-valeurs, permet par exemple d'indiquer à l'utilisateur l'auteur du Dockerfile.
- ARG : Variables temporaires qu'on peut utiliser dans un Dockerfile.
- ENV : Variables d'environnements utilisables dans votre Dockerfile et conteneur.
- RUN : Exécute des commandes Linux ou Windows lors de la création de l'image. Chaque instruction RUN va créer une couche en cache qui sera réutilisée dans le cas de modification ultérieure du Dockerfile.

DOCKER – Dockerfile

- COPY : Permet de copier des fichiers depuis notre machine locale vers le conteneur Docker.
- ADD : Même chose que COPY mais prend en charge des liens ou des archives (si le format est reconnu, alors il sera décompressé à la volée).
- ENTRYPOINT : comme son nom l'indique, c'est le point d'entrée de votre conteneur, en d'autres termes, c'est la commande qui sera toujours exécutée au démarrage du conteneur. Il prend la forme de tableau JSON (ex : CMD ["cmd1","cmd1"]) ou de texte.
- CMD : Spécifie les arguments qui seront envoyés au ENTRYPOINT, (on peut aussi l'utiliser pour lancer des commandes par défaut lors du démarrage d'un conteneur). Si il est utilisé pour fournir des arguments par défaut pour l'instruction ENTRYPOINT, alors les instructions CMD et ENTRYPOINT doivent être spécifiées au format de tableau JSON.
- WORKDIR : Définit le répertoire de travail qui sera utilisé pour le lancement des commandes CMD et/ou ENTRYPOINT et ça sera aussi le dossier courant lors du démarrage du conteneur.
- EXPOSE : Expose un port.
- VOLUMES : Crée un point de montage qui permettra de persister les données.
- USER : Désigne quel est l'utilisateur qui lancera les prochaines instructions RUN, CMD ou ENTRYPOINT (par défaut c'est l'utilisateur root).

DOCKER – Dockerfile COPY et ADD

- Ils permettent tous les deux de copier un fichier/dossier local vers un conteneur.
- ADD autorise les sources sous forme d'url et si jamais la source est une archive dans un format de compression reconnu (ex : zip, tar.gz, etc ...), alors elle sera décompressée automatiquement vers votre cible.
- Dans les best-practices de docker, ils recommandent d'utiliser l'instruction COPY quand les fonctionnalités du ADD ne sont pas requises.

DOCKER – Dockerfile ENV et ARG

- Ils permettent tous les deux de stocker une valeur
- ARG permet de stocker en tant que variable temporaire, utilisable qu'au niveau de votre Dockerfile.
- ENV permet de stocker une variable d'environnements accessible depuis le Dockerfile et votre conteneur.

DOCKER – Dockerfile RUN, CMD, ENTRYPOINT

- L'instruction RUN est exécutée pendant la construction de votre image, elle est souvent utilisée pour installer des packages logiciels qui formeront les différentes couches de votre image.
- L'instruction ENTRYPOINT est exécutée pendant le lancement de votre conteneur et permet de configurer un conteneur qui s'exécutera en tant qu'exécutable.
- L'instruction CMD est aussi exécutée pendant le lancement de votre conteneur, elle définit les commandes et/ou les paramètres de l'instruction ENTRYPOINT par défaut, et qui peuvent être surchargées à la fin de la commande docker run.

DOCKER – Docker compose

- Docker Compose est un outil permettant de **définir le comportement de vos conteneurs et d'exécuter des applications Docker à conteneurs multiples.**
- La config se fait à partir d'un fichier YAML, et ensuite, avec une seule commande, vous **créez et démarrez tous vos conteneurs de votre configuration.**
- Installation de docker compose

DOCKER – Docker compose

- Pour utiliser docker compose, il faut créer un fichier docker-compose.yml
- Docker compose lance les conteneurs en tant que services
- Dans docker-compose.yml on doit :
- Indiquer la version (Il existe plusieurs version de docker-compose, les versions à partir du 3 sont compatible avec docker swarm)
- Indiquer les services (les services sont nos conteneurs)
- Chaque service est défini par son nom
- Chaque service peut avoir les options suivantes:
- Image
- Container_name
- Restart
- Volumes
- Environment
- Ports

DOCKER – Docker compose

- Docker compose fournit un cli pour démarrer les services et les arrêter
- Commande pour démarrer les services à partir d'un docker compose
- Docker-compose up
- Commande pour arrêter les services
- Docker-compose down
- On peut ajouter –d au commande de démarrage de services pour les démarrer en arrière plan

DOCKER – Docker compose

- Première exemple de docker compose
- Démarrer deux services
- Premier service pour démarrer un serveur web
- Deuxième service pour démarrer une base de données mysql

DOCKER – Docker compose

- Docker compose permet de générer des services directement à partir de dockerfile à l'aide de l'option build
- Dans l'option build, il faut indiquer le chemin du dockerfile à l'utiliser pour créer le conteneur

DOCKER – Nettoyage

- Docker nous donne la possibilité d'agir sur les process linux pour effectuer un nettoyage.
- Des images.
- Des conteneurs.
- On peut effectuer un nettoyage à l'aide de la commande prune.
- Exemple.

DOCKER – registry

- Docker Registry est une application open source qui permet de mettre en place un serveur de distribution d'image Docker
- L'utilisation des Docker registry est recommandé dans les cas suivants :
- Contrôler la sécurité d'accès au images Docker
- Contrôler l'utilisation des images dans nos pipelines d'intégration et de déploiement

DOCKER – Cration

- Pour crer un Docker Registry :
- On peut installer directement docker-registry
- On peut utiliser l’image Docker registry pour dployer un Docker registry

DOCKER – Registry Déploiement d'image

- Pour déployer une image sur un Docker-registry privé
- Il faut créer un tag pour identifier l'image
- Le tag doit être sous le format suivant :
- <SERVER NAME REGISTRY>:<PORT SERVER REGISTRY>/<IMAGE_NAME>
- Le déploiement se fait à l'aide de la commande docker push
- La récupération se fait à l'aide de la commande docker pull
- Dans le cadre, de l'utilisation de l'image registry, on peut accéder au images directement par API rest des ressources

DOCKER – Débogage des conteneurs

Pour déboguer un conteneur, Docker nous met à disposition, une multitude de fonctionnalités pour analyser le comportement de nos conteneurs.

On peut trouver :

- Docker stats : cette fonctionnalité nous fournit un rapport d'analyse sur l'utilisation des ressources par le conteneur
- La commande docker stats permet d'utiliser l'option 'format' pour un meilleur export des résultats
- Docker inspect : cette fonctionnalité nous fournit des informations détaillées sur le conteneur sous format json, elle accepte également l'option 'format'
- Docker logs : cette fonctionnalité nous fournit la commande en cours d'exécution sur notre conteneur
- Docker history: cette fonctionnalité nous fournit des informations sur l'historique de la construction de l'image

DOCKER – Débogage des conteneurs

Exercice:

Réaliser un script qui permet, à l'aide des fonctionnalités de débogage, d'extraire la liste :

- Des adresses IP de chaque conteneurs.
- Les adresses MAC
- Les ports utilisés
- Les adresses GATEWAY

DOCKER – Débogage des conteneurs

Exercice:

Réaliser un script qui permet, à l'aide des fonctionnalités de débogage, d'extraire la liste :

- Des adresses IP de chaque conteneurs.
- Les adresses MAC
- Les ports utilisés
- Les adresses GATEWAY

DOCKER – UID et GID

- Docker crée les conteneurs avec uid 0
- Nous avons la possibilité de spécifier UID ou GID au moment de créer l'image pour indiquer user à utiliser dans notre conteneur.
- Nous avons également la capacité de surcharger l'utilisateur au moment de créer le conteneur.
- Exemple

DOCKER – Utilisation de namespace remapping et Cgroup

- Dans le cadre d'applications qui doivent être exécuter à l'intérieur d'un conteneur avec des privilèges root, mais avec une isolation du root système, on utilisera le mécanisme de namespace remapping.
- Le namespace remapping consiste à mapper un utilisateur non root du système vers root à l'intérieur du conteneur.
- La configuration peut se faire à l'intérieur du daemon.json
- Exemple

DOCKER – Utilisation de namespace remapping et Cgroup

- Les cGroups fournissent une limitation des ressources et une capacité de création de rapports dans l'espace du conteneur.
- Ils permettent également un contrôle granulaire sur les ressources d'hôte allouées aux conteneurs et à quel moment elles sont allouées.
- Les ressources qu'on peut limiter sont :
 - CPU
 - Memory
 - Network Bandwidth
 - Disk
- Exemple

DOCKER – Attack Surface

- Comme Docker nécessite des privilèges root pour fonctionner, il faut savoir qui a accès à l'hôte docker et au daemon docker.
- Il faut toujours séparer les conteneurs, pour limiter et séparer l'accès au daemon docker.
- Il faut également limiter les capacités de conteneurs strict nécessaire à l'exécution d'application.
- ...

DOCKER – L'utilisation des volumes pour Escalade de privilèges

- Comme un conteneur est créé par défaut avec les droits root, dans le cadre de montage de volume, le conteneur peut agir et modifier des éléments sur le l'hôte
- Exemple.
- Exercice
- Ajouter un utilisateur dans l'hôte à l'aide d'un conteneur qui partage le volume /etc/

DOCKER – L'utilisation des capacités

- Chaque conteneur crée possède les capacités suivantes:
- CHOWN, DAC_OVERRIDE, FSETID, FOWNER, MKNOD, NET_RAW, SETGID, SETUID, SETFCAP, SETPCAP, NET_BIND_SERVICE, SYS_CHROOT, KILL, AUDIT_WRITE
- Dans certains cas d'utilisations, on peut avoir le besoins de supprimer des capacités ou d'en ajouter.
- C'est possible à l'aide des flags –cap-add ou –cap-drop.
- Exemple

DOCKER – docker.sock

- docker.sock est le socket UNIX que le démon Docker écoute. C'est le principal point d'entrée de l'API Docker.
- Il peut également s'agir d'un socket TCP, défaut, Docker utilise par défaut le socket UNIX.
- Le client cli Docker utilise ce socket pour exécuter les commandes docker par défaut.
- Il peut y avoir différentes raisons pour lesquelles vous devrez peut-être monter le socket Docker à l'intérieur d'un conteneur.
- Exemple docker nginx.

DOCKER – docker.sock – Evasion des conteneurs

- Dans le cadre d'un deamon docker accessible par TCP.
- On peut exécuter des conteneurs avec un volume monté.
- On peut utiliser ce volume pour exécuter des actions sur le l'hôte.
- On peut même prendre le contrôle de l'hôte.
- Exemple.

DOCKER – docker API

- Docker expose une API rest qui permet d'interagir avec docker daemon.
- A l'aide de l'api, on peut:
 - Créer des images
 - Démarrer des conteneurs.
 - ...
 - Exemple

SOMMAIRE - Ansible

1. Introduction
 1. Ansible ?
 2. Autres produits
2. Mise en œuvre de Ansible
 1. Installation
 2. Définition de l'inventaire
 3. Les variables
 4. Les outils / les accès
3. Utilisation des modules
 1. Définition / syntaxe
 2. Exemples

SOMMAIRE - Ansible

4. Les Playbooks

1. Définition
2. Syntaxe
3. Exemples

5. Structures de contrôle

1. Définitions
2. Facts / Conditions / boucles / inclusions
3. Exemples

6. Templates / Rôles

1. Templates
2. Rôles

SOMMAIRE - Ansible

7- Surveillance des journaux et défense automatisée sans serveur
(Elastic Stack dans AWS)

8- Automatisation des tests de sécurité des applications Web à l'aide
d'OWASP ZAP

9- Durcissement de la sécurité :

 9-1 Pour les applications et les réseaux

 9-2 Avec des références telles que CIS, STIG et NIST

10- Automatisation des vérifications d'audit de sécurité :

 10-1 Pour les périphériques réseau à l'aide d'Ansible

 10-2 Pour les applications utilisant Ansible

11- Approches de correctifs automatisés à l'aide d'Ansible

Ansible - Introduction

1. Introduction

- Ansible ?

Ansible est un projet récent (2012) qui a été développé entièrement en python et qui répond à des besoins de remplacement, de déploiement ou de changements de serveur. Ces actions peuvent être menées sur l'ensemble ou une partie des serveurs.

C'est un système agentless capable de piloter des systèmes Windows, Linux, Unix et aussi des équipements réseau tel que Cisco ou Juniper.

Pour fonctionner ansible n'a besoin que d'un accès ssh et de python ou des APIs. Il n'y a pas de serveur central, tout ordinateur ayant Ansible peut commander les autres.

- Autres produits ?

Les autres produits qui permettent l'orchestration de serveur :

- **Puppet (client/serveur)**

C'est projet qui a été créé en 2005 et qui est écrit en RUBY.

- **Chef (client/serveur)**

Il a été créé en 2009 par des ex-employés de Puppetlabs. Il a été écrit en RUBY puis ré-écrit en Erlang.

- **Saltstack (client/serveur ou agentless)**

Cette plateforme a vu le jour en 2011. Elle est écrite en python.

Ansible – Mise en oeuvre

2. Mise en œuvre de ANSIBLE

- Installation

Ansible s'installe facilement et uniquement sur l'ordinateur de management.
Elle peut se faire via les paquets de la distribution (UBUNTU/DEBIAN) ou via des dépôts tiers EPEL (REDHAT/CENTOS). Elle peut se faire aussi via PIP.

Exemple : **pip install ansible**

Nota :

- Via pip on peut installer une version spécifique d'ansible
- Python 2.5 au minimum doit être installé sur les machines cibles
- Pour les machines utilisant SELINUX il faudra installer le paquet python-selinux

- Définition de l'inventaire

Le fichier inventaire "hosts" se trouve dans "/etc/ansible" par défaut. Il définit la liste des machines qui pourrait répondre aux requêtes d'Ansible. Les ordinateurs définis dans le fichier "hosts" peuvent être organisés en groupe. On peut même créer des groupes de groupe. Ce fichier est modifiable dans la configuration d'ansible ou même spécifiable à l'exécution des commandes ansible.

Exemple :

[Ordonnateur]

10.2.2.5

[file_slow]

[File_fast]

10.2.2.10

Ansible – Mise en oeuvre

- Les variables

On peut définir des variables arbitraires qui peuvent être associées à une machine ou un groupe de machines. Elles permettent de modifier le comportement d'Ansible (info d'authentification). Les variables se définissent soit sur la ligne correspondante à la machine soit dans une section dédiée.

- Il est commun d'utiliser la variable "type" qui sera utile non pas à ansible mais qui sera utilisée par la suite dans un rôle ou un playbook (ensemble de tâches à effectuer).

```
[file_slow]
name1.example.com type=master
name2.example.com type=slave
```

- Un autre exemple de variable est ansible_ssh_user. Elle est utilisé pour établir des connexions SSH avec le bon utilisateur.

Exemple :

```
[file_slow]
```

```
name1.example.com ansible_ssh_user=root
```

- Enfin il est possible de définir des variables dans les dossiers group_vars et host_vars (Format YML)

Exemple :

```
[file_slow]
```

```
name1.example.com type=master
```

```
name2.example.com type=slave
```

Contenu de : /etc/ansible/group_vars/file_slow
ansible_ssh_user : ansible

Ansible – Mise en oeuvre

Contenu de : /etc/ansible/host_vars/

nlame1.example.com

type : **maste**
r

Contenu de : /etc/ansible/host_vars/

nlame2.example.com

type : **slave**

- Les outils / les accès

Les outils :

ansible fournit plusieurs outils en ligne de commande :

Outil	Description
ansible	Execution d'une tâche
ansible-playbook	Execution de playbook (ensemble de tâches à effectuer)
ansible-doc	Accès au listing + documentation des serveurs
ansible-vault	Gestion de fichiers chiffrés (stockage variable - mot de passe)
ansible-galaxy	Accès au dépôt des rôles d'ansible

Le module **ping** permet de valider les accès d'un inventaire. Les machines cibles répondront par pong.

Exemple :

ansible -i hosts file_slow -m ping

nota :

- L'option -i spécifie le chemin vers l'inventaire
- L'option -m spécifie le module ansible à utiliser
- L'option all mis à la place de file_slow aurait impacté toutes les machines présentent dans l'inventaire

Ansible – Mise en oeuvre

Les accès :

L'accès SSH aux hôtes peut être explicité de plusieurs manières :

- En utilisant les informations de connexion dans l'inventaire
- En précisant toutes les informations sur la ligne de commande ansible
- En configurant la machine de management pour une connexion transparente

En pratique la méthode "*configurer la machine de management*" est la plus pratique à utiliser.

Pour cela il faudra :

- Créer une paire de clé SSH (ssh-keygen)
- Déployer la clé publique sur chacun des serveurs (ssh-copy-id)
- Créer une configuration sudo sans mot de passe sur les hôtes distants ou un login en tant que root. Le login utilisateur pour chaque hôte peut être spécifié dans l'inventaire.

Exemple :

\$ssh-keygen

Generating public/private rsa key pair.

Enter file in which to save the key
(/home/user/.ssh/id_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

...

\$ssh-copy-id ansible@name1.example.com

Nota: sudo peut être configuré pour ne pas demander de mot de passe lors de son utilisation

\$cat /etc/sudoers.d/ansible

ansible ALL=(ALL) NOPASSWD : ALL

Ansible – Utilisation des modules

3. Utilisation des modules

- Définition

Les modules sont la base des actions exécutées par Ansible. Chaque module est lié à une action spécifique. Les modules acceptent des arguments. La liste des modules proposé par l'application ansible est accessible grâce à la commande "`ansible-doc --list`". On peut retrouver la liste des modules à cette adresse :

http://docs.ansible.com/ansible/modules_by_category.html

La syntaxe est :

`ansible (hote/groupe/all) -m MODULE [-a "arg1=val1"]`

- Exemple :

`$ ansible all -m yum -a "name=httpd state=latest"`

Cette commande permet de vérifier si la dernière version d'apache est installée, si non, alors elle installe la dernière version.

Voici une liste de modules couramment utilisés :

- `ping` : validation de l'inventaire
- `setup` : retourne une liste d'informations matériels de l'hôte
- `shell` et `command` : permettent d'exécuter des commandes sur les hôtes
- `user` : permet de gérer des utilisateurs sur les hôtes
- `file` : permet de gérer des droits sur des fichiers
- `service` : permet de gérer les services systèmes tel que : arrêt/démarrage/redémarrage ou activation et désactivation au boot
- `yum/apt/zypper` : permet de gérer l'installation, la mise à jour et la suppression de paquets

Ansible – Playbook

4. Les Playbooks :

1. Définition :

Les playbooks permettent d'exécuter un ensemble de tâches à effectuer sur les machines hôtes de manière séquentielle sur tout ou partie de l'inventaire.

Chaque tâche utilise un module de Ansible. On utilise le langage YAML pour écrire un playbook. Un playbook est constitué au minimum :

- D'une variable *hosts* qui désigne les machines cibles
- D'une variable *task* qui définit une action à accomplir.

Les playbooks peuvent aussi :

- Emettre des notifications, qui seront utiles pour déclencher une action dans certaines conditions
- Effectuer des actions conditionnelles selon leur valeur (par exemple le type de distribution du système)
- Utiliser des templates pour créer ou modifier des fichiers

On trouve aussi dans les playbooks :

- Des rôles à utiliser
- Des handlers (tâches spécifiques se déclenchant grâce à une notification)
- Des éléments de configuration pour Ansible (par exemple l'utilisation de sudo)
- Des variables spécifiques qui ne sont pas reliées à l'inventaire.

Ansible – Playbook

Exemple :

```
• hosts: file_slow
  tasks:
    - name: Install apache
      yum:
        name: httpd
        state: present
        tags:
          -install_httpd
```

L'exécution du playbook install_apache.yml se fait grâce à la commande ansible-playbook :

```
$ansible-playbook install_apache.yml
```

Par défaut, un playbook s'exécute sur tous les hôtes concernés. Il est possible de cibler ces actions grâce aux options :

- Tags : cette option se définit dans le playbook et on l'appelle --tags nom_du_tag ou --skiptags
- L'option -l qui permet de cibler une machine en particulier

Exemple :

```
$ansible-playbook install_apache.yml --tags
install_httpd -l nname1.example.com
```

La syntaxe YAML (Yet Another Markup Language)

C'est un langage facile à lire et à écrire. On peut y inclure une syntaxe JSON. On peut y manipuler du texte, des listes et des dictionnaires. L'indentation est très strict (2 espaces / indentation). On démarre un playbook toujours avec --- puis on va à la ligne. Enfin les commentaires sont définis grâce au #

Ansible – Playbook

Exemple :

```
---
```

```
# Liste des commutateurs juniper présent à l'IBMP
```

```
commutateurs:
```

```
  - ex3300
```

```
  - ex4200
```

```
  - ex2200
```

```
  - ex4300
```

```
---
```

Les tâches :

Chaque tâche est définie par un nom (rubrique *name*) pour commenter l'action et d'un module à appeler.
Des attributs supplémentaires permettent de rendre les playbooks plus interactifs :

- **notify** : défini un Handler à appeler
- **register** : sauve le résultat d'une action dans une variable
- **include** : inclusion d'un fichier externe
- **when** : exécute une action sous certaines conditions
- ...

Notification et handlers :

Les handlers permettent de définir des actions qui ne seront exécutées qu'au déclenchement d'une notification. Les handlers ne sont pris en compte qu'après la fin de toutes les tâches présentes dans le playbook.

Ansible – Playbook

Exemple :

- hosts: file_slow
become: no tasks:
 - name: Action qui fait une modification file:
path: /tmp/toto state: touch mode: 0644 notify:
-handler1

Attention l'argument de *notify* et le nom du *handler* doivent être rigoureusement identique.

5. Structures de contrôle

- Définition :

Les structures de contrôle sont probablement la partie la plus utile. En effet grâce aux structures de contrôle on peut manipuler les données provenant des hôtes de façon très flexible et puissante.

- Facts / Conditions / boucles / inclusions :

- Les facts : La première action effectuée lors de l'exécution d'un playbook est de récolter des informations de chacun des hôtes distants. Ces *facts* alimentent des variables (préfixées par Ansible). Ces variables seront utilisables dans les filtres des tests et des boucles du playbook. Pour avoir une liste de tous les *facts* qu'Ansible met à votre disposition il faut utiliser le module setup

Ansible – Playbook

- Les conditions : Il est utile que certaines tâches du playbook soient effectuables seulement si une condition spécifique est remplie ou non. La directive `when` permet d'effectuer ce test

Exemple :

```
- name: Installation apache (Debian)
  apt:
    name: apache2
    path: /tmp/toto
    state: present
  when: ansible_os_family == 'Debian'
- name: Installation apache (REHL)
```

```
yum :
  name: httpd
  state: present
  when: ansible_os_family == 'RedHat'
```

- Les boucles : Les boucles permettent d'itérer sur les variables de type liste et dictionnaire. Cette technique permet d'exécuter plusieurs fois la même action sur un nombre d'éléments indéfinis lors de l'écriture du rôle ou playbook. La liste complète des boucles est disponible dans la [documentation ansible](#).

Exemple :

`with_items` permet de boucler sur une liste. A chaque itération une variable `item` prend la valeur suivante de la liste :

```
- name: Installation
  d'éditeurs
  yum:
    name: "{{ item }}"
  with_items:
    - vim
    - emacs
    - nano
```

Ansible – Playbook

- Les inclusions: La directive *include* permet d'importer une liste de tâches ou de handlers. Cette technique permet de rendre génériques certaines actions.

Exemple :

```
hosts: all
tasks:
- include: common-setup.yml
    - name : something else
    ...
```

Ansible – Templates / Rôles

6. Templates / Rôles

1. Définition des Templates :

Ansible fonctionne sur un système de modèles (templates). Ils s'appuient sur Jinja2 et permettent de gérer des boucles, des tests logiques, des listes ou des variables.

- Une variable se déclare en la mettant au milieu de deux accolades : variable : {{variable}}
- Les instructions sont délimitées par accolades et pourcentage :
 {% instruction %}
- Les commentaires sont délimités par accolade et dièse

Ansible – Templates / Rôles

Exemple :

```
{# Définition de notre liste #}
{% set myList = [ 1 , 2 , 3 ] %}
{# Parcours de la liste #}
{# Pour afficher l'ensemble des valeurs présentes #}
{% for value in myList %}
    Valeur : {{ value }}
{% endfor %}
```

2. Définition des rôles :

Les rôles sont des playbooks génériques, qui peuvent être intégrés dans d'autres playbooks. Cette notion est essentielle afin de créer des tâches complexes. En effet afin de rendre des playbooks lisibles il vaut mieux construire des rôles que l'on peut combiner ensemble plutôt que créer un rôle complexe qui sera difficile à maintenir et complexe à utiliser et peu lisible.

Pour chaque élément d'un rôle (tâche, handlers, ...) un fichier nommé main.yml sert de point d'entrée.

Exemple d'arborescence :

Dans le répertoire playbooks, où se trouvent ansible.cfg, hosts et deploy.yml, on va créer un dossier « roles » et on lancera la commande :

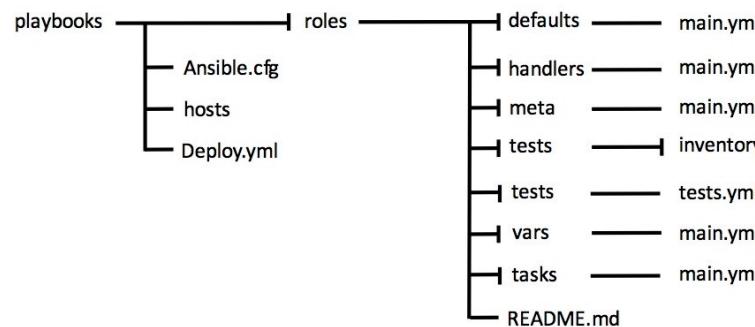
```
$ ansible-galaxy init -p roles/ test
```

L'argument **init** permet d'initier la création d'un rôle et il est suivi de l'option **-p** pour indiquer le chemin où l'on désire créer le rôle. Si l'option **p** n'est pas définie le rôle sera créé dans le répertoire courant. Ici on crée un rôle test.

- test was created successfully

Ansible – Templates / Rôles

Notre rôle est bien créé et il contient les répertoires suivants :



Grâce à cette arborescence il ne sera pas nécessaire de nommer les directives `vars`, `tasks` ou `handlers` dans les fichiers `vars/main.yml`, `tasks/main.yml` et `handlers.yml`. Cette information sera obtenue à partir du nom du répertoire. De la même façon les modules `copy` et `template` iront chercher les fichiers sources directement dans les répertoires `files` et `templates`, sans avoir besoin de préciser le path.

Obtenir des rôles :

Ansible fournit une bibliothèque de rôles en ligne : [Galaxy](#).

Les rôles disponibles sur Galaxy sont soumis par la communauté Ansible, et leur qualité varie. Un système de notation permet de trouver les meilleurs rôles.

Exemple : Installation

`$ ansible-galaxy install geerlingguy.apache`

Suppression

`$ ansible-galaxy remove geerlingguy.apache`