



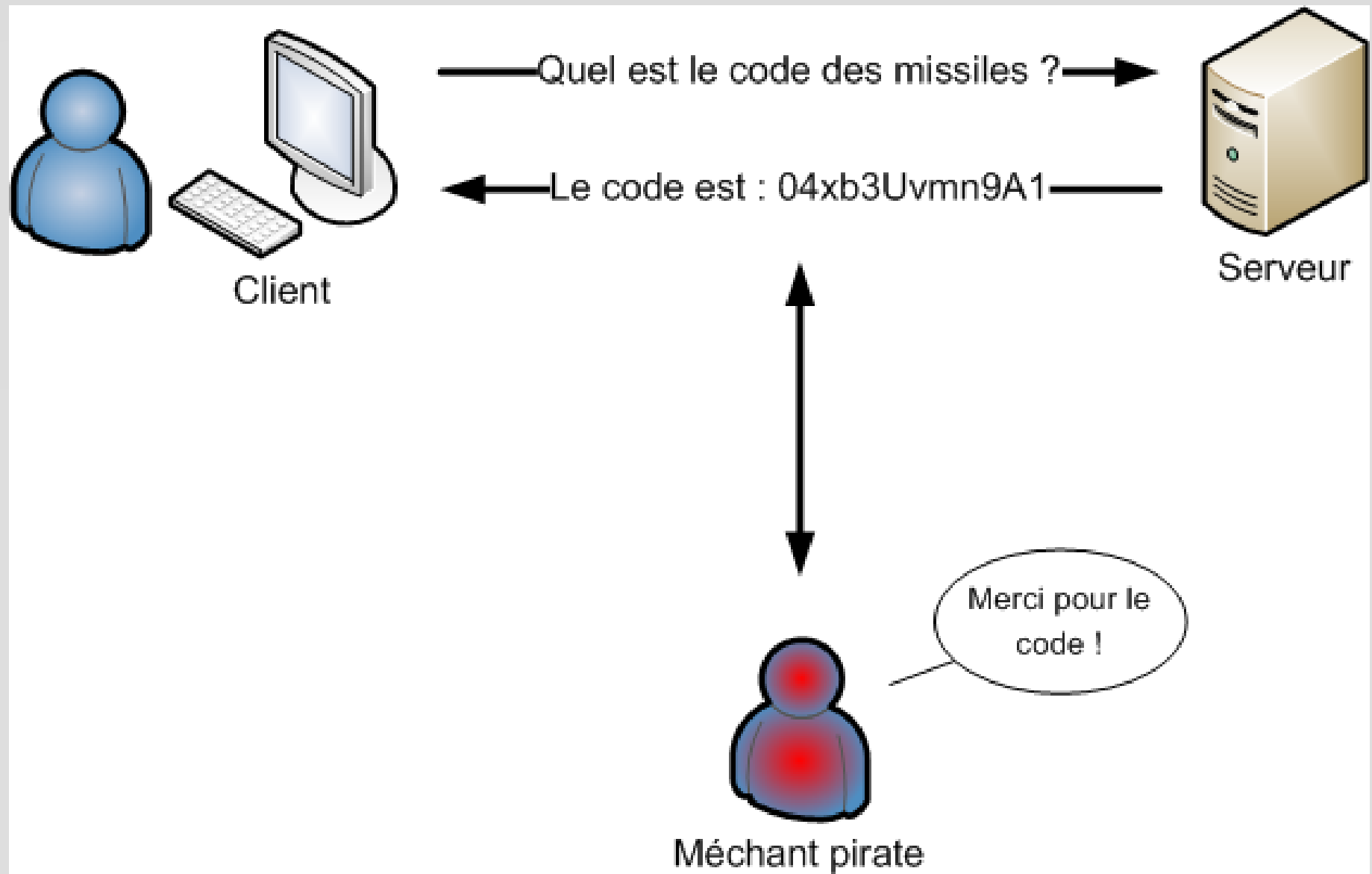
DEVOPS

PLAN DU COURS

1. SSH/Virtualisation
2. Présentation du métier d'ops
3. Intégration d'une application
4. Les outils, du build à la mise en production
5. Containerisation (Docker/Kubernetes)

SSH

LA ATTAQUES "MAN IN THE MIDDLE"



LE SYSTÈME DE CLÉS DE SSH

SSH signifie Secure SHell.

C'est un protocole qui permet de faire des connexions sécurisées (i.e. cryptées) entre un serveur et un client SSH.

1. La théorie de la cryptographie asymétrique :

SSH utilise la cryptographie asymétrique RSA ou DSA.

En cryptographie asymétrique, chaque personne dispose d'un couple de clé : une clé publique et une clé privée. La clé publique peut être librement publiée tandis que la clé privée doit rester secrète. La connaissance de la clé publique ne permet pas d'en déduire la clé privée.

Si la personne A veut envoyer un message confidentiel à la personne B, A crypte le message avec la clé publique de B et l'envoie à B sur un canal qui n'est pas forcément sécurisé. Seul B pourra décrypter le message en utilisant sa clé privée.

2. La théorie de la cryptographie symétrique :

SSH utilise également la cryptographie symétrique.

Son principe est simple : si A veut envoyer un message confidentiel à B, A et B doivent d'abord posséder une même clé secrète. A crypte le message avec la clé secrète et l'envoie à B sur un canal qui n'est pas forcément sécurisé. B décrypte le message grâce à la clé secrète. Toute autre personne en possession de la clé secrète peut décrypter le message.

La cryptographie symétrique est beaucoup moins gourmande en ressources processeur que la cryptographie asymétrique... mais le gros problème est l'échange de la clé secrète entre A et B. Dans le protocole SSL, qui est utilisé par SSH et par les navigateurs Web, la cryptographie asymétrique est utilisée au début de la communication pour que A et B puissent s'échanger une clé secrète de manière sécurisée... puis la suite la communication est sécurisée grâce à la cryptographie symétrique en utilisant la clé secrète échangée.

3. L'établissement d'une connexion SSH :

Un serveur SSH dispose d'un couple de clés RSA stocké dans le répertoire `/etc/ssh/` et généré lors de l'installation du serveur. Le fichier `ssh_host_rsa_key` contient la clé privée et a les permissions 600. Le fichier `ssh_host_rsa_key.pub` contient la clé publique et a les permissions 644.

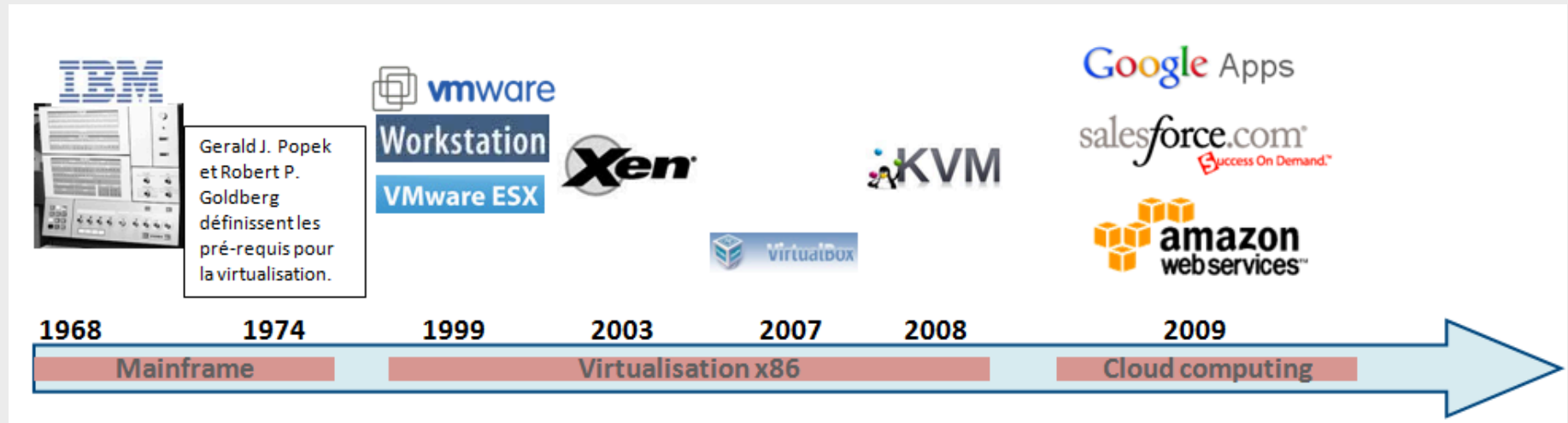
Nous allons suivre par étapes l'établissement d'une connexion SSH :

1. Le serveur envoie sa clé publique au client.
2. Le client génère une clé secrète et l'envoie au serveur, en cryptant l'échange avec la clé publique du serveur (cryptographie asymétrique). Le serveur décrypte la clé secrète en utilisant sa clé privée, ce qui prouve qu'il est bien le vrai serveur.
3. Pour le prouver au client, il crypte un message standard avec la clé secrète et l'envoie au client. Si le client retrouve le message standard en utilisant la clé secrète, il a la preuve que le serveur est bien le vrai serveur.
4. Une fois la clé secrète échangée, le client et le serveur peuvent alors établir un canal sécurisé grâce à la clé secrète commune (cryptographie symétrique).
5. Une fois que le canal sécurisé est en place, le client va pouvoir envoyer au serveur le login et le mot de passe de l'utilisateur pour vérification. Le canal sécurisé reste en place jusqu'à ce que l'utilisateur se déconnecte.

La seule contrainte est de s'assurer que la clé publique présentée par le serveur est bien sa clé publique... sinon le client risque de se connecter à un faux serveur qui aurait pris l'adresse IP du vrai serveur (ou toute autre magouille).

Une bonne méthode est par exemple de demander à l'administrateur du serveur quelle est le fingerprint de la clé publique du serveur avant de s'y connecter pour la première fois.

LA VIRTUALISATION



Intérêts :

- Utilisation optimale des ressources d'un parc de machines (répartition des machines virtuelles sur les machines physiques en fonction des charges respectives).
- Installation, déploiement et migration facile des machines virtuelles d'une machine physique à une autre, notamment dans le contexte d'une mise en production à partir d'un environnement de qualification ou de pré-production, livraison facilitée.
- Économie sur le matériel par mutualisation (consommation électrique, entretien physique, surveillance, support, compatibilité matérielle, etc.).
- Installation, tests, développements, cassage et possibilité de recommencer sans casser le système d'exploitation hôte.
- Sécurisation et/ou isolation d'un réseau (cassage des systèmes d'exploitation virtuels, mais pas des systèmes d'exploitation hôtes qui sont invisibles pour l'attaquant, tests d'architectures applicatives et réseau).
- Isolation des différents utilisateurs simultanés d'une même machine.
- Allocation dynamique de la puissance de calcul en fonction des besoins de chaque application à un instant donné.
- Diminution des risques liés au dimensionnement des serveurs lors de la définition de l'architecture d'une application, l'ajout de puissance (nouveau serveur, etc.) étant alors transparent.

Inconvénients :

- L'accès aux ressources des serveurs hôtes via la couche d'abstraction matérielle nuit aux performances, et l'exécution de n'importe quel logiciel virtualisé consommera davantage de ressources qu'en mode natif.
- En cas de panne d'un serveur hôte, l'ensemble des machines virtuelles hébergées sur celui-ci seront impactées. Mais la virtualisation est souvent mise en œuvre avec des redondances, qu'elle facilite.
- La mise en œuvre est complexe et demande un investissement initial.
- Il y a des contraintes d'administration spécifiques (déploiement, sauvegarde...).

TP 1

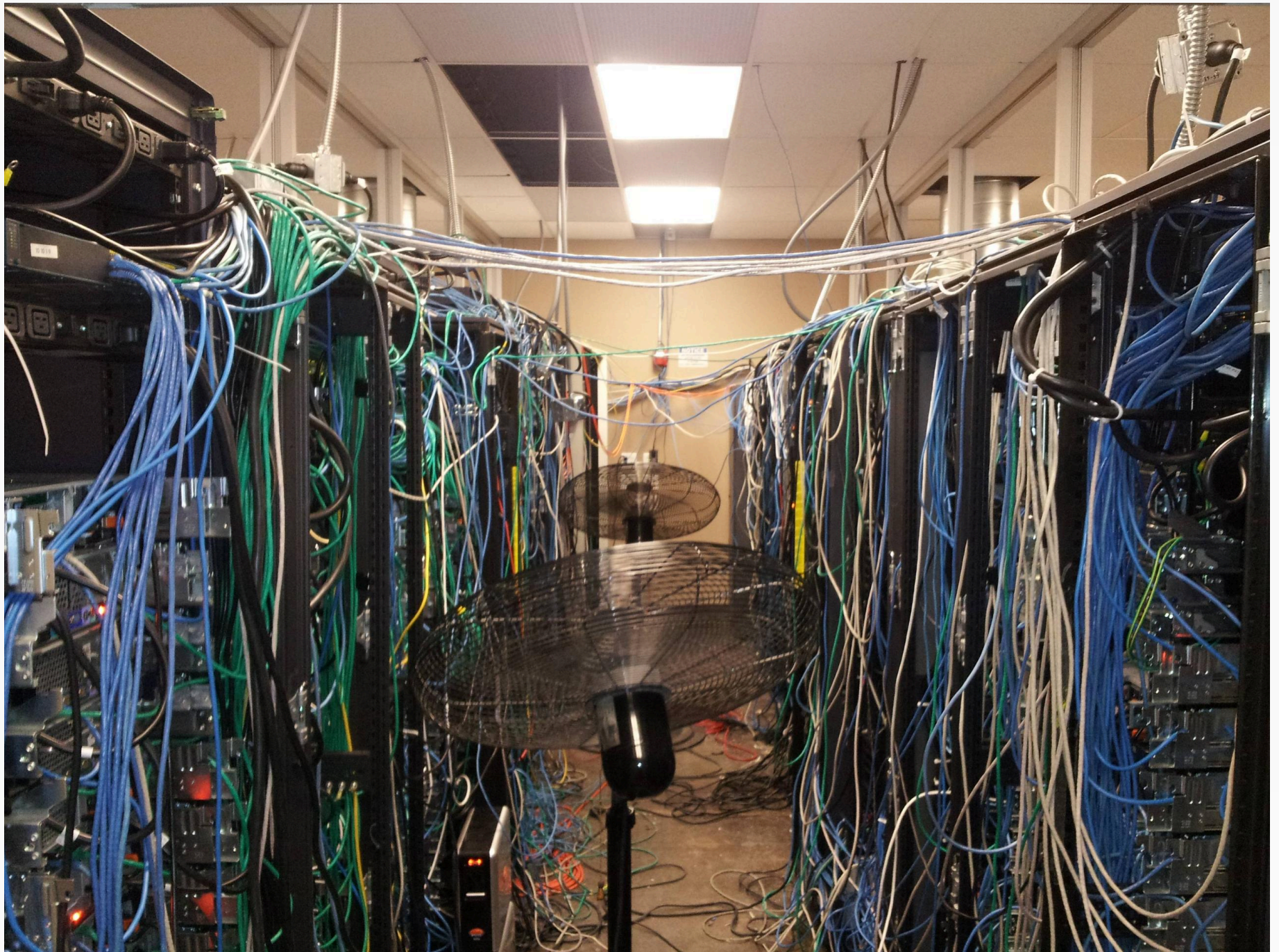
PRÉSENTATION DU MÉTIER D'OPS

- Mettre à disposition des autres équipes IT de l'outillage
- Permettre le packaging des applications
- Garant du déploiement des applications en prod (parfois les autres environnements)
- Maintenance de l'infrastructure
- Gestion du réseau

A L'ANCIENNE

L'outillage et les process du siècle dernier :

- Build des livrables (Jar, War, Exe, ...) à la main
- Déploiement des livrables à la main (SCP était ton ami)
- Process et communication à base de PTI (Procédure Technique d'Intervention, souvent des documents word de plusieurs dizaines de pages)
- Gestion des serveurs sous forme de serveur physique ou au mieux de VM linux (administration à la main à base de scripts shell)
- Le réseau était géré via des équipements physiques, la maintenance était aussi souvent... physique



DE L'OPS VERS LE DEVOPS

Constat :

Les projets informatiques se sont agilisés. Cela a induit des cycle de développement et de livraison de plus en plus courts.

Les process et outils traditionnels des Ops étaient de moins en moins en phase avec les exigences de l'IT.

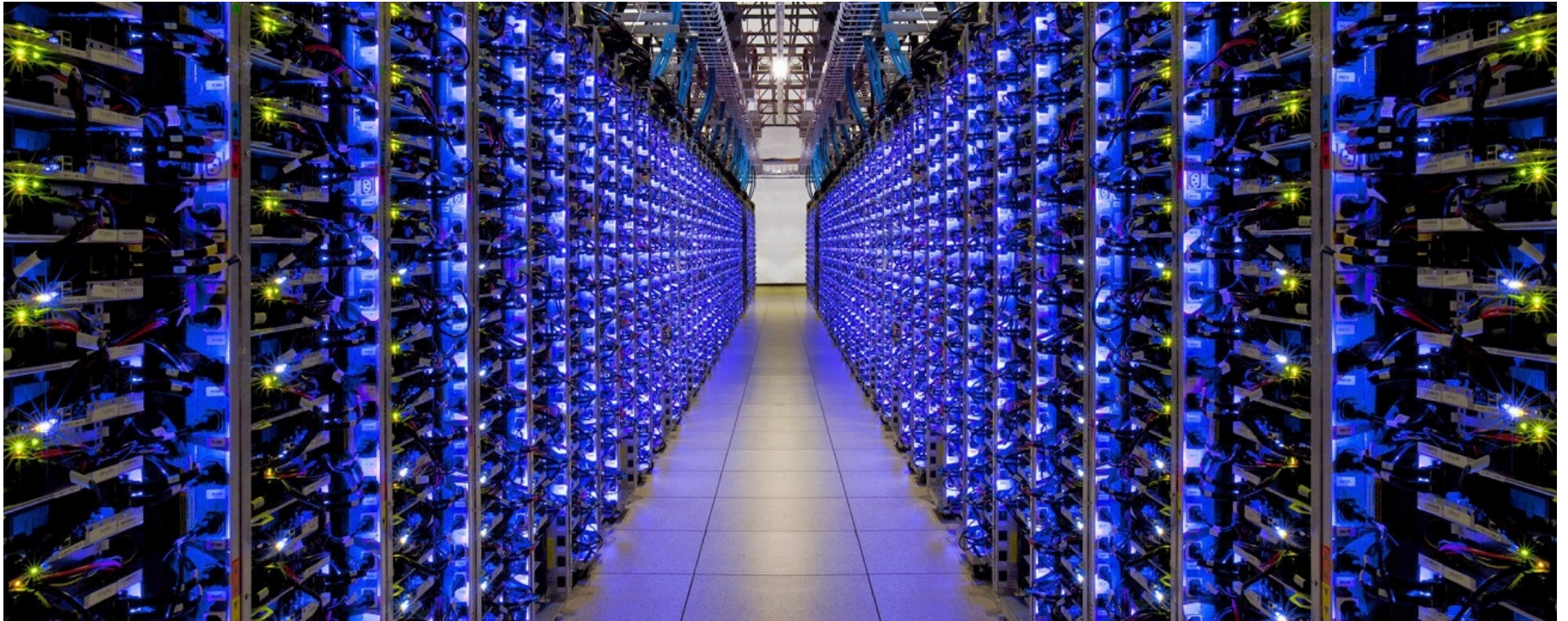
Au début des années 2010 il y a eu un vrai sursaut dans la communauté Dev et par extention dans celle des Ops sur la nécessité de développer des outils d'automatisation.

Depuis une dizaine d'année le métier d'Ops a donc subit une profonde mutation pour passer du "techos linuxien bidouilleur" à celui de développeur et mainteneur d'outillage automatisés.

Les outils d'aujourd'hui :

- Builds automatisés (Jenkins, Gitlab, ...)
- Déploiements automatisés (Repository, Ansible, ...)
- Infrastructure Cloudisée et/ou containerisée (CF Docker)

Une fiche métier DevOps : <https://unlck.fr/fiches-metiers/devops/>



INTÉGRATION D'UNE APPLICATION

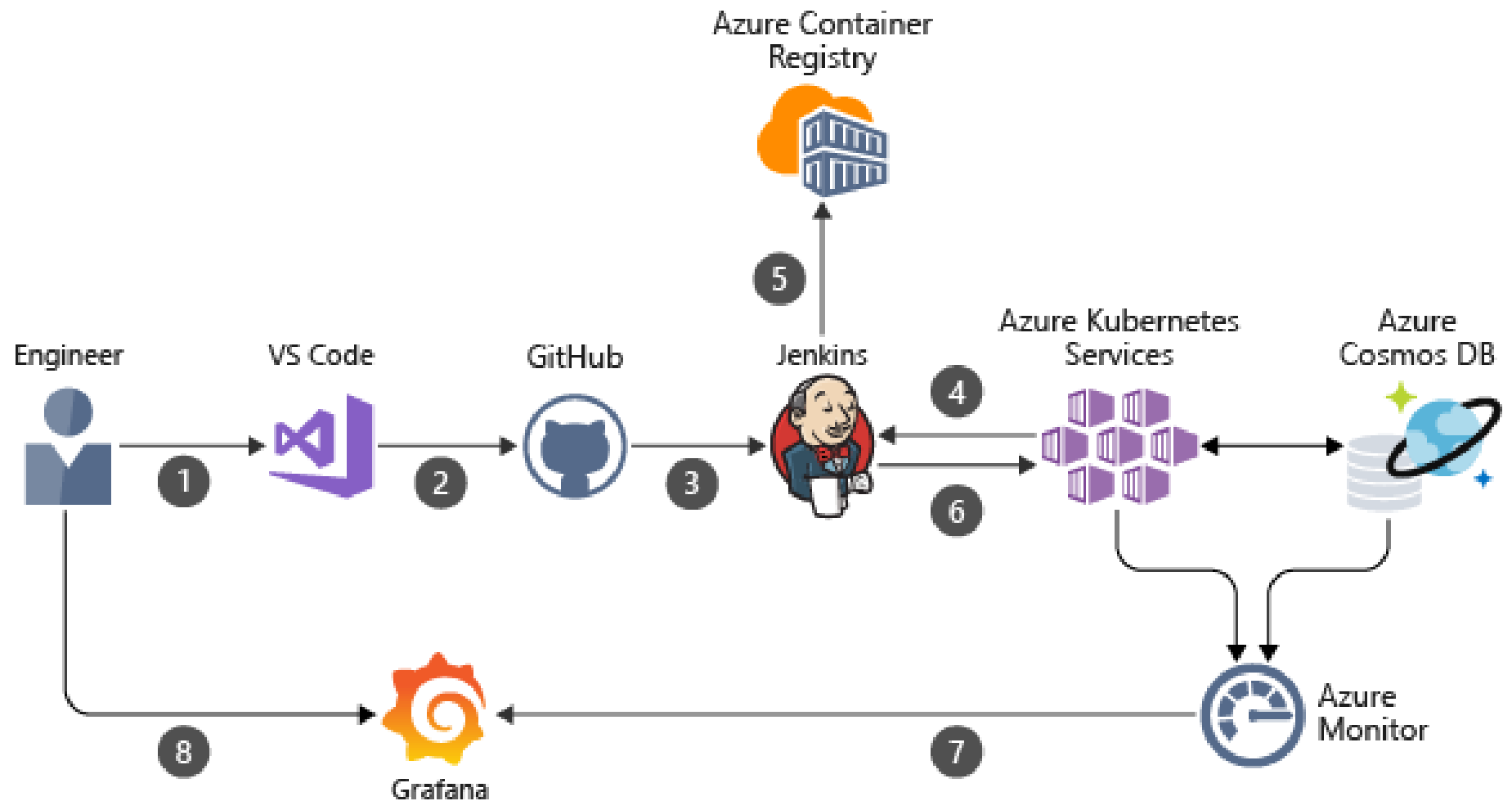
GESTION DE VERSION

Les différents outils de gestion de version :

- Rsync -_-"
- Git
- Apache Subversion (SVN)
- Mercurial
- IBM Rational VC
- ...

Des questions ?

CI, CD AND ... CD



Continuous Integration :

Les développeurs pratiquant l'intégration continue fusionnent leurs modifications dans la branche principale aussi souvent que possible. Les modifications du développeur sont validées en créant un build et en exécutant des tests automatisés sur le build.

Cela permet d'éviter les problèmes d'intégration qui peuvent survenir lorsque vous attendez le jour de la publication pour fusionner les modifications dans la branche de publication.

L'intégration continue met l'accent sur l'automatisation des tests pour vérifier que l'application n'est pas interrompue chaque fois que de nouveaux commits sont intégrés dans la branche principale.

Continuous delivery :

La livraison continue est une extension de l'intégration continue car elle déploie de manière automatisée toutes les modifications de code dans un environnement de test et/ou de production après l'étape de construction.+

Cela signifie qu'en plus des tests automatisés, vous disposez d'un processus de publication automatisé et que vous pouvez déployer votre application à tout moment en cliquant sur un bouton.

En théorie, avec la livraison continue, vous pouvez décider de publier quotidiennement, hebdomadairement, bimensuellement ou selon les besoins de votre entreprise.

- ❗ Si vous voulez vraiment profiter des avantages de la livraison continue, vous devez déployer en production le plus tôt possible pour vous assurer de livrer de petits lots faciles à dépanner en cas de problème et vous éviter des problèmes de code obsolète.

Continuous deployment :

Le déploiement continu va encore plus loin que la livraison continue. Avec cette pratique, chaque changement qui passe toutes les étapes de votre pipeline arrivent en production. Il n'y a pas d'intervention humaine, et seul un test raté empêchera le déploiement d'une nouvelle modification en production.

Le déploiement continu est un excellent moyen d'accélérer la boucle de rétroaction avec vos clients et de soulager l'équipe car il n'y a plus de Release Day . Les développeurs peuvent se concentrer sur la création de logiciels et voient leur travail être mis en ligne quelques minutes après avoir fini de travailler dessus.

Des questions ?

LOGS

Des logs pour quoi faire ?

- Debugguer
- Collecter de l'information métier
- La sécurité

Et un système de centralisation des logs pour quoi faire ?

- Simplifier la gestion des logs
- Un monitoring / reporting plus complet
- Une sécurité renforcée

Principes

Un processus est chargé de collecter les logs et de les envoyer vers un système de centralisation des logs.

Le processus peut être intégré à votre programme (Log4j ou Logback par exemple en java) ou être séparé (agent logstash).

Dans tous les cas les logs doivent être formatés de manière à être compréhensible par le système de centralisation des logs dont le rôle est d'aggréger les logs et permettre leur lecture efficacement par la suite.

Quelques produits :

- Graylog
- Rsyslog
- ELK
- Datadog

MONITORING / ALERTING

QU'EST-CE QUE LE MONITORING (OU SUPERVISION) ?

Il s'agit de détecter et de comprendre les problèmes qui peuvent survenir lors de l'utilisation d'une application.

OBSERVABILITÉ

L'observabilité est le pilier d'une démarche de monitoring.
Il s'agit de pouvoir répondre en temps réel à la question
“que se passe-t-il ?”.

Dans un contexte DevOps, il faut envisager deux niveaux
d'observabilité : bas niveau et applicatif.

Observabilité bas niveau : ce qu'il se passe dans mon système

- Consommation de mémoire
- Consommation de réseau
- État de mes clusters Kubernetes etc...

Observabilité applicative : ce qu'il se passe dans mon service

- Bugs
- Performances.

Dans une démarche DevOps, l'intérêt est de lier les deux car les corrélations sont très fortes.

L'ALERTING

On sait donc analyser ce qu'il se passe dans le système grâce au monitoring. Encore faut-il être au courant quand un incident survient. C'est la fonction de l'alerting, qui permet d'intervenir le plus vite possible, afin de limiter l'impact de l'incident en question.

KPI

Une autre fonction du monitoring, apparue plus récemment, est de permettre de suivre l'évolution des indicateurs business.

Ainsi, lorsque les outils de monitoring sont en place, il est possible de suivre l'impact des mises à jour de l'applicatif auprès des utilisateurs.

MONITORING ET DEVOPS

Comme nous l'avons vu, le monitoring est un excellent outil de communication entre développeurs et Ops. Il aide à aligner leurs enjeux, découvrir les corrélations entre les problèmes qu'ils rencontrent, et à avoir des discussions factuelles.

Le monitoring est aussi intimement lié au DevOps car il est indispensable pour la mise en place d'une feedback loop efficace.

DEMO

Kibana démo :

<https://demo.elastic.co/app/uptime>

Graphana démo :

[https://play.grafana.org/d/0000000012/grafana-play-home?
orgId=1](https://play.grafana.org/d/0000000012/grafana-play-home?orgId=1)

Datadog démo :

<https://app.datadoghq.eu/logs>

TP 2

MISE EN PRODUCTION ET INDUSTRIALISATION

- Sonar
- Repository Manager
- Jenkins / GitLab / Ansible

SONAR

Sonar est un logiciel permettant d'assurer la qualité du code. Il est articulé autour des features principales suivantes :

- Detection d'anomalies de code (code smell)
- Detection de duplication de code
- Couverture de tests

DÉMO

https://sonarcloud.io/dashboard?id=apache_struts

TP 3

REPOSITORY MANAGER : NEXUS/ARTIFACTORY

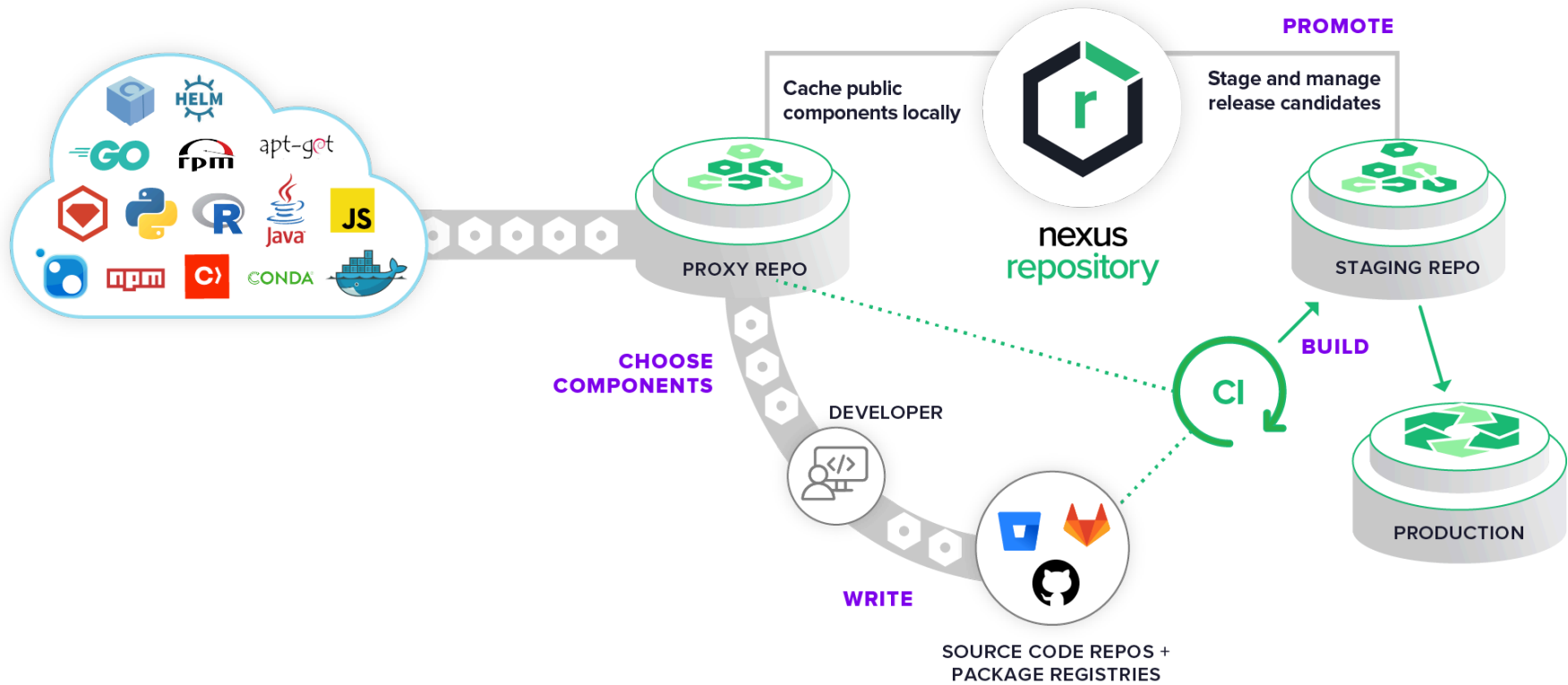
Un Repository Manager (ou gestionnaire de dépôts d'objets binaires) est un outil logiciel conçu pour optimiser le téléchargement et le stockage des objets binaires utilisés et produits dans le cadre du développement de logiciels.

Il centralise la gestion de tous les artefacts générés et utilisés par les logiciels de l'organisation.

Il permet de répondre à la problématique de complexité découlant de la diversité des types d'objets binaires, de leur position dans l'ensemble du flux de travail ainsi que les dépendances entre eux.

Un dépôt d'objets binaires est un dépôt de logiciels pour les paquets, artefacts et leurs métadonnées.

Il peut être utilisé pour stocker les fichiers binaires produits par l'organisation elle-même (releases et builds intermédiaires des produits) ou pour stocker les binaires tiers (dépendances).



Exemples de repo :

- npm : <https://registry.npmjs.org/>
- yarn : <https://registry.yarnpkg.com/>
- maven central : <https://mvnrepository.com/repos/central>
- yum :
<https://yum.oracle.com/repo/OracleLinux/OL8/developer/EF>

Utilisation dans Maven :

settings.xml

```
<settings>

  <proxies>
    <proxy>
      <active>true</active>
      <protocol>https</protocol>
      <host>10.42.42.42</host>
      <port>8080</port>
      <username>login</username>
      <password><![CDATA[MotDePass]]></password>
    </proxy>
  </proxies>



  <servers>
    <server>
      <id>carrefour</id>
      <username>deve_loper</username>
      <password>unautremotdepasse</password>
    </server>
    <server>
      <id>maven-central</id>
      <username>deve_loper</username>
      <password>encoreunmotdepasse</password>
    </server>
  </servers>

  <mirrors>
    <mirror>
      <id>maven-central</id>
```

JENKINS

Jenkins est un serveur Web, multiplateforme, développé en JAVA. Jenkins est construit en suivant un modèle maître/esclave.

Ainsi, vous configurez vos projets à travers le serveur Web sur la machine maître et les esclaves (qui sont des instances légères de Jenkins) vont effectuer les actions configurées.

-  La machine maître peut aussi exécuter des tâches. Plus précisément, chaque machine (maître ou esclave) possède un ou plusieurs exécuteurs et chaque exécuteur peut faire une tâche à la fois.
-  Il est conseillé de ne pas exécuter de tâches sur l'instance maître. En effet :
 - une tâche sur l'instance maître peut accéder aux fichiers de configuration de Jenkins (notamment les modifier, quant bien même le créateur de la tâche n'a habituellement pas de tels droits)+
 - les tâches exécutées sur l'instance maître ralentiront l'exécution de Jenkins
 - les outils installés sur la machine maître pour exécuter les tâches augmentent inévitablement la surface d'attaque.

DEMO : <http://localhost:9007/>

TP 4

GITLAB

GitLab est un système de contrôle de version.

Mais il offre également les fonctionnalités suivantes :

ANSIBLE

Jusqu'à maintenant (enfin il y a 10 ans) on déployait une application sur un seul serveur.

Comment gérer la complexité des architectures distribuées ?

Ansible est un logiciel de gestion de configuration.

Il s'appuie sur 3 principales fonctionnalités :

- La connection à distance (SSH pour les serveur linux, Winrm pour Windows) à des machines listées dans un "inventory"
- Le templating de configuration
- L'exécution de taches (locale ou distante) définies dans un "playbook"

Les avantages :

- Simple à déployer (ne nécessite qu'un serveur pour déployer Ansible, les serveurs administrés ne nécessitent pas de manipulation préalable)
- Automatisation des tâches
- Reproductibilité des environnements
- Les modules Ansible sont idempotents (il décrit l'état attendu et non la manière d'y parvenir)
- Un produit open source avec une communauté active et des dizaines de modules (Ansible Galaxy)

Les inconvénients :

- Codé en python, necessite donc une connaissance de l'environnement python (PIP)
- Les fichiers de descriptions (playbook, inventory, template, ...) sont en YAML
- Le fait d'avoir des modules idempotents implique l'execution de code parfois couteux en temps et inutile (ne produisant pas de modifications)
- La hiérarchie du projet Ansible et la résolution de variable est parfois abscons

EXAMPLE CONFIG ANSIBLE :

https://docs.ansible.com/ansible/latest/user_guide/sample_se

LES SERVICES WINDOWS/LINUX

Un service ou daemon est un programme qui tourne en arrière plan et s'active sous certaines conditions. Par exemple, le service hddtemp surveille la température de vos disques durs et déclenche une alerte si elle dépasse un certain seuil.

TP

service configurer windows : <https://nssm.cc/> service
manager linux : <https://doc.ubuntu-fr.org/systemd>

CONTAINERISATION

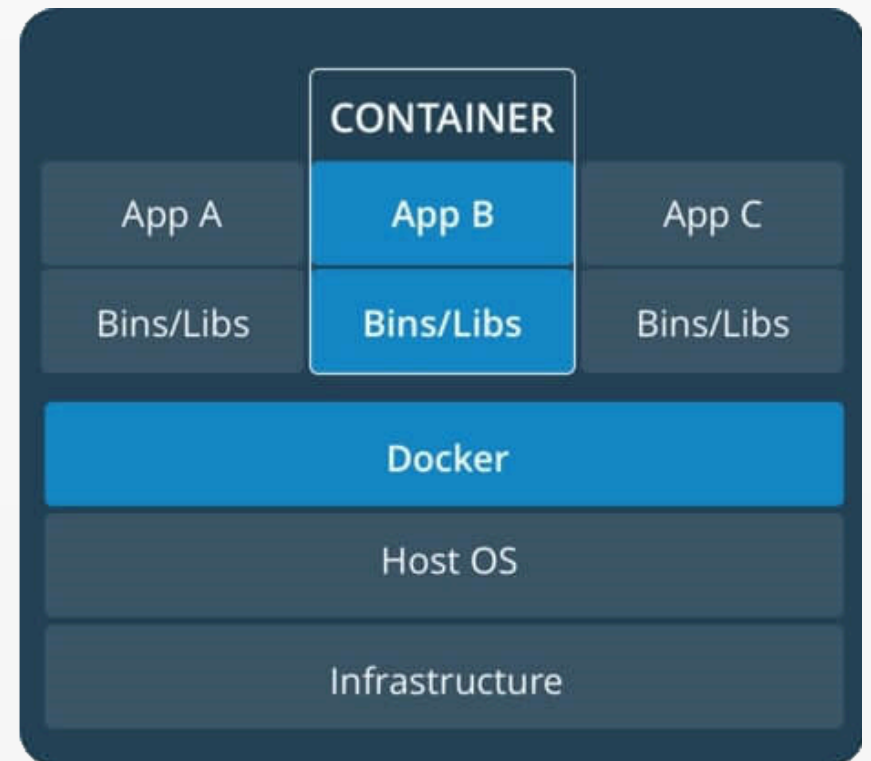
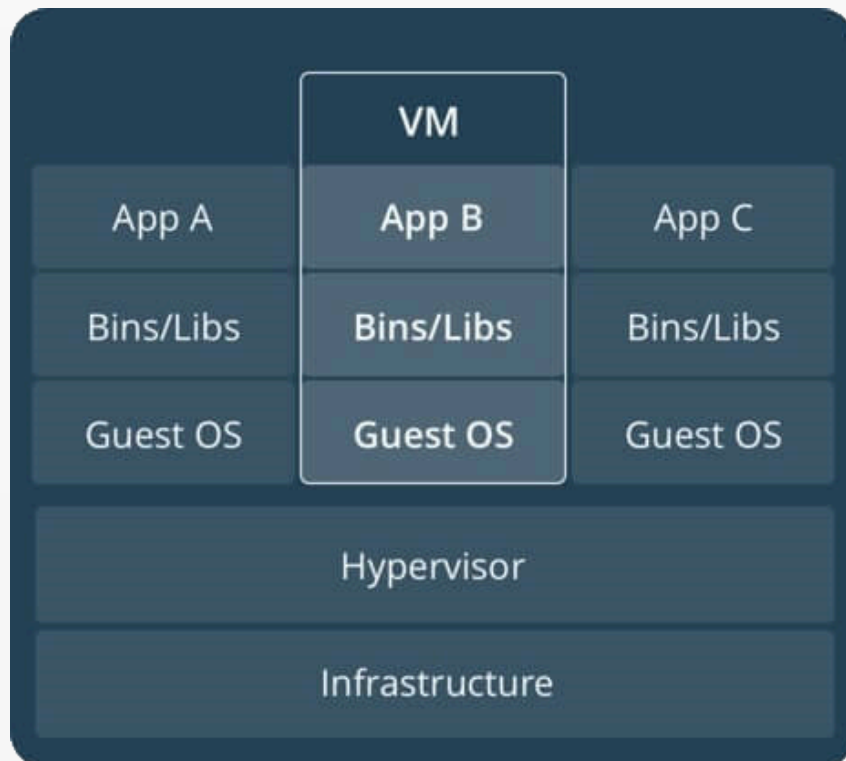
DOCKER : LA CONTENEURISATION VS VIRTUALISATION

L'isolation

Dans le cas de la virtualisation l'isolation des VMs se fait au niveau matérielles (CPU/RAM/Disque) avec un accès virtuel aux ressources de l'hôte via un hyperviseur. De plus, généralement les ordinateurs virtuels fournissent un environnement avec plus de ressources que la plupart des applications n'en ont besoin.

Par contre dans le cas de la conteneurisation, l'isolation se fait au niveau du système d'exploitation. Un conteneur va s'exécuter sous Linux de manière native et va partager le noyau de la machine hôte avec d'autres conteneurs. ne prenant pas plus de mémoire que tout autre exécutable, ce qui le rend léger.+

L'image ci-dessous illustre cette phase d'abstraction de l'OS.



Avantages de la conteneurisation par rapport à la virtualisation traditionnelle

Les machines virtuelles intègrent elles-mêmes un OS pouvant aller jusqu'à des Giga-octets. Ce n'est pas le cas du conteneur.

Le conteneur appelle directement l'OS pour réaliser ses appels système et exécuter ses applications. Il est beaucoup moins gourmand en ressources.

Le déploiement est un des points clés à prendre en compte de nos jours.

On peut déplacer les conteneurs d'un environnement à l'autre très rapidement.

On peut bien sûr faire la même chose pour une machine virtuelle en la déplaçant entièrement de serveurs en serveurs mais n'oubliez pas qu'il existe cette couche d'OS qui rendra le déploiement beaucoup plus lent, sans oublier le processus d'émulation de vos ressources physiques, qui lui-même demandera un certain temps d'exécution et donc de la latence en plus.

Les avantages de conteneurisation

La conteneurisation est de plus en plus populaire car les conteneurs sont :

- Flexible: même les applications les plus complexes peuvent être conteneurisées.
- Léger: les conteneurs exploitent et partagent le noyau hôte.
- Interchangeable: vous pouvez déployer des mises à jour à la volée
- Portable: vous pouvez créer localement, déployer sur le cloud et exécuter n'importe où votre application.
- Évolutif: vous pouvez augmenter et distribuer automatiquement les réplicas (les clones) de conteneur.
- Empilable: Vous pouvez empiler des services verticalement et à la volée.

DOCKER : COMMENT ÇA MARCHE

Le noyau Linux offre quelques fonctionnalités comme les namespaces (ce qu'un processus peut voir) et les cgroups (ce qu'un processus peut utiliser en terme de ressources), qui permettent d'isoler les processus Linux les uns des autres.

Lorsque vous utilisez ces fonctionnalités, on appelle cela des conteneurs.

Docker tire parti de plusieurs ces fonctionnalités proposées par le noyau Linux pour fournir ses fonctionnalités.

 à noter que l'on peut faire de la containerisation nativement

LES NAMESPACES

Les **namespaces** sont une fonctionnalité du noyau Linux qui partitionne les ressources du noyau de telle sorte qu'un ensemble de processus voit un ensemble de ressources tandis qu'un autre ensemble de processus voit un ensemble différent de ressources.

Concrètement les namespaces permettent de partitionner ce que les processus voient de l'OS.

Il existe différents types de namespaces sous linux :

- Mount (mnt) : Controle les points de montage (disque dur, lecteur externes, ...)
- Process ID (pid) : Controle le nommage (la numérotation) des process
- Network (net) : Permet de virtualiser localement la gestion du réseaux (création d'interface réseau virtuelle)
- Interprocess Communication (ipc) : Permet de controller la communication inter-processus
- UTS (Unix Time Sharing) : Permet de définir des hostname différents par UTS namespace
- User ID (user) : Permet une isolation des privilèges (il devient possible de définir un user root dans un namespace qui n'a aucun droit sur les users des autres namespaces)

LES CGROUPS

Le but des cgroups est d'exécuter sur un ensemble de process un contrôle via certaines règles et paramètres.

Une des fonctionnalités offerte par les cgroups est la possibilité d'organiser ces groupes de process selon une hiérarchie.

Ainsi, un cgroup controller permet d'effectuer un contrôle sur un type de ressource, par exemple le cgroup controller memory permet de limiter l'utilisation totale de RAM ou de swap pour les process dans ce cgroup.

Un process peut être dans plusieurs cgroups, pour être sous le joug de plusieurs limites de différents types.

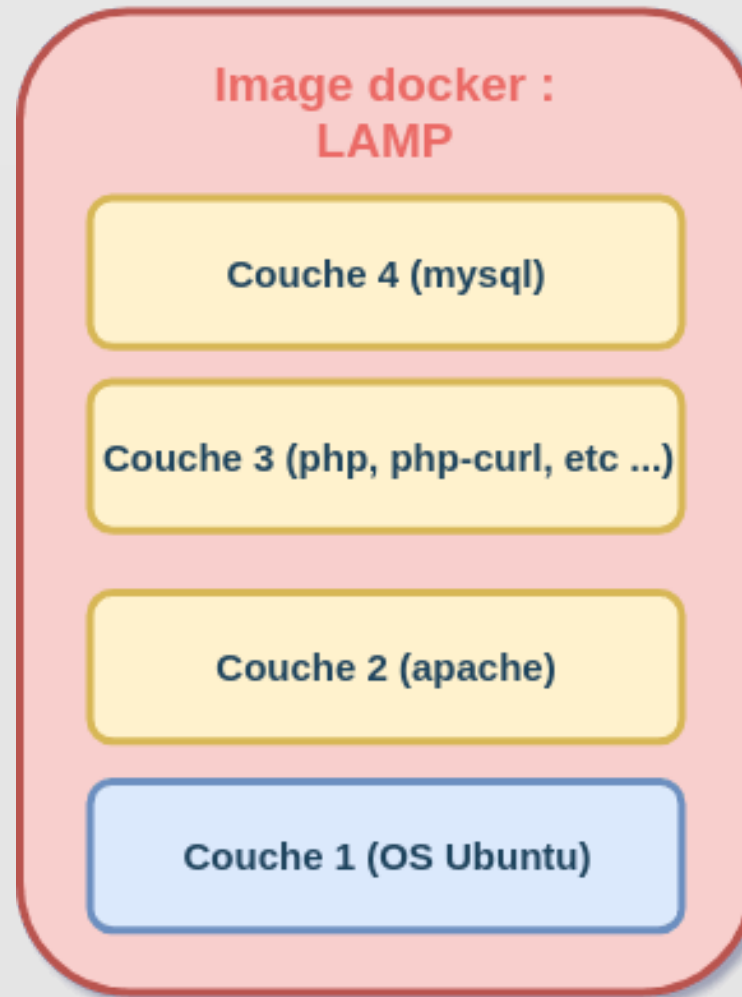
Le système hiérarchique des cgroups permet de limiter les ressources de plusieurs cgroups d'un coup, un cgroup situé en aval d'un autre ne peut dépasser les limites imposées par ce dernier.

DOCKER : IMAGE

Un conteneur est lancé en exécutant une image. Une image est un template (en lecture seule) avec les informations nécessaire à la création d'un container. À savoir :

- Le code
- L'exécution
- Les variables d'environnement
- Les bibliothèques
- Les fichiers de configuration

Une image est un modèle composé de plusieurs couches, ces couches contiennent notre application ainsi que les fichiers binaires et les bibliothèques requises.



DOCKER : CONTAINER

Un conteneur est une instance exécutable d'une image.

Vous pouvez créer, démarrer, arrêter, déplacer ou supprimer un conteneur à l'aide de l'API ou de la CLI Docker. Vous pouvez connecter un conteneur à un ou plusieurs réseaux, y attacher du stockage ou même créer une nouvelle image en fonction de son état actuel.

Par défaut, un conteneur est relativement bien isolé des autres conteneurs et de sa machine hôte. Vous pouvez contrôler le degré d'isolement du réseau, du stockage ou d'autres sous-systèmes sous-jacents d'un conteneur par rapport aux autres conteneurs ou à la machine hôte.

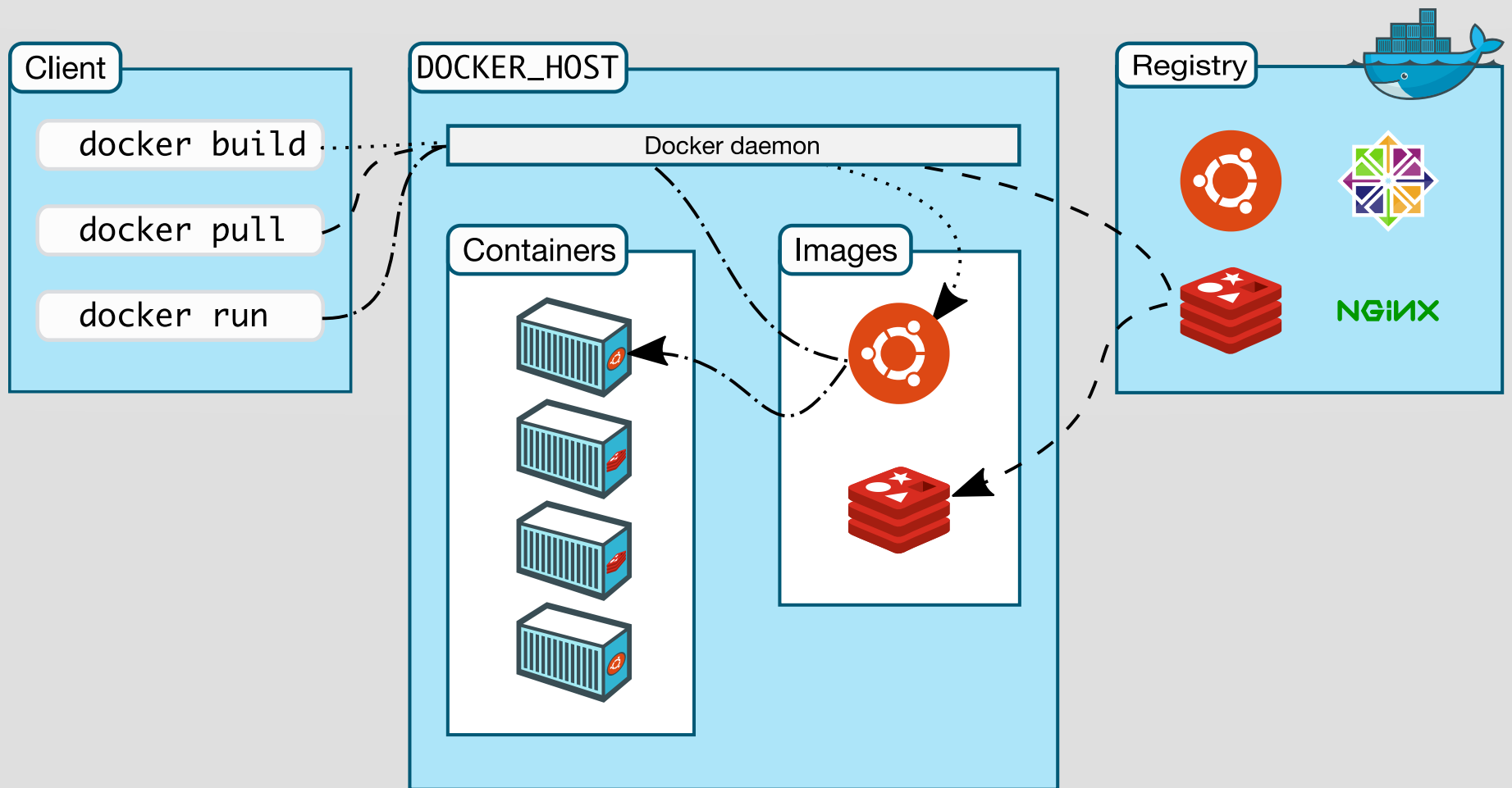
Un conteneur est défini par son image ainsi que par les options de configuration que vous lui fournissez lorsque vous le créez ou le démarrez. Lorsqu'un conteneur est supprimé, toutes les modifications apportées à son état qui ne sont pas stockées dans le stockage persistant disparaissent.

DOCKER : ARCHITECTURE

Docker utilise une architecture client-serveur.

Le client Docker parle au démon Docker , qui s'occupe de la construction, de l'exécution et de la distribution de vos conteneurs Docker.

Le client et le démon Docker peuvent s'exécuter sur le même système, ou vous pouvez connecter un client Docker à un démon Docker distant.



DOCKER : COMMANDES DE BASE - LES IMAGES

```
## Afficher de l'aide
docker help
docker <sous-commande> --help

## Afficher des informations sur l'installation de Docker
docker --version
docker version
docker info

## Executer une image Docker
docker run hello-world

## Lister des images Docker
docker image ls
# ou
docker images

## Supprimer une image Docker
docker images rmi <IMAGE_ID ou IMAGE_NAME> # si c'est le nom de l'image qui est spécifié a
-f ou --force : forcer la suppression

## Supprimer tous les images Docker
docker rmi -f $(docker images -q)

## Rechercher une image depuis le Docker hub Registry
docker search ubuntu
--filter "is-official=true" : Afficher que les images officielles

## Télécharger une image depuis le Docker hub Registry
```

DOCKER : COMMANDES DE BASE - LES CONTAINERS

Exécuter une image Docker

`docker run <CONTAINER_ID ou CONTAINER_NAME>`

- `-t` ou `--tty` : Allouer un pseudo TTY
- `--interactive` ou `-i` : Garder un STDIN ouvert
- `--detach` ou `-d` : Exécuter le conteneur en arrière-plan
- `--name` : Attribuer un nom au conteneur
- `--expose`: Exposer un port ou une plage de ports
- `-p` ou `--publish` : Mapper un port "`<PORT_CIBLE:PORT_SOURCE>`"
- `--rm` : Supprimer automatiquement le conteneur quand on le quitte

Lister des conteneurs en état running Docker

`docker container ls`

ou

`docker ps`

- `-a` ou `--all` : Afficher tous les conteneurs peut-importe leur état

Supprimer un conteneur Docker

`docker rm <CONTAINER_ID ou CONTAINER_NAME>`

- `-f` ou `--force` : forcer la suppression

Supprimer tous les conteneurs Docker

`docker rm -f $(docker ps -aq)`

Exécuter une commande dans un conteneur Docker

`docker exec <CONTAINER_ID ou CONTAINER_NAME> <COMMAND_NAME>`

- `-t` ou `--tty` : Allouer un pseudo TTY
- `-i` ou `--interactive` : Garder un STDIN ouvert
- `-d` ou `--detach` : lancer la commande en arrière plan

TP

install docker windows :

<https://www.docker.com/products/docker-desktop> install

docker centos :

<https://docs.docker.com/engine/install/centos/>

LIENS UTILES :

Tuto SSH : <https://doc.ubuntu-fr.org/ssh>

Tutos docker : <https://geekflare.com/fr/docker-tutorials/>

Les namespaces linux :

https://en.wikipedia.org/wiki/Linux_namespaces

Cours docker online : <https://antoine-vaisset-pro.github.io/dockernco>

