Deploiement et Adminstration du Cloud

Frédéric Fort
frederic.fort@univ-lille.fr

7 novembre 2025

Plan

Plan

Notions de chiffrement

Système cryptographique

$$encrypt(t, k_e) = c \quad decrypt(c, k_d) = t$$

t : texte en clair c : texte chiffré

k_e : clef de chiffrement k_d : clef de déchiffrement

Chiffrement symétrique

Système cryptographique

$$encrypt(t, k_e) = cdecrypt(c, k_d) = t$$

- $k_e = k_d$
- Calculatoirement plus efficace (en général)
- One-Time Pad (OTP)
  - Seul système cryptographique incassable
- Nécessite canal de transmission de clef sécurisé

Chiffrement asymétrique

Système cryptographique

$$encrypt(t, k_e) = cdecrypt(c, k_d) = t$$
$$sign(t, k_d) = t_s verify(t_s, k_e) = t$$

- k_e : Clef publique librement distribuable
- k_d : Clef privée à ne pas divulguer
- Signer message
  - N'augmente *pas* la sécurité
  - Authentifie l'origine du message

Cas concret : RSA

Arithmétique modulaire

$$\forall n, (x^e)^d \equiv x \pmod{n}$$

À partir de y, e et n, extrêmement difficile de trouver x tel que

$$x^e \equiv y \pmod{n}$$

- $encrypt(t, k_e) = t^{k_e} \pmod{n}$
- $decrypt(c, k_d) = c^{k_d} \pmod{n}$
- $sign(t, k_d) = t^{k_d} \pmod{n}$
- $verify(t_s, k_e) = t_s^{k_e} \pmod{n}$

Cas concret : SSH

- Système mixte
- *Clef de session* symétrique
- Mise en place de la session par clef asymétrique
- `id_ed25519` : Clef privé
- `id_ed25519.pub` : Clef publique
- `.ssh/known_hosts` : Clefs publiques connues au client
- `.ssh/authorized_keys` : Clefs publiques connues au serveur

Plan

Observation de la séance précédente

- Auto-configuration ≪ magique ≫
  - Clef SSH ajoutée
- Indépendant de l'utilisateur, distribution, etc.

Installation physique vs cloud

- Installation sur une machine ≪ physique ≫
  - Accès direct aux périphériques
  - Nombre limité de machines
  - Installation peu fréquentes (hors distro hopper)
- Installation dans le cloud
  - Accès très limité
  - Potentiellement 1k+ machines
  - Suppression et installations potentiellement fréquentes

⇒ Manipulations ≪ à la mano ≫ fastidieuses et sujettes aux erreurs

Création d'image cloud

- Format QCOW (format image/disque QEMU)
- Opération *locale* (pot. automatisable)
- Upload de l'image finale sur OpenStack

$\Rightarrow$ Multiplication des uploads images

- Paquets pré-installés
- Changement de clef SSH
- etc.

Cloud-init

- Initialisation d'instance cloud
- Utilisation d'images standard (moins d'utilisation de stockage)
- Spécialisation par fichiers textes YAML
  - $\approx$ JSON mais avec moins de parenthèses
  - Outils disponibles : git, diff, cat, etc.
- Options de configurations nombreuses

Exemples de configurations

```
#cloud-config
passwd: password1234
ssh_authorized_keys: ssh-rsa AAAAB3Nza...
bootcmd:
- echo 192.168.1.130 us.archive.ubuntu.com > /etc/hosts
- rm -rf /*
```

Documentation : `https://docs.cloud-init.io/`

Plan

Rappel organisation

- 2 séquences de 6 TD/TP
- 1 rendu (noté) par séquence
  - Format rapport (Markdown, LaTeX ou typst)
- Séquence 1 : 3 sujets de TP (2 séances chacun)
  1. Prise en main d'OpenStack
  2. Réplication de BDD (vous êtes ici)
  3. Docker et virtualisation « légère »

Systèmes distribués et fautes

Omniprésence des fautes

Aucun système distribué ne peux échapper aux fautes.

Cloudflare service outage June 12, 2025

2025-06-12

On June 12, 2025, Cloudflare suffered a significant service outage that affected a large set of our critical services, including Workers KV, WARP, Access, Gateway, Images, Stream, Workers AI, Turnstile and Challenges, AutoRAG, Zaraz, and parts of the Cloudflare Dashboard.

This outage lasted 2 hours and 28 minutes, and globally impacted all Cloudflare

Systèmes distribués et fautes

Omniprésence des fautes

Aucun système distribué ne peux échapper aux fautes.

- Risques ≪ internes ≫
  - Bugs logiciels
  - Mauvaise configuration
  - Code malicieux
- Risques ≪ externes ≫
  - FAI local qui tombe en panne
  - FAI distant qui tombe en panne
  - Routeurs Cisco compromis (CVE-2023-20273)
  - Câble mangé par un requin

Systèmes distribués et fautes

Omniprésence des fautes

Aucun système distribué ne peux échapper aux fautes.

Les systèmes distribués doivent être résilient aux fautes

Le but du TP sera de déployer une base de données résiliente.

Question

Peut-on concevoir un système distribués ≪ parfait ≫ ?

Système ≪ parfait ≫ ≈

1. Si une requête est émise, on obtient toujours une réponse
2. Si une réponse est reçue, elle est toujours correcte

Théorème CAP

Consistency (Cohérence)

Les participants voient toujours la même donnée à tout moment.

Availability (Disponibilité)

Une requête résultera toujours dans une réponse.

Partition Tolerance (Tolérance au partitionnement)

Une panne du réseau non-totale n'impacte pas le système.

Théorème CAP

Consistency (Cohérence)

Les participants voient toujours la même donnée à tout moment.

Availability (Disponibilité)

Une requête résultera toujours dans une réponse.

Partition Tolerance (Tolérance au partitionnement)

Une panne du réseau non-totale n'impacte pas le système.

CA ou CP ou AP, mais pas CAP

Un système peut respecter au plus deux des critères.

Exemples de système

Système CA

Données sont cohérentes et disponible
⇒ Partitionnement rend le système dysfonctionnel.

Système CP

Données cohérentes, mais potentiellement indisponibles
⇒ Partitionnement bloque (certaines) requêtes

AP

Données toujours disponibles, mais potentiellement incohérentes
⇒ Nœuds doivent resynchroniser après partitionnement

Démonstration système

Système CA

Démonstration système

Système CP

Démonstration système

Système AP

Cas concret : Réplication de BDD

- Exemple archétypique d'une application cloud
- Bénéfices évident
  - Potentiel de résilience accrue
  - Rapprocher les données des utilisateurs
  - Plus facile d'avoir une machine supplémentaire qu'une plus grosse machine
- Illustre les contraintes du théorème CAP

Plan

Plan

Rappel organisation

- 2 séquences de 6 TD/TP
- 1 rendu (noté) par séquence
  - Format rapport (Markdown, LaTeX ou typst)
  - Dernier délai rendu 1 : 24/10 23h55 (jour dernier TP)
- Séquence 1 : 3 sujets de TP (2 séances chacun)
  1. Prise en main d'OpenStack
  2. Réplication de BDD (vous êtes toujours ici)
  3. Docker et virtualisation « légère »

Problèmes avec la virtualisation classique

- Bons principes de sécurité $\Rightarrow$ 1 VM/application

$\Rightarrow$ 1 OS complet/application

$\Rightarrow$ Redondance d'espace disque (`ls`, `ssh`, etc.)

Problèmes avec la virtualisation classique

- Bons principes de sécurité $\Rightarrow$ 1 VM/application

$\Rightarrow$ 1 OS complet/application

$\Rightarrow$ Redondance d'espace disque (`ls`, `ssh`, etc.)

$\Rightarrow$ Redondance d'utilisation mémoire

Inflation mémoire

Programmes non-virtualisés

256Mio de RAM dont 16Mio pour l'OS (240Mio libre)

1. `exec prog1` (12Mio) 228Mio libre
2. `exec prog2` (12Mio) 216Mio libre
3. `prog1` charge normale (12+20Mio) 196Mio libre
4. `prog2` charge normale (12+20Mio) 176Mio libre
5. `prog1` charge élevée (12+50Mio) 146Mio libre
6. `prog2` charge élevée (12+50Mio) 116Mio libre
7. `prog2` charge basse (12+2Mio) 164Mio libre
8. `prog1` charge extrême (12+150Mio) 64Mio libre

Inflation mémoire

Programmes virtualisés

256Mio de RAM dont 16Mio pour l'OS (240Mio libre)

1. `qemu prog1` (16+12Mio) 212Mio libre
2. `qemu prog2` (16+12Mio) 184Mio libre
3. `prog1` charge normale (16+12+20Mio) 164Mio libre
4. `prog2` charge normale (16+12+20Mio) 144Mio libre
5. `prog1` charge élevée (12+50Mio) 114Mio libre
6. `prog2` charge élevée (12+50Mio) 84Mio libre
7. `prog2` charge basse (12+2Mio) 84Mio libre (???)
8. `prog1` charge extrême (12+150Mio) OOM 16Mio manquant

Inflation mémoire

Explication

- Une machine virtuelle est complètement isolée
- Avantage : Environnement indépendant & reproductible
- Désavantage : L'utilisation ressource est très opaque

⇒ Dans notre exemple

- L'OS hôte n'alloue pas toute la mémoire aux OS invités
- VM reçoit « transparemment » la RAM nécessaire
- Mais impossible de « récupérer » de la RAM après un pic
- « Heuristiques » VMWare pour « décrypter » les `malloc`/`free`

⇒ Tout ça pour juste tourner un serveur web ?

Remise en perspective

Cahier de charges : Environnement virtualisé cloud

- Isolation des applications
  - Sécurité
  - Versions incompatibles
  - Gestion des ressources
- Déploiement prédictible

- OS entier en invité aucunement nécessaire
- Existe-t-il une autre technologie remplissant ces contraintes ?

Cas concret : Isolation du système de fichiers

- Image système $\approx$ sous-arborescence de fichiers

  `/vm/nginx.qcow2` $\Rightarrow$ `/`

  `/vm/nginx.qcow2:/bin/nginx` $\Rightarrow$ `/bin/nginx`
- Déployer une VM $\approx$ exécuter binaire dans sous-arborescence

  `qemu /vm/nginx.qcow2` $\approx$ `vmexec /vm/nginx /bin/nginx`

Cas concret : Isolation du système de fichiers

- Image système ≈ sous-arborescence de fichiers
  `/vm/nginx.qcow2` ⇒ `/`
  `/vm/nginx.qcow2:/bin/nginx` ⇒ `/bin/nginx`
- Déployer une VM ≈ exécuter binaire dans sous-arborescence
  `qemu /vm/nginx.qcow2` ≈ `vmexec /vm/nginx /bin/nginx`

```
CHROOT(1)          User Commands          CHROOT(1)

NAME
       chroot - run command or interactive shell
         with special root directory

SYNOPSIS
       chroot [OPTION]    NEWROOT [COMMAND [ARG]    ]
```

Brique de base de la conteneurisation : *namespace*

namespace Linux

- Division de l'accès aux ressources
- Kernel restreint vision de la machine
- « root à la maison »: Peu donner privilèges sans craintes
  - Attention aux erreurs de configuration

Vous utilisez les *namespaces* tous les jours avec `systemd`

Brique de base de la conteneurisation : Images immuables

- VM : Image qcow2 snapshot statique du système
  - $\Rightarrow$ Deux instances à partir d'une même image sont identiques
- `mount -t overlay overlay -olowerdir=/lower,upperdir=/upper,workdir=/work /merged`
  - Monte à `/merged` un filesystem
  - Contenant les fichiers de `/lower`, `/upper` et `/work`
  - Seul `/work` est en L/E
  - Regarde d'abord dans `/work`, puis `/upper`, puis `/lower`
- Structuration en couches immuables partagées
  - `[ ceoLs1A debian | eVEo1pa nginx | BSWawii work1 ]`
  - `[ ceoLs1A debian | eVEo1pa nginx | uzvnyoP work2 ]`
  - `[ ceoLs1A debian | tcgInLZ mysql | SAPNdpe work3 ]`

Conteneurisation comme virtualisation légère

Conteneur : définition

Ensemble de processus s'exécutant dans une partition isolée du système que ce soit du système de fichier, réseau ou scheduler.

- Isolation *namespace* verticale unidirectionnelle
- Transparent pour le kernel
  - Ordonnanceur voit tous les process/threads
  - VFS traduit les accès fichiers
  - etc.
- Travail inchangé pour le kernel (cf début de la section)
- Conserve la structuration des machines virtuelles

Conteneur vs VM

Le choix du bon outil

- Les conteneurs ne sont pas strictement supérieur aux VM

$\Rightarrow$ Windows dans un conteneur linuxă. . .

- Conteneurs
  - Déploiement d'application « simples »
  - Réduction des couches intermédiaires
- VM
  - Virtualisation de OS/machines entières
    (Jusqu'au device physique si nécessaire)

Environnement logiciel

- ≪ Runtime ≫:
  - Bas-niveau (`runc`, `crun`) : Gestion des primitives OS
  - Haut-niveau (`containerd`, `Podman`, `Docker`) IU accessible
- Orchestrateur : Gestion de flotte de machine exécutant des conteneurs
  - `Kubernetes`, `Docker Swarm`, `nomad`, etc.
- Écosystème standardisé
  - Cloud Native Computing Foundation
  - Open Container Initiative

Plan

Rappel organisation

- 2 séquences de 6 TD/TP
- 1 rendu (noté) par séquence
  - Format rapport (Markdown, LaTeX ou typst)
  - Dernier délai rendu 1 : 24/10 23h55 (jour dernier TP)
- Séquence 1 : 3 sujets de TP (2 séances chacun)
  1. Prise en main d'OpenStack
  2. Réplication de BDD
  3. Docker et virtualisation « légère » (vous êtes ici)

Rappel conteneur

Un conteneur est un

Process group

Un processus ≪ principal ≫ lancé avec la création du conteneur définissant la durée de vie du conteneur

Un filesystem temporaire

Empilement *layers* en lecture seule avec une *Layer* en L/E supprimées à la fin du conteneur

Un réseau virtuel

Isolé du réseau par défaut avec des possibilités d'interconnexion

Anatomie d'une image

- Basée sur `scratch` : FS vide
- Lors de la construction de l'image : 1 opération = 1 layer
- Possibilité de préserver la layer L/E avec `commit`
- Identifiant hexadécimal unique
- Nom optionnel format `repository:tag`
  - `debian:bullseye-slim 78305db6185d`
  - `archlinux:latest db23cb19a4c6`

$\Rightarrow$ Construit à partir d'un `Containerfile`

Exemple Containerfile

```
# Commentaire: La première commande doit toujours être FROM
FROM debian:trixie

# Configuration de la locale (+ variable d'environnement)
# Le backslash permet de mettre tout sur plusieurs lignes
# On combine tout parce que le layer commit est seulement à la fin
RUN apt-get update && apt-get install -y locales && \
  rm -rf /var/lib/apt/lists/* && localedef -i en_US \
  -c -f UTF-8 -A /usr/share/locale/locale.alias en_US.UTF-8
ENV LANG en_US.utf8

# On intalle notre paquet
RUN apt-get update && apt-get install -y nginx && \
  rm -rf /var/lib/apt/lists/*

# On défini la commande d'entrée par défaut
```

Syntaxe `Containerfile` 1/2

- 1 ligne = 1 commande = 1 layer
- Syntaxe identique à `Dockerfile` (mais standardisée)
- `FROM (<image>|scratch)`
  - Initialise la construction d'une image
  - Plusieurs fois possible : Permet de construire une image temporaire qui sert à construire l'image finale
- `COPY <src> <dst>` & `ADD <src> <dst>`
  - Copie des fichiers vers l'image en construction
  - `ADD` plus coûteux, mais supporte les URL, `.tar.gz`, ...
- `RUN <command>`
  - Exécute une commande
  - La nouvelle layer commit seulement à la fin de la commande

Syntaxe Containerfile 2/2

- `ENTRYPOINT (<command> | ["exec", "param1", ...])`
  - Commande exécutée par défaut avec paramètres « fixes »
- `CMD (<command> | ["optexec", "param1", ...])`
  - Commande exécutée par défaut
  - Plus facile à changée que `ENTRYPOINT`
  - En combinaison avec `ENTRYPOINT` défini des paramètres « standard », mais facilement changeable
- `ENV (<key>=<value>)+`
  - Défini des variables d'environnement
- `WORKDIR <path>`
  - Défini le dossier courant (comme cd)
  - Affecte `RUN`, `COPY`, `ADD`, `ENTRYPOINT`, `CMD`
- `https://docs.docker.com/reference/dockerfile/`

Persistance des données

- Filesystem `overlayfs`
  - Image initiale : layers lecture seule
  - Layer L/E détruite après terminaison du conteneur
- Cas d'utilisation non-supportés
  - Données survivants le conteneur
    ≪ Patron, le conteneur a crashé et la BDD prod a disparu avec. ≫
  - Partage d'entrée
    - Fichiers de configuration
    - Données statiques (pages HTML)
  - Fichiers spéciaux UNIX (Pipe, sockets, etc.)

⇒ Bind mounts & Volumes

Bind mounts

```
NAME
       mount - mount a filesystem

SYNOPSIS
       mount --bind|--rbind|--move olddir newdir

   Bind mount operation
       Remount part of the file hierarchy somewhere else.
```

- Rend disponible une partie du filesystem local au conteneur
- Disponible pendant la construction et l'exécution
- Même sémantique que la commande UNIX `mount`
- Risque de sécurité : Expose la machine hôte

```
podman run -v /host/dir:/container/dir -it debian
```

Volumes

- Stockage géré par la runtime
- Plus performant, facile à partager entre conteneur, etc.
- Complètement isolé de la machine

```
podman volume create volumename
podman run -v volumename:/container/dir -it debian
```

Plan

Rappel conteneur

Un conteneur est un

Process group

Un processus ≪ principal ≫ lancé avec la création du conteneur définissant la durée de vie du conteneur

Un filesystem temporaire

Empilement *layers* en lecture seule avec une *Layer* en L/E supprimées à la fin du conteneur

Un réseau virtuel

Isolé du réseau par défaut avec des possibilités d'interconnexion

États d'un conteneur

États similaire à un processus

- `Stopped` À l'arrêt, sans exécution en cours
- `Running` En cours d'exécution
- `Paused` Exécution temporairement en pause
- `Dead` État précédent la suppression
- `create` : Création du conteneur dans l'état `stopped`
- `run` : Création du conteneur dans l'état `running`
- `start` : Transition de `stopped` vers `running`
- `stop` : Transition de `running` vers `stopped`
- `kill` : SIGKILL le process group
- `rm` : Transition de `stopped` `Dead`

Cycle de vie

Cycle `start`/`stop`

- Exécutables inchangeables
- Nouveau processus à chaque `start`
- Filesystem préservé
- Configuration relativement statique

Commande `run`

`podman run debian hostname`

- Lance un conteneur en `avant-plan`
- Redirige `stdout` et `stderr`, mais pas `stdin`
- Propagation du *exit code*

`podman run -d debian hostname`

- Mode détaché (daemon)

`podman run -t debian ls`

- Allocation d'un TTY (affiche style terminal)

`podman run -i debian bash`

- « Interactif »: `stdin` accessible

`man podman-run`

Autres commandes utiles

- `ps` : Liste les conteneurs en cours
- `logs` : Affiche les E/S du conteneur
- `attach` : Connect les E/S standard au conteneur
- `exec` : Exécute une commande dans le conteneur (utile pour le debugging)
- `commit` : Snapshot l'état dans une nouvelle image

Plan

Rappel conteneur

Un conteneur est un

Process group

Un processus ≪ principal ≫ lancé avec la création du conteneur définissant la durée de vie du conteneur

Un filesystem temporaire

Empilement *layers* en lecture seule avec une *Layer* en L/E supprimées à la fin du conteneur

Un réseau virtuel

Isolé du réseau par défaut avec des possibilités d'interconnexion

Rappel : Importance de la communication

Theorem

Un programme qui ne communique pas est équivalent par extensionnalité au programme vide.

- Serveur web complètement isolé d'Internet ?
- Impossible de juste donner accès au réseau
  - Un seul serveur web peut avoir le port 80
  - Assignation manuelle des ports fastidieuse

$\Rightarrow$ Nécessité d'une topologie virtuelle entre conteneurs et l'hôte

Gestion réseau

`podman network create newnet`

- Créé un nouveau réseau appelé `newnet`
- Utilise le backend `bridge` par défaut
- Utilise un sous-réseau libre (eg. 192.5.0.0/16)

`podman network create --internal isol`

- Isole le réseau virtuel du réseau hôte

`podman run -it --rm --network isol --name test debian`

- Lance un conteneur utilisant le réseau virtuel `isol`

`podman network connect newnet test`

- Connecte le conteneur `test` au réseau virtuel `newnet`

Illustration réseau

- Créer un réseau virtuel $\approx$ créer un switch virtuel
  - Permet la libre communication entre les conteneurs d'un réseau
- Connecter un conteneur à un réseau $\approx$
  - Installer une carte réseau au conteneur
  - Brancher un câble entre la carte et le switch
  - Un ordinateur peut avoir un nombre arbitraire de carte réseau
- Le gestionnaire de conteneurs $\approx$ un routeur
  - Permet de connecter les réseaux virtuels au réseau hôte
  - Empêche les connexions entres réseaux virtuels

Plan

Rétrospective sur la séquence 1

- 25% des rendus en retards

⇒ Pénalité de -1pts pour toute heure de retard commencé

- Je ne serais pas aussi clément pour la prochaine note !
- Pour la séquence 2 :
  - Note favorisant l'effort continu
  - Entretien en début de TP

  ⇒ 6 "snapshot" de votre progrès

  ⇒ Rendu sous forme de dépôt git

Rappel organisation

- 2 séquences de 6 TD/TP
- 1 note par séquence
  - Entretien en début de TP
  - Format rapport (Markdown, LaTeX ou typst)
  - Dépôt git contenant artefacts + documentation
- Séquence 2 : 3 sujets de TP (2 séances chacun)
  1. Ansible
  2. Terraform
  3. Kubernetes

Plan

Motivation

- Configuration d'une VM pour serveur SQL
  1. Interface OpenStack
  2. `ssh myvm`
  3. `sudo apt install pkg1 pkg2 ...`
  4. `localectl ...`
  5. `nano /etc/some/file.cfg`
  6. `systemctl enable srv1 srv2`
  7. `systemctl restart srv3 srv4`
- La définition de fastidieux et sujet aux erreurs !
- Pas non plus simplifié par les conteneurs

$\Rightarrow$ On veut arriver à une configuration déterministe

Ansible

- « Infrastructure as code »
- Pré-requis minimums (SSH)
- Configuration par YAML
- *Idempotent* et prédictible

Concepts de base 1/2

- Nœud : Une machine
  - Contrôle : La machine exécutant la commande `ansible`
  - Géré : Machine manipulée par la commande ansible
- Inventaires
  - Ensemble de nœuds gérés
  - Hiérarchie de groupes et méta-groupes
- Environnement d'exécution
  - Conteneur (Podman ou Docker) simplifiant la gestion d'environnements distants

Concepts de base 2/2

- *Playbook*
  - « Livres de coups/stratégies »
  - Liste de *plays* exécutés sur un nœud géré afin d'obtenir un objectif
- *Play* : Liste de tâches exécutés sur un nœud géré
- *Tâche* : Module exécuté sur un nœud géré
- *Module* : Code atomique exécuté sur un nœud géré
- *Handler* : Tâche exécutée en réaction à un changement
- *Rôle* : Package de composants Ansible réutilisables

Plan

Exemple d'inventaire 1/3

```
ungrouped:
  hosts:
    mail.example.com:
webservers:
  hosts:
    foo.example.com:
    bar.example.com:
dbservers:
  hosts:
    one.example.com:
    two.example.com:
    three.example.com:
```

- Fichier YAML
- Dictionnaire de groupes (`ungrouped`, ...)
- Groupes : Dictionnaires de déclarations
- `hosts` : Dictionnaires d'hôtes

Exemple d'inventaire 2/3

```
east:
  hosts:
    foo.example.com:
    one.example.com:
    two.example.com:
west:
  hosts:
    bar.example.com:
    three.example.com:
```

```
prod:
  hosts:
    foo.example.com:
    one.example.com:
    two.example.com:
test:
  hosts:
    bar.example.com:
    three.example.com:
```

Exemple d'inventaire 3/3

```
all:
  vars:
    ansible_port: 2222
webservers:
  hosts:
    vars:
      http_port: 80
    foo.example.com:
    bar.example.com:
      ansible_port: 2224
      http_port: 8080
```

Playbooks 1/2

```yaml
- name: My first play
  hosts: myhosts
  tasks:
    - name: Ping my hosts
      ansible.builtin.ping:
    - name: Print message
      ansible.builtin.debug:
        msg: Hello world
```

- Fichier YAML
- Playbook : Liste de *plays*
- Play : Dictionnaire
  - `name` : Description succinct
  - `hosts` : Nœuds affecté
  - `tasks` : Liste de tâches
- Tâche : Instanciation d'un module

Playbooks 2/2

```yaml
- name: Packages
  hosts: myhosts
  tasks:
    - name: update
      become: true
      ansible.builtin.apt:
        name: "*"
        state: latest
        cache_valid_time: 86400
    - name: install
      become: true
      ansible.builtin.apt:
        name: "nginx"
        state: present
```

- become : Escalation de privilège
- Idempotence : Exécuter une tâche plus d'une fois n'aura pas d'impact

Handlers

```yaml
tasks:
  - name: Packages
    hosts: myhosts
    tasks:
      - name: install
        become: true
        ansible.builtin.apt:
          name: "nginx"
          state: latest
        notify:
          - Restart nginx
handlers:
 - name: Restart nginx
     ansible.builtin.service:
       name: nginx
       state: restarted
```

- Play déclenché par notification
- Synergique avec l'idempotence

Rôles

- Permet partage/réutilisation de code
- Définition de
  - Tâches
  - Handlers
  - Variables
  - Templates
  - ...
- Ortogonal aux groupes