جامعة الحسن الثاني بالدار البيضاء
UNIVERSITÉ HASSAN II DE CASABLANCA

FSBM
FACULTE DES SCIENCES BEN M'SICK
UNIVERSITE HASSAN II DE CASABLANCA

MODULE :

Le Cloud Computing & DevOps

Pr. F. Benabbou
Master DSBD
Faculté des Sciences Ben M'Sik Casablanca

TABLE OF CONTENTS

02

Gérer les déploiements de l'infrastructure avec Ansible et Terraform

Infrastructure en tant que Code (IaC)

Déploiement traditionnel de l'infrastructure

ANSIBLE

- Supposons que l'admin doit installer le paquet nginx sur un ensemble de serveurs distants.
- L'approche traditionnelle est une approche séquentielle où chaque étape est manuelle ou semi-automatisée.
- les étapes seraient :
  - Connexion à chacun des serveurs un par un.
  - Configuration de chaque serveur.
  - Installation de ngix sur chacun.
- On remarque que les mêmes tâche se répètent et sont traité séquentiellement.

```
$ ssh www1.example.com
www1$ sudo vi /etc/resolv.conf
www1$ sudo apt-get install nginx
:
$
$ ssh www2.example.com
www2$ sudo vi /etc/resolv.conf
www2$ sudo apt-get install nginx
:
$
$ ssh www3.example.com
www3$ sudo vi /etc/resolv.conf
www3$ sudo apt-get install nginx
:
:
: etc …
```

Automatiser avec script shell

ANSIBLE

- On peut écrire un script shell ou les tâches seront dans une boucle.
- Mais cela reste difficile à écrire et difficile à maintenir quand on veut personnaliser la configuration sur chaque serveur.
- Il y a un risque de de provoquer des erreurs

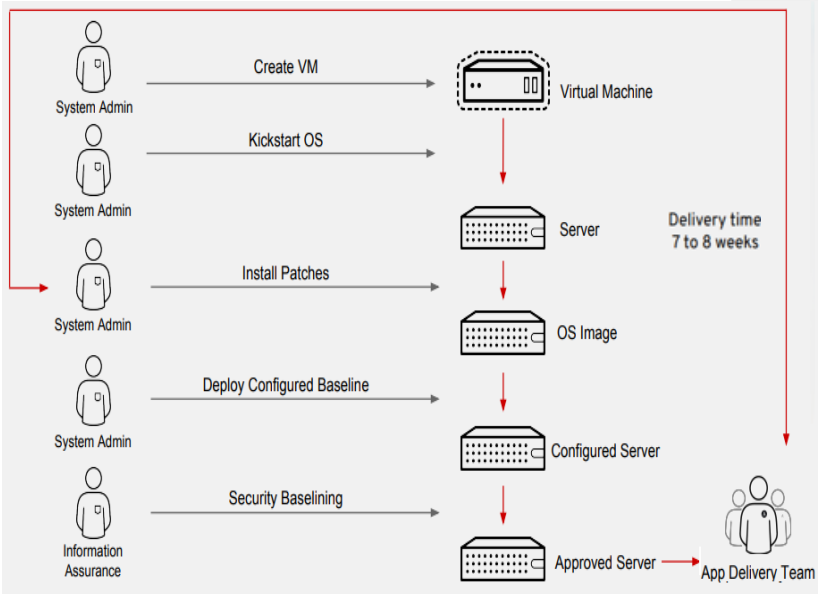
```
#!/bin/sh

HOSTS="
 www1.example.com
 www2.example.com
 www3.example.com


"

for host in $HOSTS
do
      # Copy DNS settings to all servers
      scp resolv.conf  $host:/etc/resolv.conf

      # Install Nginx
      ssh $host "sudo apt-get install nginx"
done
```

Un autre exemple

ANSIBLE

- On désire déployer une VM avec un OS sur plusieurs serveurs.
- Pour créer une machine il faut faire manuellement les étapes, suivantes:

  - Create VM: créer une instance de machine virtuelle et définition des ressources (CPU, RAM, espace disque)
  - Kickstart OS: Kickstart est un système d'installation automatisé pour les distributions Linux.
  - Install patches: c.-à-d. les mises à jour de sécurité et les correctifs pour le système d'exploitation et les logiciels installés.
  - Deploy basic config: le déploiement de la configuration réseau et système de base de la VM (install App/DB, etc.).
  - Security: mise en place des mesures de sécurité pour protéger la VM.
  - Approved server: Une fois toutes les étapes précédentes terminées, le serveur est validé et approuvé avant d'être remis à l'équipe qui déploie les applications.



Modèle traditionnel de Déploiement de d'infrastructures

- Constructions manuelles et séquentielle
- Les mêmes tâches ou configurations ne peuvent être exécutées plusieurs fois, sur plusieurs systèmes ou environnements, avec les mêmes résultats garantis
- Faible efficacité
- Risque d'erreur humaine
- Absence de monitoring globale et gestion de la détection des erreurs
- Exige des personnes très techniques

Déploiement automatique de l'infrastructure avec Ansible

ANSIBLE

- Ansible est un outil d'automatisation open-source qui permet de gérer de manière efficace et reproductible l'infrastructure : Infrastructure as code (IaC).
- Ecrit en python, il a été créé en 2012, racheté par Red Hat depuis 2015.
- Ansible permet l'exécutions des tâches (task) en parallèle sur plusieurs hôtes, ce qui permet d'accélérer les déploiements.
- Une 'task' correspond à une action spécifique exécutée sur les hôtes cibles, comme installer un paquet, copier un fichier ou redémarrer un service.

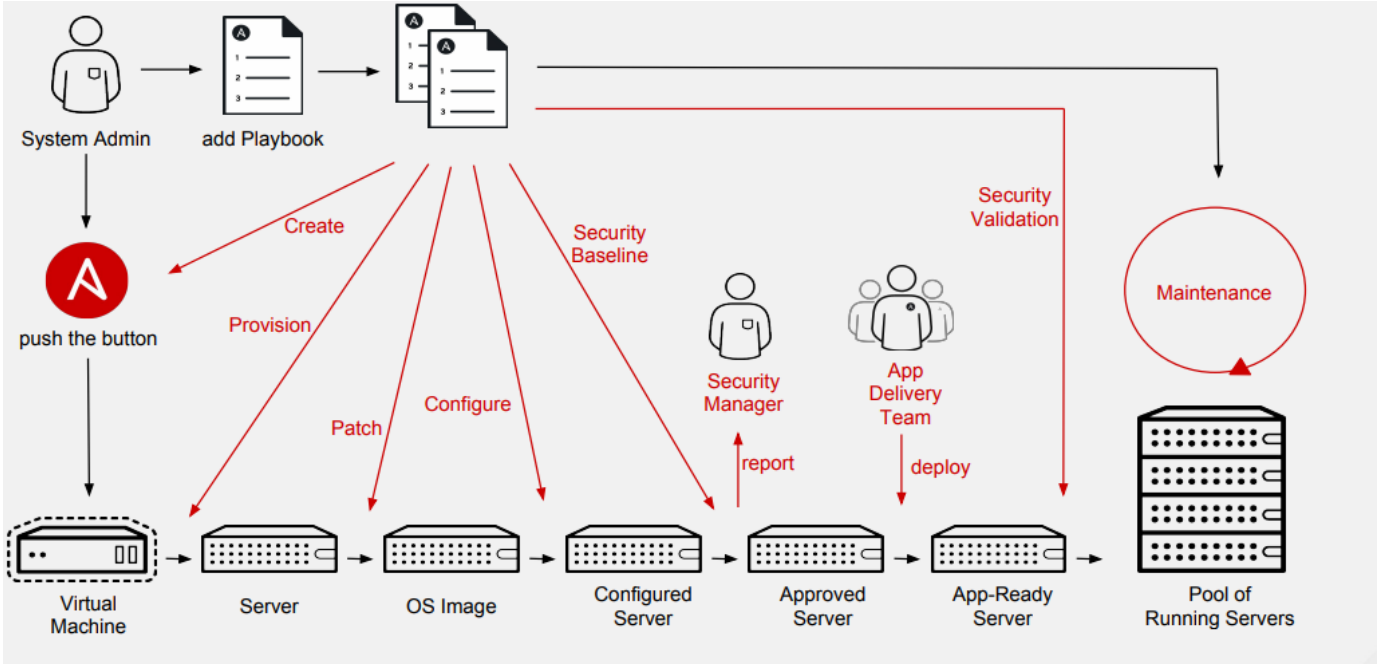
Déploiement automatique de l'infrastructure avec Ansible

ANSIBLE

- Ansible assure :
  - Le déploiement et configuration de nouveaux serveurs.
  - L'automatisation des configurations des bases de données, du stockage, des réseaux, des pare-feu, de plusieurs serveurs en même temps.
  - Le déploiement d'applications sur plusieurs environnements et gestion correctifs (Full Stack Deployment: de l'installation des outils dependance au déploiement de l'application).
  - L'orchestration et coordination des tâches complexes sur plusieurs hôtes.
  - La sécurité en utilisant le protocole SSH ou Remote PowerShell pour la gestion des ressources à distance (Sans agent).

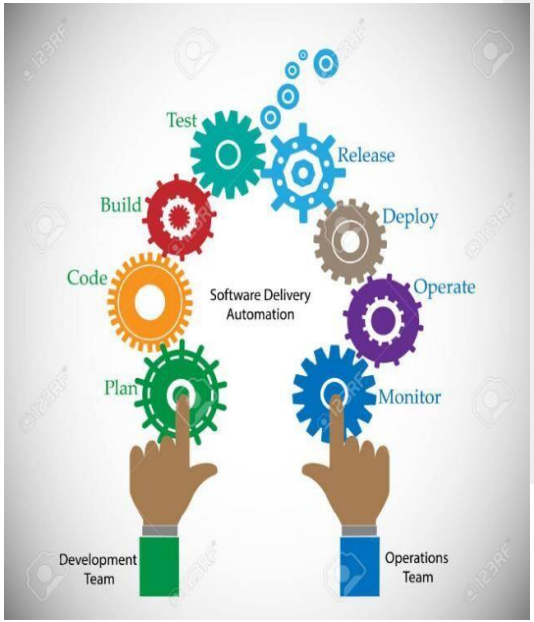
Déploiement automatique de l'infrastructure avec Ansible

Ansible

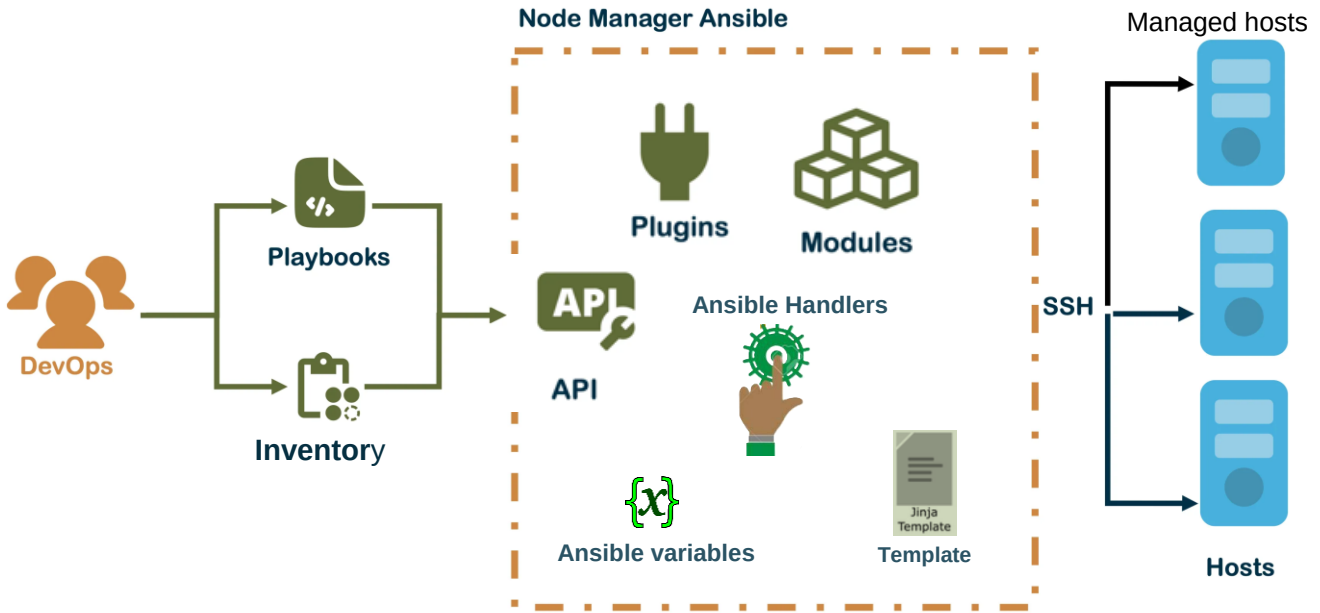


Déploiement automatique de l'infrastructure avec Ansible

- Ansible fonctionne sur de nombreux systèmes d'exploitation, y compris Unix et Windows.
- Ansible peut être utilisé dans plusieurs étapes du pipeline **DevOps**, notamment:
  - Déployer des applications sur des serveurs ou des conteneurs.
  - Configurer les bases de données, les environnements applicatifs ou les réseaux.
  - Tester automatiquement et vérifier que les configurations ou les applications sont déployées correctement.
  - Appliquer les mises à jour et les correctifs logiciels à grande échelle sans intervention manuelle (Operational Tasks).
  - Monitorer en s'intégrant aux outils de monitoring DevOps.

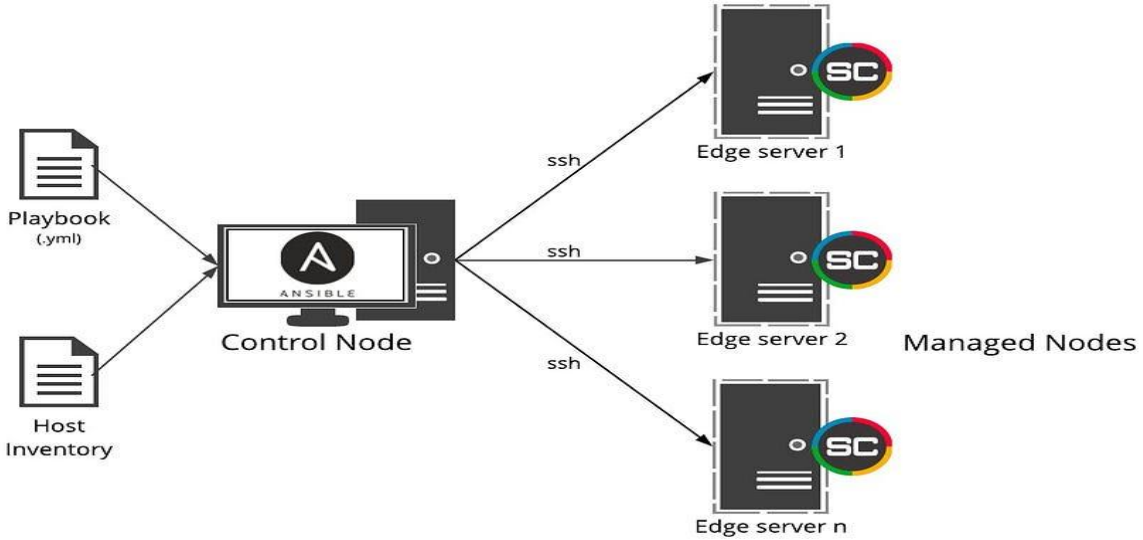


Architecture Ansible



Managed Nodes (cibles)

- Ce sont les machines ciblées pas la configuration



API Ansible

- L'API Ansible offre un ensemble de fonctionnalités permettant aux développeurs et aux outils d'interagir avec le moteur d'Ansible.
- L’API permet de :
  - Déclencher des playbooks de manière programmatique
  - Récupérer des informations sur l'état des tâches
  - Créer des modules personnalisés
  - Intégrer Ansible avec des outils de gestion de configuration, des systèmes de suivi des incidents, etc.

Inventory

- Le fichier inventaire inventory.yml contient une liste des hôtes à gérer, avec leurs adresses IP, noms d'hôte, groupes, etc.

```
all:
  hosts:
    server1:
      ansible_host: 192.168.1.10
    server2:
      ansible_host: 192.168.1.11
  children:
    web_servers:
      hosts:
        server1:
        server2:
```

- `All` : définit le groupe global
- `Hosts` : liste les hôtes sous le groupe all, ici server1 et server2.
- `children`: spécifie les groupes enfants, ici c'est web_servers.
- `Hosts` : Hôtes du groupe web_servers : server1, server2

Playbooks

- Un playbook Ansible est un fichier, généralement écrit en YAML
- Il contient un ensemble d'instructions pour automatiser des tâches qu'on veut exécuter sur un ou plusieurs systèmes.
- Il décrit la situation souhaitée de l'infrastructure, et Ansible s'occupe de mettre en œuvre les changements nécessaires pour atteindre cet état.
- Ce playbook permet d'installer le paquet apachesur tous les hôtes appartenant au groupe web_servers.

```
- name: Installer Apache
  hosts: web_servers
  tasks:
    - name: Installer le paquet Apache
      ansible.builtin.apt:
        name: Apache
        state: present
    - name: Démarrer le service Apache
      service:
        name: Apache
        state: started
```

• - `name`: Installer Nginx: définit la tâche principale, ici on veut installer et démarrer le serveur web Nginx.
• `hosts: web_servers` : indique le groupe d'hôtes concerné par cette tâche, ici c'est web_servers défini dans l'inventaire Ansible.
• `tasks`: définit les actions spécifiques (tâches) qu'Ansible doit exécuter sur les hôtes ciblés, ici on a 2 tâches : Installer le paquet apache via apt et démarrer le serveur.
• `state: present`: garantit que s'il n'est pas déjà présent, il sera installé.

Variables

ANSIBLE

- Les variables permettent de stocker et livrer des informations sur les données système.
- Elles rendent les playbooks plus flexibles et adaptables à différents environnements.
- Elles peuvent être définies au niveau du playbook, d'un inventaire ou même passées en ligne de commande.

```yaml
- name: Créer un fichier texte personnalisé
  hosts: all
  become: true

  vars:
    file_path: /tmp/mon_fichier.txt
    file_content: |
      Bienvenue dans la configuration automatisée avec Ansible !
      Ce fichier a été créé automatiquement.

  tasks:
    - name: Créer un fichier avec du contenu
      copy:
        dest: "{{ file_path }}"
        content: "{{ file_content }}"
```

• **vars** permet de définir les variables file_path et file_content au niveau du playbook

• Elles pourront être utilisées pour générer le fichier sur chaque host.

Les facts d'Ansible

- Les « gather facts », sont des variables qu'Ansible recueille sur les machines managées : le système d'exploitation, les adresses IP, la mémoire, le disque, etc.
- Ces informations sont stockées dans des variables qu'on nomme facts, et peuvent etre utilisées lors de l'execution du playbook pour personnaliser les configurations.

```yaml
---
- name: Installer des paquets en fonction du système d'exploitation
  hosts: all
  gather_facts: true
  tasks:
    - name: Installer Apache sur Debian/Ubuntu
      ansible.builtin.apt:
        name: apache2
        state: present
      when: ansible_os_family == "Debian"

    - name: Installer Apache sur Red Hat/CentOS
      ansible.builtin.yum:
        name: httpd
        state: present
      when: ansible_os_family == "RedHat"
```

Les facts d'Ansible

Variable	Description
`ansible_hostname`	Nom de l'hôte
`ansible_distribution`	Distribution Linux (ex. Ubuntu, CentOS)
`ansible_distribution_version`	Version de la distribution Linux
`ansible_os_family`	Famille du système d'exploitation (Debian, RedHat, etc.)
`ansible_default_ipv4.address`	Adresse IP principale
`ansible_processor`	Liste des processeurs
`ansible_memtotal_mb`	Mémoire totale en Mo
`ansible_architecture`	Architecture du système (x86_64, etc.)

Template

- Les templates permettent de créer des fichiers de configuration dynamique et réutilisables pour des applications telles que Nginx, Apache et les bases de données en utilisant des variables.
- **Jinja2** est un moteur de templates utilisé dans Ansible pour créer des fichiers de configuration dynamiques en intégrant des variables, des boucles, des conditions et d'autres structures de contrôle directement dans des modèles de texte.

Template

- les variables http_port, hostname et document_root sont utilisées pour générer le fichier de configuration d'Apache.
- La template `apache.j2` est utilisée pour configurer le serveur apache.

apache2.j2

```
# Configuration minimale d'Apache
ServerRoot /etc/apache2
DocumentRoot {{document_root}}
<VirtualHost *:{{ http_port }}>
    ServerName {{ hostname }}
    ServerAlias www.{{ hostname }}
    <Directory />
        AllowOverride None
        Require all granted
    </Directory>
    <Directory "/var/www/html">
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

Playbook: install_apache.yml

```yaml
---
- name: Déployer Apache
  hosts: webservers
  become: true

  vars:
    http_port: 80
    document_root: /var/www/html

  tasks:
    - name: Installer Apache2
      apt:
        name: apache2
        state: present

    - name: Créer le fichier de configuration Apache
      template:
        src: apache2.j2
        dest: /etc/apache2/sites-available/default
      notify: restart apache2

    - name: Activer le site
      service:
        name: apache2
        state: started
        enabled: yes

  handlers:
    - name: redémarrer Apache
      service:
        name: apache2
        state: restarted
```

```
ansible-playbook -i inventory.yaml install_apache.yaml
```

Modules

ANSIBLE

- Un module Ansible est un petit programme écrit dans langage supportant (tel que Python, Perl, Ruby, Bash, etc.) qui exécute des opérations spécifiques sur une machine.
- Ces opérations peuvent aller de l'installation d'un paquet à la configuration d'un service réseau en passant par la gestion d'utilisateurs.
- C'est l'unité de base qui permet à Ansible de réaliser des tâches automatisées sur les serveurs.

Modules

ANSIBLE

- Il existe plus de 1000 modules fournis par Ansible pour automatiser chaque partie de l'environnement, voici des exemples :
  - Modules de gestion de systèmes d'exploitation: apt, yum, service, user, group, etc.
  - Modules de gestion de réseau: ping, setup, route, etc.
  - Modules de gestion de fichiers: copy, file, template, etc.
  - Modules de gestion de conteneurs: docker, kubernetes, etc.
  - Modules spécifiques à des fournisseurs cloud: ec2, azure_rm, gcp, etc.

Modules

ANSIBLE

- Exemples
  - apt : Gère les paquets sur les systèmes basés sur Debian/Ubuntu.
  - service : Gère les services système, tels que le démarrage, l'arrêt et le redémarrage.
  - user : Gère les comptes utilisateurs, notamment la création, la suppression et la modification.… etc.
  - ping : Vérifie la connectivité entre la machine de contrôle et les hôtes cibles.
  - file : Gère les fichiers et les répertoires, y compris les permissions, la propriété et les liens symboliques.
  - copy : Copie des fichiers de la machine de contrôle vers les hôtes distants.

Exemple de playbook avec modules

deploy_apache.yaml

```yaml
- name: Déployer un serveur web Apache
  hosts: web_servers  # Groupe d'hôtes cible (à définir dans l'inventaire)
  become: true        # Exécuter les tâches avec des privilèges administrateur (sudo)
  tasks:
    # 1. Mettre à jour les paquets
    - name: Mettre à jour la liste des paquets
      ansible.builtin.apt:  # Utilisation explicite du module intégré
        update_cache: yes

    # 2. Installer Apache
    - name: Installer Apache
      ansible.builtin.apt:
        name: apache2
        state: present

    # 3. Configurer le fichier d'index
    - name: Remplacer le fichier d'index par défaut
      ansible.builtin.copy:
        dest: /var/www/html/index.html  # Fichier cible sur le serveur
        content: |
          <!DOCTYPE html>
          <html>
          <head>
              <title>Bienvenue sur Apache</title>
          </head>
          <body>

    # 4. Démarrer et activer le service Apache
    - name: Vérifier qu'Apache est démarré et activé
      ansible.builtin.service:
        name: apache2   # Nom du service (sur Ubuntu/Debian, c'est "apache2")
        state: started  # S'assurer que le service est démarré
        enabled: yes    # Activer le service pour démarrage automatique
```

ansible-playbook -i inventory.yaml deploy_apache.yaml

Roles

- Un rôle est une manière de **structurer** et **réutiliser** des tâches d'automatisation.
- C'est un concept qui regroupe l'ensemble des éléments nécessaires pour déployer et configurer un service ou une application spécifique .
- Les Roles facilitent la **réutilisation**, favorisent davantage la **modularisation** de la configuration et **simplifient** l'écriture des Playbooks complexes en le divisant logiquement en **composants réutilisables**.
- Un role est associé à une seul activité par exemple : installation d’un serveur Apache ou configurer une BD.
- Tout rôle peut être exécuté sur n'importe quel hôte ou groupe d'hôtes.

Roles

playbook-example.yaml

- Exemple de role pour installer Apache

Avec role playbook

```yaml
- name: Déployer Apache avec un rôle
  hosts: webservers
  become: yes
  roles:
    - apache
```

Le module apache peut être
en python, C, etc.

```
ansible-playbook -i inventory.yaml deploy_apache.yaml
```

Roles

- Exemple sans role pour installer Apache playbook

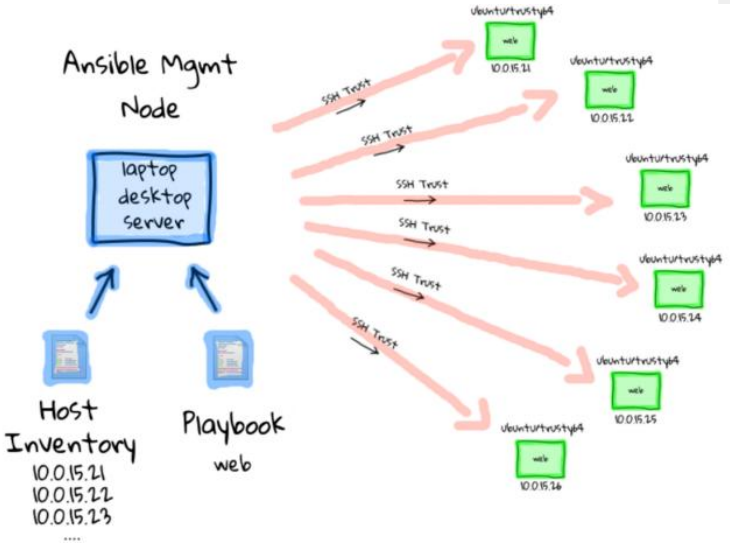
playbook-example.yaml

```yaml
---
- name: Installation et configuration d'Apache
  hosts: webservers
  become: yes

  tasks:
    - name: Installer Apache2
      apt:
        name: apache2
        state: present
        update_cache: yes
      notify:
        - restart apache

    - name: Activer les modules Apache
      loop: "{{ apache_modules }}"
      service:
        name: "{{ item }}"
        state: started
        enabled: yes
      notify:
    - name: Copier la configuration principale
      template:
        src: httpd.conf.j2
        dest: /etc/apache2/apache2.conf
      notify:
        - restart apache

    - name: Créer le répertoire des documents
      file:
        path: /var/www/html
        state: directory

  handlers:
    - name: Redémarrer Apache
      service:
        name: apache2
        state: restarted

  vars:
    apache_modules:
      - mod_ssl
      - mod_rewrite
```

```
ansible-playbook -i inventory.yaml deploy_apache.yaml
```

Fonctionnement d’Ansible

- Ce diagramme illustre le flux d'exécution d'un playbook Ansible :
  - Le node manager Ansible récupère le playbook et l'inventaire.
  - L'inventaire est chargé pour identifier les hôtes à cibler.
  - Pour chaque hôte, il exécute les tâches définies dans le playbook.
  - Les résultats de l'exécution sont retournés au Node manager.



Ansible & cloud

- Ansible peut être utilisé pour gérer l'infrastructure en cloud.
- Il propose plusieurs modules spécialement conçus pour travailler avec différents fournisseurs de cloud.
- Ces modules offrent diverses fonctionnalités, telles que la création d'instances, la fermeture d'instances, l'attachement de volumes de stockage, la gestion de groupes de sécurité, etc.
- Parmi les modules cloud les plus populaires, on peut citer :
  - ec2: Manages Amazon Web Services (AWS) EC2 instances.
  - gcp_compute_instance: Manages Google Cloud Platform (GCP) compute instances.
  - azure_rm_virtualmachine: Manages Microsoft Azure virtual machines.
  - openstack: Manages OpenStack resources.
  - digitalocean_droplet: Manages DigitalOcean droplets.

cloud

deploy_aws_vm.yaml

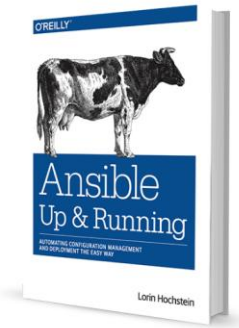
```yaml
---
- name: Déploiement de VM sur AWS
  hosts: localhost
  gather_facts: no
  tasks:
    - name: Créer une instance EC2
      amazon.aws.ec2:
        aws_access_key: "{{ aws_access_key }}"
        aws_secret_key: "{{ aws_secret_key }}"
        key_name: "{{ key_name }}"              # Nom de la clé SSH
        instance_type: "t2.micro"               # Type d'instance (t2.micro = gratuit)
        image_id: "{{ ami_id }}"                # AMI ID (Ubuntu/Debian/RedHat)
        region: "us-east-1"                     # Région AWS (ex: us-east-1)
        group: "{{ security_group }}"           # Groupe de sécurité
        count: 1                                # Nombre d'instances
        wait: yes                               # Attendre la création de l'instance
        instance_tags:
          Name: "Ansible-Test-Instance"         # Tag de l'instance
        assign_public_ip: yes                   # Attribuer une IP publique
      register: ec2_instances                   # variable qui contient des détails sur les
                                                #instances créées.

    - name: Afficher les informations sur les instances
      debug:
        var: ec2_instances
```

vars/main.yaml

```yaml
aws_access_key: "your_aws_access_key"
aws_secret_key: "your_aws_secret_key"
key_name: "your_ssh_key_name"
ami_id: "ami-0c02fb55956c7d316"        # AMI Ubuntu 20.04 (exemple pour us-east-1)
security_group: "your_security_group_name"
```

```
ansible-playbook -i localhost, deploy_aws_vm.yaml
```

Les avantages d'Ansible



SIMPLE

Ansible utilise un langage de description de configuration (YAML) très lisible, ce qui le rend accessible même aux débutants.

COMMUNAUTE

Il bénéficie d'une communauté active et de nombreux modules créés par la communauté. Top 10 open source projects in 2017,

AGENTLESS

Il ne nécessite pas d'agent logiciel à installer sur les machines à gérer, ce qui simplifie la complexité de déploiement.

POWERFUL

Il offre une large gamme de modules pour automatiser une multitude de tâches, de l'installation de paquets à la configuration de services complexes.

Idempotents

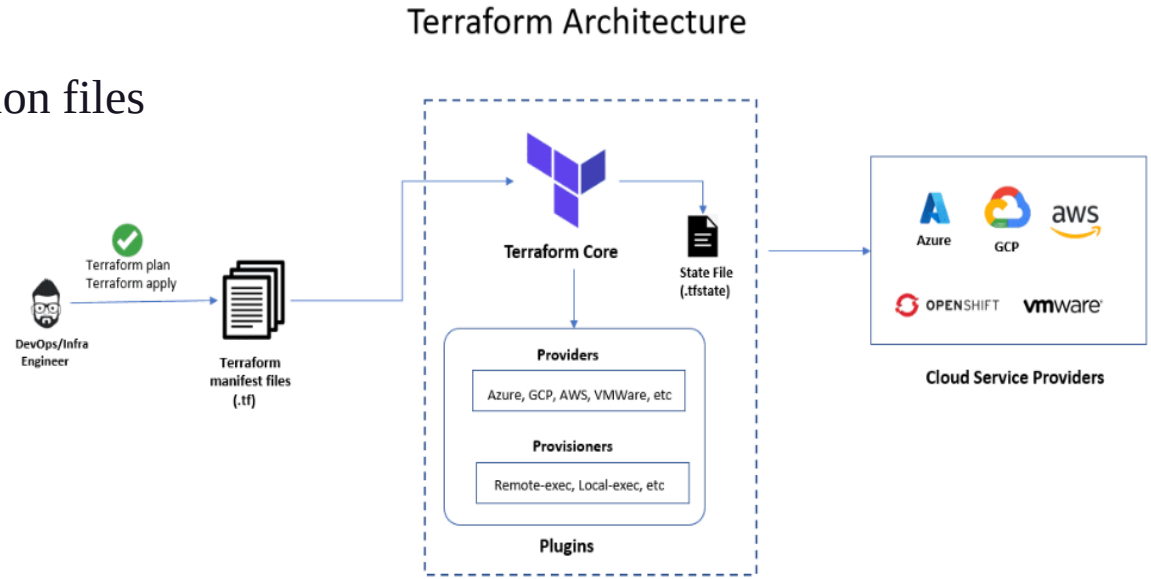
Les playbooks Ansible sont idempotents, ce qui signifie qu'ils peuvent être exécutés plusieurs fois sans modifier l'état du système s'il est déjà dans l'état souhaité.

Terraform

- Terraform est un outil Infrastructure as code (IaC), open source, développé en 2014 par HashiCorp, une entreprise spécialisée dans les outils pour les infrastructures distribuées.
- Il permet de décrire, provisionner et gérer des infrastructures cloud ou sur site de manière programmable avec un langage déclaratif (HCL, Hashicorp Configuration Language).
- Il permet aux développeurs et aux DevOps de créer et de gérer facilement des ressources informatiques telles que des serveurs, des réseaux et des bases de données dans différents fournisseurs de cloud.
- Souvent appelé Infrastructure as Code.

Composant de Terraform

- L'architecture de Terraform se compose principalement des éléments suivants :
  - Terraform Core
  - Provider plugins
  - State file
  - Configuration files



Terraform Core

- Le Terraform Core est le moteur principal qui orchestre l'ensemble des opérations.
- Il gère les interactions entre la configuration, les plugins (providers), et l'état.
- Ses responsabilités incluent :
  - Analyser les fichiers de configuration pour comprendre l'état souhaité.Gérer l'état de l'infrastructure (terraform.tfstate).
  - Créer un plan d'exécution (différence entre état actuel et souhaité).
  - Appliquer les modifications nécessaires (via terraform apply).

Providers Terraform

- Un provider Terraform est un composant essentiel qui agit comme un plugin pour interagir avec des services spécifiques, comme des fournisseurs cloud (AWS, Azure, GCP), des plateformes SaaS, ou des infrastructures locales.
- Les providers permettent à Terraform de créer, lire, mettre à jour et supprimer des ressources sur ces services en utilisant leurs API respectives.
- Il traduit les configurations Terraform en requêtes API pour interagir avec des plateformes externes.
- Il agit comme un traducteur entre le langage de configuration de Terraform et le langage spécifique de chaque service.

Providers Terraform

- Les providers interagissent directement avec les API des plateformes cloud pour créer, modifier ou supprimer des ressources.
- Chaque provider expose un ensemble de ressources spécifiques à la plateforme qu'il supporte.
- Les ressources dans Terraform sont des composants de l’infrastructure, tels que des machines virtuelles, des réseaux, ou des bases de données.
- Exemples de providers courants :
    - AWS : Gère les ressources comme les instances EC2, S3, etc.
    - Google Cloud : Gère les ressources GCP comme Compute Engine et Cloud Storage.
    - AzureRM : Gère les ressources Azure.
    - Outscale : Gère les ressources dans le cloud Outscale, comme les machines virtuelles et les réseaux.

Le state de Terraform

- Les états Terraform représente une mémoire pour l’infrastructure
- L'état Terraform, ou state, est un fichier JSON qui enregistre l'état actuel de l’infrastructure gérée par Terraform.
- Par défaut, l'état Terraform est stocké dans un fichier nommé terraform.tfstate
- Il contient des informations cruciales telles que :
    - Les ID des ressources: Chaque ressource créée par Terraform (instance EC2, réseau VPC, etc.) possède un ID unique qui est enregistré dans l'état.
    - Les dépendances entre les ressources: Terraform utilise l'état pour comprendre les relations entre les différentes ressources et ainsi gérer les mises à jour de manière cohérente.
    - La configuration actuelle: L'état contient une version de la configuration Terraform au moment de la dernière application.

Les fichiers de configuation

- Ce sont du code écrit en langage de configuration HashiCorp (HCL)
- C’est un langage déclaratif pour décrire l'état souhaité des ressources de l'infrastructure.
- HCL utilise une syntaxe basée sur des clés-valeurs
- Pour un projet très simple, on peut utiliser seulement le fichier main.tf, sinon on peut adopter une structure qui favorise la réutilisation
- Ces fichier permettent de définir :
  - Les ressources à gérer (création, modification, suppression).
  - Les propriétés de ces ressources, telles que les tailles, emplacements, types, etc.
  - Les dépendances entre les ressources pour garantir un déploiement ordonné.

Les fichiers de configurations

- Structure d’un projet complexe :
  - main.tf:
    - ✓ C'est le fichier principal où sont définies les configurations de base de l'infrastructure à provisionner
    - ✓ il contient les ressources, les modules, et les providers nécessaires pour déployer l'infrastructure.
    - ✓ Il permet la configuration des dépendances entre les ressources.
    - ✓ Par fois appel à des modules Terraform.

Les fichiers de configuration

- Structure d’un projet complexe :
  - variables.tf: utilisé pour définir les variables qui seront remplacées par des valeurs spécifiques lors de l'exécution de Terraform.
  - Outputs.tf: Utilisées pour exporter des valeurs à partir de la configuration, ces valeurs peuvent être utilisées dans d'autres parties de l’ infrastructure ou dans d'autres outils.
  - Provider.tf: définissent les fournisseurs de cloud et les ressources disponibles.
  - Resources.tf: définissent les ressources disponibles.

```
variable "ami_id" {
description = "ID de l'image AMI pour l'instance EC2"
type = string
default = "ami-0c02fb55956c7d316" # AMI Amazon
Linux 2 (us-east-1)
}

variable "instance_type" {
description = "Type de l'instance EC2"
type = string
default = "t2.micro"
}

variable "key_name" {
description = "Nom de la clé SSH pour accéder à
l'instance"
type = string
default = "my-ssh-key" # Remplacez par le nom de votre
clé existante
}
```

Les modules

- Comme dans ansible, les modules sont définis dans Terraform pour organiser et réutiliser des configurations.
- Leur utilisation permet de structurer le code, de le rendre modulaire et réutilisable, ce qui est particulièrement utile dans des infrastructures complexes.

```
modules/ec2
├── main.tf
├── variables.tf
├── outputs.tf
└── README.md
```

```
# modules/ec2/main.tf
resource "aws_instance" "web_server" {
 ami = var.ami
 instance_type = var.instance_type
 key_name = var.key_name
 tags = var.tags
 }
 output "public_ip" {
 value = aws_instance.example.public_ip
 }
```

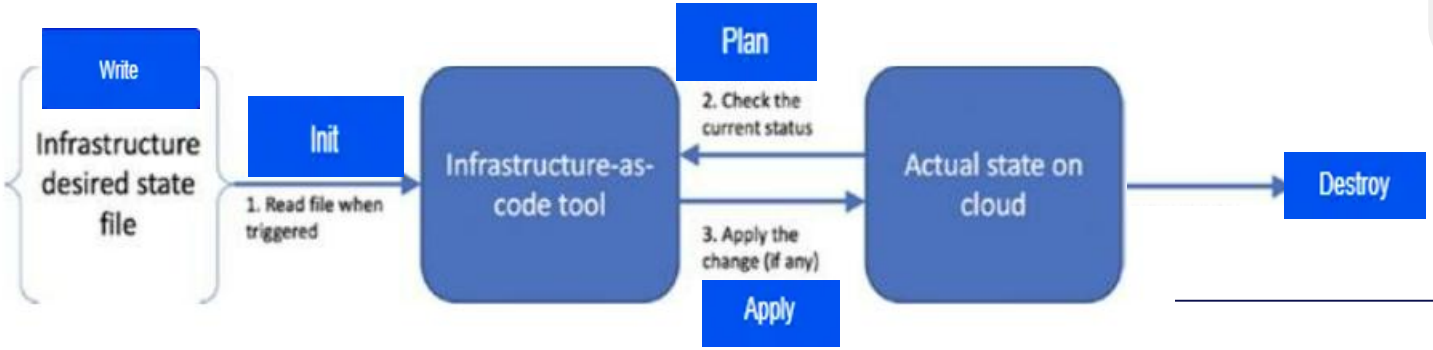
```
module "ec2_instance" {
  source        = "./modules/aws-ec2-instance"
  instance_type = "t2.micro"
  ami_id        = var.ami_id
  key_name      = var.key_name
  tags = {
    Name = "My-EC2-Instance"
    Env  = "Development"
  }
}
```

Data source de Terraform

- Une data source est un outil qui permet à Terraform de récupérer des informations sur des ressources existantes et l'état actuel de l'infrastructure.
- Utilités des Data Sources :
  - Récupération dynamique : Obtenir des informations comme l'ID d'un groupe de sécurité ou l'adresse IP d'une ressource.
  - Réutilisation de ressources existantes et évite la duplication des données.
  - Gestion des dépendances : Facilite la création de nouvelles ressources en fonction de celles déjà existantes (ex. : récupérer un ID de sous-réseau avant de créer une instance EC2).
- Les data sources rendent les configurations Terraform plus dynamiques et flexibles

Le workflow de Terraform

- Le workflow de Terraform reposent sur cinq étapes clés : Write, Init, Plan, Apply et Destroy et sur l'enregistrement du state:
  - Write: écrire la configuration.
  - Init: initialise la configuration pour installer les dépendances nécessaires.
  - Plan: scan l'infrastructure existante et la compare en termes de codes on parle de miroir de l'infra et le code.
  - Apply: applique les changements à l'infrastructure réelle.
  - Destroy: libère toute l'infrastructure



Exemple de fichier de configuration

main.tf

```
# Spécifie le Fournisseur AWS,
# la région où les ressources AWS seront créées
provider "aws" {
 region = "us-west-2"
}
#définit une data source pour récupérer
#des informations sur un sous-réseau public
data "aws_subnet" "public" {
  vpc_id = "vpc-12345678"
  cidr_blocks = ["10.0.1.0/24"]
}
# Ressource : Instance EC2
resource "aws_instance" "web_server" {
  ami           = "ami-08d4ac5b634553e16"  #ubuntu 22. 04 LTS
  instance_type = "t2.micro"
  subnet_id     = data.aws_subnet.public.id
}
#l'adresse publique de l'instance EC2 sera
#attribuée automatiquement par AWS lors du déploiement
output "instance_ip" {
 value = aws_instance.web_server.public_ip
}
```

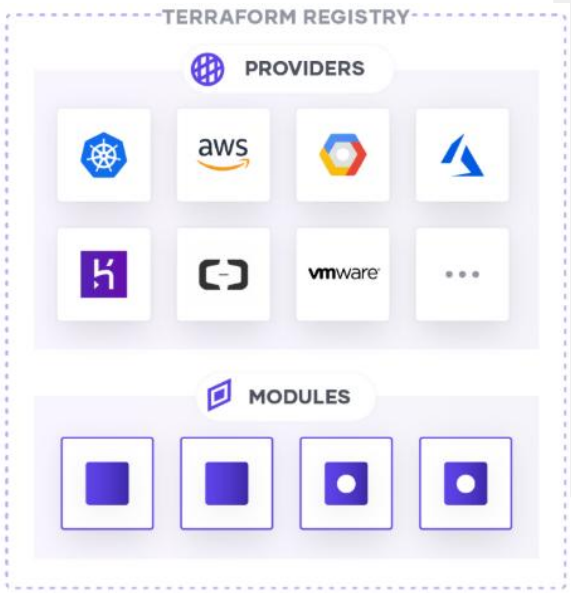
Commandes CLI :

```
terraform init
Terraform plan
terraform apply
```

Type d'instance EC2 t2.micro
vCPU : 1 unité de CPU virtuel.
RAM : 1 Go de RAM.
Stockage : Ne comprend pas de stockage par défaut, mais peut utiliser des volumes EBS (Elastic Block Store).

Terraform Registry

- La Terraform Registry est une plateforme en ligne qui centralise et met à disposition une vaste collection de modules Terraform créés par la communauté ou par des entreprises.
- Ces modules sont des ensembles de configurations Terraform pré-écrits, conçus pour automatiser la création et la gestion de ressources cloud spécifiques.
- Si on veut déployer un cluster Kubernetes sur AWS, on recherche un module Kubernetes dans la Terraform Registry et on l’utiliser directement dans la configuration au lieu d'écrire tout le code manuellement.



Avantanges de Terraform

Automatisation

permet d'automatiser le provisionnement et la gestion de l'infrastructure, réduisant ainsi les erreurs humaines et accélérant les déploiements.

Scalabilité

permet de gérer l'infrastructure à grande échelle en ajoutant ou réduisant rapidement des ressources en modifiant simplement le code d'infrastructure.

contrôle des changements

facilite le suivi des modifications apportées à l'infrastructure au fil du temps, permettant ainsi un contrôle de version.

Flexibilité et portabilité

Migrer entre différents fournisseurs de cloud ou tester différentes configurations sans engagement à long terme.

Synthèse des outils DevOps

Outil	Définition	Objectif principal	Cas d'utilisation	Avantages	Inconvénients
Docker	Outil de conteneurisation permettant d'exécuter des applications dans des environnements isolés (conteneurs).	Fournir des conteneurs légers et portables.	Déploiement d'applications isolées, tests dans des environnements uniformes, microservices.	- Léger et rapide - Portabilité multiplateforme - Large écosystème d'images sur Docker Hub - Idéal pour les microservices.	- Nécessite une orchestration pour les environnements complexes - Moins adapté pour les systèmes état-centrés (stateful).
Kubernetes	Système d'orchestration de conteneurs open source qui automatise le déploiement, la mise à l'échelle et la gestion des conteneurs.	Orchestrer des conteneurs à grande échelle.	Gestion d'applications distribuées, scalabilité automatique, haute disponibilité.	- Scalabilité automatique - Gestion simplifiée des déploiements complexes - Écosystème riche (Ingress, volumes, secrets, etc.).	- Complexité initiale élevée - Peut être surdimensionné pour les petites applications.
Jenkins	Serveur d'intégration et de livraison continues (CI/CD) open source.	Automatiser les pipelines CI/CD.	Intégration continue, déploiement continu, tests automatisés, builds d'applications.	- Hautement extensible avec des plugins - Compatible avec de nombreux outils et technologies - Open source et gratuit.	- Gestion des plugins peut devenir complexe - Interface utilisateur moins intuitive comparée à des solutions modernes comme GitHub Actions ou GitLab CI.
Ansible	Outil d'automatisation de la gestion de configuration et d'orchestration basé sur YAML.	Automatiser les configurations et les déploiements.	Configuration des serveurs, déploiement d'applications, provisionnement d'infrastructures.	- Simple à apprendre grâce à YAML - Agentless (pas besoin d'installer un agent sur les cibles) - Large communauté et extensibilité.	- Moins performant pour les tâches très complexes ou nécessitant une orchestration avancée - Pas idéal pour la gestion d'infrastructure "as code".
Terraform	Outil d'infrastructure en tant que code (IaC) permettant de gérer des infrastructures de manière déclarative.	Provisionner et gérer des infrastructures.	Création et gestion des ressources cloud (AWS, Azure, GCP), infrastructures hybrides, gestion de réseaux complexes.	- Gestion déclarative des ressources - Compatible avec plusieurs fournisseurs (multi-cloud) - Gestion d'état avec des fichiers .tfstate.	- Courbe d'apprentissage plus élevée pour les débutants - Gestion d'état centralisée nécessite des configurations supplémentaires pour les équipes.

Synthèse des outils Cloud

Outil	Définition	Objectif principal	Cas d'utilisation	Avantages	Inconvénients
ESXi	Hyperviseur bare-metal développé par VMware qui permet de virtualiser des serveurs physiques.	Virtualisation d'infrastructure physique.	.Héberger plusieurs machines virtuelles (VM) sur un seul serveur physique. .	- Performant et stable. - Administration via vSphere.- Support pour de nombreux systèmes d'exploitation	- Payant pour les fonctionnalités avancées. - Courbe d'apprentissage initiale.
vSphere Client	Interface graphique pour gérer les hôtes ESXi et leurs machines virtuelles.	Administrer et superviser les environnements ESXi.	Gestion des VM, surveillance des ressources, migration de VM (vMotion).	- Interface conviviale. - Accès centralisé à la configuration et à la supervision des VM.	- Dépendance à VMware pour la plateforme.
Daylight	Plateforme SDN open source basée sur OpenFlow qui permet de gérer les réseaux programmables.	Automatiser et contrôler les réseaux via SDN.	Contrôleurs SDN dans les centres de données, gestion réseau automatisée, optimisation de trafic.	- Plateforme modulaire et extensible. - Supporte OpenFlow et d'autres protocoles.	- Configuration complexe. - Nécessite des compétences avancées en SDN.
Mininet	Simulateur de réseau SDN (Software Defined Networking) permettant de créer et tester des topologies réseau.	Simuler des environnements réseau pour la recherche et le développement.	Tester des architectures SDN, vérifier des protocoles réseau, former des étudiants.	- Léger et facile à utiliser. - Compatible avec OpenFlow.	- Limité aux tests. - Moins performant pour des réseaux complexes ou réels.
RAID	Technologie combinant plusieurs disques physiques en une seule unité logique pour redondance ou performance.	Améliorer la redondance ou les performances des systèmes de stockage.	Serveurs, NAS, systèmes critiques nécessitant une haute disponibilité ou des performances accrues.	- Sécurité accrue (RAID 1, 5, 6). - Performances augmentées (RAID 0, 10).	- Coût supplémentaire en matériel. - Certains niveaux (ex. RAID 0) n'offrent pas de redondance.
RDS (Bureau à Distance)	Technologie Microsoft permettant l'accès à distance aux bureaux Windows via le protocole RDP.	Fournir un accès distant aux bureaux et applications Windows.	Accéder aux postes de travail ou serveurs depuis n'importe quel appareil connecté.	- Accès distant simplifié. - Multi-utilisateur sur serveur.	- Nécessite une bonne configuration réseau. - Peut être lent sur des connexions à faible débit.

Fin du module