Automatisation de la Création et de la Gestion de Serveurs par Scripting

Table des matières

Module 1 : Introduction au scripting

Objectif du module :

L'objectif de ce module est de fournir une introduction complète au scripting, en mettant l'accent sur son importance et ses applications dans l'automatisation des tâches administratives. Ce module est conçu pour que les étudiants comprennent le rôle du scripting dans la gestion des serveurs et des systèmes d'exploitation.

Contenu du module :

1.1 Qu'est-ce que le scripting ?

Le scripting est l'art d'écrire des programmes informatiques simples, appelés scripts, qui automatisent des tâches répétitives ou complexes. Ces scripts peuvent être exécutés sur des systèmes d'exploitation pour exécuter des commandes, manipuler des fichiers, gérer des utilisateurs, et bien plus encore. Le scripting est essentiel dans les environnements de serveurs pour réduire la charge de travail manuelle et garantir la cohérence des configurations.

Exemple concret :

Supposons que vous ayez plusieurs serveurs à mettre à jour avec les dernières versions de logiciels et que vous deviez exécuter la même série de commandes sur chacun d'eux. Un script permettrait d'automatiser cette tâche, vous permettant de mettre à jour tous les serveurs en un seul clic, plutôt que de devoir le faire manuellement sur chaque serveur.

1.2 Pourquoi utiliser le scripting ?

L'utilisation de scripts présente de nombreux avantages :

- **Automatisation des tâches répétitives :** Le scripting permet d'automatiser des tâches telles que la gestion des fichiers, l'installation de logiciels, la configuration de services ou la mise à jour de systèmes.
- **Réduction des erreurs humaines :** L'automatisation élimine les risques d'erreurs humaines qui peuvent survenir lors de l'exécution manuelle des commandes.
- **Gain de temps :** Les scripts permettent de réaliser des tâches complexes en quelques secondes ou minutes, contrairement aux méthodes manuelles.
- **Cohérence et standardisation :** Le scripting garantit que les mêmes actions seront exécutées de manière cohérente sur tous les systèmes ou serveurs.

Exemples d'utilisation du scripting :

- Déploiement d'applications sur plusieurs serveurs.
- Création de sauvegardes régulières.
- Surveillance de l'état des services et envoi de rapports par email.

1.3 Types de scripts et langages utilisés

Il existe plusieurs langages de script, chacun ayant ses propres avantages selon l'environnement et les besoins :

- **Shell Bash (Linux) :** Utilisé principalement sous Linux, Bash est un langage puissant pour l'automatisation des tâches système.
- **PowerShell (Windows) :** Un langage de script utilisé pour l'administration des systèmes Windows, offrant une riche fonctionnalité pour la gestion des systèmes et des applications.
- **Python :** Un langage polyvalent, très utilisé pour l'automatisation, l'administration réseau, et la gestion de serveurs grâce à sa syntaxe simple et ses bibliothèques étendues.
- **Perl :** Un autre langage populaire pour l'automatisation, notamment dans les scripts de traitement de texte et de gestion de fichiers.

Exemples de langages de script :

- **Bash :** Utilisé sur Linux pour des tâches comme la gestion des utilisateurs, la configuration des réseaux, ou l'automatisation de processus système.
- **PowerShell :** Utilisé pour automatiser l'administration de systèmes Windows, comme l'installation de logiciels, la gestion des utilisateurs, et la configuration des services.

1.4 Structure d'un script

Un script est essentiellement une séquence d'instructions exécutées par le système d'exploitation ou l'interpréteur de commande. Un script typique contient :

- **En-tête (optionnel) :** Certaines langues, comme Bash, incluent un en-tête qui spécifie l'interpréteur à utiliser, par exemple : `#!/bin/bash`.
- **Commandes :** Ce sont les instructions que le script exécutera. Elles peuvent être des commandes système, des boucles, des conditions, etc.
- **Commentaires :** Les commentaires expliquent le code et ne sont pas exécutés. Ils permettent à l'auteur du script de documenter son travail.

Exemple d'un script simple en Bash :

```
#!/bin/bash
# Ce script affiche "Hello, World!" à l'écran
echo "Hello, World!"
```

- **Ligne 1** : L'en-tête qui spécifie que ce script doit être exécuté avec l'interpréteur Bash.
- **Ligne 2** : Un commentaire qui explique ce que fait le script.
- **Ligne 3** : La commande `echo` qui affiche le texte "Hello, World!" dans le terminal.

1.5 Les commandes de base du scripting

Les scripts utilisent des commandes du système d'exploitation pour accomplir des tâches. Ces commandes peuvent être combinées pour créer des processus plus complexes.

Commandes courantes sous Linux (Bash) :

- **echo** : Affiche un message à l'écran.
- **ls** : Liste les fichiers et répertoires dans un répertoire.
- **cd** : Change le répertoire de travail.
- **cp** : Copie des fichiers ou répertoires.
- **mv** : Déplace ou renomme des fichiers.
- **rm** : Supprime des fichiers ou répertoires.

Exemple avec la commande `echo` :

```
#!/bin/bash
echo "Bienvenue dans le monde du scripting !"
```

Ce script affiche un message dans le terminal.

1.6 Variables et opérateurs

Les variables sont utilisées pour stocker des informations qui peuvent être utilisées ultérieurement dans un script. Les opérateurs permettent d'effectuer des opérations sur les données.

Exemples :

- **Définir une variable** :

```
#!/bin/bash
nom="Alice"
echo "Bonjour, $nom!"
```

Ici, la variable `nom` stocke la valeur "Alice", et le script affiche "Bonjour, Alice!".

- **Opérateurs mathématiques** :

```
#!/bin/bash
a=10
b=5
result=$((a + b))
echo "La somme de $a et $b est $result."
```

1.7 Structures conditionnelles et boucles

Les scripts peuvent contenir des structures de contrôle pour exécuter certaines parties du code en fonction de conditions (if, else) ou pour répéter des actions (boucles).

Exemple avec `if` et `else` :

```
#!/bin/bash
# Vérifier si un fichier existe
if [ -f "mon_fichier.txt" ]; then
    echo "Le fichier existe."
else
    echo "Le fichier n'existe pas."
fi
```

- Ce script vérifie si un fichier nommé "mon_fichier.txt" existe et affiche un message en conséquence.

Exemple de boucle `for` :

```
#!/bin/bash
# Afficher les nombres de 1 à 5
for i in {1..5}
do
    echo "Nombre $i"
done
```

1.8 Exécution et débogage d'un script

Une fois que le script est écrit, il doit être exécuté. Cela se fait généralement en rendant le fichier exécutable avec la commande `chmod`, puis en l'exécutant directement.

Exécution d'un script sous Linux :

1. Rendre le script exécutable :

```
chmod +x mon_script.sh
```

2. Exécuter le script :

```
./mon_script.sh
```

Le débogage des scripts peut inclure l'ajout de messages de débogage ou l'utilisation de commandes comme `set -x` pour afficher chaque commande exécutée.

1.9 Pratiques et bonnes habitudes

Voici quelques bonnes pratiques à suivre lors de l'écriture de scripts :

- **Utiliser des commentaires** pour expliquer le code.
- **Respecter les conventions de nommage** (par exemple, utiliser des noms de variables explicites).
- **Tester les scripts dans des environnements contrôlés** avant de les déployer sur des serveurs de production.

Exercice 1 : Créer un répertoire avec un nom donné en argument et afficher un message de succès ou d'échec

Solution du script Bash

1. **Créer un fichier script** : Utilisez un éditeur de texte pour créer un fichier de script, par exemple `create_directory.sh`.

bash Copier Modifier

```
nano create_directory.sh
```

2. **Écrire le script** : Dans ce script, nous allons :

- Vérifier si un argument a été passé (le nom du répertoire).
- Créer le répertoire avec `mkdir`.
- Afficher un message de succès ou d'échec en fonction du résultat.

Voici le code du script :

bash Copier Modifier

```bash
#!/bin/bash

# Vérifier si un argument a été passé
if [ -z "$1" ]; then
  echo "Erreur : Aucun nom de répertoire spécifié."
  exit 1
fi

# Créer le répertoire
mkdir "$1"

# Vérifier si le répertoire a été créé avec succès
if [ $? -eq 0 ]; then
  echo "Le répertoire '$1' a été créé avec succès."
else
  echo "Erreur lors de la création du répertoire '$1'."
fi
```

Explication :

- `if [ -z "$1" ]` : Vérifie si le premier argument est vide (aucun nom de répertoire spécifié).
- `mkdir "$1"` : Crée le répertoire avec le nom passé en argument.
- `if [ $? -eq 0 ]` : Vérifie le code de sortie de la commande précédente. Un code de sortie `0` indique un succès, alors qu'un code différent de `0` indique une erreur.

Exercice 2 : Modifier le script pour afficher la date et l'heure actuelles dans un fichier texte

Solution du script Bash modifié

1. **Créer ou modifier un fichier script** : Vous pouvez modifier le script `create_directory.sh` pour qu'il ajoute la date et l'heure actuelles dans un fichier texte.

Voici le code modifié du script :

bash Copier Modifier

```bash
#!/bin/bash

# Vérifier si un argument a été passé
if [ -z "$1" ]; then
  echo "Erreur : Aucun nom de répertoire spécifié."
  exit 1
fi

# Créer le répertoire
mkdir "$1"

# Vérifier si le répertoire a été créé avec succès
if [ $? -eq 0 ]; then
  echo "Le répertoire '$1' a été créé avec succès."
else
  echo "Erreur lors de la création du répertoire '$1'."
fi

# Ajouter la date et l'heure actuelles dans un fichier texte
date '+%Y-%m-%d %H:%M:%S' > date.txt
echo "La date et l'heure actuelles ont été enregistrées dans 'date.txt'."
```

Explication :

- Après avoir créé le répertoire, le script ajoute la date et l'heure actuelles dans le fichier `date.txt`.
- La commande `date '+%Y-%m-%d %H:%M:%S'` formate la date et l'heure dans le format `YYYY-MM-DD HH:MM:SS` et redirige la sortie vers le fichier `date.txt`.

Module 2 : La connaissance des principaux systèmes d'exploitation : Linux

Objectif du module :

Ce module a pour objectif d'introduire et de détailler les principales caractéristiques du système d'exploitation Linux, de ses différentes distributions à sa gestion quotidienne. Les administrateurs de serveurs doivent avoir une maîtrise complète de Linux, car c'est l'un des systèmes les plus utilisés dans les environnements de serveurs. Le module couvrira les bases de Linux, la gestion des utilisateurs, les services systèmes, et la gestion des fichiers.

2.1 Introduction à Linux

Linux est un système d'exploitation de type Unix qui est largement utilisé dans les environnements serveur, en particulier pour l'hébergement Web, les serveurs de bases de données, et les systèmes embarqués. C'est un logiciel libre, ce qui signifie que son code source est accessible à tous et peut être modifié.

Caractéristiques principales :

- **Open-source** : Le code source est disponible et modifiable par la communauté.
- **Multitâche et multi-utilisateur** : Plusieurs utilisateurs peuvent utiliser le système en même temps sans interférer les uns avec les autres.
- **Sécurisé** : Grâce à ses permissions et son architecture, Linux est considéré comme l'un des systèmes d'exploitation les plus sécurisés.
- **Personnalisable** : Linux peut être configuré et adapté à des besoins spécifiques en fonction des distributions et des paquets installés.

2.2 Les principales distributions Linux

Linux existe sous différentes distributions (ou "distros"), chacune ayant des particularités et répondant à des besoins spécifiques. Les plus populaires sont :

Distributions orientées serveur :

- **Ubuntu Server** : Très populaire et facile à utiliser, avec un vaste support communautaire.
- **CentOS** : Une version libre de Red Hat Enterprise Linux (RHEL), utilisée dans des environnements professionnels.
- **Debian** : Connue pour sa stabilité, souvent utilisée dans des environnements de production.
- **Fedora** : Utilisée principalement par les développeurs, elle offre les dernières versions des logiciels.
- **Arch Linux** : Très flexible, elle est souvent utilisée par des utilisateurs avancés qui souhaitent tout contrôler.

Distributions orientées bureau :

- **Ubuntu Desktop** : Un choix populaire pour les utilisateurs qui souhaitent une interface graphique simple et un environnement utilisateur convivial.
- **Linux Mint** : Basée sur Ubuntu, mais avec un bureau plus traditionnel et une interface utilisateur facile à comprendre.

2.3 Gestion des utilisateurs et des permissions sous Linux

La gestion des utilisateurs et des permissions est un aspect fondamental de l'administration des systèmes Linux. Elle permet de sécuriser l'accès aux ressources du système.

Gestion des utilisateurs :

- **Création d'un utilisateur :**

  ```
  sudo adduser nom_utilisateur
  ```

  Cette commande crée un nouvel utilisateur et lui attribue un répertoire personnel.

- **Modification des informations d'un utilisateur :**

  ```
  sudo usermod -aG groupe nom_utilisateur
  ```

  Cette commande ajoute un utilisateur à un groupe spécifique.

- **Suppression d'un utilisateur :**

  ```
  sudo deluser nom_utilisateur
  ```

Gestion des permissions :

- **Changement de propriétaire et de groupe :**

  ```
  sudo chown utilisateur:group fichier
  ```

  Cela modifie le propriétaire et le groupe associés à un fichier.

- **Modification des permissions :**

  ```
  sudo chmod 755 fichier
  ```

  Cela accorde des permissions de lecture, écriture, et exécution pour le propriétaire, et des permissions de lecture et exécution pour les autres utilisateurs.

- **Vérification des permissions :**

  ```
  ls -l fichier
  ```

2.4 Les services systèmes sous Linux

Les services sous Linux (aussi appelés **daemons**) sont des programmes qui s'exécutent en arrière-plan pour accomplir des tâches système comme la gestion des utilisateurs, les serveurs Web, ou les services réseau. Linux utilise principalement **systemd** pour gérer ces services.

Principaux services sous Linux :

- **Apache HTTP Server (httpd)** : Serveur Web populaire.
- **MySQL/MariaDB** : Systèmes de gestion de bases de données.

- **SSH (Secure Shell)** : Permet la connexion distante au système.
- **Nginx** : Un autre serveur Web, souvent préféré pour sa légèreté.
- **Samba** : Permet de partager des fichiers et des imprimantes avec des systèmes Windows.

Commandes pour gérer les services :

- **Vérifier le statut d'un service :**

```
systemctl status apache2
```

- **Démarrer un service :**

```
sudo systemctl start apache2
```

- **Arrêter un service :**

```
sudo systemctl stop apache2
```

- **Activer un service au démarrage :**

```
sudo systemctl enable apache2
```

- **Redémarrer un service :**

```
sudo systemctl restart apache2
```

2.5 La gestion des fichiers sous Linux

Linux est un système de fichiers hiérarchique, où tous les fichiers et répertoires sont organisés sous un seul répertoire racine (`/`).

Principaux répertoires sous Linux :

- **`/bin`** : Contient les exécutables essentiels du système.
- **`/etc`** : Contient les fichiers de configuration.
- **`/home`** : Contient les répertoires personnels des utilisateurs.
- **`/var`** : Contient les fichiers de données variables comme les logs ou les bases de données.
- **`/tmp`** : Contient les fichiers temporaires.
- **`/usr`** : Contient les applications et bibliothèques de l'utilisateur.

Commandes courantes pour la gestion des fichiers :

- **Lister les fichiers d'un répertoire :**

```
ls /etc
```

- **Copier un fichier :**

```
cp fichier.txt /home/utilisateur/
```

- **Déplacer ou renommer un fichier :**

```
mv fichier.txt /home/utilisateur/nouveau_nom.txt
```

- **Supprimer un fichier :**

  ```
  rm fichier.txt
  ```

- **Créer un répertoire :**

  ```
  mkdir mon_repertoire
  ```

- **Afficher le contenu d'un fichier texte :**

  ```
  cat fichier.txt
  ```

2.6 Les journaux et l'audit système sous Linux

Les journaux systèmes sont cruciaux pour diagnostiquer les problèmes et auditer l'activité du serveur. Ces journaux sont souvent stockés dans le répertoire `/var/log`.

Commandes utiles pour les journaux :

- **Afficher les derniers journaux du système :**

  ```
  sudo tail -f /var/log/syslog
  ```

- **Vérifier les erreurs d'authentification :**

  ```
  sudo tail -f /var/log/auth.log
  ```

Utilisation de `journalctl` (pour systemd) :

- **Afficher les journaux du système :**

  ```
  sudo journalctl
  ```

- **Filtrer les journaux par service :**

  ```
  sudo journalctl -u apache2
  ```

2.7 La gestion des processus sous Linux

Linux est un système multitâche, ce qui signifie qu'il peut exécuter plusieurs processus en même temps. La gestion des processus est cruciale pour assurer des performances optimales.

Commandes pour gérer les processus :

- **Afficher les processus en cours :**

  ```
  ps aux
  ```

- **Utiliser `top` pour surveiller les processus en temps réel :**

  ```
  top
  ```

- **Arrêter un processus (grâce à son PID) :**

  ```
  kill PID
  ```

2.8 Sécurité et firewall sous Linux

La sécurité est une priorité dans la gestion des serveurs Linux. Il existe plusieurs moyens de sécuriser un serveur, y compris l'utilisation de **firewall** et la gestion des permissions.

Utilisation de `ufw` (Uncomplicated Firewall) pour gérer le firewall :

- **Activer `ufw` :**

  ```
  sudo ufw enable
  ```

- **Autoriser un port spécifique (ex. SSH sur le port 22) :**

  ```
  sudo ufw allow 22
  ```

- **Vérifier l'état du firewall :**

  ```
  sudo ufw status
  ```

Module 3 : Les différentes versions de Windows Server et leurs fonctionnalités

Objectif du module :

Ce module vise à fournir une compréhension approfondie des différentes versions de Windows Server, leurs fonctionnalités spécifiques et comment elles sont utilisées dans un environnement professionnel. Cela inclut les versions principales de Windows Server, ainsi que leurs améliorations et particularités au fil des années. L'objectif est que l'apprenant soit capable de différencier chaque version et de savoir quelle version utiliser en fonction des besoins d'une infrastructure serveur.

3.1 Introduction à Windows Server

Windows Server est une série de systèmes d'exploitation développée par Microsoft pour être utilisée sur des serveurs. Il offre une plateforme robuste pour exécuter des applications critiques, gérer des réseaux et fournir des services aux utilisateurs et aux ordinateurs connectés. Windows Server se distingue par sa facilité d'intégration avec d'autres produits Microsoft et son interface graphique conviviale.

3.2 Les principales versions de Windows Server

Depuis son lancement en 1993, Windows Server a évolué à travers plusieurs versions, chaque version apportant des fonctionnalités nouvelles, une meilleure sécurité et une plus grande stabilité. Voici un aperçu des versions les plus importantes :

3.2.1 Windows Server 2003

- **Date de lancement :** 2003
- **Principales caractéristiques :**
  - **Active Directory (AD)** : Une des fonctionnalités les plus importantes de Windows Server 2003, permettant de centraliser la gestion des utilisateurs et des ordinateurs dans un réseau.
  - **Amélioration des performances et de la sécurité** : Introduit un noyau 32 bits plus robuste et une gestion améliorée des utilisateurs et des permissions.
  - **Services IIS 6.0** : Le serveur web Microsoft IIS (Internet Information Services) a été mis à jour pour améliorer les performances et la sécurité.
  - **Réseau amélioré** : Windows Server 2003 a introduit des outils pour une gestion améliorée des réseaux et une prise en charge des connexions VPN plus simples.
- **Usage typique :** Cette version était utilisée dans de petites et moyennes entreprises pour les applications Web, la gestion d'annuaires et la mise en réseau. Aujourd'hui, elle est obsolète et plus prise en charge par Microsoft.

3.2.2 Windows Server 2008

- **Date de lancement :** 2008
- **Principales caractéristiques :**
  - **Hyper-V** : Introduction de la virtualisation avec **Hyper-V**, permettant la création et la gestion de machines virtuelles, ce qui a révolutionné la gestion des ressources serveurs.
  - **Améliorations de la sécurité** : Introduction de **Windows Firewall** avec des fonctionnalités avancées, ainsi que des contrôles d'accès plus stricts et des mises à jour de sécurité automatiques.
  - **Gestion améliorée des rôles** : Le concept de rôles de serveur a été renforcé, permettant une gestion plus facile des services serveur comme DNS, DHCP, etc.
  - **Windows Server Core** : Permet d'installer une version de Windows Server avec une interface graphique minimale (pas d'interface utilisateur graphique), ce qui réduit la consommation de ressources.
- **Usage typique :** Utilisé principalement pour les entreprises ayant besoin de virtualisation (avec Hyper-V), de services réseau ou de gestion centralisée des utilisateurs et des ressources.

3.2.3 Windows Server 2012

- **Date de lancement :** 2012
- **Principales caractéristiques :**
  - **Interface améliorée (Metro)** : L'interface utilisateur a été redessinée pour se concentrer davantage sur l'administration à distance et l'automatisation des tâches via PowerShell.
  - **Virtualisation et Cloud** : Renforcement des capacités de virtualisation et une meilleure prise en charge des environnements Cloud, avec des outils comme **Windows Server Hyper-V 3.0**.
  - **Nouveau système de gestion de stockage** : Fonctionnalités de stockage avancées comme **Storage Spaces**, permettant de créer des pools de stockage et de gérer des volumes de manière plus flexible.
  - **Amélioration de PowerShell** : Introduction de la version 3 de PowerShell avec des cmdlets étendues, permettant une automatisation encore plus poussée des tâches d'administration.
  - **Windows Server Core amélioré** : La version sans interface graphique devient beaucoup plus populaire et bien soutenue.
- **Usage typique :** Utilisé par les entreprises ayant des besoins de virtualisation, de Cloud computing ou cherchant à automatiser la gestion de leurs serveurs. La gestion via PowerShell est devenue une pratique essentielle.

3.2.4 Windows Server 2016

- **Date de lancement :** 2016
- **Principales caractéristiques :**
  - **Docker et Conteneurs Windows** : Introduction de la prise en charge de Docker et des conteneurs, permettant l'exécution d'applications de manière isolée et plus légère.
  - **Nouveaux outils de gestion** : Introduction de **Windows Admin Center**, un outil de gestion graphique pour administrer des serveurs à distance.
  - **Security Improvements** : Renforcement de la sécurité avec des fonctionnalités comme **Windows Defender** et l'**Advanced Threat Protection (ATP)**.
  - **Améliorations de la virtualisation avec Hyper-V** : Ajout de nouvelles fonctionnalités pour une gestion plus flexible et performante des environnements virtuels.
  - **Nano Server** : Une version encore plus minimaliste de Windows Server, pour des déploiements ultra-légers dans des environnements Cloud et des applications modernes.
- **Usage typique :** Adapté aux entreprises modernes utilisant des environnements Cloud, des conteneurs, ou souhaitant renforcer la sécurité et la virtualisation.

3.2.5 Windows Server 2019

- **Date de lancement :** 2018
- **Principales caractéristiques :**
  - **Prise en charge du Cloud hybride** : Intégration améliorée avec **Microsoft Azure**, permettant une gestion simplifiée des ressources Cloud et sur site.
  - **Sécurité renforcée** : Ajout de fonctionnalités de sécurité comme **Windows Defender ATP** et **Security Subsystem** pour améliorer la protection des données et des systèmes.
  - **Conteneurs et Kubernetes** : Amélioration de la prise en charge des conteneurs, avec une meilleure intégration avec **Kubernetes** pour la gestion des applications conteneurisées.
  - **Déploiement simplifié avec Windows Admin Center** : Les outils d'administration ont été mis à jour pour permettre un déploiement, une gestion et une surveillance plus simples des serveurs, avec une interface utilisateur graphique plus conviviale.
  - **Améliorations de Hyper-V** : Gestion améliorée des environnements virtuels, plus de flexibilité dans la gestion des machines virtuelles et des performances accrues.
- **Usage typique :** Idéal pour les entreprises qui adoptent une architecture hybride Cloud/on-premise et qui ont besoin de virtualisation avancée, de sécurité renforcée, ou d'intégration avec des plateformes de conteneurs.

3.2.6 Windows Server 2022

- **Date de lancement :** 2021
- **Principales caractéristiques :**
  - **Sécurité avancée** : Renforcement des fonctionnalités de sécurité avec des technologies comme **Secure Core Server**, une meilleure gestion de la sécurité matérielle et de la virtualisation.
  - **Amélioration de la virtualisation et des conteneurs** : Meilleure prise en charge des applications conteneurisées et de la gestion des conteneurs.
  - **Amélioration des capacités de stockage** : Avec **Storage Migration Service**, il est plus facile de migrer les serveurs de fichiers vers de nouveaux serveurs.
  - **Optimisation du Cloud hybride** : Une meilleure gestion des ressources à travers **Azure Arc**, permettant de gérer les environnements multi-cloud.
  - **Amélioration des performances réseau** : Des améliorations dans la gestion du réseau, avec une latence plus faible et une meilleure bande passante pour les serveurs virtuels.
- **Usage typique :** Adapté aux entreprises recherchant une solution de serveur ultra-sécurisée, optimisée pour l'usage du Cloud hybride et la gestion avancée des machines virtuelles.

3.3 Fonctionnalités clés de Windows Server

Gestion des rôles et fonctionnalités

Windows Server permet de configurer différents rôles et fonctionnalités pour un serveur, permettant à une machine de devenir un serveur de fichiers, un contrôleur de domaine, un serveur de messagerie, un serveur Web, etc. Ces rôles sont installés et configurés à l'aide du **Gestionnaire de serveur**.

Active Directory

Active Directory (AD) est un service essentiel dans les environnements Windows Server, permettant de centraliser la gestion des utilisateurs, des groupes et des ordinateurs dans un domaine.

Hyper-V (Virtualisation)

Hyper-V est la solution de virtualisation de Microsoft, permettant de créer et de gérer des machines virtuelles. Il est utilisé dans presque toutes les versions de Windows Server récentes.

PowerShell

PowerShell est l'outil de ligne de commande et de script de Windows Server. Il permet l'automatisation des tâches administratives et est largement utilisé pour la gestion à distance des serveurs.

Windows Admin Center

Un outil de gestion graphique centralisé qui facilite l'administration à distance des serveurs, permettant une gestion simple et directe des rôles et des services sans utiliser PowerShell ou des interfaces multiples.

Module 4 : La structure du script et commandes de base

Objectif du module :

L'objectif de ce module est d'introduire les concepts fondamentaux de la création de scripts, ainsi que les commandes de base couramment utilisées dans les scripts Shell et PowerShell. Ce module est essentiel pour comprendre la structure d'un script et les outils de base pour automatiser les tâches sur un serveur.

Les concepts abordés incluent :

- La structure de base d'un script.
- Les commandes fondamentales pour naviguer dans le système et gérer les fichiers.
- L'écriture et l'exécution de scripts simples pour l'automatisation des tâches.
- La manipulation de variables et l'utilisation de boucles et conditions.

4.1 Introduction à la structure d'un script

Un script est une série de commandes exécutées dans un environnement d'interprétation (par exemple, Bash pour Linux ou PowerShell pour Windows). Les scripts sont utilisés pour automatiser des tâches répétitives et gérer des systèmes efficacement.

4.1.1 Structure d'un script en Bash (Linux)

Voici la structure d'un script de base en Bash (Linux) :

```
#!/bin/bash  # Spécifie que le script doit être exécuté avec l'interpréteur bash.

# Commentaire : Cette ligne est un commentaire et n'est pas exécutée.

# Définir une variable
variable="Bonjour, monde"

# Afficher une sortie à l'écran
echo $variable  # Affiche "Bonjour, monde"
```

- **Shebang (#!/bin/bash)** : La première ligne indique à quel interpréteur utiliser pour exécuter le script (ici, `bash`).
- **Commentaires** : Les commentaires commencent par un `#` et sont ignorés par l'interpréteur.
- **Déclaration de variables** : Les variables peuvent être créées sans type explicite, comme dans `variable="Bonjour, monde"`.
- **Commande `echo`** : Utilisée pour afficher des informations à l'écran ou dans un fichier.

4.1.2 Structure d'un script en PowerShell (Windows)

Voici la structure de base d'un script en PowerShell :

```
# Commentaire : Cette ligne est un commentaire et n'est pas exécutée

# Définir une variable
$variable = "Bonjour, monde"

# Afficher une sortie à l'écran
Write-Output $variable  # Affiche "Bonjour, monde"
```

- **Commentaires :** En PowerShell, les commentaires commencent par un #.
- **Déclaration de variables** : Les variables sont précédées de `$` (exemple : `$variable`).
- **Commande `Write-Output`** : Permet d'afficher une sortie à l'écran.

4.2 Commandes de base

Les commandes de base sont essentielles pour manipuler le système, gérer les fichiers et exécuter des tâches courantes dans un script.

4.2.1 Commandes de base en Bash (Linux)

1. **Navigation dans les répertoires :**
   - `pwd` : Affiche le répertoire courant.
   - `cd` : Change de répertoire.
   - `ls` : Liste le contenu du répertoire.

Exemple :

```
cd /home/user  # Va dans le répertoire /home/user
ls             # Liste les fichiers dans le répertoire courant
```

2. **Gestion des fichiers :**
   - **touch** : Crée un fichier vide.
   - **cp** : Copie un fichier.
   - **mv** : Déplace ou renomme un fichier.
   - **rm** : Supprime un fichier.
   - **cat** : Affiche le contenu d'un fichier.
   - **echo** : Affiche une chaîne de texte ou redirige une sortie vers un fichier.

   Exemple :

```
touch fichier.txt   # Crée un fichier nommé fichier.txt
echo "Hello World" > fichier.txt  # Écrit "Hello World" dans fichier.txt
cat fichier.txt     # Affiche le contenu de fichier.txt
```

3. **Permissions de fichiers :**
   - **chmod** : Modifie les permissions d'un fichier.
   - chown : Change le propriétaire d'un fichier.

   Exemple :

```
chmod 755 script.sh   # Donne des permissions d'exécution au propriétaire
chown user:user fichier.txt  # Change le propriétaire de fichier.txt
```

4.2.2 Commandes de base en PowerShell (Windows)

1. **Navigation dans les répertoires :**
   - **Get-Location** : Affiche le répertoire courant.
   - **Set-Location** : Change de répertoire.
   - **Get-ChildItem** : Liste le contenu du répertoire.

   Exemple :

```
Set-Location C:\Users\Public  # Va dans le répertoire C:\Users\Public
Get-ChildItem                 # Liste les fichiers dans le répertoire courant
```

2. **Gestion des fichiers :**
   - **New-Item** : Crée un nouveau fichier ou répertoire.
   - **Copy-Item** : Copie un fichier ou répertoire.
   - **Move-Item** : Déplace ou renomme un fichier ou répertoire.
   - **Remove-Item** : Supprime un fichier ou répertoire.
   - **Get-Content** : Affiche le contenu d'un fichier.
   - **Set-Content** : Écrit dans un fichier.

   Exemple :

```
New-Item -Path "C:\temp\fichier.txt" -ItemType File  # Crée un fichier
Set-Content -Path "C:\temp\fichier.txt" -Value "Hello World"  # Écrit dans fichier.txt
Get-Content -Path "C:\temp\fichier.txt"  # Affiche le contenu de fichier.txt
```

3. **Permissions de fichiers :**
    - `Get-Acl` : Récupère les permissions d'un fichier.
    - `Set-Acl` : Modifie les permissions d'un fichier.

    Exemple :

```
Get-Acl -Path "C:\temp\fichier.txt"  # Affiche les permissions de fichier.txt
```

4.3 Variables et Types de Données

Les scripts utilisent des variables pour stocker des données. En fonction du langage, la manière de déclarer et d'utiliser les variables peut différer.

4.3.1 Variables en Bash (Linux)

- **Déclaration de variables :** `variable="Valeur"`
- **Utilisation de la variable :** `echo $variable  # Affiche "Valeur"`
- **Types de données** : En Bash, il n'est pas nécessaire de spécifier un type pour une variable (elles sont typées dynamiquement).

4.3.2 Variables en PowerShell (Windows)

- **Déclaration de variables :** `$variable = "Valeur"`
- **Utilisation de la variable :** `Write-Output $variable  # Affiche "Valeur"`
- **Types de données** : PowerShell gère les types de données de manière plus explicite que Bash, mais les variables peuvent être de différents types comme `String`, `Int`, `Boolean`, etc.

4.4 Conditions et Boucles

Les conditions et les boucles sont des concepts clés pour rendre un script dynamique et adaptable.

4.4.1 Conditions (if, elif, else)

- **Bash :**

```bash
if [ $variable -eq 10 ]; then
  echo "La variable est égale à 10."
else
  echo "La variable n'est pas égale à 10."
fi
```

- **PowerShell :**

```powershell
if ($variable -eq 10) {
    Write-Output "La variable est égale à 10."
} else {
    Write-Output "La variable n'est pas égale à 10."
}
```

4.4.2 Boucles (for, while)

- **Bash (for) :**

```
for i in {1..5}; do
  echo "Compteur : $i"
done
```

- **PowerShell (for) :**

```
for ($i = 1; $i -le 5; $i++) {
    Write-Output "Compteur : $i"
}
```

- **Boucles `while`** : Les boucles `while` exécutent des instructions tant qu'une condition est vraie.

```
bash

while [ $i -le 5 ]; do
  echo "Compteur : $i"
  ((i++))
done
```

Module 5 : Introduction au Shell Bash

🎯 Objectif du module :

Ce module a pour but de vous initier à **Bash (Bourne Again SHell)**, un interpréteur de commandes très répandu dans les systèmes Linux. Vous apprendrez à comprendre le rôle du shell, à exécuter des commandes simples, à écrire vos premiers scripts Bash et à commencer à automatiser des tâches répétitives.

❓ 5.1 Qu'est-ce que le Shell Bash ?

Bash est un **interpréteur de commandes** qui permet de dialoguer avec le système d'exploitation en ligne de commande.

- C'est un **langage de script** utilisé pour automatiser des tâches.
- Bash est le **Shell par défaut** sur la plupart des distributions Linux (Ubuntu, Debian, CentOS, etc.).
- Il permet d'exécuter **des commandes Unix/Linux** directement ou via des scripts.

📄 5.2 Lancement du Shell Bash

Vous pouvez lancer Bash :

- En ouvrant un **terminal** (Ctrl + Alt + T sur Ubuntu).
- En se connectant à distance via SSH : **`ssh utilisateur@adresse_IP`**

🔤 5.3 Commandes Bash de base à connaître

Commande	Description
`pwd`	Affiche le répertoire courant
`ls`	Liste les fichiers du répertoire
`cd`	Change de répertoire
`touch fichier`	Crée un fichier
`mkdir dossier`	Crée un répertoire
`rm fichier`	Supprime un fichier
`mv`	Déplace ou renomme un fichier
`cp`	Copie un fichier
`echo`	Affiche un message à l'écran
`cat fichier`	Affiche le contenu d'un fichier
`chmod`	Change les permissions d'un fichier
`chown`	Change le propriétaire d'un fichier

📜 5.4 Écrire son premier script Bash

✓ Étapes :

1. Créer un fichier : `nano premier_script.sh`
2. Écrire le contenu suivant :

```
#!/bin/bash
echo "Bonjour, bienvenue dans le monde de Bash !"
```

3. Rendre le fichier exécutable : `chmod +x premier_script.sh`
4. L'exécuter : `./premier_script.sh`

❓ 5.5 Variables dans Bash

```
nom="Alice"
echo "Bonjour $nom"  # Affiche : Bonjour Alice
```

- Pas d'espace autour du `=`
- Les variables sont appelées avec `$`

5.6 Conditions dans Bash

```
nombre=10

if [ $nombre -eq 10 ]; then
  echo "Le nombre est 10"
else
  echo "Le nombre n'est pas 10"
fi
```

- `-eq`, `-ne`, `-lt`, `-gt`, `-le`, `-ge` sont utilisés pour les comparaisons numériques.

5.7 Boucles dans Bash

For Loop :

```
for i in {1..5}; do
  echo "Itération $i"
done
```

While Loop :

```
compteur=1
while [ $compteur -le 5 ]; do
  echo "Compteur : $compteur"
  ((compteur++))
done
```

5.8 Manipulation de fichiers avec Bash

```
# Créer un fichier
touch rapport.txt

# Écrire dans un fichier
echo "Rapport généré le $(date)" > rapport.txt

# Lire un fichier
cat rapport.txt
```

5.9 Gestion des permissions

```
chmod +x mon_script.sh       # Donne les droits d'exécution
chmod 644 fichier.txt        # rw-r--r-- (lecture/écriture propriétaire, lecture autres)
```

5.10 Bonnes pratiques en scripting Bash

- Commencez toujours vos scripts par `#!/bin/bash`
- Commentez votre code `(#)`
- Testez vos scripts en conditions réelles
- Vérifiez les erreurs (`set -e`, `set -u`, `set -o pipefail`)
- Utilisez des noms de variables clairs et explicites

Exercice de fin de module :

1. Créez un script qui demande un nom à l'utilisateur et affiche un message personnalisé.
2. Modifiez ce script pour qu'il crée un fichier `hello_<nom>.txt` contenant la date du jour.

Indice :

```bash
#!/bin/bash
read -p "Entrez votre prénom : " nom
echo "Bonjour $nom"
echo "Fichier créé le $(date)" > hello_${nom}.txt
```

Exemple 1 : Script d'installation automatique d'un serveur Apache avec page d'accueil personnalisée

```bash
#!/bin/bash

# Vérifie si le script est exécuté en tant que root
if [[ $EUID -ne 0 ]]; then
   echo "Ce script doit être exécuté en tant que root."
   exit 1
fi

echo "Mise à jour des paquets..."
apt update -y && apt upgrade -y

echo "Installation du serveur Apache..."
apt install apache2 -y

echo "Création d'une page d'accueil personnalisée..."
echo "<h1>Bienvenue sur mon serveur Apache !</h1>" > /var/www/html/index.html

echo "Démarrage et activation d'Apache..."
systemctl start apache2
systemctl enable apache2

echo "Installation terminée. Le site est disponible sur http://$(hostname -I | awk '{print $1}')"
```

🔐 Exemple 2 : Script de configuration du pare-feu avec `ufw`

bash Copier Modifier

```bash
#!/bin/bash

echo "Activation du pare-feu UFW..."
ufw allow 22   # SSH
ufw allow 80   # HTTP
ufw enable

echo "Configuration terminée. Règles UFW actuelles :"
ufw status
```

🔧 Exemple 3 : Script d'installation et configuration d'un serveur SSH sécurisé

bash Copier Modifier

```bash
#!/bin/bash

echo "Installation du service OpenSSH..."
apt install openssh-server -y

echo "Configuration de SSH..."
sed -i 's/#Port 22/Port 22/' /etc/ssh/sshd_config
sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin no/' /etc/ssh/sshd_config
sed -i 's/#PasswordAuthentication yes/PasswordAuthentication no/' /etc/ssh/sshd_config

echo "Redémarrage du service SSH..."
systemctl restart ssh

echo "SSH sécurisé activé. Pensez à configurer vos clés publiques si ce n'est pas déjà fait."
```

📦 Exemple 4 : Script de sauvegarde automatique quotidienne d'un dossier

bash Copier Modifier

```bash
#!/bin/bash

# Répertoire à sauvegarder
SOURCE="/var/www/html"
# Répertoire de destination
DEST="/backup"
# Nom du fichier de sauvegarde avec date
DATE=$(date +%F)
FILE="sauvegarde-$DATE.tar.gz"

# Création du dossier de sauvegarde s'il n'existe pas
mkdir -p $DEST

# Création de l'archive
tar -czf $DEST/$FILE $SOURCE

echo "Sauvegarde créée : $DEST/$FILE"
```

🛠️ Exemple 5 : Script de création d'un utilisateur avec mot de passe généré

bash Copier Modifier

```bash
#!/bin/bash

read -p "Entrez le nom de l'utilisateur à créer : " username

# Génération d'un mot de passe aléatoire
password=$(openssl rand -base64 12)

# Création de l'utilisateur et définition du mot de passe
useradd -m $username
echo "$username:$password" | chpasswd

echo "Utilisateur $username créé avec succès."
echo "Mot de passe : $password"
```

📋 Exemple 6 : Script de rapport système complet

```bash
#!/bin/bash

echo "===== RAPPORT SYSTÈME ====="
echo "Date : $(date)"
echo "Nom de la machine : $(hostname)"
echo "Adresse IP : $(hostname -I | awk '{print $1}')"
echo "Uptime : $(uptime -p)"
echo "Utilisation disque :"
df -h
echo "Mémoire disponible :"
free -h
```

Module 6 : La création du Premier Script Shell Bash

🎯 Objectifs pédagogiques :

À la fin de ce module, l'apprenant sera capable de :

- Créer un script Bash simple.
- Exécuter un script correctement dans un environnement Linux.
- Utiliser les **commandes de base**, **les variables**, **l'affichage**, et **les conditions** dans un script.
- Déboguer un script simple.

6.1 Pourquoi écrire des scripts Bash ?

- **Automatiser des tâches répétitives** : nettoyage de fichiers temporaires, sauvegardes, mises à jour...
- **Standardiser les installations** : sur plusieurs serveurs ou VM.
- **Réduire les erreurs humaines** : pas besoin de retaper les commandes.

📜 6.2 Structure de base d'un script Bash

Un script Bash est simplement un **fichier texte** contenant une série de **commandes** que le shell exécutera **de haut en bas**.

```
#!/bin/bash
# Ceci est un commentaire

echo "Bonjour, ceci est mon premier script Bash !"
```

Détails :

- **#!/bin/bash** → Shebang : indique au système que le script doit être interprété avec Bash.
- **echo** → affiche un message à l'écran.
- **#** → permet d'écrire des commentaires (ignorer par Bash).

🛠 6.3 Étapes pour créer un script Bash

✓ Étape 1 : Créer un fichier script

nano mon_script.sh

✓ Étape 2 : Écrire le code suivant

```
#!/bin/bash
echo "Bienvenue dans le monde des scripts Bash !"
```

✓ Étape 3 : Rendre le script exécutable

chmod +x mon_script.sh

✓ Étape 4 : Exécuter le script

./mon_script.sh

🔤 6.4 Utiliser des variables dans un script

```
#!/bin/bash

nom="Alice"
echo "Bonjour $nom"
```

💡 **Attention** : ne pas mettre d'espaces autour du = lors de la déclaration.

❓ 6.5 Lire une entrée utilisateur

```
#!/bin/bash

read -p "Quel est votre prénom ? " prenom
echo "Enchanté, $prenom !"
```

❓ 6.6 Ajout de conditions

```
#!/bin/bash

read -p "Entrez un nombre : " nb

if [ $nb -gt 10 ]; then
  echo "Le nombre est supérieur à 10"
else
  echo "Le nombre est inférieur ou égal à 10"
fi
```

🔁 6.7 Ajouter une boucle

```
#!/bin/bash

for i in {1..5}; do
  echo "Itération numéro $i"
done
```

6.8 Exercice guidé : Crée ton propre script interactif

🎯 Objectif : créer un script qui :

1. Demande à l'utilisateur son prénom.
2. Crée un fichier `bonjour_<prenom>.txt`.
3. Écrit dedans un message de bienvenue avec la date du jour.

```
#!/bin/bash

read -p "Entrez votre prénom : " prenom
date=$(date)

echo "Bonjour $prenom, bienvenue ! Nous sommes le $date." > "bonjour_${prenom}.txt"
echo "Fichier créé : bonjour_${prenom}.txt"
```

Module 7 : La prise en main de PowerShell

🎯 Objectifs pédagogiques :

À l'issue de ce module, l'apprenant sera capable de :

- Comprendre ce qu'est PowerShell et pourquoi l'utiliser.
- Exécuter des commandes simples.
- Écrire un script PowerShell de base.
- Manipuler des objets, variables, conditions et boucles.
- Interagir avec le système (fichiers, services, utilisateurs, etc.).

7.1 Qu'est-ce que PowerShell ?

PowerShell est un **shell en ligne de commande** et un **langage de script** développé par Microsoft pour :

- **Administrer des systèmes Windows** (et maintenant Linux/macOS grâce à PowerShell Core).
- Automatiser des tâches administratives : gestion des utilisateurs, fichiers, services, etc.
- Manipuler des objets (PowerShell est basé sur .NET).

7.2 Lancer PowerShell

- **Windows** : clic droit sur le menu Démarrer → **Windows PowerShell (admin)**
- **Linux/macOS** : via PowerShell Core (`pwsh`)
- Pour un script `.ps1` : clic droit → Exécuter avec PowerShell ou dans un terminal :

  ```
  ./mon_script.ps1
  ```

⍰ 7.3 Premières commandes PowerShell

Commande	Description
`Get-Command`	Liste les commandes disponibles
`Get-Help`	Aide sur une commande (`Get-Help Get-Process`)
`Get-Process`	Affiche les processus en cours
`Get-Service`	Liste les services
`Start-Service`	Démarre un service
`Stop-Service`	Arrête un service
`Set-ExecutionPolicy`	Change la politique d'exécution

⚠ Pour exécuter un script :

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

🔤 7.4 Variables en PowerShell

```
$nom = "Alice"
Write-Output "Bonjour, $nom"
```

- Toutes les variables commencent par `$`
- Pas besoin de déclarer le type

🎁 7.5 Manipulation de fichiers

```
New-Item -Path "C:\Users\Public\" -Name "demo.txt" -ItemType "File"
Set-Content -Path "C:\Users\Public\demo.txt" -Value "Ceci est une démo."
Get-Content "C:\Users\Public\demo.txt"
```

🔁 7.6 Conditions et boucles

✓ Condition `if`

```
$age = 20
if ($age -ge 18) {
  Write-Output "Majeur"
} else {
  Write-Output "Mineur"
}
```

🔁 Boucle `for`

```
for ($i = 1; $i -le 5; $i++) {
  Write-Output "Itération $i"
}
```

Boucle `foreach`

```
$services = Get-Service | Where-Object {$_.Status -eq "Running"}
foreach ($service in $services) {
  Write-Output $service.Name
}
```

7.7 Créer un script PowerShell de base

1. Créer un fichier `script.ps1`
2. Exemple de script :

```
# script.ps1
$name = Read-Host "Quel est votre prénom ?"
Write-Output "Bonjour, $name. Bienvenue sur PowerShell !"
```

3. Lancer avec :

```
.\script.ps1
```

7.8 Exercice guidé : Créer un utilisateur local

Créez un script PowerShell qui :

- Crée un utilisateur local nommé `stagiaire01`.
- Lui assigne un mot de passe.
- Ajoute cet utilisateur au groupe `Utilisateurs`.

```
$password = ConvertTo-SecureString "P@ssword123!" -AsPlainText -Force
New-LocalUser "stagiaire01" -Password $password -FullName "Stagiaire Test"
Add-LocalGroupMember -Group "Users" -Member "stagiaire01"
Write-Output "Utilisateur créé avec succès."
```

Exercice de fin de module

Créez un script PowerShell qui :

- Demande un nom de fichier à l'utilisateur
- Crée le fichier avec la date et l'heure actuelles
- Affiche un message de confirmation

📝 Script PowerShell – Création d'un fichier avec date/heure

powershell Copier Modifier

```powershell
# Demande du nom de fichier à l'utilisateur
$fileName = Read-Host "Entrez le nom du fichier (sans extension)"

# Récupère la date et l'heure actuelles au format lisible
$date = Get-Date -Format "yyyy-MM-dd_HH-mm-ss"

# Construit le nom complet du fichier avec timestamp
$fullName = "$fileName-$date.txt"

# Crée le fichier dans le répertoire courant
New-Item -Path "." -Name $fullName -ItemType "File" | Out-Null

# Message de confirmation
Write-Host "Fichier '$fullName' créé avec succès dans le dossier : $(Get-Location)"
```

✅ Exemple d'utilisation :

bash Copier Modifier

```bash
Entrez le nom du fichier (sans extension) : rapport
→ Fichier 'rapport-2025-04-24_14-28-59.txt' créé avec succès dans le dossier : C:\Users\...
```

Module 8 : La création d'un script d'automatisation PowerShell sous Windows

🎯 Objectifs pédagogiques :

À la fin de ce module, l'apprenant sera capable de :

- Créer un script PowerShell structuré.
- Automatiser des tâches administratives Windows (utilisateurs, services, fichiers, installation de logiciels).
- Gérer les erreurs et les journaux d'activité.
- Appliquer des scripts en environnement réel (VM, postes utilisateurs, serveurs).

🔎 8.1 Qu'est-ce qu'un script d'automatisation PowerShell ?

Un script d'automatisation PowerShell est un **fichier `.ps1`** qui permet d'exécuter une ou plusieurs **tâches sans intervention manuelle** :

- Création automatique d'utilisateurs
- Configuration réseau ou système
- Installation de logiciels
- Surveillance système

- Planification de tâches

8.2 Structure d’un script PowerShell automatisé

```
<#
    Auteur : Votre nom
    Description : Script d’automatisation de [nom de la tâche]
    Date : 24/04/2025
#>

# Paramètres
# Fonctions
# Corps principal
# Journalisation
```

8.3 Exemple pratique 1 : Automatiser la création d’un utilisateur local avec mot de passe

```
# Paramètres
$username = "stagiaire02"
$password = ConvertTo-SecureString "Pa$$w0rd2025" -AsPlainText -Force

# Création de l’utilisateur
New-LocalUser -Name $username -Password $password -FullName "Utilisateur Stagiaire" -Description
Add-LocalGroupMember -Group "Users" -Member $username

Write-Output "Utilisateur $username créé avec succès."
```

8.4 Exemple pratique 2 : Automatiser l'installation de logiciels avec `winget`

```
# Installe Notepad++ et VLC via Winget
winget install --id Notepad++.Notepad++ -e --accept-package-agreements --accept-source-agreements
winget install --id VideoLAN.VLC -e --accept-package-agreements --accept-source-agreements

Write-Host "Logiciels installés automatiquement."
```

8.5 Exemple pratique 3 : Sauvegarde automatisée d’un dossier

```
$source = "C:\Users\Public\Documents"
$destination = "D:\sauvegardes"
$date = Get-Date -Format "yyyy-MM-dd_HH-mm-ss"
$archive = "$destination\sauvegarde_$date.zip"

Compress-Archive -Path $source -DestinationPath $archive

Write-Host "Sauvegarde créée avec succès : $archive"
```

8.6 Gestion des erreurs et journalisation

```
try {
    # Code principal
    Get-Service "Spouleur d'impression" | Start-Service
    Write-Output "Service démarré."
} catch {
    # En cas d'erreur
    Write-Error "Erreur : $_"
    Add-Content -Path "C:\logs\erreurs.log" -Value "$(Get-Date) : $_"
}
```

8.7 Planifier l'exécution automatique d'un script

Via l'outil Planificateur de tâches Windows :

1. Créer une tâche → **Déclencheur** : à l'ouverture de session ou quotidiennement.
2. **Action** : `powershell.exe -File "C:\scripts\mon_script.ps1"`
3. Vérifier l'exécution automatique.

8.8 Bonnes pratiques d'automatisation PowerShell

✅ Toujours :

- Tester les scripts sur une **VM** ou un environnement de test.
- Ajouter des **commentaires clairs**.
- Valider les actions sensibles avec `-WhatIf`.
- Gérer les erreurs (`try/catch`) et faire des **logs d'activité**.
- Centraliser les variables/modifiables en haut du script.

8.9 Mini-projet de fin de module :

🎯 Créez un script PowerShell qui :

- Crée un utilisateur local avec mot de passe sécurisé.
- Installe 2 logiciels via Winget.
- Crée un répertoire de travail pour l'utilisateur.
- Enregistre toutes les actions dans un fichier de log.

Voici la **solution complète du mini-projet 8.9** : un **script PowerShell automatisé** qui :

✅ Crée un utilisateur local
✅ Installe 2 logiciels avec `winget`
✅ Crée un répertoire personnel pour l'utilisateur
✅ Journalise chaque étape dans un fichier log

```powershell
<#
    Script : automatisation.ps1
    Objectif : Automatiser la création d'un utilisateur, l'installation de logiciels, la création de dossier, et le logging
    Auteur : Votre Nom
    Date : 2025-04-24
#>

# === Paramètres ===
$nomUtilisateur = "stagiaire03"
$motDePasse = ConvertTo-SecureString "Secur3P@ss2025" -AsPlainText -Force
$cheminLog = "C:\Logs\automatisation.log"
$logiciels = @("Notepad++.Notepad++", "VideoLAN.VLC")
$cheminDossierUtilisateur = "C:\Utilisateurs\Travail_$nomUtilisateur"

# === Fonction de log ===
function Log($message) {
    $horodatage = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
    "$horodatage - $message" | Add-Content -Path $cheminLog
    Write-Host $message
}

# === Vérification / création du dossier de logs ===
if (!(Test-Path -Path "C:\Logs")) {
    New-Item -Path "C:\Logs" -ItemType Directory | Out-Null
}

Log "Démarrage du script d'automatisation."

# === Étape 1 : Création de l'utilisateur ===
try {
    New-LocalUser -Name $nomUtilisateur -Password $motDePasse -FullName "Utilisateur stagiaire" -Description "Ajout automatique" -ErrorAction Stop
    Add-LocalGroupMember -Group "Users" -Member $nomUtilisateur -ErrorAction Stop
    Log "✅ Utilisateur '$nomUtilisateur' créé et ajouté au groupe Users."
} catch {
    Log "❌ Erreur lors de la création de l'utilisateur : $_"
}

# === Étape 2 : Installation des logiciels avec Winget ===
foreach ($app in $logiciels) {
    try {
        winget install --id $app -e --accept-package-agreements --accept-source-agreements
        Log "✅ Logiciel installé : $app"
    } catch {
        Log "❌ Échec d'installation pour $app : $_"
    }
```

```powershell
# === Étape 3 : Création d'un répertoire personnel ===
try {
    if (!(Test-Path -Path $cheminDossierUtilisateur)) {
        New-Item -Path $cheminDossierUtilisateur -ItemType Directory | Out-Null
        Log "✅ Dossier de travail créé : $cheminDossierUtilisateur"
    } else {
        Log "⚠️ Dossier déjà existant : $cheminDossierUtilisateur"
    }
} catch {
    Log "❌ Erreur lors de la création du dossier : $_"
}

Log "🎉 Script terminé avec succès."
```

Module 9 : La création d'un script d’automatisation avec Python

Objectifs pédagogiques :

À l’issue de ce module, l’apprenant sera capable de :

- Comprendre les bases de l’automatisation avec Python.
- Manipuler les fichiers, dossiers, processus système.
- Créer des scripts réutilisables et structurés.
- Automatiser des tâches courantes : gestion de fichiers, installation de packages, interaction réseau, etc.

9.1 Pourquoi utiliser Python pour automatiser ?

- **Langage universel** : multi-OS (Windows, Linux, macOS)
- Lisible, rapide à écrire et maintenir
- Dispose de **nombreuses bibliothèques** pour la gestion de fichiers, réseau, web, etc.
- Idéal pour automatiser les tâches système, DevOps, admin IT, cloud...

9.2 Pré-requis et environnement

- Python installé (3.10 ou +)
- Utilisation de l'éditeur : VSCode, PyCharm, ou éditeur simple (Notepad++, etc.)
- Exécution dans terminal :

```
python mon_script.py
```

9.3 Structure d’un script d’automatisation

```python
#!/usr/bin/env python3
"""
Script : mon_script.py
Description : Exemple d'automatisation avec Python
Auteur : Votre nom
Date : 2025-04-24
"""

import os
import datetime

# Fonctions ici

def main():
    # Code principal ici
    pass

if __name__ == "__main__":
    main()
```

9.4 Automatiser la gestion des fichiers et dossiers

✓ Créer un dossier

```
import os

nom_dossier = "Sauvegarde"
if not os.path.exists(nom_dossier):
    os.makedirs(nom_dossier)
    print(f"Dossier '{nom_dossier}' créé.")
```

✓ Lister les fichiers

```
fichiers = os.listdir(".")
for f in fichiers:
    print(f)
```

9.5 Automatiser l'installation de packages (Python ou système)

Avec pip (dans le script)

```
import subprocess

subprocess.run(["pip", "install", "requests"])
```

Exécuter une commande système

```
subprocess.run(["ping", "google.com"])
```

9.6 Script pratique : créer une archive ZIP automatiquement

```
import zipfile
import os

def sauvegarder_dossier(source, destination):
    with zipfile.ZipFile(destination, 'w') as archive:
        for dossier, sous_dossiers, fichiers in os.walk(source):
            for fichier in fichiers:
                chemin = os.path.join(dossier, fichier)
                archive.write(chemin, os.path.relpath(chemin, source))
    print(f"Archive créée : {destination}")

sauvegarder_dossier("C:/Users/Public/Documents", "sauvegarde.zip")
```

9.7 Utiliser des fichiers logs pour le suivi

```
import logging

logging.basicConfig(filename='automation.log', level=logging.INFO)

logging.info("Script lancé")
try:
    # Tâche automatisée
    logging.info("Tâche exécutée avec succès")
except Exception as e:
    logging.error(f"Erreur rencontrée : {e}")
```

9.8 Script complet : Création utilisateur + dossier + log

```python
import os
import logging
import subprocess

# Configuration du log
logging.basicConfig(filename="auto_python.log", level=logging.INFO)

def creer_utilisateur_windows(nom_utilisateur):
    try:
        subprocess.run(["net", "user", nom_utilisateur, "P@ssword123", "/add"], check=True)
        logging.info(f"Utilisateur {nom_utilisateur} créé avec succès.")
    except subprocess.CalledProcessError as e:
        logging.error(f"Erreur lors de la création de l'utilisateur : {e}")

def creer_dossier_utilisateur(nom_utilisateur):
    dossier = f"C:/UtilisateursPython/{nom_utilisateur}"
    os.makedirs(dossier, exist_ok=True)
    logging.info(f"Dossier {dossier} créé.")
    return dossier

def main():
    utilisateur = "testpython"
    creer_utilisateur_windows(utilisateur)
    creer_dossier_utilisateur(utilisateur)

if __name__ == "__main__":
    main()
```

9.9 Exercice de fin de module

📌 Crée un script Python qui :

- Demande un nom de dossier à l'utilisateur
- Crée ce dossier s'il n'existe pas
- Crée un fichier `info.txt` à l'intérieur avec la date, l'heure et un message
- Enregistre un log dans `journal.log`

✅ Objectif du script :

- Demander un nom de dossier à l'utilisateur
- Créer ce dossier s'il n'existe pas
- Créer un fichier `info.txt` avec la date, l'heure et un message personnalisé
- Journaliser toutes les étapes dans `journal.log`

🪢 Script Python – `creation_dossier.py`

```python
import os
from datetime import datetime
import logging

# Configuration du fichier de journalisation
logging.basicConfig(
    filename='journal.log',
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)

def main():
    # Demande du nom de dossier
    nom_dossier = input("Entrez le nom du dossier à créer : ").strip()

    if not nom_dossier:
        print("❌ Nom de dossier invalide.")
        return
```

```python
    # Création du dossier
    try:
        if not os.path.exists(nom_dossier):
            os.makedirs(nom_dossier)
            print(f"✅ Dossier '{nom_dossier}' créé.")
            logging.info(f"Dossier '{nom_dossier}' créé avec succès.")
        else:
            print(f"⚠️ Le dossier '{nom_dossier}' existe déjà.")
            logging.warning(f"Dossier '{nom_dossier}' déjà existant.")
    except Exception as e:
        print(f"❌ Erreur lors de la création du dossier : {e}")
        logging.error(f"Erreur création dossier : {e}")
        return

    # Création du fichier info.txt
    try:
        chemin_fichier = os.path.join(nom_dossier, "info.txt")
        with open(chemin_fichier, "w") as fichier:
            now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            fichier.write(f"Fichier généré automatiquement.\nDate : {now}\nMerci d'utiliser ce
        print(f"✅ Fichier 'info.txt' créé dans le dossier '{nom_dossier}'.")
        logging.info(f"Fichier 'info.txt' créé dans '{nom_dossier}'.")
    except Exception as e:
        print(f"❌ Erreur lors de l'écriture du fichier : {e}")
        logging.error(f"Erreur fichier info.txt : {e}")

if __name__ == "__main__":
    main()
```

```
fichier.write(f"Fichier généré automatiquement.\nDate : {now}\nMerci d'utiliser ce script !\n")
```

💡 **Exemple d'exécution :**

```
bash

Entrez le nom du dossier à créer : RapportAvril
✅ Dossier 'RapportAvril' créé.
✅ Fichier 'info.txt' créé dans le dossier 'RapportAvril'.
```

Contenu du fichier `info.txt` :

```
vbnet

Fichier généré automatiquement.
Date : 2025-04-24 15:12:47
Merci d'utiliser ce script !
```

Module 10 : La création d'un script d'automatisation avec Bash

🎯 Objectifs pédagogiques :

À la fin de ce module, l'apprenant sera capable de :

- Créer et exécuter un script Bash autonome
- Automatiser des tâches système (création d'utilisateurs, gestion de services, installations…)
- Ajouter des conditions, des boucles, des fonctions dans un script
- Rendre un script interactif avec des arguments ou des menus

📁 10.1 Introduction : Pourquoi Bash ?

- Présent **par défaut** sur toutes les distributions Linux
- Rapide à écrire pour l'automatisation d'administration système
- Compatible avec des outils puissants : `cron`, `systemctl`, `awk`, `sed`, etc.
- Très utilisé en DevOps, scripting Cloud, CI/CD, serveurs…

🖋️ 10.2 Structure d'un script Bash

```bash
#!/bin/bash
# Nom du script : mon_script.sh
# Auteur : Votre nom
# Date : 24/04/2025
# Description : Exemple de script Bash

echo "Hello World !"
```

✅ Le `#!/bin/bash` est obligatoire : il indique quel interpréteur utiliser.

🛠 10.3 Exécution d’un script

```
chmod +x mon_script.sh      # Donne les droits d'exécution
./mon_script.sh             # Exécute le script
```

🔣 10.4 Variables et entrées utilisateur

```
nom="Alice"
echo "Bonjour $nom !"

read -p "Entrez votre ville : " ville
echo "Vous habitez à $ville."
```

❓ 10.5 Conditions

```
if [ "$1" == "start" ]; then
    echo "Service démarré"
else
    echo "Option inconnue"
fi
```

🔁 10.6 Boucles

```
for fichier in *.txt; do
    echo "Fichier trouvé : $fichier"
done
```

🔧 10.7 Fonctions

```
afficher_date() {
    echo "Nous sommes le $(date)"
}
afficher_date
```

📂 10.8 Gestion de fichiers et de permissions

```
mkdir /opt/backup
touch /opt/backup/info.txt
chmod 600 /opt/backup/info.txt
chown root:root /opt/backup/info.txt
```

❓ 10.9 Exécuter des commandes système

```
df -h > rapport_disque.txt
systemctl restart apache2
ping -c 3 google.com
```

10.10 Exemple : Script d'automatisation complet

✅ Objectif :

- Créer un utilisateur
- Créer un répertoire perso
- Lui attribuer les bons droits
- Logguer les actions

```bash
#!/bin/bash

utilisateur="stagiaire10"
motdepasse="MotDePasse123"
dossier="/home/$utilisateur"
log="/var/log/script_automatisation.log"

echo "📌 Création de l'utilisateur $utilisateur..." | tee -a $log

if id "$utilisateur" &>/dev/null; then
    echo "⚠️ L'utilisateur existe déjà." | tee -a $log
else
    useradd -m -d "$dossier" "$utilisateur"
    echo "$utilisateur:$motdepasse" | chpasswd
    echo "✅ Utilisateur $utilisateur créé avec succès." | tee -a $log
fi

echo "📁 Attribution des droits..." | tee -a $log
chmod 700 "$dossier"
chown "$utilisateur":"$utilisateur" "$dossier"

echo "🎉 Script terminé." | tee -a $log
```

💡 Astuce : Rendre le script *portable*

- Utiliser `/usr/bin/env bash` au lieu de `/bin/bash`
- Ne pas utiliser de chemins spécifiques à une distro (ex. : `/etc/init.d/` est obsolète)
- Ajouter des **tests de compatibilité** (`command -v`, `which`, etc.)

10.11 Exercice de fin de module

Crée un script Bash qui :

- Prend un nom de dossier en argument
- Crée ce dossier (s'il n'existe pas)
- Crée un fichier `rapport.txt` avec la date/heure
- Donne les permissions `755` au dossier
- Loggue les étapes dans `script.log`

🎯 Objectif :

- Créer un script Bash qui :
  1. Prend un nom de dossier en argument
  2. Crée ce dossier (s'il n'existe pas)
  3. Crée un fichier `rapport.txt` avec la date/heure actuelle
  4. Donne les permissions `755` au dossier
  5. Loggue toutes les étapes dans un fichier `script.log`

```bash
#!/bin/bash

# Vérification que le nom du dossier a été fourni en argument
if [ -z "$1" ]; then
    echo "❌ Veuillez fournir un nom de dossier en argument."
    exit 1
fi

# Initialisation des variables
dossier="$1"
rapport="$dossier/rapport.txt"
log="/var/log/script.log"

# Fonction pour ajouter au log
log_message() {
    message="$1"
    echo "$(date '+%Y-%m-%d %H:%M:%S') - $message" | tee -a $log
}

# Création du dossier
if [ ! -d "$dossier" ]; then
    mkdir "$dossier"
    log_message "✅ Dossier '$dossier' créé."
else
    log_message "⚠️ Le dossier '$dossier' existe déjà."
fi

# Création du fichier rapport.txt avec la date et l'heure
echo "Rapport généré le : $(date '+%Y-%m-%d %H:%M:%S')" > "$rapport"
log_message "✅ Fichier 'rapport.txt' créé dans '$dossier'."
```

```bash
# Attribution des permissions 755
chmod 755 "$dossier"
log_message "✅ Permissions 755 attribuées au dossier '$dossier'."

# Fin du script
log_message "🎉 Script terminé avec succès."
```

Explication du script :

1. **Vérification du paramètre** :
   - Le script commence par vérifier si un argument (le nom du dossier) a été fourni. Si non, il affiche un message d'erreur et arrête l'exécution.
2. **Variables** :
   - `dossier` contient le nom du dossier fourni en argument.
   - `rapport` définit le chemin du fichier `rapport.txt` à créer dans ce dossier.
   - `log` est le fichier où le script enregistre les logs (`/var/log/script.log`).
3. **Fonction de log** :
   - La fonction `log_message` enregistre un message avec la date et l'heure dans le fichier `script.log` et le montre aussi dans le terminal.
4. **Création du dossier** :
   - Si le dossier n'existe pas, il est créé et un message de confirmation est loggé. Si le dossier existe déjà, un avertissement est affiché.
5. **Création du fichier `rapport.txt`** :
   - Un fichier `rapport.txt` est créé dans le dossier avec la date/heure actuelles.
6. **Permissions 755** :
   - Les permissions `755` sont attribuées au dossier, permettant au propriétaire de lire, écrire et exécuter, et aux autres d'accéder et exécuter.
7. **Fin du script** :
   - Un message de succès est loggé pour indiquer que le script s'est exécuté correctement.

Exemple d'exécution du script :

1. Exécution :

```bash
./automatisation.sh MonDossier
```

2. Résultat dans le terminal :

```bash
2025-04-24 15:30:00 - ✅ Dossier 'MonDossier' créé.
2025-04-24 15:30:00 - ✅ Fichier 'rapport.txt' créé dans 'MonDossier'.
2025-04-24 15:30:00 - ✅ Permissions 755 attribuées au dossier 'MonDossier'.
2025-04-24 15:30:00 - 🎉 Script terminé avec succès.
```

3. Contenu du fichier `rapport.txt` dans le dossier `MonDossier` :

```yaml
Rapport généré le : 2025-04-24 15:30:00
```

4. Log dans `/var/log/script.log` :

```bash
2025-04-24 15:30:00 - ✅ Dossier 'MonDossier' créé.
2025-04-24 15:30:00 - ✅ Fichier 'rapport.txt' créé dans 'MonDossier'.
2025-04-24 15:30:00 - ✅ Permissions 755 attribuées au dossier 'MonDossier'.
2025-04-24 15:30:00 - 🎉 Script terminé avec succès.
```

Module 11 – Création automatique d'une machine virtuelle de serveur

🎯 Objectifs pédagogiques

À la fin de ce module, vous serez capable de :

- Comprendre les outils de virtualisation (local et cloud)
- Créer automatiquement une machine virtuelle avec un script
- Automatiser l'installation d'un OS serveur
- Définir des paramètres réseau, disques et ressources
- Déployer un serveur prêt à l'emploi avec des services préinstallés

---

🖥️ 11.1 Introduction à la virtualisation

Qu'est-ce qu'une machine virtuelle ?

Une machine virtuelle (VM) est un système d'exploitation qui s'exécute de manière isolée sur un hôte, via un hyperviseur.

🛠️ Outils courants :

- **VirtualBox** : simple à utiliser, multiplateforme
- **KVM** : hyperviseur natif Linux
- **VMware Workstation/ESXi** : solution pro
- **Vagrant** : outil d'automatisation VM (VirtualBox, VMware, etc.)
- **cloud-init** : standard Cloud pour le provisioning automatique

---

⚙️ 11.2 Préparation de l'environnement

Prérequis :

- Hyperviseur installé (VirtualBox ou KVM)
- Image ISO d'un OS serveur (ex. : Ubuntu Server, CentOS, Debian, Windows Server)
- Outils de ligne de commande : `VBoxManage`, `virsh`, `vagrant`

11.3 Création d'une VM avec VirtualBox (script Bash)

```bash
#!/bin/bash

VM_NAME="ServeurAuto"
ISO_PATH="/home/user/isos/ubuntu-server.iso"
DISK_PATH="/home/user/VMs/${VM_NAME}.vdi"

# Créer une VM
VBoxManage createvm --name "$VM_NAME" --ostype Ubuntu_64 --register

# Configuration de la VM
VBoxManage modifyvm "$VM_NAME" --memory 2048 --cpus 2 --nic1 nat

# Création du disque dur
VBoxManage createhd --filename "$DISK_PATH" --size 20000

# Attacher disque dur et ISO
VBoxManage storagectl "$VM_NAME" --name "SATA Controller" --add sata --controller IntelAHCI
VBoxManage storageattach "$VM_NAME" --storagectl "SATA Controller" --port 0 --device 0 --type hdd --medium "$DISK_PATH"
VBoxManage storageattach "$VM_NAME" --storagectl "SATA Controller" --port 1 --device 0 --type dvddrive --medium "$ISO_PATH"

# Activer le démarrage automatique
VBoxManage modifyvm "$VM_NAME" --boot1 dvd --boot2 disk --boot3 none --boot4 none

# Lancer la VM
VBoxManage startvm "$VM_NAME"
```

11.4 Automatiser l'installation de l'OS (preseed, kickstart, autounattend)

Ubuntu/Debian : Preseed ou Autoinstall (20.04+)

- Fichier `.cfg` avec réponses automatiques à l'installation
- Utilisation via :

```
append auto=true priority=critical preseed/url=http://serveur/preseed.cfg
```

RedHat/CentOS : Kickstart

- Fichier `.ks` :

```bash
url --url="http://mirror.centos.org/..."
rootpw --plaintext password
timezone Europe/Paris
```

Windows Server : Autounattend.xml

- Utilisé avec une clé USB ou ISO customisé pour répondre automatiquement à l'installation

---

11.5 Utilisation de Vagrant pour automatiser la création

```
vagrant init ubuntu/bionic64

# Modifier le fichier Vagrantfile
vagrant up
```

Exemple de configuration Vagrantfile :

```ruby
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/bionic64"
  config.vm.hostname = "serveurauto"
  config.vm.network "private_network", type: "dhcp"
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "2048"
    vb.cpus = 2
  end
end
```

🌐 11.6 Script cloud-init (VM Cloud / KVM)

```yaml
# cloud-init.yaml
#cloud-config
users:
  - name: admin
    sudo: ALL=(ALL) NOPASSWD:ALL
    ssh-authorized-keys:
      - votre_clé_ssh_publique

packages:
  - apache2
  - curl

runcmd:
  - systemctl start apache2
```

Utilisable avec une image cloud officielle : `.qcow2`, `.img`, etc.

💡 11.7 Bonnes pratiques

- Ne jamais utiliser de mots de passe en clair → préférer SSH + clés
- Documenter les scripts d'automatisation
- Tester la VM dans un environnement de développement avant mise en prod
- Vérifier la compatibilité OS / hyperviseur

❓ 11.8 Exercice de fin de module

Créer un script Bash ou Vagrantfile qui :

- Crée une VM Ubuntu Server 22.04
- Alloue 2 Go de RAM, 1 CPU, 20 Go disque
- Configure le réseau en NAT
- Installe Apache automatiquement
- Affiche un message de succès dans un fichier `/home/vagrant/info.txt`

✅ Objectif de l'exercice :

Créer un environnement automatisé avec les caractéristiques suivantes :

- VM Ubuntu Server 22.04
- 2 Go de RAM, 1 CPU, 20 Go de disque
- Réseau en **NAT**
- Apache installé automatiquement
- Un message "Installation terminée" écrit dans `/home/vagrant/info.txt`

📦 Solution avec **Vagrant**

1. 📁 Créer un dossier de projet :

bash

```
mkdir vm-serveur-auto && cd vm-serveur-auto
vagrant init ubuntu/jammy64
```

2. 📝 Modifier le `Vagrantfile` :

Voici le contenu **complet** du fichier `Vagrantfile` :

ruby Copier Modifier

```ruby
Vagrant.configure("2") do |config|
  # Choix de la box Ubuntu Server 22.04 (jammy64)
  config.vm.box = "ubuntu/jammy64"
  config.vm.hostname = "serveurauto"

  # Configuration des ressources de la VM
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "2048"
    vb.cpus = 1
    vb.customize ["modifyvm", :id, "--storagectl", "SATA Controller", "--portcount", "1"]
    vb.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]
  end

  # Réseau NAT (par défaut avec Vagrant)
  config.vm.network "forwarded_port", guest: 80, host: 8080

  # Provisionnement : installer Apache et créer un fichier d'info
  config.vm.provision "shell", inline: <<-SHELL
    sudo apt update
    sudo apt install -y apache2
    echo "Installation terminée le $(date)" > /home/vagrant/info.txt
    sudo chown vagrant:vagrant /home/vagrant/info.txt
  SHELL
end
```

3. 🚀 Lancer la création de la VM :

```bash
vagrant up
```

Vagrant va :

- Télécharger Ubuntu Server 22.04 (si ce n'est pas déjà fait)
- Créer une VM avec les ressources spécifiées
- Installer Apache automatiquement
- Créer le fichier `info.txt`

4. ✅ Vérification :

🔧 Connexion à la VM :

```bash
vagrant ssh
```

📂 Vérifier le fichier :

```bash
cat /home/vagrant/info.txt
```

Exemple de sortie :

```bash
Installation terminée le 2025-04-24 16:45:02
```

🌍 Tester Apache depuis l'hôte :

Ouvre ton navigateur et accède à http://localhost:8080 → Tu devrais voir la page par défaut Apache.

Module 12 : Installation du serveur DHCP principal

Objectifs pédagogiques

À la fin de ce module, vous serez capable de :

- Comprendre le rôle et le fonctionnement d'un serveur DHCP
- Installer un serveur DHCP sur Linux
- Configurer des plages d'adresses IP (scopes)
- Définir des baux statiques (DHCP reservations)
- Vérifier et dépanner le service DHCP

12.1 Rappel : Qu'est-ce que DHCP ?

DHCP (Dynamic Host Configuration Protocol) permet d'attribuer automatiquement :

- Adresse IP
- Masque de sous-réseau
- Passerelle (gateway)
- DNS

Cycle DHCP :

1. **DHCP Discover** (le client cherche un serveur)
2. **DHCP Offer** (le serveur répond avec une IP)
3. **DHCP Request** (le client demande l'IP proposée)
4. **DHCP ACK** (le serveur valide)

12.2 Installation du serveur DHCP sur Ubuntu/Debian

Installation :

```
sudo apt update
sudo apt install isc-dhcp-server
```

Service :

- Binaire principal : `/usr/sbin/dhcpd`
- Fichier de configuration : `/etc/dhcp/dhcpd.conf`
- Interfaces écoutées : `/etc/default/isc-dhcp-server`

⚙ 12.3 Configuration du serveur DHCP

Fichier **`/etc/dhcp/dhcpd.conf`** (exemple simple) :

```
default-lease-time 600;
max-lease-time 7200;
authoritative;

subnet 192.168.56.0 netmask 255.255.255.0 {
  range 192.168.56.100 192.168.56.150;
  option routers 192.168.56.1;
  option domain-name-servers 8.8.8.8, 1.1.1.1;
}
```

Pour lier à une interface :

```
# Dans /etc/default/isc-dhcp-server
INTERFACESv4="enp0s8"
```

12.4 Baux statiques (IP fixes par adresse MAC)

```
host clientfixe {
  hardware ethernet 08:00:27:12:34:56;
  fixed-address 192.168.56.200;
}
```

🔄 12.5 Redémarrer le service

```
sudo systemctl restart isc-dhcp-server
sudo systemctl status isc-dhcp-server
```

12.6 Tester le serveur DHCP

Depuis une autre machine :

```
dhclient -v enp0s8
```

12.7 Déboguer

Journaux :

```
sudo journalctl -u isc-dhcp-server
```

Vérifier l'écoute :

```
sudo netstat -anu | grep 67
```

Module 13 : L'automatisation de la configuration du serveur

Objectifs pédagogiques

À la fin de ce module, vous saurez :

- Automatiser la configuration initiale d'un serveur
- Utiliser des scripts (Bash, PowerShell, Python) pour configurer des services
- Gérer les dépendances logicielles, les services système et les paramètres réseau
- Maintenir une configuration cohérente et reproductible
- Documenter et versionner les configurations

13.1 Qu'entend-on par "automatisation de la configuration" ?

C'est le processus par lequel un serveur est prêt à être utilisé **sans intervention humaine manuelle**. Cela inclut :

- Mise à jour du système
- Installation de services
- Création d'utilisateurs
- Déploiement de fichiers de configuration
- Activation de services au démarrage
- Sécurisation du système (pare-feu, SSH…)

13.2 Outils courants d'automatisation

Outil	Cas d'usage typique
Bash	Systèmes Linux, automatisations simples
PowerShell	Systèmes Windows
Python	Cross-platform, logique plus complexe
Ansible	Automatisation à grande échelle, sans agent
cloud-init	Initialisation de VMs cloud/KVM

🛠️ 13.3 Exemple d'automatisation avec un script Bash

🎯 Objectif : configurer un serveur LAMP

```bash
#!/bin/bash

# Mise à jour
apt update && apt upgrade -y

# Installation de Apache, MySQL, PHP
apt install -y apache2 mysql-server php libapache2-mod-php php-mysql

# Activer les services
systemctl enable apache2
systemctl enable mysql

# Déployer un index.html personnalisé
echo "<h1>Serveur prêt à l'emploi</h1>" > /var/www/html/index.html

# Créer un utilisateur admin
useradd -m -s /bin/bash admin
echo "admin:password123" | chpasswd
```

💡 13.4 Exemples de tâches automatisables

Tâche	Exemple
Création d'utilisateurs	Script Bash ou PowerShell
Installation de paquets	`apt`, `yum`, `choco`, `winget`
Activation de services	`systemctl enable` ou `Set-Service`
Déploiement de configuration	`cp`, `rsync`, ou par templates (Ansible, etc.)
Sécurisation (pare-feu)	`ufw`, `iptables`, `netsh`, `firewalld`
Monitoring et logs	Setup de Prometheus, rsyslog, ou alertes par mail

---

🔄 13.5 Exemple avec PowerShell (Windows)

```powershell
# Installer IIS
Install-WindowsFeature -Name Web-Server -IncludeManagementTools

# Déployer une page HTML
Set-Content -Path "C:\inetpub\wwwroot\index.html" -Value "<h1>Serveur Windows prêt</h1>"

# Activer le service
Start-Service W3SVC
Set-Service -Name W3SVC -StartupType Automatic
```

13.6 Tester l'automatisation

- Créez une machine virtuelle vierge
- Appliquez votre script
- Vérifiez si tous les services sont actifs, les ports ouverts, les pages web disponibles
- Utilisez `curl`, `ping`, `systemctl status`, etc.

13.7 Bonnes pratiques

- Utiliser des **logs** dans les scripts (`tee`, `logger`)
- Toujours valider avec un **test de configuration** (ex. `apache2ctl configtest`)
- Séparer les scripts par tâche (modularité)
- Ajouter des **vérifications** de dépendances
- Documenter et versionner dans **Git**

13.8 Intégration avec Vagrant ou cloud-init

Vagrant :

```ruby
config.vm.provision "shell", path: "scripts/setup-server.sh"
```

cloud-init :

```yaml
runcmd:
  - apt update && apt install -y nginx
  - echo "Bonjour VM cloud" > /var/www/html/index.html
```

13.9 Exercice de fin de module (Mini-projet)

Objectif : Créer un script Bash (ou PowerShell) qui :

• Configure un serveur web (Apache ou IIS) • Déploie une page personnalisée • Crée un utilisateur avec un mot de passe • Active automatiquement les services nécessaires • Enregistre les étapes dans un fichier log `/var/log/setup.log`	**Objectifs atteints :** • Mise à jour système • Installation Apache • Déploiement d'une page HTML • Création d'un utilisateur • Activation des services • Logging dans `/var/log/setup.log`

`setup-server.sh` (Bash – pour Ubuntu/Debian)

```bash
#!/bin/bash

LOG_FILE="/var/log/setup.log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "===== Début de la configuration - $(date) ====="

# Mise à jour du système
echo "[1/6] Mise à jour du système..."
apt update && apt upgrade -y

# Installation d'Apache
echo "[2/6] Installation d'Apache..."
apt install -y apache2

# Déploiement d'une page web personnalisée
echo "[3/6] Déploiement de la page d'accueil..."
echo "<h1>Serveur Web configuré automatiquement 🚀</h1>" > /var/www/html/index.html

# Création d'un utilisateur
echo "[4/6] Création de l'utilisateur 'adminweb'..."
useradd -m -s /bin/bash adminweb
echo "adminweb:motdepasse123" | chpasswd

# Activation d'Apache au démarrage
echo "[5/6] Activation d'Apache au démarrage..."
systemctl enable apache2
systemctl restart apache2
```

```bash
# Vérification du statut du service
echo "[6/6] Vérification du statut d'Apache..."
systemctl status apache2 | grep Active
```

🚀 Lancer le script

```bash
chmod +x setup-server.sh
sudo ./setup-server.sh
```

🧪 Vérifications

- Le service Apache est actif :

```bash
sudo systemctl status apache2
```

- La page est accessible :

```bash
curl http://localhost
```

- L'utilisateur a bien été créé :

```bash
id adminweb
```

- Le fichier `/var/log/setup.log` contient toutes les étapes.

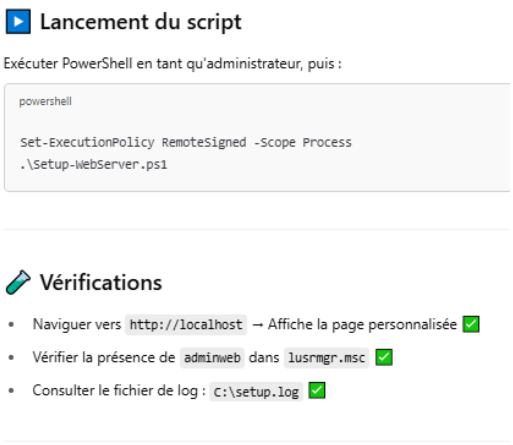
Script PowerShell : Automatisation de la configuration d'un serveur Web IIS

`Setup-WebServer.ps1`

🎯 Objectifs couverts :

- Installer IIS (Internet Information Services)
- Déployer une page HTML
- Créer un utilisateur local avec mot de passe
- Activer le service IIS
- Journaliser les étapes dans un fichier `C:\setup.log`

```powershell
# Emplacement du fichier log
$LogFile = "C:\setup.log"

Function Log {
    param ([string]$message)
    $timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
    "$timestamp - $message" | Out-File -FilePath $LogFile -Append
    Write-Host $message
}

Log "===== Début de la configuration du serveur IIS ====="

# 1. Installer IIS
Log "[1/5] Installation de IIS..."
Install-WindowsFeature -Name Web-Server -IncludeManagementTools

# 2. Déployer une page HTML
Log "[2/5] Création de la page d'accueil..."
$WebPath = "C:\inetpub\wwwroot\index.html"
"<h1>Serveur IIS configuré automatiquement 🎉</h1>" | Set-Content -Path $WebPath

# 3. Créer un utilisateur local
Log "[3/5] Création de l'utilisateur local 'adminweb'..."
$Password = ConvertTo-SecureString "MotDePasse123!" -AsPlainText -Force
New-LocalUser "adminweb" -Password $Password -FullName "Administrateur Web" -Description "Compte créé par script"
Add-LocalGroupMember -Group "Administrators" -Member "adminweb"

# 4. Activer et démarrer le service IIS
Log "[4/5] Activation du service IIS..."
Start-Service W3SVC
Set-Service -Name W3SVC -StartupType Automatic

# 5. Vérifier le statut du service
$serviceStatus = Get-Service -Name W3SVC
Log "[5/5] Statut du service : $($serviceStatus.Status)"

Log "===== Configuration terminée avec succès ====="
```



Module 14 : La vérification de la configuration et de la conformité

Objectifs pédagogiques

À la fin de ce module, vous serez capable de :

- Vérifier que les configurations appliquées correspondent aux attentes
- Détecter les écarts de conformité (drift) sur un serveur
- Utiliser des outils de vérification automatique (audit, test, scan)
- Écrire des scripts de contrôle
- Mettre en place un rapport d'audit régulier

14.1 Pourquoi vérifier la configuration ?

Conformité = configuration + sécurité + bonnes pratiques

Objectifs :

- Éviter les erreurs humaines
- Assurer la reproductibilité sur plusieurs serveurs
- Respecter des normes de sécurité (CIS, ANSSI, ISO 27001…)
- Faciliter les audits

14.2 Éléments à vérifier

Composant	Exemple de vérification
Services	Apache/Nginx/MySQL sont-ils installés & actifs ?
Ports	Le pare-feu autorise-t-il uniquement les ports nécessaires ?
Utilisateurs	Les utilisateurs sont-ils créés avec les bons droits ?
Fichiers de config	`/etc/apache2/apache2.conf` est-il conforme ?

Composant	Exemple de vérification
Tâches cron/scheduled	Présence d'automatisations non prévues ?
Journaux système	Erreurs ou anomalies détectées ?
Packages	Versions à jour, pas de paquets interdits

14.3 Outils et méthodes de vérification

Outil	Utilité principale
Shell / PowerShell	Scripts personnalisés
Ansible	`ansible-playbook --check` (mode dry-run)
AuditD (Linux)	Audit des accès aux fichiers et commandes
OpenSCAP	Analyse de conformité sécurité (CIS, etc.)
Lynis (Linux)	Audit de sécurité système
Windows DSC	Desired State Configuration (PowerShell)

⚙ 14.4 Exemple de script Bash de vérification

```bash
#!/bin/bash
echo "=== Audit de configuration Apache ==="

if systemctl is-active --quiet apache2; then
  echo "Apache est actif ✅"
else
  echo "Apache est inactif ❌"
fi

if grep -q "ServerTokens Prod" /etc/apache2/conf-enabled/security.conf; then
  echo "ServerTokens est correctement configuré ✅"
else
  echo "ATTENTION : ServerTokens doit être mis à 'Prod' ❌"
fi
```

⚙ 14.5 Exemple PowerShell (Windows Server IIS)

```
Write-Host "=== Vérification IIS ==="
if ((Get-Service W3SVC).Status -eq "Running") {
    Write-Host "IIS est actif ✅"
} else {
    Write-Host "IIS est inactif ❌"
}

$homepage = "C:\inetpub\wwwroot\index.html"
if (Test-Path $homepage) {
    Write-Host "Page d'accueil présente ✅"
} else {
    Write-Host "Page d'accueil absente ❌"
}
```

14.6 Génération de rapports d'audit

En Bash :

```
bash check-config.sh > /var/log/config-audit-$(date +%F).log
```

En PowerShell :

```
.\Check-Config.ps1 | Out-File "C:\Logs\config-audit-$(Get-Date -f yyyy-MM-dd).log"
```

14.7 Exercice de fin de module

Créez un script (Bash ou PowerShell) qui :

- Vérifie si un serveur web est installé et actif
- Contrôle l'existence d'une page d'accueil personnalisée
- Vérifie si un utilisateur spécifique existe
- Enregistre le résultat dans un fichier log

Voici la **solution de l'exercice 14.8** sous deux formes :

- ✅ En **Bash** (Linux / Apache)
- ✅ En **PowerShell** (Windows / IIS)

Solution Bash – Vérification serveur Apache (Linux)

```bash
#!/bin/bash

LOGFILE="/var/log/verify-server.log"
exec > >(tee -a "$LOGFILE") 2>&1

echo "===== Vérification du serveur - $(date) ====="

# 1. Vérification du service Apache
echo "[1/3] Vérification du service Apache..."
if systemctl is-active --quiet apache2; then
    echo "✅ Apache est actif"
else
    echo "❌ Apache est inactif"
fi

# 2. Vérification de la page d'accueil
echo "[2/3] Présence de la page /var/www/html/index.html"
if [ -f /var/www/html/index.html ]; then
    echo "✅ Page d'accueil présente"
else
    echo "❌ Page d'accueil absente"
fi

# 3. Vérification de l'utilisateur
echo "[3/3] Vérification de l'utilisateur adminweb..."
if id "adminweb" &>/dev/null; then
    echo "✅ Utilisateur adminweb existe"
else
    echo "❌ Utilisateur adminweb non trouvé"
fi

echo "===== Fin de vérification - $(date) ====="
```

📂 Fichier log : `/var/log/verify-server.log`

▶️ Exécution :

```bash
sudo chmod +x verify-server.sh
sudo ./verify-server.sh
```

Solution PowerShell – Vérification serveur IIS (Windows)

```
$logPath = "C:\Logs\verify-server.log"
New-Item -Path "C:\Logs" -ItemType Directory -Force | Out-Null

Function Log($message) {
    $timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
    "$timestamp - $message" | Out-File -FilePath $logPath -Append
    Write-Host $message
}

Log "===== Début de la vérification IIS ====="

# 1. Service IIS
Log "[1/3] Vérification du service IIS..."
if ((Get-Service W3SVC).Status -eq "Running") {
    Log "✅ IIS est actif"
} else {
    Log "❌ IIS est inactif"
}

# 2. Page d'accueil
Log "[2/3] Présence de la page d'accueil..."
if (Test-Path "C:\inetpub\wwwroot\index.html") {
    Log "✅ Page présente"
} else {
    Log "❌ Page absente"
}
```

```
# 3. Utilisateur adminweb
Log "[3/3] Vérification de l'utilisateur adminweb..."
try {
    $user = Get-LocalUser -Name "adminweb"
    Log "✅ Utilisateur adminweb existe"
} catch {
    Log "❌ Utilisateur adminweb non trouvé"
}

Log "===== Fin de la vérification - $(Get-Date) ====="
```

📂 Fichier log : `C:\Logs\verify-server.log`

▶️ Exécution (en admin) :

```
powershell

.\Verify-Server.ps1
```

Module 15 – La rédaction et la mise à jour de la documentation d'exploitation

Objectifs pédagogiques

À l'issue de ce module, vous serez capable de :

- Comprendre l'importance d'une documentation d'exploitation claire et à jour
- Rédiger une documentation technique utile, réutilisable, et lisible
- Structurer l'information pour différents profils (techniques ou non)
- Mettre en place des méthodes pour la mise à jour continue

15.1 Pourquoi documenter ?

Raisons clés	Bénéfices
Continuité de service	Remplaçabilité des administrateurs
Maintenance plus rapide	Diagnostic facilité
Réduction des erreurs humaines	Moins de dépendance à la mémoire
Traçabilité des modifications système	Meilleure conformité
Transfert de compétences	Formation plus efficace

15.2 Contenu d'une documentation d'exploitation

Structure type recommandée :

1. **Présentation générale**
   - Nom du serveur / service
   - Objectif du système
   - Environnement (dev, test, prod)
2. **Architecture technique**
   - IP, DNS, OS, versions
   - Schéma réseau / VM
3. **Procédures d'installation**
   - Étapes manuelles ou scriptées
   - Chemins exacts
   - Scripts d'automatisation inclus
4. **Procédures de configuration**
   - Fichiers modifiés, paramètres importants
   - Variables d'environnement
5. **Procédures de supervision / contrôle**
   - Vérification du bon fonctionnement
   - Services critiques et commandes associées
6. **Journal des modifications**
   - Changements, mises à jour, correctifs
   - Date, auteur, motif
7. **Procédures de secours / rollback**
   - Sauvegardes
   - Réinstallation rapide
8. **Annexes**
   - Liens utiles, glossaire, captures, commandes fréquentes

✍ 15.3 Bonnes pratiques de rédaction

Conseils	Pourquoi
Utiliser un langage simple et direct	Compréhensible par tous
Être précis et factuel	Pas de place pour l'interprétation
Mettre à jour régulièrement	Une doc obsolète = inutile
Versionner les documents	Suivi des modifications
Ajouter des exemples de commandes	Facilité d'utilisation
Utiliser des modèles standardisés	Homogénéité

🛠 15.4 Outils recommandés

Type	Outils
Traitement texte	Word, LibreOffice, Markdown
Documentation technique	MkDocs, Docusaurus, GitBook
Versionnement	Git (avec GitHub/GitLab)
Wiki collaboratif	Confluence, MediaWiki

📂 15.5 Exemple de documentation d'exploitation (extrait)

```
## Serveur : web-prod-01

**Adresse IP** : 192.168.1.10
**OS** : Ubuntu Server 22.04
**Service** : Apache 2.4 – serveur web principal

### Installation automatisée

Script utilisé : `setup-server.sh`
Chemin : `/opt/scripts/setup-server.sh`

### Vérification du service

```bash
systemctl status apache2
curl http://localhost
```
```

Module 16 : Diagnostic et correction des dysfonctionnements

Objectifs pédagogiques

À la fin de ce module, vous serez capable de :

1. **Identifier et diagnostiquer** les problèmes courants dans un système ou une application.
2. **Utiliser les outils de diagnostic** pour analyser les logs, les performances et l'intégrité des systèmes.
3. **Appliquer des méthodes de correction** pour résoudre les problèmes détectés.
4. **Documenter les incidents** et leur résolution pour un suivi optimal.

16.1 Introduction au diagnostic des dysfonctionnements

Pourquoi diagnostiquer correctement un dysfonctionnement ?

Le diagnostic précis d'un problème permet d'éviter les erreurs de manipulation et de résoudre plus rapidement les incidents. Un diagnostic efficace repose sur :

- **Une bonne collecte de données** : Consulter les logs, les alertes, les erreurs système.
- **Des outils de diagnostic adaptés** : Utiliser des outils spécifiques à l'application ou au système concerné.
- **Un raisonnement structuré** : Analyser les symptômes pour identifier la cause du problème.

Étapes clés du diagnostic :

1. **Identification du problème** : Quelle est la nature du dysfonctionnement ? Quel impact a-t-il ?
2. **Collecte d'informations** : Logs, alertes, messages d'erreur.
3. **Analyse des informations collectées** : Chercher des motifs récurrents ou des indices sur la cause du problème.
4. **Validation des hypothèses** : Tester différentes solutions possibles.
5. **Correction** : Appliquer la solution qui résout le problème.

16.2 Outils de diagnostic

Il existe plusieurs outils pour diagnostiquer les problèmes dans un système informatique. Voici les plus couramment utilisés :

1. Outils pour les systèmes Linux

- `dmesg` : Affiche les messages du noyau, souvent utilisés pour diagnostiquer les problèmes matériels ou les erreurs au démarrage.

  ```
  dmesg | tail
  ```

- `journalctl` : Permet de consulter les logs système pour diagnostiquer les erreurs de services, de processus, ou de réseau.

  ```
  journalctl -xe
  ```

- `top` / `htop` : Affiche l'utilisation des ressources système en temps réel, utile pour repérer des processus gourmands en ressources.
- `netstat` / `ss` : Affiche les connexions réseau et les ports ouverts.

```
ss -tuln
```

- `lsof` : Liste les fichiers ouverts par les processus.

```
lsof
```

2. Outils pour les systèmes Windows

- **Event Viewer** : Utilisé pour visualiser les logs système, de sécurité et d'application sous Windows.
- **Task Manager** : Permet de surveiller les performances du système et d'arrêter les processus qui posent problème.
- **Performance Monitor (perfmon)** : Outil de diagnostic avancé pour surveiller les performances du système.
- **PowerShell** : Outil puissant pour effectuer des diagnostics et automatiser des tâches de correction.

3. Outils pour les réseaux

- `ping` : Permet de tester la connectivité réseau entre votre machine et une autre.

```
ping www.google.com
```

- `traceroute` / `tracert` : Affiche le chemin emprunté par les paquets pour atteindre une destination.

```
traceroute www.google.com
```

- `nslookup` : Permet de résoudre les noms de domaine en adresses IP.

```
nslookup www.google.com
```

- **Wireshark** : Outil de capture et d'analyse de paquets réseau.

⚙ 16.3 Diagnostic des dysfonctionnements courants

1. Problèmes de performance

Les problèmes de performance peuvent être dus à un manque de ressources, à une mauvaise configuration, ou à des logiciels qui consomment excessivement des ressources.

Comment diagnostiquer ?

- **Vérifier l'utilisation du CPU et de la mémoire** avec `top` (Linux) ou **Task Manager** (Windows).
- **Vérifier l'espace disque** avec `df -h` (Linux) ou **Disk Management** (Windows).
- **Analyser les processus gourmands** avec `htop` (Linux) ou **Resource Monitor** (Windows).

Solutions possibles :

- Ajouter de la mémoire ou du stockage.
- Réduire la charge des processus gourmands en ressources.
- Optimiser la configuration des services.

2. Problèmes de réseau

Les problèmes de réseau peuvent être liés à la connectivité, la bande passante, ou la résolution de noms.

Comment diagnostiquer ?

- **Vérifier la connectivité** avec `ping`.
- **Vérifier les ports ouverts** avec `netstat` (Linux) ou `netstat -an` (Windows).
- **Tracer le chemin des paquets** avec `traceroute` (Linux) ou `tracert` (Windows).
- **Vérifier les configurations DNS** avec `nslookup`.

Solutions possibles :

- Redémarrer les services réseau.
- Vérifier la configuration du pare-feu et des routes.
- Contacter votre fournisseur d'accès à Internet en cas de panne générale.

3. Problèmes d'applications (serveurs Web, bases de données, etc.)

Les applications peuvent rencontrer des erreurs dues à des configurations incorrectes, des ressources insuffisantes ou des erreurs logicielles.

Comment diagnostiquer ?

- **Vérifier les logs d'application** : Les logs d'Apache, Nginx, MySQL, etc., fournissent des détails cruciaux.
- **Vérifier les erreurs de configuration** avec `apachectl configtest` (Apache) ou `nginx -t` (Nginx).
- **Utiliser les outils de supervision** comme **Nagios**, **Zabbix**, ou **Prometheus**.

Solutions possibles :

- Vérifier les fichiers de configuration.
- Rechercher des mises à jour ou des patches pour l'application.
- Redémarrer le service ou réinstaller l'application si nécessaire.

🛠️ 16.4 Correction des dysfonctionnements

1. Correction des erreurs de configuration

- **Correction manuelle** : Modifier les fichiers de configuration (par exemple `/etc/nginx/nginx.conf` ou `/etc/httpd/httpd.conf`).
- **Correction via script** : Automatiser les corrections avec des scripts (Bash, PowerShell, Python).

2. Correction des problèmes de performance

- **Optimisation des services** : Modifier les paramètres pour limiter l'utilisation des ressources (par exemple, augmenter les limites de connexions dans un serveur Web).
- **Ajout de ressources** : Si la machine est sous-dimensionnée, ajouter plus de RAM, de CPU ou d'espace disque.
- **Suppression de processus inutiles** : Désactiver ou supprimer des services ou applications non utilisés.

3. Correction des problèmes réseau

- **Redémarrer les interfaces réseau**.
- **Modifier les règles du pare-feu** : Autoriser ou interdire certains ports.
- **Vérification des configurations de route**.

16.5 Documentation des incidents et de leur résolution

Documenter un incident est essentiel pour la gestion future et la prévention d'incidents similaires. Une bonne documentation doit inclure :

1. **Description du problème** : Quel était le symptôme ?
2. **Analyse de la cause** : Quelle a été la cause du dysfonctionnement ?
3. **Solution appliquée** : Quelles mesures ont été prises pour résoudre le problème ?
4. **Actions préventives** : Que peut-on faire pour éviter que le problème ne se reproduise ?
5. **Journal des modifications** : Lorsque le problème a-t-il été résolu ? Qui a effectué l'intervention ?

16.6 Exercice de fin de module

Objectif : Vous avez un serveur Apache qui ne répond plus correctement aux requêtes HTTP. Diagnostiquez et corrigez le problème.

1. **Vérification des logs Apache** : Consultez `/var/log/apache2/error.log.`
2. **Vérification de l'état du service Apache** : Utilisez `systemctl status apache2.`
3. **Vérification des configurations Apache** : Utilisez `apachectl configtest.`
4. **Redémarrez Apache** : Si une erreur est détectée, appliquez la solution et redémarrez le service.
5. **Vérification du site web** : Testez l'accessibilité du site avec `curl`.

Module 17 : La consultation de la documentation technique rédigée en anglais

Objectifs pédagogiques

À la fin de ce module, vous serez capable de :

- Lire, comprendre et exploiter efficacement la documentation technique rédigée en anglais.
- Identifier et extraire les informations pertinentes dans un manuel, une page de documentation ou une base de connaissances.
- Utiliser la terminologie informatique anglaise dans un contexte professionnel.
- Naviguer sur les sites de référence technique (Microsoft Learn, Linux man pages, Stack Overflow, etc.).

17.1 Pourquoi lire la documentation en anglais ?

L'anglais est **la langue internationale de l'informatique**. La majorité des documentations officielles, tutoriels, forums techniques, guides d'installation, et API sont rédigés en anglais.

Avantages :

- Accès aux **informations les plus à jour**.
- Meilleure compréhension des **messages d'erreur**.
- Autonomie dans la **résolution de problèmes**.

📖 17.2 Types de documentation technique

🔹 Documentation officielle

- Rédigée par les éditeurs de logiciels ou développeurs (ex : Microsoft, Red Hat, Python, Docker...).
- Très structurée : introduction, installation, configuration, bonnes pratiques, dépannage.

🔹 Manuels en ligne / man pages

- Disponibles directement dans les systèmes Linux via `man`.

```
man ls
```

🔹 Forums techniques

- Exemples : Stack Overflow, Server Fault, Reddit, Super User.
- Très utile pour des problèmes spécifiques.

🔹 Guides et blogs

- Moins formels mais souvent très pédagogiques.
- Ex. : DigitalOcean Community, Medium, Dev.to, blogs de développeurs.

🧠 17.3 Outils pour faciliter la lecture

📌 Vocabulaire technique de base

Voici quelques mots-clés à connaître :

Anglais	Français
Setup / Install	Installation
Troubleshooting	Dépannage
Issue / Bug	Problème / Anomalie
Output	Résultat / Sortie
Command	Commande
Configuration	Paramétrage
Step / Procedure	Étape / Procédure
Requirement	Prérequis

📌 Astuces

- Utilisez **DeepL** ou **Google Translate** pour des phrases complexes.
- Lisez **le sommaire ou les titres** pour naviguer rapidement.
- Recherchez les **sections “Getting Started”, “Usage”, “Examples”**.

17.4 Étude de cas : Lire une documentation officielle

Exemple : Documentation NGINX

- Page d'accueil : https://nginx.org/en/docs/
- Objectif : comprendre comment configurer un proxy inverse.

Extrait :

```
To configure NGINX as a reverse proxy:
1. Use the proxy_pass directive inside a location block.
2. Define the backend server (e.g., http://127.0.0.1:3000).
3. Ensure that the correct Host headers are passed.
```

Traduction :

Pour configurer NGINX en tant que proxy inverse :

1. Utilisez la directive `proxy_pass` dans un bloc `location`.
2. Définissez le serveur en arrière-plan (ex : http://127.0.0.1:3000).
3. Assurez-vous que les bons en-têtes `Host` sont transmis.

Module 18 : Révision des principes de la virtualisation

Objectifs pédagogiques

À la fin de ce module, vous serez capable de :

- Comprendre le **principe de la virtualisation** et son intérêt dans les infrastructures IT modernes.
- Identifier les différents types de virtualisation.
- Distinguer les hyperviseurs de type 1 et type 2.
- Connaître les outils de virtualisation les plus répandus.
- Créer et gérer une machine virtuelle (VM) dans un environnement simulé.

18.1 Qu'est-ce que la virtualisation ?

La virtualisation consiste à créer une version virtuelle (plutôt que physique) de quelque chose, comme un système d'exploitation, un serveur, un périphérique de stockage ou des ressources réseau.

18.2 Types de virtualisation

Type de virtualisation	Description
Virtualisation de serveurs	Création de plusieurs machines virtuelles sur un serveur physique.
Virtualisation de postes	Hébergement d'environnements de bureau accessibles à distance.
Virtualisation du stockage	Abstraction du stockage physique en une seule unité logique.
Virtualisation réseau	Création de réseaux virtuels indépendants du matériel physique.

❓ 18.3 Les hyperviseurs

Un hyperviseur est un logiciel qui permet de créer et de gérer des machines virtuelles.

🔻 Hyperviseur de type 1 (bare-metal)

Installé directement sur le matériel.

- Exemples : **VMware ESXi**, **Microsoft Hyper-V**, **Xen**, **Proxmox**.

🔻 Hyperviseur de type 2 (hosted)

Installé sur un système d'exploitation.

- Exemples : **VirtualBox**, **VMware Workstation**, **KVM (via QEMU)**.

💻 18.4 Avantages de la virtualisation

- 💸 **Réduction des coûts matériels**
- ⚙️ **Meilleure utilisation des ressources**
- 🔁 **Facilité de sauvegarde et restauration**
- ❓ **Environnements de test isolés**
- 🚀 **Déploiement plus rapide de services**
- 🔐 **Sécurité et isolation des VM**

🏢 18.5 Cas d'usage dans un environnement professionnel

- **Environnement de test** : Déployer rapidement plusieurs systèmes pour du développement.
- **Infrastructure virtualisée** : Utiliser Proxmox pour créer un cluster de VM pour les différents services d'une entreprise.
- **Sauvegarde/restauration** : Snapshot d'une VM avant une mise à jour critique.

🛠️ 18.6 Outils populaires de virtualisation

Outil	Type d'hyperviseur	Plateforme
VMware ESXi	Type 1	Serveur
VirtualBox	Type 2	PC (Windows/Linux)
Proxmox VE	Type 1	Open source (Debian)
Hyper-V	Type 1/2	Windows
KVM + QEMU	Type 1	Linux

🔁 18.7 Exemples pratiques

Exemple 1 : Création d'une VM avec VirtualBox

```
VBoxManage createvm --name "Ubuntu-Test" --register
VBoxManage modifyvm "Ubuntu-Test" --memory 2048 --cpus 2
VBoxManage createhd --filename ~/VirtualBox\ VMs/Ubuntu-Test.vdi --size 20000
```

Exemple 2 : Création d'une VM avec KVM + cloud-init

```
virt-install \
--name ubuntu-cloud \
--memory 2048 \
--vcpus 2 \
--disk path=/var/lib/libvirt/images/ubuntu.qcow2,size=10 \
--cdrom /iso/ubuntu.iso \
--network network=default \
--os-type linux \
--graphics none
```

🧑‍🔬 18.8 Mini-projet de fin de module (facultatif)

Objectif : Créez une machine virtuelle sous Linux à l'aide d'un hyperviseur de votre choix (VirtualBox, Proxmox, ou KVM), et :

1. Installez un système Linux (Ubuntu Server ou CentOS). 2. Donnez un nom personnalisé à la VM. 3. Allouez 2 Go de RAM et 10 Go de disque. 4. Démarrez la machine et accédez-y via SSH ou console.	🎯 **Objectif :** Créer une machine virtuelle Linux avec : • 2 Go de RAM • 10 Go de disque • Nom personnalisé • Installation d'un système Ubuntu Server • Accès à la console ou via SSH

✅ Étapes de réalisation

📌 1. Se connecter à l'interface web de Proxmox

Accédez à l'interface via votre navigateur :

🔑 **`https://<IP_du_serveur_proxmox>:8006`**

Connectez-vous avec votre compte administrateur (`root@pam`).

📌 2. Téléchargement de l'ISO Ubuntu Server

Depuis l'interface Proxmox :

- Allez dans **"Datacenter > Local (storage) > ISO Images"**
- Cliquez sur **"Upload"** pour envoyer l'image ISO de **Ubuntu Server** (ex: `ubuntu-22.04-live-server-amd64.iso`)
- Vous pouvez aussi le télécharger directement avec :

```
cd /var/lib/vz/template/iso
wget https://releases.ubuntu.com/22.04/ubuntu-22.04-live-server-amd64.iso
```

📌 3. Créer une nouvelle machine virtuelle

- Dans l'interface web : **Create VM**
- Onglet **General** :
  - Node : sélectionnez votre serveur Proxmox
  - VM ID : auto-généré
  - Name : `Ubuntu-Test-VM`
- Onglet **OS** :
  - ISO Image : sélectionnez l'image Ubuntu Server téléchargée
  - Type : Linux
- Onglet **System** :
  - BIOS : Default
  - Machine : q35 ou i440fx (selon préférence)
- Onglet **Hard Disk** :
  - Bus/Device : SCSI ou VirtIO
  - Disk Size : `10 GiB`
- Onglet **CPU** :
  - Cores : `2`
- Onglet **Memory** :
  - RAM : `2048 MB`
- Onglet **Network** :
  - Bridge : `vmbr0` (généralement le pont réseau par défaut)
  - Model : VirtIO (recommandé)

✅ Terminez avec **"Finish"**

📌 4. Démarrage de la VM et installation d'Ubuntu

- Allez sur la nouvelle VM > **Console**
- Lancez la machine > **Démarrer**
- Suivez le processus d'installation classique d'Ubuntu Server (choisissez LVM ou ext4 simple).
- Configurez le réseau et l'utilisateur.

📌 5. (Optionnel) Activer l'accès SSH

Pendant l'installation d'Ubuntu, cochez l'option :
✅ *Install OpenSSH Server*

Après l'installation :

- Accédez à la console Proxmox
- Connectez-vous à Ubuntu
- Vérifiez que le SSH fonctionne :

```
sudo systemctl status ssh
```

- Depuis un autre poste :

```
ssh votre_utilisateur@adresse_IP_de_la_VM
```

⚙ Provisionnement automatisé avec cloud-init sur Proxmox

🎯 Objectifs

- Créer une VM Ubuntu Server avec **cloud-init** pour :
  - Définir automatiquement un nom d'hôte
  - Créer un utilisateur avec un mot de passe ou une clé SSH
  - Configurer le réseau
  - Exécuter des scripts au démarrage

🎁 Prérequis

- ISO Ubuntu Server (cloud image) : `.img` ou `.qcow2`
- Proxmox installé et configuré
- Accès root à l'interface web ou SSH

📥 1. Télécharger l'image Ubuntu Cloud

```
cd /var/lib/vz/template/qcow2
wget https://cloud-images.ubuntu.com/jammy/current/jammy-server-cloudimg-amd64.img
```

Renommez le fichier si besoin :

```
mv jammy-server-cloudimg-amd64.img ubuntu-22.04-cloud.qcow2
```

📄 2. Créer un modèle de base (template)

```
qm create 9000 --name ubuntu-cloud --memory 2048 --cores 2 --net0 virtio,bridge=vmbr0
qm importdisk 9000 ubuntu-22.04-cloud.qcow2 local-lvm
qm set 9000 --scsihw virtio-scsi-pci --scsi0 local-lvm:vm-9000-disk-0
qm set 9000 --ide2 local-lvm:cloudinit
qm set 9000 --boot c --bootdisk scsi0
qm set 9000 --serial0 socket --vga serial0
qm template 9000
```

Cela crée un template cloud-init prêt à cloner.

🖱 3. Créer une VM à partir du template

```
qm clone 9000 101 --name ubuntu-test-vm
```

✍ 4. Définir les options cloud-init

```
qm set 101 --ciuser devops --cipassword secret123
qm set 101 --ipconfig0 ip=192.168.100.50/24,gw=192.168.100.1
qm set 101 --nameserver 8.8.8.8 --searchdomain example.local
qm set 101 --sshkeys "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC..."
```

☞ Vous pouvez remplacer `--cipassword` par une **clé SSH uniquement**, pour plus de sécurité.

🚀 5. Démarrer la VM

```
qm start 101
```

🛠 6. (Optionnel) Ajouter un script d'initialisation

Si vous voulez exécuter un script au premier boot (cloud-init `user-data`) :

```
qm set 101 --cicustom "user=local:snippets/cloudinit-user.yml"
```

Créez ensuite le fichier :

```
nano /var/lib/vz/snippets/cloudinit-user.yml
```

Exemple de contenu :

```
#cloud-config
packages:
  - nginx
runcmd:
  - systemctl enable nginx
  - systemctl start nginx
```

✓ Résultat

- La VM démarre automatiquement avec :
  - Réseau configuré
  - Utilisateur prêt avec mot de passe ou clé SSH
  - Logiciels préinstallés
  - Nginx démarré automatiquement

Module 19 : La connaissance du principe de base des calculs utilisés dans l'adressage IP

📃 Objectifs pédagogiques :

À l'issue de ce module, vous serez capable de :

- Comprendre la structure d'une adresse IPv4
- Calculer un masque de sous-réseau
- Identifier le réseau, le broadcast, et la plage d'adresses utilisables
- Comprendre la notation CIDR
- Appliquer les calculs d'adressage IP dans un contexte d'automatisation réseau

---

⍰ 1. Qu'est-ce qu'une adresse IP ?

Une adresse IPv4 est une **adresse logique** composée de 4 octets (32 bits), représentés sous forme décimale séparée par des points. **Exemple** : `192.168.1.10`

Chaque octet varie de 0 à 255.
En binaire : `11000000.10101000.00000001.00001010`

🔠 2. Composition : Adresse IP + Masque de sous-réseau

Élément	Exemple
Adresse IP	`192.168.1.10`
Masque	`255.255.255.0`
Notation CIDR	`192.168.1.10/24`

Le masque de sous-réseau sert à :

- Séparer la **partie réseau** et la **partie hôte** de l'adresse.
- Définir la taille d'un sous-réseau.

📐 3. Calculs à connaître

📌 a. Nombre d'adresses dans un sous-réseau

Formule : `2^(32 - nombre_de_bits_du_masque)`

- `/24` → 2^8 = **256 adresses**
- `/30` → 2^2 = 4 adresses (2 utilisables)

📌 b. Adresse réseau, broadcast, plage d'hôtes

Exemple : `192.168.1.10/24`

- Masque : `255.255.255.0`
- Adresse réseau : `192.168.1.0`
- Adresse de broadcast : `192.168.1.255`
- Plage d'hôtes utilisables : `192.168.1.1 → 192.168.1.254`

📌 c. Notation CIDR

CIDR = Classless Inter-Domain Routing
Indique le **nombre de bits à 1 dans le masque**.
Ex : `/24` = `255.255.255.0`

🎯 4. Classes d'adresses IP (théoriques)

Classe	Plage IP	Masque par défaut	Utilisation
A	1.0.0.0 – 126.255.255.255	255.0.0.0	Grandes organisations
B	128.0.0.0 – 191.255.255.255	255.255.0.0	Moyennes entreprises
C	192.0.0.0 – 223.255.255.255	255.255.255.0	Petites entreprises

⚠ De nos jours, **CIDR** a remplacé les classes pour plus de flexibilité.

🛠 5. Calcul rapide de sous-réseaux (exemple Bash)

Voici un petit script Bash pour convertir une adresse IP en binaire :

```
ip="192.168.1.10"
IFS=. read -r i1 i2 i3 i4 <<< "$ip"
printf "%08b.%08b.%08b.%08b\n" $i1 $i2 $i3 $i4
```

🔄 6. Application dans l'automatisation

- Génération de plans d'adressage IP en script (Python, Bash)
- Calcul automatique d'un sous-réseau pour une machine virtuelle
- Détection de conflits IP dans des déploiements automatisés

Exercice de fin de module – Module 19

❓ Objectif : Appliquer les calculs d'adressage IP à un scénario réseau

📝 Énoncé :

Vous êtes en train de configurer un réseau pour une petite entreprise.
Le réseau principal est `192.168.10.0/24`.

1. Calculez :
    - a. Le **nombre total d'adresses IP**
    - b. Le **nombre d'adresses utilisables**
    - c. L'**adresse réseau**
    - d. L'**adresse de broadcast**
    - e. La **plage des hôtes utilisables**
2. Le client souhaite 4 sous-réseaux identiques à partir de ce réseau.
   Donnez :
    - a. Le **nouveau masque** CIDR
    - b. La **plage IP de chaque sous-réseau** (réseau, plage utilisable, broadcast)

✅ Solution :

🔹 1. Réseau `192.168.10.0/24`

- a. **Nombre total d'adresses** = 2^8 = **256**
- b. **Adresses utilisables** = 256 – 2 = **254**
- c. **Adresse réseau** = `192.168.10.0`
- d. **Adresse broadcast** = `192.168.10.255`
- e. **Plage hôtes utilisables** = `192.168.10.1` → `192.168.10.254`

🔹 2. Diviser en 4 sous-réseaux

Pour diviser un `/24` en 4 sous-réseaux, il faut **2 bits supplémentaires** ($2^2 = 4$).
Donc, **nouveau masque : `/26`** (soit `255.255.255.192`)

Chaque sous-réseau contient 64 adresses (2^6), donc 62 adresses utilisables.

Détail des 4 sous-réseaux `/26` :

Sous-réseau	Adresse réseau	Plage hôtes utilisables	Broadcast
#1	192.168.10.0/26	192.168.10.1 → 192.168.10.62	192.168.10.63
#2	192.168.10.64/26	192.168.10.65 → 192.168.10.126	192.168.10.127
#3	192.168.10.128/26	192.168.10.129 → 192.168.10.190	192.168.10.191
#4	192.168.10.192/26	192.168.10.193 → 192.168.10.254	192.168.10.255

🎓 Bonus : Script Bash pour découpage réseau

```
#!/bin/bash
ipcalc 192.168.10.0/24 -s 64 64 64 64
```

Ou version Python avec **`ipaddress`** :

```python
import ipaddress

network = ipaddress.ip_network("192.168.10.0/24")
subnets = list(network.subnets(new_prefix=26))

for i, subnet in enumerate(subnets, 1):
    print(f"Subnet #{i}: {subnet}")
    print(f"  Hosts: {list(subnet.hosts())[0]} → {list(subnet.hosts())[-1]}")
    print(f"  Broadcast: {subnet.broadcast_address}\n")
```