IAAS

Administration, Déploiement & Introduction aux Conteneurs

Loïc Rouquette

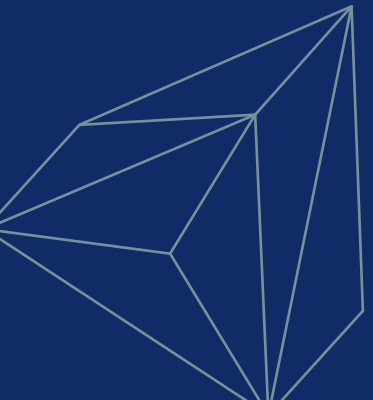
Sommaire

Objectifs de la séance

À la fin du cours, vous devrez être capable de :

- gérer les aspects fondametnaux des machines virtuelles en environnement IaaS, la sélection des types d'instances appropriées et l'administration du stockage bloc ;
- appliquer les étapes séquentielles d'un déploiement manuel d'une application web simple ;
- expliquer les concepts clés de la conteneurisation et de Docker. Utiliser les commandes Docker essentielles pour construire, exécuter et gérer des applications conteneurisées simples ;
- identifier les modèles de tarifications courants en IaaS (paiement à l'usage, instances réservées, instances Spot).

Gestion des VMs et du stockage



1.1. Le Cycle de Vie d'une VM en IaaS

Cycle de Vie d'une VM :

Le cycle de vie d'une VM (Virtual Machine) en IAAS (Infrastructure as a Service) représente l'ensemble des étapes par lesquelles une VM passe, depuis sa création jusqu'à sa suppression définitive.

Importance :

- Assurer le bon fonctionnement des services hébergés.
- Maîtriser les coûts associés aux ressources cloud.

Risques d'une gestion inadéquate :

- Dépenses inutiles.
- Vulnérabilités de sécurité.

Le Cycle de Vie d'une VM : Vue d'ensemble

- Création (Provisionnement)
- Configuration
- États Opérationnels (Gestion Courante)
- Gestion et Surveillance
- Terminaison (Suppression)

Cycle de Vie d'une VM : 1. Création (Provisionnement)

- Début du cycle.
- Allocation des ressources : CPU, RAM, stockage, réseau.
- Selon les spécifications choisies par l'utilisateur.
- Crucial : doit être adapté aux besoins de la charge de travail (workload).

Création : Méthodes de Provisionnement

Manuellement

- Via la console web du fournisseur IaaS ;
- Intuitif, visuel ;
- Adapté pour les besoins simples ou apprentissage.

Automatiquement

- Via CLI, APIs du fournisseur ;
- Outils d'Infrastructure as Code (IaC) : Terraform, Ansible.

Création : Avantages de l'Automatisation

- Déployer rapidement des environnements complexes.
- Garantir la cohérence entre les instances.
- Réduire les erreurs humaines.
- Permettre la scalabilité.

Création : Bonnes Pratiques

- Utilisation d'images VM :
  - Préconfigurées par le fournisseur.
  - Personnalisées par l'utilisateur.
- Assure la cohérence des configurations initiales.

Création : État Transitoire

- Pendant l'allocation des ressources et le démarrage.
- État souvent appelé `pending` (en attente).
- Avant d'atteindre l'état `running` (en cours d'exécution).

Cycle de Vie : 2. Configuration

- Nécessaire une fois la VM créée et démarrée.
- Actions :
  - Configuration fine du système d'exploitation (OS).
  - Installation des applications et dépendances logicielles.
  - Mise en place des paramètres réseau spécifiques (IP, pare-feu via groupes de sécurité).
  - Configuration des accès utilisateurs et permissions.

Configuration : Automatisation Initiale

- Utilisation de scripts de configuration (user data).
  - Exemple : cloud-init pour Linux.
- Exécutés automatiquement au premier démarrage de la VM.

Cycle de Vie : 3. États Opérationnels (Gestion Courante)

Une VM en fonctionnement peut passer par plusieurs états :

- `Running` (En cours d'exécution)
- `Stopping` / `Starting` (Arrêt / Démarrage)
- `Rebooting` (Redémarrage)
- `Force Stop` (Arrêt Forcé)

États Opérationnels : Running

- La VM est active et opérationnelle.
- Les applications et services fonctionnent.

États Opérationnels : Stopping / Starting (1/3)

- **Arrêt (`Stopping`) :**
  - Généralement une extinction propre de l'OS.
  - Peut être redémarrée (`Starting`) ultérieurement.
- Pratique clé pour l'optimisation des coûts (paiement à l'usage).
  - Frais de calcul cessent.
  - Frais de stockage persistent pour les volumes attachés.

États Opérationnels : Stopping / Starting (2/3)

- Une VM arrêtée conserve généralement :
  - Son identifiant unique.
  - Son adresse IP privée.
  - Ses volumes de stockage persistants attachés.

États Opérationnels : Stopping / Starting (3/3)

- Ressources éphémères pouvant être perdues (sauf configuration spécifique) :
  - Stockage d'instance.
  - Certaines adresses IP publiques.
- Solutions : IP élastiques (AWS), tags spécifiques (Outscale osc.fcu.eip.auto-attach).

États Opérationnels : Rebooting

- Équivalent à un redémarrage matériel.
- Relance l'OS sans modifier la configuration persistante de la VM.

Cycle de Vie : 4. Gestion et Surveillance

- Processus continu.
- Inclut :
  - Surveillance des performances (CPU, RAM, disque, réseau).
  - Application des mises à jour et correctifs de sécurité (OS, applications).
  - Gestion continue des accès et règles de pare-feu.
  - Évaluation régulière de l'utilisation des ressources (adéquation besoins/budget).
- Fournisseurs IaaS proposent souvent des outils intégrés.

Cycle de Vie : 5. Terminaison (Suppression)

- Dernière étape du cycle.
- Supprime définitivement la VM et ses ressources éphémères associées.
- États transitoires : `shutting-down` -> `terminated`.
- VM généralement irrécupérable après un court délai.

Terminaison : Précautions

- Crucial : S'assurer que toutes les données nécessaires ont été sauvegardées ou migrées AVANT la terminaison.
- La suppression de la VM ne supprime pas automatiquement les volumes de stockage persistants attachés.
  - Ceux-ci doivent être gérés séparément (détachés et éventuellement supprimés) pour éviter des frais inutiles.

Terminaison : Comportement configurable

- Certains fournisseurs permettent de configurer le comportement lors d'un arrêt initié depuis l'OS.
  - Exemple : `InstanceInitiatedShutdownBehavior` chez AWS/Outscale (peut être configuré pour terminer automatiquement la VM).

Importance Stratégique de la Gestion du Cycle de Vie

- Pas une simple succession d'opérations techniques.
- Démarche stratégique continue.
- Objectif : Aligner constamment les ressources déployées avec :
  - Exigences de performance.
  - Contraintes budgétaires.
  - Maintien d'une posture de sécurité adéquate.

Implications du Cycle de Vie : Coûts et Sécurité

- **Coûts :**
  - Ressources actives.
  - Stockage persistant.
- **Sécurité :**
  - Mises à jour.
  - Configuration des accès.
  - Suppression sécurisée.

Risques d'une Gestion Déficiente du Cycle de Vie

- Oubli de terminer des VMs inutilisées.
- Manque de surveillance de l'utilisation.
- Conséquences :
  - Dépassements de coûts imprévus.
  - Failles de sécurité potentielles.

Bonnes Pratiques de Gouvernance du Cycle de Vie

- Adopter des politiques claires (ex: utilisation de tags pour organisation et facturation).
- Implémenter des mécanismes de surveillance.
- Automatiser autant que possible les tâches de gestion (arrêts programmés, alertes d'utilisation).

1.2. Types d'Instances

- Fournisseurs IaaS proposent une vaste gamme de "types", "familles" ou "séries" d'instances VM.
- Conçus pour répondre à différents besoins en optimisant l'équilibre entre :
  - Puissance de calcul (CPU)
  - Mémoire vive (RAM)
  - Capacité et performance du stockage
  - Bande passante réseau
- Choisir le type approprié est fondamental (performance applicative et coûts).

Catégories Générales d'Instances (1/2)

- **Usage Général (General Purpose) :**
  - Équilibre CPU, RAM, réseau.
  - Cas d'usage : Serveurs web, dev/test, BDD petites/moyennes, microservices.
  - Exemples : AWS M, T ; Azure D, B ; GCP E2, N2.
- **Optimisées pour le Calcul (Compute Optimized) :**
  - Ratio CPU/RAM élevé.
  - Cas d'usage : HPC, batch processing, encodage multimédia, serveurs de jeux.
  - Exemples : AWS C ; Azure F ; GCP C2, C2D.

Catégories Générales d'Instances (2/2)

- **Optimisées pour la Mémoire (Memory Optimized) :**
  - Grande quantité de RAM par vCPU.
  - Cas d'usage : BDD volumineuses, caches en mémoire (Redis), analyse Big Data temps réel.
  - Exemples : AWS R, X ; Azure E, M ; GCP M1, M2, M3.

Autres Catégories d'Instances

- **Optimisées pour le Stockage (Storage Optimized) :**
  - Haut débit disque (IOPS) et/ou grande capacité de stockage local (souvent SSD NVMe).
  - Cas d'usage : Data warehouses, BDD distribuées (faible latence).
- **Calcul Accéléré (Accelerated Computing) :**
  - Intègrent des accélérateurs matériels (GPU, FPGA).
  - Cas d'usage : Entraînement Machine Learning, rendu graphique.

Variété au sein des Familles d'Instances

- Nombre de vCPUs.
- Quantité de RAM (souvent GiB par vCPU).
- Type et performance du stockage attaché (SSD vs HDD, local vs réseau).
- Bande passante réseau garantie.
- Matériel spécialisé.
- Certains fournisseurs (ex: GCP) permettent des types personnalisés (vCPU/RAM).

Sélection du Type d'Instance : Un Équilibre Crucial

- **Sur-provisionner :**
  - Instance plus puissante que nécessaire.
  - Gaspillage financier (calcul = poste de dépense important).
- **Sous-provisionner :**
  - Instance trop faible.

  -Mauvaises performances applicatives, expérience utilisateur dégradée.

Évolution Technologique et Complexité

- Multiplication des options (différentes générations de processeurs : ARM, Intel, AMD).
- Volonté des fournisseurs de s'adapter à des charges de travail spécifiques.
- Objectif : Optimiser le rapport prix/performance.
- Complexité croissante : un choix optimal aujourd'hui peut ne plus l'être demain.
  - Nécessite veille technologique et évaluation continue (performances, coûts).

Tableau Récapitulatif des Catégories d'Instances

Catégorie	Caractéristique clé	Cas d'Usage Typiques	Séries Exemples (AWS/Azure/GCP)
Usage Général	Équilibre CPU/RAM/Réseau	Serveurs Web, Dev/Test, Base de données (petite/moyenne), Microservices, Bureaux Virtuels, etc.	M, T / D, B / E2, N2
Optimisée pour le calcul	Ratio CPU/RAM élevé	HPC, Batch Processing, Encodage média, Serveurs de jeux, Modélisation Scientifique	C / F / C2, C2D
Optimisée pour la mémoire	Ratio RAM / CPU élevé	Bases de données volumineures, Caches en mémoire, Analyse Big Data temps réel, SAP HANA	R, X / E, M / M1, M2, M3
Optimisée pour le Stockage	Haut débit / IOPS disque	Data warehouses, Base de données distribuées (faible latence), Systèmes de fichiers large	I, D / L / Z3 (avec disques locaux SSD)
Calcul Accéléré	Présense de GPU/FPGA	Machine Learning (entraînement), Rendu graphique, Calcul scientifique intensif	P, G / N / A2, G2

1.3. Gestion du Stockage Bloc

- Exemples : Amazon EBS, Azure Managed Disks, Google Persistent Disks.
- Fournit des volumes de stockage persistants, attachés au réseau.
- Se comportent comme des disques durs virtuels pour les VMs.
- Caractéristique essentielle : Persistance indépendante du cycle de vie de la VM.
  - Si VM arrêtée ou terminée, les données sur les volumes persistent.

Gestion du Stockage Bloc : Attachement de Volumes (1/3)

Prérequis :

1. Volume doit être `available` (disponible, non attaché).
2. Volume et instance cible dans la même zone de disponibilité (AZ).
3. Certains volumes chiffrés nécessitent des types d'instances compatibles.

Action :

4. Via console ou CLI : sélectionner le volume, choisir "Attacher".

Gestion du Stockage Bloc : Attachement de Volumes (2/3)

Configuration :

5. Sélectionner l'instance cible (même AZ).
6. Spécifier un nom de périphérique (suggéré ou personnalisé, ex: `/dev/sdf`).
   - Nom vu par l'hyperviseur ; l'OS peut le mapper différemment (ex: `/dev/nvme1n1`).
7. Valider l'attachement. Le volume passe à l'état `in-use`.

Gestion du Stockage Bloc : Attachement de Volumes (3/3) - Étape OS

Configuration dans l'OS (Cruciale) : L'attachement IaaS ne suffit pas. Connexion à la VM (SSH/RDP) pour :

1. Identifier nom de périphérique OS (ex: `lsblk`).
2. Vérifier si système de fichiers existe (ex: `sudo file -s /dev/xvdf`).
3. Si nouveau volume, créer système de fichiers (formater) (ex: `sudo mkfs.ext4 /dev/xvdf`).
4. Créer un point de montage (répertoire vide) (ex: `sudo mkdir /data`).
5. Monter le volume (ex: `sudo mount /dev/xvdf /data`).
6. Assurer persistance au redémarrage (ex: modifier `/etc/fstab` sous Linux).

Gestion du Stockage Bloc : Détachement de Volumes (1/2)

Prérequis OS (Impératif) :

1. Démonter le volume au niveau de l'OS AVANT de le détacher au niveau IaaS.
   - Linux : `sudo umount /data`.
   - Windows : mettre disque hors ligne (Gestion des disques).
   - Évite perte ou corruption de données en attente d'écriture.
2. Pour volumes racine (OS), arrêter généralement l'instance avant détachement.

Gestion du Stockage Bloc : Détachement de Volumes (2/2)

Action :

3. Via console ou CLI : sélectionner le volume (`in-use`), choisir “Détacher”.
4. Valider. Le volume retourne à l’état `available`.

Option Risquée :

5. Détachement forcé existe mais ne garantit pas cohérence des données (à n’utiliser qu’en cas d’instance défaillante).

Interaction IaaS / OS pour le Stockage Bloc

- Gestion du stockage bloc = interaction constante entre :
  - Plan de contrôle IaaS (attacher/détacher).
  - Configuration au niveau de l'OS invité (formater, monter, fstab).
- Négliger l'étape OS peut rendre volumes inutilisables ou causer pertes de données.

Gestion du Stockage Bloc : Instantanés de Volume (Snapshots) - Concept

- Sauvegardes ponctuelles (point-in-time) d'un volume de stockage bloc.
- Capturent l'état du volume à un instant T.
- Généralement stockés de manière redondante et durable (souvent sur stockage objet type S3).

Snapshots : Création (1/2)

- Opération lancée depuis console ou CLI.
- Possible même si volume attaché et en cours d'utilisation.
- **Recommandation pour cohérence applicative (BDD) :**
  - Suspendre opérations d'écriture sur le volume.
  - Voire démonter temporairement avant création.
  - Solutions avancées : mécanismes de quiescence applicative.

Snapshots : Création (2/2)

- Premier snapshot : copie complète.
- Snapshots suivants : incrémentiels (seuls les blocs modifiés depuis le dernier snapshot sont sauvegardés).
  - Rend snapshots rapides et économes en stockage (et coût).
- Action : “Créer un snapshot”, sélectionner volume source, ajouter description/tags.

Snapshots : Utilité (1/2)

- **Sauvegarde et Restauration :**
  - Restaurer état d'un volume à un point antérieur en créant un nouveau volume à partir du snapshot.
  - Base de la reprise après sinistre (Disaster Recovery).
- **Création de Nouveaux Volumes :**
  - Un snapshot peut servir de modèle pour créer de nouveaux volumes pré-remplis.

Snapshots : Utilité (2/2)

- **Migration :**
  - “Déplacer” un volume vers une autre AZ (créer snapshot, restaurer dans nouvelle AZ).
- **Redimensionnement :**
  - Peut être utilisé pour redimensionner (créer snapshot, créer nouveau volume plus grand, attacher).

Snapshots : Avantages

- Nature incrémentielle.
- Utilisation de stockage objet économique en backend.
- Méthode de sauvegarde et reprise après sinistre souple et rentable.
- Élément fondamental de la gestion des données dans le cloud IaaS.

1.4. Images VM Personnalisées (Custom Images)

- **Définition :** Modèle de machine virtuelle créé par un utilisateur à partir d'une VM préalablement configurée.
- **Encapsule :** OS, logiciels installés, configurations système, mises à jour, potentiellement code applicatif.
- **Objectif :** Accélérer et standardiser le déploiement de nouvelles VMs.

Images VM Personnalisées : Processus de Création (1/2)

1. **Préparation de la VM Source :** Lancer une instance VM à partir d'une image de base.
2. **Configuration :** Installer logiciels, appliquer MàJ sécurité, configurer système, installer dépendances applicatives.
3. **Généralisation (Cruciale) :** Supprimer informations spécifiques à l'instance source pour que nouvelles VMs aient identité unique. **Suppression SID (Windows), reset nom d'hôte, suppression clés SSH hôte, etc.** Outils : `sysprep` (Windows), `cloud-init` (Linux).

Images VM Personnalisées : Processus de Création (2/2)

4. **Arrêt de la VM :** Arrêter proprement la VM source généralisée.
5. **Création de l'Image :** Utiliser outils fournisseur (Console/CLI) pour créer image à partir du disque système de la VM arrêtée.
   - Image stockée dans catalogue du fournisseur (ex: AMI AWS, Azure Compute Gallery).

Avantages des Images Personnalisées

- **Rapidité de Déploiement :** VMs démarrent plus vite avec environnement requis.
- **Cohérence et Standardisation :** Toutes instances partagent même config de base, versions logicielles, paramètres sécurité.
- **Réduction “Dérive de Configuration” :** Maintient état standardisé, minimise modifs manuelles post-déploiement.
- **Conformité et Sécurité :** Permet d’intégrer configurations de sécurité renforcées (“hardening”) et outils de conformité dans l’image de base.

Principe Clé : “Cuire” la Configuration (Baking)

- “Cuire” (bake) la configuration dans l’image lors de sa construction.
- Plutôt que configurer chaque instance individuellement à son démarrage (scripts).
- Favorise :
  - Déploiements plus rapides.
  - Pratiques d’infrastructure immuable (instances non modifiées après lancement, mais remplacées par de nouvelles basées sur MàJ images).

Gestion des Images Personnalisées : Défis

- Peut devenir complexe (“image sprawl”).
- Assurer :
  - Versionnement.
  - Stockage (a un coût).
  - Mise à jour régulière des images (correctifs sécurité).
  - Validation des images.
- Outils pour automatiser : Azure Image Builder, AWS EC2 Image Builder.

1.5. Connexion et Administration de Base

- Essentiel une fois VM provisionnée.
- Méthodes et premières étapes varient selon OS.

Méthodes de Connexion : SSH (Secure Shell) pour Linux (1/3)

- Protocole standard et sécurisé pour accès ligne de commande distant.
- **Authentification :**
  - Paires de clés SSH (clé publique sur VM, clé privée sur client) **fortement recommandée**.
  - Plus sécurisée que mot de passe.
  - Clé publique souvent fournie lors création VM.

Méthodes de Connexion : SSH pour Linux (2/3)

Connexion (Client Linux/macOS/Windows récent) :

1. Clé privée (fichier `.pem`, `.key`) stockée localement (ex: `~/.ssh/`).
2. Permissions restrictives sur clé privée : `chmod 400 ~/.ssh/votre_cle.pem`.
3. Commande `ssh` : `ssh -i /chemin/cle.pem utilisateur@IP_Publique_VM_ou_DNS`.
   - Utilisateur par défaut dépend de l'image (ex: `ec2-user`, `ubuntu`).
4. Première connexion : vérifier et accepter empreinte (fingerprint) du serveur.

Méthodes de Connexion : SSH pour Linux (3/3)

Connexion (Client Windows avec PuTTY) :

- Utiliser PuTTYgen pour générer/convertir clé privée au format `.ppk`.
- Configurer session PuTTY : IP publique VM, port 22, spécifier fichier `.ppk`.

Côté Serveur :

- Service SSH (`sshd`) doit être en cours d'exécution sur VM.
- Port 22 (TCP) autorisé dans pare-feu OS et groupes de sécurité IaaS.

Méthodes de Connexion : RDP (Remote Desktop Protocol) pour Windows (1/2)

- Protocole standard pour accès graphique distant à un système Windows.
- **Prérequis :**
  - VM doit avoir une adresse IP publique.
  - Bureau à distance activé dans propriétés système Windows.
  - Port 3389 (TCP) ouvert dans pare-feu Windows et groupes de sécurité IaaS.

Méthodes de Connexion : RDP pour Windows (2/2)

Connexion (Client Windows) :

- Utiliser client “Connexion Bureau à distance” intégré.
- Entrer IP publique VM, connecter, fournir identifiants utilisateur autorisé.

Connexion (Client macOS/Linux) :

- Clients RDP : Microsoft Remote Desktop (macOS), Remmina (Linux).

Autres Méthodes de Connexion (Mention)

- Connexion via navigateur web (ex: Azure Bastion, AWS EC2 Instance Connect).
  - Simplifie accès, évite exposition directe ports SSH/RDP sur internet.
- Console série : pour dépannage au niveau du démarrage.

Tâches d'Administration Initiales (Post-Création) : Linux (1/2)

Une fois connecté :

1. **Mettre à jour le système :**
   - `sudo apt update && sudo apt upgrade -y` (Debian/Ubuntu)
   - `sudo dnf upgrade -y` (Fedora/CentOS/RHEL)
2. Définir un nom d'hôte (`hostname`) pertinent.
3. Configurer le fuseau horaire.

Tâches d'Administration Initiales : Linux (2/2)

4. Configurer le réseau (IP statique si requise).
5. Créer des comptes utilisateurs non-root (`useradd`), mots de passe (`passwd`), groupes (`usermod`).
6. Configurer accès `sudo` pour admins (via `visudo` ou groupe `sudo`/`wheel`).
7. Configurer pare-feu OS de base (ex: `ufw` : `sudo ufw allow ssh`, `sudo ufw enable`).
8. **Sécuriser SSH :**
   - Désactiver connexion root (`PermitRootLogin no`).
   - Autoriser uniquement authentification par clé (`PasswordAuthentication no`).

Tâches d'Administration Initiales : Windows (1/2)

1. Vérifier et installer toutes les mises à jour Windows.
2. Configurer fuseau horaire.
3. Renommer l'ordinateur.
4. Configurer IP statique et serveurs DNS.
5. Configurer Pare-feu Windows (autoriser ports nécessaires, ping).

Tâches d'Administration Initiales : Windows (2/2)

6. Installer rôles et fonctionnalités requis (ex: IIS, .NET Framework).
7. Créer comptes utilisateurs non-administrateurs.
8. Joindre à un domaine Active Directory si nécessaire.
9. Désactiver fonctionnalités non nécessaires (ex: Config sécurité renforcée IE).
10. Installer outils spécifiques hyperviseur si besoin (ex: VMware Tools).

Importance de la Sécurisation Post-Installation

- Accès distant sécurisé (SSH/RDP) = porte d'entrée.
- **Configuration initiale doit inclure renforcement sécurité :**
  - Privilégier authentification par clé SSH.
  - Désactiver connexion root directe.
  - Configurer pare-feu (IaaS et OS).
  - Créer comptes utilisateurs avec privilèges limités.
- Constitue la première ligne de défense de la VM.

Responsabilité Utilisateur et Automatisation

- IaaS fournit infrastructure virtualisée.
- **Utilisateur responsable de la configuration et maintenance de l'OS.**
- Pour gérer > quelques VMs, automatisation des tâches répétitives devient indispensable.
  - Outils de gestion de configuration : Ansible, Puppet, Chef, PowerShell DSC.
  - Comblent écart entre provisionnement initial et état opérationnel final.

1.6. Interfaces de Gestion: Console Web vs. CLI

- Principales interfaces pour interagir avec services IaaS et gérer ressources.
- **Console Web** (Interface Graphique)
- **Interface en Ligne de Commande (CLI)**

Console Web (Interface Graphique)

- Description : Interface utilisateur graphique (GUI) via navigateur web. Souvent premier point de contact.
- **Avantages :**
  - Facilité d'utilisation (intuitive, visuelle). Idéale pour débutants/occasionnels.
  - Découverte (explorer services et options).
  - Visualisation (représentation graphique, tableaux de bord).
  - Tâches Simples (lancer VM, vérifier état).

Console Web : Inconvénients

- Lenteur (peut nécessiter plusieurs clics pour opérations complexes).
- Automatisation Difficile (ne se prête pas bien aux tâches répétitives/gestion en masse).
- Reproductibilité Difficile (garantir config complexe exacte effectuée manuellement).
- Gestion à Grande Échelle Inefficace.

Interface en Ligne de Commande (CLI)

- Description : Interface textuelle (commandes dans un terminal). Chaque fournisseur a sa CLI (AWS CLI, Azure CLI, gcloud CLI).
- **Avantages :**
  - Rapidité et Efficacité (actions complexes avec une seule commande).
  - Automatisation et Scripting (choix pour déploiements, configs, tâches répétitives). Essentiel DevOps/IaC.
  - Gestion en Masse (opérations sur nombreuses ressources simultanément).

CLI : Avantages (suite)

- Précision et Contrôle (contrôle fin sur options et paramètres).
- Reproductibilité (scripts garantissent opérations identiques).
- Intégration (pipelines CI/CD, autres outils d'automatisation).

CLI : Inconvénients

- Courbe d'Apprentissage (nécessite apprendre syntaxe spécifique et concepts). Moins intuitif au début.
- Moins Visuel (pas de vue d'ensemble graphique).

API (Interface de Programmation Applicative)

- Important : Console Web et CLI communiquent généralement avec API REST sous-jacente du fournisseur.
- API peut aussi être utilisée directement par développeurs pour intégrer gestion infra dans applications/scripts.

Quelle Interface Choisir ? Console ou CLI ?

- Dépend fortement du contexte.
- **Console :** Excellente pour apprentissage, exploration, visualisation, dépannage ponctuel, tâches simples/peu fréquentes.
- **CLI :** Indispensable pour automatisation, gestion grande échelle, reproductibilité, intégration DevOps/IaC, utilisateurs expérimentés (efficacité, contrôle précis).

Utilisation en Pratique

- Administrateurs et ingénieurs cloud utilisent souvent une combinaison des deux.
- MAIS : Maîtrise de la CLI = compétence fondamentale pour professionnels du cloud.
  - Débloque potentiel d'automatisation et scalabilité.
- S'appuyer uniquement sur console limite adoption pratiques matures (IaC).

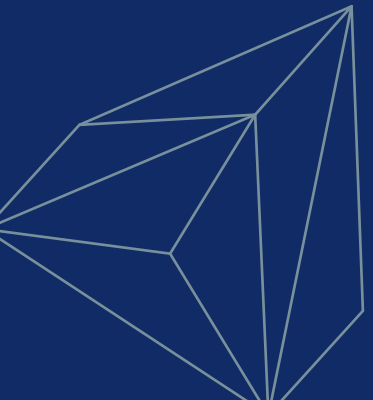
Tableau Comparatif : Console Web vs. CLI

Caractéristique	Console Web	Interface Ligne de Commande (CLI)
Facilité d'usage	Élevée (graphique, intuitive)	Faible (commandes à apprendre)
Rapidité	Moyenne à lente (clics)	Élevée
Automatisation	Faible / Difficile	Élevée
Gestion à l'échelle	Peu Efficace	Très Efficace
Courbe d'apprentissage	Faible	Élevée
Contrôle	Bon	Très Précis
Reproductibilité	Faible	Élevée
Cas d'Usage Idéal	Apprentissage, Visualisation, Tâches Simples/ponctuelles	Automatisation, IaC, Geestion de Masse, Utilisateurs Avancés

Module 1 : Récapitulatif et Questions

- Cycle de vie d'une VM
- Types d'instances
- Gestion du stockage bloc (volumes, snapshots)
- Images VM personnalisées
- Connexion et administration de base
- Console Web vs. CLI
- Des questions ?

Déploiement Manuel d'Application sur VM



2.1. Stratégie et Étapes d'un Déploiement Manuel

- Définition : Installer et configurer une application directement sur une VM via commandes et modification de fichiers de configuration sur l'OS invité.
- Importance de la compréhension manuelle :
  - Même si méthodes automatisées (Ansible, Kubernetes, PaaS) préférées en prod.
  - Permet de saisir interactions : application, dépendances, OS, infrastructure.

Stratégie Générale du Déploiement Manuel

Décomposition typique en phases séquentielles :

1. **Préparation de la VM :** OS à jour, outils de base, accès réseau/pare-feu OK.
2. **Installation des Dépendances :** Logiciels prérequis (serveur web, BDD, runtime, bibliothèques).
3. **Déploiement du Code Applicatif :** Transférer code source ou artefacts compilés sur VM.
4. **Configuration des Services :** Configurer serveur web, BDD, application.
5. **Tests :** Vérifier accessibilité et fonctionnement.

2.2. Étapes Détaillées du Déploiement Manuel

Détaillons chaque phase (exemples pour Linux Debian/Ubuntu).

Étapes Détaillées : 1. Préparation de la VM

- Assurer configuration de base (cf. Module 1.5).
- Mettre à jour paquets : `sudo apt update && sudo apt upgrade -y`.
- Installer outils de base si absents : `sudo apt install git wget curl nano -y`.
- Configurer pare-feu (IaaS et OS) pour autoriser trafic entrant sur ports application (ex: 80 HTTP, 443 HTTPS).
  - Exemple `ufw` : `sudo ufw allow 80/tcp`, `sudo ufw allow 443/tcp`.

Étapes Détaillées : 2. Installation des Dépendances

- Identifier les prérequis :
  - Serveur web (Apache, Nginx).
  - Serveur de base de données (MySQL, PostgreSQL).
  - Runtime (PHP, Python, Node.js, Java).
  - Bibliothèques système ou spécifiques au langage.
- Installer via gestionnaire de paquets.
  - Exemple LAMP (Ubuntu) : `sudo apt install apache2 mysql-server php libapache2-mod-php php-mysql -y`.
  - Exemple LEMP (Ubuntu) : `sudo apt install nginx mysql-server php-fpm php-mysql -y`.

Étapes Détaillées : 3. Déploiement du Code Applicatif (1/2)

- Choisir méthode pour transférer code sur VM :
  - **Cloner depuis Git :** `sudo git clone https://votre-repo.git /var/www/html/mon-app`.
  - **Copie sécurisée (scp) :** `scp -i ~/.ssh/cle.pem -r ./local utilisateur@IP_VM:/var/www/html/mon-app`.
  - **Télécharger archive :** `wget https://url/app.tar.gz` puis `sudo tar -xzf app.tar.gz -C /var/www/html/`.

Étapes Détaillées : 3. Déploiement du Code Applicatif (2/2)

- Placer code dans répertoire racine serveur web (ex: `/var/www/html/`).
- Ajuster propriété et permissions des fichiers pour serveur web :
  - Exemple propriétaire : `sudo chown -R www-data:www-data /var/www/html/mon-app` (www-data pour Apache/Nginx sur Debian/Ubuntu).
  - Exemple permissions : `sudo find /dossier -type d -exec chmod 755 {} \;` et `sudo find /dossier -type f -exec chmod 644 {} \;`.

Étapes Détaillées : 4. Configuration des Services - Serveur Web (1/2)

- Configurer hôte virtuel (Apache) ou bloc serveur (Nginx) :
  - `ServerName` (nom de domaine).
  - `DocumentRoot` (chemin vers code).
  - Directives spécifiques (réécriture URL, gestion PHP via FPM pour Nginx).
- Fichiers de configuration : `/etc/apache2/sites-available/` ou `/etc/nginx/sites-available/`.

Étapes Détaillées : 4. Configuration des Services - Serveur Web (2/2)

- Activer configuration :
  - Apache : `sudo a2ensite mon-app.conf`.
  - Nginx : `sudo ln -s /etc/nginx/sites-available/mon-app /etc/nginx/sites-enabled/`.
- Tester configuration :
  - Apache : `sudo apache2ctl configtest`.
  - Nginx : `sudo nginx -t`.
- Recharger/Redémarrer service : `sudo systemctl reload apache2` (ou `nginx`).

Étapes Détaillées : 4. Configuration des Services - Serveur BDD (1/2)

- Sécuriser installation initiale (ex: `sudo mysql_secure_installation` pour MySQL).
- Se connecter au SGBD (ex: `sudo mysql -u root -p`).
- Créer base de données pour l'application (ex: `CREATE DATABASE mon_app_db;`).
- Créer utilisateur dédié (ex: `CREATE USER 'user'@'localhost' IDENTIFIED BY 'pass';`).

Étapes Détaillées : 4. Configuration des Services - Serveur BDD (2/2)

- Donner privilèges à utilisateur sur BDD (ex: `GRANT ALL PRIVILEGES ON mon_app_db.* TO 'user'@'localhost';`).
- Appliquer changements (ex: `FLUSH PRIVILEGES;`). Quitter.
- S'assurer que service démarré et activé au boot (ex: `sudo systemctl start mysql`, `sudo systemctl enable mysql`).

Étapes Détaillées : 4. Configuration des Services - Application

- Configurer application pour connexion BDD :
  - Éditer fichier de configuration (`.env`, `config.php`, etc.).
  - Indiquer : nom base, utilisateur, mot de passe, hôte (localhost).
- Configurer autres paramètres : clés API, variables d'environnement, etc.

Étapes Détaillées : 5. Tests

- Vérifier état des services (ex: `sudo systemctl status apache2 mysql`).
- Accéder à l'application via navigateur web (IP publique ou DNS de la VM).
- Consulter journaux (logs) en cas d'erreur :
  - Serveur web : `/var/log/apache2/error.log`, `/var/log/nginx/error.log`.
  - Application : dépend de l'application.
- Effectuer tests fonctionnels de base (navigation, formulaires, BDD).

Déploiement Manuel : Complexité et Fragilité

- Met en évidence dépendance étroite : application, prérequis logiciels, config OS, infra.
- Modification à un niveau (ex: MàJ OS) peut casser l'application.
- Maintenance complexe et source d'erreurs.
- Justifie adoption méthodes plus modernes/robustes (conteneurisation, PaaS).

2.3. Exemple Concret: Déploiement d'un Site Web Simple (LAMP/LEMP)

Illustrons avec un site PHP simple + MySQL sur Ubuntu.

Exemple Concret : 1. Préparation de la VM (Ubuntu)

```
# Connexion SSH
ssh -i ~/.ssh/ma_cle.pem ubuntu@<IP_VM>

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

# Outils de base
sudo apt install -y git nano

# Pare-feu UFW (si actif)
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp # Si HTTPS
sudo ufw enable
sudo ufw status
```

Exemple Concret : 2. Installation Dépendances (Choix: LAMP)

```
# Installation Apache, MySQL, PHP et module PHP pour MySQL
sudo apt install -y apache2 mysql-server php libapache2-mod-php php-mysql

# Alternative LEMP:
# sudo apt install -y nginx mysql-server php-fpm php-mysql
```

Exemple Concret : 3. Déploiement Code Applicatif (PHP simple)

```
# Création fichier PHP simple pour tester
sudo nano /var/www/html/index.php
```

Contenu `index.php` :

```
<?php
phpinfo();
?>
```

```
# Ajustement permissions (si nécessaire)
sudo chown -R www-data:www-data /var/www/html
```

Exemple Concret : 4. Configuration Services - Apache

- Config par défaut (`/etc/apache2/sites-enabled/000-default.conf`) pointe vers `/var/www/html`.
- Peut suffire pour site simple.

```
sudo systemctl status apache2
sudo systemctl enable apache2 # Activer au démarrage
```

Exemple Concret : 4. Configuration Services - MySQL/MariaDB (1/2)

```
# Sécuriser l'installation
sudo mysql_secure_installation
# (Suivre instructions: mot de passe root, etc.)

# Connexion à MySQL
sudo mysql -u root -p
# (Entrer mot de passe root)
```

Exemple Concret : 4. Configuration Services - MySQL/MariaDB (2/2)

Dans le shell MySQL :

```
CREATE DATABASE simple_app_db;
CREATE USER 'simple_user'@'localhost' IDENTIFIED BY 'UnMotDePasseTresSecret';
GRANT ALL PRIVILEGES ON simple_app_db.* TO 'simple_user'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

```
# Vérifier statut MySQL
sudo systemctl status mysql
sudo systemctl enable mysql # Activer au démarrage
```

Exemple Concret : 4. Configuration Services - Application (Test DB)

Modifier `index.php` :

```
sudo nano /var/www/html/index.php
```

Nouveau contenu `index.php` :

```
<!DOCTYPE html>
<html>
<head><title>Test Connexion DB</title></head>
<body>
<h1>Test Connexion Base de Données</h1>
<?php
$servername = "localhost";
$username = "simple_user";
$password = "UnMotDePasseTresSecret"; // ATTENTION: Jamais en dur en prod!
$dbname = "simple_app_db";

$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) {
  die("<p style='color:red;'>Échec connexion: " . $conn->connect_error . "</p>");
}
echo "<p style='color:green;'>Connexion à la BDD réussie!</p>";
$conn->close();
?>
</body>
</html>
```

Exemple Concret : 5. Tests

- Ouvrir navigateur web : `http://<IP_VM>`.
- Doit afficher "Connexion à la base de données réussie!".
- Si erreur, vérifier :
  - Statut services (`systemctl status apache2 mysql`).
  - Logs Apache (`/var/log/apache2/error.log`).
  - Logs PHP (peut nécessiter config `php.ini`).
  - Identifiants BDD dans code PHP.
  - Règles pare-feu.

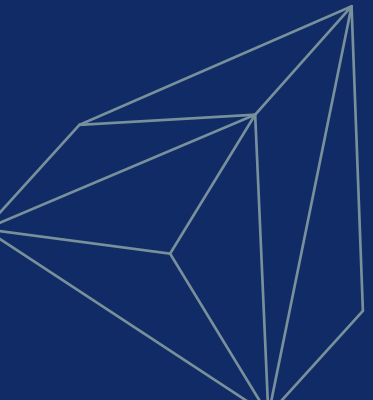
Déploiement Manuel : Limites de l'Exemple

- Processus de base illustré.
- Déploiement réel :
  - Configuration Virtual Hosts/Server Blocks.
  - Gestion dépendances PHP avec Composer.
  - Configuration HTTPS, etc.

Module 2 : Récapitulatif et Questions

- Stratégie et étapes du déploiement manuel.
- Détail des phases : préparation VM, dépendances, code, config services, tests.
- Exemple concret LAMP.
- Complexité et justification des approches modernes.
- Des questions ?

Introduction aux Conteneurs et Docker



3.1. Contexte et Motivation: Pourquoi les Conteneurs? (1/2)

Défis du déploiement traditionnel / sur VM :

- **Conflits de dépendances :** Différentes apps sur même serveur peuvent nécessiter versions incompatibles de bibliothèques/runtimes.
- **Problème “ça marche sur ma machine” :** Différences subtiles entre environnements (dev, test, prod) -> erreurs imprévues.

3.1. Contexte et Motivation: Pourquoi les Conteneurs? (2/2)

- Surcoût des VMs :
  - Chaque VM embarque un OS complet (noyau, services système).
  - Consomme ressources (CPU, RAM, disque).
  - Ralentit temps de démarrage et déploiement.
  - Taille des images VM conséquente.

Solution : La Conteneurisation

- Forme de virtualisation au niveau du système d'exploitation.
- Permet d'exécuter applications dans environnements isolés : **conteneurs**.
- Conteneurs partagent le noyau (kernel) de l'OS hôte.
- Disposent de leur propre espace de noms (processus, système de fichiers, réseau) -> isolation.

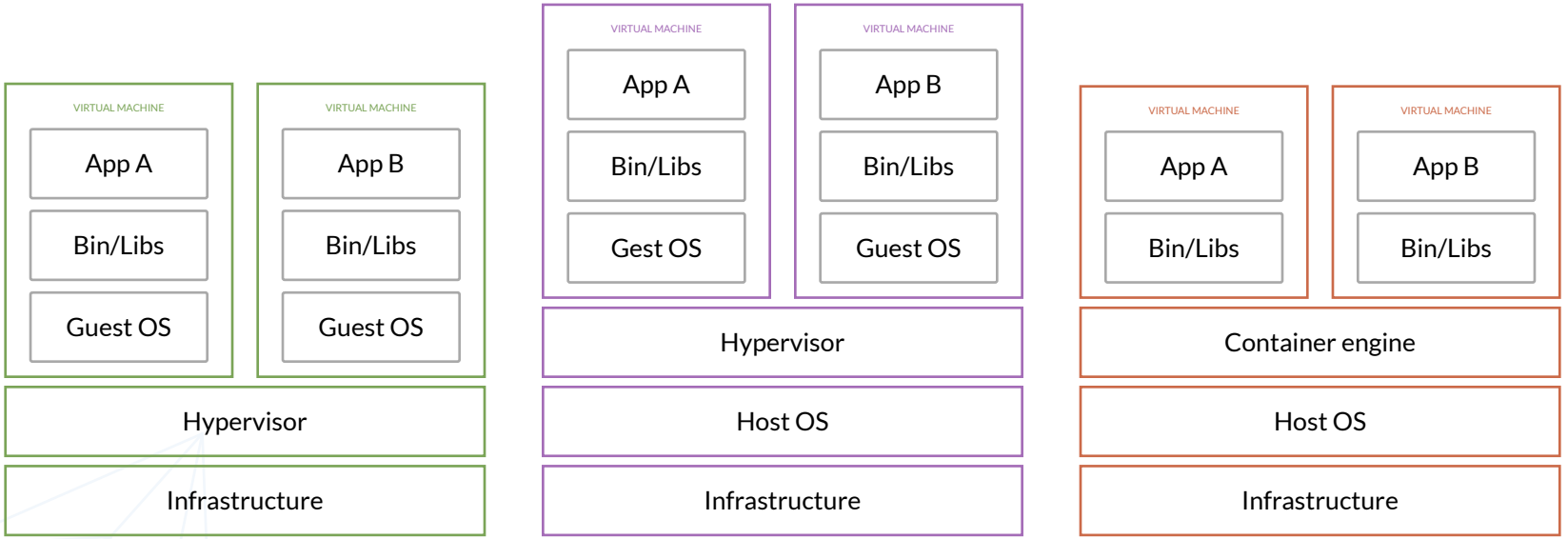
Comparaison : Machines Virtuelles (VMs) vs. Conteneurs (1/2)

Aspect	Machines Virtuelles (VMs)	Conteneurs
Architecture	Hyperviseur + OS Invité complet par VM	Moteur de conteneurs + Partage Noyau OS Hôte
Isolation	Forte (Niveau matériel/OS)	Bonne (Niveau processus)
Surcharge	Élevée (OS complet par VM)	Faible (Partage OS Hôte)
Démarrage	Minutes (Boot OS)	Secondes (Processus)

Comparaison : VMs vs. Conteneurs (2/2)

Aspect	Machines Virtuelles (VMs)	Conteneurs
Taille	Grande (Go)	Petite (Mo à Go)
Densité	Plus faible	Plus élevée
Portabilité	Limitée par dépendances OS	Élevée (entre hôtes avec moteur compatible)
Flexibilité OS	Peut exécuter différents OS	Limité à l'OS hôte (Linux ou Windows)
Cas d'Usage	Isolation forte, OS différents, Legacy Apps	Microservices, Web Apps, CI/CD, Dev/Test

Schéma Comparatif : VMs vs. Conteneurs



Hyperviseur de Type I | **Hyperviseur de Type II** | **Container**

Avantages des Conteneurs

- **Cohérence des environnements :** Application et dépendances packagées ensemble (dev -> prod).
- **Efficacité des ressources :** Moins de RAM, CPU, disque (partage OS hôte).
- **Déploiement et Scalabilité rapides :** Démarrent quasi instantanément.
- **Isolation applicative :** Apps isolées les unes des autres sur même hôte.
- **Portabilité :** Image tourne sur tout système avec moteur compatible (ex: Docker).

Paradigme : Déploiement Centré Application

- Passage d'un déploiement centré machine (VM) à centré application.
- Encapsulation application + dépendances directes -> abstraction OS sous-jacent.
- Favorise architectures microservices et pratiques DevOps.
- Unité standardisée, légère, efficace pour construire, distribuer, exécuter applications.

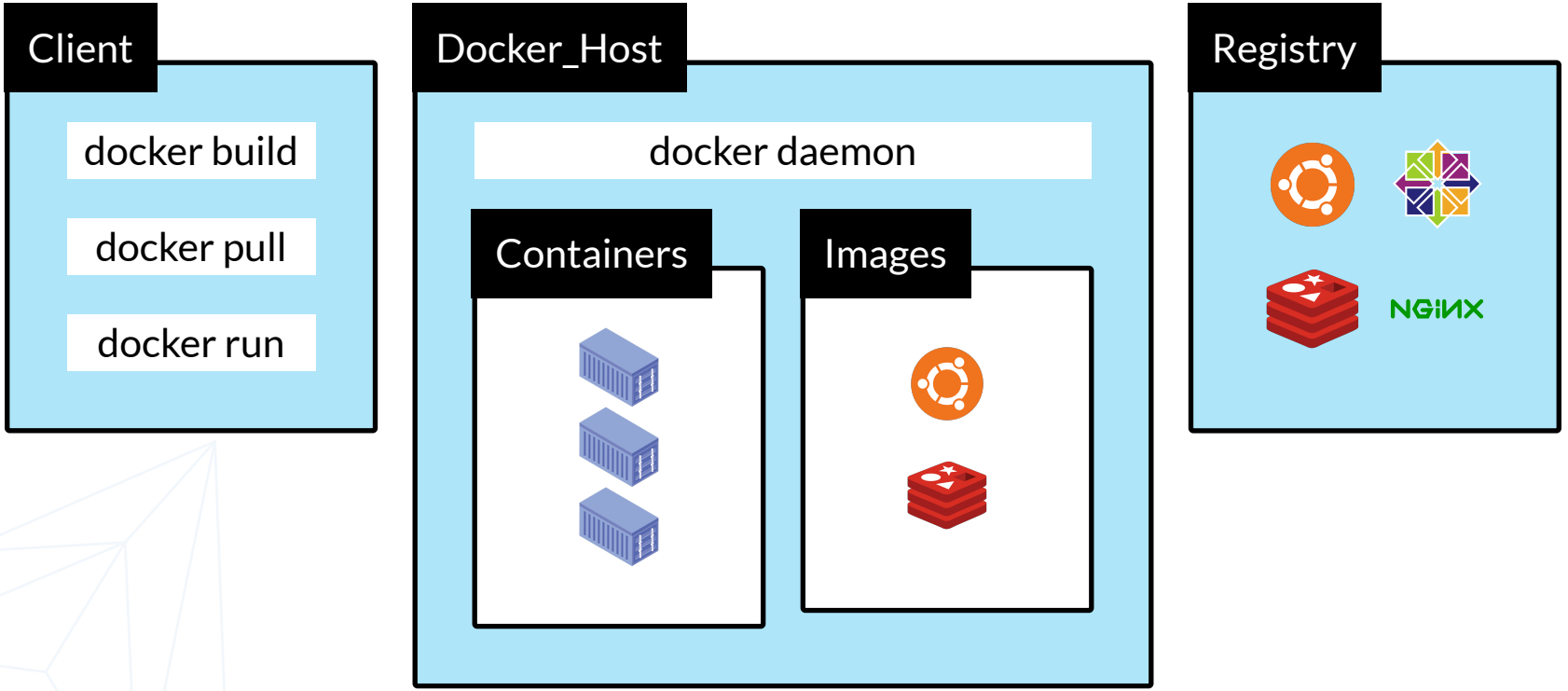
3.2. Concepts Clés de Docker

- Docker : plateforme de conteneurisation la plus populaire.
- Simplifie création, gestion, déploiement des conteneurs.
- Concepts fondamentaux :
  - Docker Engine
  - Image Docker
  - Conteneur Docker
  - Dockerfile
  - Registre Docker

Concepts Docker : Docker Engine

- Cœur de Docker, application client-serveur.
- Comprend :
  - Daemon Docker (`dockerd`) :
    - Service en arrière-plan sur hôte.
    - Écoute requêtes API Docker, gère objets Docker (images, conteneurs, réseaux, volumes).
    - Construit, exécute, distribue conteneurs.
  - Client Docker (`docker`) :
    - Interface en ligne de commande (CLI) pour interagir avec daemon (ex: `docker run`).

Schéma : Architecture Docker Engine



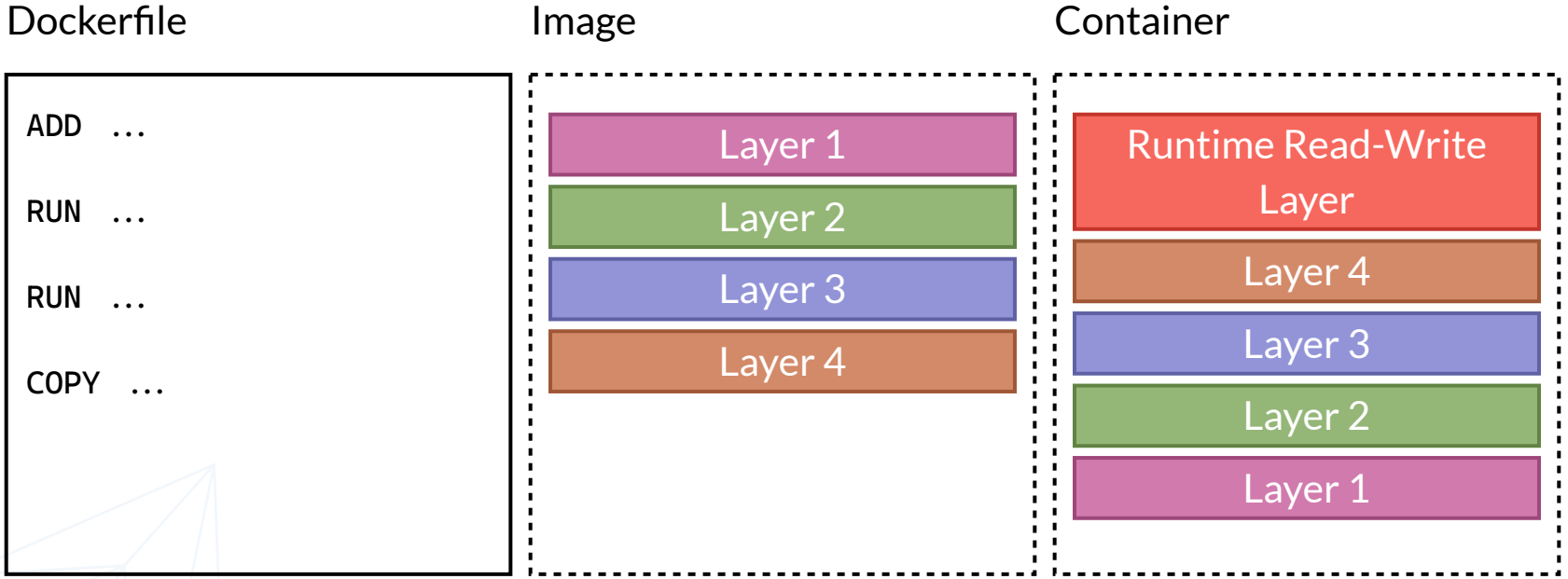
Concepts Docker : Image Docker (1/2)

- Modèle en **lecture seule**, basé sur des **couches** (layers).
- Contient tout le nécessaire pour exécuter une application :
  - Code
  - Runtime
  - Bibliothèques
  - Variables d'environnement
  - Fichiers de configuration
- Construites à partir d'un `Dockerfile`.

Concepts Docker : Image Docker (2/2)

- Nature en couches rend images efficaces :
  - Reconstruction : seules couches modifiées refaites (cache).
  - Couches communes partagées entre plusieurs images (économie espace disque, bande passante).

Schéma : Couches d'une Image Docker

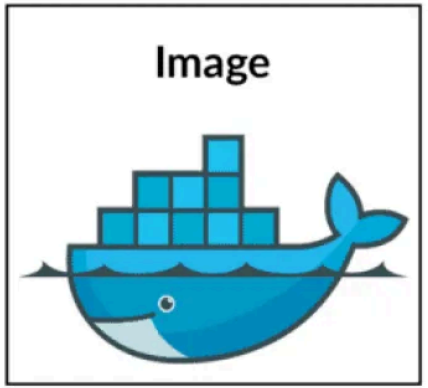


Concepts Docker : Conteneur Docker

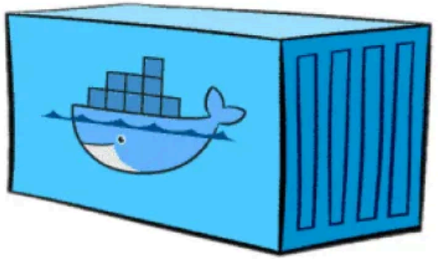
- Instance **exécutable** d'une image Docker.
- Processus isolé qui tourne sur l'OS hôte.
- On peut créer, démarrer, arrêter, déplacer, supprimer un conteneur.
- Par défaut, isolés les uns des autres et de l'hôte.
- Couche accessible en écriture par-dessus les couches de l'image.

Relation Image / Conteneur Docker

Dockerfile



Docker Image



Docker Container

Concepts Docker : Dockerfile (1/2)

- Fichier texte contenant série d'instructions.
- Permet à Docker Engine de construire automatiquement une image Docker.
- Agit comme une recette ou un script de construction.
- Infrastructure as Code pour l'environnement d'exécution applicatif.
- Définit création images de manière versionnable, répétable, automatisée -> cohérence.

Concepts Docker : Dockerfile (2/2) - Instructions Courantes

- `FROM <image_de_base>` : Définit image de départ (ex: `ubuntu:22.04`).
- `WORKDIR /chemin/conteneur` : Définit répertoire de travail.
- `COPY <source_hôte> <dest_conteneur>` : Copie fichiers/répertoires.
- `RUN <commande>` : Exécute commande pendant construction image (crée nouvelle couche).
- `EXPOSE <port>` : Documente port d'écoute applicatif (n'expose pas réellement).
- `CMD ["executable", "param1"]`

  : Commande par défaut au démarrage conteneur.

  - (Alternative : `ENTRYPOINT`).

Concepts Docker : Registre Docker (Registry)

- Système de stockage et de distribution pour images Docker.
- **Docker Hub :** Registre public par défaut (milliers d'images officielles/communautaires).
- Entreprises peuvent déployer registres privés (images internes).
- Commandes : `docker pull` (télécharger), `docker push` (envoyer).

3.3. Commandes Docker Essentielles (1/4)

- `docker build -t <nom_image>[:<tag>] .`
  - Construit image Docker à partir d'un `Dockerfile` dans répertoire courant (`.`).
  - `-t` : attribue nom et tag (version, ex: `latest`, `1.0`).
- `docker images`
  - Liste images Docker présentes localement.

Commandes Docker Essentielles (2/4)

- `docker run <options> <nom_image>[:<tag>] [COMMANDE_OVERRIDE]`
  - Crée et démarre un nouveau conteneur à partir d'une image.
  - Options courantes :
    - `-d` : Détaché (arrière-plan).
    - `-p <port_hôte>:<port_conteneur>` : Publie un port.
    - `--name <nom_conteneur>` : Attribue un nom.
    - `-v <volume_ou_chemin_hôte>:<chemin_conteneur>` : Monte un volume/répertoire.
    - `--rm` : Supprime conteneur à l'arrêt.
    - `-it` : Mode interactif + pseudo-TTY (ex: `docker run -it ubuntu bash`).

Commandes Docker Essentielles (3/4)

- `docker ps`
  - Liste conteneurs en cours d'exécution.
  - `docker ps -a` : Liste tous conteneurs (actifs et arrêtés).
- `docker stop <nom_conteneur_ou_ID>`
  - Arrête un conteneur (SIGTERM, puis SIGKILL).
- `docker rm <nom_conteneur_ou_ID>`
  - Supprime un conteneur arrêté.
  - `-f` : force suppression conteneur en cours (non recommandé).

Commandes Docker Essentielles (4/4)

- `docker pull <nom_image>[:<tag>]`
  - Télécharge image depuis registre (Docker Hub par défaut).
- `docker push <registre_si_non_hub>/<user>/<image>[:<tag>]`
  - Envoie image locale vers un registre (nécessite `docker login`).
- `docker logs <nom_conteneur_ou_ID>`
  - Affiche logs (stdout/stderr) d'un conteneur.
  - `-f` : suit logs en temps réel.
- `docker exec -it <nom_conteneur_ou_ID> <commande>`
  - Exécute commande dans conteneur déjà en cours (ex: `docker exec -it mon_app bash`).

Flux de Travail Typique avec Docker

1. Modification code ou `Dockerfile`.
2. `docker build ...`
3. `docker run ...`
4. Vérification (`docker ps`, `docker logs`).
5. `docker stop ...` (si besoin).
6. `docker rm ...` (si besoin).
7. Répétition jusqu'au comportement désiré.
8. `docker push ...` (pour partager image).

– Séparation build (image immuable) / run (conteneur éphémère).

3.4. Introduction à la Persistance et au Réseau Docker

Crucial pour applications conteneurisées utiles en pratique.

- Persistance des données.
- Communication réseau.

Persistance des Données : Le Problème

- Par nature, système de fichiers d'un conteneur est **éphémère**.
- Si conteneur supprimé, données écrites à l'intérieur (BDD, uploads) sont **perdues**.

Persistance des Données : La Solution - Volumes Docker

- Mécanisme pour stocker données de manière persistante.
- Gérés par Docker, stockés sur hôte (ex: `/var/lib/docker/volumes/`) mais **hors cycle de vie des conteneurs**.
- Types principaux :
  - Volumes Nommés (recommandé).
  - Volumes Anonymes.
  - Bind Mounts (pour développement surtout).

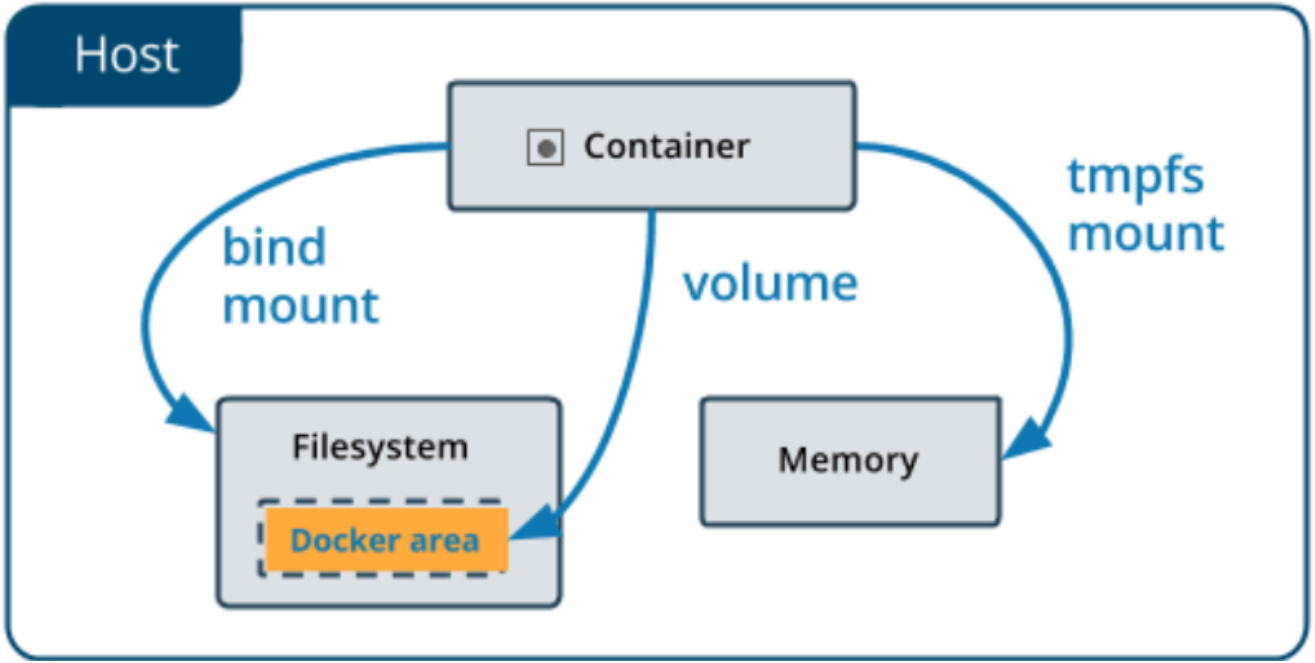
Volumes Docker : Volumes Nommés

- Créés explicitement : `docker volume create mon_volume`.
- Ou implicitement : `docker run -v mon_volume:/chemin/conteneur ...`.
- Faciles à référencer et gérer. **Méthode recommandée.**
- Cas d'usage : BDD, uploads, logs importants, configs persistantes.
- Permettent partage données entre plusieurs conteneurs.

Volumes Docker : Bind Mounts

- Lient directement un fichier/répertoire de l'hôte dans le conteneur.
  - `docker run -v /chemin/hôte:/chemin/conteneur ...`.
- Utile en développement (refléter changements code source instantanément).
- MAIS :
  - Dépendant système de fichiers hôte.
  - Moins portable.
  - Peut poser problèmes permissions.
- Volumes (nommés) sont gérés par Docker, découplent stockage de l'hôte.

Schéma : Volumes vs. Bind Mounts



Réseau Docker : Le Besoin

- Conteneurs doivent pouvoir communiquer :
  - Entre eux (ex: web server <-> database).
  - Avec monde extérieur (recevoir requêtes, accéder APIs externes).

Réseau Docker : Pilotes Réseau (Network Drivers)

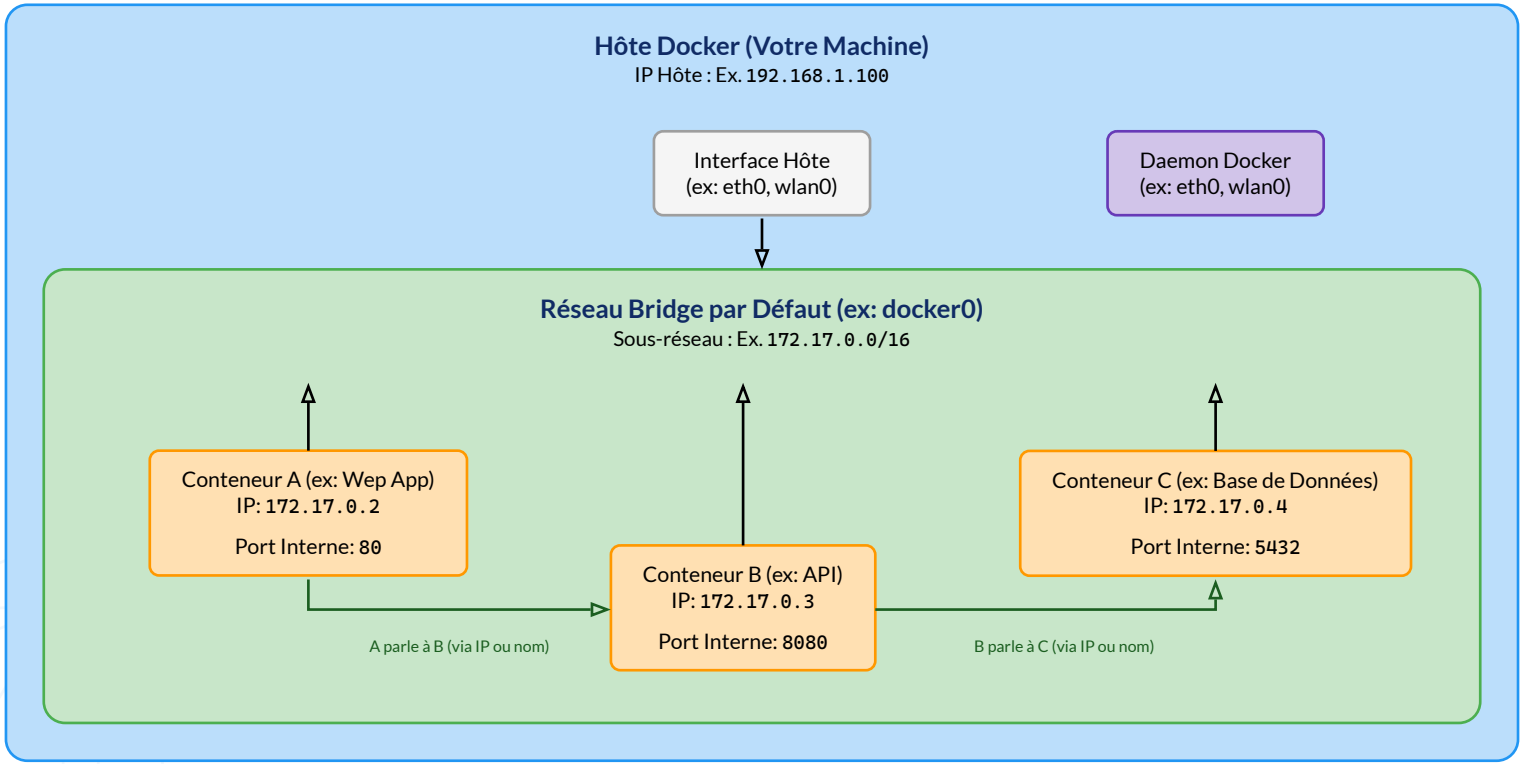
Docker utilise pilotes réseau pour gérer connectivité. Plus courants :

- `bridge` (Pont) :
  - Défaut pour conteneurs autonomes sur un seul hôte.
  - Crée réseau privé virtuel sur hôte (ex: `172.17.0.0/16`).
  - Conteneurs attachés obtiennent IP, peuvent communiquer.
  - Accès externe : mapper port hôte vers port conteneur (`-p 8080:80`).
  - On peut créer plusieurs réseaux `bridge` personnalisés (isolation).

Pilotes Réseau (suite)

- `host` (Hôte) :
  - Supprime isolation réseau. Conteneur utilise directement interface réseau hôte (même IP, ports).
  - Meilleures performances mais pas d'isolation, risque conflits ports.
- `overlay` (Superposition) :
  - Pour connecter conteneurs sur plusieurs hôtes Docker (ex: Docker Swarm, Kubernetes).
  - Crée réseau virtuel distribué.
- `none` :
  - Désactive toute connectivité réseau pour le conteneur.

Schéma : Réseau Docker `bridge` par défaut



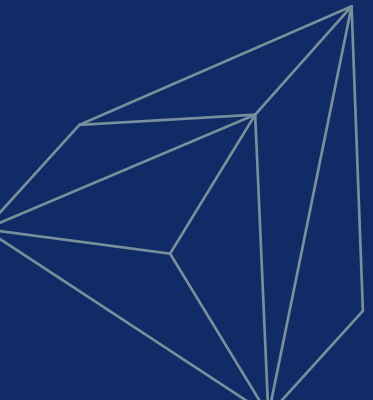
Communication Inter-Conteneurs (sur même réseau défini par l'utilisateur)

- Si conteneurs sur même réseau `bridge` personnalisé ou `overlay`.
- Docker fournit DNS interne.
- Permet découverte et communication en utilisant simplement **nom du conteneur** comme nom d'hôte.
- Abstrait complexité réseau sous-jacent.

Module 3 : Récapitulatif et Questions

- Pourquoi les conteneurs ? VMs vs Conteneurs.
- Concepts Docker : Engine, Image, Conteneur, Dockerfile, Registre.
- Commandes Docker essentielles.
- Persistance (Volumes) et Réseau Docker.
- Des questions ?

Introduction à la Tarification IaaS



4.1. Modèles de Tarification Courants

Fournisseurs IaaS proposent plusieurs modèles pour ressources de calcul (VMs) :

- Paiement à l'Usage / À la Demande (Pay-as-you-go / On-Demand)
- Instances Réservées (Reserved Instances - RIs)
- Instances Spot / VMs Préemptives

Modèle : Paiement à l'Usage / À la Demande

- Principe : Plus flexible. Paie uniquement pour ressources consommées (seconde, minute, heure). Sans engagement.
- Avantages : Flexibilité max (démarrer, arrêter, redimensionner). Idéal pour charges imprévisibles, dev/test, projets courts.
- Inconvénients : Coût par unité le plus élevé. Coûts difficiles à prévoir si utilisation varie fortement.

Modèle : Instances Réservées (RIs)

- Principe : Engagement à utiliser quantité de ressources (type instance, région) pour durée déterminée (1 ou 3 ans). En échange, réduction significative (jusqu'à 70-75%). Paiement initial, partiel ou mensuel.
- Avantages : Réductions coûts substantielles pour charges stables/prévisibles. Peut inclure réservation capacité.
- Inconvénients : Moins flexible (engagement ferme). Si besoins changent ou sous-utilisation, économie réduite/annulée. Certaines RIs échangeables/vendables (avec contraintes).
- Cas d'usage : Apps critiques 24/7, BDD stables, serveurs apps charge constante.

Modèle : Instances Spot / VMs Préemptives

- Principe : Fournisseurs vendent capacité calcul inutilisée à prix très bas (jusqu'à 90% réduction) via enchères ou prix Spot fluctuant.
- Avantages : Coût calcul extrêmement bas.
- Inconvénients : Aucune garantie disponibilité. Fournisseur peut reprendre instance à tout moment (préavis court/nul) si capacité requise pour autres ou prix Spot > max défini. Nécessite apps tolérantes pannes/interruptions.
- Cas d'usage : Tâches non critiques (batch, analyse données, HPC, test, rendu).

Stratégie d'Optimisation des Coûts

- Combine souvent ces modèles :
  - **Instances Réservées :** Charge de base prévisible.
  - **Instances On-Demand :** Variabilité, pics de charge.
  - **Instances Spot :** Tâches tolérantes aux pannes.
- Optimisation coûts cloud (FinOps) = discipline à part entière (outils, expertise).

Tableau Récapitulatif des Modèles de Tarification

Modèle	Description	Niveau Coût	Flexibilité	Engagement	Risque Interruption	Cas d'Usage Idéal
On-Demand	Paiement à l'usage, sans engagement	Élevé	Maximale	Aucun	Faible	Dev/Test, Charges variables, Projets courts
Reserved Instances	Engagement 1-3 ans pour forte réduction	Moyen/Bas	Faible	Élevé	Très Faible	Charges stables 24/7, Production prévisible
Spot Instances	Utilisation capacité excédentaire à bas prix	Très Bas	Variable	Aucun	Élevé	Tâches tolérantes aux pannes, Batch, Analyse Data

4.2. Facteurs Clés Influant sur les Coûts IaaS

Facture IaaS = somme de coûts granulaires. Principaux postes :

- Calcul (Compute)
- Stockage (Storage)
- Réseau (Network / Data Transfer)
- Autres (IP publiques, Load Balancers, monitoring…)

Facteurs Coûts : Calcul (Compute) (1/2)

Souvent coût le plus important. Facteurs :

- Type et Taille de l'Instance :
  - vCPUs, RAM, génération processeur, famille instance.
  - Instances avec GPU/matériel spécialisé = plus chères.
- Durée d'Utilisation :
  - Facturation seconde/heure, même si VM inactive (`idle`) mais `running`.

Facteurs Coûts : Calcul (Compute) (2/2)

- Système d'Exploitation :
  - Licences OS (Windows Server) peuvent ajouter coût vs Linux gratuit.
- Modèle de Prix :
  - Coût horaire varie drastiquement (On-Demand, Reserved, Spot).
- Région :
  - Tarifs peuvent légèrement varier entre régions géographiques.

Facteurs Coûts : Stockage (Storage)

Facteurs :

- **Volume Stocké :** Facturation principale au Go/mois.
- Type de Stockage :
  - Stockage Bloc (EBS, Managed Disks) : coût varie selon perf (SSD vs HDD). SSDs plus chers/rapides. Certains types permettent provisionner (et payer pour) IOPS/débit.
  - Stockage Objet (S3, Blob Storage) : coût dépend classe stockage (standard, accès peu fréquent, archive). Classes “froides” moins chères, mais coûts récupération/latence possibles.
  - Snapshots/Sauvegardes : consomment espace, facturés.
- **Requêtes :** Pour stockage objet, frais minimes par requête (GET, PUT).

Facteurs Coûts : Réseau (Network / Data Transfer) (1/2)

- Transfert Entrant (Ingress) :
  - Trafic réseau entrant vers cloud depuis Internet = généralement **gratuit**.
- Transfert Sortant (Egress) :
  - Où se situent coûts principaux et souvent complexes.
  - Trafic sortant depuis réseau fournisseur vers Internet = généralement **payant** (au Go transféré).

Facteurs Coûts : Réseau (Network / Data Transfer) (2/2)

Variations Egress :

- Destination (Internet vs autre région vs autre AZ même région). Trafic intra-région moins cher/gratuit.
- Volume de données transféré (tarifs dégressifs par paliers parfois).
- Région d'origine. Impact :
- Frais egress peuvent devenir très importants (apps servant beaucoup contenu, transferts inter-régions).
- Source fréquente de "mauvaises surprises". Architecture applicative (ex: CDN) a impact majeur.

Facteurs Coûts : Autres (Mention)

- Adresses IP publiques statiques.
- Load Balancers (répartiteurs de charge).
- Services de surveillance et de logging (collecte, stockage métriques/logs).
- Requêtes API vers certains services managés (BDD, serverless).

Complexité de la Facturation IaaS

- Granularité offre transparence sur utilisation.
- MAIS rend estimation et optimisation des coûts complexes.
- Anticiper flux réseau et utilisation services annexes, pas juste calcul/stockage.

4.3. Calculateurs de Coûts

- Outils en ligne proposés par fournisseurs IaaS.
- Objectif : Aider à estimer coûts potentiels AVANT déploiement.
- Essentiels pour : planification budgétaire, comparaison architectures/options, prise de décision.

Calculateurs de Coûts : Fonctionnement

1. Utilisateur sélectionne services prévus (VMs, stockage, BDD, réseau…).
2. Spécifie détails configuration pour chaque service :
   - VMs : type, taille, nombre, région, OS, durée, modèle prix.
   - Stockage : type, capacité, perf, volume snapshots.
   - Réseau : estimation volume données sortantes.
   - Autres services : unités spécifiques.
3. Calculateur applique tarifs (varient par région), fournit estimation (mensuelle/annuelle).

Calculateurs de Coûts : Exemples

- AWS Pricing Calculator
- Azure Pricing Calculator (inclut TCO calculator)
- Google Cloud Pricing Calculator
- Autres : IBM, DigitalOcean, Linode…
- Outils tiers pour comparer entre fournisseurs (Holori, Cloudorado).

Calculateurs de Coûts : Utilisation et Limites

- Utilisation : Modéliser scénarios, comparer impact choix (région, instance, modèle prix). Estimations sauvegardables/exportables.
- Limites :
  - Fournissent **estimations**. Précision dépend qualité hypothèses utilisateur.
  - Coûts réels peuvent varier (utilisation effective, fluctuations charge, frais egress difficiles à estimer, ressources oubliées).
  - Estiment bien coûts ressources provisionnées, moins bien coûts dynamiques liés à utilisation réelle.
  - Réévaluation régulière coûts réels vs estimations nécessaire.

Module 4 : Récapitulatif et Questions

- Modèles de tarification (On-Demand, Reserved, Spot).
- Facteurs influant sur les coûts (Calcul, Stockage, Réseau).
- Utilité et limites des calculateurs de coûts.
- Des questions ?

Conclusion : Acquis d'Apprentissage du CM2 (1/2)

À l'issue de ce CM2, vous devriez être capables de :

1. Décrire la gestion des VMs et du stockage associé (AAV 1) :
   - Expliquer cycle de vie VM.
   - Identifier catégories types d'instances et cas d'usage.
   - Détailler procédures gestion stockage bloc (attacher/détacher volumes, snapshots).
   - Comprendre concept/avantages images VM personnalisées.
   - Savoir se connecter à VM (SSH/RDP), tâches admin de base, distinguer console/CLI.

Conclusion : Acquis d'Apprentissage du CM2 (2/2)

1. Lister les étapes d'un déploiement manuel (AAV 2) :
   - Énumérer/expliquer phases pour déployer manuellement app web simple (LAMP/LEMP) sur VM IaaS.
2. Expliquer les bases des conteneurs et de Docker :
   - Définir conteneurisation, avantages vs virtualisation tradi.
   - Maîtriser vocabulaire Docker (Image, Conteneur, Engine, Dockerfile, Registre).
   - Utiliser commandes Docker essentielles (build, run, ps, etc.).
   - Comprendre concepts volumes Docker (persistance) et réseaux Docker.
3. Identifier les principaux postes de coût IaaS et utiliser un calculateur (AAV 3) :
   - Reconnaître modèles tarification, facteurs majeurs influençant facture.
   - Comprendre utilité calculateurs, savoir les utiliser pour estimation simple.

Prochaines Étapes

- Ces connaissances théoriques = base essentielle pour :
  - TP2 (Déploiement manuel)
  - TP3 (Kubernetes - s'appuie sur concepts conteneurs)
- Poursuivre exploration technologies cloud.