paul.millet@lameduse.fr

LaMeDuSe

DCK : Docker, fonctionnalités avancées

Version : 04/08/2024

Prérequis

- Bonnes connaissances pour mettre en œuvre et déployer des conteneurs virtuels Docker pour Linux.

Objectifs

- Administrer les réseaux Docker virtuels
- Gérer des images en masse
- Mettre en œuvre une architecture en haute disponibilité
- Déployer des applications Docker multiconteneurs avec Docker Compose
- Administrer, superviser des conteneurs avec Docker Swarm
- Mettre en place un registre local
- Sécuriser l'accès aux conteneurs

Tour de table

- Présentation des membres
- Vos attentes vis-à-vis de la formation

Programme de la formation

1. Le moteur Docker
2. Images et conteneurs
3. Le réseau sous-jacent
4. Le stockage Docker, mise en haute disponibilité et en production
5. Docker Compose
6. Docker Swarm
7. Mise en œuvre d'un registre
8. La sécurité dans Docker et monitoring
9. Ressources des TP
10. Annexe & Ressources connexes

paul.millet@lameduse.fr

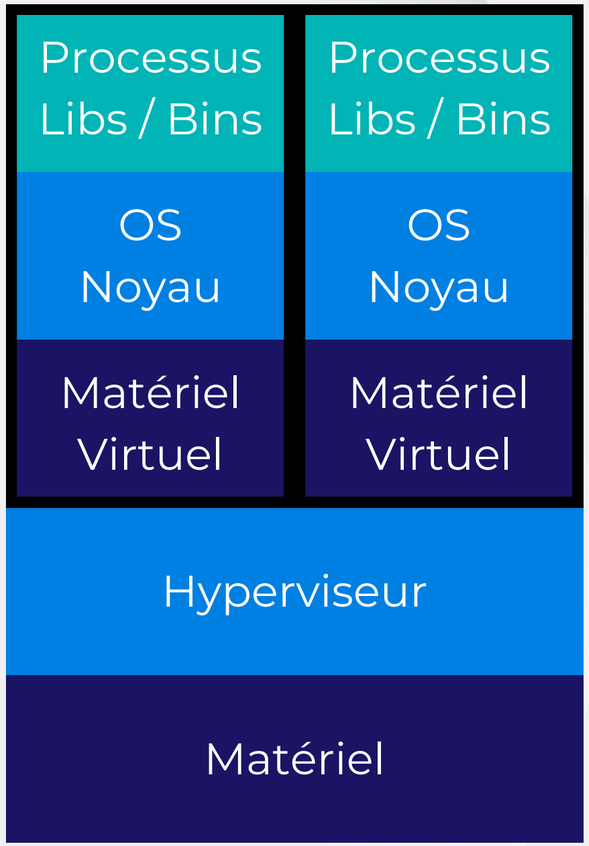
LaMeDuSe

DCK : Docker, fonctionnalités avancées

Le moteur Docker

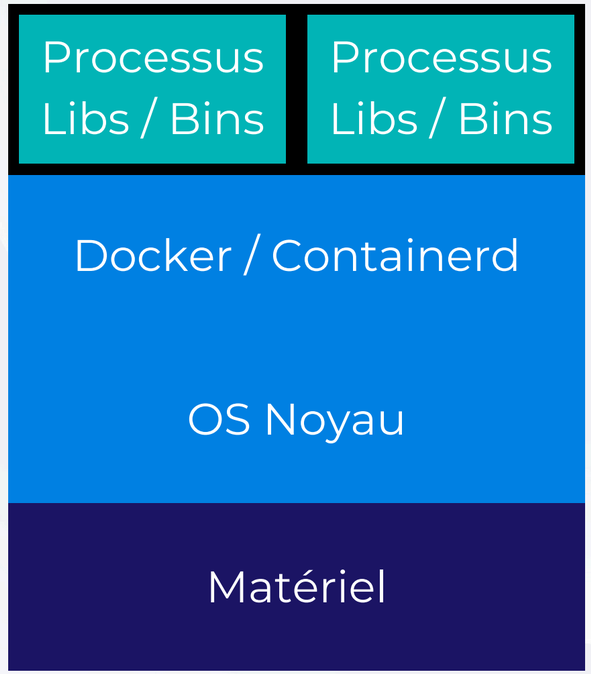
1. Architecture du moteur Docker.
2. Fonctionnalités avancées et configuration.
3. Gestion des rôles et principales options.
4. Intégration de Docker dans une infrastructure d'applications existantes.
5. Les commandes importantes.

Les différents types de virtualisation



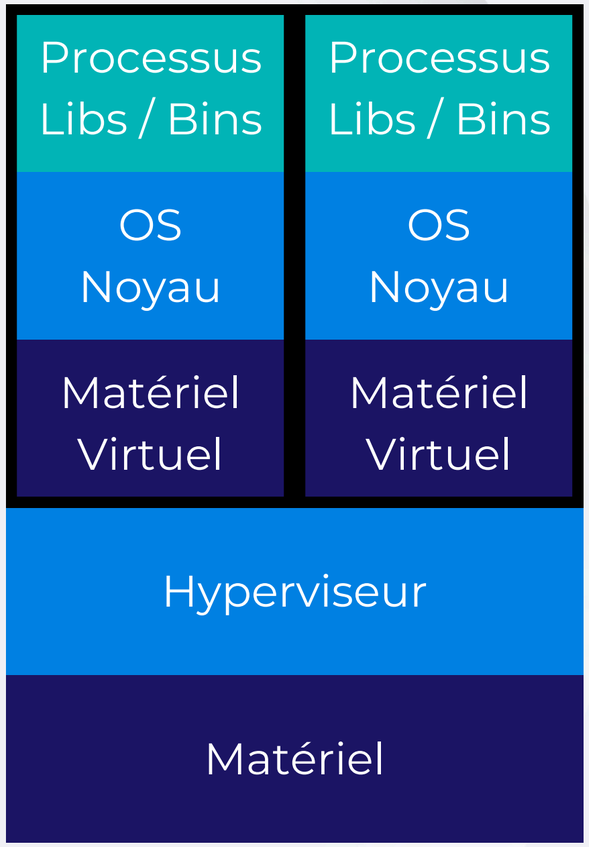
Machine virtuelle

Stockage persistant (Statefull)	Stockage non persistant (Stateless) OU Stockage persistant
Séparation des fichiers et librairie	Séparation des fichiers et librairie
Système d'exploitation indépendant (as son propre kernel)	Utilise les ressources du système d'exploitation (kernel)
Matériel virtualisé	Pas de virtualisation matérielle
Consommation importante Isolation totale	Consommation réduite Isolation partielle



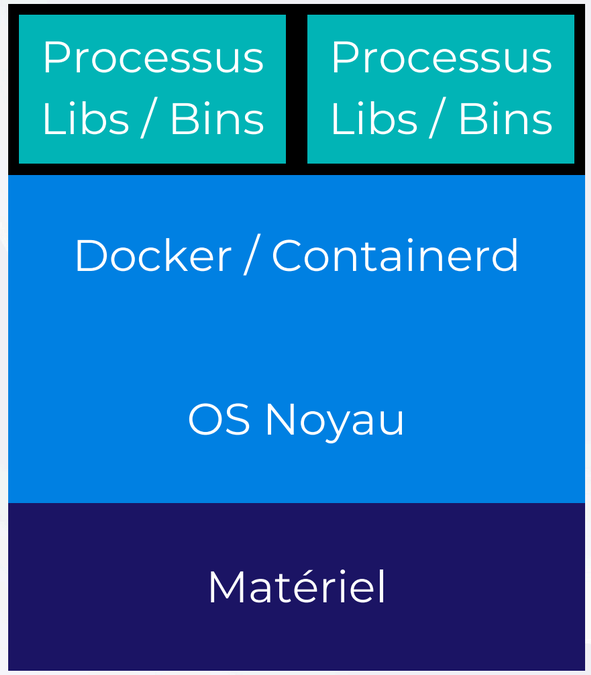
Conteneurisation

La contenerisation : Virtualisation VS Docker



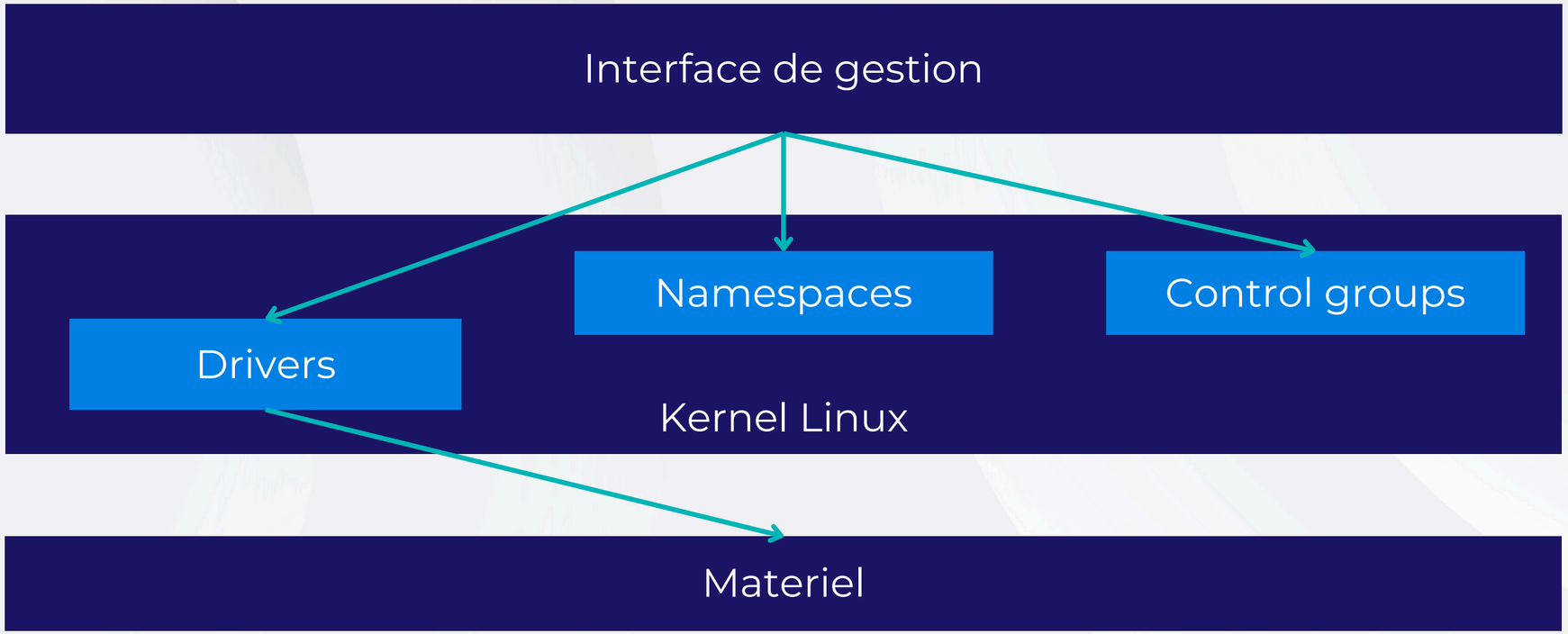
Machine virtuelle

Stockage persistant (Statefull)	Stockage non persistant par défaut (Stateless)
Séparation des fichiers et librairie	Séparation des fichiers et librairie
Système d'exploitation indépendant (as son propre kernel)	Utilise les ressources du système d'exploitation (kernel)
Matériel virtualisé	Pas de virtualisation matérielle
Consommation importante Isolation totale	Consommation réduite Isolation partielle



Conteneurisation
Docker / Containerd

La contenerisation : namespaces, control-groups et drivers



Qu’est ce que les namespaces 1/3

Définition : permettent l’isolation de ressources pour un ou plusieurs processus.

Types :

- PID : Isolation de l’identifiant de processus. Deux processus peuvent avoir un même identifiant dans deux espaces séparé.
- UTS : Isolation du nom d’hôte et de domaine.
- Mount: Isolation du système de fichier, point de montage, permettant à chaque espace d’avoir sa propre vu du système de fichier.

Qu’est ce que les namespaces 2/3

Types :

- IPC: Isolation de la communication inter-processus (mémoire partagée, semaphore, message queues).
- Network: Isolation des ressources réseaux (Interface, adresse IP, table de routage, règle de pare feu).
- User: Isolation de l’utilisateur et des groupes. Les processus de différent espace ont une vue différente des utilisateurs et groupes. Permets d’exécuter un processus avec des permissions administrateur, alors que le container a des privilèges limités sur le système hôte.

Qu’est ce que les namespaces 3/3

- Cgroup: Isolation de la vue des controlgroups permettant que les processus ne voient que le controlgroup aux quel ils appartiennent et donc sans avoir connaissances des autres présents sur le système.
- Time: Isole le temps du système permettant au processus de différent namespace d’avoir une notion du temps différente (e.g. différentes horloges)

Qu’est ce que les controlgroups

- Permet une limitation de l’usage des ressources par controlgroup.
- CPU: Limitation de l’usage du temps processeurs.
- Memory: Limitation de l’usage de la mémoire. En cas d’excès, le kernel peut exécuter des actions telles que le swapping ou tuer les processus dans le groupe.
- Block I/O: Limitation de l’usage de l’I/O du disque. Empêchant ainsi la saturation du disque.
- Network: Limitation de la bande passante réseau.
- Devices: Autorise ou restreint l’accès à des ressources matérielles (e.g. disques, interfaces réseau).
- PIDs: Limitation du nombre de processus par groupe.

Architecture du moteur Docker

Docker Daemon :

- Gère les objets Docker (images, conteneurs, volumes, réseaux).
- Communique avec la CLI via l'API REST.

Docker CLI (Command Line Interface) :

- Outil en ligne de commande pour interagir avec le daemon.
- Interface principale pour la gestion des conteneurs et des images.

Docker API :

- Interface programmée permettant de communiquer avec Docker.
- Utilisée par des outils externes pour contrôler Docker.

Images et conteneurs :

- Images : Snapshot d'un système de fichiers et d'applications.
- Conteneurs : Instances exécutables basées sur les images.

Architecture du moteur Docker

Fonctionnalités avancées et configuration

- Plugins :
  - Extensions pour ajouter de nouvelles fonctionnalités (réseau, stockage).
- Drivers de logs :
  - Permet de rediriger et stocker les logs des conteneurs vers des systèmes externes (ex. syslog, fluentd, json-file).
- Configuration des limites de ressources :
  - Limites CPU (--cpus, --cpu-shares).
  - Limites mémoire (--memory)

Gestion des rôles et permissions

- RBAC (Role-Based Access Control) :
  - Gestion des utilisateurs et des rôles dans Docker Enterprise.
  - Définit des permissions spécifiques pour chaque rôle.
- Contrôle d’accès :
  - Gestion des privilèges pour sécuriser l’accès aux conteneurs et aux images.
- Utilisateurs root et non-root :
  - Importance de ne pas exécuter les conteneurs en tant que root pour des raisons de sécurité.
  - Utilisation de l'option --user pour exécuter des processus en tant qu'utilisateur non privilégié.

Intégration de Docker dans une infrastructure existante

- Intégration avec CI/CD :
  - Docker est largement utilisé dans les pipelines d'intégration continue (Jenkins, GitLab CI).
  - Automatisation des builds, tests, et déploiements dans des conteneurs.
- Surveillance et monitoring :
  - Intégration avec des outils de monitoring tels que Prometheus, Grafana, ou ELK Stack.
  - Suivi de la santé et des performances des conteneurs en temps réel.
- Sauvegarde et récupération :
  - Stratégies de backup pour les volumes Docker.
  - Utilisation d'outils de gestion comme Docker Compose et Kubernetes pour orchestrer plusieurs conteneurs.

Les commandes Docker importantes

Gestion des conteneurs :

- docker run : Crée et démarre un nouveau conteneur.
- docker stop : Arrête un conteneur en cours d'exécution.
- docker rm : Supprime un conteneur.

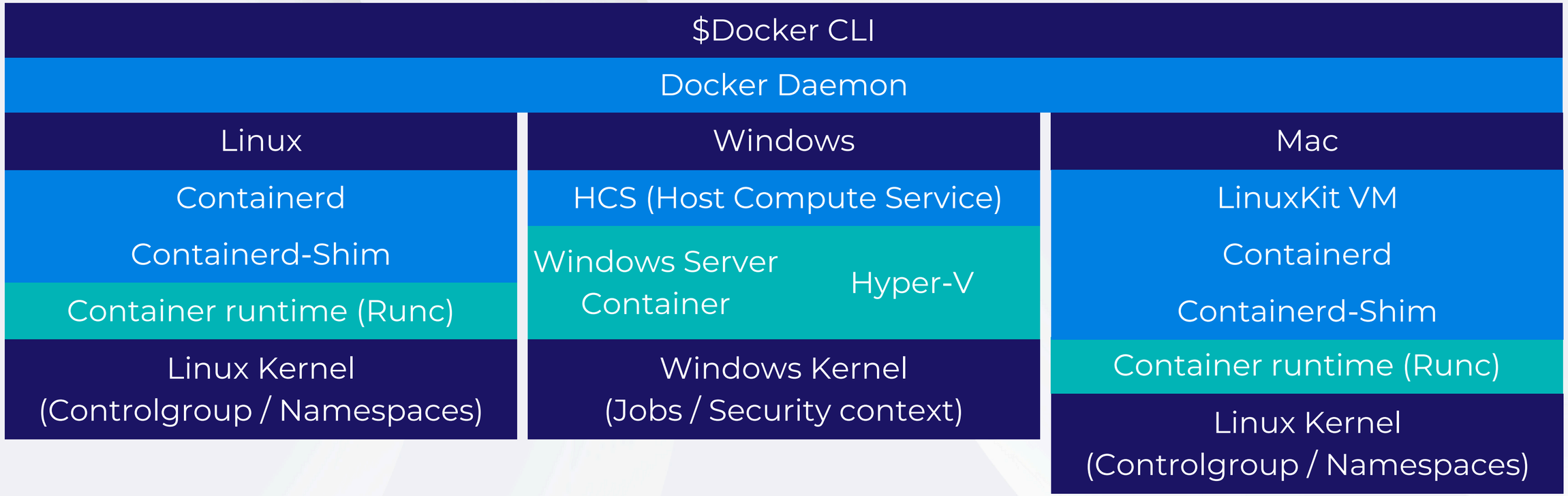
Gestion des images :

- docker pull : Télécharge une image depuis un registre (ex. Docker Hub).
- docker build : Crée une image à partir d'un Dockerfile.
- docker rmi : Supprime une image.

Surveillance et diagnostic :

- docker ps : Liste les conteneurs en cours d'exécution.
- docker logs : Affiche les logs d’un conteneur.
- docker inspect : Affiche des informations détaillées sur un conteneur ou une image.

L’architecture de Docker



paul.millet@lameduse.fr

LaMeDuSe

DCK : Docker, fonctionnalités avancées

Images et conteneurs

- Les images en détail. Gestion des images en masse.
- Transfert et portabilité des images.
- Architecture d'un conteneur.
- Bonnes pratiques et mise en œuvre.
- Les dockerfiles en détail.
- La gestion des images intermédiaires.

Images Docker en détail

- Qu’est-ce qu’une image Docker ?
  - Une image est un fichier immuable composé de plusieurs couches (layers).
  - Chaque couche est une version du fichier système ou d'une application ajoutée à l'image.
- Cycle de vie d'une image :
  - Création : via Dockerfile ou docker commit.
  - Stockage : dans un registre Docker (Docker Hub, registre privé).
  - Utilisation : déploiement d'un conteneur via docker run.
- Images parent :
  - Les images peuvent être basées sur d’autres images (ex : Alpine, Ubuntu).
  - Les images parent fournissent la base du système.

Gestion des images en masse

Téléchargement et suppression d'images :

- docker pull <image> : Récupérer une image d’un registre.
- docker rm <image> : Supprimer une image locale.

Nettoyage des images inutilisées :

- docker image prune : Supprimer les images non utilisées.
- docker system prune : Supprimer les conteneurs, réseaux et images inutilisés.

Compression des images :

- Réduire la taille des images avec des bases légères (Alpine).
- Utilisation des multi-stage builds pour minimiser la taille des images finales.

Transfert et portabilité des images

Transfert d'images :

- Utilisation de registres publics (Docker Hub) ou privés pour partager des images.
- Commandes : docker push, docker pull.

Portabilité :

- Les images Docker peuvent être transférées entre différents environnements (local, cloud, on-premise).
- Utilisation de l'option --platform pour cibler des architectures spécifiques (x86, ARM).

Partage via fichiers tar :

- docker save et docker load pour sauvegarder et restaurer des images à partir de fichiers tar.

Architecture d'un conteneur

Fonctionnement d'un conteneur :

- Un conteneur partage le noyau du système hôte mais exécute son propre espace utilisateur.

Isolation des conteneurs :

- Namespaces : Permettent l’isolation des processus, du réseau et des utilisateurs.
- Cgroups : Limitent l'utilisation des ressources (CPU, mémoire).

Performance des conteneurs :

- Légers et rapides, car ils partagent le noyau du système hôte, contrairement aux machines virtuelles.

Comparaison avec une VM :

- Les conteneurs démarrent en quelques secondes et sont plus efficaces en termes de ressources.

Bonnes pratiques et mise en œuvre

Minimisation de l'image :

- Utiliser des images légères comme Alpine.
- Limiter le nombre de couches dans le Dockerfile.

Sécurité :

- Ne pas exécuter les conteneurs en tant que root.
- Utiliser des images signées pour garantir leur intégrité (Docker Content Trust).

Stateless vs Stateful :

- Privilégier les conteneurs stateless (sans état).
- Utiliser des volumes pour stocker les données persistantes à l'extérieur du conteneur.

Les dockerfiles en détail

Principales instructions Dockerfile :

- FROM : Définit l'image de base.
- COPY et ADD : Ajout de fichiers dans l'image.
- RUN : Exécute une commande pendant la construction.
- CMD et ENTRYPOINT : Commandes exécutées lors du démarrage du conteneur.

Optimisation des Dockerfiles :

- Minimiser les couches (RUN combinées).
- Utiliser le cache Docker pour éviter de reconstruire les couches inchangées.

Gestion des images intermédiaires

Images intermédiaires :

- Chaque instruction dans un Dockerfile crée une nouvelle couche intermédiaire.

Réduire la taille des images :

- Utiliser les multi-stage builds pour supprimer les couches inutiles.
- Combiner plusieurs commandes dans une seule instruction RUN pour réduire le nombre de couches.

Suppression des images intermédiaires :

- docker image prune --filter "dangling=true" pour supprimer les images intermédiaires non utilisées.
-

paul.millet@lameduse.fr

LaMeDuSe

DCK : Docker, fonctionnalités avancées

Le réseau sous-jacent

- Architecture du réseau Docker.
- Administration avancée du réseau virtuel.
- Mise en réseau (Intra/inter et conteneurs/hôtes).
- Concevoir des réseaux scalables et portables.

Architecture du réseau Docker

Types de réseaux Docker :

- Bridge : Réseau par défaut pour les conteneurs sur le même hôte.
- Host : Le conteneur partage l'interface réseau de l'hôte, pas d'isolation.
- Overlay : Réseau utilisé pour la communication entre plusieurs hôtes (nécessite Docker Swarm ou Kubernetes).
- None : Pas de réseau attribué au conteneur (isolation complète).

Fonctionnement des réseaux Docker :

- Chaque conteneur connecté à un réseau possède une adresse IP unique.
- Docker utilise le DNS interne pour la résolution des noms entre conteneurs.

Architecture du réseau Docker

Gestion des réseaux :

- Création : docker network create <nom_du_reseau>.
- Visualisation : docker network ls pour lister les réseaux disponibles.
- Connexion d'un conteneur à un réseau : docker network connect <nom_du_reseau> <nom_du_conteneur>.

Configuration avancée :

- Spécifier des sous-réseaux et des plages d'adresses IP personnalisées.
- Isolation des réseaux pour garantir la sécurité entre les services.

DNS interne Docker :

- Docker DNS permet la résolution de noms automatiques entre conteneurs dans le même réseau.
- Les conteneurs peuvent se connecter via leur nom au lieu de leur adresse IP.

Mise en réseau entre conteneurs (Intra/inter hôte)

Réseautage intra-hôte :

- Les conteneurs sur un même hôte peuvent communiquer via le réseau bridge.
- Utilisation des liens entre conteneurs pour la communication (--link).

Réseautage inter-hôte :

- Les réseaux overlay permettent la communication entre des conteneurs situés sur différents hôtes dans un cluster Docker Swarm ou Kubernetes.
- Requiert l'activation du mode Swarm ou un orchestrateur.

Exposition des ports :

- Utilisation de l'option -p ou --publish pour mapper des ports du conteneur vers l'hôte.

Mise en réseau des conteneurs avec l’hôte

Réseau host :

- Le conteneur partage l'interface réseau de l'hôte.
- Pas d'isolation réseau, utile pour des applications nécessitant des performances réseau optimales.

Utilisation de la pile réseau hôte :

- Peut améliorer la latence réseau pour les applications nécessitant un accès direct aux interfaces réseau de l’hôte.

Scénarios d’utilisation :

- Services nécessitant un accès direct aux interfaces réseau de l’hôte, comme les outils de surveillance et de logging.

Concevoir des réseaux scalables et portables.

Réseaux scalables :

- Utilisation de Docker Swarm pour créer des réseaux overlay pour plusieurs hôtes.
- Utilisation d'outils de monitoring pour suivre les performances et ajuster les configurations réseau.

Segmentation du réseau :

- Isoler des services dans des sous-réseaux pour améliorer la sécurité et la gestion.

Exemples de conception :

- Réseaux overlay avec Docker Swarm pour interconnecter des services répartis sur plusieurs hôtes.
- Utilisation de proxys inversés (comme Traefik ou NGINX) pour diriger le trafic entre plusieurs services conteneurisés.

4 - Le stockage Docker, mise en haute disponibilité et en production

paul.millet@lameduse.fr

LaMeDuSe

DCK : Docker, fonctionnalités avancées

Le stockage Docker, mise en haute disponibilité et en production

- Les différents types de stockage.
- Mise en œuvre et configuration.
- Déploiement et gestion des conteneurs multiples.
- Mise en œuvre d'une architecture en haute disponibilité.
- Gestion des microservices.
- Orchestration et automatisation des processus Docker

Les différents types de stockage Docker

Volumes :

- Les volumes Docker sont le moyen privilégié pour stocker des données persistantes en dehors du cycle de vie des conteneurs.
- Commandes : docker volume create, docker volume ls, docker volume rm.

Bind mounts :

- Lier un dossier spécifique de l'hôte à un conteneur.
- Utilisation de chemins spécifiques pour monter des répertoires de l'hôte dans le conteneur.

Tmpfs (tempfs) :

- Stockage en mémoire, très rapide mais non persistant.
- Utilisé pour stocker des données temporaires sensibles ou volatiles.

Les différents types de stockage Docker

Choisir entre les options :

- Les volumes pour les données persistantes et partageables entre conteneurs.
- Les bind mounts pour accéder à des fichiers spécifiques sur l'hôte.
- Tmpfs pour la vitesse et les données temporaires.

Mise en œuvre et configuration du stockage

- Création et utilisation des volumes :
  - docker volume create <nom_du_volume> : Créer un volume.
  - docker run -v <nom_du_volume>:/path : Monter un volume dans un conteneur.
- Gestion des volumes partagés :
  - Partager un volume entre plusieurs conteneurs pour stocker des fichiers communs (ex. : bases de données, fichiers de configuration).
- Sauvegarde et restauration des volumes :
  - Sauvegarde avec docker run --rm -v <nom_du_volume>:/volume -v $(pwd):/backup busybox tar cvf /backup/backup.tar /volume.
  - Restauration avec docker run --rm -v <nom_du_volume>:/volume -v $(pwd):/backup busybox tar xvf /backup/backup.tar.

Déploiement et gestion de conteneurs multiples

Gestion de plusieurs conteneurs :

- Utilisation de Docker Swarm pour gérer et orchestrer des ensembles de conteneurs avec des services interconnectés.

Isolation des services :

- Chaque service dans Docker Swarm peut avoir ses propres paramètres de réseau, de stockage, et de ressources.

Stratégies de redémarrage et mise à jour :

- Utilisation des politiques de redémarrage (restart: always) pour garantir la résilience des services.
- Utiliser des mises à jour en rolling pour minimiser les interruptions lors des mises à jour des conteneurs.

Mise en œuvre d’une architecture en haute disponibilité

- Docker Swarm et Kubernetes :
  - Ces outils permettent de gérer des clusters de conteneurs et assurent une haute disponibilité en cas de défaillance d’un nœud.
- Répartition de charge (load balancing) :
  - Docker Swarm dispose de mécanismes de répartition de charge intégrés pour distribuer les requêtes sur plusieurs nœuds.
  - Kubernetes utilise un Service pour équilibrer la charge entre les réplicas.

Mise en œuvre d’une architecture en haute disponibilité

- Failover automatique :
  - En cas de panne d’un nœud, Docker Swarm ou Kubernetes réassignent automatiquement les services à d’autres nœuds disponibles.
- Persistances des données :
  - Utiliser des volumes partagés entre nœuds pour garantir que les données restent accessibles, même en cas de défaillance d’un conteneur.

Gestion des microservices avec Docker

Microservices :

- Chaque service d'une application est séparé dans un conteneur Docker distinct, interconnecté via un réseau interne.

Avantages des microservices :

- Déploiement indépendant de chaque composant.
- Scalabilité fine par service.
- Faible couplage entre services, facilitant la maintenance.

Orchestration des microservices :

- Docker Compose pour des petites architectures multi-conteneurs.
- Docker Swarm ou Kubernetes pour orchestrer des applications distribuées et scalables.

Orchestration et automatisation des processus Docker

Orchestration avec Docker Swarm :

- Docker Swarm permet de gérer des clusters et des nœuds multiples pour une architecture distribuée.
- Commandes : docker swarm init, docker service create, docker node ls.

Automatisation du déploiement :

- Utilisation de pipelines CI/CD pour automatiser la construction, les tests, et le déploiement des conteneurs.
- Intégration avec des outils comme Jenkins, GitLab CI pour automatiser le processus de livraison continue.

Outils d’automatisation :

- Utilisation d’Ansible, Terraform, ou Helm pour automatiser le provisionnement des infrastructures conteneurisées.

paul.millet@lameduse.fr

LaMeDuSe

DCK : Docker, fonctionnalités avancées

Docker Compose

- Architecture de Docker Compose.
- Mise en œuvre et administration de Docker Compose.
- Notions avancées de Docker Compose.

Architecture de Docker Compose

Fichier docker-compose.yml :

- Définit les services, réseaux et volumes pour une application multi-conteneurs.
- Utilise le format YAML pour décrire les configurations.

Services :

- Représentent les conteneurs dans l'application.
- Chaque service peut être configuré avec ses propres options, comme les ports exposés, les volumes montés, et les variables d’environnement.

Réseaux et Volumes :

- Définis dans le même fichier pour faciliter la gestion des connexions entre services et des données persistantes.
- Docker Compose crée et gère ces ressources automatiquement.

Mise en œuvre de Docker Compose

- Création d'un fichier docker-compose.yml :
  - Définir des services : Exemple de service web avec une image spécifique et des variables d’environnement.
  - Définir des réseaux : Exemple de réseau par défaut ou personnalisé pour les services.
  - Définir des volumes : Exemple de volumes pour stocker les données persistantes.
- Commandes de base :
  - docker-compose up : Démarre les services définis dans le fichier.
  - docker-compose down : Arrête et supprime les services, réseaux et volumes.
  - docker-compose logs : Affiche les logs des services.

Mise en œuvre de Docker Compose

Gestion des services :

- docker-compose ps : Affiche l'état des conteneurs gérés par Docker Compose.
- docker-compose restart : Redémarre les services.

Déploiement :

- Utilisation des options -d pour démarrer les services en arrière-plan.
- Utilisation des options --build pour forcer la reconstruction des images avant le démarrage.

Configuration et environnements :

- Utilisation de fichiers .env pour gérer les variables d’environnement.
- Création de configurations pour différents environnements (développement, test, production) avec des fichiers YAML spécifiques.

Création d'un fichier YAML de configuration (exemple)

```
services:
  db:
    image: mariadb:10.6.4-focal #définie l'image
    command: '--default-authentication-plugin=mysql_native_password' #Option CMD dans docker run
    volumes:
      - db_data:/var/lib/mysql #definie l'utilisation et le montage du volume db_data
    restart: always #stratégie de redémarrage
    environment:  #variable d'environnement
      - MYSQL_ROOT_PASSWORD=somewordpress
      ...
    expose: #quel port sont exposés pour les autres services
      - 3306
  wordpress:
    image: wordpress:latest
    ports: #quel port sont exposés publiquement
      - 80:80
    restart: always
    environment:
      - WORDPRESS_DB_HOST=db
      ...
volumes: #définition du volume
  db_data:
```

Notions avancées de Docker Compose

Utilisation des réseaux personnalisés :

- Définition et configuration de réseaux pour isoler les services ou les regrouper logiquement.
- Exemples d'utilisation de réseaux bridge ou overlay.

Volumes partagés entre services :

- Partage de volumes entre plusieurs services pour des données communes.
- Configuration des volumes pour une meilleure gestion des données persistantes.

Notions avancées de Docker Compose

Déploiement avec Docker Swarm :

- Utilisation de docker-compose avec Docker Swarm pour déployer des applications multi-conteneurs en cluster.
- Commande : docker stack deploy -c docker-compose.yml <stack_name>

paul.millet@lameduse.fr

LaMeDuSe

DCK : Docker, fonctionnalités avancées

LaMeDuSe

Docker Swarm

- Architecture de Docker Swarm.
- Les différents types de nœuds.
- Gestion des logs et surveillance.
- Mise en œuvre et administration.

Rappel : Docker Swarm

Qu'est-ce que Docker Swarm ?

1. Outil natif de Docker pour l’orchestration et la gestion de clusters de conteneurs.
2. Permet de déployer et de gérer des applications distribuées et haute disponibilité à l’échelle.

Technologie connexe :

- Kubernetes

Architecture Docker Swarm

Nœuds :

- Manager Nodes : Responsables de la gestion du cluster, de la planification des tâches et du maintien de l'état global.
- Worker Nodes : Exécutent les tâches et les services attribués par les nœuds managers.

Services :

- Représentent des applications conteneurisées déployées dans le cluster.
- Les services sont répartis sur les différents nœuds pour assurer la haute disponibilité et la scalabilité.

Architecture Docker Swarm

Tasks :

- Les tâches sont les instances de conteneurs créées pour exécuter les services.

Overlay Network :

- Permet la communication entre les conteneurs répartis sur différents hôtes du cluster.

Architecture Docker Swarm

Manager Node :

- Coordonne les activités dans le cluster.
- Gère l'état du cluster et les services en utilisant un consensus distribué (Raft).

Worker Node :

- Exécute les conteneurs et les tâches distribuées par les nœuds managers.
- Reçoit les instructions de déploiement et de gestion des services.

Promotions et démotions :

- Les nœuds workers peuvent être promus en managers et vice versa.
- Commandes : docker node promote, docker node demote.

Architecture Docker Swarm

Gestion des logs :

- Docker Swarm centralise les logs des conteneurs pour faciliter la gestion et l'analyse.
- Utilisation de pilotes de logs (par exemple, json-file, syslog, fluentd).

Outils de surveillance :

- Docker Stats : docker stats pour obtenir des statistiques en temps réel des conteneurs.
- Prometheus et Grafana : Pour une surveillance avancée et une visualisation des performances du cluster.

Configuration des alertes :

- Mise en place d’alertes basées sur les métriques collectées pour surveiller l’état du cluster et des services.

Architecture Docker Swarm

Initialisation d’un cluster :

- docker swarm init pour configurer un nœud comme manager.
- docker swarm join pour ajouter des nœuds workers au cluster.

Déploiement des services :

- Utilisation de docker service create pour déployer des services dans le cluster.
- Gestion des mises à jour avec docker service update et des rollbacks.

Scalabilité :

- Utilisation de docker service scale pour ajuster le nombre de répliques d’un service.
- Gestion des ressources avec les options --limit-cpu et --limit-memory.

Architecture Docker Swarm

Sécurité :

- Utilisation de TLS pour sécuriser les communications entre les nœuds.
- Gestion des secrets et des configurations sensibles via docker secret et docker config.

Répartition des tâches :

- Utilisation des stratégies de placement pour optimiser la répartition des tâches.
- Définition des contraintes pour éviter les conflits et optimiser l'utilisation des ressources.

Résilience :

- Configuration des politiques de redémarrage pour les services (--restart-condition).
- Surveillance continue et gestion proactive des incidents.

paul.millet@lameduse.fr

LaMeDuSe

DCK : Docker, fonctionnalités avancées

Mise en œuvre d'un registre

- Introduction aux différents types de registres.
- Déploiement de registres.
- Notions de découvertes de services et de load-balancing avec UCP.
- Notions "DTR" et "DDC".
- Signature des objets.

Types de registres Docker

Docker Hub :

- Registre public officiel de Docker avec un accès aux images publiques et privées.
- Possibilité de créer des dépôts privés pour des images confidentielles.

Registres privés :

- Permet de déployer un registre Docker interne pour une utilisation en entreprise.
- Utilisation pour stocker des images sensibles ou internes à une organisation.

Solution de registre

- Docker Hub (freemium/cloud) : Registre officiel de docker
- Docker Registry (oss/selfhosted) : Solution de registre de Docker open-source (pas de dashboard)
- Harbor (oss/selfhosted) : Solution de registre faite par VMware (dashboard + option de sécurité)
- ghcr.io (freemium/cloud) : Registre propulsé par GitHub
- GitLab container registry (freemium/cloud & hosted) :
    - Registre propulsé par GitLab

Déploiement d’un registre privé Docker

Installation et configuration :

- Commande : docker run -d -p 5000:5000 --name registry registry:2.
- Ce conteneur exécute un registre local sur le port 5000.

Accès et sécurité :

- Configuration de l'accès sécurisé avec HTTPS pour garantir la confidentialité et l'intégrité des images.
- Utilisation d'un certificat SSL pour sécuriser les communications avec le registre.

Pousser et tirer des images :

- Pousser une image : docker tag <image> localhost:5000/<image> puis docker push localhost:5000/<image>.
- Tirer une image : docker pull localhost:5000/<image>.

Découverte de services et de load balancing avec UCP

Universal Control Plane (UCP) :

- Solution de gestion d'infrastructure conteneurisée de Docker Enterprise.
- Offre des fonctionnalités de gestion et de sécurité avancées pour Docker Swarm et Kubernetes.

Découverte de services :

- Permet aux services de trouver automatiquement d'autres services dans le réseau sans configuration manuelle.
- Facilite la communication entre les services au sein d'un cluster.

Découverte de services et de load balancing avec UCP

Répartition de charge (load balancing) :

- UCP offre des fonctionnalités de répartition de charge intégrées.
- Assure que le trafic est réparti équitablement entre les conteneurs et améliore la résilience de l’application.

Introduction à Docker Trusted Registry (DTR)

Qu’est-ce que Docker Trusted Registry (DTR) ?

- Registre Docker sécurisé, utilisé pour stocker et gérer des images dans Docker Enterprise.
- Offre des fonctionnalités de contrôle d'accès, de gestion des vulnérabilités et de validation des images.

Fonctionnalités principales :

- Contrôle d'accès basé sur les rôles (RBAC) pour la gestion des utilisateurs et des permissions.
- Scans de sécurité intégrés pour détecter les vulnérabilités dans les images.

Intégration avec Docker Enterprise :

- DTR s'intègre parfaitement avec UCP pour offrir une solution complète de gestion des images et des clusters.

Notions DDC (Docker Data Center)

Docker Data Center :

- Ensemble de solutions de Docker Enterprise, incluant UCP, DTR et des outils pour gérer les conteneurs en production.

Fonctionnalités clés :

- Gestion centralisée des images et des conteneurs avec une interface graphique conviviale.
- Sécurité avancée avec la validation d’image et la gestion des secrets.
- Outils d'intégration pour CI/CD, permettant l'automatisation des processus de déploiement.

Signature des objets dans Docker

Docker Content Trust (DCT) :

- Permet de signer et de vérifier l'intégrité des images Docker.
- Garantit que seules des images signées et validées sont déployées dans l'infrastructure.

Utilisation de DCT :

- Activer DCT avec export DOCKER_CONTENT_TRUST=1.
- Signer une image : docker trust sign <image>.
- Vérification automatique lors de l'extraction d'une image : Docker vérifie que l'image est signée et correspond à la signature attendue.

Signature des objets dans Docker

Avantages :

- Sécurité renforcée pour éviter l’utilisation d’images compromises ou modifiées.
- Assurance que seules des images authentifiées sont déployées en production.

paul.millet@lameduse.fr

LaMeDuSe

DCK : Docker, fonctionnalités avancées

La sécurité dans Docker et monitoring

- Vue d'ensemble des bonnes pratiques de sécurité dans Docker.
- Configuration des principales bonnes pratiques.
- Utilisation des modules de sécurisation.
- Gestion des vulnérabilités.
- Gestion des isolations et des limitations.
- Les outils d'analyses du monitoring.
- Les logs du daemon Docker.

La sécurité dans Docker et monitoring

- Utiliser des images sécurisées et vérifiées :
  - Privilégier les images officielles et celles provenant de sources fiables.
  - Scanner régulièrement les images pour détecter des vulnérabilités.
- Gérer les utilisateurs et permissions :
  - Éviter d'exécuter les conteneurs avec des privilèges root.
  - Utiliser des rôles spécifiques pour limiter les actions disponibles à l'intérieur des conteneurs.
- Maintenir Docker à jour :
  - Mettre à jour régulièrement Docker pour bénéficier des dernières corrections de sécurité.
  - Utiliser des outils comme Docker Bench pour auditer la configuration de sécurité.

La sécurité dans Docker et monitoring

Configuration des politiques de redémarrage :

- Définir des politiques de redémarrage pour gérer les défaillances inattendues des conteneurs.
- Utiliser l'option --restart pour automatiser le redémarrage des conteneurs.

Sécuriser les communications réseau :

- Utiliser TLS pour sécuriser les communications entre le daemon Docker et les clients.
- Limiter les accès réseau externes aux conteneurs en configurant les règles de pare-feu (firewall).

avancées.

La sécurité dans Docker et monitoring

Contrôle d'accès basé sur les rôles (RBAC) :

- Implémenter des règles RBAC pour restreindre les actions que les utilisateurs peuvent exécuter sur Docker.
- Utilisation de Docker Enterprise pour des fonctionnalités de RBAC

Modules de sécurisation dans Docker

Seccomp (Secure Computing Mode) :

- Permet de restreindre les appels système qu'un conteneur peut faire.
- Utilisation de profils personnalisés pour restreindre les capacités d'un conteneur.

AppArmor :

- Module de sécurité Linux qui limite les actions des processus.
- Les profils AppArmor appliquent des restrictions spécifiques aux conteneurs.

SELinux (Security-Enhanced Linux) :

- Un autre mécanisme de sécurité qui applique des politiques de contrôle d'accès obligatoires.
- Assure une séparation stricte entre les processus à l'intérieur et à l'extérieur du conteneur.

Gestion des vulnérabilités

Scan d'images Docker :

- Utilisation de Docker Hub ou d’outils comme Clair, Anchore ou Trivy pour scanner les images Docker et détecter les vulnérabilités connues.

Mise à jour régulière des images :

- Recréer et pousser régulièrement les images Docker après correction des vulnérabilités.
- Automatiser le scan des images avec des pipelines CI/CD.

Politique de gestion des dépendances :

- Identifier et gérer les dépendances dans les images pour éviter des versions vulnérables.
- Utilisation d’outils de gestion des dépendances pour surveiller les bibliothèques et packages dans les images.

Gestion des isolations et des limitations

Isolation des conteneurs :

- Utilisation des espaces de noms (namespaces) pour isoler les processus, les utilisateurs et les réseaux entre les conteneurs.
- Limiter l'accès aux ressources du système hôte pour éviter les intrusions.

Limitation des ressources :

- Utiliser les cgroups pour limiter les ressources CPU, mémoire, et E/S des conteneurs.
- Commande : docker run --memory, docker run --cpus pour limiter l'usage des ressources par conteneur.

Isolation réseau :

- Configurer des réseaux personnalisés pour isoler les conteneurs et contrôler leur accès aux autres réseaux.

Outils d’analyse et de monitoring de Docker

Docker Stats :

- Permet de surveiller l’utilisation des ressources des conteneurs en temps réel : CPU, mémoire, disque et réseau.
- Commande : docker stats.

Outils de monitoring :

- Prometheus : Collecte les métriques du daemon Docker et des conteneurs pour surveiller les performances.
- Grafana : Visualisation des données de Prometheus avec des tableaux de bord personnalisables.

Outils d’analyse et de monitoring de Docker

Intégration avec ELK Stack :

- Elasticsearch, Logstash, Kibana (ELK) : Outils pour collecter, analyser et visualiser les logs des conteneurs Docker.
- Permet d’avoir une vue centralisée sur les événements des conteneurs et les performances.

Les logs du daemon Docker

Importance des logs dans Docker :

- Les logs permettent d'identifier les problèmes, d'analyser les événements passés et de surveiller les actions des conteneurs et du daemon.

Accéder aux logs du daemon Docker :

- docker logs <container_id> pour afficher les logs d'un conteneur spécifique.
- Utilisation de drivers de logs (ex. json-file, syslog, fluentd) pour configurer et centraliser les logs.

Gestion des logs à grande échelle :

- Utilisation de systèmes de gestion de logs distribués comme Graylog ou Splunk pour des environnements de production à grande échelle.

paul.millet@lameduse.fr

LaMeDuSe

DCK : Docker, fonctionnalités avancées

Ressources des TP

1. TP : Docker Swarm
2. TP : Docker + Seccomp
3. TP : Docker + Apparmor
4. TP : Docker + Trivy
5. TP : Docker Stats
6. TP : Docker + Graphana + Prometheus
7. TP : Docker + Logstash + ElasticSearch + Kibana

1- TP: Déploiment de docker swarm

Sujet 1 : Déploiment de docker swarm

- Création noeud maitre
- Création noeud travail x 2
- Vérifier le résultat

2- TP: Docker + Seccomp

Sujet 1 :

- Vérifier les appels système supportés par le noyau :
  - Utiliser la commande man 2 syscalls pour lister les appels système pris en charge par le noyau Linux.
- Créer un profil Seccomp personnalisé :
  - Créez un fichier seccomp_profile.json avec le contenu suivant pour autoriser uniquement certains appels système basiques :
  - https://blog.lamedusegroup.com/courses
- docker run --security-opt seccomp=./seccomp_profile.json -d nginx
- Vérifier le résultat

3- TP: Docker + AppArmor

Sujet 1 :

- Créer un profil AppArmor simple :
  - Créez un fichier /etc/apparmor.d/docker-nginx-profile avec le contenu suivant :
  - https://blog.lamedusegroup.com/courses
- sudo apparmor_parser -r /etc/apparmor.d/docker-nginx-profile
- docker run --security-opt apparmor=docker-nginx-profile -d nginx
- Vérifier le résultat (ping 8.8.8.8)

4- TP: Docker + Trivy

Sujet 1 :

- Installer Trivy
  - sudo apt-get install -y trivy
- Télécharger une image
  - docker pull nginx:1.17
- Executer trivy
  - trivy image nginx:1.17
-

5- TP: Docker Stats

Sujet 1 :

- Executer deux container
  - docker run -d --name webserver nginx
  - docker run -d --name dbserver postgres
- docker stats

6- TP: Docker + Prometheus + Grafana

Sujet 1 :

- Configurer Prometheus pour surveiller Docker :
  - Installez Prometheus et ajoutez une configuration pour surveiller Docker avec le plugin cAdvisor :
  - Créer ./prometheus.yml
    - https://blog.lamedusegroup.com/courses
  - docker run -d --name prometheus -p 9090:9090 -v ./prometheus.yml:/etc/prometheus/prometheus.yml prom/prometheus
- Accédez à l’interface Web de Prometheus via http://localhost:9090 et assurez-vous que le job cadvisor apparaît et collecte des données.

6- TP: Docker + Prometheus + Grafana

Sujet 2 :

- cAdvisor (Container Advisor) est un outil qui collecte les métriques des conteneurs Docker, telles que l’utilisation de la CPU, de la mémoire, et du réseau.
- Lancer cAdvisor
  - Commande sur https://blog.lamedusegroup.com/courses

6- TP: Docker + Prometheus + Grafana

Sujet 3 :

- Executer grafana
  - docker run -d --name=grafana -p 3000:3000 grafana/grafana
  - Accédez à l’interface Web de Grafana via http://localhost:3000. Connectez-vous avec les identifiants par défaut (admin / admin).
- Ajouter Prometheus comme source de données :
  - Dans l'interface Grafana :
  - Cliquez sur "Configuration" puis sur "Data Sources".
  - Ajoutez une nouvelle source de données avec le type "Prometheus".
  - Dans l’URL, entrez http://localhost:9090 pour connecter Grafana à Prometheus.

6- TP: Docker + Prometheus + Grafana

Sujet 3bis :

- Dans Grafana, allez dans "Dashboards" -> "Import".
- Utilisez un tableau de bord Docker déjà disponible en ligne. Vous pouvez par exemple utiliser l'ID 893 de Grafana (Docker and system monitoring).
- Cela permet de visualiser instantanément les métriques CPU, mémoire, disque et réseau des conteneurs Docker.

7- TP: Docker + ELK Stack

Sujet 1:

- Configurer Docker pour utiliser json-file comme driver de log
  - Modifiez le fichier /etc/docker/daemon.json pour spécifier le driver de log json-file avec des options de rotation des logs :
  - https://blog.lamedusegroup.com/courses-dck
- Redémarrer docker : sudo systemctl restart docker
- Executer elasticsearch
  - docker run -d --name elasticsearch -p 9200:9200 -e "discovery.type=single-node" elasticsearch:7.9.3
- Executer Logstash : https://blog.lamedusegroup.com/courses-dck
- Executer Kibana
  - docker run -d --name kibana -p 5601:5601 kibana:7.9.3
- Créer le réseau docker (docker network create / connect)

paul.millet@lameduse.fr

LaMeDuSe

DCK : Docker, fonctionnalités avancées

Sources

Sources :

- https://github.com/containerd/containerd/blob/main/core/runtime/v2/README.md

Q&A

Q: A quoi corresponds le contexte ?
A: Le contexte est l'endroit où se situent tous les fichiers de votre application nécessaires à la construction de votre container.
Le contexte est un répertoire qui souvent contient à sa racine un fichier dockerfile.
Q: Où se situes les volumes ?
A: /var/lib/docker/volumes

Docker Desktop + Ubuntu 24.04

Fix: sudo sysctl -w kernel.apparmor_restrict_unprivileged_userns=0

Instruction Docker avancée

- HEALTHCHECK Check a container's health on startup.
- LABEL Add metadata to an image.
- MAINTAINER Specify the author of an image.
- ONBUILD Specify instructions for when the image is used in a build.
- STOPSIGNAL Specify the system call signal for exiting a container.
- VOLUME Create volume mounts.

Création d'un fichier YAML de configuration (Options)

```yaml
version: '3.8'  # Version du format de fichier Docker Compose.
services:
  app:
    image: nginx:latest  # Spécifie l'image Docker à utiliser pour le conteneur.
    build:
      context: ./app  # Répertoire pour la construction de l'image.
    command: "nginx -g 'daemon off;'"  # Commande à exécuter dans le conteneur.
    environment:
      - DEBUG=true  # Variable d'environnement pour le conteneur.
    ports:
      - "8080:80"  # Mappage des ports (hôte:conteneur).
    volumes:
      - app-data:/usr/share/nginx/html  # Volume monté dans le conteneur.
    networks:
      - webnet  # Réseau auquel se connecter.
    restart: always  # Politique de redémarrage du conteneur.
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost/"]  # Test de la santé du service.
    logging:
      driver: json-file  # Pilote de journalisation.
    user: "1000:1000"  # Utilisateur pour exécuter le conteneur.
    working_dir: /usr/src/app  # Répertoire de travail dans le conteneur.
  db:
    image: postgres:13  # Image Docker pour le service de base de données.
    environment:
      POSTGRES_PASSWORD: example  # Mot de passe pour PostgreSQL.
    volumes:
      - db-data:/var/lib/postgresql/data  # Volume pour les données de la base de données.
    networks:
      - webnet  # Réseau auquel se connecter.
# Définition des volumes utilisés par les services
volumes:
  app-data:  # Volume pour l'application.
  db-data:   # Volume pour la base de données.
# Définition des réseaux personnalisés
networks:
  webnet:  # Réseau pour la communication entre services.
    driver: bridge  # Pilote de réseau.
```