Docker – Perfectionnement et usages avancés

Formateur : Halim BOUHOUI

Objectifs du cours

Ce module vise la maîtrise avancée de Docker dans des contextes proches de la production.
Compétences visées :

- administration et configuration du Docker Engine,
- sécurisation des flux et de l'API Docker,
- déploiement dans un environnement distribué,
- sécurisation et gouvernance des images,
- orchestration et micro-services.

Plan du cours

Docker en environnement professionnel

Docker est aujourd'hui utilisé bien au-delà du simple développement local. Il est présent :

- dans les pipelines CI/CD,
- en environnement de production,
- dans des architectures distribuées,
- dans des contextes critiques.

Usage simple vs usage avancé

Usage simple :

- exécution locale de conteneurs,
- tests fonctionnels.

Usage avancé :

- sécurité,
- orchestration,
- haute disponibilité,
- gouvernance des images.

Architecture Docker

Docker repose sur trois composants :

- le client Docker,
- le démon Docker,
- l'API Docker.

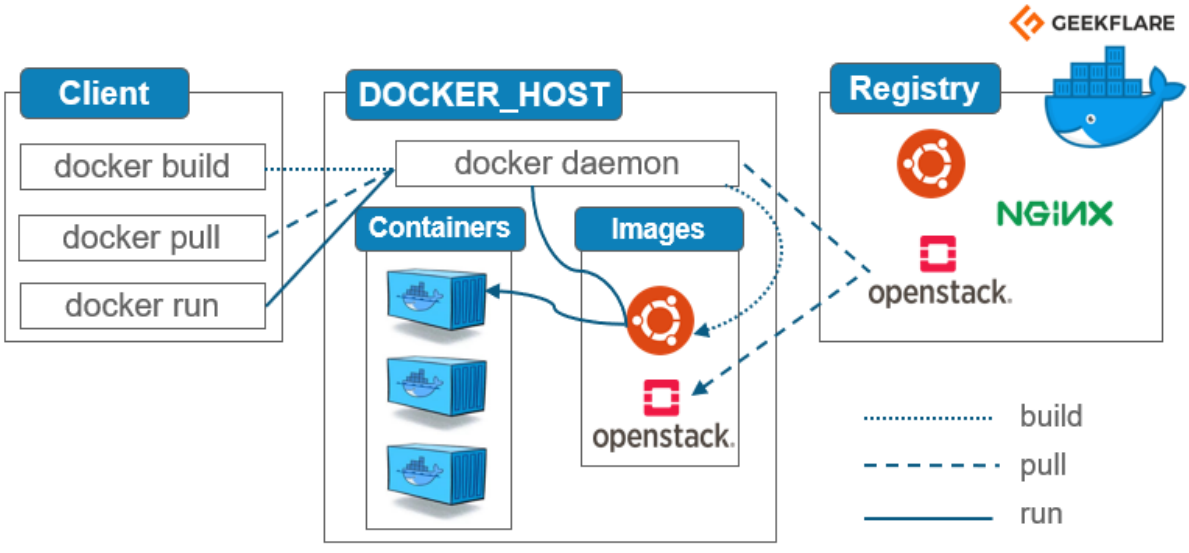
Le démon est responsable de toutes les opérations critiques.

Rôle du démon Docker

Le démon Docker :

- gère les images,
- orchestre les conteneurs,
- administre les réseaux et volumes,
- expose une API puissante.

Un accès non maîtrisé au démon équivaut à un accès administrateur.


GEEKFLARE
Client
docker build
docker pull
docker run
DOCKER_HOST
docker daemon
Containers
Images
openstack.
Registry
NGINX
openstack.
build
pull
run

Configuration du démon Docker

La configuration du démon permet d'agir sur :

- les paramètres réseau,
- la gestion des logs,
- les options de sécurité,
- l'exposition de l'API.

Gestion des logs Docker

Les logs sont essentiels pour :

- le diagnostic,
- la supervision,
- la gestion des incidents.

Un mauvais paramétrage peut provoquer une saturation disque.

Exposition du démon Docker

Le démon Docker peut écouter :

- sur un socket Unix (par défaut),
- sur une interface TCP.

L'exposition réseau doit toujours être sécurisée.

TP 1 – Administration avancée du démon Docker

Objectifs pédagogiques :

- Identifier les paramètres critiques du Docker Engine
- Comprendre les impacts de configuration du démon
- Analyser les mécanismes de journalisation

Axes abordés :

- configuration du service Docker,
- gestion des logs,
- communication client / démon.

Problèmes de sécurité courants

Les risques Docker fréquemment observés :

- démon exposé sans authentification,
- conteneurs exécutés avec privilèges élevés,
- images non vérifiées,
- registres non protégés.

Principe du chiffrement TLS

TLS permet :

- le chiffrement des communications,
- l'authentification des entités,
- l'intégrité des échanges.

Docker et TLS

Docker peut être configuré pour :

- refuser les connexions non chiffrées,
- accepter uniquement des clients authentifiés.

Exemple réel d'incident

Un démon Docker exposé sans TLS permet à un attaquant de lancer des conteneurs malveillants et de compromettre l'hôte.

TP 2 – Sécurisation des communications Docker avec TLS

Objectifs pédagogiques :

- Identifier les risques liés à l'exposition du démon Docker
- Comprendre les mécanismes TLS
- Mettre en œuvre une communication sécurisée

Axes abordés :

- sécurisation de l'API Docker,
- gestion des certificats,
- contrôle des accès distants.

Limites d'un hôte Docker unique

Un seul hôte présente :

- un point de défaillance unique,
- des limites de charge,
- une faible résilience.

Réseaux Docker multi-hosts

Les réseaux multi-hosts permettent :

- la communication inter-conteneurs multi-machines,
- la construction d'architectures distribuées.

Cas d'usage professionnel

Une application répartie sur plusieurs nœuds reste disponible malgré la perte d'un serveur.

TP 3 – Environnement Docker multi-nœuds

Objectifs pédagogiques :

- Comprendre les enjeux du multi-host
- Identifier les mécanismes réseau Docker
- Appréhender la distribution des services

Axes abordés :

- réseaux overlay,
- communication inter-nœuds,
- résilience applicative.

Problématique des images Docker

Toute image peut contenir :

- des vulnérabilités,
- des backdoors,
- du code malveillant.

Principe du Docker Content Trust

Docker Content Trust repose sur :

- la signature cryptographique,
- la vérification d'intégrité,
- la confiance dans la chaîne de livraison.

Cas d'usage en entreprise

Certaines organisations imposent l'utilisation exclusive d'images signées en production.

TP 4 – Sécurisation des images avec Docker Content Trust

Objectifs pédagogiques :

- Identifier les risques liés aux images non vérifiées
- Comprendre les mécanismes de signature
- Appliquer une politique de confiance

Axes abordés :

- signature d'images,
- contrôle d'intégrité,
- validation avant déploiement.

Pourquoi une registry privée

Une registry privée permet :

- le stockage interne des images,
- le contrôle des accès,
- la sécurisation du cycle CI/CD.

Cycle de vie des images Docker

Les images suivent un cycle :

- construction,
- validation,
- stockage,
- déploiement.

Sécurisation d'une registry

Une registry doit être :

- authentifiée,
- chiffrée,
- surveillée.

TP 5 – Création et sécurisation d'une registry privée

Objectifs pédagogiques :

- Comprendre le rôle d'une registry privée
- Maîtriser la gestion des images
- Appliquer des contrôles d'accès

Axes abordés :

- registry privée,
- push / pull d'images,
- sécurisation des accès.

Docker Trusted Registry

Docker Trusted Registry apporte :

- une gouvernance des images,
- un contrôle fin des accès,
- un modèle RBAC.

RBAC et gouvernance

La gestion repose sur :

- organisations,
- équipes,
- rôles.

Cas d'usage

Séparation claire entre :

- équipes de développement,
- équipes de validation,
- équipes de production.

TP 6 – Gouvernance des images avec DTR

Objectifs pédagogiques :

- Comprendre le modèle RBAC
- Structurer les accès aux images
- Contrôler les opérations push / pull

Axes abordés :

- utilisateurs et équipes,
- permissions,
- gouvernance.

Pourquoi Docker Machine

Docker Machine permet :

- le provisionnement automatique,
- la gestion d'hôtes distants,
- l'intégration cloud.

Cas d'usage cloud

Création automatisée de machines Docker dans des environnements cloud.

TP 7 – Provisionnement automatisé avec Docker Machine

Objectifs pédagogiques :

- Comprendre l'automatisation Docker
- Identifier les usages de Docker Machine
- Gérer le cycle de vie des hôtes

Axes abordés :

- création de machines,
- gestion distante,
- intégration cloud.

Orchestration de conteneurs

L'orchestration permet :

- la gestion de services distribués,
- la montée en charge,
- la tolérance aux pannes.

Architecture Docker Swarm

Swarm repose sur :

- managers,
- workers,
- services.

Scheduling et résilience

Le scheduler Swarm répartit les services et assure leur redéploiement en cas de panne.

TP 8 – Orchestration avec Docker Swarm

Objectifs pédagogiques :

- Comprendre les mécanismes d'orchestration
- Identifier les rôles du cluster Swarm
- Appréhender la haute disponibilité

Axes abordés :

- cluster Swarm,
- déploiement de services,
- résilience.

Architecture micro-services

Les micro-services consistent à découper une application en services indépendants et faiblement couplés.

Apport de Docker

Docker facilite :

- l'isolation des services,
- la portabilité,
- la reproductibilité.

Exemple d'architecture

Frontend, backend et base de données déployés dans des conteneurs distincts.

Pourquoi Docker Compose

Docker Compose permet :

- la définition d'applications multi-conteneurs,
- la simplification des déploiements,
- la cohérence des environnements.

Compose et Swarm

Compose est adapté au développement et aux tests. Swarm est privilégié pour la production.

TP 9 – Micro-services avec Compose et Swarm

Objectifs pédagogiques :

- Comprendre une architecture micro-services complète
- Identifier les rôles de Compose et Swarm
- Appréhender le passage à l'échelle

Axes abordés :

- application multi-conteneurs,
- orchestration,
- scalabilité.

Synthèse

Ce module couvre l'ensemble des usages avancés de Docker :

- administration,
- sécurité,
- orchestration,
- architecture distribuée.

Conclusion finale

La maîtrise avancée de Docker constitue un socle essentiel des environnements DevOps modernes.