Mise en production d'applications avec conteneurisation

Modalités

- Travail individuel / Binôme en autonomie
- Durée : 8 heures (suggéré comme une journée complète ou 2 demi-journées).
- Prérequis : Connaissances basiques des systèmes d'exploitation basés sur Unix/Linux, notions de base en réseau
- Outils requis : Machine sous Linux (Ubuntu)

Objectifs de l'activité

Découvrir la méthode de mise en production d'un site web de manière conteneurisée et de comprendre les bénéfices de cette approche largement utilisée dans l'industrie actuelle.

Compétences

- Savoir créer un Dockerfile
- Savoir utiliser les lignes de commandes Docker
- Savoir utiliser Docker-compose
- Savoir utiliser une régistry Docker
- Savoir déployer un site de bout en bout (du build jusqu'à la mise en production)
- Pouvoir mettre en place Traefik pour du reverse proxy

Consignes

1 — La conteneurisation et Docker : c'est quoi ?

L'informatique, à l'instar de nombreux domaines technologiques, est caractérisée par une évolution constante et rapide. Au fil du temps, des outils et des méthodes ont été développés pour répondre à des problèmes spécifiques. Docker, avec sa proposition innovante de conteneurisation, est l'un de ces outils qui a radicalement transformé la manière dont nous développons, déployons et exécutons les logiciels.

L'évolution du déploiement et de la mise en production :

1. Déploiement manuel sur serveur physique :

Avant l'automatisation, tout était manuel. Les administrateurs système devaient installer chaque composant individuellement, ce qui posait d'énormes problèmes de cohérence et de reproductibilité.

2. Machines virtuelles (VM) :

La virtualisation a permis d'isoler les applications, mais chaque VM était lourde, avec son propre système d'exploitation, et prenait du temps à démarrer et à arrêter.
Ces méthodes traditionnelles ont jeté les bases pour comprendre les défis que Docker visait à résoudre.

Docker et la naissance de la conteneurisation moderne :

Docker a été introduit en 2013 par Solomon Hykes. Si la technologie de conteneurisation n'était pas nouvelle, Docker l'a rendue pratique, efficace et populaire. Les conteneurs offrent :

- Isolation
- Portabilité
- Efficacité
- Scalabilité

En somme, Docker et la conteneurisation ont été la réponse à une série de défis persistants dans le monde de l'informatique. En combinant la reproductibilité, la portabilité et l'efficacité, ils ont offert une solution élégante aux problèmes que les méthodes traditionnelles peinaient à résoudre. La popularité croissante de Docker témoigne de son impact profond et de son rôle central dans le paysage technologique moderne.

Livrables

➔ Faites une recherche sur les différents avantages qu'apporte la conteneurisation et essayé d'expliquer les grands principes avec vos mots

2 — Docker : les concepts de base

Docker repose sur plusieurs concepts clés qui permettent de comprendre son fonctionnement et son utilité. Voici les concepts de base associés à Docker :

- **Image** : Une image Docker est un modèle immuable utilisé pour créer un conteneur. Elle contient le code source de l'application, les bibliothèques, les dépendances et autres fichiers nécessaires à l'exécution de l'application.
- **Dockerfile** : C'est un fichier de script qui contient les instructions pour construire une image Docker.
- **Conteneur** : Un conteneur est une instance exécutable d'une image Docker. Il s'agit d'une encapsulation légère d'une application et de son environnement d'exécution, fonctionnant de manière isolée sur le système hôte.
- **Registry** : Zone de stockage pour les images Docker. Il peut être public ou privé (pour une utilisation interne en entreprise par exemple). Docker Hub est un service cloud public pour partager et stocker des images Docker. C'est comme un "GitHub" pour les images Docker, où les développeurs peuvent push ou pull des images.
- **Volume** : Les volumes sont utilisés pour persister les données et partager des fichiers entre le conteneur et l'hôte. Ils sont essentiels pour éviter la perte de données lorsque les conteneurs sont arrêtés ou supprimés.
- **Réseau Docker** : Docker possède sa propre gestion du réseau, permettant aux conteneurs de communiquer entre eux et avec des ressources extérieures. Il offre plusieurs modes réseau comme "bridge", "host" et "overlay"
- **Docker Compose** : C'est un outil pour définir et gérer des applications multi-conteneurs. Avec Docker Compose, on peut définir une application à l'aide de plusieurs conteneurs dans un seul fichier, puis démarrer ces conteneurs simultanément avec une seule commande. Par exemple, vous voulez déployer une application PHP qui utilise une base de données MySQL. Vous allez écrire un fichier docker-compose qui va décrire la configuration du conteneur pour PHP et le conteneur MySQL. Cela va éviter de lancer les commandes à la main et permettre d'avoir un suivi dans Git.

Maintenant que vous avez les différents concepts théoriques, passons un peu à la pratique. Dans cette partie, nous allons exécuter des commandes de bases qui vont vous permettre d'appréhender de manière pratique les différents éléments vus plus haut. On va rester dans le contexte de votre machine local pour des questions de praticité.

Dans un premier temps, assurez vous que vous avez Docker d'installé sur votre machine (comme pour toutes les commandes linux, faites un petit `docker --version`).

Lancer votre premier container

Vous allez devoir lancer un conteneur utilisant l'image nginx avec le tag latest qui se trouve sur Docker Hub (je suis sympa, je vous donne le lien vers la registry https://hub.docker.com/_/nginx).
Le conteneur doit se lancer sur le port 8080 de votre machine local.

Hint : Utilisez la bonne ligne de commande pour pouvoir lister les images présentes sur votre machine. Faite cela avant et après avoir lancer ce conteneur. Que remarquez-vous ?

Livrables

- ➔ Pouvoir accéder à nginx à l'url http://localhost:8080. Vous devez voir apparaître la phrase **Welcome to nginx!**
- ➔ Quelle est la différence entre une image Docker et un conteneur ?
- ➔ Que fait Docker si l'image demandée n'est pas présente localement ?

Découvrez les volumes persistant et les applications stateful

Dans cette partie, nous allons découvrir le principe de volume persistant et découvrir en quoi il est utile dans un contexte de production. Pour nous appuyer sur un exemple

concret, nous allons utiliser une image Mariadb (tag latest) pour faire nos manipulations.

Dans un premier temps, lancez un conteneur avec l'image Mariadb. Votre conteneur devra écouter sur le port 3306 et vous définirez les credentials (user/pwd) nécessaires pour la connexion. Une fois cela fait, vous allez devoir vous connecter avec un outil de gestion de BDD (comme vu dans le module de php / data viz). Vous allez créer une base de données, avec une table et injecter des données fictives de votre choix. Par exemple, vous pouvez choisir :

- Nom de la base de données : mydatabase
- Nom d'une table : user
- Champs de la table : nom, prénom, age.

Ne perdez pas votre temps sur la création de cette base de données (et le type de champs), elle n'est là qu'à titre de support pour notre cours.
Un fois cela fait, stopper et supprimer votre conteneur. Puis essayer de recréer le conteneur. **Est ce que vos données sont encore présentes ?**

Maintenant, vous allez essayer de refaire les premières étapes en ajoutant un volume persistant à votre conteneur. Pour rappel, les étapes sont :

- Lancement de mon conteneur avec l'image MariaDB (avec le volume persistant cette fois)
- Création BDD + Table + injection de données
- Stop et suppression du conteneur
- Relancer le conteneur

Est ce que vos données sont encore présentes ?

Livrables

➔ Avoir une base de données qui tourne sur le port 3306 avec des données persistante
➔ Comprendre le principe de volume docker
➔ Quelle est la différence entre une application statefull et stateless

Lancer une application multi conteneur

Dans cette partie, nous allons voir comment déployer une application multi conteneur. Hein ? un conteneur c'est pas suffisant, pourquoi il faut que ça soit multi-conteneur ? Et bien parce que, dans le cas où vous avez une application web qui utilise une base de données, vous allez avoir un conteneur pour l'application web et un conteneur pour la base de données. D'où le terme multi-conteneur.

Pour faciliter la gestion de ce type d'application, on peut utiliser docker-compose. Avec Docker Compose, vous pouvez utiliser un fichier `docker-compose.yml` pour configurer les services de votre application. Plutôt que de gérer chaque conteneur individuellement, Docker Compose vous permet de gérer et d'interconnecter plusieurs conteneurs comme s'ils étaient une seule application.

Voici quelques points clés concernant Docker Compose :

- Fichier `docker-compose.yml` : Le cœur de Docker Compose est le fichier YAML qui décrit la structure et les paramètres des services, des réseaux et des volumes.
- Services : Dans Docker Compose, chaque conteneur est décrit comme un "service". Chaque service est une configuration pour un conteneur particulier et son image.
- Commande `up` : Avec la commande docker-compose up, vous pouvez démarrer tous les services définis dans le fichier docker-compose.yml.
- Intégration avec Docker : Docker Compose travaille avec Docker, ce qui signifie que toutes les commandes et fonctionnalités de Docker sont également accessibles.

Pour nous exercer, nous allons donc déployer une application wordpress. Cette application est composée :

- 1 service Wordpress qui utilise l'image latest
- 1 service MariaDB qui utilise l'image mariadb:10.6.4-focal
- Attention à bien paramétrer les variables d'environnement des deux services
- La base de données ne doit pas être accessible depuis l'extérieur. Autrement dit, il faudra regarder du côté des réseaux Docker.
- La base de données doit pouvoir persister les données

Livrables

➔ Avoir une application fonctionnelle sur l'url http://localhost:80
➔ Comment configurez-vous les variables d'environnement ?
➔ Comment l'application wordpress communique-t-elle avec le conteneur MariaDB ?

Ecrivez votre première image

Dans cette partie, nous allons voir comment créer une image Docker Custom pour un projet qui ne possède pas encore d'image Docker. Pour cela, cloner le repo Github suivant : https://github.com/WilliamBurillon/devops-training-nodejs

A l'aide des informations du Readme qui vous explique comment lancer le projet sans Docker, vous êtes en mesure de construire une image docker et de lancer un conteneur.
Vous devez donc :

- Ecrire un fichier Dockerfile pour la création de l'image
- Ecrire un fichier docker-compose qui va vous permettre de build et de lancer le conteneur

Livrables

➔ Avoir une application fonctionnelle sur le port de votre choix

3 — L'entreprise TechFlow adopte Docker !

Maintenant que vous avez survolé les différentes fonctionnalités, nous allons revenir à notre cas d'entreprise. TechFlow a désormais compris les limites d'un déploiement "traditionnel" et pense désormais à l'utilisation de Docker pour ces applications en production. Et ce que vous allez faire dans cette partie.

Premièrement, rappelez vous des étapes nécessaires pour l'ancienne mise en production :

- Récupérer le code source
- Installation des packages php, php-fpm
- Installation de Nginx
- Configuration de Nginx pour chaque application (port, socket php-fpm avec la bonne version…)

Maintenant, pour chaque application, vous allez devoir :

- Conteneurisé l'application php en utilisant une image incluant php-fpm (avec la bonne version) et nginx
- Tester de lancer votre conteneur en local sur votre machine
- Si l'application fonctionne sur votre machine, vous pouvez push l'image sur le registry docker (https://registry.williamburillon.com)
- Deployer votre application en utilisant docker compose sur le serveur

Livrables

➔ Avoir le site Web php-7 qui fonctionne sur le port 80 de votre machine
➔ Avoir le site Web php-8 qui fonctionne sur le port 81 de votre machine

4 — Mettez en place un reverse proxy avec Traefik

Traefik, c’est quoi ?

Le monde du déploiement d'applications web modernes est complexe et en constante évolution. L'architecture des applications s'est déplacée des monolithes vers des microservices, et avec cette évolution, la nécessité d'outils de gestion plus sophistiqués s'est accrue. C'est ici qu'interviennent les concepts de Reverse Proxy et d'outils comme Traefik.

Un reverse proxy est un serveur qui se trouve entre les clients et les serveurs web. Il reçoit toutes les requêtes client et décide à quel serveur back-end les transmettre. Les avantages sont :

- Équilibrage de charge: Distribue les requêtes entrantes entre plusieurs serveurs.
- Cache: Stocke du contenu souvent demandé pour réduire la charge sur les serveurs.
- Sécurité: Masque la structure et les détails du réseau interne.
- Compression: Réduit la taille des réponses pour accélérer les temps de chargement.
- SSL/TLS Termination: Décharge les serveurs back-end du coût de la gestion du trafic crypté.

Traefik est un reverse proxy moderne conçu pour les architectures basées sur des conteneurs comme Docker, Kubernetes, etc. Il est spécialement conçu pour s'adapter à des environnements dynamiques.
Les caractéristiques principales de Traefik sont :

- Simplicité de Configuration: Pas besoin de redémarrer le service pour les nouvelles configurations.
- Intégration avec Docker: Découvre automatiquement les nouveaux services ou conteneurs Docker.
- Dashboard Intégré: Offre un tableau de bord pour le monitoring en temps réel.
- Routage Dynamique: Ajuste le routage sans redémarrer.
- Automatisation SSL: Intégration native avec Let's Encrypt pour générer des certificats SSL.
- Haute Disponibilité: Support pour différents algorithmes d'équilibrage de charge.

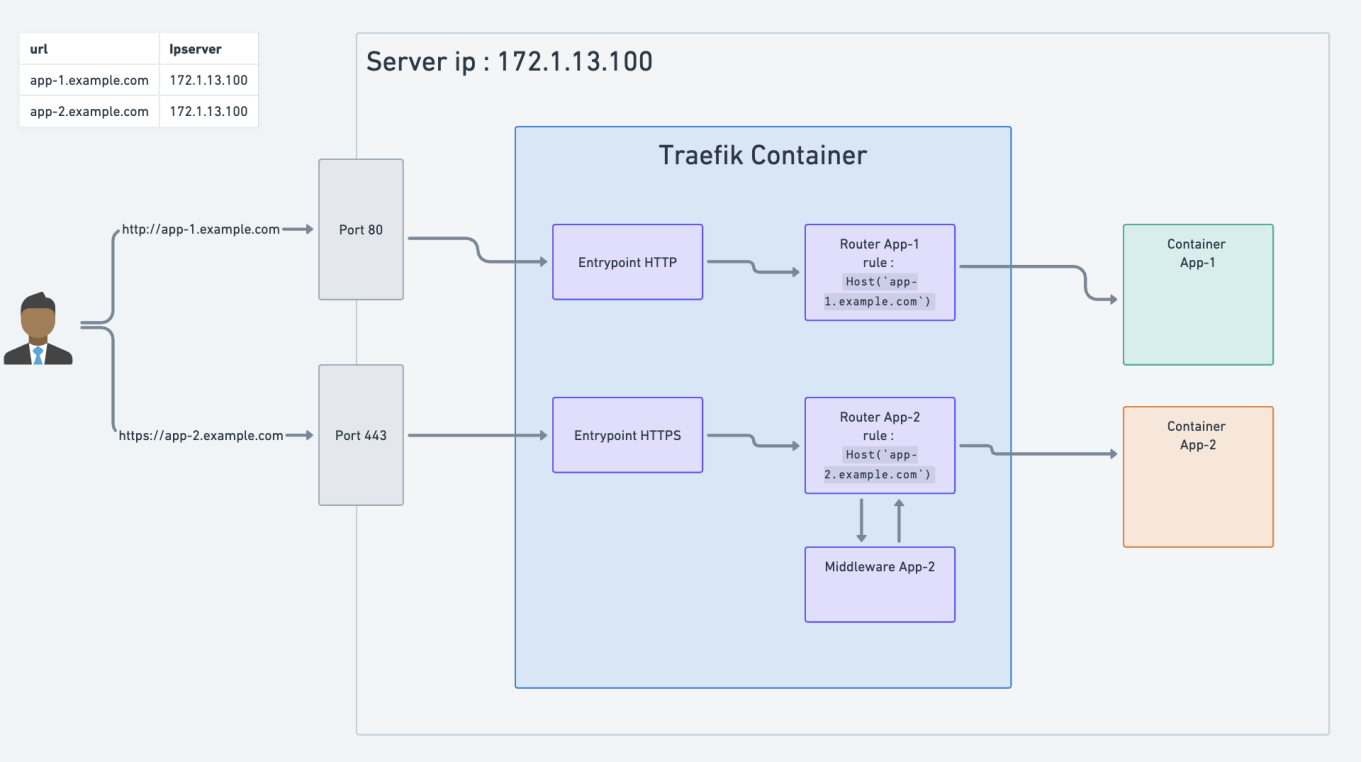
Le reverse proxy est un composant clé dans l'architecture des applications web modernes. Traefik se démarque comme une solution moderne qui excelle dans des environnements dynamiques et conteneurisés. Avec sa facilité de configuration et son tableau de bord intégré, il offre une alternative puissante et flexible aux options traditionnelles.

Comment ça marche concrètement ?

Traefik est composer des différents composants suivant :

- Entrypoint : Les "Entrypoints" sont les points d'entrée du réseau vers Traefik. Ils définissent sur quelles adresses et ports le reverse proxy doit écouter pour les connexions entrantes.
  Importance :
  - Permet d'établir les ports sur lesquels Traefik écoute.
  - Vous pouvez définir plusieurs entrypoints, par exemple, un pour HTTP et un autre pour HTTPS.
- Router : Les "Routers" prennent les décisions de routage. Ils reçoivent les requêtes des entrypoints et les acheminent vers les services internes en fonction des règles de routage configurées.
  Importance :
  - C'est dans ce composant qu'on définit les règles de routage (si l'entrypoint reçoit une requête avec le host "www.test.com" , alors c'est toi qui prend la requête et qui va l'acheminer vers le service associé.
  - Permettent l'application de middleware pour des fonctionnalités comme la redirection, le taux de requêtes, etc.
- Service : Les "Services" sont les points de terminaison auxquels les routeurs de Traefik acheminent le trafic. Un service peut être un conteneur Docker, un pod Kubernetes, une machine virtuelle, etc.
  Importance :
  - Fournissent des algorithmes d'équilibrage de charge pour distribuer efficacement le trafic.
  - Sont découverts dynamiquement dans des environnements comme Docker et Kubernetes.
- Middleware : Les "Middlewares" sont des composants qui peuvent manipuler les requêtes et/ou les réponses pour ajouter des fonctionnalités comme la redirection, la limitation du taux, l'authentification, etc.
  Importance :
  - Ajoutent une couche de fonctionnalités supplémentaires entre le routeur et le service.
  - Sont extrêmement flexibles et peuvent être chaînés pour des fonctionnalités complexes.

Voici un schéma pour comprendre les principes théoriques et comment on les applique à un cas concret.



Nous avons dans un premier temps paramétré notre serveur DNS pour que les adresses IP pointe vers notre serveur.
Nous avons paramétré 2 entrypoints :

- HTTP : qui écoute sur le port 80
- HTTPS : qui écoute sur le port 443

Nous avons paramétré 2 router :

- Router App-1 : il est link à l'entrypoint HTTP et nous avons défini une rule de type Host avec l'url de l'app-1. Ce routeur est associé au conteneur qui héberge l'App-1.
- Router App-2 : il est link à l'entrypoint HTTPS et nous avons défini une rule de type Host avec l'url de l'app-2. Ce routeur est associé au conteneur qui héberge l'App-2.

Livrables

➔ Avec les explications fournies par votre formateur, expliquez ce qu'il se passe quand l'utilisateur fait une requête à l'url `https://app-2.example.com`

Passons à l'action !

Dans un premier temps, on vous a acheté à chacun un nom de domaine pour cet exercice. Connectez-vous à Cloudflare avec les identifiants que votre formateur vous a fournis. Il faudra modifier le serveur DNS pour que votre nom de domaine et tous les sous-domaines pointent vers l'ip de votre serveur.

Une fois cela fait, on va configurer Traefik maintenant que vous avez compris comment il marche sur le principe (enfin j'espère).

Deux étapes sont nécessaires :

- Installation de Traefik via Docker. On pourra utiliser un fichier docker-compose pour lancer le conteneur car il nécessite de la configuration.
- Modification des fichiers docker-compose des différents projets pour créer les routeurs et associer les services Traefik au conteneur.

Je vais volontairement ne pas vous donner plus d'informations concernant la mise en place de Traefik. Mais je suis sympa, je vous donne l'url de la documentation https://doc.traefik.io/traefik/. Suivant où vous en êtes, je vous donnerai les éléments pour que vous puissiez avancer. Have fun !

Livrables

- ➔ Avoir Traefik fonctionnel
- ➔ Avoir le site Web php-7 qui fonctionne sur l'url http://php7.<votrenomdedomaine>
- ➔ Avoir le site Web php-8 qui fonctionne sur l'url http://php8.<votrenomdedomaine>