Sysadmin/DevOps Challenge

Void Agency

Yahya Chahine

Livré le : 01.05.2024

Sommaire

Présuppositions

Choix du Système d'exploitation du serveur

Il n'est mentionné nulle part dans les instructions de l'exercice ni sur le README de la GitHub repository quel système d'exploitation est censé être utilise pour le serveur.

Je me suis naturellement tourné vers votre site web void.fr plus spécifiquement la page Jobs pour le poste d'Ingénieur DevOps dessus sont mentionnées les distribution Linux Red Hat et Debian.

Fonctionnalité	Debian	Red Hat Enterprise Linux (RHEL)
Stabilité	• Extrêmement stable • Idéal pour la production	• Enterprise-grade • Extrêmement stable
Communauté et Support	• Open Source • Grande Communauté	• Pas entièrement Open Source • Support Payant
Package Manager	• apt	• yum (peut être plus complexe)
Disponibilité des Logiciels et Outils	• Versions Stable • Potentiellement pas les dernières versions	• Versions Stable • Les dernières versions pour certains Outils et Logiciels

La distribution open source CentOS, « calquée » sur Red Hat Entreprise Linux (RHEL) atteindra sa EOL (End Of Life) le 30 Juin 2024 et ne recevra plus de mise à jour de sécurité et autre, les utilisateurs CentOS devront donc migre vers d'autres distribution je ne l'ai donc pour cela pas inclus dans le comparatif

En conclusion, Debian et RHEL offrent tous deux des fondations solides pour déployer et faire fonctionner le stack de l'exercice (Drupal, React, NGINX, Docker, Ansible). Le choix de l'OS pourrait être diffèrent selon les projets et leurs besoins

J'ai donc opté pour la solution Open Source, et disposant de la plus grosse communauté et la plus grande quantité de ressources en ligne à savoir **Debian**.

Machine Virtuelle

Le but de l'exercice étant de mesurer les compétences en configuration et résolutions de problèmes, compte tenu des différences entre les différents OS hôtes (j'utilise MacOS), j'ai jugé plus prudent, efficace, et économique en temps et en effort d'utiliser **HashiCorp Vagrant Community** une solution gratuite et open source permettant de mettre en place des environnements de développement portables et consistants en utilisant un fichier de configuration (Vagrantfile).

Connaissance en Drupal

Dans le cadre de cette tâche technique, j'ai été confronté au défi de configurer un projet `React/Drupal/MySQL/NGINX`, un environnement comprenant plusieurs composants avec lesquels j'avais une expérience limitée, notamment Drupal. Malgré ce manque de connaissance, j'ai abordé la tâche avec une attitude proactive, en utilisant des ressources d'auto-formation et de documentation pour rapidement comprendre les principes essentiels de l'architecture, de la configuration et de l'intégration de Drupal au sein de l'écosystème du projet.

Git

Afin préserver l'intégrité de la repository Git d'origine, j'ai créé un fork du dépôt et c'est cette version qui est utilisée dans la machine virtuelle pour résoudre les défis proposés. Cette approche permet de travailler sur une copie de la repository tout en préservant l'état original du dépôt principal.

Configuration et Provisionnement de la VM

L'utilisation de Vagrant pour déployer une machine virtuelle a considérablement simplifié le processus d'installation et de configuration par rapport à une installation manuelle sur VirtualBox. Grâce à Vagrant, j'ai pu automatiser la création de la machine virtuelle en définissant simplement sa configuration dans un fichier `Vagrantfile`. Ce fichier contient toutes les spécifications de la machine virtuelle, y compris le système d'exploitation, la mémoire, les disques, les réseaux et les logiciels à installer. En exécutant une simple commande `vagrant up`, Vagrant a automatiquement provisionné et configuré la machine virtuelle selon les paramètres spécifiés, ce qui a considérablement réduit le temps nécessaire pour déployer l'environnement. Comparé à une installation manuelle sur VirtualBox, qui aurait nécessité une configuration étape par étape, l'utilisation de Vagrant a permis un gain de temps significatif et a simplifié le processus d'installation, tout en assurant une cohérence et une reproductibilité accrues de l'environnement de développement.

Ci-dessous une brève explication du contenu du Vagrantfile:

- `Vagrant.configure("2") do |config|`: Cette ligne démarre la configuration de Vagrant et spécifie la version de configuration (ici "2"). La configuration principale est définie à l'intérieur du bloc do lconfigl.

- `config.vm.box = "debian/bookworm64"`: Cette ligne définit la box Vagrant à utiliser pour créer la machine virtuelle. Ici, la box est "debian/bookworm64", ce qui signifie qu'elle utilise une version 64 bits de Debian nommée "bookworm".

- `config.vm.network "private_network", ip: "10.0.2.15"`: Cette ligne configure un réseau privé pour la machine virtuelle avec l'adresse IP spécifiée. Dans cet exemple, l'adresse IP est "10.0.2.15".

- `config.vm.provision "ansible", playbook: "playbook.yml"`: Cette ligne configure la provision de la machine virtuelle avec Ansible en utilisant le playbook spécifié ("playbook.yml"). Cela signifie que le playbook Ansible sera exécuté pour effectuer la configuration de la machine virtuelle.

- `config.vm.provider "virtualbox" do |vb|`: Cette ligne configure l'hyperviseur de la machine virtuelle, qui dans ce cas est VirtualBox. Les options spécifiques à VirtualBox sont définies à l'intérieur du bloc do lvbl.

- `vb.memory = "1024"`: Cette ligne définit la quantité de mémoire RAM attribuée à la machine virtuelle. Dans cet exemple, la machine virtuelle disposera de 1024 Mo (1 Go) de mémoire RAM.

Maintenant, après avoir examiné la configuration de la machine virtuelle à l'aide de Vagrant, nous nous pencherons sur le contenu du playbook Ansible qui sera utilisé pour provisionner cette machine. Il est important de noter que Vagrant et Ansible sont des outils qui s'intègrent parfaitement, permettant une automatisation transparente de la configuration et du déploiement de machines virtuelles. Vagrant fournit une infrastructure pour créer et gérer les machines virtuelles, tandis qu'Ansible prend en charge la configuration et le déploiement des logiciels sur ces machines. En combinant ces deux outils, nous pouvons automatiser efficacement l'ensemble du processus de développement et de déploiement d'une infrastructure logicielle.

Ci-dessous une brève explication du contenu du `playbook.yml` :

Ce playbook Ansible est essentiel pour la provision d'une machine virtuelle (VM) qui servira d'environnement pour exécuter des conteneurs Docker. Son objectif est de garantir un déploiement fluide de Docker et de configurer des paramètres essentiels pour son fonctionnement optimal.

Pour commencer, le playbook commence par garantir la propreté et la mise à jour du système en supprimant les paquets inutiles et en actualisant les listes de paquets disponibles. Ensuite, il procède à l'installation des paquets prérequis pour Docker, incluant les certificats, curl et les outils de gestion des conteneurs.

Une étape cruciale est la mise en place d'un environnement apt approprié pour Docker. Cela comprend la création d'un répertoire dédié pour les clés GPG utilisées pour authentifier les paquets Docker. Ensuite, il télécharge la clé GPG nécessaire depuis l'URL spécifiée et la place dans le répertoire approprié. Une fois la clé téléchargée, le playbook configure le référentiel Docker dans les sources apt du système en fonction de la distribution et de la version Debian spécifiées.

Il aurait été plus facile de simplement installer les paquets `docker-io`, `docker-compose` cependant ce sont des paquets non-officiels il est donc préférable de suivre les directives de la documentation officielle de Docker et d'installer les paquets officiels.

Après l'ajout du référentiel Docker, le playbook met à jour les listes de paquets disponibles pour inclure les nouveaux paquets Docker. Ensuite, il procède à l'installation des composants Docker, notamment `docker-ce`, `docker-ce-cli`, `containerd.io`, `docker-buildx-plugin`, `docker-compose-plugin` et `cron` pour la gestion des tâches planifiées.

Une fois Docker installé, le playbook vérifie si le fichier de configuration daemon.json existe déjà. S'il n'existe pas, il le crée et configure les options de journalisation Docker, notamment le pilote de journalisation et la limite de taille du journal.

Enfin, pour assurer la maintenance régulière du système Docker, le playbook configure une tâche Cron pour nettoyer périodiquement les conteneurs Docker non utilisés. Cela garantit que la propreté et l'optimisation de l'environnement Docker pour les performances à long terme.

Configuration et installation d'une application Drupal PHP avec son infrastructure Docker

Au chapitre 2, notre attention se tourne vers la configuration et l'installation d'une application Drupal ainsi qu'une application React, accompagnées du déploiement de Nginx et MySQL. Alors que Nginx et MySQL sont principalement destinés à soutenir Drupal, la configuration complète englobe l'intégration de ces éléments pour créer un environnement robuste. À travers des processus de configuration et d'installation méticuleux décrits dans ce chapitre, nous visons à établir un écosystème cohérent qui accueille harmonieusement à la fois Drupal et React, chacun jouant un rôle essentiel dans leurs domaines respectifs. En combinant ces composants de manière harmonieuse,

nous aspirons à poser des bases solides propices au développement efficace et au fonctionnement sans heurts de nos applications web.

Drupal

La configuration de drupal est composée de 3 parties en excluant le fichier docker-`compose.yml` qui sera discuter plus en détails plus tard, ladite configuration est composée d'un Dockerfile permettant de gérer l'infrastructure Docker et de 2 fichiers .ini permettant de configurer `opcache` et `php`.

Le Dockerfile pour Drupal est conçu pour fournir un environnement PHP optimisé et robuste pour exécuter des applications Drupal. Il commence par utiliser l'image php:8.3-fpm-alpine comme base, ce qui offre une distribution légère d'Alpine Linux avec PHP 8.3 préinstallé. Ce choix de base d'image est stratégique car Alpine Linux est connu pour sa légèreté et sa sécurité, ce qui permet de réduire la taille globale de l'image et de minimiser les vulnérabilités potentielles.

Ensuite, le Dockerfile installe diverses dépendances nécessaires à PHP et à Drupal. Par exemple, les extensions GD, MySQL, PostgreSQL, Zip, et d'autres sont installées à l'aide des commandes `apk add` et `docker-php-ext-install`. Ces extensions sont cruciales pour le bon fonctionnement de Drupal, notamment pour la gestion des images, la connexion à la base de données et le traitement des fichiers compressés.

La configuration d'`opcache` est également effectuée pour améliorer les performances en stockant en mémoire les scripts PHP précompilés, réduisant ainsi le temps de chargement des pages Drupal. De plus, l'installation de Redis en tant que mécanisme de mise en cache est encouragée pour accélérer davantage les performances et réduire la charge sur la base de données.

Le fichier ini fournit des directives de configuration PHP spécifiques pour optimiser la sécurité et les performances de l'environnement PHP. Par exemple, la désactivation de `expose_php` masque la version de PHP utilisée, réduisant ainsi les risques de sécurité liés à l'identification de la version PHP. De même, l'activation d'`opcache` et de ses paramètres de mémoire aide à accélérer l'exécution des scripts PHP en les stockant en mémoire.

React

La configuration de l'application React.js est plutôt simple elle est composée d'un fichier Dockerfile.

Ce Dockerfile vise à fournir un environnement de développement et de déploiement efficace pour exécuter des applications React. Il commence par utiliser l'image officielle de Node.js comme base, en particulier l'image node:current-alpine, qui est une version légère et actuelle d'Alpine Linux avec Node.js préinstallé. Ce choix de base

d'image est judicieux car Alpine Linux offre une distribution légère, ce qui permet de réduire la taille de l'image finale et d'améliorer les performances globales.

Ensuite, le Dockerfile définit le répertoire de travail (WORKDIR) dans le conteneur dans le dossier /app, où tout le code de l'application sera copié et exécuté. Les fichiers package.json et package-lock.json sont copiés dans ce répertoire de travail pour installer les dépendances de l'application à l'aide de la commande npm install. Cette étape est cruciale car elle garantit que toutes les dépendances requises sont correctement installées dans le conteneur.

Une fois les dépendances installées, le reste du code de l'application est copié dans le répertoire de travail à l'aide de la commande COPY . ., ce qui inclut les fichiers source, les composants React et les fichiers statiques nécessaires à l'application.

Ensuite, l'application React est compilée pour la production à l'aide de la commande npm run build, qui génère une version optimisée de l'application prête à être déployée. Cette étape est essentielle pour optimiser les performances de l'application et réduire la taille des fichiers à transférer sur le réseau.

Enfin, le conteneur expose le port 80 pour permettre l'accès extérieur à l'application React à l'aide de la directive `EXPOSE 80`, et spécifie la commande à exécuter lorsque le conteneur démarre à l'aide de `CMD ["npx", "serve", "-s", "-l", "80", "build"]`. Cette commande utilise le package serve pour servir les fichiers statiques de l'application React sur le port 80, permettant ainsi aux utilisateurs d'accéder à l'application via un navigateur web.

Nginx et MySQL

Comme indique précédemment le fichier docker-compose.yml et la pièce maitresse reliant les différents composant de la stack et sera donc discute en dernier, mais avant cela il parait judicieux de donner des explications détaillées quant à la configuration de Nginx et de MySQL

Le fichier de configuration Nginx définit la manière dont le serveur Nginx doit traiter les requêtes HTTP entrantes vers l'application Drupal. Voici une explication détaillée de chaque directive et section :

- `listen`: Cette directive spécifie les ports sur lesquels le serveur Nginx écoute les connexions entrantes. Dans ce cas, le serveur écoute les connexions sur le port 80, à la fois pour IPv4 et IPv6.

- `index`: Cette directive spécifie les fichiers à utiliser comme index par défaut lorsqu'un répertoire est demandé. Dans cet exemple, les fichiers index seront recherchés dans l'ordre suivant : index.php, index.html, index.htm.

- `root`: Cette directive définit le répertoire racine à partir duquel Nginx servira les fichiers pour cette configuration. Ici, il est défini sur /var/www/html/web.

- `location /`: Cette section définit la manière dont Nginx traite les requêtes pour les URL qui ne correspondent à aucun autre bloc de configuration. Il tente de résoudre les URL demandées en essayant les fichiers ou répertoires existants, et renvoie à /index.php si aucun fichier correspondant n'est trouvé.

- `location ~ .php$`: Cette section définit la manière dont Nginx traite les requêtes pour les fichiers PHP. Il renvoie les requêtes à un serveur FastCGI sur drupal:9000 (généralement un serveur PHP-FPM) pour les traiter. Il transmet également des paramètres FastCGI comme le nom du script PHP.

- `*location ~ .(?:css|gif|ico|jpeg|jpg|js|png)$**`: Cette section définit la manière dont Nginx traite les requêtes pour les fichiers statiques tels que les images, les fichiers CSS et les fichiers JavaScript. Il définit une expiration maximale pour ces fichiers et désactive l'enregistrement des erreurs pour les fichiers non trouvés.

- `location ~ /.ht`: Cette section interdit l'accès à tout fichier dont le nom commence par .ht. Cela empêche l'accès aux fichiers de configuration sensibles d'Apache qui pourraient être exposés par erreur.

- `location = /favicon.ico`: Cette section désactive l'enregistrement des erreurs et des accès pour le fichier favicon.ico, souvent utilisé comme icône de site dans les navigateurs web.

- `location = /robots.txt`: Cette section désactive également l'enregistrement des erreurs et des accès pour le fichier robots.txt, qui est utilisé par les moteurs de recherche pour découvrir les pages à indexer sur le site. Il permet l'accès à tous les utilisateurs.

Quant au fichier de configuration de MySQL, il comporte quelques directives importantes pour la configuration du serveur MySQL. Voici une explication détaillée :

- `!includedir /etc/mysql/conf.d/`: Cette directive inclut tous les fichiers de configuration présents dans le répertoire /etc/mysql/conf.d/. Cela permet d'organiser la configuration en plusieurs fichiers, ce qui peut faciliter la gestion et la personnalisation du serveur MySQL.

- `!includedir /etc/mysql/mysql.conf.d/`: De manière similaire à la directive précédente, cette directive inclut tous les fichiers de configuration présents dans le répertoire /etc/mysql/mysql.conf.d/. Encore une fois, cela permet d'organiser la configuration en plusieurs fichiers.

- `[mysqld]`: Cette section spécifie les paramètres de configuration spécifiques au serveur MySQL (daemon mysqld). Toutes les directives suivantes dans ce fichier qui sont situées sous [mysqld] s'appliquent au serveur MySQL.

- `max_connections=50`: Cette directive définit le nombre maximum de connexions simultanées autorisées au serveur MySQL. Dans cet exemple, le nombre maximum de connexions est fixé à 50, ce qui signifie que le serveur MySQL peut gérer jusqu'à 50 connexions simultanées. Cette valeur peut être ajustée en fonction des besoins spécifiques du serveur et des ressources disponibles.

Docker Compose (docker-compose.yml)

Le fichier docker-compose.yml est un fichier de configuration utilisé par Docker Compose pour définir et gérer plusieurs conteneurs Docker qui composent une application. Il permet de spécifier les services, les réseaux, les volumes, les variables d'environnement et d'autres options nécessaires à la construction et à l'exécution d'une application distribuée.

La principale importance d'un fichier docker-compose.yml réside dans sa capacité à orchestrer facilement et de manière reproductible des environnements de développement, de test et de production complexes. En fournissant une description claire et détaillée des services nécessaires à une application, ainsi que de leurs dépendances et configurations, un fichier docker-compose.yml facilite la mise en place rapide et cohérente d'environnements de développement pour les équipes de développement logiciel.

L'utilisation de Docker Compose simplifie également le processus de gestion des dépendances et des configurations entre les services, en permettant aux développeurs de spécifier toutes les interactions nécessaires entre les différents composants de l'application dans un seul fichier. Cela garantit une cohérence et une reproductibilité élevées lors du déploiement de l'application sur différentes plates-formes et environnements.

En ce qui nous concerne, le fichier docker-compose.yml pour notre stack contient la configuration pour plusieurs services Docker. Voici une explication détaillée :

- `version: "3"`: Indique la version de la spécification Docker Compose utilisée dans ce fichier.

- `services`: Cette section définit les différents services Docker à créer.

- `mysql`: Définit un service MySQL en utilisant l'image MySQL version 8.3. Il configure les paramètres de connexion et les variables d'environnement pour définir le mot de passe root, la base de données, l'utilisateur et le mot de passe MySQL. De plus, il spécifie les fichiers de journalisation pour les erreurs, les journaux généraux et les requêtes lentes, ainsi que le nombre maximal de connexions autorisées. Les volumes sont montés pour le stockage persistant des données de la base de données et pour remplacer la configuration par défaut avec un fichier my.cnf personnalisé.

- `drupal`: Construit un service Drupal en utilisant un Dockerfile situé dans le répertoire ./drupal. Montre également le répertoire local ./drupal dans le conteneur pour la

personnalisation de l'application Drupal. Il dépend du service MySQL pour fonctionner correctement.

- `webserver`: Définit un service Nginx utilisant l'image Nginx version 1.25.5. Ce service est responsable du serveur web qui sert l'application Drupal. Les volumes sont montés pour personnaliser la configuration Nginx à l'aide d'un fichier nginx.conf.

- `react-app`: Construit un service pour une application React en utilisant un Dockerfile situé dans le répertoire ./react-app/. Il monte également le répertoire local ./react-app dans le conteneur et expose le port 5001 pour permettre l'accès à l'application React depuis l'extérieur du conteneur. Il configure également la variable d'environnement `API_URL` pour pointer vers l'API fournie par le service Nginx.

- `networks`: Cette section définit les réseaux utilisés par les services. Les réseaux `internal` et `external` sont définis comme des ponts pour permettre la communication entre les services Docker et avec l'extérieur.

- `volumes`: Cette section définit les volumes Docker utilisés pour stocker de manière persistante les données de la base de données Drupal et pour permettre la personnalisation de la configuration MySQL via un fichier `my.cnf`.

Conclusion

Je tiens à exprimer ma sincère gratitude à Void pour l'opportunité de passer l'entretien pour le poste d'Ingénieur DevOps. Participer à ce processus d'entretien a été une expérience d'apprentissage inestimable pour moi. Je suis reconnaissant(e) de la chance qui m'a été donnée de mettre en avant mes compétences et mes qualifications, et je suis vraiment enthousiaste à l'idée de rejoindre votre équipe.

Les perspectives acquises tout au long du processus d'entretien ont été enrichissantes, et j'ai hâte de les appliquer dans mes projets futurs. Je suis sincèrement enthousiaste à l'idée de contribuer à la mission et à la vision de [Nom de l'entreprise]. L'engagement de votre entreprise envers [mentionner des valeurs ou des initiatives spécifiques] résonne en moi, et je suis impatient(e) de faire partie d'une équipe aussi innovante et dynamique.

J'attends avec impatience d'avoir de vos nouvelles bientôt et espère pouvoir poursuivre nos discussions plus avant. Merci encore de considérer ma candidature.