ECOLEPMN

KANAGARAJAH ELILAN

PROJET FINAL

16 SEPTEMBRE 2024

SOMMAIRE

1-EXPLICATION DU PROJET

Ce projet de fin de formation est conçu pour valider les compétences nécessaires au titre professionnel d'Administrateur Système DevOps. L'objectif est de concevoir une infrastructure cloud automatisée, sécurisée, et performante, tout en respectant des pratiques DevOps modernes.

Nous commencerons par la mise en place d'une machine virtuelle (VM) dans le cloud, en utilisant Azure. Cette machine hébergera plusieurs services, chacun isolé dans son propre container Docker. Le choix de Docker permet de s'assurer que chaque service est indépendant et portable, facilitant ainsi la gestion et le déploiement. Sur cette infrastructure, nous déploierons un serveur web NGINX qui hébergera une application web. Cette application, développée en PHP, proposera des fonctionnalités CRUD (Create, Read, Update, Delete) pour interagir avec une base de données relationnelle, MariaDB, également déployée dans un container.

Au-delà du déploiement de services, une attention particulière est portée à la supervision. En production, il est crucial de surveiller la performance et l'état des services. Pour cela, nous mettons en place Prometheus, un outil de supervision qui permet de monitorer les containers, collecter des métriques, et alerter en cas de dysfonctionnement. L'ensemble du système sera donc constamment surveillé pour garantir son bon fonctionnement.

En termes de sécurité, des mesures sont prises pour protéger l'accès aux différents services. Le contrôle des accès se fait à travers des groupes de sécurité, en bloquant tous les ports non essentiels et en limitant les accès uniquement aux services nécessaires. Par ailleurs, pour les échanges web, nous ajoutons une couche de sécurité supplémentaire en utilisant des certificats SSL, garantissant que toutes les communications entre le serveur et les utilisateurs sont chiffrées.

L'un des enjeux de ce projet est d'automatiser au maximum ces processus. L'automatisation permet non seulement de déployer l'infrastructure rapidement, mais aussi de la rendre reproductible et prévisible. Grâce à des outils comme Ansible ou des scripts Bash, nous pouvons automatiser l'installation des services dans des containers, évitant ainsi de répéter manuellement chaque étape.

L'architecture globale que nous avons conçue met en avant une approche DevOps centrée sur l'automatisation, la portabilité, la supervision et la sécurité. Elle est conçue de manière à être flexible et adaptable pour évoluer selon les besoins. L'ensemble de cette solution reflète les pratiques professionnelles actuelles, où l'automatisation et la gestion des containers sont devenues des éléments centraux dans la gestion des infrastructures modernes.

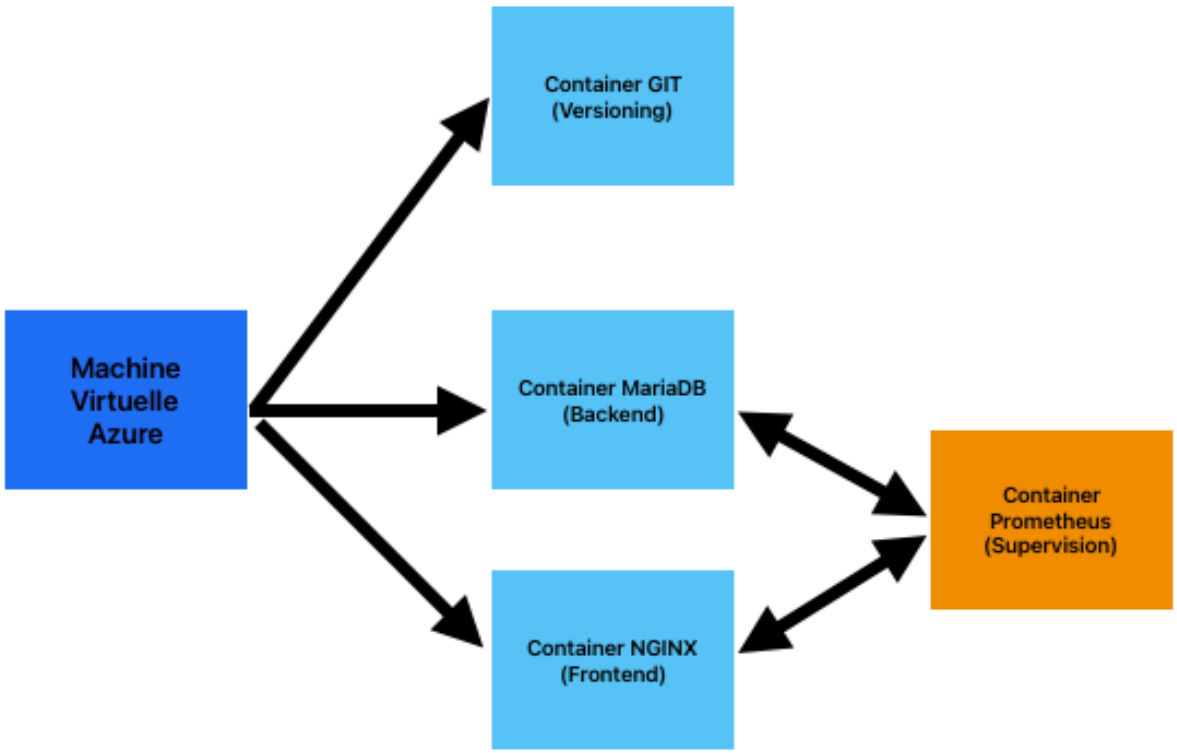


Diagramme Explicatif

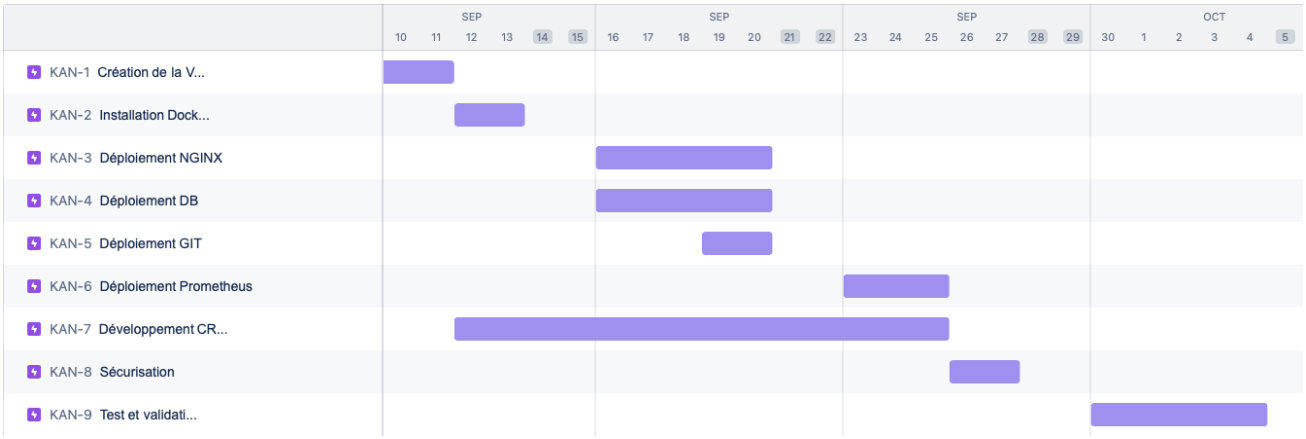


Diagramme de Gantt

2 -CHOIX DE LA PLATEFORME

Le choix de la plateforme est une étape cruciale, car il détermine les outils et services dont nous disposerons. Nous avons opté pour Azure, qui est reconnue pour sa fiabilité et sa large gamme de services. Azure propose des options de déploiement flexibles, allant des machines virtuelles aux services de conteneurs, tout en assurant une intégration fluide avec des outils de développement et d'automatisation. L'interface intuitive d'Azure permet de configurer rapidement des ressources, et ses capacités de scalabilité sont particulièrement avantageuses, car elles nous permettent d'ajuster facilement les ressources en fonction de la charge. Par ailleurs, la sécurité intégrée d'Azure, avec des outils de gestion des identités et des accès, nous rassure sur la protection des données et des services.

https://www.google.com/url?sa=i&url=https://news.microsoft.com/fr-fr/ms-azure_logo_stacked_c-gray_rgb/&psig=AOvVaw0i6R1gLCxVMS7XXp5u8-Wy&ust=1727086572232000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCICF5_6o1ogDFQAAAAAdAAAAABBx

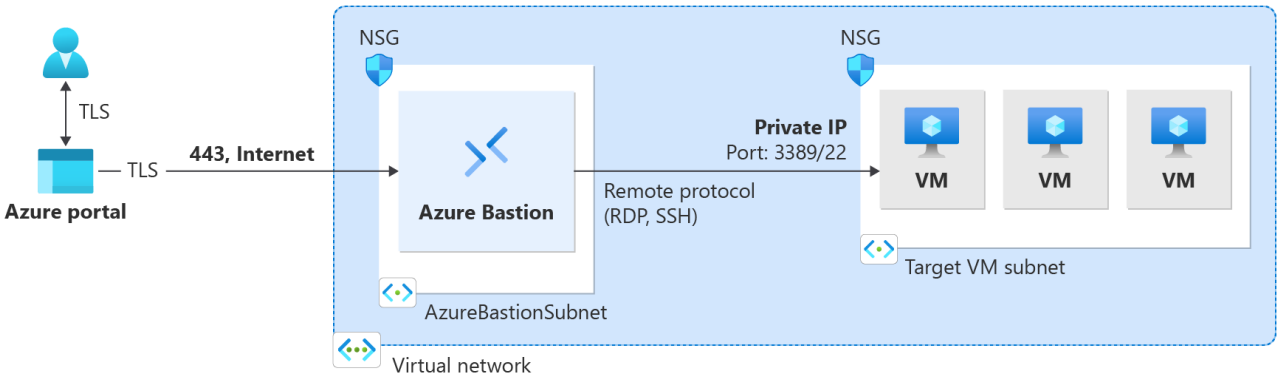
2 -CRÉATION D'UNE MACHINE VIRTUELLE

Dans le cadre de notre projet, l'utilisation du compte Azure de l'école nous a offert l'opportunité d'explorer les fonctionnalités avancées de la plateforme sans frais supplémentaires. Le processus a commencé par notre connexion au portail Azure, un espace convivial qui simplifie la gestion des ressources cloud.

Une fois connectés, nous avons navigué vers l'option de création d'une nouvelle machine virtuelle. L'interface nous a guidés à travers les différentes options disponibles, et nous avons choisi un système d'exploitation Linux, plus précisément Ubuntu, pour sa légèreté et sa popularité dans le milieu du développement.

La configuration de la machine virtuelle a été une étape cruciale. Nous avons sélectionné un type de VM qui était disponible dans le cadre du compte de l'école. En optant pour une instance B1S, nous savions que nous aurions suffisamment de ressources pour nos besoins tout en respectant les limites imposées par le niveau gratuit. En parallèle, nous avons pris soin de configurer le stockage, choisissant un disque SSD pour garantir des performances optimales.

Un autre aspect fondamental a été la sécurité. Nous avons mis en place des groupes de sécurité pour protéger notre VM. Cela impliquait la définition de règles spécifiques qui permettaient l'accès uniquement aux ports nécessaires, comme le port SSH, afin de limiter les connexions non autorisées. Ce souci de sécurité était primordial, car il garantissait que notre environnement de développement reste protégé tout en étant accessible pour notre travail.



https://learn.microsoft.com/fr-fr/azure/bastion/quickstart-host-portal

Une fois tous les paramètres configurés, il ne restait plus qu'à finaliser la création de la VM. Après quelques minutes d'attente, notre instance était prête. À ce moment-là, nous avons pu établir une connexion via SSH, ce qui nous a permis d'accéder à notre environnement et de commencer à installer les outils nécessaires, que nous verrons dans la prochaine partie.

3 -DÉPLOIEMENT DE DOCKER

Le déploiement de Docker sur notre machine virtuelle a été une étape déterminante dans la mise en place de notre projet. Tout a commencé lorsque nous avons établi une connexion SSH à notre VM Ubuntu. Une fois connectés, nous avons procédé à l'installation de Docker. En mettant à jour le système d'exploitation et en exécutant quelques commandes simples, nous avons rapidement installé Docker, profitant de la documentation claire qui guide le processus.

```
sudo apt update

sudo apt install apt-transport-https ca-certificates curl software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

sudo apt update

sudo apt install docker-ce

sudo systemctl status docker
```

Après l'installation, il a fallu s'assurer que tout fonctionnait parfaitement. Nous avons configuré Docker pour qu'il démarre automatiquement avec la machine virtuelle, ce qui nous permettrait d'y accéder sans tracas à chaque connexion. En ajoutant notre utilisateur au groupe Docker, nous avons facilité l'exécution des commandes, rendant notre travail plus fluide.

```
sudo usermod -aG docker $USER
```

4 -CONFIGURATION DES CONTAINERS

Avec Docker opérationnel, nous avons entamé la configuration de nos différents containers. Nous avons d'abord concentré nos efforts sur le container NGINX, chargé de servir notre front-end. Pour ce faire, nous avons commencé par rédiger un fichier de configuration, `nginx.conf`, qui définissait des éléments essentiels comme les règles de routage et les paramètres du serveur.

```
server {
    listen 80;
    server_name localhost;
    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
        try_files $uri $uri/ =404;
    }
}
```

Une fois la configuration peaufinée, nous avons lancé le container avec la commande suivante :

```
docker run -d --name nginx-container -p 80:80 -v /path/to/your/nginx.conf:/etc/
nginx/nginx.conf:ro -v /path/to/your/site:/usr/share/nginx/html nginx
```

En parallèle, nous avons également configuré le container MariaDB, qui jouerait un rôle clé en tant que base de données. Pour préparer ce container, nous avons créé une image

comprenant les utilisateurs et les schémas nécessaires. Nous avons utilisé des fichiers d'initialisation en SQL pour ce faire.

```
docker run -d --name mariadb-container -e MYSQL_ROOT_PASSWORD=root -e
MYSQL_DATABASE=mydatabase -v /path/to/your/db_data:/var/lib/mysql mariadb
```

Nous avons également mis en place un container Git, essentiel pour la gestion du versionnage de notre code.

```
docker run -d --name git-container -v /path/to/your/repo:/repo git
```

Enfin, nous avons intégré le container de supervision grâce à Prometheus. Pour cela, nous avons créé un fichier de configuration `prometheus.yml`, où nous avons spécifié les endpoints à surveiller.

```
# prometheus.yml

global:

  scrape_interval: 15s

scrape_configs:

  - job_name: 'nginx'

    static_configs:

      - targets: ['nginx-container:80']

  - job_name: 'mariadb'

    static_configs:

      - targets: ['mariadb-container:3306']
```

```
- job_name: 'git'
  static_configs:
    - targets: ['git-container:80']
- job_name: 'app'
  static_configs:
    - targets: ['app-container:8080']
```

Ensuite, nous avons lancé le container avec cette commande :

```
docker run -d --name prometheus-container -p 9090:9090 -v /path/to/your/
prometheus.yml:/etc/prometheus/prometheus.yml prometheus
```

5 -DÉVELOPPEMENT DE L'APPLICATION WEB

Avec nos containers en place, nous avons vraiment commencé à plonger dans le développement de notre application web en PHP. La première étape a été de structurer notre projet, en créant des dossiers pour le frontend et le backend. Pour le backend, nous avons rapidement mis en place un fichier de configuration, config.php, qui nous permettrait de connecter notre application à la base de données MariaDB. Dans ce fichier, on a défini les paramètres de connexion, comme l'hôte et le nom de la base de données.

```
<?php

$host = 'mariadb-container';

$dbname = 'mydatabase';

$username = 'root';

$password = 'root';

try {

   $pdo = new PDO("mysql:host=$host;dbname=$dbname", $username,
$password);

   $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

} catch (PDOException $e) {

   echo "Connection failed: " . $e->getMessage();

}
```

?>

Une fois cette connexion en place, nous avons commencé à construire les fonctionnalités de base de l'application, en nous concentrant sur les opérations CRUD : créer, lire, mettre à jour et supprimer des données. Pour la partie création, nous avons écrit un script create.php. Ce script prenait les données d'un formulaire et les insérait dans notre base de données.

```
<?php

require 'config.php';

if ($_SERVER["REQUEST_METHOD"] == "POST") {

    $data = $_POST['data'];

    $stmt = $pdo->prepare("INSERT INTO mytable (data) VALUES (:data)");

    $stmt->bindParam(':data', $data);

    if ($stmt->execute()) {

        echo "Data inserted successfully!";

    } else {

        echo "Error inserting data.";

    }

}

?>
```

Pour afficher les données, nous avons développé un fichier read.php. Ce script récupérait toutes les entrées de la base de données et les affichait dans un tableau.

```
<?php

require 'config.php';

$stmt = $pdo->prepare("SELECT * FROM mytable");

$stmt->execute();

$results = $stmt->fetchAll(PDO::FETCH_ASSOC);

foreach ($results as $row) {

    echo "<div>" . htmlspecialchars($row['data']) . "</div>";

}

?>
```

Ensuite, on s'est attaqué aux fonctionnalités de mise à jour et de suppression. Pour cela, nous avons créé un fichier update.php qui permettait à un utilisateur de modifier les données existantes. Ce fichier récupérait l'ID de l'entrée à modifier et affichait un formulaire prérempli.

```
<?php

require 'config.php';

if ($_SERVER["REQUEST_METHOD"] == "POST") {

    $id = $_POST['id'];

    $newData = $_POST['data'];

    $stmt = $pdo->prepare("UPDATE mytable SET data = :data WHERE id = :id");
```

```
    $stmt->bindParam(':data', $newData);
    $stmt->bindParam(':id', $id);
    if ($stmt->execute()) {
        echo "Data updated successfully!";
    } else {
        echo "Error updating data.";
    }
}
?>
```

Pour le fichier delete.php, c'était assez simple. Les utilisateurs pouvaient entrer l'ID de l'entrée qu'ils voulaient supprimer.

```
<?php
require 'config.php';
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $id = $_POST['id'];
    $stmt = $pdo->prepare("DELETE FROM mytable WHERE id = :id");
    $stmt->bindParam(':id', $id);
    if ($stmt->execute()) {
        echo "Data deleted successfully!";
```

```
        } else {
            echo "Error deleting data.";
        }
    }
    ?>
```

Tout au long de ce développement, nous avons veillé à utiliser des pratiques sécurisées. Par exemple, on a opté pour des instructions préparées pour éviter les problèmes de sécurité comme les injections SQL. En parallèle, on a utilisé Git pour suivre nos modifications. Ça nous a beaucoup aidés pour travailler en équipe et garder un historique de ce que nous avions fait.

Quand toutes les fonctionnalités CRUD étaient en place, on a commencé à tester l'application. Chaque script a été passé au crible pour vérifier qu'il fonctionnait comme prévu.

6 -SUPERVISION DES SERVICES

Sachant que nous avons déjà configuré le fichier prometheus.yml dans la partie précédente, nous sommes maintenant prêts à nous plonger dans la supervision des services. Cette étape est essentielle pour nous assurer que notre application fonctionne comme prévu et que nous pouvons rapidement identifier tout problème qui pourrait survenir.

Pour commencer, nous avons vérifié que Prometheus était bien en route et qu'il collectait les métriques des containers que nous avons déployés. En accédant à l'interface web de Prometheus à l'adresse *http://localhost:9090*, nous avons exploré l'onglet "Status" et ensuite "Targets". Cela nous a permis de voir la liste des endpoints surveillés et leur état. Si tout était bien en place, nos containers NGINX et MariaDB apparaîtraient avec un statut "UP", indiquant qu'ils étaient correctement monitorés.

Nous avons également pris le temps d'examiner les métriques disponibles. Par exemple, pour analyser la charge de travail sur NGINX, nous avons utilisé une requête dans l'onglet "Graph" pour voir combien de requêtes le serveur traitait par minute. Cela nous a donné un aperçu précieux de la performance de notre serveur.

```
rate(http_requests_total[5m])
```

En avançant, nous avons mis en place des alertes pour nous prévenir en cas de surcharge. Dans le fichier prometheus.yml, nous avons ajouté des règles d'alerte pour être avertis si le nombre de requêtes dépassait un seuil critique. Par exemple, nous avons intégré une alerte pour détecter si le nombre de requêtes par minute dépassait 100. Cela nous a permis de rester vigilants et réactifs face à d'éventuels problèmes.

```
groups:

- name: service_alerts

  rules:

  - alert: HighRequestRate

    expr: rate(http_requests_total[1m]) > 100
```

```
for: 1m
labels:
 severity: warning
annotations:
 summary: "High request rate detected on NGINX"
 description: "More than 100 requests per minute."
```

Une fois ces modifications apportées, nous avons redémarré le container Prometheus pour que les nouvelles règles prennent effet. Cela signifiait que Prometheus surveillerait activement nos services et nous alerterait en cas d'anomalie.

```
docker restart prometheus
docker ps
```

Nous avons également commencé à réfléchir à l'utilisation de Grafana pour visualiser les métriques de manière plus intuitive. Grafana propose des tableaux de bord dynamiques qui facilitent l'analyse des performances de notre application. En intégrant Grafana avec Prometheus, nous avons pu créer des graphiques clairs et attractifs, ce qui a grandement amélioré notre capacité à comprendre et à suivre le comportement de notre application au fil du temps. Cela a non seulement renforcé notre réactivité en cas de problème, mais a aussi enrichi notre compréhension globale de l'infrastructure que nous avons mise en place.

7 -SÉCURISATION DE L'INFRASTRUCTURE

La sécurisation de notre infrastructure est une étape cruciale pour protéger nos services et les données qu'ils traitent. Cela implique plusieurs aspects, allant de la configuration des groupes de sécurité dans Azure à l'implémentation de certificats SSL pour sécuriser les communications.

Pour commencer, nous avons configuré les groupes de sécurité dans Azure. Ces groupes nous permettent de contrôler le trafic entrant et sortant de nos machines virtuelles. Nous avons défini des règles pour restreindre l'accès aux ports essentiels, par exemple, en ne permettant l'accès SSH (port 22) qu'à certaines adresses IP. Cela réduit considérablement le risque d'attaques non autorisées.

```
az network nsg create --resource-group projet--name securite

az network nsg rule create --resource-group projet--nsg-name securite --name autoriser_ssh --protocol tcp --priority 1000 --destination-port-range 22 --access allow --source-address-prefixes <192.168.10.2>
```

Ensuite, nous avons mis en place des certificats SSL pour sécuriser les communications entre les utilisateurs et notre application. Pour cela, nous avons choisi d'utiliser Let's Encrypt, qui permet d'obtenir des certificats gratuits. L'installation de Certbot, un client pour automatiser la gestion des certificats.

```
sudo apt update

sudo apt install certbot

sudo certbot --nginx -d localhost:80
```

Cette commande configure automatiquement NGINX pour utiliser le certificat SSL, garantissant ainsi que notre site est accessible via HTTPS, offrant une couche de sécurité supplémentaire.

Nous avons également pris le temps de mettre en place des pratiques de gestion des mots de passe solides. Par exemple, nous avons veillé à ce que les mots de passe soient complexes et à les changer régulièrement. Pour notre base de données MariaDB, nous avons mis à jour les mots de passe des utilisateurs avec des commandes adaptées:

```
SET PASSWORD FOR 'mon_utilisateur'@'localhostt' = PASSWORD('EcolePMN2024');
```

Pour visualiser l'ensemble de notre architecture sécurisée, nous avons créé un diagramme qui met en lumière les différents composants et les mesures de sécurité mises en place.

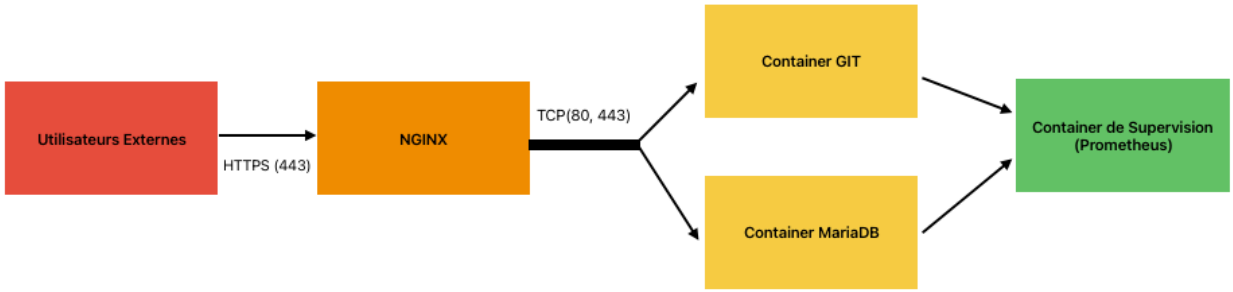


Diagramme de l'architecture Sécurisée

Avec cette configuration, nous avons non seulement sécurisé nos services, mais également créé un environnement robuste capable de résister à des tentatives d'intrusion. La sécurité est un processus continu, et nous sommes déterminés à réévaluer régulièrement nos mesures de sécurité pour nous adapter aux nouvelles menaces qui pourraient émerger.

8 -AUTOMATISATION DU DÉPLOIEMENT

L'automatisation du déploiement est un aspect fondamental de notre projet, car elle nous permet de gérer notre infrastructure de manière efficace et fiable. Nous avons choisi d'utiliser des outils comme Terraform et Ansible, qui facilitent la définition de notre infrastructure en tant que code et l'automatisation des installations et configurations.

Pour commencer, nous avons rédigé un fichier de configuration Terraform, que nous avons appelé main.tf. Ce fichier contient toutes les ressources dont nous avons besoin, comme la machine virtuelle, les groupes de sécurité et les réseaux.

Une fois le fichier de configuration préparé, nous avons exécuté quelques commandes simples pour déployer notre infrastructure. Tout d'abord, nous avons initialisé notre projet.

```
terraform init

terraform plan

terraform apply
```

Après avoir configuré notre infrastructure, nous avons utilisé Ansible pour automatiser l'installation des services sur notre machine virtuelle. Nous avons écrit un playbook appelé playbook.yml pour réaliser cette tâche.

```
- hosts: all

  become: true

  tasks:

    - name: Installer les dépendances
```

```
    apt:
      name:
        - apt-transport-https
        - ca-certificates
        - curl
        - software-properties-common
      state: present
  - name: Installer Docker
    apt:
      name: docker-ce
      state: latest
  - name: Démarrer et activer Docker
    systemd:
      name: docker
      state: started
      enabled: true
```

```
ansible-playbook -i hosts playbook.yml
```

9-TESTS ET VALIDATION

Une fois notre infrastructure déployée et notre application développée, nous avons consacré une attention particulière à la phase de tests et de validation. Cette étape est cruciale pour garantir que tous les composants fonctionnent comme prévu et que l'application répond aux exigences définies.

Nous avons commencé par établir une suite de tests qui couvrait différents aspects de notre application. D'abord, nous avons mis en place des tests unitaires. Pour ce faire, nous avons écrit des scripts PHP dédiés, permettant de vérifier chaque fonction individuellement. Par exemple, nous avons créé un test pour la fonction de création de données dans la base de données, en nous assurant qu'elle renvoyait les résultats escomptés.

```
public function testCreateData(){

   $data = ['name' => 'Test', 'value' => 123];

   $result = $this->createData($data);

   $this->assertTrue($result);

}
```

Une fois les tests unitaires en place, nous avons effectué des tests d'intégration. Ces tests vérifient comment différents modules interagissent entre eux. Par exemple, nous avons vérifié que notre application pouvait récupérer les données de la base de données et les afficher correctement sur le frontend. Pour cela, nous avons utilisé des outils comme Postman pour simuler des requêtes HTTP et inspecter les réponses.

Nous avons aussi réalisé des tests fonctionnels, où nous avons simulé des scénarios utilisateurs. Cela nous a permis de vérifier que toutes les fonctionnalités de l'application fonctionnaient de manière fluide. Nous avons ainsi testé des cas tels que l'ajout de nouvelles entrées, la modification et la suppression de données, tout en nous assurant que les messages d'erreur appropriés étaient affichés en cas de problème.

En parallèle, nous avons mis en place des tests de performance. Pour cela, nous avons utilisé des outils comme Apache JMeter pour simuler des charges utilisateurs. L'objectif était de voir comment notre application réagissait sous pression et d'identifier

d'éventuels goulets d'étranglement. Par exemple, nous avons testé le temps de réponse de notre API lors de multiples requêtes simultanées.

Pour chaque test effectué, nous avons documenté les résultats et les comportements observés. Cela nous a permis de vérifier que notre application répondait aux exigences initiales et de détecter rapidement les problèmes. En cas d'échec d'un test, nous avons analysé les logs pour identifier la source du problème et mis en œuvre les correctifs nécessaires.

Enfin, après avoir effectué tous les tests et validé que tout fonctionnait comme prévu, nous avons pris le temps de discuter des résultats en équipe. Cela nous a permis de nous assurer que tous les membres du groupe avaient une compréhension claire de l'application et des ajustements effectués. Cette phase de validation a renforcé notre confiance dans le déploiement final de l'application et nous a préparés à la phase de présentation.

10-PRÉSENTATION FINALE

Après plusieurs semaines de travail intensif, nous arrivons à la présentation finale du projet. Cette étape est décisive puisqu'elle synthétise l'ensemble des efforts fournis, des choix techniques, des défis rencontrés et des solutions mises en œuvre. L'objectif est de démontrer la compréhension du sujet et de présenter un produit fonctionnel qui répond au cahier des charges établi.

Introduction au projet

Nous avons débuté la présentation en introduisant les enjeux du projet, qui consistait à déployer une application web entièrement fonctionnelle, en utilisant des outils modernes de gestion d'infrastructure comme Docker et Terraform. L'application est conçue pour fonctionner dans un environnement cloud, avec une architecture basée sur des containers pour garantir modularité, scalabilité et sécurité.

L'introduction s'est focalisée sur l'importance du choix d'Azure comme plateforme cloud, en expliquant que cette solution a été choisie pour sa stabilité et ses capacités d'intégration avec des outils DevOps. Nous avons également mis en avant l'utilisation de Docker pour le déploiement des containers, ce qui a permis d'isoler les services et de les gérer de manière indépendante.

Démarche et méthodologie

Ensuite, nous avons décrit la méthodologie suivie tout au long du projet. Nous avons opté pour une approche itérative, où chaque étape – de la création de la machine virtuelle à la configuration des containers – a été réalisée par des cycles d'amélioration continue. Chaque composant du projet a été testé, optimisé, puis intégré au reste de l'infrastructure. Cela a permis une meilleure gestion des erreurs et une adaptation rapide aux défis techniques rencontrés.

Nous avons également expliqué que l'infrastructure a été gérée en tant que code grâce à Terraform, ce qui nous a permis d'automatiser et de versionner l'ensemble du processus de déploiement. Nous avons détaillé la manière dont Ansible a été utilisé pour automatiser l'installation et la configuration des différents services à l'intérieur des containers, rendant les processus plus fluides et moins sujets aux erreurs humaines.

Architecture du projet

Nous avons ensuite présenté un schéma de l'architecture globale du projet. Cette architecture repose sur plusieurs containers Docker :

-Un container NGINX pour le front-end

-Un container MariaDB pour la base de données

-Un container Git pour la gestion des versions

-Un container Prometheus pour la supervision

Ce schéma permet de visualiser l'organisation des différents composants de notre infrastructure. Chacun des services est déployé indépendamment dans son propre container, ce qui garantit que le problème d'un service ne perturbe pas les autres. Par exemple, si MariaDB rencontre un problème, NGINX continue de fonctionner normalement.

Nous avons également expliqué comment les données sont persistées et sécurisées via des volumes Docker, assurant ainsi que les informations ne sont pas perdues lors du redémarrage des containers. Cette approche garantit à la fois résilience et flexibilité.

Démonstration de l'application web

Nous avons ensuite procédé à une démonstration en direct de l'application web que nous avons développée. L'interface utilisateur, simple mais fonctionnelle, permettait de réaliser des opérations CRUD (Create, Read, Update, Delete) avec la base de données MariaDB. Nous avons montré comment l'utilisateur peut ajouter une nouvelle entrée, la modifier ou la supprimer, et comment ces actions sont reflétées instantanément dans la base de données.

L'accent a été mis sur la sécurité de l'application, en particulier sur la manière dont nous avons implémenté des mécanismes pour protéger contre les injections SQL. Nous avons également mentionné les certificats SSL générés pour sécuriser les communications entre le front-end NGINX et l'utilisateur final.

Pour illustrer la gestion du code, nous avons montré le fonctionnement du container Git, où chaque modification du code est versionnée, facilitant la collaboration et la traçabilité des changements. Cela a aussi permis de montrer comment, en cas de besoin, il était possible de revenir à une version antérieure du projet.

Monitoring et supervision

Nous avons ensuite présenté le système de monitoring avec Prometheus. Nous avons montré que l'application surveillait en temps réel l'état de santé des services critiques (comme NGINX et MariaDB), et que des alertes étaient prêtes à être déclenchées en cas d'anomalie, telles qu'une surcharge du CPU ou une chute du taux de disponibilité.

Le monitoring est un aspect essentiel, car il garantit que l'infrastructure est sous surveillance constante. Cela nous permet d'anticiper les problèmes avant qu'ils ne deviennent critiques et de maintenir un haut niveau de disponibilité.

Retour sur les défis rencontrés

Nous avons ensuite pris le temps d'expliquer certains des défis techniques que nous avons rencontrés. Par exemple, lors de l'intégration de la base de données avec le frontend, nous avons dû faire face à quelques problèmes de compatibilité entre les versions de MariaDB et PHP. Pour résoudre ces problèmes, nous avons dû ajuster certaines configurations et choisir des versions compatibles des services.

Un autre défi majeur concernait l'automatisation du déploiement. Bien que Terraform et Ansible aient largement facilité le processus, certaines étapes, comme la gestion des permissions d'accès et des règles de sécurité dans Azure, ont nécessité une attention particulière. Nous avons dû affiner nos scripts pour garantir que chaque service disposait des permissions minimales nécessaires pour fonctionner, sans compromettre la sécurité.

Sécurisation de l'infrastructure

Nous avons ensuite détaillé les mesures prises pour sécuriser l'infrastructure. Par exemple, nous avons configuré des groupes de sécurité dans Azure pour limiter l'accès aux services critiques uniquement aux adresses IP autorisées. Nous avons également mis en place un pare-feu au niveau des containers et activé des certificats SSL pour sécuriser les échanges de données.

Les règles de sécurité ont été gérées à travers Terraform, ce qui nous a permis de les appliquer de manière consistante sur tous nos environnements de test et de production. Nous avons expliqué comment cette automatisation réduit les risques d'erreurs humaines et garantit une infrastructure sécurisée à tout moment.

Conclusion et perspectives

Enfin, nous avons conclu la présentation en faisant un bilan de ce projet. Nous avons mis en avant l'apprentissage technique qu'il nous a permis d'acquérir, que ce soit dans l'utilisation des outils DevOps, la gestion d'infrastructures cloud ou encore le développement d'applications web. Le projet nous a également permis de renforcer nos compétences en gestion de projets complexes, tout en nous sensibilisant aux bonnes pratiques de sécurité et de supervision.

Pour l'avenir, nous avons évoqué quelques pistes d'amélioration, comme l'ajout d'une gestion des utilisateurs plus avancée, ou l'intégration d'outils CI/CD pour un déploiement encore plus automatisé. Nous avons également discuté de la possibilité d'ajouter des systèmes de sauvegarde pour les données critiques, ainsi que l'utilisation de services cloud managés pour simplifier encore davantage l'administration de l'infrastructure.

LETTRE DE REMERCIEMENT

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont pris le temps de lire et d'évaluer ce projet. Votre attention et vos retours sont précieux pour moi. Un grand merci également à mes professeurs, dont les enseignements et la patience ont été essentiels tout au long de cette formation. Vos conseils m'ont permis de grandir, non seulement en tant que professionnel, mais aussi en tant qu'individu.

Je souhaite également remercier chaleureusement Armand Devillard, dont l'aide a été inestimable tout au long de la réalisation de ce projet. Son expertise, ses conseils avisés et sa disponibilité ont joué un rôle clé dans l'avancée de chaque étape. Son soutien technique et moral a été une véritable source de motivation, et je lui suis reconnaissant d'avoir contribué à la concrétisation de ce projet.

Je tiens à ajouter que cette année a été, dans l'ensemble, une belle expérience enrichissante, tant sur le plan personnel que professionnel. J'ai eu l'opportunité d'apprendre et de développer de nouvelles compétences, ce qui m'a énormément apporté. Cependant, je dois reconnaître que l'école m'a parfois déçu sur certains aspects, notamment en termes d'organisation ou de certaines attentes qui n'ont pas été totalement comblées. Malgré ces moments de frustration, je garde en mémoire les aspects positifs et les connaissances acquises qui m'ont permis de mener à bien ce projet.

ANNEXE

https://azure.microsoft.com/fr-fr/get-started/azure-portal/

https://www.docker.com/blog/how-to-use-the-official-nginx-docker-image/

https://docs.github.com/fr/get-started/start-your-journey/about-github-and-git

https://ubuntu.com/download/server

https://prometheus.io/docs/introduction/overview/

https://grafana.com/docs/grafana/latest/

https://developer.hashicorp.com/terraform?product_intent=terraform

https://docs.ansible.com/?extIdCarryOver=true&sc_cid=701f2000001OH7YAAW

ENGLISH: PROJECT OVERVIEW

This final project serves to validate the skills required for achieving the professional title of DevOps System Administrator. The goal is to create a cloud infrastructure that is automated, secure, and efficient while following modern DevOps principles.

We start by setting up a virtual machine (VM) in the cloud, using Azure as our platform. This VM will host several services, each running in its own Docker container. By using Docker, we can ensure that each service is independent and portable, making it easier to manage and deploy. On this infrastructure, we'll deploy an NGINX web server that will host a web application developed in PHP. This app will include basic functionality like creating, reading, updating, and deleting (CRUD) records, which will interact with a MariaDB relational database, also running in its own container.

Monitoring the services is just as important as deploying them. In a production environment, it's crucial to keep an eye on performance and ensure everything is running smoothly. To do this, we've integrated Prometheus, a monitoring tool that helps track container performance, collect metrics, and send alerts if something goes wrong. This way, the entire system will be constantly monitored to ensure its health and reliability.

Security plays a key role in this infrastructure. We've implemented strict access controls, using security groups to block non-essential ports and limit access only to critical services. Additionally, we've added SSL certificates to encrypt all communications between the server and users, providing an extra layer of protection.

One of the main goals of this project is to automate as much as possible. Automation not only speeds up the deployment of the infrastructure but also makes it repeatable and reliable. Using tools like Ansible or Bash scripts, we can automate the setup of services in containers, avoiding the need to manually repeat tasks each time.

The architecture we've designed highlights a DevOps-focused approach, prioritizing automation, portability, monitoring, and security. It's built to be flexible and adaptable, capable of growing and evolving as needed. This project reflects current industry standards, where automation and containerized environments are at the core of managing modern infrastructures.