Documentation du Projet Veliko

1. Introduction

1.1 Contexte du projet

Le projet Veliko vise à développer une solution permettant aux utilisateurs d'accéder en temps réel aux informations de disponibilité des vélos en libre-service à Paris et en Île-de-France. Ce projet repose sur une application web et un environnement déployé sur une infrastructure multi-serveurs.

1.2 Objectif

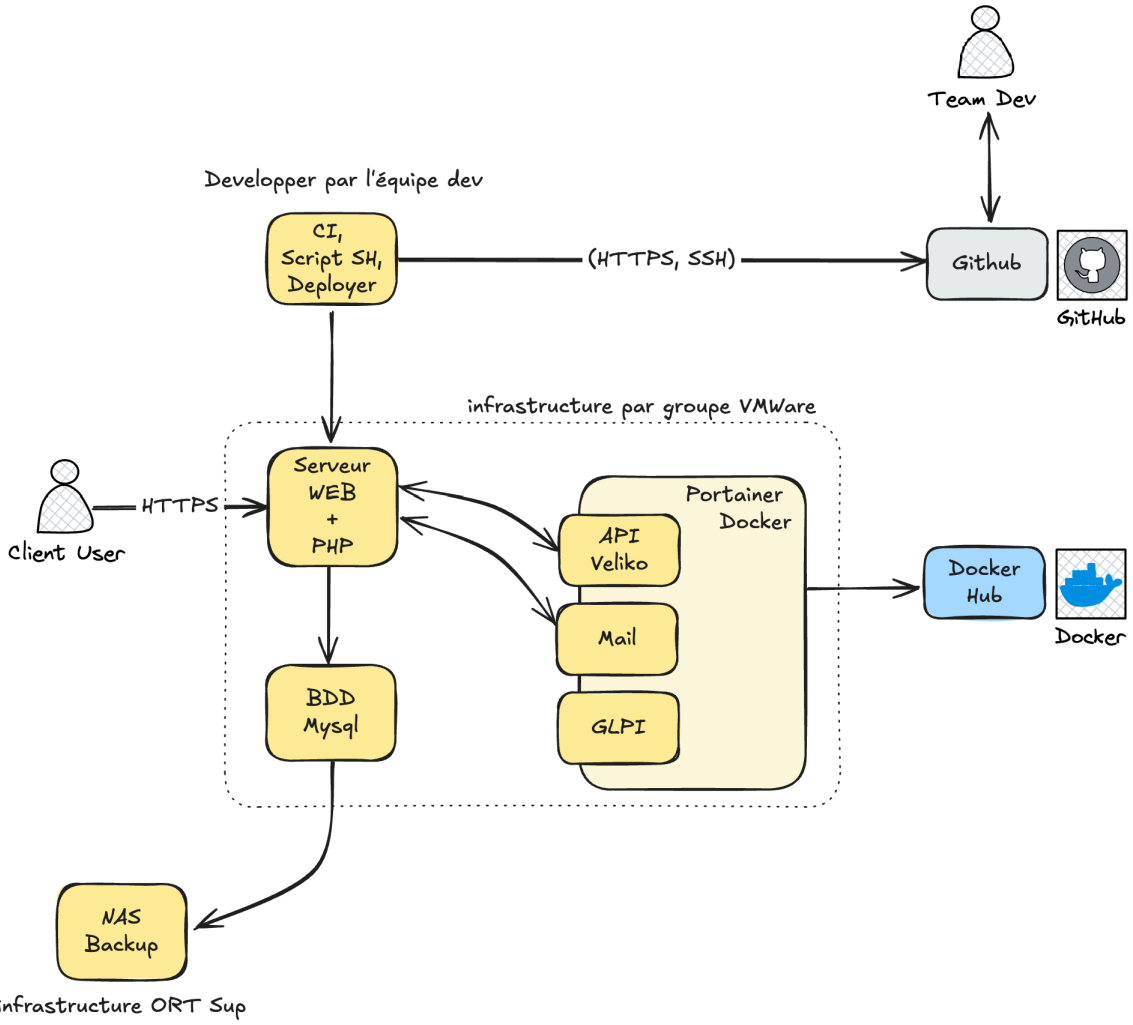
L'objectif principal était de mettre en place une infrastructure stable et fonctionnelle pour supporter l'application web et les différents services requis. En tant que DevOps, le travail a consisté à :

- Mettre en place des machines virtuelles (VM) pour déployer les différents services.
- Automatiser les déploiements via un Webhook.
- Assurer la communication entre les différentes VMs et leur sauvegarde via un NAS.
- Configurer un serveur Windows pour le domaine et l'Active Directory (AD).

2. Architecture de l'infrastructure

L'infrastructure repose sur quatre machines virtuelles (VM) hébergées sous **VMWare ESXi**. Voici la topologie du projet :

2.1 Schéma global



2.2 Détail des machines

Machine	Rôle	Adresse IP	Services
VM1	Infrastructure Docker	192.168.17.161	API Veliko, Mail, GLPI
VM2	Serveur Web + PHP	192.168.17.7	Symfony (Application web)
VM3	Serveur Base de données	192.168.17.94	MariaDB, Backup NAS
VM4	Serveur Windows	192.168.17.4	Active Directory, DNS

3. Mise en place des machines virtuelles

3.1 VM1 - Infrastructure Docker

Objectif

Mettre en place une VM pour déployer des conteneurs Docker gérant les services suivants :

- **API Veliko** : Service backend de l'application.
- **Serveur Mail** : Gestion des emails.
- **GLPI** : Gestionnaire de parc informatique.

Étapes réalisées

1. **Installation de Docker et Portainer** :

```
sudo apt update
sudo apt install -y docker.io
sudo systemctl start docker
sudo systemctl enable docker
sudo docker volume create portainer_data
sudo docker run -d -p 9000:9000 --name=portainer --restart=always \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v portainer_data:/data portainer/portainer-ce
```

2. **Configuration des services Docker avec Portainer** :
   - API Veliko, Mail et GLPI ont été déployés via un stack Docker Compose.
3. **Tests et validation** :
   - Accès à Portainer via `http://192.168.17.161:9000`.
   - Tests des services conteneurés.

3.2 VM2 - Serveur Web + PHP (Symfony)

Objectif

Mettre en place un serveur LAMP et déployer l'application Symfony.

Étapes réalisées

1. **Installation de LAMP** :

```
sudo apt update
sudo apt install -y apache2 mariadb-server php libapache2-mod-php p
hp-mysql \
   php-xml php-mbstring php-curl
```

2. **Installation de Composer** :

```
curl -sS https://getcomposer.org/installer | php
sudo mv composer.phar /usr/local/bin/composer
```

3. **Clonage du projet Symfony** :

```
git clone git@github.com:projet/veliko.git /var/www/html/veliko
cd /var/www/html/veliko
composer install
```

4. *Création du fichier **`` dans le dossier `public` :

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.*)$ index.php [QSA,L]
```

5. **Configuration Apache pour Symfony** : Modification du fichier de configuration Apache :

```
<VirtualHost *:80>
   DocumentRoot /var/www/html/veliko/public
   <Directory /var/www/html/veliko/public>
      AllowOverride All
      Require all granted
   </Directory>
</VirtualHost>
```

6. *Configuration du fichier **`` :

```
DATABASE_URL="mysql://veliko_user:Respons11@192.168.17.3:3306/vel
iko_db?serverVersion=10.11.2-MariaDB&charset=utf8mb4"
```

7. **Mise à jour de la base de données** :

```
php bin/console doctrine:migrations:migrate --no-interaction
php bin/console doctrine:schema:update --force
```

8. **Tests et validation** :
    - Accès à l'application via l'URL du serveur.

3.3 VM3 - Serveur MariaDB + Backup NAS

Objectif

Configurer une base de données centralisée avec des sauvegardes automatiques.

Étapes réalisées

1. **Installation de MariaDB** :

```
sudo apt update
sudo apt install -y mariadb-server
sudo systemctl enable mariadb
sudo systemctl start mariadb
```

2. **Configuration de la base** :

```
CREATE DATABASE veliko_db;
CREATE USER 'veliko_user'@'%' IDENTIFIED BY 'Respons11';
GRANT ALL PRIVILEGES ON veliko_db.* TO 'veliko_user'@'%';
FLUSH PRIVILEGES;
```

3. **Sauvegarde via NAS** :
    - Montage du NAS via NFS.
    - Automatisation avec un script Bash pour les backups.

3.4 VM4 - Serveur Windows (Domaine et AD)

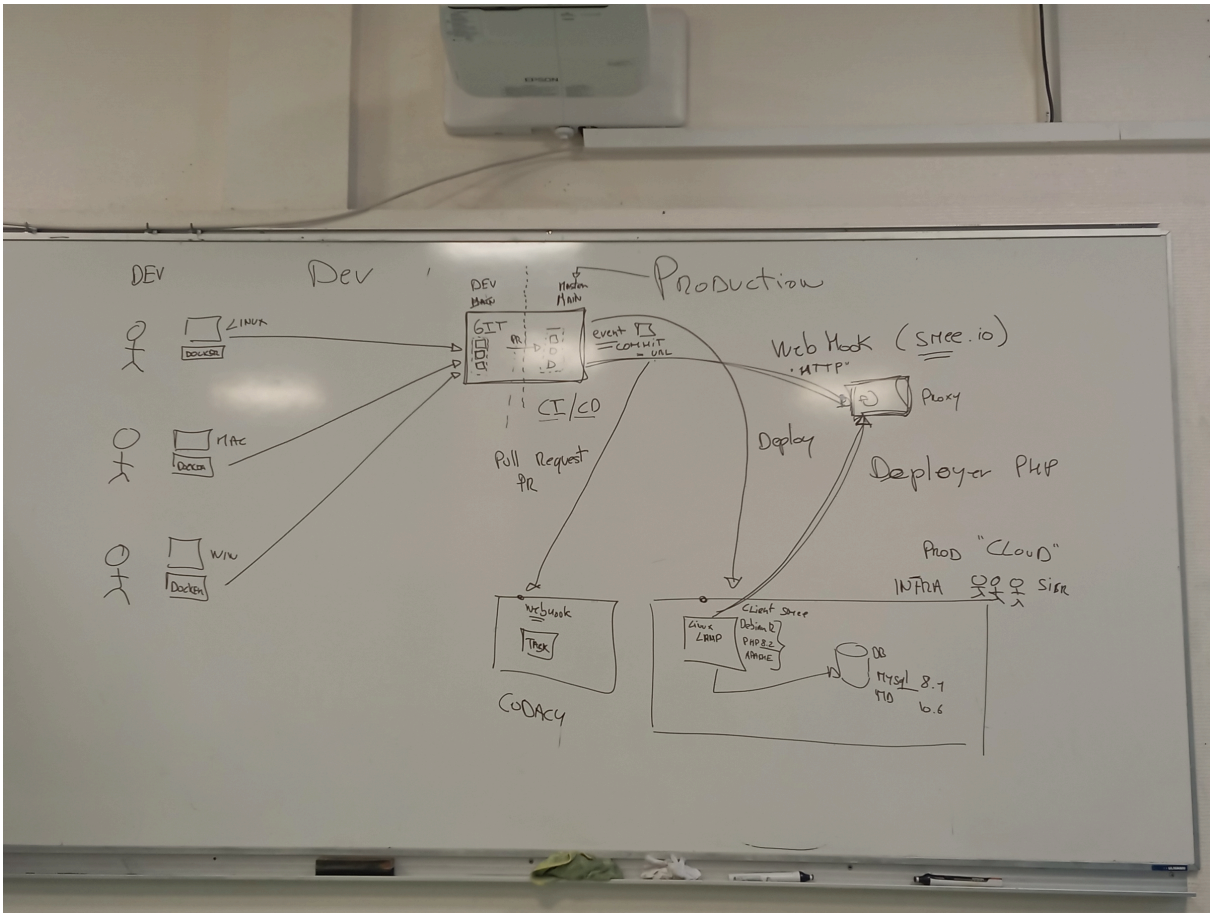
Objectif

Mettre en place un serveur Windows pour l'Active Directory, le DNS et la gestion des utilisateurs du domaine `gr4.veliko.lan`.

Étapes réalisées

1. **Installation de Windows Server** :
   - Installer Windows Server 2019 ou 2022 sur VM4.
2. **Configuration des services Active Directory** :
   - Ouvrir l'assistant **Ajouter des rôles et fonctionnalités**.
   - Ajouter le rôle **Active Directory Domain Services**.
   - Configurer le domaine `gr4.veliko.lan`.
3. **Configuration DNS** :
   - Le rôle DNS sera installé automatiquement avec l'AD.
   - Ajouter les enregistrements A pour les machines (VM1, VM2, VM3).
4. **Gestion des utilisateurs et des groupes** :
   - Créer des utilisateurs pour l'accès au domaine.
5. **Vérification des communications** :
   - Tester la résolution de noms depuis les machines Linux.

Documentation Complète : Automatisation de Déploiement avec Webhook, Smee.io, et PM2



Introduction

Cette documentation détaille la mise en place d'un système de déploiement automatique pour une application Symfony. Ce système utilise un webhook GitHub relayé via Smee.io, un script d'automatisation écrit en Node.js, et des processus gérés par PM2 pour une exécution continue.

Pré-requis

1. Environnement serveur

- **VM2** : Serveur Web + PHP (avec Apache et Symfony installé).
- **Node.js** installé sur le serveur.
- **PM2** pour gérer les processus.

2. Accès GitHub

- Un dépôt configuré avec les branches nécessaires.
- Permissions pour configurer les Webhooks dans le dépôt GitHub.

3. Autres outils

- **Smee.io** : Un outil pour relayer les événements Webhook.
- **Composer** : Gère les dépendances PHP.

Étape 1 : Préparation de l'environnement

1. Installer les dépendances système

Assurez-vous que Node.js et npm sont installés sur la VM :

```
apt update
apt install -y nodejs npm
```

Installez PM2 pour gérer les processus :

```
npm install -g pm2
```

2. Préparer le répertoire pour le Webhook

Créez un répertoire dédié pour les scripts d'automatisation :

```
mkdir -p /var/www/html/veliko/bin/auto-deploy
cd /var/www/html/veliko/bin/auto-deploy
```

Étape 2 : Initialisation du projet Node.js

1. Initialiser le projet Node.js

Dans le répertoire créé, initialisez un projet Node.js :

```
npm init -y
```

Cela crée un fichier `package.json` qui gère les dépendances du projet.

2. Installer les dépendances nécessaires

Installez les modules nécessaires :

```
npm install express dotenv
```

Cela ajoute les dépendances dans le fichier `package.json` :

```
{
  "dependencies": {
    "express": "^4.18.2",
    "dotenv": "^16.0.3"
  }
}
```

Étape 3 : Configuration du Webhook

1. Créer le fichier `webhook.js`

Ajoutez le fichier suivant dans `/var/www/html/veliko/bin/auto-deploy/webhook.js` :

```
const express = require('express');
const { exec } = require('child_process');
const fs = require('fs');
const crypto = require('crypto');
require('dotenv').config({ path: '/var/www/html/veliko/.env' });

const app = express();
app.use(express.json());

// Fonction pour vérifier la signature
function verifySignature(req, secret) {
    const payload = JSON.stringify(req.body);
    const signature = `sha256=${crypto.createHmac('sha256', secret).updat
e(payload).digest('hex')}`;
    return req.headers['x-hub-signature-256'] === signature;
}

app.post('/webhook', (req, res) => {
    console.log('Webhook reçu');
    fs.appendFileSync('./webhook.log', `Webhook reçu : ${JSON.stringify(re
```

```
q.body)}\\n`);

    if (!verifySignature(req, process.env.WEBHOOK_SECRET)) {
        console.error('Signature invalide. Requête rejetée.');
        fs.appendFileSync('./webhook.log', 'Signature invalide. Requête rejeté
e.\\n');
        return res.status(401).send('Invalid signature');
    }

    console.log('Signature valide. Déploiement en cours...');
    exec('./deploy.sh', (error, stdout, stderr) => {
        if (error) {
            console.error(`Erreur lors du déploiement : ${error.message}`);
            fs.appendFileSync('./webhook.log', `Erreur : ${error.message}\\n`);
            return res.status(500).send('Erreur lors du déploiement');
        }

        console.log(`Déploiement réussi : ${stdout}`);
        fs.appendFileSync('./webhook.log', `Sortie : ${stdout}\\n`);
        res.status(200).send('Déploiement terminé avec succès');
    });
});

app.listen(3000, '0.0.0.0', () => {
    console.log('Serveur Webhook en écoute sur le port 3000');
    fs.appendFileSync('./webhook.log', `${new Date().toISOString()} - Serve
ur Webhook démarré sur le port 3000\\n`);
});
```

1. **Génération de la clé secrète**
   Pour sécuriser les communications entre GitHub et votre serveur, générez une clé secrète :

```
openssl rand -hex 32
```

Copiez cette clé et ajoutez-la dans le fichier `.env` sous la forme :

2. Ajouter la clé secrète dans `.env`

Dans le fichier `.env` du projet Symfony, ajoutez une clé secrète pour sécuriser le Webhook :

```
WEBHOOK_SECRET=my_secret_key
```

Ajoutez la même clé dans les paramètres du Webhook GitHub.

Étape 4 : Script de déploiement `deploy.sh`

1. Créez le fichier `deploy.sh` :

   ```
   nano /var/www/html/veliko/bin/auto-deploy/deploy.sh
   ```

2. Contenu du fichier :

   ```
   #!/bin/bash

   APP_DIR="/var/www/html/veliko"
   BRANCH="main"

   echo "$(date '+%Y-%m-%d %H:%M:%S') - ⟹ Début du déploiement automatique" >> ./webhook.log

   cd "$APP_DIR" || exit
   git fetch origin
   git reset --hard origin/$BRANCH

   COMPOSER_ALLOW_SUPERUSER=1 composer install --no-dev --optimize-autoloader --no-interaction

   php bin/console doctrine:schema:update --force
   php bin/console cache:clear --env=prod
   php bin/console cache:warmup --env=prod

   systemctl restart apache2
   echo "$(date '+%Y-%m-%d %H:%M:%S') - ⟹ Déploiement terminé avec succès !" >> ./webhook.log
   ```

3. Rendre le script exécutable :

```
chmod +x /var/www/html/veliko/bin/auto-deploy/deploy.sh
```

Étape 5 : Automatisation avec PM2

1. Ajouter `webhook.js` à PM2 :

```
pm2 start /var/www/html/veliko/bin/auto-deploy/webhook.js --name w
ebhook-server
```

2. Ajouter Smee.io à PM2 :

```
pm2 start "smee --url <https://smee.io/your_smee_url> --target <http://
192.168.17.7:3000/webhook>" --name smee-client
```

3. Sauvegarder la configuration :

```
pm2 save
pm2 startup
```

4. Redémarrer le serveur pour vérifier la persistance :

```
reboot
```

Étape 6 : Test et validation

1. Vérifiez que les processus PM2 sont actifs :

```
pm2 list
```

2. Testez un push GitHub et vérifiez les logs :

```
git add .
git commit -m "Test webhook avec clé"
```

```
git push origin main
tail -f /var/www/html/veliko/bin/auto-deploy/webhook.log
```

Conclusion

Cette documentation couvre toutes les étapes nécessaires pour configurer un système de déploiement automatique, sécurisé, et persistant. Si tout est suivi correctement, chaque push dans le dépôt GitHub déclenchera un déploiement automatisé sur le serveur Web.

sans la clé :

```
app.post('/webhook', (req, res) ⇒ {
console.log('Webhook reçu');
fs.appendFileSync('./webhook.log', Webhook reçu : ${JSON.stringify(req.b
ody)}\\n);

// Suppression temporaire de la vérification de la clé
console.log('Clé secrète désactivée pour les tests.');

// Exécuter le script deploy.sh
exec('./deploy.sh', (error, stdout, stderr) ⇒ {
    if (error) {
        console.error(`Erreur lors du déploiement : ${error.message}`);
        fs.appendFileSync('./webhook.log', `Erreur : ${error.message}\\n`);
        return res.status(500).send('Erreur lors du déploiement');
    }

    console.log(`Déploiement réussi : ${stdout}`);
    fs.appendFileSync('./webhook.log', `Sortie : ${stdout}\\n`);
    res.status(200).send('Déploiement terminé avec succès');
```

```
});
});
```