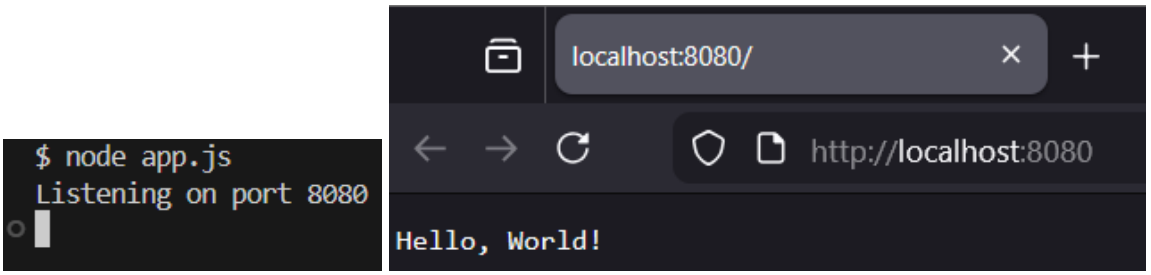
Lab 1 :

An Introduction to Deploying Apps

Faire tourner une application localement

Objectif de l'étape : L'objectif principal est de maîtriser le déploiement de base sur un serveur unique, ici notre propre machine. Cela nous permet de construire et tester le code localement avant de l'exposer sur un serveur distant.

Protocole suivi :
Ouvrir un terminal Cygwin sur VS Code.
Accéder au disque local via le chemin /cygdrive/c/.
Créer la structure de répertoire devops_base/lab1/sample-app et s'y déplacer.
Créer le fichier app.js à l'aide de l'éditeur nano ou de tout autre éditeur, et le remplir avec le contenu fourni par le sujet.
S'assurer que Node.js est correctement installé et configuré (cf lab 0).
Démarrer l'application depuis le terminal Cygwin.



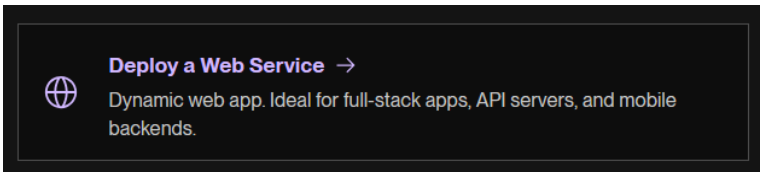
Explication : L'application est un serveur utilisant le module http de Node.js. Elle est configurée pour répondre à toutes les requêtes avec un code de statut 200 et le texte "Hello, World!". Par défaut, elle écoute sur le port 8080 sauf si une variable d'environnement PORT est spécifiée. Le serveur fonctionne sur localhost (127.0.0.1), un nom d'hôte pointant vers l'interface réseau locale accessible seulement depuis notre propre ordinateur.

Déployer une application avec Render (Platform as a Service)

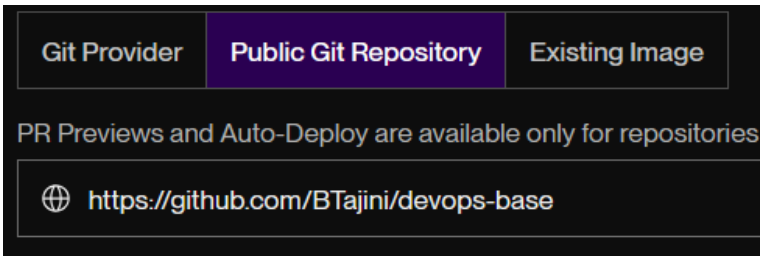
Objectif : Découvrir le modèle PaaS (Platform as a Service) en déployant l'application sur un serveur cloud. Le PaaS nous permet de se concentrer sur le packaging et le déploiement de

l'application sans avoir à gérer manuellement l'installation des dépendances, la configuration du réseau ou les serveurs sous-jacents.

Protocole suivi :
Aller sur la plateforme d'hébergement cloud Render render.com, et créer un compte.
Dans https://dashboard.render.com/, cliquer sur Deploy a Web Service.



Connecter le dépôt Git public contenant le code en entrant l'URL du dépôt https://github.com/BTajini/devops-base et en cliquant sur Connect.



Configurer le service avec les paramètres suivants :
- le nom : sample-app,
- le répertoire racine (Root Directory) : ch1/sample-app,
- la commande de démarrage (Start Command) : node app.js,
- Le type d'instance : Free ($0 / Month).
- Laisser les autres paramètres par défaut.

Cliquer sur Deploy Web Service et sur l'onglet Logs pour suivre l'avancement du déploiement.

Suspendre l'application.

Explication : Render permet de déployer du code sans configurer de build complexe ou de conteneurs. L'utilisateur connecte un dépôt Git public, configure le serveur avec le répertoire racine ch1/sample-app du dépôt et lance l'exécution via la commande node app.js sur une instance gratuite. Une fois l'instance démarrée, l'application web est accessible mondialement avec une URL publique sécurisée en HTTPS.

Déployer une application avec AWS (Infrastructure as a Service)

Objectif : Expérimenter le modèle IaaS (Infrastructure as a Service), qui donne un accès direct aux ressources informatiques de bas niveau comme les serveurs. Contrairement au PaaS (Render), l'IaaS oblige à gérer soi-même les aspects logiciels comme les installations des dépendances, la configuration réseau et la sécurité.

Protocole suivi :

Se rendre à l'adresse https://aws.amazon.com et créer un compte AWS (Root).
Se rendre à l'adresse console.aws.amazon.com/iam/ et créer un compte utilisateur IAM (Identity and Access Management).
Ajouter la politique AdministratorAccess pour éviter d'utiliser les identifiants racines au quotidien : Dans Permission options, choisir Attach policies directly, et dans Permissions policies, chercher et sélectionner les politiques prédéfinies AdministratorAccess (AWS managed - job function). Cliquer sur créer l'utilisateur, et enregistrer les informations d'authentification. Se connecter avec cet utilisateur IAM.

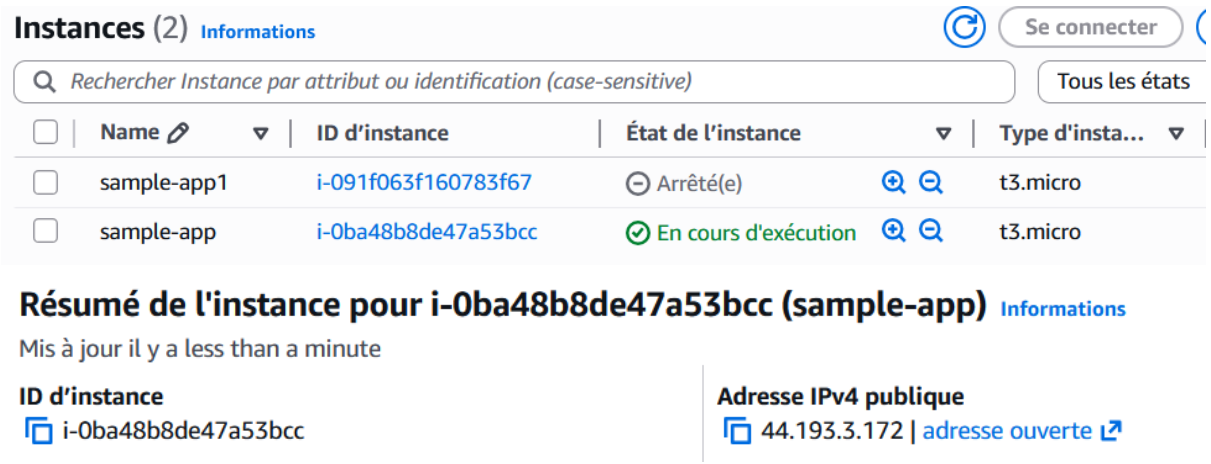
Configurer l'instance EC2 : Se rendre sur https://console.aws.amazon.com/ec2/home et cliquer sur Lancer une instance. Choisir les paramètres suivants :

- le nom : sample-app
- la paire de clés (connexion) : Continuer sans paire de clés
- désactiver l'Autorisation du trafic SSH
- activer l'Autorisation du trafic HTTP depuis l'Internet

Dans détails avancés, données utilisateur, ajouter

```
#!/usr/bin/env bash
set -e
# [1]
curl -fsSL https://rpm.nodesource.com/setup_21.x | bash -
yum install -y nodejs
# [2]
tee app.js > /dev/null << "EOF"
const http = require('http');
const server = http.createServer((req, res) => {
res.writeHead(200, { 'Content-Type': 'text/plain' });
res.end('Hello, World!\n');
});
# [3]
const port = process.env.PORT || 80;
server.listen(port,() => {
console.log(`Listening on port ${port}`);
});
EOF
# [4]
nohup node app.js &
```

Cliquer sur Lancer l'instance.



Terminer l'instance.

Explication : Nous louons une machine virtuelle EC2 sur laquelle nous avons un contrôle total. Le script User Data automatise la configuration logicielle que le PaaS gérait pour nous. Ce script Bash est exécuté au premier démarrage du serveur pour installer Node.js via yum, écrire le code de l'application dans app.js, et lancer l'application en arrière-plan avec nohup node app.js & pour qu'elle continue de tourner après l'arrêt du script. Le port 80 est utilisé (port standard pour le trafic web HTTP). Une fois l'instance en état "Running", l'application est accessible sur l'adresse IP publique fournie sur AWS.

Résumé (ce que nous avons retenu de ce lab) :

Le laboratoire a permis de comparer trois méthodes de déploiement d'une même application Node.js :

1. Localement, en faisant tourner l'application directement sur son propre ordinateur avec node app.js.
2. Avec un modèle PaaS (plateforme Render), le déploiement est automatisé en connectant un dépôt Git public à Render qui récupère le code et l'expose sur une URL publique. Il n'y a pas besoin de configurer manuellement le serveur.
3. Avec un modèle IaaS. AWS est utilisé pour louer des serveurs virtuels nommés instances EC2. L'utilisateur a un contrôle total sur l'environnement (création d'un compte IAM, Security Group…). Le script User Data est exécuté au démarrage pour installer Node.js et lancer l'application en arrière-plan.