Les commande docker

docker run: Lance un conteneur à partir d'une image Docker spécifiée.

docker build: Construit une nouvelle image Docker à partir d'un fichier de configuration appelé Dockerfile.

docker pull: Télécharge une image Docker depuis un registre, par exemple Docker Hub.

docker push: Envoie une image Docker vers un registre, par exemple Docker Hub, afin de la rendre disponible pour d'autres utilisateurs.

docker stop: Arrête l'exécution d'un ou plusieurs conteneurs.

docker start: Démarre un ou plusieurs conteneurs arrêtés.

docker restart: Redémarre un ou plusieurs conteneurs.

docker rm: Supprime un ou plusieurs conteneurs.

docker rmi: Supprime une ou plusieurs images Docker.

docker ps: Affiche les conteneurs en cours d'exécution.

docker images: Liste les images Docker présentes sur le système.

docker exec: Exécute une commande à l'intérieur d'un conteneur en cours d'exécution.

docker logs: Affiche les journaux de sortie d'un conteneur.

docker inspect: Récupère des informations détaillées sur un ou plusieurs conteneurs, images, réseaux ou volumes.

docker network: Gère les réseaux Docker, notamment la création, la suppression et la gestion des connexions entre les conteneurs.

Les commandes Docker-compose :

docker-compose up: Démarre les conteneurs définis dans le fichier docker-compose.yml. Si les conteneurs n'existent pas, ils seront créés.

docker-compose down: Arrête et supprime les conteneurs définis dans le fichier docker-compose.yml. Les volumes Docker associés ne sont pas supprimés par défaut, mais vous pouvez utiliser l'option --volumes pour les supprimer également.

docker-compose build: Construit ou reconstruit les images des services définis dans le fichier docker-compose.yml. Cela est utile lorsque vous avez apporté des modifications au code source ou aux dépendances.

docker-compose restart: Redémarre les conteneurs définis dans le fichier docker-compose.yml sans les reconstruire.

docker-compose stop: Arrête les conteneurs définis dans le fichier docker-compose.yml sans les supprimer.

docker-compose start: Démarre les conteneurs définis dans le fichier docker-compose.yml qui ont été arrêtés.

docker-compose ps: Affiche l'état des conteneurs définis dans le fichier docker-compose.yml, y compris les ports exposés.

docker-compose logs: Affiche les journaux de sortie des conteneurs définis dans le fichier docker-compose.yml.

docker-compose exec: Exécute une commande à l'intérieur d'un conteneur en cours d'exécution défini dans le fichier docker-compose.yml.

docker-compose down --volumes: Arrête et supprime les conteneurs définis dans le fichier docker-compose.yml, ainsi que les volumes Docker associés.

Définition du cloud :

Le terme « Cloud » désigne les serveurs accessibles sur Internet ainsi que les logiciels et bases de données qui fonctionnement sur ces serveurs

. Les avantages du cloud :

- Faible coût et disponibilité continue - Maintenance allégée et automatisée - Les employés peuvent travailler de n'importe où.

Définition du cloud native :

Le Cloud Native décrit une approche de développement logiciel dans laquelle les applications sont dès le début conçues pour une utilisation sur le Cloud.

Les avantages du cloud native :

- La flexibilité : les modifications apportées au code n'ont pas d'impact sur le logiciel dans son ensemble. - L'évolutivité : éviter la mise à niveau coûteuse du matériel en cas d'augmentation des exigences pour un service. - L'automatisation : qui permet d'éliminer les erreurs. - La vitesse et l'agilité : répondre rapidement aux conditions du marché

Définition de l'architecture micro-services :

L'architecture micro-services consiste à décomposer l'application en un ensemble de petites services indépendantes appelés micro-services. Chaque micro-service possède son propre logique et sa propre base de données.

Les avantages de l'architecture micro-services :

- L'agilité technologique : le choix technologique dans une architecture micro□services peut être adapté aux besoins spécifiques de chaque micro-service - Un déploiement continue et rapide : grâce à l'utilisation des conteneurs - La scalabilité : selon le trafic du système, on peut ajouter des instances d'un micro□service ou en détruire.

Définition de RabbitMQ :

RabbitMQ est un système de messagerie open-source basé sur le modèle de message broker. Il agit comme un intermédiaire entre les applications distribuées, leur permettant d'échanger des messages de manière fiable et asynchrone

Les avantages de RabbitMQ :

- La durabilité : les données ne seront pas perdues en cas de pannes du serveur ou de redémarrage. - Haut disponibilité et mise en cluster : plusieurs nœuds RabbitMQ travaillent ensemble pour assurer la fiabilité et la disponibilité des messages.

Définition de Azure :

Azure est la plateforme de cloud computing de Microsoft. Il s'agit d'un ensemble de services et de ressources informatiques disponibles à la demande via Internet. Il permet aux entreprises de déployer, gérer et exécuter diverses applications et charges de travail.

Les avantages de Azure :

- Évolutivité : faire face à des besoins variables en termes de ressources informatiques. - Fiabilité : grâce à la réplication des données et à la redondance des services sur des centres de données multiples. - Sécurité : des fonctionnalités de sécurité avancées pour protéger les données et les applications. - Flexibilité : création et déploiement des applications hybrides. - Tarification flexible : payer uniquement pour les ressources réellement utilisées. - Intégration avec les outils Microsoft. - Large gamme de services.

Définition de Docker :

Docker est une plateforme qui permet de créer facilement des conteneurs et des applications basées sur les conteneurs.

Avantages des conteneurs :

- La portabilité : les conteneurs peuvent s'exécuter sur n'importe quelle plateforme ou cloud. - L'isolation : les conteneurs fonctionnent d'une manière indépendante. - Mise à l'échelle flexible : le nombre des conteneurs démarrés est en fonction du nombre d'utilisateurs. - La facilité de gestion : grâce à l'utilisation des plateformes d'orchestration des conteneurs. - La vitesse

Définition de Dockerfile :

Un Dockerfile est un fichier qui liste les instructions à exécuter pour construire une image.

```
const express = require('express');
const app = express();
const mysql = require('mysql');
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '',
  database: 'four',
});
app.use(express.json()); // Enable JSON request body parsing
// Get all fournisseurs
app.get('/fournisseur', (req, res) => {
  connection.query('SELECT * FROM fournisseur', (error, results) => {
    if (error) {
      console.error(error);
      return res.status(500).json({ error: 'Internal Server Error' });
    }
    res.status(200).json(results);
  });
});

// Get fournisseur by ID
app.get('/fournisseur/:id', (req, res) => {
  const id = req.params.id;
  connection.query('SELECT * FROM fournisseur WHERE id = ?', [id], (error, results) => {
    if (error) {
      console.error(error);
      return res.status(500).json({ error: 'Internal Server Error' });
    }
    if (results.length === 0) {
      return res.status(404).json({ error: 'fournisseur not found' });
    }
    res.status(200).json(results[0]);
  });
});

// Create a new fournisseur
app.post('/fournisseur', (req, res) => {
  const { Reseau_social, Adresse, Telephone } = req.body;
  connection.query(
    'INSERT INTO fournisseur (Reseau_social, Adresse, Telephone) VALUES (?, ?, ?)',
    [Reseau_social, Adresse, Telephone],
    (error, result) => {
      if (error) {
        console.error(error);
        return res.status(500).json({ error: 'Internal Server Error' });
      }
      const id = result.insertId; // Retrieve the auto-generated id
      res.status(201).json({ id, Reseau_social, Adresse, Telephone });
    }
  );
});

// Update fournisseur by ID
app.put('/fournisseur/:id', (req, res) => {
  const id = req.params.id;
  const { Reseau_social, Adresse, Telephone } = req.body;
  connection.query(
    'UPDATE fournisseur SET Reseau_social = ?, Adresse = ?, Telephone = ? WHERE id =
?',
    [Reseau_social, Adresse, Telephone, id],
    (error) => {
      if (error) {
        console.error(error);
        return res.status(500).json({ error: 'Internal Server Error' });
      }
      res.status(200).json({ id, Reseau_social, Adresse, Telephone });
    }
  );
});

// Delete fournisseur by ID
app.delete('/fournisseur/:id', (req, res) => {
  const id = req.params.id;
  connection.query('DELETE FROM fournisseur WHERE id = ?', [id], (error, results) => {
    if (error) {
      console.error(error);
      return res.status(500).json({ error: 'Internal Server Error' });
    }
    if (results.affectedRows === 0) {
      return res.status(404).json({ error: 'fournisseur not found' });
    }
    res.status(200).json({ message: 'fournisseur deleted' });
  });
});

app.listen(8080, () => {
  console.log("Server is running on port 8080");              });
```
