Professeur : Hamid Mcheick	Groupe : **en équipe** de deux étudiants
Trimestre : Automne 2024	Date de distribution : 2 octobre 2024
Pondération : 15 points	Date de remise : 12 novembre 2024

Parmi les sujets donnés ici-bas, **vous devez choisir la Question I ou deux autres questions (2, 3, 4, 5, 6)** :
• Le rapport et les documents doivent être remis (téléversés) à la page web du cours. Des pénalités de 10% par jour seront appliquées pour tout travail remis après cette date.

Question I

Dans ce projet, vous allez déployer une application web à deux niveaux avec une base de données et un frontend, en utilisant des services, des déploiements et des volumes persistants.

Étape 1: Prérequis

Assurez-vous d'avoir Minikube, Docker et kubectl installés.

1) **Installer Docker**: Kubernetes utilise Docker pour gérer les conteneurs.
   Sur Ubuntu:

```
sudo apt-get update
sudo apt-get install docker.io
```

2) **Installer kubectl: C'est l'outil de ligne de commande pour interagir avec le cluster Kubernetes.**
   Sur Ubuntu:

```
sudo apt-get update && sudo apt-get install -y apt-transport-https
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add - echo "deb
https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sourc
sudo apt-get update
sudo apt-get install -y kubectl
```

3) **Installer Minikube: C'est un outil qui permet de créer un cluster Kubernetes localement.**
   Sur Ubuntu:

```
curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux
sudo mv minikube /usr/local/bin/
```

Étape 2: Déployer une base de données

1. **Déploiement de la base de données**:

Créez un fichier `mysql-deployment.yaml`:

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
name: mysql
spec:
replicas: 1
selector:
matchLabels:
app: mysql
template:
metadata:
labels:
app: mysql
spec:
containers:
- name: mysql
image: mysql:5.7
env:
- name: MYSQL_ROOT_PASSWORD
value: "password"
ports:
- containerPort: 3306
```

Appliquez le déploiement:

```
kubectl apply -f mysql-deployment.yaml
```

2. **Service MySQL:**

Créez un fichier `mysql-service.yaml`:

```
apiVersion: v1
kind: Service
metadata:
name: mysql
spec:
ports:
- port: 3306
selector:
app: mysql
```

Appliquez le service:

```
kubectl apply -f mysql-service.yaml
```

Étape 3: Déployer une application frontend

1) **Déploiement du frontend**:

Créez un fichier `frontend-deployment.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: frontend
```

```
spec:
replicas: 2
selector:
matchLabels:
app: frontend
template:
metadata:
labels:
app: frontend
spec:
containers:
- name: frontend
image: [VOTRE_IMAGE_FRONTEND]
ports:
- containerPort: 80
```

Remplacez `[VOTRE_IMAGE_FRONTEND]` par l'image Docker de votre frontend.

Appliquez le déploiement:

```
kubectl apply -f frontend-deployment.yaml
```

2) **Service Frontend:**

Créez un fichier `frontend-service.yaml`:

```
apiVersion: v1
kind: Service
metadata:
name: frontend
spec:
type: NodePort
ports:
- port: 80
targetPort: 80
selector:
app: frontend
```

Appliquez le service:

```
kubectl apply -f frontend-service.yaml
```

Étape 4: Volumes Persistants

Pour que la base de données conserve ses données même après un redémarrage, nous utiliserons unvolume persistant.

1. **Créer un PersistentVolumeClaim**:

Créez un fichier `mysql-pvc.yaml`:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: mysql-pvc
```

```
spec:
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 1Gi
```

Appliquez le PVC:

```
kubectl apply -f mysql-pvc.yaml
```

2. **Modifiez le fichier mysql-deployment.yaml pour ajouter le volume:**

```
...spec:
containers:
...
volumeMounts:
- name: mysql-persistent-storage
mountPath: /var/lib/mysql
volumes:
- name: mysql-persistent-storage
persistentVolumeClaim:
claimName: mysql-pvc
```

Appliquez les modifications:

```
kubectl apply -f mysql-deployment.yaml
```

Étape 5: Accéder à l'application

Utilisez la commande suivante pour obtenir l'URL du frontend:

```
minikube service frontend --url
```

Visitez l'URL pour accéder à votre application.

Étape 6: Nettoyage

Supprimez les ressources créées:

```
kubectl delete -f frontend-service.yaml
kubectl delete -f frontend-deployment.yaml
kubectl delete -f mysql-service.yaml
kubectl delete -f mysql-deployment.yaml
kubectl delete -f mysql-pvc.yaml
```

Arrêtez Minikube:

```
minikube stop
```

Description du Projet Kubernetes à Deux Niveaux

Dans ce projet, nous avons déployé une application web structurée en deux niveaux sur un clusterKubernetes local à l'aide de Minikube. L'architecture est composée des éléments suivants:

1) **Base de données (Backend)**:

Déploiement MySQL: Nous avons utilisé une image Docker officielle de MySQL pour déployer une instance de base de données dans notre cluster.
Service MySQL: Pour permettre à d'autres composants du cluster de communiquer avec MySQL, nous avons créé un service Kubernetes.
Volume Persistant: Afin de garantir la persistance des données de la base de données, même après le redémarrage des pods ou du cluster, nous avons utilisé un volume persistant. Ce volume est lié à un Persistent Volume Claim (PVC) qui réserve une certaine quantité d'espace de stockage pour notre base de données.
Application Frontend: **Déploiement Frontend**: Cette partie représente l'interface utilisateur de notre application. Nous avons supposé que vous aviez une image Docker pour le frontend, qui serait déployée dans le cluster.

2) **Service Frontend:** Pour rendre l'application frontend accessible depuis l'extérieur du cluster, nous avons créé un service de type NodePort.

L'application frontend communique avec la base de données MySQL pour effectuer diverses opérations CRUD (Create, Read, Update, Delete). Grâce à l'architecture de Kubernetes, il est possible de mettre àl'échelle, de surveiller et de gérer facilement cette application à deux niveaux.

Question II

Objective

The purpose of this exercise is to demonstrate the fundamental principles of load balancing using HAProxy on an Ubuntu system. We will set up HAProxy to distribute incoming HTTP requests between two simple Node.js web servers running on the same machine. This will illustrate how HAProxy can efficiently manage traffic to ensure high availability and reliability of services.

What is Load Balancing?
Load balancing is a technique used to distribute network or application traffic across multiple servers. This distribution helps in optimizing resource use, maximizing
throughput, minimizing response time, and avoiding overload on any single server. Load balancers like HAProxy act as the traffic cop sitting in front of your serversand routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance.

What is HAProxy?
HAProxy is a free, open-source software that provides a high availability load balancer and proxy server for TCP and HTTP-based applications that spread requests

across multiple servers. It is particularly suited for very high traffic web sites and powers quite a number of the world's most visited ones.

Exercise Overview
In this exercise, we will:
1. Install Node.js and npm, which are prerequisites for running our simple web servers.
2. Write the source code for two basic Node.js web servers.
3. Start both web servers on different ports on the same Ubuntu machine.
4. Install HAProxy and configure it to balance the load between our two Node.js servers.
5. Test the load balancing to ensure HAProxy correctly distributes incoming HTTP requests between the two servers.
6. Access the HAProxy statistics page to monitor its performance and the status of the backend servers.
By the end of this exercise, you will have a working example of a load-balanced application using HAProxy, which you can use as a starting point for more complex load balancing scenarios.

Question III

En se basant sur le laboratoire du cours (Docker, …), vous devez réaliser les deux points suivants (1) et (2) :
(1). Créer une image Docker :
• Choisissez une application simple comme un serveur Web (Apache, nginx, etc.)
• Écrivez un Dockerfile pour créer une image de l'application (par exemple héberger une page html statique, un site web personnel, etc.)
• Utilisez l'interface de ligne de commande Docker pour créer l'image et la tester en exécutant un conteneur à partir de l'image et en accédant au serveur Web dans un navigateur

(2). Docker Hub :
• Transférez l'image Docker créée à l'étape 1 vers Docker Hub
• Extrayez l'image de Docker Hub et exécutez-la sur une autre machine pour vérifier qu'elle fonctionne correctement.

Évaluation de la mission :
(1). (80%) Implémenter les fonctionnalités de l'application et fournir un code source bien documenté pour développer cette application.
(2). (20%) Démonstration et documentation.

Notes:
• Vous pouvez utiliser n'importe quel langage de programmation pour développer l'application avec lesquels vous êtes à l'aise
• Fournir des instructions claires sur la façon d'exécuter l'application et de la tester
• Inclure des captures d'écran et/ou des extraits de code pour illustrer les étapes que vous avez suivies.

Question IV

Considérons les modèles de services en Cloud Computing suivants :

IaaS, PaaS, SaaS et FaaS

Taches à réaliser :

a) Expliquer brièvement ces modèles (20%)
b) Développer un exemple de code complet (application exécutable) de chaque modèle (60%)
c) Identifier les avantages et désavantages de ces modèles (20%)

Question V

Le Kubernates orchestre multi-conteneurs sur plusieurs nœuds à travers le monde. Cette technologie permet de rendre plusieurs services dans le Cloud Computing, tels que :

- Gérer les déploiements et leurs emplacements
- gérer l'équilibrage des charges
- gérer la communication entre les conteneurs
- gérer la découverte de services
- gérer les mises à jour
- gérer la montée en échelle
- gérer le stockage nécessaire à la persistance des données
- gérer la configuration et les secrets
- Mais aussi, gérer les dysfonctionnements.

Vous allez étudier l'architecture du Kubernetes et développer les points suivants :

a) Montrer et expliquer l'architecture du Kubernetes, composantes, virtualisation, etc. 25%
b) Faire une étude de cas : utiliser Kubernetes pour développer un exemple de code complet exécutable 25%
c) Identifier les avantages et les inconvénients de Kubernetes 25%
d) Comparer Kubernetes avec les conteneurs Docker et les machines virtuelles MV par rapport aux : maintenance, Scalabilité et types d'interaction (even-driven) 25%

Question VI

gRPC est utilisé à l'interne par Google et dans Kubernetes, ainsi que par quelques compagnies comme Netflix, Cisco, Morgan Stanley...

Vous allez étudier cette approche gRPC et développer les points suivants :

a) Expliquer l'architecture du gRPC, composantes, etc. 20%
b) Faire une étude de cas : utiliser gRPC pour développer une application avec code complet 60%
c) Identifier les avantages et les inconvénients de gRPC 20%

Voici quelques documents utiles pour ce TP :

- Comprendre le fonctionnement de gRPC https://grpc.io/docs/guides/

- Comprendre le fonctionnement de gRPC avec C++ https://grpc.io/docs/tutorials/

basic/cpp/

- Comprendre le fonctionnement et le rôle du traçage système avec LTTng https://lttng.

org/docs/v2.10/#doc-nuts-and-bolts

- Courte introduction au fonctionnement de Docker : https://docs.docker.com/get-started/

ion au fonctionnement de Docker : https://docs.docker.com/get-started/