

Rapport Virtualisation

Rayan SOBH

Valentin DUMAS

Ce rapport présente le processus de transformation d'une application de quiz initialement conçue pour un projet à l'ESIEE. L'accent est mis sur le déploiement et l'orchestration de l'application dans un environnement Kubernetes, illustrant la transition vers des pratiques de déploiement cloud-native et l'adoption de conteneurs pour améliorer la scalabilité, la gestion, et la portabilité de l'application.



Le projet initial du quiz était structuré autour de deux composants principaux : une API back-end avec le front-end Flask, et une interface front-end développée en Vue.js, offrant une expérience utilisateur interactive et réactive. Pour faciliter le déploiement et assurer une séparation claire entre le back-end et le front-end, chaque partie était encapsulée dans sa propre image Docker, préparant le terrain pour une orchestration plus sophistiquée avec Kubernetes. Le langage utilisé est du **Python**



Les dépendances et les bibliothèques utilisés sont disponibles dans le fichier requirements.txt

Docker

En utilisant `docker build -t gabi201265/quiz-prod-api:latest`, on crée des images docker pour l'API et une commande similaire pour l'UI, en me plaçant dans le répertoire contenant le Dockerfile approprié. Les images ont été publiées sur Docker Hub en utilisant `docker push`, permettant ainsi leur déploiement facile sur Kubernetes.

Deploiement Kubernetes

Dans le cadre du projet de migration vers Kubernetes, la création des déploiements pour les composants quiz-api et quiz-ui a été une étape cruciale pour assurer la scalabilité et la gestion efficace de l'application dans un environnement cloud.

L'objectif de cette phase était de déployer de manière fiable et scalable notre application quiz, composée de deux parties principales : l'API backend et l'interface utilisateur frontend. Pour cela, nous avons utilisé Kubernetes, un système d'orchestration de conteneurs, qui nous permet de gérer automatiquement le déploiement, la mise à l'échelle et l'exploitation de nos conteneurs d'applications.

Pour y arriver, il a fallu créer des fichiers de configuration YAML pour Kubernetes. Ces fichiers décrivent le déploiement souhaité, y compris le nombre de réplicas, les images Docker à utiliser, et la configuration nécessaire pour exposer l'application.

Voici un exemple pour l'API backend développé avec Flask :

My-quiz-api-deployment.yaml :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: quiz-api-deployment
  namespace: quiz-namespace
spec:
  replicas: 2
  selector:
    matchLabels:
      app: quiz-api
  template:
    metadata:
      labels:
        app: quiz-api
    spec:
      containers:
        - name: quiz-api
          image: gabi201265/quiz-prod-api:latest
          ports:
            - containerPort: 5000
```

```
! my-quiz-api-deployment.yaml
! my-quiz-api-service.yaml
! my-quiz-namespace.yaml
! my-quiz-ui-deployment.yaml
! my-quiz-ui-service.yaml
```

my-quiz-api-deployment.yaml: Ce fichier contient la définition du déploiement pour votre API back-end. Il spécifie les détails tels que l'image Docker à utiliser, le nombre de réplicas, les ports à ouvrir et d'autres paramètres essentiels au déploiement de l'API.

my-quiz-api-service.yaml: Ce fichier définit le service Kubernetes pour l'API. Un service Kubernetes est une abstraction qui définit un ensemble de pods et une politique d'accès à eux, souvent via un réseau. Ce service permet à d'autres applications ou utilisateurs d'accéder à votre API, potentiellement à travers un équilibrage de charge. Le type de service que nous avons utilisé est ClusterIP.

my-quiz-namespace.yaml: Ce fichier définit un espace de noms (namespace) dans Kubernetes. Les espaces de noms permettent de séparer les ressources au sein d'un cluster, ce qui facilite la gestion des permissions, la limitation des ressources et l'organisation logique de vos composants Kubernetes.

my-quiz-ui-deployment.yaml: De manière similaire au déploiement de l'API, ce fichier configure le déploiement de l'interface utilisateur de votre application. Il détaille comment les instances de votre UI doivent être créées, gérées et mises à l'échelle.

my-quiz-ui-service.yaml: Ce fichier crée un service Kubernetes pour l'interface utilisateur, permettant aux utilisateurs finaux d'accéder à votre application via un navigateur web. Comme nous l'avons configuré en LoadBalancer, il peut également attribuer une adresse IP externe pour que le service soit accessible sur Internet.

Après avoir créé ces fichiers de configuration, ils ont été appliqués au cluster Kubernetes en utilisant la commande `kubectl apply -f <fichier>.yaml` pour chaque déploiement.

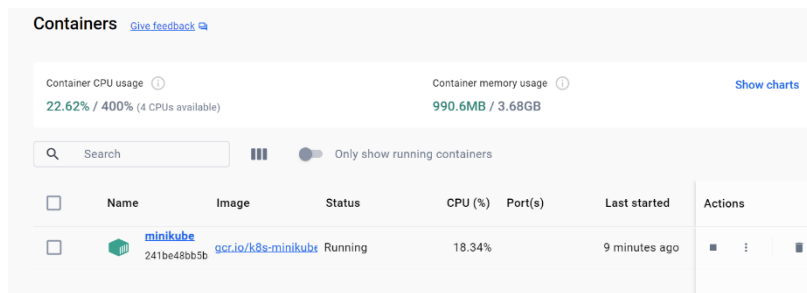
Nous avons ensuite vérifié l'état des déploiements et des pods pour nous assurer que tout fonctionnait comme prévu :

```
kubectl get deployments
```




```
kubectl get pods
```

Test de l'application

L'objectif de cette section est de détailler les procédures mises en place pour tester et valider l'application quiz au sein de notre cluster Kubernetes. Le processus débute par le lancement de Docker Desktop, qui est requis pour notre utilisation de Minikube en tant que couche d'abstraction pour Kubernetes en local. Avec le service Docker actif, nous positionnons le contexte Docker sur les paramètres par défaut via la commande `docker context use default`. Ensuite, Minikube est démarré avec `minikube start --driver=docker`, instaurant notre environnement de test local.



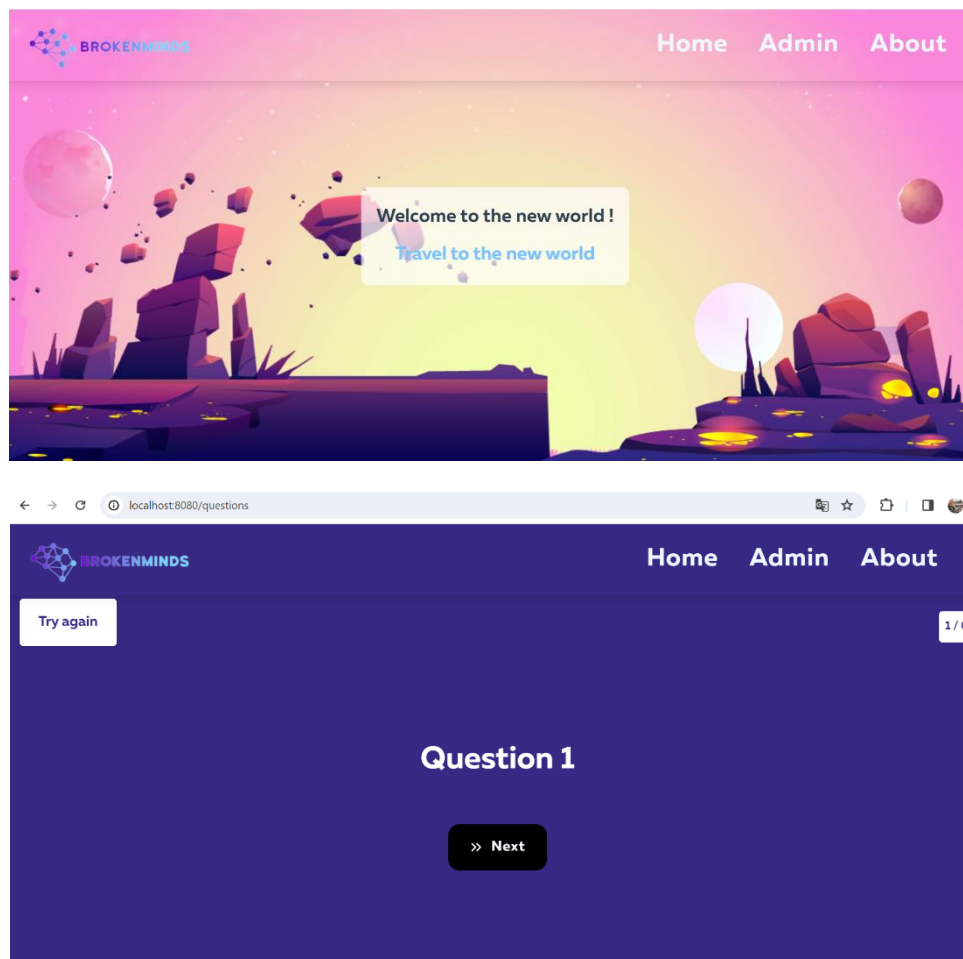
The screenshot shows the Docker Desktop interface. At the top, it displays 'Containers' with a 'Give feedback' link. Below this, there are two summary cards: 'Container CPU usage' at 22.62% / 400% (4 CPUs available) and 'Container memory usage' at 990.6MB / 3.68GB. A 'Show charts' link is also present. Below the summary cards is a search bar and a toggle for 'Only show running containers'. The main table lists the containers:

	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
<input type="checkbox"/>	minikube	gcr.io/k8s-minikube	Running	18.34%		9 minutes ago	  

Une fois Minikube actif, comme le confirme la commande `minikube status`, nous procédons à la redirection d'un port local vers le service de notre application en utilisant

`kubectrl port-forward service/quiz-ui-service 8080:80 -n quiz-namespace`.

Maintenant, se connecter à <http://localhost:8080/>



Cependant, il a été observé que les questions de quiz ne s'affichaient pas sur l'interface utilisateur, suggérant une défaillance dans la communication avec le backend. Le dépannage initial incluait la vérification des logs des pods et des services, mais des investigations plus approfondies sont nécessaires pour résoudre complètement ce problème.