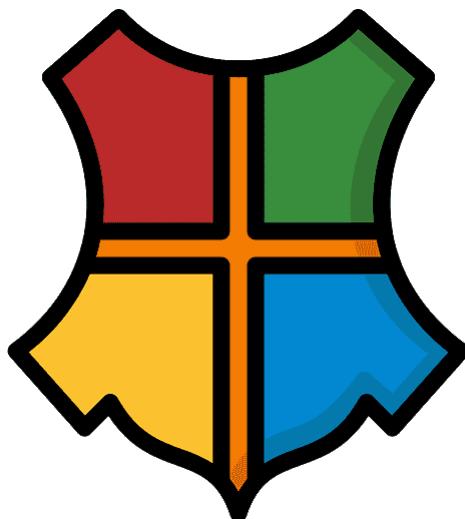


# Projet de virtualisation



*PotterPlus*



Lucie DESRIAUX • Richard FOUQUOIRE • Emily RENARD

# Table des matières

<b>Progression sur les Qwiklabs Google</b>	<b>3</b>
<b>Introduction</b>	<b>6</b>
<b>Présentation du projet</b>	<b>7</b>
<b>Mise à disposition des composants de l'application</b>	<b>10</b>
Création des images via Dockerfile	10
Envoi des images sur la registry Docker Hub	13
<b>Mise en place du cluster kubernetes</b>	<b>15</b>
Configuration des deployments	16
Configuration des services	18
Communication entre services	20
quiz-api <-> mail-api	20
quiz-ui <-> mail-api	20
Mise en place du gateway Ingress	20
<b>Déploiement sur un cloud provider</b>	<b>23</b>
Configuration d'un nouveau projet sur Google Cloud	23
Configuration du cluster kubernetes	24
Re-configuration du gateway ingress	25
Attribution d'un nom de domaine et configuration DNS	27
<b>Conclusion</b>	<b>29</b>

# Progression sur les Qwiklabs Google

Lucie DESRIAUX

Lucie Desriaux  
Date d'abonnement : 2023  
Votre profil n'est pas public ni accessible.  
[Rendre votre profil public](#)

Parcours de formation      Activités      Badges

Cours   Atelier   Quête   Quiz   Jeu   En cours   Terminée

Activité	Type	Date de début	Date de fin	Score	Réussie
La journalisation dans le cloud est un moteur Kubernetes	Atelier	il y a 13 jours	il y a 13 jours	100.0/100.0	✓
Gérer l'état Terraform	Atelier	il y a 13 jours	il y a 13 jours	100.0/100.0	✓
Interact with Terraform Modules	Atelier	il y a 13 jours	il y a 13 jours	100.0/100.0	✓
Infrastructure as Code avec Terraform	Atelier	6 mars 2023	6 mars 2023	100.0/100.0	✓
Principes de base de Terraform	Atelier	6 mars 2023	6 mars 2023	100.0/100.0	✓
Creating Infrastructure on Google Cloud with Terraform	Quête	6 mars 2023	il y a 13 jours		✓
Configuring Networks via gcloud	Atelier	15 févr. 2023	15 févr. 2023	100.0/100.0	✓
Orchestrer le cloud avec Kubernetes	Atelier	6 févr. 2023	6 févr. 2023	100.0/100.0	✓
Orchestrer le cloud avec Kubernetes	Atelier	6 févr. 2023	6 févr. 2023	20.0/100.0	
Kubernetes Engine : Qwik Start	Atelier	6 févr. 2023	6 févr. 2023	100.0/100.0	✓
Présentation de Docker	Atelier	14 janv. 2023	14 janv. 2023	0.0/100.0	
Premiers pas avec Cloud Shell et gcloud	Atelier	13 janv. 2023	13 janv. 2023	100.0/100.0	✓
Créer une machine virtuelle	Atelier	9 janv. 2023	9 janv. 2023	100.0/100.0	✓
Présentation de Qwiklabs et de Google Cloud	Atelier	9 janv. 2023	9 janv. 2023	100.0/100.0	✓

A noter un problème de connexion internet durant la première tentative de l'atelier "Orchestrer le cloud avec Kubernetes".

## Richard FOUQUOIRE



Richard Fouquoire  
Date d'abonnement : 2023  
Votre profil n'est pas public ni accessible.  
[Rendre votre profil public](#)

≡v Parcours de formation    ≡ Activités    ⚙️ Badges

Cours Atelier Quête Quiz Jeu En cours Terminée

Activité	Type	Date de début	Date de fin	Score	Réussie
La journalisation dans le cloud est un moteur Kubernetes	Atelier	6 mars 2023	6 mars 2023	100.0/100.0	✓
Gérer l'état Terraform	Atelier	6 mars 2023	6 mars 2023	100.0/100.0	✓
Interact with Terraform Modules	Atelier	6 mars 2023	6 mars 2023	100.0/100.0	✓
Infrastructure as Code avec Terraform	Atelier	6 mars 2023	6 mars 2023	100.0/100.0	✓
Principes de base de Terraform	Atelier	6 mars 2023	6 mars 2023	100.0/100.0	✓
Creating Infrastructure on Google Cloud with Terraform	Quête	6 mars 2023	6 mars 2023		✓
Configuring Networks via gcloud	Atelier	13 févr. 2023	13 févr. 2023	100.0/100.0	✓
Orchestrer le cloud avec Kubernetes	Atelier	13 févr. 2023	13 févr. 2023	100.0/100.0	✓
Kubernetes Engine : Qwik Start	Atelier	13 févr. 2023	13 févr. 2023	100.0/100.0	✓
Getting Started with Cloud Shell and gcloud	Atelier	23 janv. 2023	23 janv. 2023	100.0/100.0	✓
Présentation de Docker	Atelier	23 janv. 2023	23 janv. 2023	0.0/100.0	
Créer une machine virtuelle	Atelier	9 janv. 2023	9 janv. 2023	100.0/100.0	✓
Présentation de Qwiklabs et de Google Cloud	Atelier	9 janv. 2023	9 janv. 2023	100.0/100.0	✓

## Emily RENARD



Emily Renard  
Date d'abonnement : 2023  
Votre profil n'est pas public ni accessible.  
[Rendre votre profil public](#)

Parcours de formation Activités Badges

Cours Atelier Quête Quiz Jeu En cours Terminée

Activité	Type	Date de début	Date de fin	Score	Réussie
La journalisation dans le cloud est un moteur Kubernetes	Atelier	il y a 39 minutes	il y a 0 minute	100.0/100.0	✓
Gérer l'état Terraform	Atelier	il y a 1 heure	il y a 46 minutes	100.0/100.0	✓
Interagir avec les modules Terraform	Atelier	il y a 1 heure	il y a 1 heure	100.0/100.0	✓
Infrastructure as Code avec Terraform	Atelier	il y a 2 heures	il y a 1 heure	90.0/100.0	
Principes de base de Terraform	Atelier	il y a 2 heures	il y a 2 heures	100.0/100.0	✓
Configuring Networks via gcloud	Atelier	il y a 2 heures	il y a 2 heures	100.0/100.0	✓
Orchestrer le cloud avec Kubernetes	Atelier	il y a 3 heures	il y a 3 heures	100.0/100.0	✓
Orchestrer le cloud avec Kubernetes	Atelier	il y a 4 heures	il y a 3 heures	25.0/100.0	
Kubernetes Engine : Qwik Start	Atelier	19 févr. 2023	19 févr. 2023	100.0/100.0	✓
Présentation de Docker	Atelier	28 janv. 2023	28 janv. 2023	0.0/100.0	
Getting Started with Cloud Shell and gcloud	Atelier	23 janv. 2023	23 janv. 2023	100.0/100.0	✓
Créer une machine virtuelle	Atelier	9 janv. 2023	9 janv. 2023	100.0/100.0	✓
Présentation de Qwiklabs et de Google Cloud	Atelier	9 janv. 2023	9 janv. 2023	100.0/100.0	✓

J'ai rencontré des problèmes de connexion et d'actualisation avec les ateliers qui ne sont pas à 100/100, mais cela n'a pas été bloquant pour aller jusqu'au bout.

# Introduction

Ce projet vient conclure notre unité 5I-SY3-VIRTUALISATION. Il a été réalisé en trinôme par notre groupe constitué de **Lucie DESRIAUX** , **Richard FOUQUOIRE** et **Emily RENARD** .

Le travail consiste au développement d'une application dans un langage choisi, tout en intégrant le maximum de technologies vues en cours telles que :

- Docker
- Kubernetes
- Architectures micro-services..

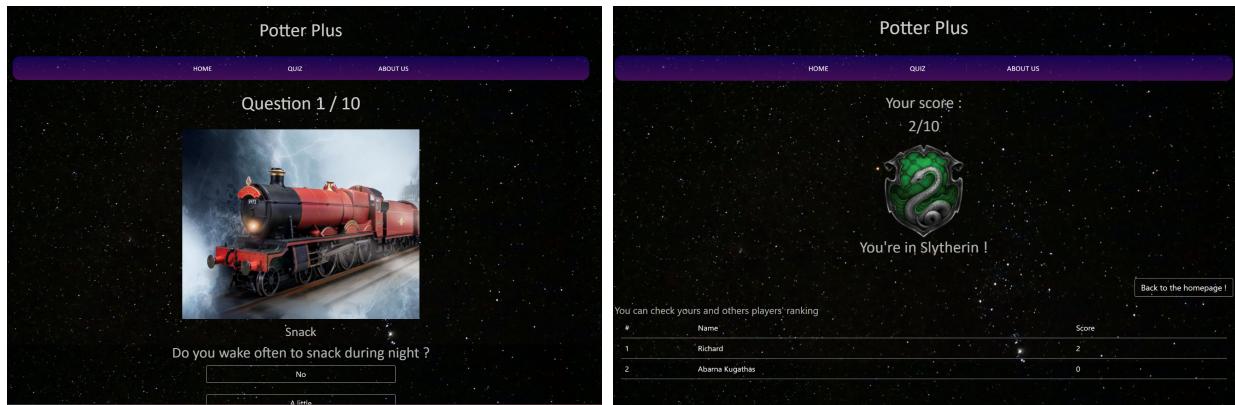
Nous avons pour notre part choisi de reprendre un ancien projet de E4 puis l'adapter afin de pouvoir le conteneuriser et le rendre scalable.

Très intéressante, cette unité nous a permis de mieux appréhender la virtualisation et la scalabilité de nos applications pour résoudre de nombreux problèmes avec des solutions modernes.

# Présentation du projet

Dans le cadre de la réalisation du projet, nous reprenons un ancien travail “*PotterPlus*” et l’adaptons pour sa virtualisation.

Il s’agit là d’une application simple, dans laquelle on retrouve un quiz sur le thème de l’œuvre *Harry Potter*. Elle contient un tableau de scores, un questionnaire à choix multiples, et d’autres pages statiques.



Les différents quiz, images, questions, participants et scores sont renseignés dans une base de données à laquelle notre application accède pour pouvoir fonctionner.

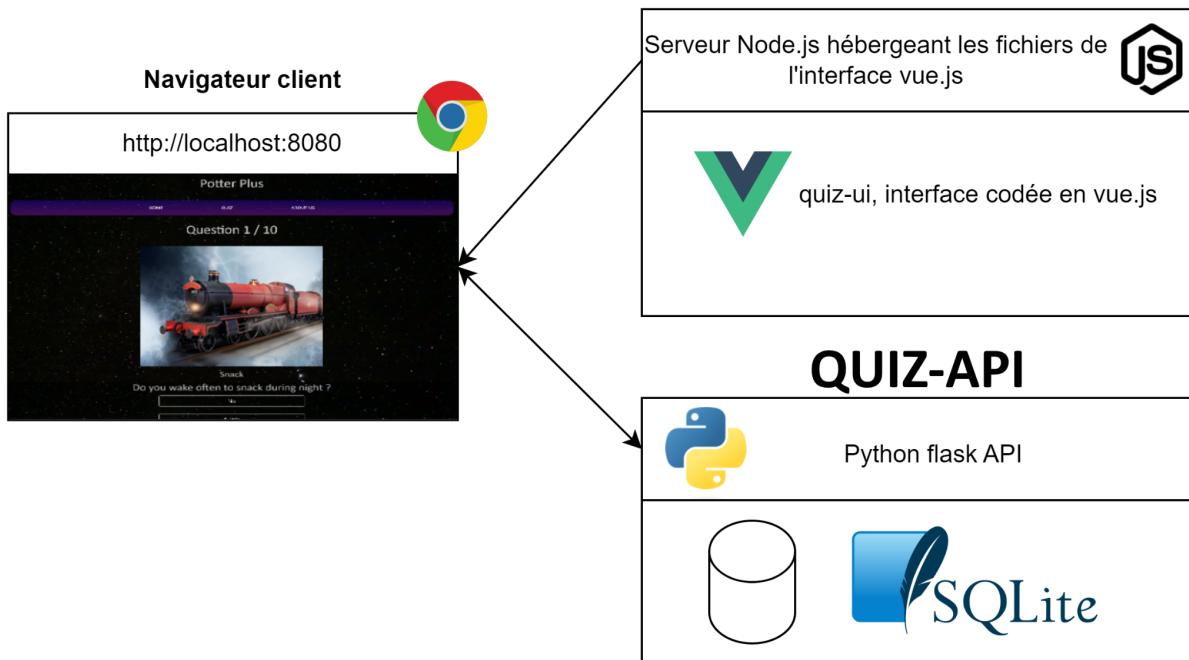
L’interface exécute du Vue.js pour le front-end, ce qui permet une UI qui s’exécute entièrement au sein du navigateur. Ainsi, les appels au back-end se font nécessairement via HTTP et une API qu’il faut mettre à disposition.

Nous soulignons le fait que l’interface de notre application s’exécute purement dans le navigateur du client, le javascript, via le framework Vue.js va générer dynamiquement l’interface, les boutons, les couleurs directement dans le navigateur. Contrairement à une interface générée par un serveur comme PHP, Tomcat, Springboot, etc. qui eux, vont pouvoir faire des accès en base de données avant de retourner le front généré, en évitant ainsi d’exposer la base de données via une API.

Le back-end, quant à lui, est propulsé par une API Python Flask dans laquelle nous avons lié des endpoints à des accès à notre base SQLite. Ce format très simple nous permet d’enregistrer notre base dans un fichier .db, ce qui nous évite d’exécuter un serveur pour son fonctionnement comme le nécessiteraient Oracle ou MySQL.

Voici un schéma simplifié résumant le fonctionnement de l’application et des échanges de flux pour son fonctionnement.

## QUIZ-UI

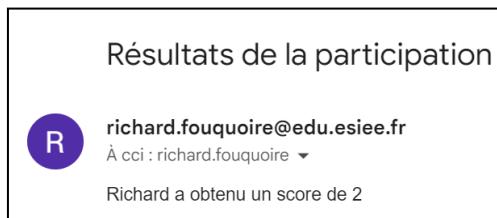


Comme demandé dans le cadre de ce projet, nous voulons faire communiquer une autre brique applicative avec l'application, en plus de la base de données. Voilà pourquoi nous mettons en place un service "mail-api" que nous avons créé nous même, qui est une API déclenchée par requête HTTP POST qui permet d'envoyer des mails avec un titre renseigné, un contenu, etc.

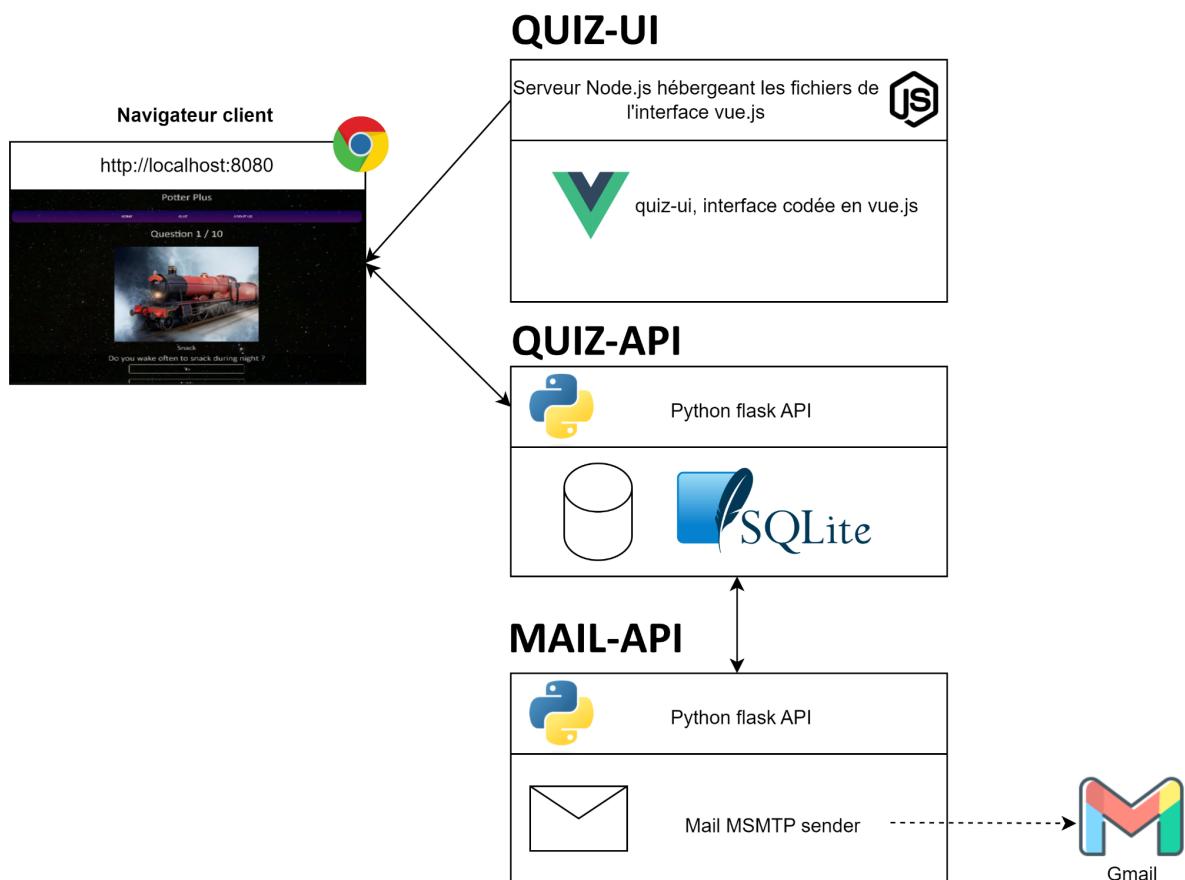
Techniquement, cette API est aussi propulsée par Python Flask, mais le mail est envoyé via l'application debian "msmtp" qui permet d'envoyer des mails via un serveur smtp externe, en ayant renseigné les paramètres de connexion au préalable. En l'occurrence, nous nous servirons ici du serveur smtp de Gmail, en utilisant l'adresse [richard.fouquoire@edu.esiee.fr](mailto:richard.fouquoire@edu.esiee.fr).

L'API met donc à disposition un endpoint : `http://localhost:5000/commands/mail`, qui, si l'on y accède via une requête POST, déclenchera un script shell utilisant msmtp pour envoyer le courriel.

Pour l'instant, nous souhaitons l'utiliser comme simple mail de notification envoyant le score d'un joueur à la fin de la participation d'un questionnaire. Bien sûr, on peut l'amener à évoluer plus tard pour ajouter plus de granularité dans nos informations.



Nous voici donc désormais avec l'infrastructure suivante :



# Mise à disposition des composants de l'application

## Création des images via Dockerfile

Nous disposons déjà de notre application, prête à fonctionner, il faut maintenant la conteneuriser en plus de faire des petits ajustements. Pour ce faire, nous créons un Dockerfile pour l'API et un autre pour l'UI.

Ces dockerfiles, renseignés à la racine de chaque composant (quiz-ui, quiz-api et mail-api), vont nous permettre de créer des images docker en installant les composants nécessaires à leur fonctionnement, exposer un port et renseigner les commandes à exécuter lors du lancement d'un conteneur créé à partir de l'image correspondante.

### Dockerfile quiz-api

```
FROM python:3.8-slim-buster

# make the 'app' folder the current working directory
WORKDIR /app

# copy requirements.txt and installing required libraries
COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt

# copy project files and folders to the current working directory (i.e. 'app' folder)
COPY . .

# opening port 5000
EXPOSE 5000
# running command at launch
CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0"]
```

Ici, nous choisissons une image de base `python:3.8-slim-buster` qui va contenir python et le nécessaire pour exécuter une application. Cependant, elle ne contient pas toutes les librairies dont dépend le projet comme Flask.

Nous nous mettons dans le dossier '/app' et installons toutes les dépendances nécessaires via le fichier requirements.txt avant de copier l'entièreté du projet dans ce même dossier '/app'.

Nous nous assurons d'exposer le bon port, ici 5000, typique pour les services web propulsés par Flask. Nous renseignons ensuite la commande à exécuter lors du lancement d'un conteneur issu de cette image, ici, le lancement de l'application Flask.

#### Dockerfile mail-api

```
FROM python:3.8-slim-buster

WORKDIR /app
RUN apt-get update && apt-get install msmtprc -y && \
    rm -rf /var/lib/apt/lists/*

COPY msmtprc /etc/msmtprc
RUN chmod 600 /etc/msmtprc

COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt

COPY app.py /app/app.py
COPY mailer.sh /app/mailer.sh

EXPOSE 5000
CMD [ "python3", "-m", "flask", "run", "--host=0.0.0.0" ]
```

Nous reprenons quasiment le même dockerfile avec la même séquence de traitements, hormis le fait qu'on installe le module msmtprc, qu'on y ajoute notre fichier de configuration /etc/msmtprc, et l'ajout de notre script 'mailer.sh' qui contient le code d'envoi de mail via shell en utilisant le service msmtprc, déclenché par l'API Flask.

#### Dockerfile quiz-ui

```
FROM node:lts-alpine

# install simple http server for serving static content
RUN npm install -g http-server

# make the 'app' folder the current working directory
WORKDIR /app

# copy both 'package.json' and 'package-lock.json' (if available)
COPY package*.json ./

# install project dependencies
RUN npm install
```

```
# copy project files and folders to the current working directory (i.e. 'app' folder)
COPY . .

# build app for production with minification
RUN npm run build

# opening port 8080
EXPOSE 8080
# running command at launch
CMD [ "http-server", "dist" ]
```

Nous choisissons ici un serveur simple nodeJS pour pouvoir héberger notre UI. Nous choisissons ainsi l'image de base `node:lts-alpine`.

De la même manière que pour notre API, on se met sur le dossier '/app' dans lequel on va installer toutes nos dépendances et paramètres nécessaires en copiant les fichiers package.json et en exécutant la commande `npm install`.

On build ensuite notre application Vue.js à l'aide de la commande `npm run build`.

Notre UI est désormais prête et nous pouvons désormais l'héberger via le serveur nodeJS. Nous exposons ainsi le port 8080 et renseignons la commande à exécuter lors du lancement d'un conteneur issu de cette image, de manière à faire tourner le serveur et rendre disponible notre interface.

Nous créons enfin nos images à partir des Dockerfile, en se mettant dans le même dossier courant du dockerfile et de l'application renseignée.

#### **Création de l'image *quiz-api* à sa version *latest***

```
docker build -t quiz-api:latest .
```

#### **Création de l'image *mail-api* à sa version *latest***

```
docker build -t mail-api:latest .
```

#### **Création de l'image *quiz-ui* à sa version *latest***

```
docker build -t quiz-ui:latest .
```

# Envoi des images sur la registry Docker Hub

On tag ensuite ces images nouvellement créées en vue d'un futur push sur une registry docker.

**Tag de l'image *quiz-api* pour un futur envoi sur *richardioi/quiz-api:latest***

```
docker tag quiz-api:latest richardioi/quiz-api:latest
```

**Tag de l'image *mail-api* pour un futur envoi sur *richardioi/mail-api:latest***

```
docker tag mail-api:latest richardioi/mail-api:latest
```

**Tag l'image *quiz-ui* pour un futur envoi sur *richardioi/quiz-ui:latest***

```
docker tag quiz-ui:latest richardioi/quiz-ui:latest
```

On push ensuite nos nouvelles images nouvellement créées et tagués sur la registry Docker Hub, au compte richardioi. Nous avons préalablement créé un compte "richardioi" sur Docker Hub, sur lequel on va pouvoir envoyer et extraire nos images.

**Push de l'image *quiz-api* sur *richardioi/quiz-api:latest***

```
docker push richardioi/quiz-api:latest
```

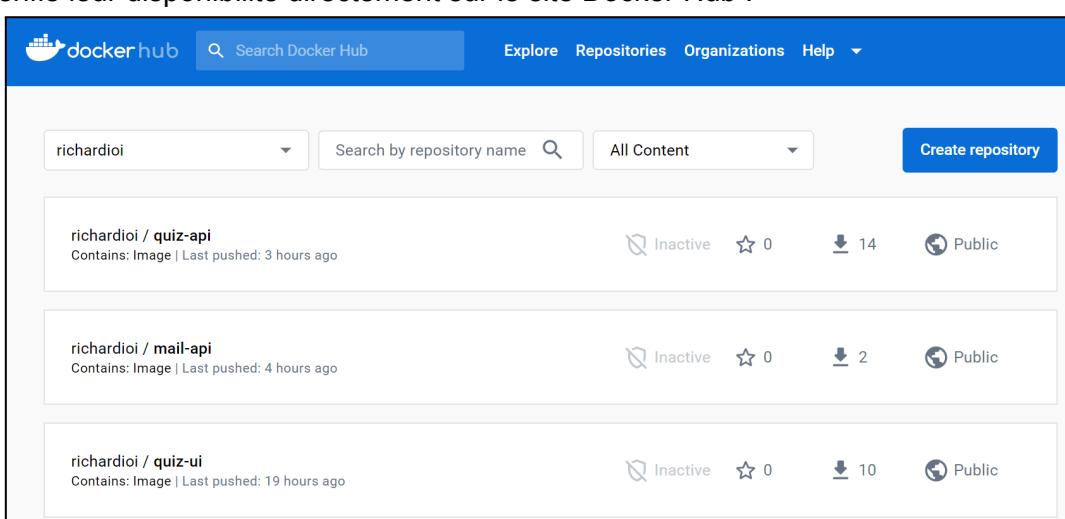
**Push de l'image *mail-api* sur *richardioi/mail-api:latest***

```
docker push richardioi/mail-api:latest
```

**Push de l'image *quiz-ui* sur *richardioi/quiz-ui:latest***

```
docker push richardioi/quiz-ui:latest
```

On vérifie leur disponibilité directement sur le site Docker Hub :



Si l'on s'intéresse à chacune des pages des images hébergées sur la registry, on pourrait renseigner plus d'informations via un README pour expliquer le fonctionnement de l'image et la bonne exécution d'un conteneur.

The screenshot shows the Docker Hub interface for a repository named 'richardioi / quiz-api'. The top navigation bar includes 'Explore', 'Repositories', 'Organizations', 'Help', 'Upgrade', and a user profile icon for 'richardioi'. The repository path 'richardioi / quiz-api / General' is visible in the breadcrumb menu. A message indicates 'Using 0 of 1 private repositories. [Get more](#)'. Below the navigation, there are tabs for 'General', 'Tags', 'Builds', 'Collaborators', 'Webhooks', and 'Settings', with 'General' being the active tab.

In the 'General' section, there is a note to 'Add a short description for this repository' with a link to 'Update'. The description area states: 'This repository does not have a description' with an edit icon. A timestamp shows 'Last pushed: 3 hours ago'.

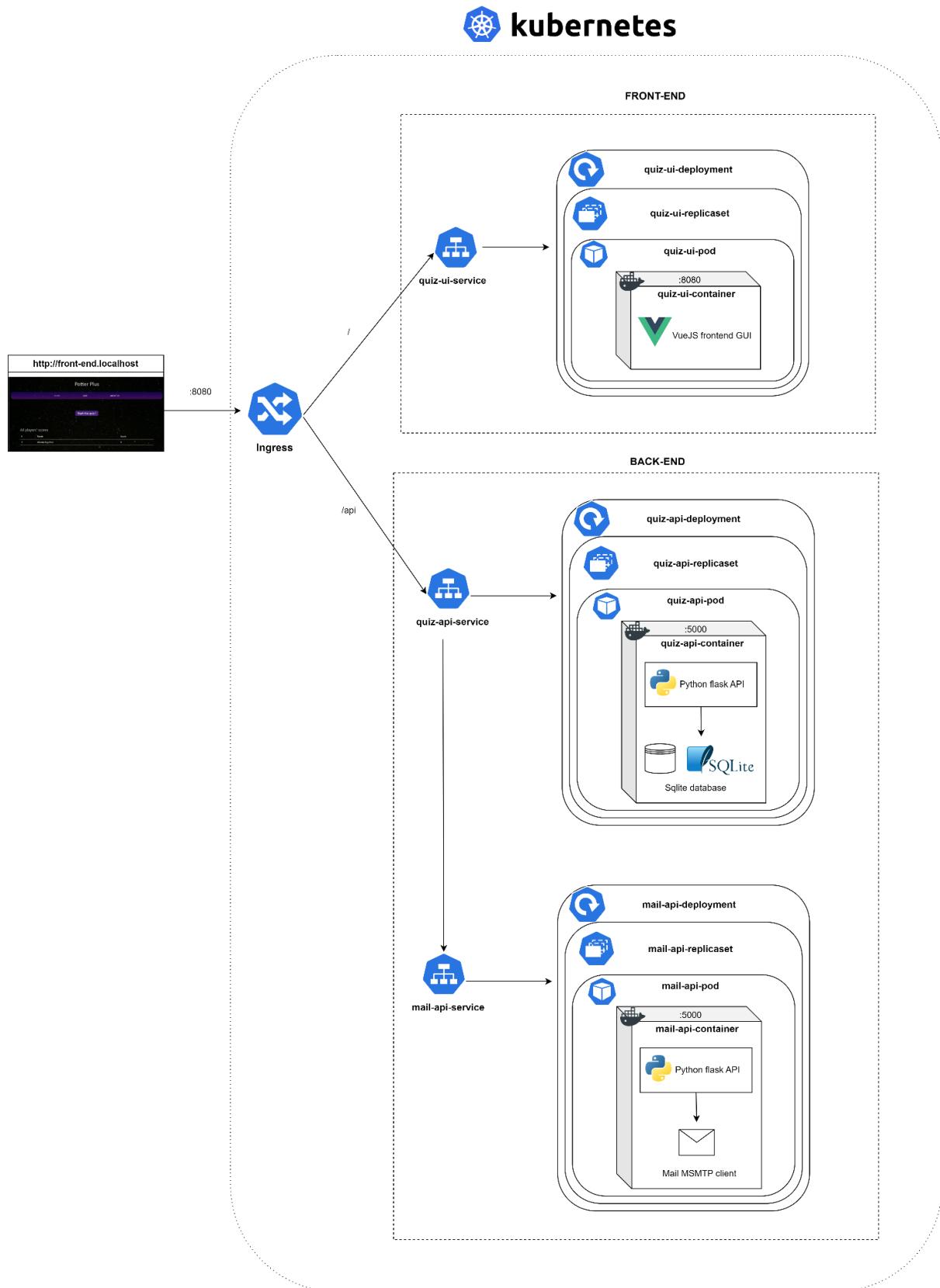
The 'Docker commands' section contains the command 'docker push richardioi/quiz-api:tagname'.

The 'Tags' section lists one tag: 'latest' (OS: Alpine, Type: Image, Pulled: --, Pushed: 3 hours ago). It also includes a link to 'Go to Advanced Image Management'.

The 'Automated Builds' section explains how to connect GitHub or Bitbucket for automatic builds and provides a link to 'Read more about automated builds'.

Nos images, étant rendues à disposition sur la registry, peuvent désormais être automatiquement téléchargées dans le cadre de l'utilisation de Kubernetes ou autre.

# Mise en place du cluster kubernetes



Nous souhaitons obtenir la configuration kubernetes ci-dessus, en mettant en place :

- Des *deployments* pour chacun de nos composants (mail-api-deployment, quiz-api-deployment, quiz-ui-deployment) avec des *replicaset* et *pods* générés automatiquement, d'où seront exécutées nos images Docker précédemment envoyées sur la registry Docker Hub.
- Des *services*, permettant d'exposer nos composants et/ou de les faire communiquer entre eux, via un service DNS interne à Kubernetes.
- Une gateway *Ingress* qui va nous permettre de gérer l'accès externe aux services dans notre cluster kubernetes, en re-routant notre trafic HTTP sur l'adresse <http://front-end.localhost>

Grâce à cette configuration, notre frontal sera le gateway Ingress mettant à disposition l'interface UI de notre application et son API, leurs deux services ne communiquant pas directement, comme vu précédemment.

En revanche, nous ferons communiquer les deux services quiz-api-service et mail-api-service pour l'envoi de mails. Ainsi, le composant mail-api n'est pas exposé inutilement et reste en interne pour des raisons de sécurité.

Afin de faciliter la configuration de nos ressources Kubernetes, nous avons tout mis sur des fichiers *yaml*, ce qui est pratique pour une reconfiguration prochaine, notamment sur un cloud provider. Pour l'instant, nous mettons en place cette configuration via notre service local *minikube*, que l'on démarre via la commande suivante :

```
minikube start
```

## Configuration des *deployments*

On met en place une configuration quasi identique pour nos trois composants. Chacun de ces *deployments* est caractérisé par des *matchLabels* où l'on renseigne le nom du composant, de manière à ce qu'ils puissent être retrouvés lors de la configuration de leurs services correspondants.

Ils ne disposent que d'un seul replica chacun, mais nous pourrions augmenter leur nombre afin de prévoir du load balancing par exemple.

Chacun extrait son image Docker correspondant à sa dernière version taguée “latest”, préalablement envoyée sur la registry Docker Hub avec l'*imagePullPolicy* à “Always”, de manière à toujours télécharger la dernière version de notre image lors de la recréation du pod.

**quiz-api-deployment.yaml :**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: quiz-api-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: quiz-api
  template:
    metadata:
      labels:
        app: quiz-api
    spec:
      containers:
        - name: quiz-api
          image: richardioi/quiz-api:latest
          imagePullPolicy: Always
```

**mail-api-deployment.yaml :**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mail-api-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mail-api
  template:
    metadata:
      labels:
        app: mail-api
    spec:
      containers:
        - name: mail-api
          image: richardioi/mail-api:latest
          imagePullPolicy: Always
```

### **quiz-ui-deployment.yaml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: quiz-ui-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: quiz-ui
  template:
    metadata:
      labels:
        app: quiz-ui
  spec:
    containers:
      - name: quiz-ui
        image: richardioi/quiz-ui:latest
        imagePullPolicy: Always
```

On crée ensuite nos *deployments* en exécutant les commandes suivantes :

```
kubectl apply -f ./quiz-api-deployment.yaml
kubectl apply -f ./mail-api-deployment.yaml
kubectl apply -f ./quiz-ui-deployment.yaml
```

## Configuration des *services*

On crée à nouveau nos fichiers yaml pour la configuration de nos services. Le champ *selector.app* de chacun va permettre de “matcher” avec les *matchLabels* définis précédemment dans nos *deployments*, on lie donc chacun de ces *services* à son *deployment* correspondant.

On vient aussi renseigner les ports exposés par le conteneur du pod de déploiement (*targetPort*), reroutés par les ports que l'on veut exposer du service (*port*).

### **quiz-api-service.yaml :**

```
apiVersion: v1
kind: Service
metadata:
  name: quiz-api-service
```

```
spec:  
  ports:  
    - name: http  
      targetPort: 5000 # This is the port exposed by the container  
      port: 8080 # This is the port exposed by the service  
  selector:  
    app: quiz-api
```

#### mail-api-service.yaml :

```
apiVersion: v1  
kind: Service  
metadata:  
  name: mail-api-service  
spec:  
  ports:  
    - name: http  
      targetPort: 5000 # This is the port exposed by the container  
      port: 5000 # This is the port exposed by the service  
  selector:  
    app: mail-api
```

#### quiz-ui-service.yaml :

```
apiVersion: v1  
kind: Service  
metadata:  
  name: quiz-ui-service  
spec:  
  ports:  
    - name: http  
      targetPort: 8080 # This is the port exposed by the container  
      port: 8080 # This is the port exposed by the service  
  selector:  
    app: quiz-ui
```

Comme précédemment, on crée nos *services* en exécutant les commandes suivantes :

```
kubectl apply -f ./quiz-api-service.yaml  
kubectl apply -f ./mail-api-service.yaml  
kubectl apply -f ./quiz-ui-service.yaml
```

On teste le bon fonctionnement de nos services et l'accès à nos pods via la commande suivante, en accédant aux IHM en local.

```
minikube service quiz-ui-service --url
```

## Communication entre services

quiz-api <-> mail-api

Nous avons dû apporter une modification à notre composant **quiz-api** pour qu'il ne fasse plus l'appel à **mail-api** sur l'adresse localhost:5000 mais sur l'adresse `http://mail-api-service.default.svc.cluster.local:5000`

En effet, même si les deux composants ne sont pas sous un même service, leurs deux services peuvent continuer à communiquer grâce à la résolution DNS interne au cluster Kubernetes fournie par `kube-dns`.

- Ainsi, l'adresse d'un service aux yeux des autres services devient : `<service_name>.<name_space>.svc.<cluster_name>`
- D'où notre lien suivant : `mail-api-service.default.svc.cluster.local`

quiz-ui <-> mail-api

De la même manière, **quiz-ui** et **quiz-api** peuvent communiquer entre eux. Cependant, comme expliqué précédemment, tous les accès en base de données se font via appel à **quiz-api** depuis le navigateur client, ce même navigateur client ne peut pas résoudre l'adresse `quiz-api-service.default.svc.cluster.local`. De plus, notre configuration en tant que telle fait que l'adresse IP exposée par le service Kubernetes est aléatoire, et ne peut pas être mise en dur dans le code.

Voilà pourquoi, pour l'instant, notre application s'affiche bien au niveau de l'UI mais n'est pas fonctionnelle car elle n'a pas encore accès aux questions, tableaux de scores, etc. fournis par l'API. Ceci va être très vite résolu grâce à l'implémentation du gateway Ingress.

## Mise en place du gateway Ingress

On commence par activer l'addon Ingress au sein de minikube grâce à la commande suivante :

```
minikube addons enable ingress
```

Puis on ajoute un autre addon pour permettre la création d'un tunnel minikube et la résolution DNS en local :

```
minikube addons enable ingress-dns
```

On rédige ensuite sa configuration dans un fichier yaml.

### front-end-ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-quiz-app
spec:
  rules:
  - host: "front-end.localhost"
    http:
      paths:
      - path: "/"
        pathType: Prefix
        backend:
          service:
            name: quiz-ui-service
            port:
              number: 8080
      - path: "/api"
        pathType: Prefix
        backend:
          service:
            name: quiz-api-service
            port:
              number: 8080
```

On crée ensuite le gateway ingress grâce au fichier yaml via la commande suivante :

```
kubectl apply -f ./quiz-api-service.yaml
```

Cette configuration nous permet de rerouter notre service **quiz-ui-service** sur le path <http://front-end.localhost/> et **quiz-api-service** <http://front-end.localhost/api> tous les deux sur le port 8080.

On remet à jour le code de notre UI de manière à ce qu'il ne tape plus sur <http://localhost:5000> pour avoir accès à l'api mais au chemin relatif local /api, du fait que l'UI et l'API sont maintenant sur le même nom de domaine front-end.localhost.

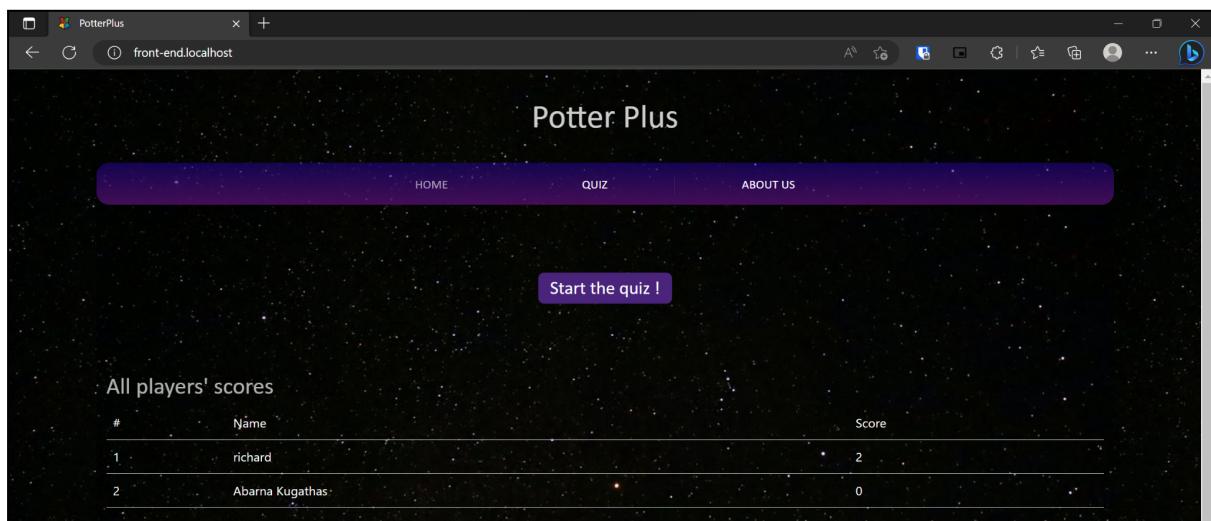
Pour faire fonctionner cette résolution DNS en local, on vient modifier le fichier hosts de notre machine à l'adresse : C:\windows\system32\drivers\etc\hosts puis on ajoute la ligne

suivante : 127.0.0.1 front-end.localhost

On allume ensuite le tunnel pour tester l'accès en local via la commande

```
minikube tunnel
```

On teste alors l'accès à l'interface, voilà qu'elle est accessible à la bonne adresse, fonctionnelle, avec les appels à l'API fonctionnels !

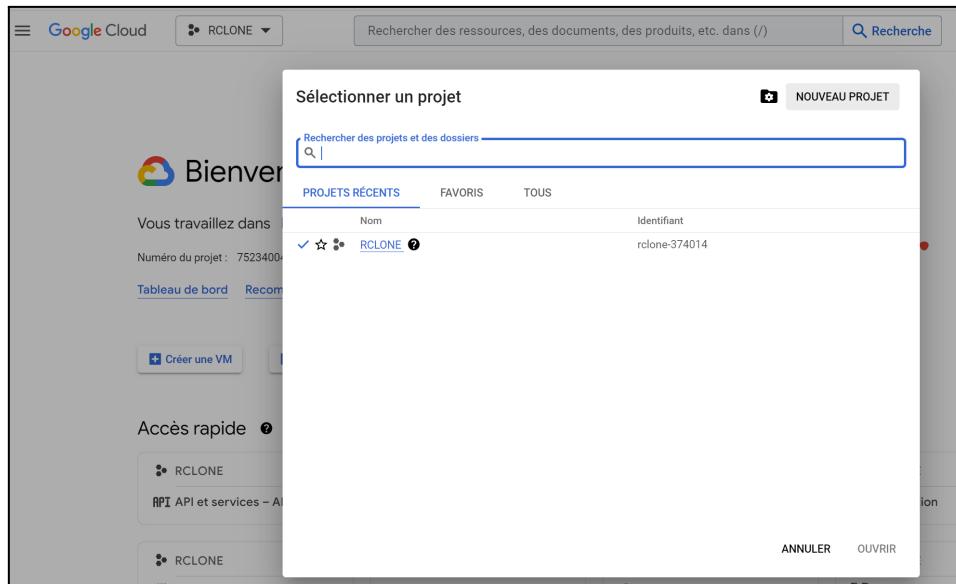


# Déploiement sur un cloud provider

## Configuration d'un nouveau projet sur Google Cloud

Nous choisissons de déployer notre application grâce aux solutions proposées par Google Cloud, ayant été initiés à la plateforme grâce aux Qwiklabs.

On se dirige sans plus tarder sur console.google.cloud.com et on clique sur “sélectionner un projet” puis “nouveau projet”.



On renseigne le nom PotterPlus puis nous créons le projet.

Il vous reste 11 projects dans votre quota. Demandez une augmentation ou supprimez des projets. [En savoir plus](#)

[MANAGE QUOTAS](#)

Nom du projet \*  
PotterPlus

ID du projet : potterplus. Vous ne pourrez pas le modifier par la suite. [MODIFIER](#)

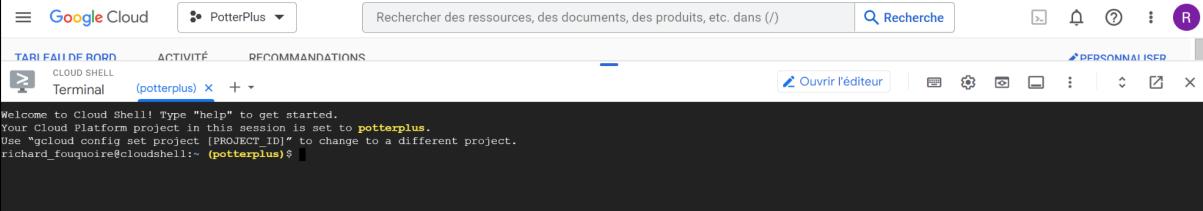
Zone \*  
Aucune organisation [PARCOURIR](#)

Organisation ou dossier parent

**CRÉER**   **ANNULER**

On se dirige sur l'accueil et on ouvre le terminal *Cloud Shell* afin de débuter la configuration de notre cluster kubernetes.

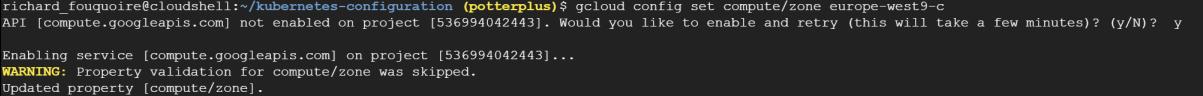
Nous allons pouvoir ainsi appliquer les mêmes étapes que précédemment pour configurer notre solution, grâce aux fichiers .yaml que nous avons défini précédemment.



The screenshot shows the Google Cloud Platform Cloud Shell interface. At the top, there's a navigation bar with 'Google Cloud' and a dropdown for 'PotterPlus'. A search bar says 'Rechercher des ressources, des documents, des produits, etc. dans ()'. On the right, there are several icons for account management and help. Below the bar, the main area has tabs for 'TARI FAII DE RORD', 'ACTIVITÉ', and 'RECOMMANDATIONS'. A 'Terminal' tab is active, showing the command line. The terminal output is as follows:

```
Welcome to Cloud Shell! Type "help" to get started.  
Your Cloud Platform project in this session is set to potterplus.  
Use "gcloud config set project [PROJECT_ID]" to change to a different project.  
richard_fouquoire@cloudshell:~ (potterplus)$
```

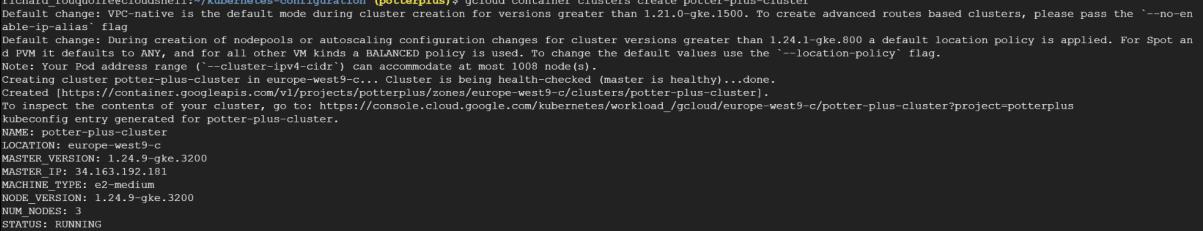
On définit la zone de calcul, ici on choisit “europe-west9-c” qui correspond à Paris.



```
richard_fouquoire@cloudshell:~/kubernetes-configuration (potterplus)$ gcloud config set compute/zone europe-west9-c  
API [compute.googleapis.com] not enabled on project [536994042443]. Would you like to enable and retry (this will take a few minutes)? (y/N)? y  
  
Enabling service [compute.googleapis.com] on project [536994042443]...  
WARNING: Property validation for compute/zone was skipped.  
Updated property [compute/zone].
```

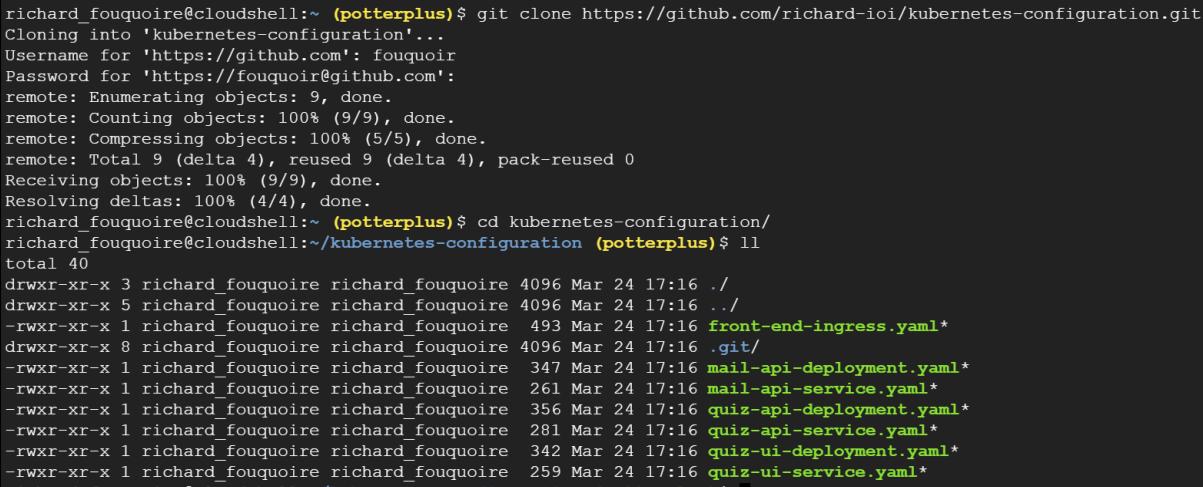
## Configuration du cluster kubernetes

On crée un cluster kubernetes que l'on appelle “potter-plus-cluster”



```
richard_fouquoire@cloudshell:~/kubernetes-configuration (potterplus)$ gcloud container clusters create potter-plus-cluster  
Default change: VPC-native is the default mode during cluster creation for versions greater than 1.21.0-gke.1500. To create advanced routes based clusters, please pass the '--no-enable-ip-alias' flag  
Default change: During creation of nodepools or autoscaling configuration changes for cluster versions greater than 1.24.1-gke.800 a default location policy is applied. For Spot an IP range is used. ANV and for all other VM kinds a BALANCED policy is used. To change the default values use the '--location-policy' flag.  
Note: Your Pod address range ('--cluster-ipv4-cidr') can accommodate at most 1008 node(s).  
Creating cluster potter-plus-cluster in europe-west9-c. Cluster is being health-checked (master is healthy)...done.  
Created [https://container.googleapis.com/v1/projects/potterplus/zones/europe-west9-c/clusters/potter-plus-cluster].  
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload/gcloud/europe-west9-c/potter-plus-cluster?project=potterplus  
kubeconfig entry generated for potter-plus-cluster.  
NAME: potter-plus-cluster  
LOCATION: europe-west9-c  
MASTER_VERSION: 1.24.9-gke.3200  
MASTER_IP: 34.163.192.181  
MACHINE_TYPE: e2-medium  
NODE_VERSION: 1.24.9-gke.3200  
NUM_NODES: 3  
STATUS: RUNNING
```

On vient cloner un dépôt Github sur lequel nous avions préalablement déposé tous nos fichiers .yaml de déploiements, services et contrôleur Ingress.



```
richard_fouquoire@cloudshell:~ (potterplus)$ git clone https://github.com/richard-ioi/kubernetes-configuration.git  
Cloning into 'kubernetes-configuration'...  
Username for 'https://github.com': fouquoir  
Password for 'https://fouquoir@github.com':  
remote: Enumerating objects: 9, done.  
remote: Counting objects: 100% (9/9), done.  
remote: Compressing objects: 100% (5/5), done.  
remote: Total 9 (delta 4), reused 9 (delta 4), pack-reused 0  
Receiving objects: 100% (9/9), done.  
Resolving deltas: 100% (4/4), done.  
richard_fouquoire@cloudshell:~ (potterplus)$ cd kubernetes-configuration/  
richard_fouquoire@cloudshell:~/kubernetes-configuration (potterplus)$ ll  
total 40  
drwxr-xr-x 3 richard_fouquoire richard_fouquoire 4096 Mar 24 17:16 ./  
drwxr-xr-x 5 richard_fouquoire richard_fouquoire 4096 Mar 24 17:16 ../  
-rwxr-xr-x 1 richard_fouquoire richard_fouquoire 493 Mar 24 17:16 front-end-ingress.yaml*  
drwxr-xr-x 8 richard_fouquoire richard_fouquoire 4096 Mar 24 17:16 .git/  
-rwxr-xr-x 1 richard_fouquoire richard_fouquoire 347 Mar 24 17:16 mail-api-deployment.yaml*  
-rwxr-xr-x 1 richard_fouquoire richard_fouquoire 261 Mar 24 17:16 mail-api-service.yaml*  
-rwxr-xr-x 1 richard_fouquoire richard_fouquoire 356 Mar 24 17:16 quiz-api-deployment.yaml*  
-rwxr-xr-x 1 richard_fouquoire richard_fouquoire 281 Mar 24 17:16 quiz-api-service.yaml*  
-rwxr-xr-x 1 richard_fouquoire richard_fouquoire 342 Mar 24 17:16 quiz-ui-deployment.yaml*  
-rwxr-xr-x 1 richard_fouquoire richard_fouquoire 259 Mar 24 17:16 quiz-ui-service.yaml*
```

Comme précédemment, on crée les ressources nécessaires via les fichiers .yaml.

```
richard_fouquoire@cloudshell:~/kubernetes-configuration (potterplus)$ kubectl apply -f ./quiz-api-deployment.yaml
kubectl apply -f ./mail-api-deployment.yaml
kubectl apply -f ./quiz-ui-deployment.yaml
deployment.apps/quiz-api-deployment created
deployment.apps/mail-api-deployment created
deployment.apps/quiz-ui-deployment created
richard_fouquoire@cloudshell:~/kubernetes-configuration (potterplus)$ kubectl apply -f ./quiz-api-service.yaml
kubectl apply -f ./mail-api-service.yaml
kubectl apply -f ./quiz-ui-service.yaml
service/quiz-api-service created
service/mail-api-service created
service/quiz-ui-service created
```

On vérifie la création et le bon fonctionnement de nos ressources grâce à la commande suivante:

```
kubectl get all
```

```
richard_fouquoire@cloudshell:~/kubernetes-configuration (potterplus)$ kubectl get all
NAME                                         READY   STATUS    RESTARTS   AGE
pod/mail-api-deployment-78ffbc8f6-sz148   1/1     Running   0          3m48s
pod/quiz-api-deployment-7bc5db94b6-f56x5   1/1     Running   0          3m49s
pod/quiz-ui-deployment-6dbff584fb-rxlcd   1/1     Running   0          3m48s

NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP  10.48.0.1   <none>        443/TCP   6m23s
service/mail-api-service   ClusterIP  10.48.4.107  <none>        5000/TCP   3m30s
service/quiz-api-service   ClusterIP  10.48.4.71   <none>        8080/TCP   3m31s
service/quiz-ui-service   ClusterIP  10.48.2.135  <none>        8080/TCP   3m30s

NAME           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mail-api-deployment   1/1     1           1           3m48s
deployment.apps/quiz-api-deployment   1/1     1           1           3m49s
deployment.apps/quiz-ui-deployment   1/1     1           1           3m48s

NAME           DESIRED   CURRENT   READY   AGE
replicaset.apps/mail-api-deployment-78ffbc8f6  1         1         1         3m48s
replicaset.apps/quiz-api-deployment-7bc5db94b6  1         1         1         3m49s
replicaset.apps/quiz-ui-deployment-6dbff584fb   1         1         1         3m48s
```

## Re-configuration du gateway ingress

On modifie légèrement notre configuration ingress située dans le fichier “front-end-ingress.yaml”.

Nous voulons simplement exposer notre frontal ingress sur une adresse IP statique, et rajouter plus tard une résolution DNS sur cette même IP pour que notre IHM soit accessible dans le monde entier via un nom de domaine.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-quiz-app
  annotations:
    kubernetes.io/ingress.global-static-ip-name: "web-static-ip"
```

```

spec:
  rules:
    #- host: "front-end.localhost"
    - http:
        paths:
          - path: "/"
            pathType: Prefix
            backend:
              service:
                name: quiz-ui-service
                port:
                  number: 8080
          - path: "/api"
            pathType: Prefix
            backend:
              service:
                name: quiz-api-service
                port:
                  number: 8080

```

Pour ce faire, nous avons mis la ligne “host” en commentaire et rajouté les lignes :

```

annotations:
  kubernetes.io/ingress.global-static-ip-name: "web-static-ip"

```

Ces lignes indiquent à kubernetes d’utiliser une adresse ip statique préalablement créée en amont de nom “web-static-ip”. Nous l’avons créé de la manière suivante :

```

richard_fouquaire@cloudshell:~/kubernetes-configuration (potterplus)$ gcloud compute addresses create web-static-ip --global
Created [https://www.googleapis.com/compute/v1/projects/potterplus/global/addresses/web-static-ip].

```

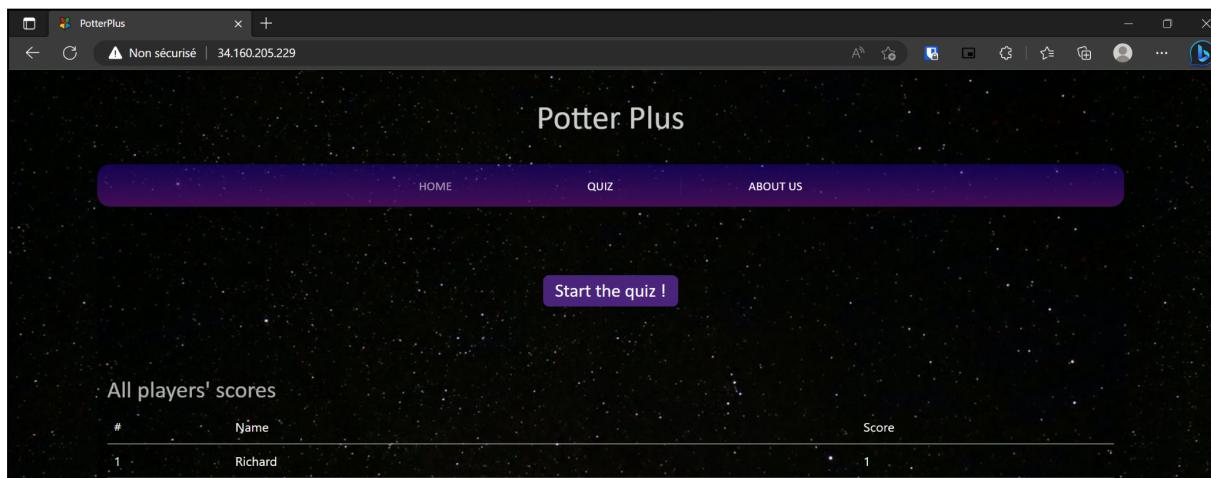
On crée enfin le gateway ingress puis vérifions son bon fonctionnement :

```

richard_fouquaire@cloudshell:~/kubernetes-configuration (potterplus)$ kubectl describe ingress
Name:           ingress-quiz-app
Labels:         <none>
Namespace:      default
Address:        34.160.205.229
Ingress Class:  <none>
Default backend: <default>
Rules:
  Host      Path  Backends
  ----      ---   -----
  *
    /          quiz-ui-service:8080 (10.44.0.4:8080)
    /api/*     quiz-api-service:8080 (10.44.1.8:5000)
Annotations:  ingress.kubernetes.io/backends:
               {"k8s-be-30172-4143fb497ad326df":"HEALTHY","k8s1-4143fb49-default-quiz-api-service-8080-61075363":"HEALTHY","k8s1-4143fb49-default-quiz-u...
               ingress.kubernetes.io/forwarding-rule: k8s2-fr-erclxbct-default-ingress-quiz-app-kyklxf52
               ingress.kubernetes.io/target-proxy: k8s2-tp-erclxbct-default-ingress-quiz-app-kyklxf52
               ingress.kubernetes.io/url-map: k8s2-um-erclxbct-default-ingress-quiz-app-kyklxf52
               kubernetes.io/ingress.global-static-ip-name: web-static-ip
Events:
  Type    Reason     Age             From                      Message
  ----    ----     --             --                      --
  Normal  Sync      39m            loadbalancer-controller  UrlMap "k8s2-um-erclxbct-default-ingress-quiz-app-kyklxf52" created
  Normal  Sync      39m            loadbalancer-controller  TargetProxy "k8s2-tp-erclxbct-default-ingress-quiz-app-kyklxf52" created
  Normal  Sync      39m            loadbalancer-controller  ForwardingRule "k8s2-fr-erclxbct-default-ingress-quiz-app-kyklxf52" created
  Normal  TPCChanged 39m            loadbalancer-controller  IP is now 34.160.205.229
  Normal  Sync      5m36s (x1 over 42m) loadbalancer-controller  Scheduled for sync

```

Notre application PotterPlus est désormais accessible n'importe où sur l'adresse IP publique créée 34.169.205.229 :



## Attribution d'un nom de domaine et configuration DNS

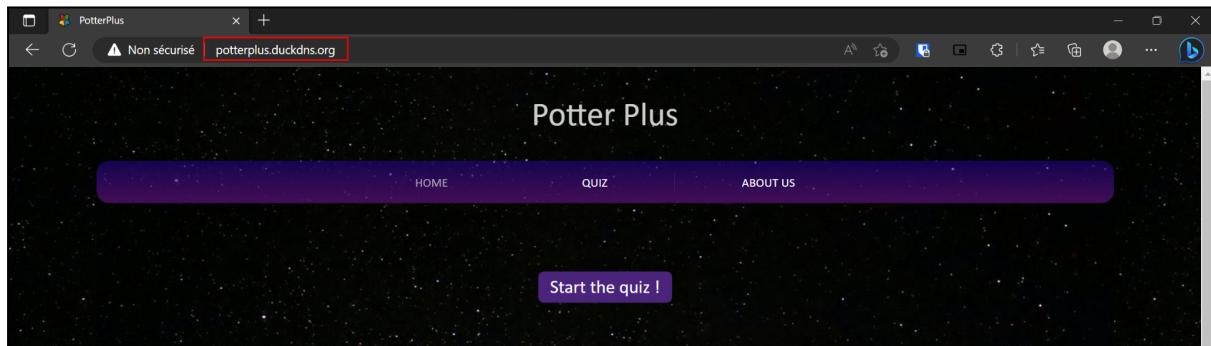
De manière à rendre notre application plus accessible, nous avons décidé d'aller plus loin et d'attribuer un nom de domaine simple à retenir. Pour cela, nous utilisons le service gratuit "Duck DNS" qui propose une résolution DNS avec l'attribution d'un nom de domaine avec suffixe \*.duckdns.org.

Nous aurions pu acheter un vrai nom de domaine et faire une vraie résolution DNS digne de ce nom mais nous nous sommes dit que ça n'était pas rentable dans le cadre de ce projet scolaire.

Nous nous rendons sur le site duckdns.org, créons un compte puis attribuons un nom de domaine à notre adresse IP statique comme suit :

domain	current ip	ipv6	changed
potterplus	34.160.205.229	update ip	0 seconds ago

Notre application est désormais accessible mondialement via le nom de domaine potterplus.duckdns.org



# Conclusion

Ce projet, très intéressant, nous a permis de ré-aborder un ancien projet mais sous une vision beaucoup plus tournée sur l'infrastructure micro-services. Cette mise en place permet de régler de nombreux problèmes futurs tels que la disponibilité de notre application, la scalabilité, la sécurité ou bien l'apport de nouvelles modifications de manière agile.

A titre personnel, nous avons beaucoup aimé le réaliser sur un de nos propres sujets, cela permet un apprentissage accru et réellement comprendre tout ce qui se passe de A à Z.