

VIRTUALISATION

BAUCHER Quentin
BISSENNIER Daphné
JOUNIOT Constantin

E5FI

ESIÉE
PARIS

Sommaire

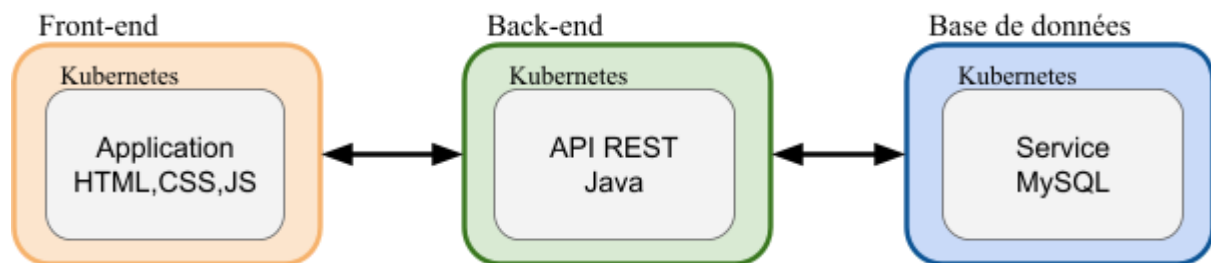
Projet	3
Hiérarchie du projet	4
Partie front-end	5
Partie base de données - Création et Virtualisation	7
Partie back-end	8
Virtualisation	9
Qwilabs	11

Projet

Pour ce projet réalisé à 3 nous avons décidé de faire un formulaire. Celui-ci est constitué :

- d'un back-end en JAVA avec le framework Spring reposant sur le principe des API REST,
- d'un front-end en HTML/CSS/JS,
- d'une base de données SQL.

On peut schématiser le lien entre chaque élément du projet ainsi :

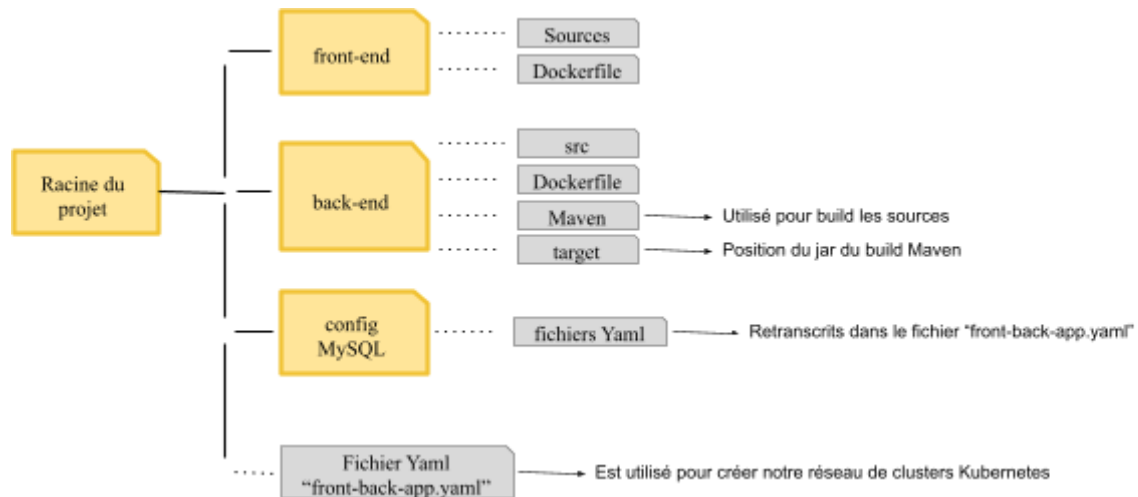


La répartition de travail dans notre groupe s'est déroulée ainsi :

- BAUCHER Quentin s'est occupé de mettre en place la base de données MySQL, comprendre le fonctionnement des fichiers YAML pour ensuite aider sur la virtualisation des deux autres parties.
- BISSONNIER Daphné s'est occupée de développer la partie Front-end, et de la déployer sur un container Kubernetes.
- JOUNIOT Constantin s'est occupé de développer la partie Back-end, et de la déployer sur un container Kubernetes.

Hiérarchie du projet

Notre projet peut être présenté sous l'arbre suivant :



Le dossier *back-end* contient les différentes classes permettant de faire fonctionner le projet ainsi que les fichiers YAML pour les déployer sur des containers.

Le dossier *db* contient les fichiers permettant la création de la base de données et notamment les différents YAML nécessaires.

Le dossier *front-end* contient tout le front pour notre formulaire et possède des fichiers YAML comme le back-end pour le déployer avec kubernetes.

Partie front-end

Le front-end est assez basique puisque c'est une SPA. Le principe du formulaire est de gérer une liste de personnes. Ainsi on peut :

- créer une personne avec son nom, prénom et email,
- supprimer une personne,
- modifier une personne.

On a ensuite la liste des différentes personnes présentes dans notre base de données.







Hello world!

Ajouter une personne

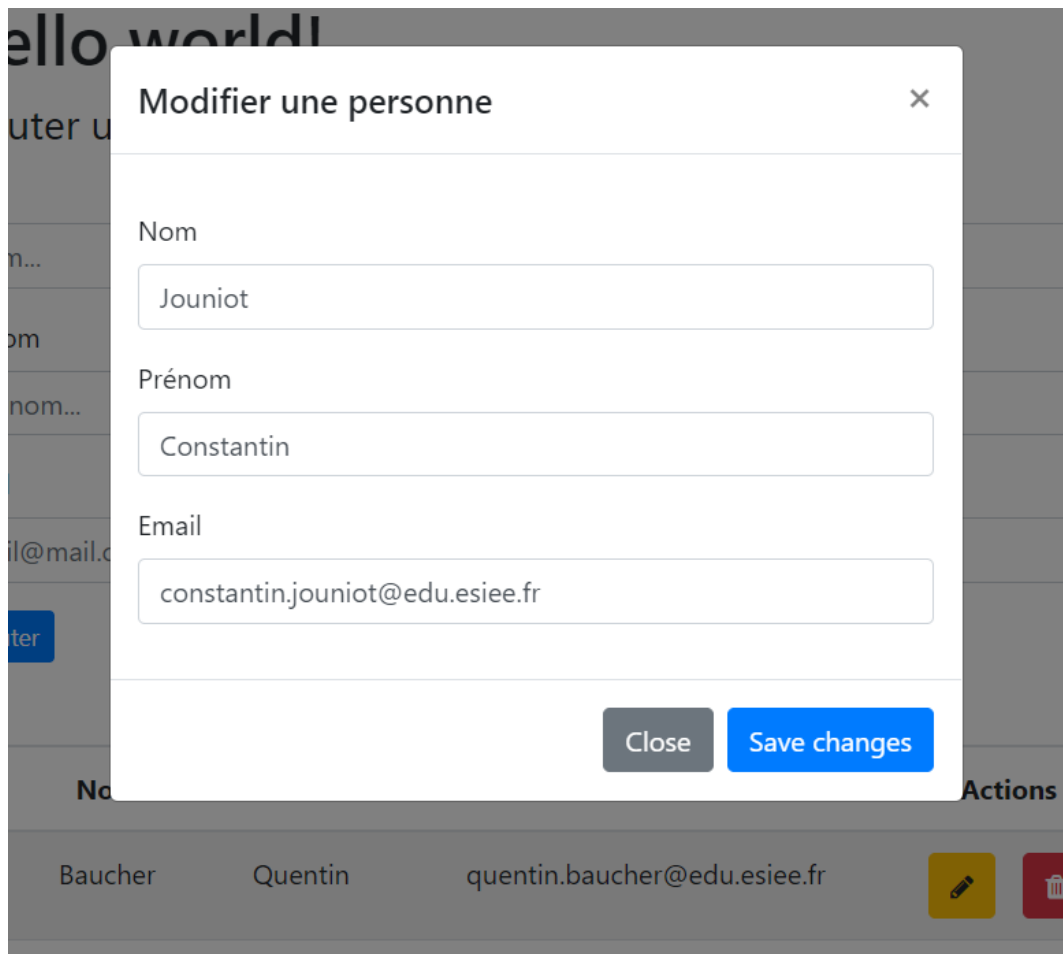
Nom

Prénom

Email

#	Nom	Prénom	Email	Actions	
1	Baucher	Quentin	quentin.baucher@edu.esiee.fr		
2	Bissonnier	Daphné	daphne.bissonnier@edu.esiee.fr		
3	Jouniot	Constantin	constantin.jouniot@edu.esiee.fr		

La modification des personnes présentes dans le formulaire passe par un modal :



The image shows a web application interface with a modal dialog titled "Modifier une personne". The modal contains three input fields: "Nom" with the value "Jouniot", "Prénom" with the value "Constantin", and "Email" with the value "constantin.jouniot@edu.esiee.fr". At the bottom right of the modal are two buttons: "Close" (grey) and "Save changes" (blue). The background shows a table with columns "Nom", "Prénom", "Email", and "Actions". The first row of the table contains the name "Baucher", the first name "Quentin", the email "quentin.baucher@edu.esiee.fr", and two icons (a pencil and a trash can).

Toute la mise en page a été faite avec Bootstrap.

Concernant la connexion entre front-end et back-end, nous passons par des appels REST en javascript, définis dans notre back-end.

Partie base de données - Création et Virtualisation

Afin d'avoir une base de données fonctionnelle, nous avons décidé de suivre un tutoriel sur un repository GitHub. Ce tutoriel nous explique comment mettre en place un cluster, puis un service Kubernetes. Le tutoriel de cette mise en place est trouvable à cette adresse : <https://github.com/charroux/noops/tree/main/mysql>

Dans ce tutoriel, nous utilisons des fichiers ".yaml" qui vont décrire la structure du cluster que nous voulons créer pour notre application MySQL. Par exemple, le fichier YAML suivant définit comment déployer le cluster avec kubernetes :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - image: mysql:5.7
          name: mysql
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-secret
                  key: password
          ports:
            - containerPort: 3306
              name: mysql
          volumeMounts:
            - name: mysql-persistent-storage
              mountPath: /var/lib/mysql
      volumes:
        - name: mysql-persistent-storage
          persistentVolumeClaim:
            claimName: mysql-pv-claim
```

Nous pouvons y définir par exemple : le port de notre container, l'application que l'on souhaite déployer ainsi que sa version, et comment on souhaite garder nos données afin qu'elles ne soient pas supprimées à la fermeture du cluster.

Dans ce cluster se trouvera notre base de données MySQL, où nous allons créer une base de données. Dans notre cas, nous avons décidé de l'appeler "db_project.db".

Nous allons aussi définir des identifiants qui seront importants dans le futur, car le back-end aura besoin de ces identifiants pour se connecter à la base de données.

Nous devons ensuite donner les autorisations nécessaires pour que ces identifiants puissent modifier la base de données créée précédemment.

Partie back-end

Afin de pouvoir gérer la base de données, nous avons décidé de créer un back-end qui communiquera avec cette dernière. Dans ce back-end nous effectuons des requêtes SQL à chaque requête HTTP sur l'API.

Dans cette API, nous avons défini les objets que nous manipulons, soit des humains :

```
// Constructors
public Human(int mID, String mLastName, String mFirstName, String mEmail) {
    super();
    this.mID = mID;
    this.mLastName = mLastName;
    this.mFirstName = mFirstName;
    this.mEmail = mEmail;
}
```

On compose donc un humain avec 4 attributs : un nom, un prénom, un e-mail, et son chiffre qui l'identifie afin de rendre unique chaque humain.

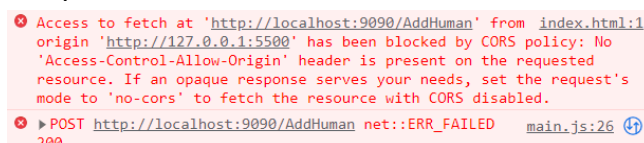
Comme expliqué précédemment, le back-end s'occupe de communiquer avec la base de données, par le biais de requêtes SQL. Voici un exemple de requête avec le back-end, afin de rajouter un humain à notre base de données :

```
@CrossOrigin(origins = "*", allowedHeaders = "*")
@PostMapping(value = "/AddHuman")
public void ajouterHumain(@RequestBody Human pHuman) {
    try {
        Class.forName(myDriver);
        Connection conn = DriverManager.getConnection(urlDB, idDB, MdPDB);
        Statement st = conn.createStatement();

        String query = "INSERT INTO HUMANS (FirstName, LastName, Email)"
            + "VALUES "
            + "(" + pHuman.getFirstName() + ", " + pHuman.getLastName() + ", " + pHuman.getEmail() + ")";
        st.executeUpdate(query);
        st.close();
    } catch (Exception e) {
        System.err.println("Got an exception! ");
        System.err.println(e.getMessage());
    }
}
```

Dans cette partie de code, on peut voir la requête "INSERT INTO HUMANS ..." qui sera ensuite transmise à la base de données afin de la mettre à jour. On peut aussi voir les différents paramètres apportés à la fonction, afin qu'elle soit liée à la requête HTTP GET à l'url : `http://<service-ip>:<port-ip>/AddHuman`.

Lors des premiers tests du front-end nous nous sommes retrouvés face à un problème de CORS. Comme par exemple :



```
✖ Access to fetch at 'http://localhost:9090/AddHuman' from index.html:1
origin 'http://127.0.0.1:5500' has been blocked by CORS policy: No
'Access-Control-Allow-Origin' header is present on the requested
resource. If an opaque response serves your needs, set the request's
mode to 'no-cors' to fetch the resource with CORS disabled.
✖ POST http://localhost:9090/AddHuman net::ERR_FAILED main.js:26
200
```

Afin de résoudre ce problème, nous devons en premier lieu changer des paramètres dans le back-end afin de donner l'autorisation à n'importe quelle application de faire un appel à l'API.

Virtualisation

Voici la procédure pour faire fonctionner notre projet :

Côté minikube

- S'assurer que l'instance Minikube est arrêtée et supprimée :
 - minikube stop
 - minikube delete
- Démarrer l'instance Minikube avec les ports 32000(back) et 32500(front) d'ouverts :
 - minikube start --ports=127.0.0.1:32000:32000,127.0.0.1:32500:32500

Côté app (si les images docker ne sont pas accessibles)

- Création des dockerfiles
- build pour faire un jar du back-end, si des modifications ont été apportées
- faire les images docker et les push sur le Docker Hub
 - docker build -t <votre id dockerHub>/front-end-image .
 - docker build -t <votre id dockerHub>/back-end-image .
 - docker push <votre id dockerHub>/front-end-image
 - docker push <votre id dockerHub>/back-end-image

Déployer notre environnement

- Appliquer notre infrastructure Kubernetes grâce à notre fichier YAML :
 - kubectl apply -f front-end-app.yaml

Configurer la base de donnée

- Accéder à notre SQL :
 - kubectl get pods
 - kubectl exec --stdin --tty <pods récupéré> -- mysql -ptest1234
- Avec la dernière commande on se retrouve dans le mySQL :
 - créer la bdd : create database db_project;
 - créer l'utilisateur : create user 'api' identified by 'api';
 - accorder les droits : grant all on db_project.* to 'api';
 - sortir de l'interface : exit

Démarrer les différents parties de notre application web

- Démarrer notre service de bdd : minikube service mysql --url
- Démarrer le back-end : minikube service back-end-service --url
- Démarrer le front-end : minikube service front-end-service --url
- Accéder à notre app front-end avec l'url retournée, et la coller dans la barre d'adresse d'un navigateur !

Pour plus de détails :

Les dockers push vont mettre les images, build du back et du front, dans le Docker Hub. Après cela, dans le fichier YAML de notre projet, nous allons faire référence à ces images afin de les déployer dans nos pods, qui seront ensuite utilisés pour créer des services hébergeants notre back, ou notre front.

Dans notre fichier YAML, nous allons définir les pods qui vont aller chercher ces images Docker, depuis le Docker Hub, directement sur les repository du profil "oxasa", profil de Constantin Jouniot.

Ensuite nous définissons des services qui vont démarrer un serveur nginx web pour le front, et exécuter un fichier .jar pour le back end.

Nous définissons ensuite le pod, le service de la base de données MySQL nous permettant ensuite de nous y connecter pour configurer la base de données, créer les identifiants et leur donner les accès nécessaires.

Dans un premier temps, le front-end refusait de communiquer avec le back-end. Nous avons essayé d'utiliser un "ingress controller" qui s'occuperait de faire office de "réceptionniste" entre les requêtes du navigateur web et le service du front end. Cependant, la configuration d'un ingress controller semble trop complexe et les documentations sur internet n'ont pas réussi à nous aider à le mettre en place.

Pour résoudre notre problème, nous avons alors décidé d'ouvrir les ports entre les deux clusters du back-end et du front-end. C'est pour cela qu'il faut démarrer minikube avec les paramètres de ports supplémentaires.

Enfin, nous avons juste à démarrer nos services avec la commande "minikube service", en les démarrant dans le bon ordre : base de données puis back-end puis front-end.

La base de données est vide lors du premier lancement. Après un "minikube delete" tous les persistent volumes sont supprimés.

C'est pour cela que vous pouvez remplir la base de données à l'aide du backend en effectuant simplement les requêtes suivantes :

`http://localhost:<port backend>/InitDB`

`http://localhost:<port backend>/FillDB`

Vous pouvez utiliser ces requêtes sur la barre d'adresse d'un navigateur en précisant le port du cluster du service du back-end que vous récupérerez lors de la commande "minikube service back-end-service -url". Il n'y a pas besoin de paramètres ou de body dans la requête, le back-end s'occupe de remplir des objets 'exemples' dans la base de données.

Qwilabs



Constantin JOUNIOT

Member since 2022

Your badge profile is not public and accessible.

[Make badge profile public](#)


[Activities](#)


Badges

Course	Lab	Quest	Quiz	Game	In progress	Finished
Activity	Type	Date started	Date finished	Score	Passed	
Set Up Network and HTTP Load Balancers	Lab	3 days ago	3 days ago	100.0/100.0	✓	
Kubernetes Engine: Qwik Start	Lab	3 days ago	3 days ago	100.0/100.0	✓	
Configuring Networks via gcloud	Lab	Feb 9, 2022	Feb 9, 2022	100.0/100.0	✓	
Configuring Networks via gcloud	Lab	Feb 9, 2022	Feb 9, 2022	80.0/100.0		
Infrastructure as Code with Terraform	Lab	Feb 4, 2022	Feb 4, 2022	100.0/100.0	✓	
Getting Started with Cloud Shell and gcloud	Lab	Jan 5, 2022	Jan 5, 2022	100.0/100.0	✓	
Creating a Virtual Machine	Lab	Jan 5, 2022	Jan 5, 2022	100.0/100.0	✓	
Google Cloud Essentials	Quest	Jan 5, 2022	3 days ago			
A Tour of Google Cloud Hands-on Labs	Lab	Jan 5, 2022	Jan 5, 2022	100.0/100.0	✓	

**Daphné BISSONNIER**

Member since 2022

Your badge profile is not public and accessible.

[Make badge profile public](#)
Paths
Activities
Badges[Course](#) [Lab](#) [Quest](#) [Quiz](#) [Game](#) [In progress](#) [Finished](#)

Activity	Type	Date started	Date finished	Score	Passed
Set Up Network and HTTP Load Balancers	Lab	Feb 18, 2022	Feb 18, 2022	100.0/100.0	✓
Kubernetes Engine: Qwik Start	Lab	Feb 18, 2022	Feb 18, 2022	100.0/100.0	✓
Configuring Networks via gcloud	Lab	Feb 9, 2022	Feb 9, 2022	100.0/100.0	✓
Infrastructure as Code with Terraform	Lab	Feb 4, 2022	Feb 4, 2022	100.0/100.0	✓
Getting Started with Cloud Shell and gcloud	Lab	Jan 5, 2022	Jan 5, 2022	100.0/100.0	✓
Google Cloud Essentials	Quest	Jan 5, 2022	Feb 18, 2022		
Creating a Virtual Machine	Lab	Jan 5, 2022	Jan 5, 2022	100.0/100.0	✓
A Tour of Google Cloud Hands-on Labs	Lab	Jan 5, 2022	Jan 5, 2022	100.0/100.0	✓

**Quentin BAUCHER**

Date d'abonnement : 2022

Votre profil n'est pas public ni accessible.

[Rendre votre profil public](#)
Activités
Badges

Cours	Atelier	Quête	Quiz	Jeu	En cours	Terminée
Activité	Type	Date de début	Date de fin	Score	Réussie	
Présentation des API de Google	Atelier	16 mars 2022	16 mars 2022	100.0/100.0	✓	
Google Assistant: Qwik Start - Dialogflow	Atelier	16 mars 2022	16 mars 2022	100.0/100.0	✓	
OK Google: Build Interactive Apps with Google Assistant	Quête	16 mars 2022				
Kubernetes Engine: Qwik Start	Atelier	4 févr. 2022	4 févr. 2022	100.0/100.0	✓	
Infrastructure as Code with Terraform	Atelier	4 févr. 2022	4 févr. 2022	100.0/100.0	✓	
Getting Started with Cloud Shell and gcloud	Atelier	5 janv. 2022	5 janv. 2022	100.0/100.0	✓	
Compute Engine: Qwik Start - Windows	Atelier	5 janv. 2022	5 janv. 2022	100.0/100.0	✓	
Compute Engine: Qwik Start - Windows	Atelier	5 janv. 2022	5 janv. 2022	100.0/100.0	✓	
Creating a Virtual Machine	Atelier	5 janv. 2022	5 janv. 2022	100.0/100.0	✓	
Google Cloud Essentials	Quête	5 janv. 2022			✓	
A Tour of Google Cloud Hands-on Labs	Atelier	5 janv. 2022	5 janv. 2022	100.0/100.0	✓	