

# Projet Virtualisation

## Objectifs principaux

1. [Réaliser plusieurs microservices](#)
  - a. Un microservice sous Angular (*front*) contenant une page d'accueil sous la forme d'un formulaire demandant les champs suivants : nom, prénom, date de naissance, numéro de téléphone, *email* et *hobby*.
  - b. Un microservice sous ExpressJS (*back*) faisant la passerelle entre le front et la base de données
  - c. Un microservice sous Postgres (base de données) pour stocker les informations obtenues grâce au formulaire côté *front* par le biais d'un bouton « *Submit* »
2. [Les initialiser via Docker et Kubernetes](#)
  - a. Générer les images Docker grâce aux Dockerfiles et les publier sur Docker Hub
  - b. Créer les fichiers YAML associés au déploiement et aux services Kubernetes
3. [Modification des routes grâce à Ingress](#)

## Annexes – Google Labs

### GitHub du projet

#### I. [Réalisation des microservices](#)

Pour commencer notre projet, nous sommes partis sur le *setup* de la partie front en premier en Angular : nous avons donc téléchargé Node.js ainsi que le *package* Angular via les commandes npm.

En lançant les commandes `ng new` et `ng generate` nous avons pu créer notre application web initiale ainsi qu'un *component* qui prendra nos informations en entrée à l'aide d'un formulaire de ce type :

**TEMPLATE FORM**

First Name:

Last Name:

Email:

Date of Birth:

Phone:

Hobby:

Figure 1: Formulaire affiché dans la page d'accueil

```
<form class="form-container" (submit)="onSubmit()">
  <div class="form-row">
    <label class="form-label">
      First Name:
      <input class="form-input" type="text" name="firstName" [(ngModel)]="firstName" required>
    </label>
  </div>

  <div class="form-row">
    <label class="form-label">
      Last Name:
      <input class="form-input" type="text" name="lastName" [(ngModel)]="lastName" required>
    </label>
  </div>

  <div class="form-row">
    <label class="form-label">
      Email:
      <input class="form-input" type="email" name="email" [(ngModel)]="email" required>
    </label>
  </div>

  <div class="form-row">
    <label class="form-label">
      Date of Birth:
      <input class="form-input" type="date" name="dob" [(ngModel)]="dob" required>
    </label>
  </div>

  <div class="form-row">
    <label class="form-label">
      Phone:
      <input class="form-input" type="tel" name="phone" [(ngModel)]="phone" required>
    </label>
  </div>

  <div class="form-row">
    <label class="form-label">
      Hobby:
      <input class="form-input" type="text" name="hobby" [(ngModel)]="hobby" required>
    </label>
  </div>

  <button class="form-button" type="submit">Submit</button>
</form>
```

Figure 2: Fichier html contenant le code associé à la page

Ensuite, nous sommes partis sur la réalisation en parallèle du microservice ExpressJS faisant le messenger entre le front et la base de données, donc nous avons créé respectivement le fichier index.js traitant le premier microservice et le fichier init.sql pour le second.

```
CREATE TABLE IF NOT EXISTS users (
  id SERIAL PRIMARY KEY,
  first_name VARCHAR(50) NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  email VARCHAR(50) NOT NULL,
  date_of_birth DATE NOT NULL,
  phone_number VARCHAR(20) NOT NULL,
  hobby VARCHAR(50) NOT NULL
);
```

Figure 3: Fichier init.sql qui créé la table

Nous avons ensuite pu tester via ng serve (donc en local) le front, ce qui a fonctionné correctement, ainsi que le back, nous retournant un message de type « is running ».

## II. Initialisation via Docker puis Kubernetes

Il faut donc ensuite générer les images Docker à l'aide des Dockerfiles présents dans chaque service, respectivement le front puis le back et enfin la base de données :

```
FROM node:14.21-alpine3.16 AS build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build --prod

FROM nginx:1.21.3-alpine
COPY --from=build /app/dist/angular-microservice /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Figure 4: Dockerfile du front

```
1 FROM node:14.21-alpine3.16
2 WORKDIR /app
3 COPY package*.json ./
4 RUN npm install
5 COPY . .
6 EXPOSE 3000
7 CMD ["npm", "start"]
```

Figure 5: Dockerfile du back

```
FROM postgres:13
ENV POSTGRES_USER=myuser
ENV POSTGRES_PASSWORD=mypassword
ENV POSTGRES_DB=mydatabase
COPY init.sql /docker-entrypoint-initdb.d/
EXPOSE 5432
CMD ["postgres"]
```

Figure 6: Dockerfile de la base de données

Après avoir mis en place les Dockerfiles dans chaque service, on réalise le build docker de l'image en spécifiant le nom de chaque service.

```
PS C:\Users\Viraen\Documents\E5FIC\Virtualisation\Projet\angular-microservice> docker build -t angular-microservice .
[+] Building 23.9s (16/16) FINISHED
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 311B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 28 0.0s
=> [internal] load metadata for docker.io/library/nginx:1.21.3-alpine 1.1s
=> [internal] load metadata for docker.io/library/node:14.21-alpine3.16 1.0s
=> [auth] library/node:pull token for registry-1.docker.io 0.0s
=> [auth] library/nginx:pull token for registry-1.docker.io 0.0s
=> [stage-1 1/2] FROM docker.io/library/nginx:1.21.3-alpine@sha256:1ff1364a1c4332341fc0a854820fd50e90e11bb0b93e 0.0s
=> [build 1/6] FROM docker.io/library/node:14.21-alpine3.16@sha256:322df1d76333426dba6008f1001e9e05a0518aa60f944 0.0s
=> [internal] load build context 4.1s
=> => transferring context: 376.63MB 4.0s
=> CACHED [build 2/6] WORKDIR /app 0.0s
=> CACHED [build 3/6] COPY package*.json ./ 0.0s
=> CACHED [build 4/6] RUN npm install 0.0s
=> [build 5/6] COPY . . 4.4s
=> [build 6/6] RUN npm run build --prod 14.2s
=> CACHED [stage-1 2/2] COPY --from=build /app/dist/angular-microservice /usr/share/nginx/html 0.0s
=> exporting to image 0.0s
=> => exporting layers 0.0s
=> writing image sha256:39451ccdf897c9fc769ce1ce873d67ae4f42e3a3dffe7e51df85d7fc8877cd1 0.0s
=> => naming to docker.io/library/angular-microservice 0.0s
```

Figure 7: Build pour le service front

```
PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet\express-microservice> docker build -t express-microservice .
[+] Building 1.3s (10/10) FINISHED
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 162B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/node:14.21-alpine3.16 0.4s
=> [1/5] FROM docker.io/library/node:14.21-alpine3.16@sha256:322df1d76333426dba6008f1001e9e05a0518aa60f94449000b 0.0s
=> [internal] load build context 0.6s
=> => transferring context: 2.66MB 0.6s
=> CACHED [2/5] WORKDIR /app 0.0s
=> CACHED [3/5] COPY package*.json ./ 0.0s
=> CACHED [4/5] RUN npm install 0.0s
=> [5/5] COPY . . 0.1s
=> exporting to image 0.1s
=> => exporting layers 0.1s
=> => writing image sha256:63f39c2ca4ce89454f7b37efb23687c1b26b88dabcfbb265702b3dd5b8ec97d 0.0s
=> => naming to docker.io/library/express-microservice 0.0s
```

Figure 8: Build pour le service back

```
PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet\postgres-microservice> docker build -t postgres-microservice .
[+] Building 5.1s (8/8) FINISHED
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 218B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/postgres:13 1.4s
=> [auth] library/postgres:pull token for registry-1.docker.io 0.0s
=> [1/2] FROM docker.io/library/postgres:13@sha256:9f263fc4bda05b418690b5b77cc34f9303ef4e55c820c61103cd1f3537ebb 0.0s
=> resolve docker.io/library/postgres:13@sha256:9f263fc4bda05b418690b5b77cc34f9303ef4e55c820c61103cd1f3537ebb 0.0s
=> sha256:f1f26f5702560b7e591bef5c4d840f76a232bf13fd5aefc4e22077a1ae4440c7 31.41MB / 31.41MB 0.5s
=> sha256:2de938a8d18db6eae223218647319501fe5935c6ba9ca3d033bc8560ede12a8df 10.56kB / 10.56kB 0.0s
=> sha256:dffc353b86ebaf024e489eb30725dec939794c5c853d8c40e8a228061ef618d 1.80kB / 1.80kB 0.3s
=> sha256:a54b771dfe68b69fb08b14acfbfb690bc191b1e7360fcb57b8c5aece238dcea 3.04kB / 3.04kB 0.0s
=> sha256:9f263fc4bda05b418690b5b77cc34f9303ef4e55c820c61103cd1f3537ebb05e 1.86kB / 1.86kB 0.0s
=> sha256:1c04f8741265c9eb018d6e0dc852fa48d9ca605cc4a8b910701149c268b46c4c 4.41MB / 4.41MB 0.3s
=> sha256:81f47e7b385280253a608f12b70c590d831395a3efe2908bc3f1eb9f18ba06b6 8.05MB / 8.05MB 0.6s
=> sha256:18c4a9e6c414dbdb0cea88de7b9d555fb825bbbd7b3ceff82ac6de73109460c58 1.47MB / 1.47MB 0.5s
=> sha256:a2c3dc85e8c3b85d3f412d47e5e3091453f92a0ca1de2154fee0efd915202eb4 149B / 149B 0.6s
=> extracting sha256:f1f26f5702560b7e591bef5c4d840f76a232bf13fd5aefc4e22077a1ae4440c7 0.5s
=> sha256:5e26c947960d4da4154c7610b237230df37598a1cc6fbcaf57e6aa25b10e7a9d 1.26MB / 1.26MB 0.6s
=> sha256:17df73636f012eaeab40d794735572017fd87f992aa4338ac562491cfcdb3b8a6c 3.20kB / 3.20kB 0.7s
=> sha256:124bb42a38523d0af62820a290343334f03d129940ef9ae0c8f65e45907d158a 90.14MB / 90.14MB 1.6s
=> sha256:dfb19482a052ca2a64e7dd591ade7f26053810ec7865d46dd9271c0caa837b60 9.36kB / 9.36kB 0.8s
=> sha256:bbb12a596105bfbd9e6047e9457c156fac202afa3cec7c9fc4359b592cc6ef6 129B / 129B 0.8s
=> sha256:aa8960c4e38366ea5ca639b3a563168c2920ebdd973c18b693d43d4324c8bc0 199B / 199B 0.9s
=> sha256:12fcfc0ea93bce5032843f1590a0582aaf2d59d33789c01226a68cd694791199 4.78kB / 4.78kB 1.0s
=> extracting sha256:1c04f8741265c9eb018d6e0dc852fa48d9ca605cc4a8b910701149c268b46c4c 0.1s
=> extracting sha256:dffc353b86ebaf024e489eb30725dec939794c5c853d8c40e8a228061ef618d 0.0s
=> extracting sha256:18c4a9e6c414dbdb0cea88de7b9d555fb825bbbd7b3ceff82ac6de73109460c58 0.0s
=> extracting sha256:81f47e7b385280253a608f12b70c590d831395a3efe2908bc3f1eb9f18ba06b6 0.2s
=> extracting sha256:5e26c947960d4da4154c7610b237230df37598a1cc6fbcaf57e6aa25b10e7a9d 0.0s
=> extracting sha256:a2c3dc85e8c3b85d3f412d47e5e3091453f92a0ca1de2154fee0efd915202eb4 0.0s
=> extracting sha256:17df73636f012eaeab40d794735572017fd87f992aa4338ac562491cfcdb3b8a6c 0.0s
=> extracting sha256:124bb42a38523d0af62820a290343334f03d129940ef9ae0c8f65e45907d158a 1.1s
=> extracting sha256:dfb19482a052ca2a64e7dd591ade7f26053810ec7865d46dd9271c0caa837b60 0.0s
=> extracting sha256:bbb12a596105bfbd9e6047e9457c156fac202afa3cec7c9fc4359b592cc6ef6 0.0s
=> extracting sha256:aa8960c4e38366ea5ca639b3a563168c2920ebdd973c18b693d43d4324c8bc0 0.0s
=> extracting sha256:12fcfc0ea93bce5032843f1590a0582aaf2d59d33789c01226a68cd694791199 0.0s
=> [internal] load build context 0.0s
=> => transferring context: 303B 0.0s
=> [2/2] COPY init.sql /docker-entrypoint-initdb.d/ 0.3s
=> exporting to image 0.1s
=> => exporting layers 0.0s
=> => writing image sha256:e86542d89a0fe5321b02fe06088ffcc09c3052136b8e5e2d707978ab22d8046 0.0s
=> => naming to docker.io/library/postgres-microservice 0.0s
```

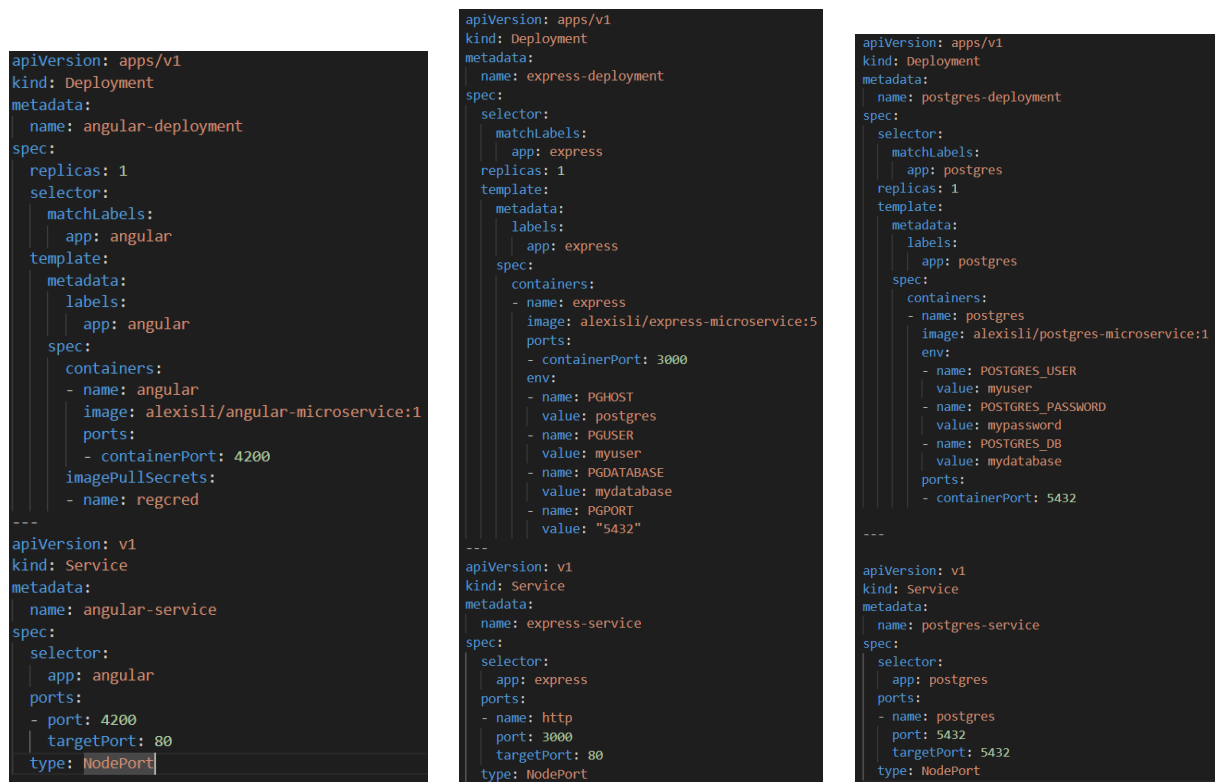
Figure 9: Build pour la base de données

Enfin, on les envoie sur Docker Hub en utilisant docker tag puis docker push (la connexion est réalisée via docker login avec les identifiants docker) :

```
PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet> docker tag angular-microservice alexis11/angular-microservice:6
PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet> docker tag express-microservice alexis11/express-microservice:6
PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet> docker tag postgres-microservice alexis11/postgres-microservice:2
PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet> docker push alexis11/angular-microservice:2
The push refers to repository [docker.io/alexis11/angular-microservice]
9ead53660782: Layer already exists
40403bebe4fd: Layer already exists
14b4e8910ea: Layer already exists
311d8db33235: Layer already exists
20d0effdf3a2: Layer already exists
ed3ce2a19ff: Layer already exists
e2eb06d8af82: Layer already exists
2: digest: sha256:626dea162d617a043920b4947227df2ac8d9fb3469acddfed915bb1e6c7e907b size: 1777
The push refers to repository [docker.io/alexis11/express-microservice]
d2becde1a24e: Pushed
6312e36fec52: Layer already exists
56f25fe0aeb: Layer already exists
d1a8763a6858: Layer already exists
8ba0f0bb3b25: Layer already exists
beea639c0727: Layer already exists
2ab04dd0d931: Layer already exists
1a596d83888: Layer already exists
8: digest: sha256:8c73e6c01369cd3522a7330b28b464da84dec7f5b91c1627a707100e24ffb09 size: 1994
PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet> docker push alexis11/postgres-microservice:2
The push refers to repository [docker.io/alexis11/postgres-microservice]
09908421848a: Pushed
69138336a47: Mounted from library/postgres
7db3bc1a6ac1: Mounted from library/postgres
df69d13d5dc0: Mounted from library/postgres
42c75349702: Mounted from library/postgres
fa9375cdcf1f: Mounted from library/postgres
5edf3610ca63: Mounted from library/postgres
96a56127e73: Mounted from library/postgres
ef59638c4e3a: Mounted from library/postgres
7bde5be9d2d: Mounted from library/postgres
018104e1297: Mounted from library/postgres
3a54725df41: Mounted from library/postgres
a9fa3183c595: Mounted from library/postgres
b4f1e49a249: Mounted from library/postgres
2: digest: sha256:ee4f172dec03442118d8f1b3c5e60154a94a4afec88408f3b4ec2cd0d30cc size: 3246
```

Lorsque les builds et les images Docker sont publiées, on passe à la création des fichiers YAML spécifiques à Kubernetes. Minikube étant déjà installé sur notre poste, nous pouvons les mettre en place en 3 fichiers distincts, chacun arborant le déploiement et le service dans un seul et même fichier.

Note : On retirera les tags après pour prendre la version la plus récente (voir [annexe](#)).



```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: angular-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: angular
  template:
    metadata:
      labels:
        app: angular
    spec:
      containers:
        - name: angular
          image: alexisli/angular-microservice:1
          ports:
            - containerPort: 4200
          imagePullSecrets:
            - name: regcred
---
apiVersion: v1
kind: Service
metadata:
  name: angular-service
spec:
  selector:
    app: angular
  ports:
    - port: 4200
      targetPort: 80
  type: NodePort

```

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: express-deployment
spec:
  selector:
    matchLabels:
      app: express
  replicas: 1
  template:
    metadata:
      labels:
        app: express
    spec:
      containers:
        - name: express
          image: alexisli/express-microservice:5
          ports:
            - containerPort: 3000
          env:
            - name: PGHOST
              value: postgres
            - name: PGUSER
              value: myuser
            - name: PGDATABASE
              value: mydatabase
            - name: PGPORT
              value: "5432"
---
apiVersion: v1
kind: Service
metadata:
  name: express-service
spec:
  selector:
    app: express
  ports:
    - name: http
      port: 3000
      targetPort: 80
  type: NodePort

```

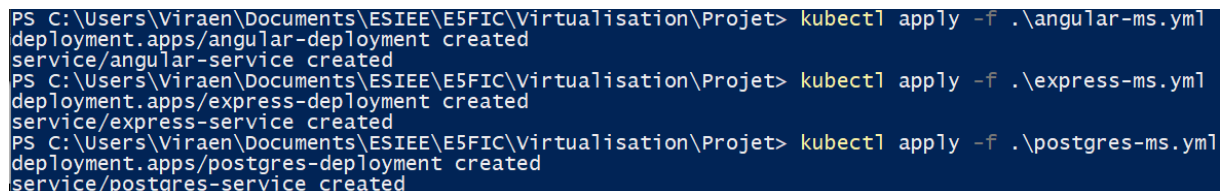
```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-deployment
spec:
  selector:
    matchLabels:
      app: postgres
  replicas: 1
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: alexisli/postgres-microservice:1
          env:
            - name: POSTGRES_USER
              value: myuser
            - name: POSTGRES_PASSWORD
              value: mypassword
            - name: POSTGRES_DB
              value: mydatabase
          ports:
            - containerPort: 5432
---
apiVersion: v1
kind: Service
metadata:
  name: postgres-service
spec:
  selector:
    app: postgres
  ports:
    - name: postgres
      port: 5432
      targetPort: 5432
  type: NodePort

```

Figure 10: Fichiers YAML pour les trois microservices

Nous avons donc nos trois services prêts à être installés sous Kubernetes :



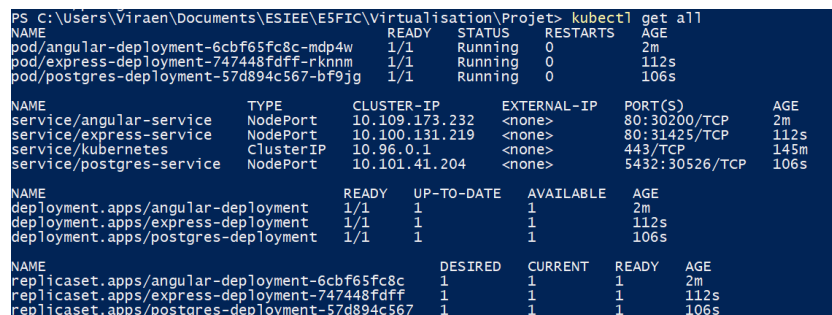
```

PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet> kubectl apply -f .\angular-ms.yml
deployment.apps/angular-deployment created
service/angular-service created
PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet> kubectl apply -f .\express-ms.yml
deployment.apps/express-deployment created
service/express-service created
PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet> kubectl apply -f .\postgres-ms.yml
deployment.apps/postgres-deployment created
service/postgres-service created

```

Figure 11: Lancement des fichiers YAML

Les fichiers étant lancés nous pouvons vérifier leur bon fonctionnement :



```

PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet> kubectl get all
NAME                                READY    STATUS    RESTARTS   AGE
pod/angular-deployment-6cbf65fc8c-mdp4w  1/1      Running   0           2m
pod/express-deployment-747448fdff-rknnm  1/1      Running   0          112s
pod/postgres-deployment-57d894c567-bf9jg  1/1      Running   0          106s

NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
service/angular-service             NodePort            10.109.173.232  <none>           80:30200/TCP     2m
service/express-service             NodePort            10.100.131.219  <none>           80:31425/TCP     112s
service/kubernetes                   ClusterIP           10.96.0.1       <none>           443/TCP          145m
service/postgres-service            NodePort            10.101.41.204   <none>           5432:30526/TCP   106s

NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/angular-deployment  1/1      1              1            2m
deployment.apps/express-deployment  1/1      1              1            112s
deployment.apps/postgres-deployment  1/1      1              1            106s

NAME                                DESIRED    CURRENT    READY    AGE
replicaset.apps/angular-deployment-6cbf65fc8c  1          1          1        2m
replicaset.apps/express-deployment-747448fdff  1          1          1        112s
replicaset.apps/postgres-deployment-57d894c567  1          1          1        106s

```

Figure 12: Vérification du bon fonctionnement des pods, services, déploiements, replicas

Ensuite, nous avons pu tester une nouvelle fois le fonctionnement de la partie front et de la partie back de manière indépendante grâce à la commande port-forward dans Kubernetes :

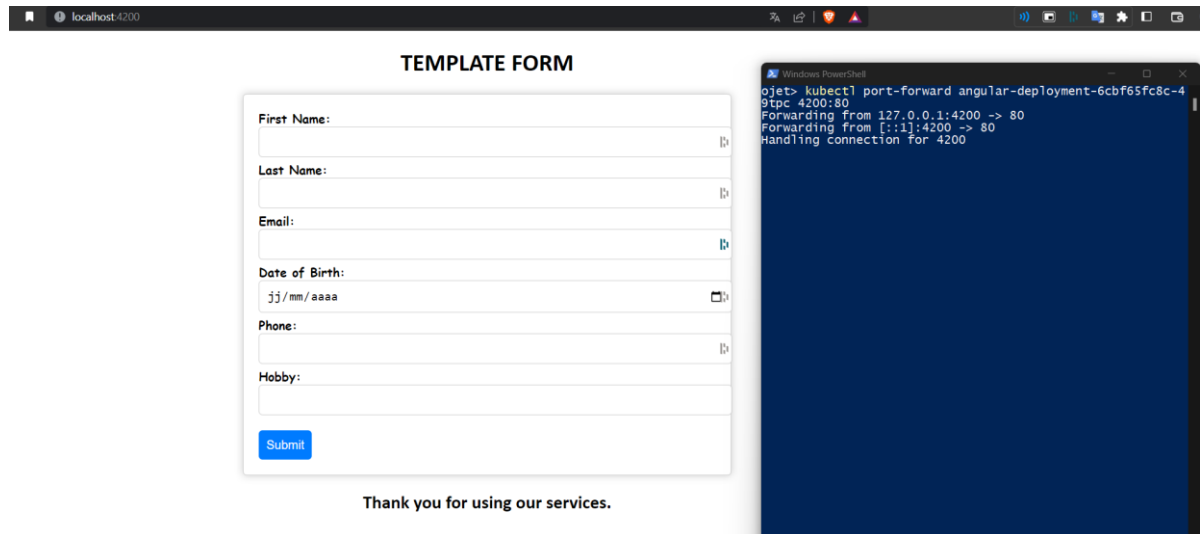


Figure 13: Test fonctionnement PF du front

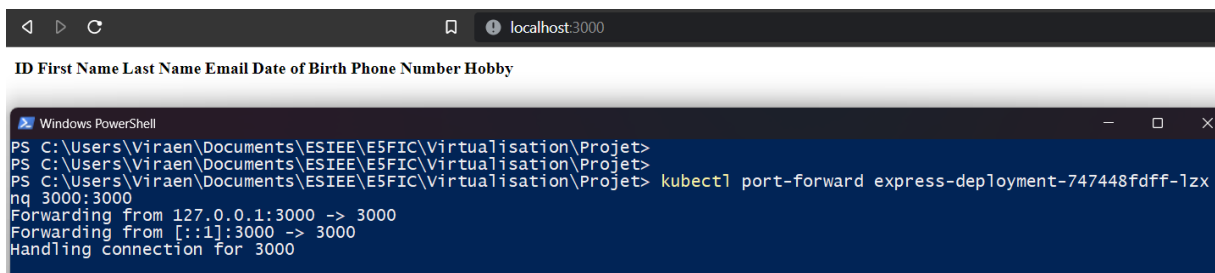


Figure 14: Test fonctionnement PF du back

Enfin pour la base de données, nous avons utilisé exec pour entrer dans le conteneur pour vérifier la création de la table que nous avons initialisé précédemment :

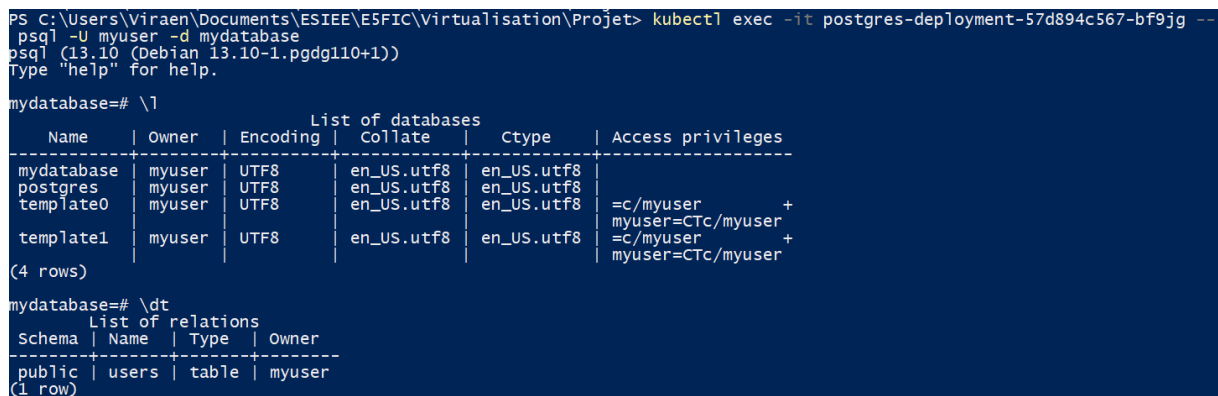


Figure 15: Test fonctionnement exec de la base de données

Nous pouvons donc maintenant déployer correctement les services via la commande minikube service <nomduservice> :

```
PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet> minikube service angular-service
```

NAMESPACE	NAME	TARGET PORT	URL
default	angular-service	4200	http://192.168.49.2:30200

\* Tunnel de démarrage pour le service angular-service.

NAMESPACE	NAME	TARGET PORT	URL
default	angular-service		http://127.0.0.1:57601

\* Ouverture du service default/angular-service dans le navigateur par défaut...  
! Comme vous utilisez un pilote Docker sur windows, le terminal doit être ouvert pour l'exécuter.

127.0.0.1:57601

## TEMPLATE FORM

First Name:

Last Name:

Email:

Date of Birth:

jj/mm/aaaa

Phone:

Hobby:

Submit

Thank you for using our services.

### III. Ingress

Nous mettons en place Ingress pour notre solution à l'aide du fichier `ingress.yml`, ce qui nous permet de ne pas à avoir à mettre une adresse IP spécifique pour accéder à nos services.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myingress
  labels:
    name: myingress
spec:
  rules:
  - host: form.default
    http:
      paths:
      - path: /angular
        pathType: Prefix
        backend:
          service:
            name: angular-service
            port:
              number: 4200
      - path: /express
        pathType: Prefix
        backend:
          service:
            name: express-service
            port:
              number: 3000
      - path: /postgres
        pathType: Prefix
        backend:
          service:
            name: postgres-service
            port:
              number: 5432
```

Figure 16: Fichier `ingress.yml` paramétrant Ingress pour Kubernetes

Nous modifions donc en conséquence le fichier `etc/hosts` de notre ordinateur pour pouvoir bénéficier de l'accès.

```
# localhost name resolution is handled within DNS itself.
# 127.0.0.1 localhost
# ::1 localhost
# Added by Docker Desktop
192.168.1.15 host.docker.internal
192.168.1.15 gateway.docker.internal
# To allow the same kube context to work on the host and the container:
127.0.0.1 kubernetes.docker.internal
127.0.0.1 form.default
# End of section
```

Figure 17: Modification du fichier `etc/hosts`



Nous pouvons donc tester (comme Ingress est déjà activé sur notre machine) la commande suivante :

```
PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet> minikube tunnel
* Tunnel démarré avec succès
* REMARQUE : veuillez ne pas fermer ce terminal car ce processus doit rester actif pour que le tunnel soit accessible...
! Accéder aux ports inférieurs à 1024 peut échouer sur windows avec les clients OpenSSH antérieurs à v8.1. Pour plus d'informa
tion, voir: https://minikube.sigs.k8s.io/docs/handbook/accessing/#access-to-ports-1024-on-windows-requires-root-permission
* Tunnel de démarrage pour le service myingress.
```

Figure 18: Commande minikube tunnel

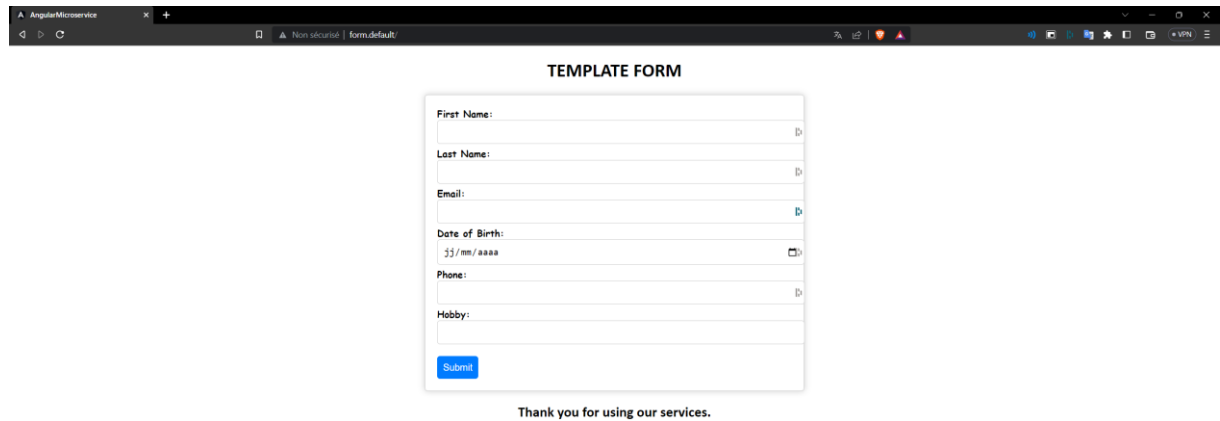


Figure 19: Utilisation de la nouvelle adresse

Au stade final de notre projet, nous avons pu réussir à lancer la nouvelle adresse sans soucis. Néanmoins des erreurs de communication entre les services sont à noter dans notre solution car lorsque l'on essaie de soumettre notre formulaire, une erreur de type 502 subvient :

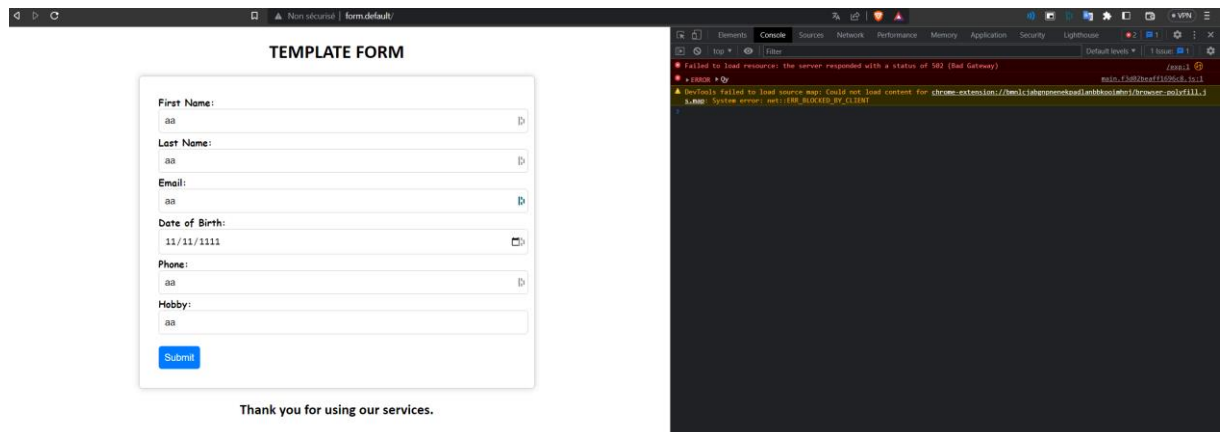


Figure 20: Erreur 502

Ceci est probablement dû à la configuration côté back qui n'est pas en cohérence avec le front, ou bien plus largement un dysfonctionnement général qui affecte tant le back que la base de données. Il est à préciser que le service express fonctionne correctement par port-forward à l'adresse suivante :

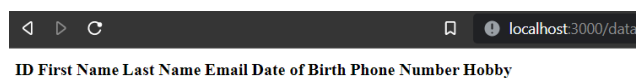


Figure 21: 2è test express par port-forward

## Annexes – Google Labs

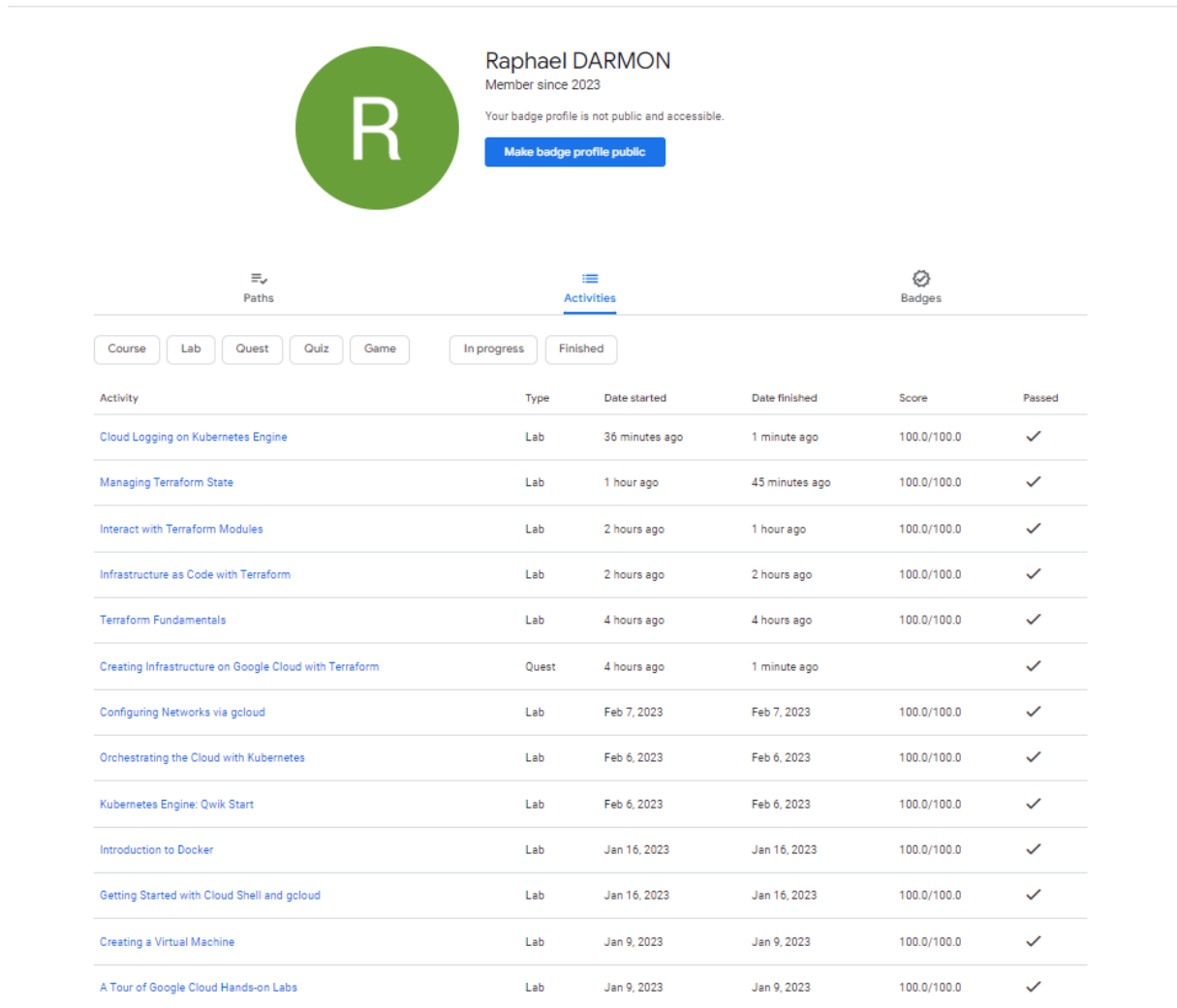



Figure 22: Google Labs - Raphaël

## Projet Virtualisation – E5FIC – Raphaël DARMON – Alexis LI



### Alexis LI

Member since 2023

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
<a href="#">Cloud Logging on Kubernetes Engine</a>	Lab	7 days ago	7 days ago	100.0/100.0	✓
<a href="#">Managing Terraform State</a>	Lab	7 days ago	7 days ago	100.0/100.0	✓
<a href="#">Interact with Terraform Modules</a>	Lab	7 days ago	7 days ago	100.0/100.0	✓
<a href="#">Infrastructure as Code with Terraform</a>	Lab	7 days ago	7 days ago	100.0/100.0	✓
<a href="#">Terraform Fundamentals</a>	Lab	7 days ago	7 days ago	100.0/100.0	✓
<a href="#">Creating Infrastructure on Google Cloud with Terraform</a>	Quest	7 days ago	0 minutes ago		✓
<a href="#">Configuring Networks via gcloud</a>	Lab	Feb 7, 2023	Feb 7, 2023	100.0/100.0	✓
<a href="#">Orchestrating the Cloud with Kubernetes</a>	Lab	Feb 6, 2023	Feb 6, 2023	100.0/100.0	✓
<a href="#">Kubernetes Engine: Qwik Start</a>	Lab	Feb 6, 2023	Feb 6, 2023	100.0/100.0	✓
<a href="#">Introduction to Docker</a>	Lab	Jan 16, 2023	Jan 16, 2023	100.0/100.0	✓
<a href="#">Getting Started with Cloud Shell and gcloud</a>	Lab	Jan 16, 2023	Jan 16, 2023	100.0/100.0	✓
<a href="#">Creating a Virtual Machine</a>	Lab	Jan 9, 2023	Jan 9, 2023	100.0/100.0	✓
<a href="#">A Tour of Google Cloud Hands-on Labs</a>	Lab	Jan 9, 2023	Jan 9, 2023	100.0/100.0	✓

Figure 23: Google Labs - Alexis

## Annexe – Build sans tags (latest)

```
PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet\angular-microservice> docker push alexisli/angular-microservice
Using default tag: latest
The push refers to repository [docker.io/alexisli/angular-microservice]
9ead53660782: Layer already exists
40403bebe4fd: Layer already exists
b4b4e85910ea: Layer already exists
311d8db33235: Layer already exists
20d0effdf3a2: Layer already exists
e6d3cea19fef: Layer already exists
e2eb06d8af82: Layer already exists
latest: digest: sha256:626dea162d617a043920b4947227df2ac8d9fb3469ac6dfed915bb1e6c7e907b size: 1777
```

```
PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet\express-microservice> docker push alexisli/express-microservice
Using default tag: latest
The push refers to repository [docker.io/alexisli/express-microservice]
bc0003d6f5b1: Pushed
c412e36fec52: Layer already exists
56f25fe0ae0b: Layer already exists
01a8763a6868: Layer already exists
9baf0bb81b25: Layer already exists
beea639c0727: Layer already exists
2ab04bd0d931: Layer already exists
aa5968d388b8: Layer already exists
latest: digest: sha256:a40d63b9d1be7e03e89e77044ca1a0b399142980ea433b337e2c3ebdfb74f275 size: 1994
```

```
PS C:\Users\Viraen\Documents\ESIEE\E5FIC\Virtualisation\Projet\postgres-microservice> docker push alexisli/postgres-microservice
Using default tag: latest
The push refers to repository [docker.io/alexisli/postgres-microservice]
09908421848a: Layer already exists
a9a138836a47: Layer already exists
7db3bc1e6aca: Layer already exists
df69d13d5dc0: Layer already exists
42c7a5349702: Layer already exists
fa397ccdc7cf: Layer already exists
5edf3610ca63: Layer already exists
96a5e612e773: Layer already exists
ef59638c463e: Layer already exists
7bde5bee9d2d: Layer already exists
0188104e1297: Layer already exists
3a54f275df41: Layer already exists
a9fa3183c595: Layer already exists
3af14c9a24c9: Layer already exists
latest: digest: sha256:eer4172decd03442118d8f1b3c58e60154a94a1aafec88408f3b4ec2cd0d30cc size: 3246
```

Ce qui nous permet de modifier les tags dans les fichiers YAML :

```
image: alexisli/angular-microservice:latest
```

```
name: express
image: alexisli/express-microservice:latest
```

```
image: alexisli/postgres-microservice:latest
```