TP1
Docker File

Récupérer le code et le transférer vers un dépôt de Git à vous

Dépôt d'origine : https://github.com/charroux/ingnum

Pour changer l'origin et la faire pointer vers votre projet :

```
git remote remove origin
git remote add origin adresseDeVotreDepotGit
```

Tester le programme sans Docker

Installer Java JDK 21

Builder le projet (compiler) :

./gradlew build

Tester le projet :

```
java -jar build/libs/RentalService-0.0.1-SNAPSHOT.jar
```

Vérifier dans votre navigateur à l'adresse :
http://localhost:8085/bonjour

Ecrire un fichier appelé Dockerfile

FROM eclipse-temurin:21

VOLUME /tmp

EXPOSE 8080

ADD ./build/libs/RentalService-0.0.1-SNAPSHOT.jar app.jar

ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]

Placer le Dockerfile dans le dossier RentalService

Lancer la création de l'image Docker :

docker build –t nomImageDpcker .

Tester le programme avec Docker

Avec docker run :

docker run –p 8080:8080 nomImageDocker

Vérifier dans votre navigateur à l'adresse :
http://localhost:8080/bonjour

Editer le README

Modifier le fichier README.md pour y inclure les commandes que vous avez fait.

Faites un piush vers votre dépôt git

Publier l'image dans votre Docker Hub

TP2

Deuxième microservice

Coder un deuxième service en PHP qui se content quand on l'appelle de retourner votre prénom. Ce service est accessible via une requête HTTP GET à partir de votre navigateur => Créer un second dans votre projet qui contient le code PHP.

Ajouter à ce dossier un Dockerfile spécifique à PHP.

Créer l'image Docker.

La tester.

La publier sur le Docker Hub.

Mettre le lien de votre image Docker dans le fichier des TPs.

Mettre à jour votre dépôt Git.

Communication entre vos deux microservices via HTTP

Vérifier l'architecture de votre application. Voir : https://github.com/charroux/docker-compose

Ecrire un fichier docker compose qui contient 2 services et 1 réseau.

Modifier le code côté Java pour qu'il envoie une requête HTTP : https://github.com/charroux/docker-compose/blob/main/RentalService/src/main/java/com/rental/controller/RentalController.java

Voir la méthode "bonjour"

Spécifier l'adresse de l'application PHP dans le fichier de configuration : https://github.com/charroux/docker-compose/blob/main/RentalService/src/main/resources/application.properties

Voir customer.service.url

Lancer Docker compose : docker-compose up

Tester avec votre navigateur : `http://localhost:8080/customer/Jean%20Dupont`

Option (notée en bonus)

Intégrer une base de données

A faire : inspirez-vous du projet https://github.com/charroux/docker-compose et l'adapter à votre projet

TP3

Kubernetes

Réaliser le TP https://github.com/charroux/kubernetes-minikube

à partir d'ici : https://github.com/charroux/kubernetes-minikube?tab=readme-ov-file#create-a-kubernetes-deployment-from-a-docker-image

Et jusque-là (non inclus) : https://github.com/charroux/kubernetes-minikube?tab=readme-ov-file#routing-rule-to-a-service-using-ingress

En utilisant une de vos images Docker.

Faites des copies d'écran du fonctionnement de Kubernetes.

Regroupez ces copies d'écran dans un mini rapport.

Vous ajouterez à ce mini rapport l'adresse Github du projet.

Vous publierez ce rapport en tant que remise du projet.

Utilisation des fichiers yml à la place des commandes

Ajouter dans votre dépôt Git, à côté du fichier docker compose un nouveau fichier yml pour Kubernetes. Ce fichier doit contenir :

- Un déploiement
- Un service

Voir les fichiers :

> https://github.com/charroux/kubernetes-minikube/blob/main/myservice-deployment.yml
>
> https://github.com/charroux/kubernetes-minikube/blob/main/myservice-loadbalancing-service.yml
>
> Que vous pouvez regrouper dans le même fichier séparé par ---

Communication entre microservices

Reprendre votre TP avec Docker Compose et ajoutez un fichier yaml pour Kubernetes. Voir : https://github.com/charroux/CodingWithKubernetes/blob/master/front-back-app.yml

Le code devrait ? rester identique.

Modifier le fichier de propriété (mettez en commentaire l'url pour docker compose ou faites-en une autre), pour y ajouter l'adresse à la norme Kubernetes. Voir : https://github.com/charroux/CodingWithKubernetes/blob/master/FrontEnd/src/main/resources/application.yml

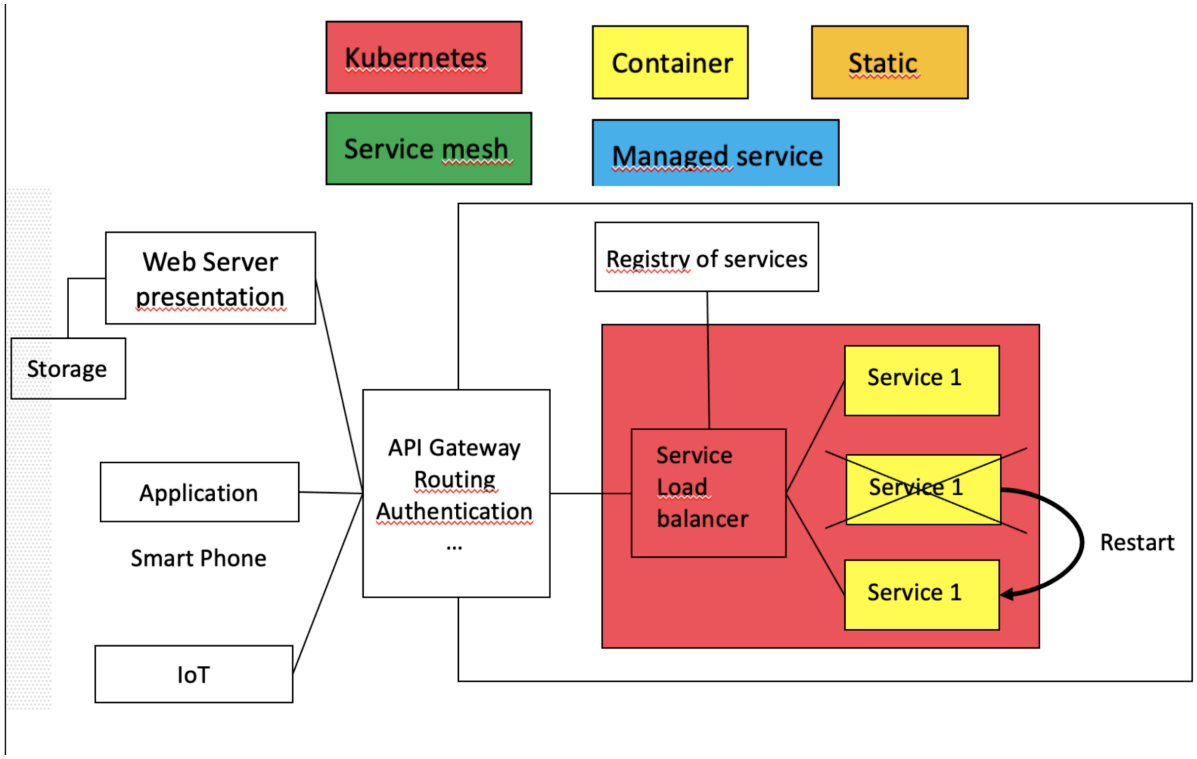
Compléter le fichier yml Kubernetes de la question précédente avec le deuxième service.

Mettez à jour votre Github

Gateway (serveur porte d'entrée devant nos microservices)

Première gateway de base : routage des requêtes

But : introduire un serveur unique qui sert de point d'entrée aux microsercices



Utiliser le serveur Web nginx avec un programme de routage : Ingress.

Réaliser la procédure : https://github.com/charroux/kubernetes-minikube/tree/main?tab=readme-ov-file#routing-rule-to-a-service-using-ingress

Ajouter le fichier Ingress dans votre dépôt Git.