

Mickael MOHAMMED

2022

M1 Cybersécurité FI

Numéro étudiant : 51804780

Projet de programmation distribuée

Documentation technique

Tables des matières :

0) Introduction

0.1) Spécification des besoins fonctionnels du web service REST

0.2) Technologies utilisées

1) Création d'une application web avec Spring Boot

2) Architecture de l'application

- 2.a) La classe Pays
- 2.b) L'interface Pays Repository
- 2.c) La classe Pays Controller
- 2.d) La classe Pays Application

3) Exécution de l'application initiale à l'aide d'une image docker dans un conteneur Docker

- 3.a) Création d'un Docker file
- 3.b) Création d'une image docker associée à notre application (docker build)
- 3.c) Exécution de l'image docker dans un conteneur docker (docker run)
- 3.d) Envoi de l'image docker vers le Docker Hub (docker push)
- 3.e) Téléchargement de cette image docker depuis le docker hub (docker pull)
- 3.f) Exécution de l'image docker importée du docker hub

4) Déploiement de notre web service à l'aide de Kubernetes Engine

- 4.a) Création d'un cluster Kubernetes
- 4.b) Création d'un déploiement de notre web-service
- 4.c) Exposer le déploiement

Note importante : projet réalisé sans binôme

0) Introduction

0.1) Spécification des besoins fonctionnels de notre web service

Le but de ce projet est de programmer une application web Service REST qu'on va coder avec Spring boot.

Cette application se focalise sur les informations générales d'un ensemble de pays.

Nous pourrons donc consulter les informations sur un ou plusieurs pays, ajouter des pays à notre base de données, modifier des informations sur un pays et supprimer un ou plusieurs pays. Nous pourrons ensuite directement exécuter notre application depuis Spring tools suite en exécutant le programme principale de notre projet, c'est-à-dire en cliquant sur « exécuter l'application avec Spring Boot ».

Une fois l'application exécutée, il nous sera possible d'effectuer nos requêtes sur cette application depuis un navigateur internet.

Nous allons ensuite créer une image docker associé à notre application initiale.

On pourra ensuite exécuter notre image docker dans un conteneur docker depuis un terminal. Nous pourrons donc exécuter nos requêtes à nouveau directement sur un navigateur internet grâce à l'exécution de l'image docker en local dans un conteneur docker.

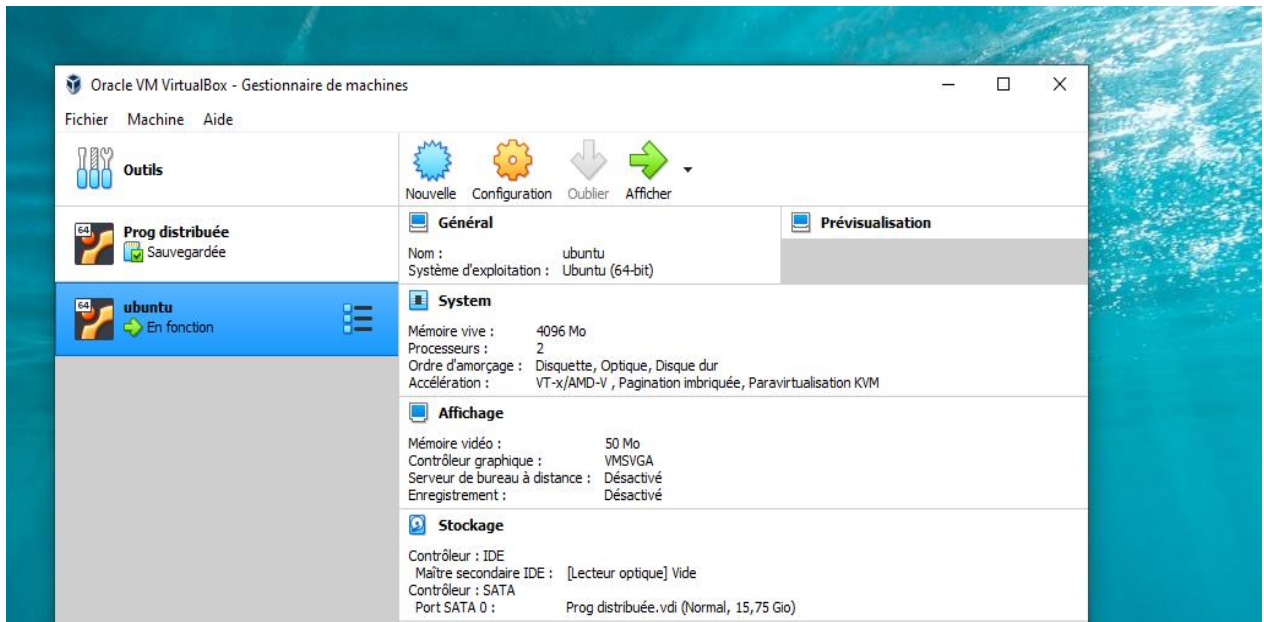
Nous allons ensuite déposer cette image dans le docker hub (docker push), pour que cette image docker de notre application soit accessible depuis le docker hub et non plus uniquement en locale. Nous allons ensuite supprimer l'image docker de notre web service en locale puis nous allons la télécharger depuis le docker hub (docker pull) et nous allons enfin re-exécuter notre application. Cela nous permettra de vérifier qu'il nous est bien possible d'exécuter notre application web service depuis une image docker qui a été importé du docker hub.

Enfin, nous allons enfin utiliser l'outil Kubernetes dans Google Cloud Platform pour pouvoir déployer notre application web service, c'est-à-dire la rendre accessible à tout le monde depuis n'importe quelle terminal. Nous allons nous servir des ressources de calcul du Google cloud avec Kubernetes pour que l'application directement de depuis Google Cloud avec Kubernetes Engine.

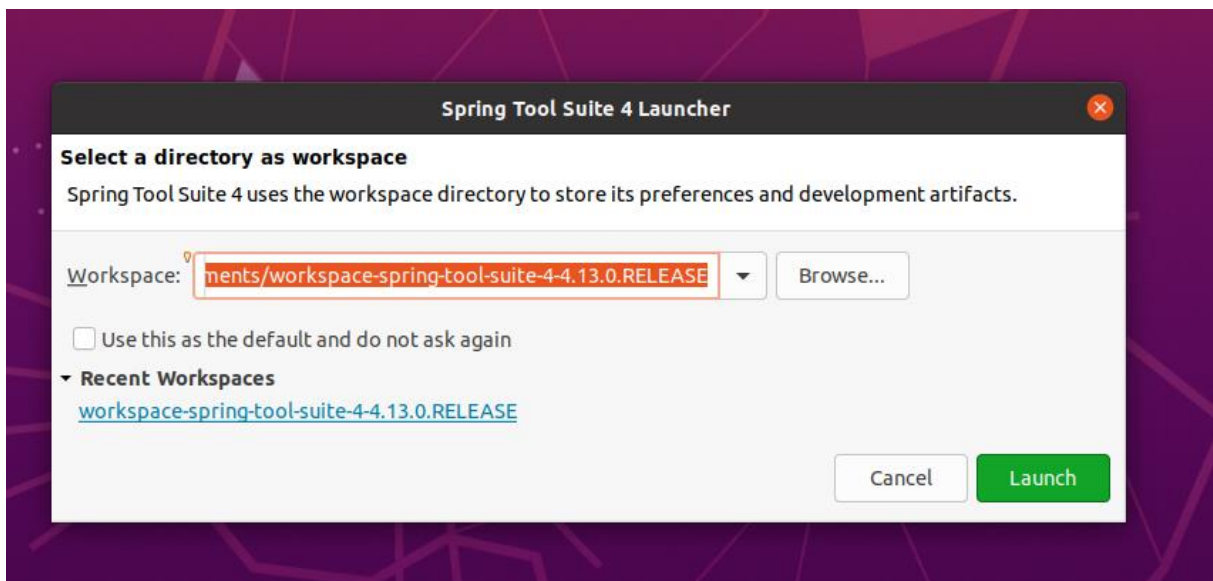
0.2) Technologies utilisées :

Les technologies utilisées seront VirtualBox, Spring boot (avec l'IDE spring tools suite 4), Docker et Kubernetes Engine

Pour utiliser Docker et créer un conteneur sur lequel pourra exécuter notre application, nous utiliserons une machine virtuelle Linux dont on aura téléchargée l'image et que l'on aura configurée et exécuté depuis VirtualBox.



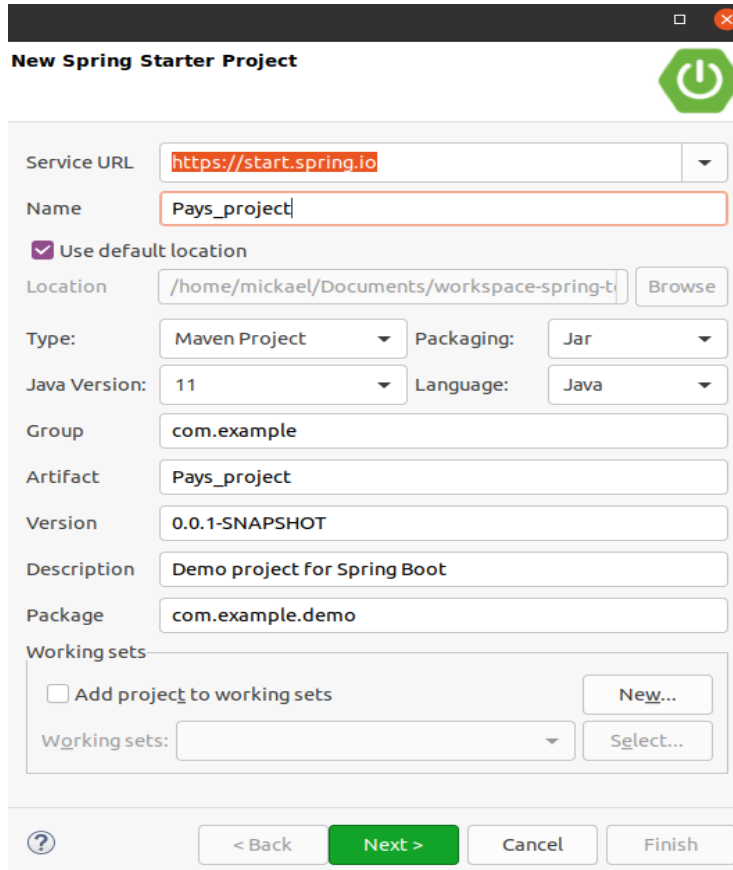
Puis nous lancerons l'IDE Spring tools suite 4 directement depuis la machine virtuelle Ubuntu.



1) Création d'une application web Spring-Boot

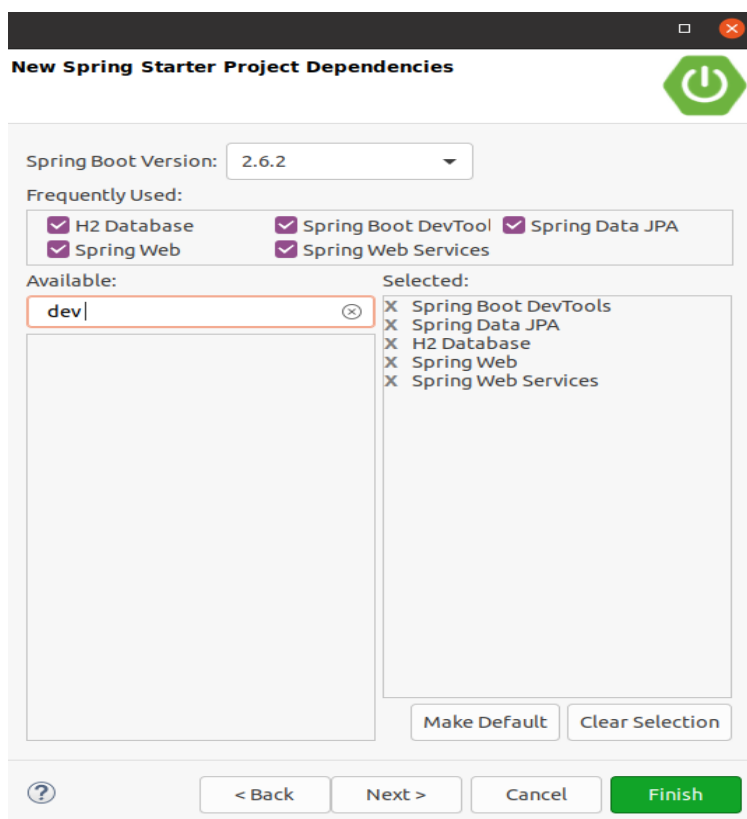
Pour créer un projet spring boot on va ouvrir l'outil de programmation Spring tools suite 4, puis on clique sur file->new->spring starter project.

Puis on choisit de créer un projet Maven et on entre le nom du projet.



The screenshot shows the 'New Spring Starter Project' dialog box. The 'Service URL' is set to 'https://start.spring.io'. The 'Name' field contains 'Pays_project'. The 'Use default location' checkbox is checked. The 'Location' is '/home/mickael/Documents/workspace-spring-ti'. The 'Type' is 'Maven Project', 'Packaging' is 'Jar', 'Java Version' is '11', and 'Language' is 'Java'. The 'Group' is 'com.example', 'Artifact' is 'Pays_project', 'Version' is '0.0.1-SNAPSHOT', 'Description' is 'Demo project for Spring Boot', and 'Package' is 'com.example.demo'. The 'Working sets' section has an unchecked checkbox for 'Add project to working sets'. At the bottom, there are buttons for '< Back', 'Next >', 'Cancel', and 'Finish'.

On ajoute les dépendances suivantes et on clique sur Finish :



The screenshot shows the 'New Spring Starter Project Dependencies' dialog box. The 'Spring Boot Version' is '2.6.2'. Under 'Frequently Used', the following dependencies are checked: 'H2 Database', 'Spring Web', 'Spring Boot DevTool', and 'Spring Data JPA'. The 'Available:' list contains 'dev'. The 'Selected:' list contains 'Spring Boot DevTools', 'Spring Data JPA', 'H2 Database', 'Spring Web', and 'Spring Web Services'. At the bottom, there are buttons for 'Make Default', 'Clear Selection', '< Back', 'Next >', 'Cancel', and 'Finish'.

2) Architecture du projet

Notre application sera divisée en plusieurs classes.

Chacune des classes sera située dans un répertoire spécifique. La classe principale servira à l'exécution du programme.

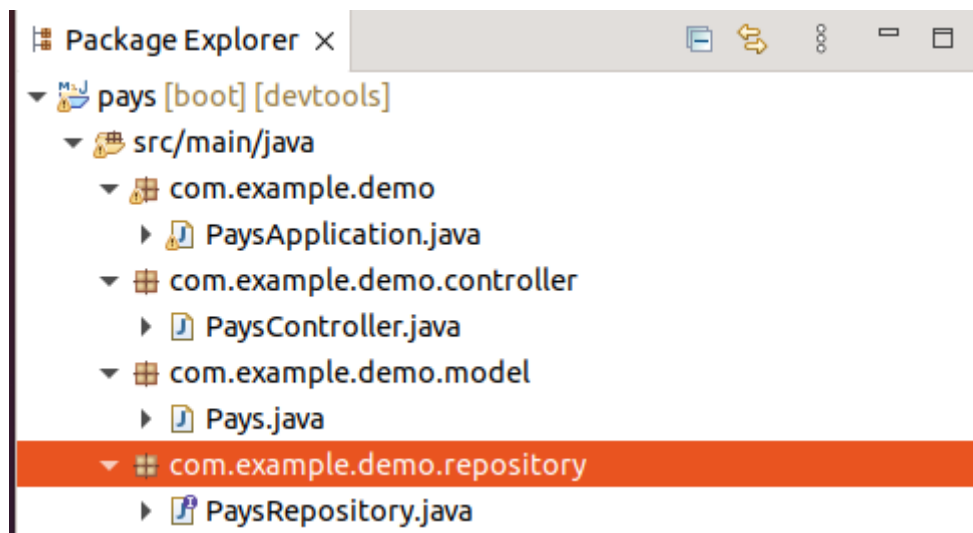
La classe intitulée « Pays » servira à définir l'objet de type pays. Cette classe joue le rôle d'entité, c'est-à-dire qu'elle représente une table d'une base de données.

La classe intitulée Pays Controller permettra de récupérer les requêtes http du client et de fournir les réponses associées. Cette classe contiendra l'annotation `@RestController`.

L'interface « PaysRepository » est une interface permettant de communiquer avec la source de données.

La classe principale « PaysApplication » permettra d'insérer des données dans la base de données d'une part et servira à l'exécution de l'application.

Voici ci-dessous un schéma de l'architecture des classes du projet.



- 2.a) La classe Pays

La classe Pays se caractérise par l'annotation `@Entity` pour indiquer que la classe correspond à une table de la base de données.

Un Pays sera décrit par son id, son nom, sa capitale, sa démographie, sa superficie, sa catégorie (Pays développé, Pays émergent, Pays peu développé, Pays très pauvre) et enfin son continent.

Chaque attribut est caractérisé par l'annotation `@Column` indiquant qu'on a à faire à une colonne d'une table de la base de données.

Nous aurons plusieurs constructeurs de cet objet Pays :

Remarque: nous n'utiliserons pas forcément tous ces constructeurs, certains ont été établis pour simplifier au plus possible la création d'un objet Pays.

On a donc un constructeur qui permet de créer un objet Pays uniquement à partir de son nom et en laissant les autres attributs comme vide.

Le 2^{ème} constructeur permettra de créer un objet Pays à partir de son nom et du continent auquel il appartient.

Un 3^{ème} constructeur servira à créer un objet Pays à partir de son nom, sa catégorie et son continent.

Un 4^{ème} constructeur servira à créer un objet pays à partir de tous les attributs de la classe.

Enfin le dernier servira à créer un objet pays sans aucun attribut.

Nous avons généré les getters et les setters de chacun des attributs. Et nous avons redéfini la méthode toString().

- 2.b) L'interface PaysRepository

Cette interface permet de communiquer avec la base de données.

- 2.c) La classe PaysController

La classe Pays Controller se distingue par ses annotations **@RestController** indiquant que cette classe reçoit les requêtes clientes http et s'assure d'y répondre.

```
@Service
@RestController
@RequestMapping("pays/")
public class PaysController {

    @Autowired
    PaysRepository repository;
```

On ajoute l'annotation **@RestController** pour indiquer qu'il s'agit d'une classe Controller donc qui va recevoir les requêtes du web liées au port 8080 ou à d'autres ports, les traiter et renvoyer le résultat au format JSON.

L'annotation **@RequestMapping** permet d'indiquer le format de la requête cliente.

Toute requête cliente vers notre application (avec l'application exécutée depuis Spring Tools Suite) commencera par :

<http://localhost:8080/pays/>

Nous avons enfin créé une instance de Pays Repository pour pouvoir y appliquer toutes les méthodes propres à la gestion des données dans ce repository.

Fonctionnalités du projet :

La classe Pays Controller est composée d'un ensemble de méthodes qui permettent de traiter les requêtes utilisateurs. Ces requêtes seront du type GET, POST , PUT et DELETE.

Fonction 1: récupérer la liste de tous les Pays au format JSON sur un navigateur :

```
@GetMapping("")  
public List<Pays> getAll(){  
    return this.repository.findAll();  
}
```

Ici @GetMapping nous sert à indiquer que nous voulons récupérer des données.

Ce qui contenu dans cette annotation sert à indiquer le chemin de ce que l'on veut récupérer. Le début du chemin de la requête est indiqué par @RequestMapping et la suite du chemin sera indiqué par @GetMapping.

L'URL de la requête pour avoir les informations sur tous les pays est donc :

<http://localhost:8080/pays/>

Nous allons dans la prochaine section vérifier le résultat d'exécution de chacune de ces requêtes.

Fonction 2 : obtenir les informations sur un pays spécifique

```
@GetMapping("nom/{nom}")  
public Pays getPays(@PathVariable(value="nom") String nom) {
```

La méthode ci-dessous nous permet de récupérer les informations sur un seul pays dont le nom sera spécifié dans l'URL. @PathVariable nous sert à indiquer que nous avons une variable dans l'URL et que le résultat d'exécution de cette méthode et de la requête utilisateur dépend de cette variable qui sera utilisée dans la méthode.

La requête correspondante pour obtenir les informations par exemple sur le pays nommé CHINE est de la forme :

<http://localhost:8080/pays/nom/CHINE>

Fonction 3 : récupérer tous les pays d'un continent

La méthode ci-dessous nous permet de récupérer une liste de pays se situant dans un continent spécifié dans l'URL.

```
@GetMapping("continent/{continent}")  
public List<Pays> getPaysDuContinent(@PathVariable(value="continent") String continent) {
```

@GetMapping indique le format de l'URL à spécifier pour exécuter ce type de requête.

@PathVariable permet d'indiquer à la méthode, le continent avec lequel on veut afficher toutes les informations de tous les pays de ce continent.

La requête correspondante pour avoir les informations de tous les pays d'un continent (par exemple ici l'Asie) passé en paramètre de la requête est :

<http://localhost:8080/pays/continent/ASIE>

Fonction 4: récupérer tous les pays d'une catégorie donnée.

```
@GetMapping("categorie/{categorie}")  
public List<Pays> getPaysDeCategorie(@PathVariable(value="categorie") String categorie){
```

Cette méthode fonctionne selon le même principe que la fonction 2. Elle nous permet de récupérer tous les pays correspondant à un niveau de richesse ou de pauvreté donné.

Les catégories sont : pays développé, pays en développement, pays émergent, pays pauvre, pays très pauvre.

Par exemple la requête pour avoir tous les pays les plus pauvres de la planète est :

<http://localhost:8080/pays/categorie/pays très pauvre>

Fonction 5: ajouter un pays à partir de son nom

Pour ajouter un pays à la base de données nous allons nous servir de l'annotation `@PostMapping` servant à indiquer nous allons ajouter un nouvel objet Pays. Nous allons créer un nouvel objet Pays à partir de son nom.

```
@PostMapping("ajouter/nom/")  
public Pays ajouterPays(@RequestBody String nom) {
```

Attention cette fois-ci la variable liée au nom du pays ne sera pas indiquée dans l'URL. Nous allons pour indiquer le pays dans une requête http, utiliser l'outil Postman dont nous décrirons l'utilisation plus tard.

`@RequestBody` constitue le corps de la requête, elle servira donc à indiquer la variable directement dans l'outil Postman. Cette variable du nom de pays permettra de créer un objet de type Pays uniquement à partir de son nom et sans toutes les informations qui devraient lui être associées.

Nous verrons plus tard la structure de la requête dans Postman.

Fonction 6 : Ajouter un pays à partir de toutes ses informations

```
@PostMapping("ajouter/")  
public Pays ajouterPays(@RequestBody Pays pays) {
```

Nous pourrions ajouter un pays en entrant manuellement sur Postman toutes les informations qui lui sont associées. La méthode va créer à partir de ces informations un objet de type Pays et l'ajouter à la base de données.

Fonction 7 : supprimer tous les pays

```
@DeleteMapping("tout supprimer/")  
public List<Pays> supprimer() {
```

Pour supprimer des données dans la base de données, nous utiliserons l'annotation @DeleteMapping qui servira à indiquer que nous voulons supprimer des données.

Nous utiliserons également Postman pour exécuter ce type de requête.

Attention il est à noter que lorsqu'on fait supprimer toutes les données, la requête ne consiste qu'à afficher qu'une instance de la base de données initiale à laquelle on aura supprimé toutes ces données. En effet, lorsqu'on relance le service web Spring et que nous réeffectuons la même requête on pourra constater que toutes les données sont présentes.

Fonction 8 : supprimer un pays à partir de son nom

```
@DeleteMapping("supprimer/")  
public List<Pays> supprimerPays(@RequestBody String nom) {
```

Pour supprimer un pays à partir de son nom nous allons utiliser l'annotation @DeleteMapping et l'annotation @RequestBody qui servira à saisir un nom de pays depuis Postman pour que ce pays soit passé en paramètre de fonction et supprimé.

Fonction 9 : supprimer tous les pays d'un continent

```
@DeleteMapping("supprimer/continent/")  
public List<Pays> supprimerContinent(@RequestBody String continent){
```

@RequestBody permet d'indiquer dans Postman le continent avec lequel on voudra supprimer tous les pays de la base de données.

Fonction 10 : supprimer tous les pays d'une catégorie spécifiée

```
@DeleteMapping("supprimer/categorie/")  
public List<Pays> supprimerPaysDeCategorie(@RequestBody String categorie){
```

Même raisonnement que pour la fonction 9.

Fonction 11: modifier la démographie d'un pays

```
@PutMapping("/modifier demographie/{nom}")  
public Pays modifierNomPays(@PathVariable String nom,@RequestBody String demographie){
```

Cette méthode permet de modifier la démographie d'un pays dont le nom est passé en argument.

@PutMapping permet d'indiquer qu'on veut modifier des données.

@PathVariable permet d'indiquer le nom du pays dont on veut modifier la démographie.

@RequestBody permet d'indiquer la nouvelle démographie du pays correspondant.

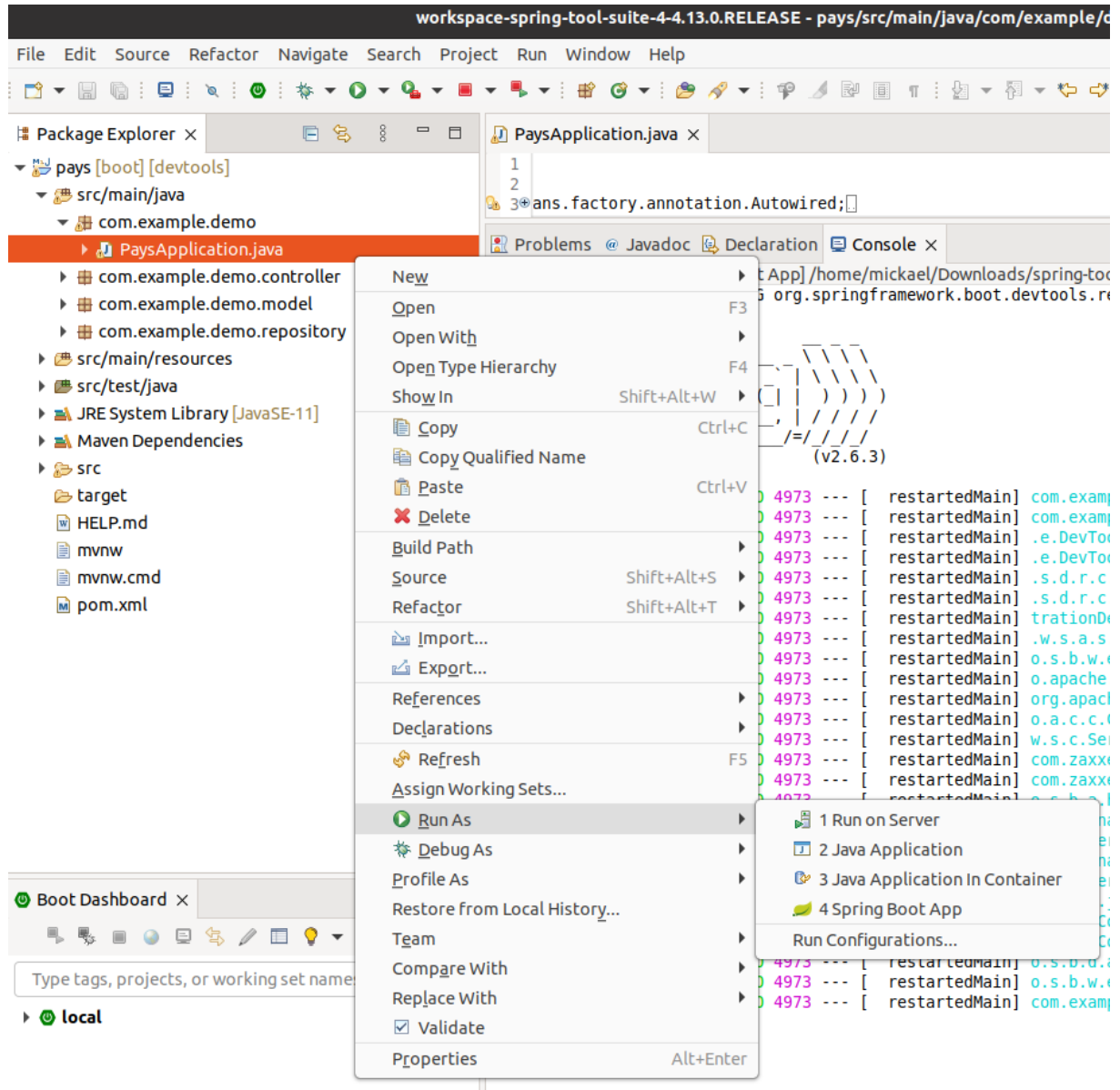
Fonction 12 : modifier la catégorie d'un pays

```
@PutMapping("/modifier categorie/{nom}")  
public Pays modifierCategoriePays(@PathVariable String nom,@RequestBody String categorie){
```

Même principe que pour la méthode 11, mais on modifie la catégorie d'un pays.

- **2.c) La classe principale (Pays Application)**

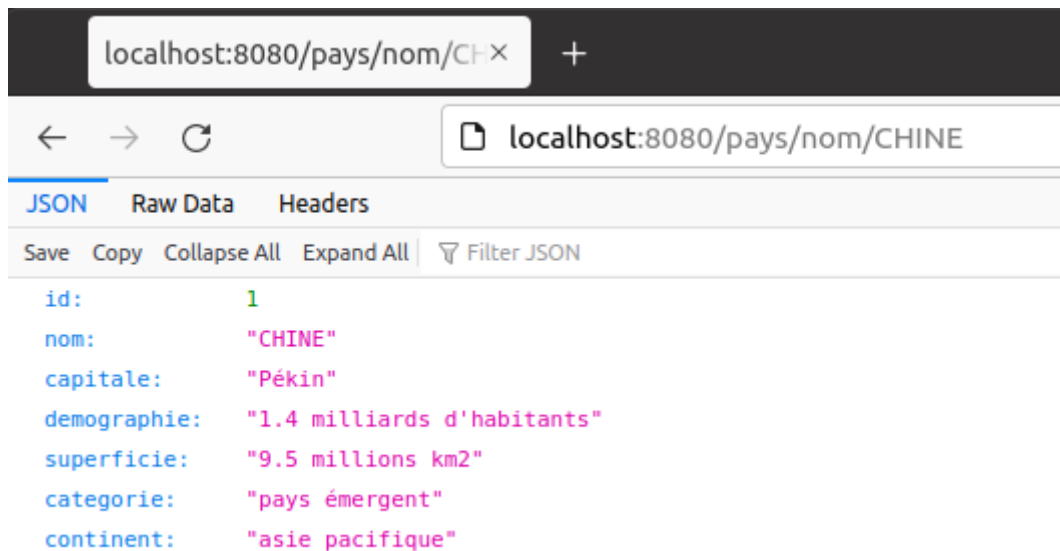
Pour exécuter notre application spring boot on réalise un clic droit sur la classe principale main du projet (ici PaysApplication) , puis on clique sur **Run as** et enfin sur **Spring Boot App**.



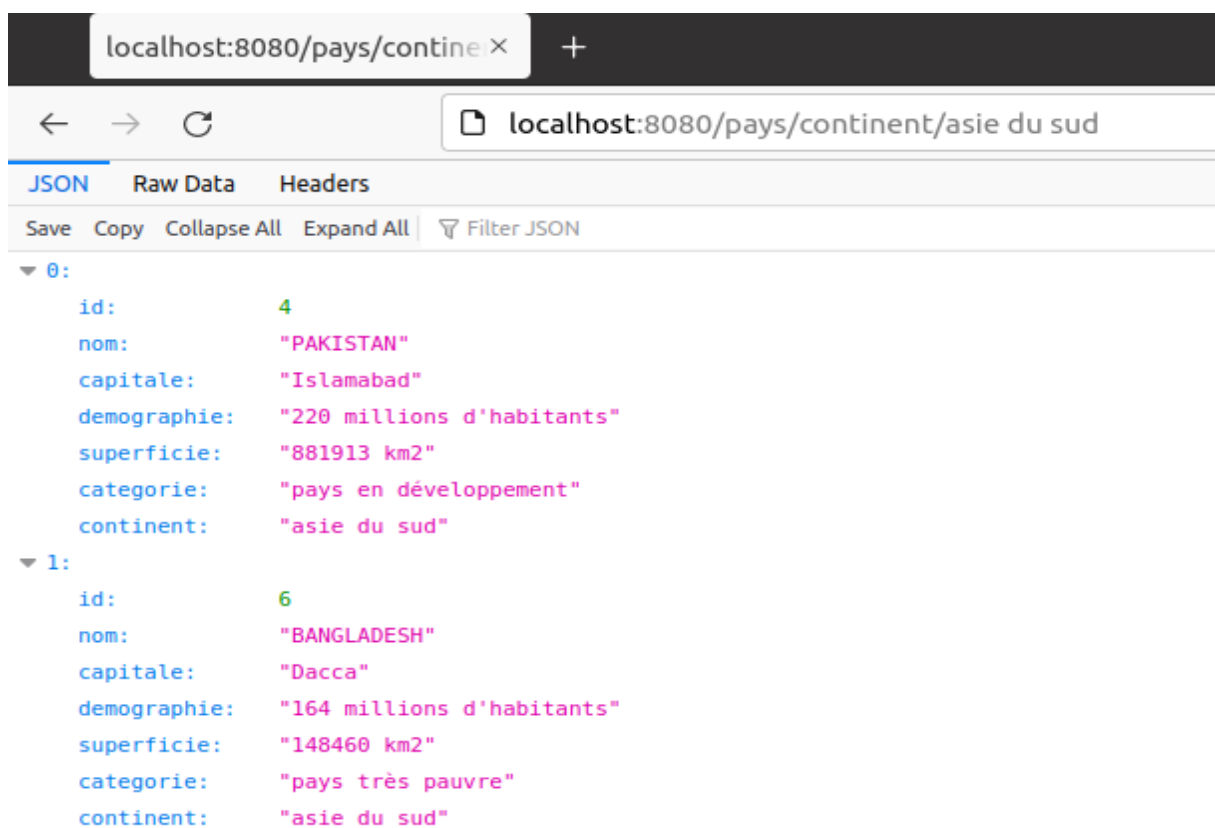
Notre application web service s'est exécuté et il nous est possible de la tester en ouvrant un navigateur. Voici ci-dessous le résultat de la requête <http://localhost:8080/pays/> qui nous permet d'afficher la liste des pays. On remarque que les données sont affichées au format JSON.



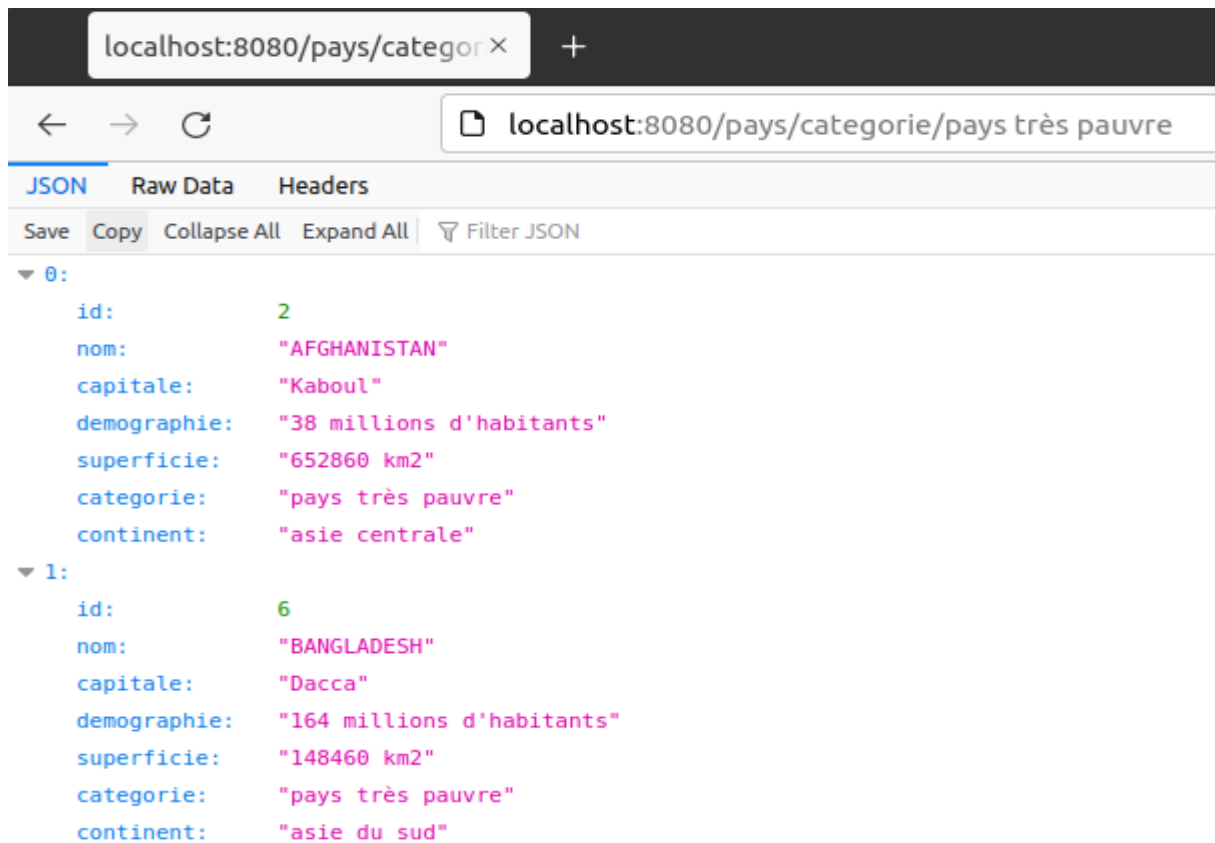
La requ  te <http://localhost:8080/pays/CHINE> nous permet d'afficher les informations relatives    un pays donn  , ici par exemple les informations sur le pays CHINE.



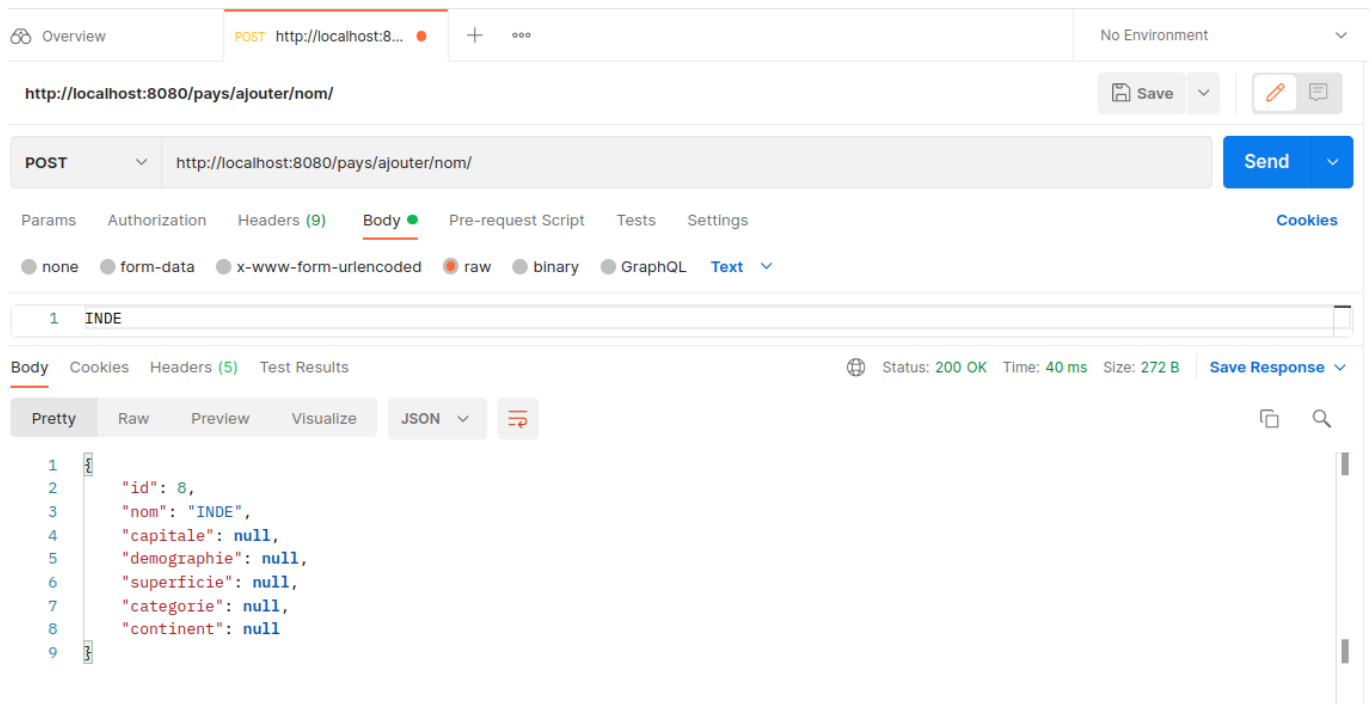
La requête <http://localhost:8080/continent/nom du continent> nous permet d'afficher tous les pays appartenant à un continent donné (ou une région d'un continent).



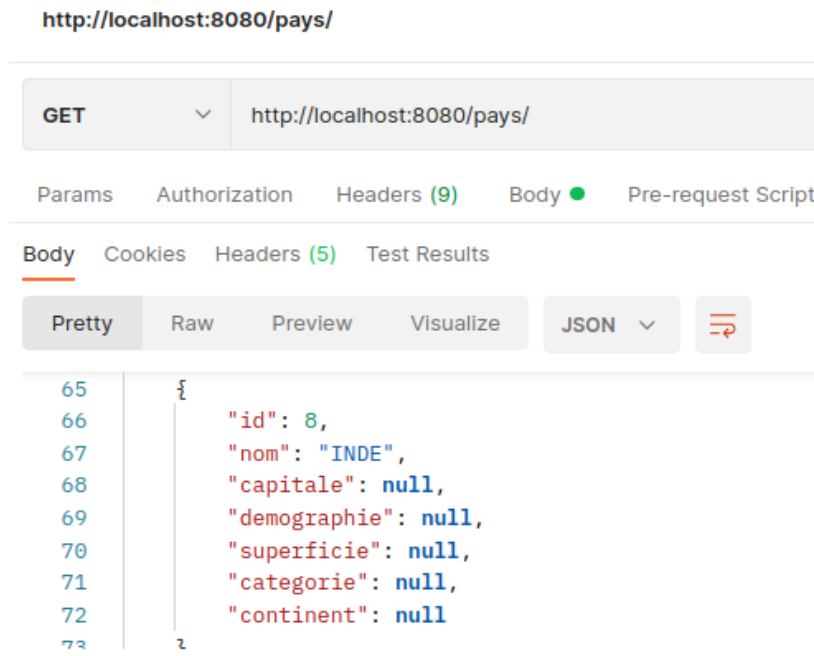
La requête <http://localhost:8080/categorie/nom de la catégorie> nous permet d'avoir les informations de tous les pays qui font pays partie d'une catégorie donnée.



Pour les requêtes POST, PUT et DELETE nous allons utiliser l'outil Postman, car elle nous permettra de choisir manuellement le type de requête http à exécuter.



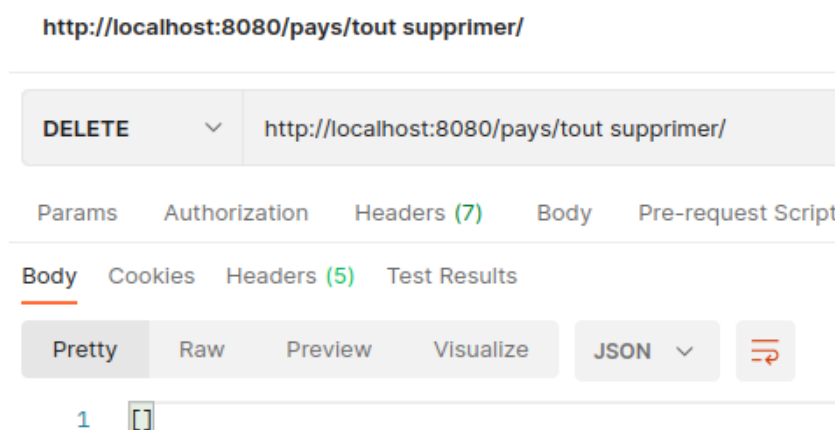
La requête <http://localhost:8080/pays/ajouter/nom/> avec le nom INDE dans le corps de la requête (champ body) et le choix de la méthode POST nous à permis d'ajouter un nouveau pays dans notre liste. Pour le vérifier, nous pouvons à nouveau demander la liste de tous les pays avec GET et on voit le nouveau pays INDE ci-dessous ajouté en fin de liste.

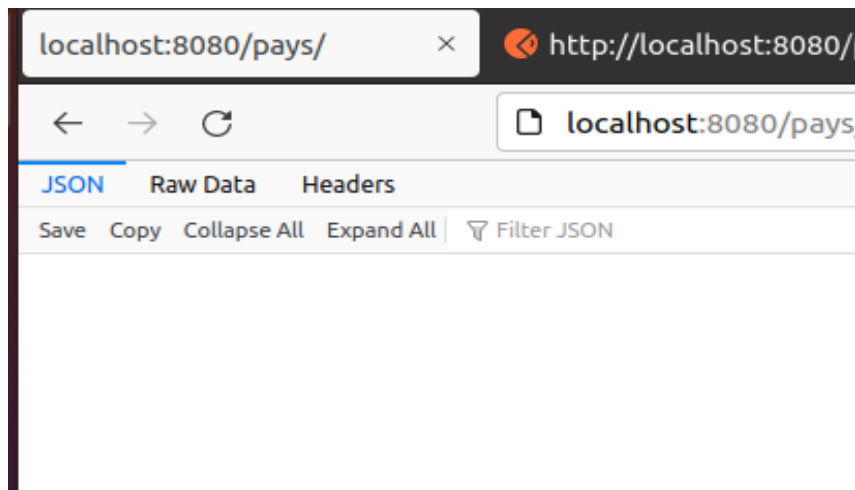


La requête <http://localhost:8080/supprimer/tout/> nous permet de supprimer tous les pays de la liste.

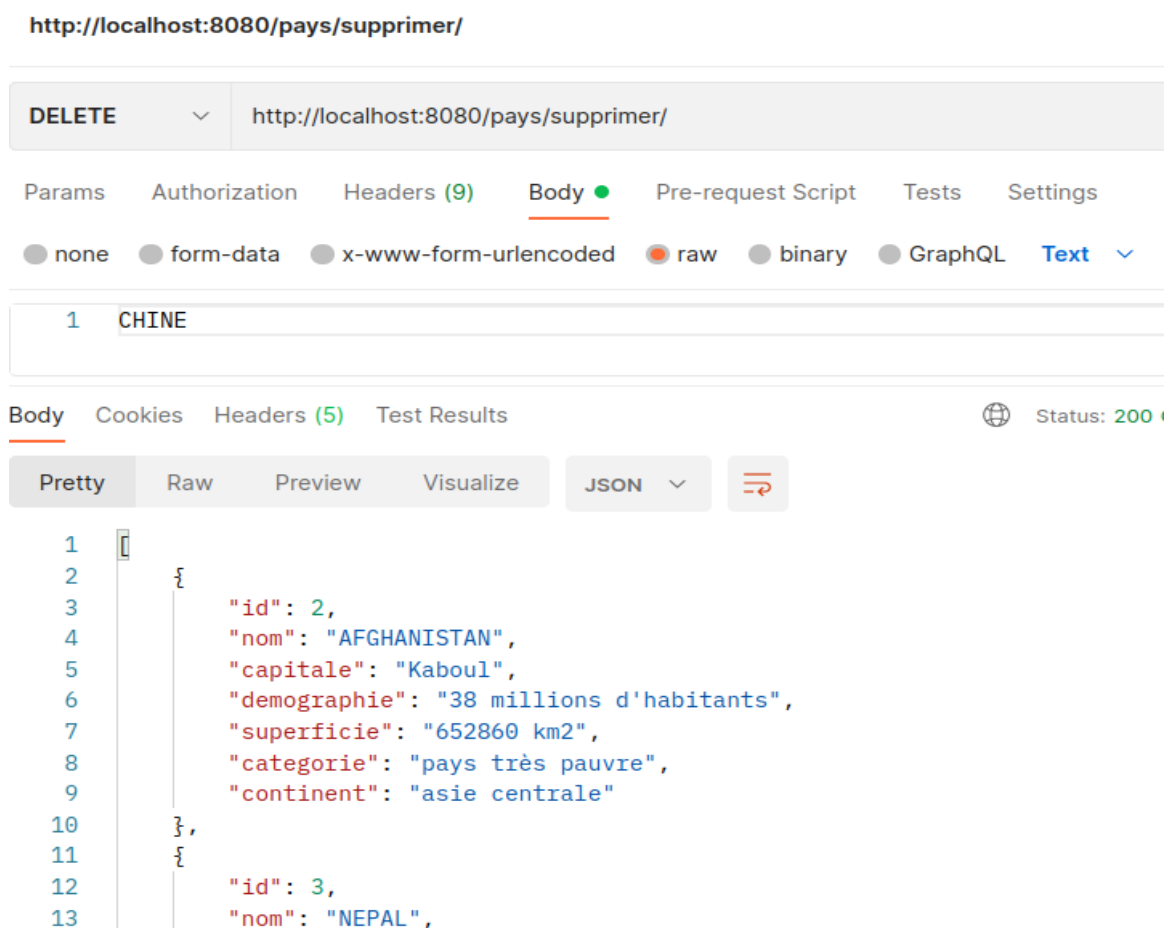
Il faudra relancer l'application pour obtenir à nouveau les pays de la liste et exécuter d'autres requêtes.

Nous pouvons voir que le résultat d'exécution de la requête est une liste vide.





Pour supprimer un pays à partir de son nom, on tape la requête : <http://localhost:8080/pays/supprimer/> et on choisit l'option DELETE et on indique dans Body le nom du pays que l'on veut supprimer de notre liste. Le résultat d'exécution de cette requête nous montre que le pays CHINE qui était le 1^{er} élément de la liste, n'est plus dans la liste car il a été supprimé.



Si on veut ensuite supprimer tous les pays d'un continent on utilise la requête <http://localhost:8080/pays/supprimer/continent/> et on choisit l'option DELETE et on indique dans body le nom du continent (ou une région d'un continent).

On voit ci-dessous qu'on a supprimé tous les pays de l'Asie-centrale.

DELETE ▼ http://localhost:8080/pays/supprimer/continent/

Params Authorization Headers (9) **Body ●** Pre-request Script

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary

1 asie centrale

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼

```
1 [
2   {
3     "id": 4,
4     "nom": "PAKISTAN",
5     "capitale": "Islamabad",
```

Pour supprimer tous les pays d'une catégorie donnée, on applique le même principe et on indique dans body le nom de la catégorie des pays à supprimer.

Par exemple, nous avons supprimé de notre liste tous les pays en développement et on voit qu'ils n'apparaissent plus dans le résultat d'exécution de cette requête.

http://localhost:8080/pays/supprimer/categorie/

DELETE ▼ http://localhost:8080/pays/supprimer/categorie/

Params Authorization Headers (9) **Body ●** Pre-request Script Tests

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ Graph

1 pays en développement

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼

```
1 [
2   {
3     "id": 5,
4     "nom": "ARABIE-SAUDITE",
```

Voici ci-dessous la procédure et les résultats d'exécution des requêtes PUT qui permettront de modifier la démographie d'un pays et de modifier la catégorie d'un pays.

<http://localhost:8080/pays/modifier categorie/CHINE>

PUT

▼

http://localhost:8080/pays/modifier categorie/CHINE

ParamsAuthorizationHeaders (9)Body●Pre-request ScriptTestsSettings

● none● form-data● x-www-form-urlencoded● raw● binary● GraphQLText ▼

1 pays riche

BodyCookiesHeaders (5)Test Results

⌚ Status: 200 OK Tim

PrettyRawPreviewVisualizeJSON ▼

1 {
2 "id": 1,
3 "nom": "CHINE",
4 "capitale": "Pékin",
5 "demographie": "1.4 milliards d'habitants",
6 "superficie": "9.5 millions km2",
7 "categorie": "pays riche",
8 "continent": "asie pacifique"
9 }

<http://localhost:8080/pays/modifier demographie/BANGLADESH>

PUT

▼

http://localhost:8080/pays/modifier demographie/BANGLADESH

ParamsAuthorizationHeaders (9)Body●Pre-request ScriptTestsSettings

● none● form-data● x-www-form-urlencoded● raw● binary● GraphQLText

1 300 millions d'habitants

BodyCookiesHeaders (5)Test Results

⌚ Status:

PrettyRawPreviewVisualizeJSON ▼

1 {
2 "id": 6,
3 "nom": "BANGLADESH",
4 "capitale": "Dacca",
5 "demographie": "300 millions d'habitants",
6 "superficie": "148460 km2",
7 "categorie": "pays très pauvre",
8 "continent": "asie du sud"
9 }

3) Exécuter notre application dans un conteneur docker depuis une image Docker

- **3.a) Création d'un Docker File**

Pour exécuter une image docker liée à notre projet dans un conteneur Docker, nous devons d'abord créer un docker file associé à notre projet.

Un Docker file est un document texte qu'on a placé à la racine du projet et qui contient toutes les commandes qu'on pourrait appeler en ligne de commande pour créer une image docker. Toutes les instructions dans le Docker file seront donc exécutées pour créer une image docker.

Dans le docker file l'instruction FROM est la première instruction qui définit l'image docker utilisé pour les autres instructions. Ici on utilise l'image docker d'openjdk version 11.

EXPOSE nous permet d'indiquer le numéro de port sur lequel sera exposé l'application, donc sur quel port réseau le conteneur docker écoute .

ADD nous permet d'indiquer le chemin dans lequel nous allons placer notre fichier d'extension .jar qui servira à créer notre image docker.

ENTRYPOINT nous permet de configurer un conteneur qui fonctionnera comme un exécutable.

Dans le fichier pom.xml qui contient toutes les dépendances du projet, il faudra ajouter la ligne indiquant le nom du fichier.jar généré à l'aide du Docker File.



```
1 FROM openjdk:11
2 EXPOSE 8080
3 ADD target/pays-docker.jar pays-docker.jar
4 ENTRYPOINT ["java", "-jar", "/pays-docker.jar"]
5
6
```

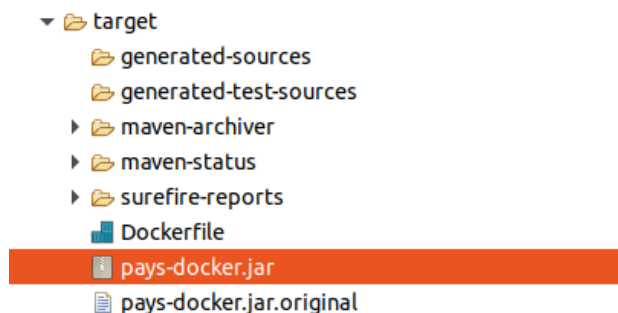
Pour créer un fichier docker-pays.jar qui nous servira à créer l'image docker, on réalise un clic droit sur le projet-> Run As-> Maven Install

Résultat :

```
W20190804111111
:: Spring Boot :: (v2.6.3)

022-01-26 20:52:39.486 INFO 10933 --- [main] com.example.demo.PaysApplicationTests : Starting PaysApplicationTests using Java 17.0
022-01-26 20:52:39.489 INFO 10933 --- [main] com.example.demo.PaysApplicationTests : No active profile set, falling back to default
022-01-26 20:52:41.285 INFO 10933 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in
022-01-26 20:52:41.614 INFO 10933 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 2
022-01-26 20:52:42.427 INFO 10933 --- [main] trationDelegatesBeanPostProcessorChecker : Bean 'org.springframework.ws.config.annotatio
022-01-26 20:52:42.708 INFO 10933 --- [main] w.s.a.s.AnnotationActionEndpointMapping : Supporting [WS-Addressing August 2004, WS-Add
022-01-26 20:52:45.103 INFO 10933 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
022-01-26 20:52:45.619 INFO 10933 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
022-01-26 20:52:45.804 INFO 10933 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [na
022-01-26 20:52:45.893 INFO 10933 --- [main] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.6.4.F
022-01-26 20:52:46.353 INFO 10933 --- [main] org.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations (5
022-01-26 20:52:46.701 INFO 10933 --- [main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.diale
022-01-26 20:52:47.957 INFO 10933 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation:
022-01-26 20:52:47.963 INFO 10933 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for pers
022-01-26 20:52:49.469 WARN 10933 --- [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default
022-01-26 20:52:50.957 INFO 10933 --- [main] com.example.demo.PaysApplicationTests : Started PaysApplicationTests in 12.221 second
INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 14.686 s - in com.example.demo.PaysApplicationTests
022-01-26 20:52:51.807 INFO 10933 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persiste
022-01-26 20:52:51.820 INFO 10933 --- [ionShutdownHook] SchemaDropperImpl$DelayedDropActionImpl : HHH000477: Starting delayed evictData of sche
022-01-26 20:52:51.911 INFO 10933 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
022-01-26 20:52:51.952 INFO 10933 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
INFO] Results:
INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
INFO]
INFO] --- maven-jar-plugin:3.2.2:jar (default-jar) @ pays ---
INFO] Building jar: /home/mickael/Documents/workspace-spring-tool-suite-4-4.13.0.RELEASE/pays/target/pays-docker.jar
INFO]
INFO] --- spring-boot-maven-plugin:2.6.3:repackage (repackage) @ pays ---
INFO] Replacing main artifact with repackaged archive
INFO]
INFO] --- maven-install-plugin:2.5.2:install (default-install) @ pays ---
INFO] Installing /home/mickael/Documents/workspace-spring-tool-suite-4-4.13.0.RELEASE/pays/target/pays-docker.jar to /home/mickael/.m2/repository
INFO] Installing /home/mickael/Documents/workspace-spring-tool-suite-4-4.13.0.RELEASE/pays/pom.xml to /home/mickael/.m2/repository/com/example/pa
INFO]
INFO] BUILD SUCCESS
INFO]
TAGS]
```

On voit que le fichier d'extension .jar a bien été ajouté dans le dossier target.



• 3.b) création d'une image docker associé à notre application (Docker build)

Le fichier pays-docker.jar a été ajouté à notre dossier target. On pourra donc ensuite construire une image docker grâce à ce .jar. On doit d'abord installer docker sur notre machine virtuelle Linux.

Pour cela on ouvre un terminal sur la machine virtuelle, puis on tape la commande :

sudo apt install docker.io

```
mickael@mickael-VirtualBox: ~/Documents/workspace-sprin...
mickael@mickael-VirtualBox:~/Documents/workspace-spring-tool-suite-4-4.13.0.RELE
ASE/pays$ sudo apt install docker.io
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd git git-man liberror-perl pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools btrfs-progs cgroupfs-mount | cgroup-lite debootstrap
  docker-doc rinse zfs-fuse | zfsutils git-daemon-run | git-daemon-sysvinit
  git-doc git-el git-email git-gui gitk gitweb git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  bridge-utils containerd docker.io git git-man liberror-perl pigz runc
  ubuntu-fan
0 upgraded, 9 newly installed, 0 to remove and 189 not upgraded.
Need to get 79,6 MB of archives.
After this operation, 398 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://fr.archive.ubuntu.com/ubuntu focal/universe amd64 pigz amd64 2.4-1
[57,4 kB]
Get:2 http://fr.archive.ubuntu.com/ubuntu focal/main amd64 bridge-utils amd64 1.
6-2ubuntu1 [30,5 kB]
Get:3 http://fr.archive.ubuntu.com/ubuntu focal-updates/main amd64 runc amd64 1.
0.1-0ubuntu2~20.04.1 [4 155 kB]
mickael@mickael-VirtualBox:~/Documents/workspace-spring-tool-suite-4-4.13.0.RELE
```

Pour ensuite construire une image docker associé à notre projet cela on ouvre un terminal sur la machine virtuelle, puis on se place à la racine du projet et on tape la commande:

sudo docker build -t pays-docker.jar

```
mickael@mickael-VirtualBox: ~/Documents/workspace-spri...
mickael@mickael-VirtualBox:~/Documents/workspace-spring-tool-suite-4-4.13.0.RELE
ASE/pays$ sudo docker build -t pays-docker.jar .
Sending build context to Docker daemon 39.83MB
Step 1/4 : FROM openjdk:11
11: Pulling from library/openjdk
0c6b8ff8c37e: Pull complete
412caad352a3: Pull complete
e6d3e61f7a50: Pull complete
461bb1d8c517: Pull complete
e442ee9d8dd9: Pull complete
542c9fe4a7ba: Pull complete
4512bdd2598c: Pull complete
Digest: sha256:87cf7cc33e87ebbaa7f5961c4d1d3f84df66af41fbe03e983251e54ea743e6c8
Status: Downloaded newer image for openjdk:11
--> ae2fd7709d3b
Step 2/4 : EXPOSE 8080
--> Running in 8e3d8ccdb128
Removing intermediate container 8e3d8ccdb128
--> 6a1dde41e822
Step 3/4 : ADD target/pays-docker.jar pays-docker.jar
--> 160cb137c9d8
Step 4/4 : ENTRYPOINT ["java","-jar","/pays-docker.jar"]
--> Running in f56e8c1834ba
Removing intermediate container f56e8c1834ba
--> 2ecacf90a229
Successfully built 2ecacf90a229
mickael@mickael-VirtualBox:~/Documents/workspace-spring-tool-suite-4-4.13.0.RELE
```

-Sudo nous permet de passer en mode super-utilisateur (utilisateur root avec tous les privilèges).

-L'option build nous permet de construire une image docker.

-l'option -t indique que le target (fichier cible) ou se trouve le fichier .jar.

On voit en tapant la commande docker image ls que l'image docker associé à notre projet a bien été créé dans le repository local des images docker.


```
mickael@mickael-VirtualBox:~/Documents/workspace-spring-tool-suite-4-4.13.0.RELEASE/Pays$ sudo docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
pays-docker.jar      latest             1096c4656909       2 minutes ago      699MB
hello-docker.jar     latest             17022485f6f8       3 weeks ago        677MB
<none>               <none>             f7b8c3ba756c       3 weeks ago        122MB
openjdk              11                 5505a9a39df1       4 weeks ago        659MB
ubuntu               latest             ba6accedd29        3 months ago       72.8MB
hello-world          latest             feb5d9fea6a5       3 months ago       13.3kB
openjdk              8-jdk-alpine       a3562aa0b991       2 years ago        105MB
mickael@mickael-VirtualBox:~/Documents/workspace-spring-tool-suite-4-4.13.0.RELEASE/Pays$
```

• 3.c) Exécution de l'image docker locale dans un conteneur docker (docker run)

Une fois l'image docker associé à notre projet crée, nous pouvons exécuter cette image depuis un conteneur docker, en tapant la commande : `sudo docker run -p 9090:8080 pays-docker.jar`

Avec le port 9090 qui indique le port sur lequel on peut accéder à l'application et l'option -p indiquant le numéro de port.

```
sudo docker run -p 9090:8080 pays-docker.jar
```

Résultat de l'exécution :

```
mickael@mickael-VirtualBox:~/Documents/workspace-spring-tool-suite-4-4.13.0.RELEASE/Pays$ sudo docker run -p 9090:8080 pays-docker.jar

:: Spring Boot ::
(v2.6.2)

2022-01-19 11:05:29.818 INFO 1 --- [main] com.example.demo.PaysApplication : Starting PaysApplication v0.0.1-SNAPSHOT using Java 11.0.13 on 9691165228f6 with PID 1 (/pays-docker.jar started by root in /)
2022-01-19 11:05:29.831 INFO 1 --- [main] com.example.demo.PaysApplication : No active profile set, falling back to default profiles: default
2022-01-19 11:05:31.364 INFO 1 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-01-19 11:05:31.406 INFO 1 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 8 ms. Found 0 JPA repository interfaces.
2022-01-19 11:05:32.546 INFO 1 --- [main] trationDelegate$BeanPostProcessorChecker : Bean 'org.springframework.ws.config.annotation.DelegatingWsConfiguration' of type [org.springframework.ws.config.annotation.DelegatingWsConfiguration$$EnhancerBySpringCGLIB$$d9bb759a] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2022-01-19 11:05:32.741 INFO 1 --- [main] .w.s.a.s.AnnotationActionEndpointMapping : Supporting [WS-Addressing August 2004, WS-Addressing 1.0]
2022-01-19 11:05:33.518 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-01-19 11:05:33.558 INFO 1 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-01-19 11:05:33.560 INFO 1 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.56]
2022-01-19 11:05:33.697 INFO 1 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-01-19 11:05:33.703 INFO 1 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 3674 ms
2022-01-19 11:05:34.274 INFO 1 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2022-01-19 11:05:34.744 INFO 1 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2022-01-19 11:05:34.943 INFO 1 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2022-01-19 11:05:35.092 INFO 1 --- [main] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.6.3.Final
2022-01-19 11:05:35.507 INFO 1 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-01-19 11:05:35.821 INFO 1 --- [main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
2022-01-19 11:05:36.399 INFO 1 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2022-01-19 11:05:36.414 INFO 1 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-01-19 11:05:36.564 WARN 1 --- [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2022-01-19 11:05:37.509 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-01-19 11:05:37.566 INFO 1 --- [main] com.example.demo.PaysApplication : Started PaysApplication in 8.929 seconds (JVM running for 10.236s)
```

Nous allons donc pouvoir vérifier que notre l'image docker liée à notre application, s'exécute bien sur un conteneur Docker et qu'il nous sera possible d'exécuter les requêtes http précédentes sans utiliser l'IDE Spring tools suite 4.

Pour cela, on va chercher l'adresse IP de notre VM grâce à la commande **ifconfig**. Il faudra d'abord installer le package responsable de cette commande avec la commande : **sudo apt-install net-tools**

On pourra voir que notre adresse ip est : 172.17.0.1

```
mickael@mickael-VirtualBox: ~/Documents/workspace-sprin...
mickael@mickael-VirtualBox:~/Documents/workspace-spring-tool-suite-4-4.13.0.RELEASE/pays$ ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    inet6 fe80::42:feff:fe70:8f80 prefixlen 64 scopeid 0x20<link>
    ether 02:42:fe:70:8f:80 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 30 bytes 5379 (5.3 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

On va se servir de cette adresse IP et exécuter sur un navigateur la requête suivante:

<http://172.17.0.1:9090/pays/>

Cette URL contient l'adresse IP de notre machine suivi du numéro de port par lequel on peut accéder à l'application et enfin la requête utilisateur lié à l'application.

The screenshot shows a web browser window with the address bar displaying `172.17.0.1:9090/pays/`. The browser's developer tools are open, showing a JSON response with an array of country objects. The first object represents China, the second Afghanistan, and the third Nepal. Overlaid on the bottom right is a terminal window showing the command `sudo docker run -p 9090:8080 pays-docker.jar` and its output logs. The logs indicate that the application is starting successfully, bootstrapping Spring Data JPA repositories, and scanning for repository interfaces.

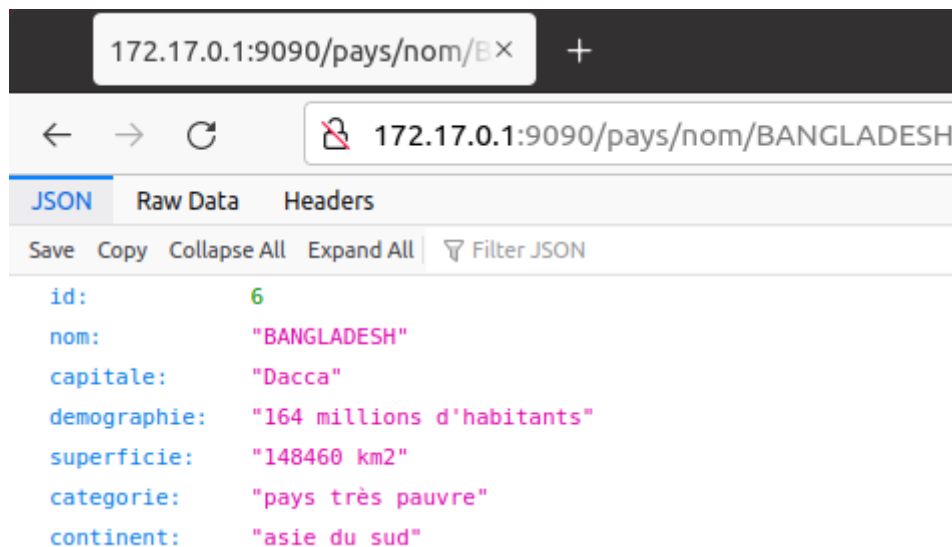
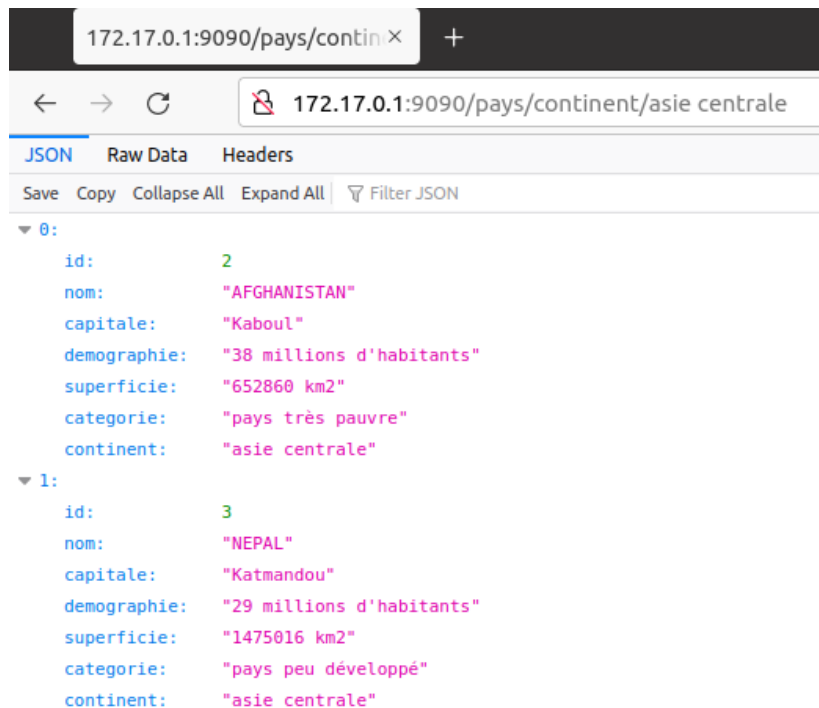
```
0:
  id: 1
  nom: "CHINE"
  capitale: "Pékin"
  demographie: "1.4 milliards d'habitants"
  superficie: "9.5 millions km2"
  categorie: "pays émergent"
  continent: "asie pacifique"
1:
  id: 2
  nom: "AFGHANISTAN"
  capitale: "Kaboul"
  demographie: "38 millions d'habitants"
  superficie: "652860 km2"
  categorie: "pays très pauvre"
  continent: "asie centrale"
2:
  id: 3
  nom: "NEPAL"
  capitale: "Katmandou"
  demographie: "29 millions d'habitants"
  superficie: "1475016 km2"
  categorie: "pays peu développé"
  continent: "asie centrale"
3:
```

```
mickael@mickael-VirtualBox: ~/Documents/workspace-spring-tool-suite-4-4.13.0.RELEASE/pays$ sudo docker run -p 9090:8080 pays-docker.jar
:: Spring Boot :: (v2.6.3)

2022-01-26 20:23:51.504 INFO 1 --- [main] com.ex
ication : Starting PaysApplication v0.0.1-SNAPSHOT u
n a6d44a45907f with PID 1 (/pays-docker.jar started by root
2022-01-26 20:23:51.506 INFO 1 --- [main] com.ex
ication : No active profile set, falling back to def
ult
2022-01-26 20:23:53.514 INFO 1 --- [main] .s.d.r
urationDelegate : Bootstrapping Spring Data JPA repositories
2022-01-26 20:23:53.659 INFO 1 --- [main] .s.d.r
urationDelegate : Finished Spring Data repository scanning i
PA repository interfaces.
2022-01-26 20:23:54.505 INFO 1 --- [main] tratio
rocessorChecker : Bean 'org.springframework.ws.config.annota
nfiguration' of type [org.springframework.ws.config.annotati
figuration$$EnhancerBySpringCGLIB$$eb12c1c6] is not eligible
```

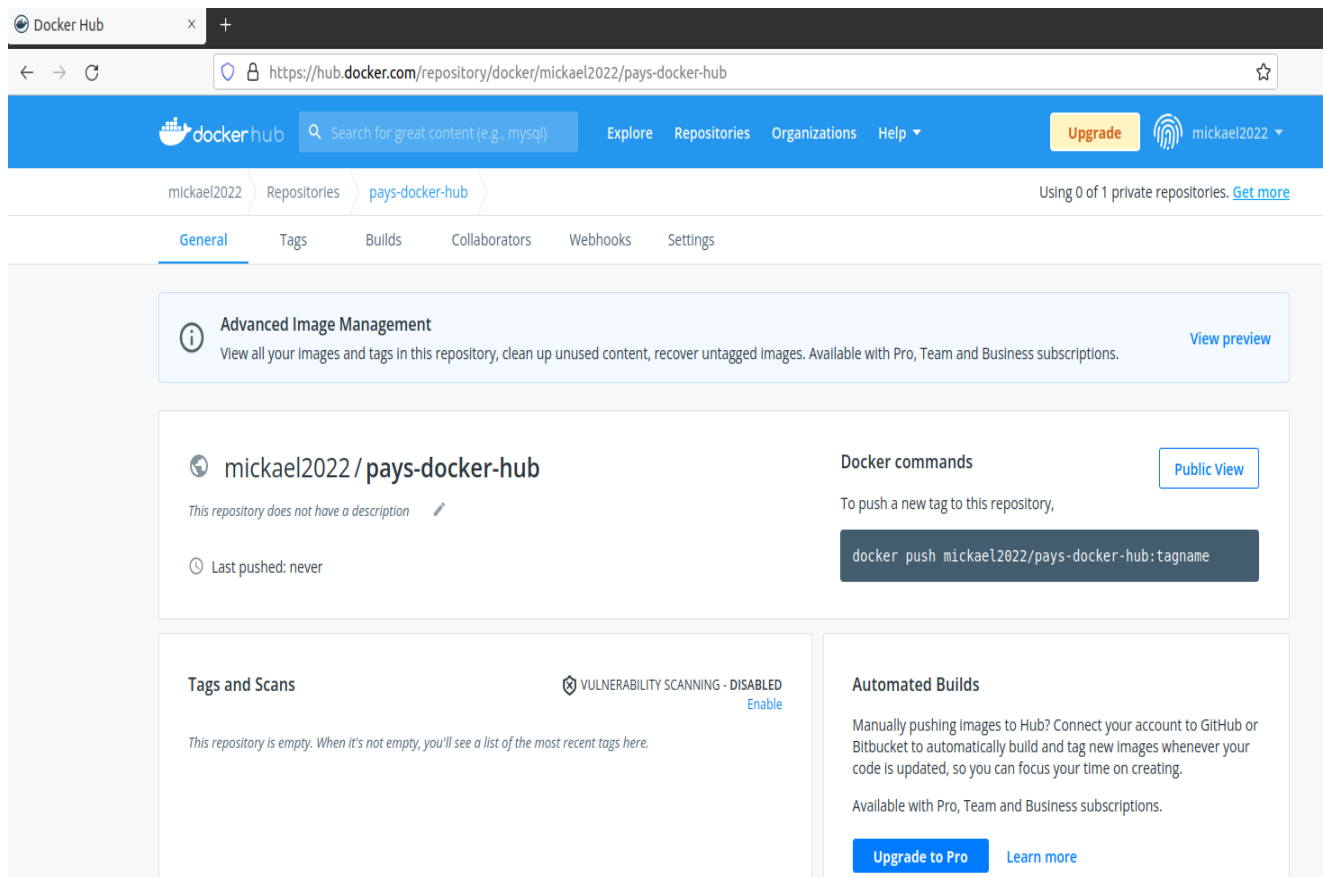
On voit que notre application s'est bien lancée depuis un conteneur Docker !!

On observe qu'on arrive à exécuter les mêmes requêtes et à avoir les mêmes résultats que si on avait exécuté l'application depuis notre IDE à la différence que l'application s'exécute désormais dans un conteneur Docker grâce à une image docker.



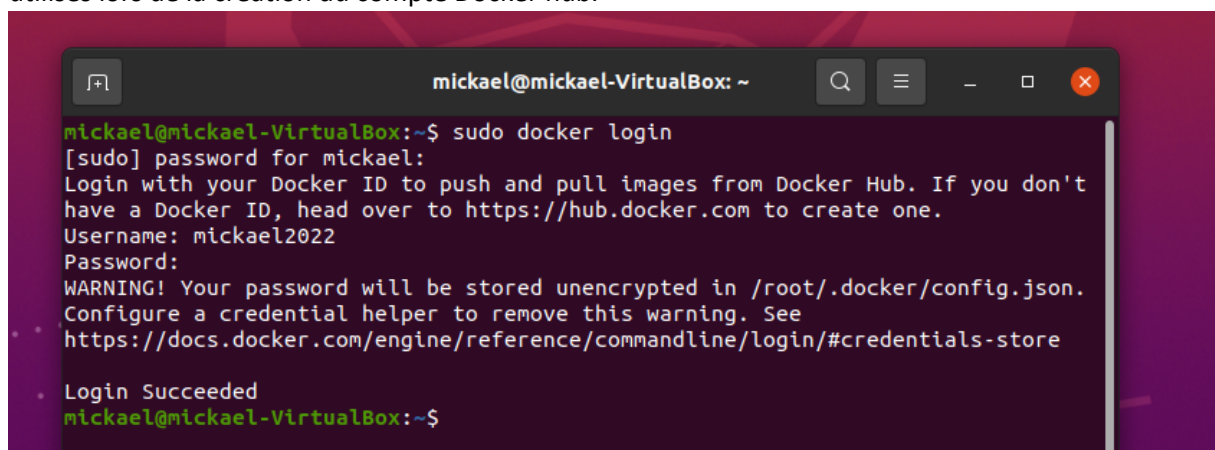
- **2.d) Envoi de l'image docker vers le Docker hub (docker push)**

Nous allons déposer notre image docker de l'application dans le Docker-hub. Pour cela, on doit d'abord créer un compte Docker, puis on doit créer un repository associé à notre compte. Ici notre repository va s'appeler pays-docker-hub. Le login de mon compte docker est mickael2022 et le chemin pour accéder au repository dans docker hub est mickael2022/pays-docker-hub.

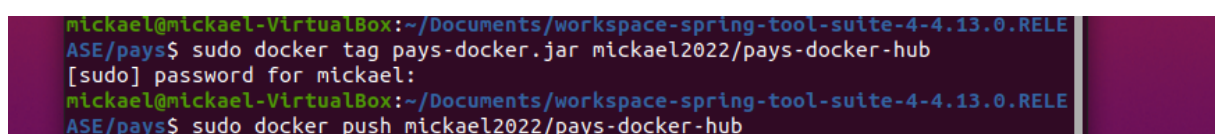


Puis nous allons nous connecter à notre compte docker depuis un terminal Linux.

Pour cela, on tape la commande docker login. Puis on entre notre login et mot de passe qui sont ceux utilisés lors de la création du compte Docker hub.



Pour pouvoir effectuer un push de notre image docker, il faudra lui associer un tag via la commande :




Nous allons enfin effectuer un push de notre image docker vers le docker hub.

```
mickael@mickael-VirtualBox:~/Documents/workspace-spring-tool-suite-4-4.13.0.RELEASE/pays$ sudo docker push mickael2022/pays-docker-hub
Using default tag: latest
The push refers to repository [docker.io/mickael2022/pays-docker-hub]
9b243a7a10be: Pushing 29.39MB/39.62MB
9b243a7a10be: Pushed
7c072cee6a29: Mounted from library/openjdk
1e5fdc3d671c: Mounted from library/openjdk
613ab28cf833: Mounted from library/openjdk
bed676ceab7a: Mounted from library/openjdk
6398d5cccd2c: Mounted from library/openjdk
0b0f2f2f5279: Mounted from library/openjdk

latest: digest: sha256:ef1f413eaf4182bbe8e4c477b98473169da541568fb0f7daf7008dc506fca0cc size: 2007
mickael@mickael-VirtualBox:~/Documents/workspace-spring-tool-suite-4-4.13.0.RELEASE/pays$
```


Nous pourrions vérifier ensuite dans notre repository du docker hub que l'image docker est bien présente dans le docker hub.

 **docker hub**


[Explore](#) [Repositories](#) [Organizations](#)

[mickael2022](#) > [Repositories](#) > [pays-docker-hub](#)


[General](#) [Tags](#) [Builds](#) [Collaborators](#) [Webhooks](#) [Settings](#)




Advanced Image Management
View all your images and tags in this repository, clean up unused content, recover untagged images...




mickael2022 / pays-docker-hub



This repository does not have a description 

 Last pushed: 3 minutes ago

Tags and Scans

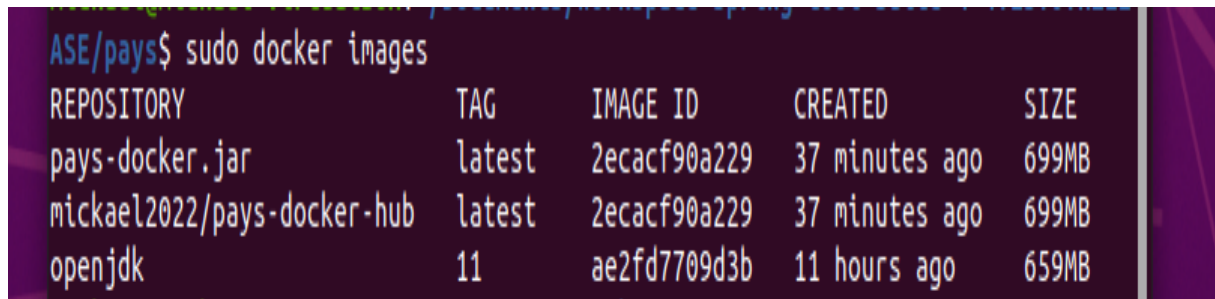
 **VULNERABILITY SCANNING - DISABLED**
[Enable](#)

This repository contains 1 tag(s).

TAG	OS	PULLED	PUSHED
 latest		3 minutes ago	3 minutes ago

[See all](#)

Nous pouvons également voir que l'image docker qu'on a déposé dans le docker hub est présente dans notre repository en local.



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
pays-docker.jar	latest	2ecacf90a229	37 minutes ago	699MB
mickael2022/pays-docker-hub	latest	2ecacf90a229	37 minutes ago	699MB
openjdk	11	ae2fd7709d3b	11 hours ago	659MB

Notre image docker de notre projet a donc bien été déposé sur le docker hub.



TAG

latest

Last pushed 7 minutes ago by mickael2022

docker pull mickael2022/pays-docke ...

DIGEST: ef1f413eaf41

OS/ARCH: linux/amd64

LAST PULL: 7 minutes ago

COMPRESSED SIZE: 352.49 MB

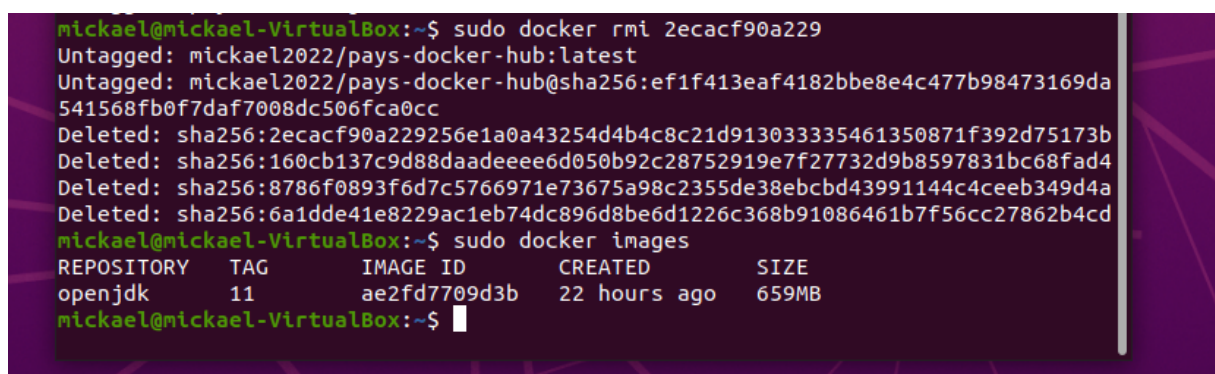
- **3.e) Téléchargement de l'image docker de notre application depuis le docker hub (docker pull)**

Nous devons donc vérifier qu'il est possible de récupérer cette image docker pour tout utilisateur.

Nous devons depuis la machine virtuelle et le même compte docker, supprimer directement l'image docker associée à notre compte docker-hub donc celle déposée dans le repository locale (dans le disque dur de la machine virtuelle).

On peut aussi supprimer l'image docker initial (pays-docker.jar).

Pour cela on utilise la commande `docker rmi [image ID]` qui nous permet d'enlever les tags associés aux images docker et de supprimer ces images docker de notre machine.



```
mickael@mickael-VirtualBox:~$ sudo docker rmi 2ecacf90a229
Untagged: mickael2022/pays-docker-hub:latest
Untagged: mickael2022/pays-docker-hub@sha256:ef1f413eaf4182bbe8e4c477b98473169da541568fb0f7daf7008dc506fca0cc
Deleted: sha256:2ecacf90a229256e1a0a43254d4b4c8c21d91303335461350871f392d75173b
Deleted: sha256:160cb137c9d88daadeeee6d050b92c28752919e7f27732d9b8597831bc68fad4
Deleted: sha256:8786f0893f6d7c5766971e73675a98c2355de38ebcbd43991144c4ceeb349d4a
Deleted: sha256:6a1dde41e8229ac1eb74dc896d8be6d1226c368b91086461b7f56cc27862b4cd
mickael@mickael-VirtualBox:~$ sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
openjdk       11        ae2fd7709d3b   22 hours ago   659MB
mickael@mickael-VirtualBox:~$
```

On voit bien que l'image docker mickael2022/pays-docker liée au docker hub et l'image docker initiale ne sont plus présentes localement. Cette image docker est néanmoins toujours présente dans le repository du docker hub.

```
mickael@mickael-VirtualBox:~$ sudo docker run -p 8080:8080 mickael2022/pays-docker-hub
Unable to find image 'mickael2022/pays-docker-hub:latest' locally
latest: Pulling from mickael2022/pays-docker-hub
```

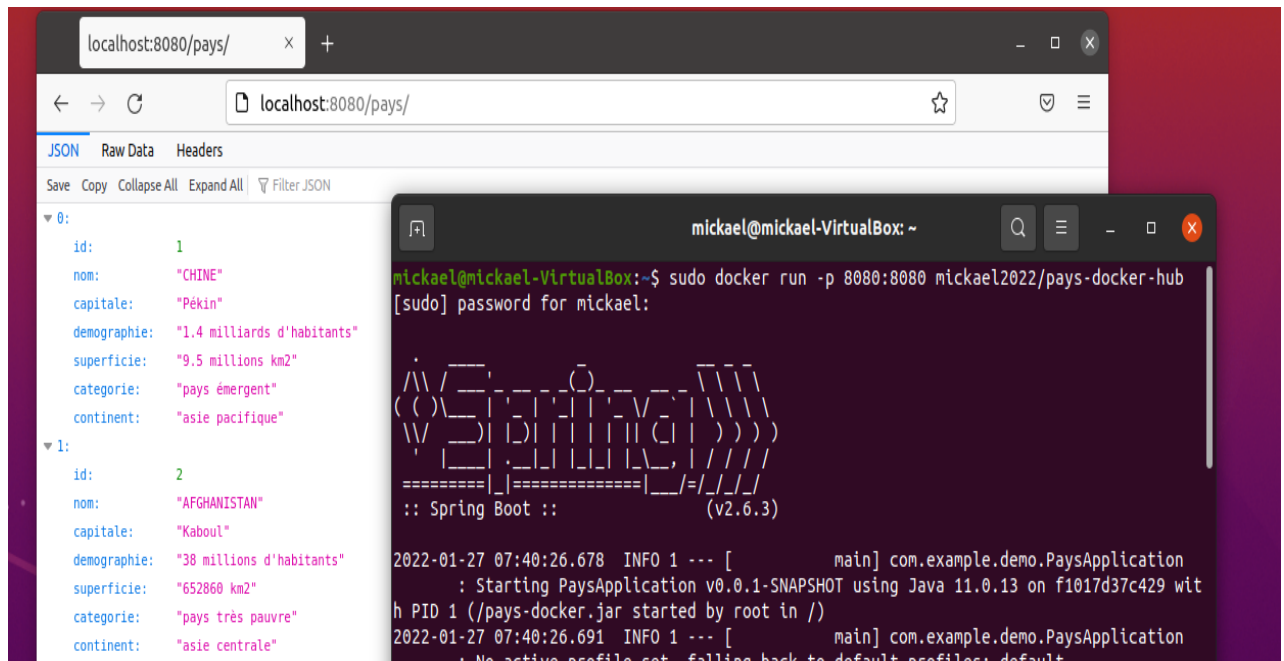
Unable to find image 'mickael2022/pays-docker-hub:latest' locally

```
latest: Pulling from mickael2022/pays-docker-hub
```

3.f) Exécution de notre application liée à une image docker importé du docker hub

[illegible]

Nous avons donc exécuté notre service web initial en ayant importé l'image docker de notre application depuis le docker hub.



4) Kubernetes

4.1 Création d'un cluster Kubernetes dans Google Cloud Platform

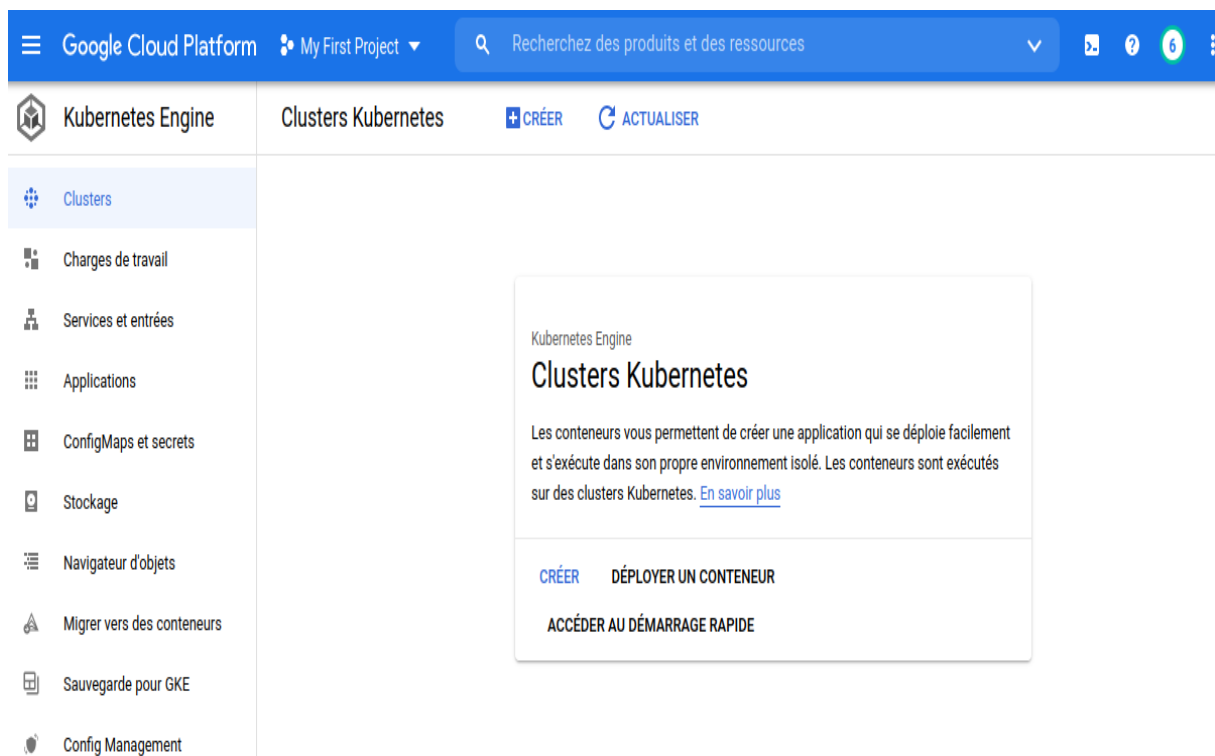
Nous voulons déployer une version de notre service web sur un cluster Kubernetes lié à Google Cloud Platform pour que ce service puisse directement être accessible à tout le monde depuis un simple navigateur internet et sur n'importe quelle terminal.

Pour cela, nous devons d'abord créer un cluster Kubernetes, puis déployer notre application dans ce cluster à l'aide d'un fichier d'extension .yaml qui indique la configuration de notre déploiement dans ce cluster.

Ce fichier d'extension .yaml a été codé au préalable depuis Spring Tools Suite 4 à la racine du projet. Nous allons ensuite importer ce fichier dans notre espace de travail associé la console Google Cloud Shell et au cluster Kubernetes et nous allons créer un déploiement de notre application à l'aide de ce fichier d'extension .yaml.

On se rend sur Google cloud platform après avoir créé un compte GCP. On clique sur l'onglet liste (tout en haut à gauche) et on clique sur Kubernetes Engine.

On clique ensuite sur le bouton Clusters et enfin sur le bouton créer un cluster.



On sera ensuite invité à saisir le nom de notre cluster, ici nous l'appellerons « pays-kubernetes ». Nous laisserons les autres paramètres par défaut et nous cliquerons enfin sur créer un cluster.

Paramètres de base du cluster

Le cluster sera créé avec le nom, la version et l'emplacement que vous indiquez ici. Une fois le cluster créé, le nom et l'emplacement ne peuvent pas être modifiés.

i Pour effectuer des tests avec un cluster abordable, sélectionnez **Mon premier cluster** dans le menu **Guides de configuration des clusters**

Nom
pays-kubernetes

Type d'emplacement

- ☒ Zonal
- ☐ Régional

Zone us-central1-c ▼ ?

- ☐
- Spécifiez les emplacements de nœuds par défaut ?

Emplacements par défaut actuels : us-central1-c

Version du plan de contrôle

Choisissez une version disponible pour la gestion automatique de la version de votre cluster et de la fréquence de mise à niveau. Optez pour une version statique pour gérer la version de votre cluster de façon plus directe. [En savoir plus.](#)

- ☐ Version stable
- ☒ Version disponible

Version disponible
Version standard (par défaut) ▼

Version 1.21.6-gke.1500 (par défaut) ▼

Ces versions ont réussi les tests de validation interne et sont considérées de qualité production, mais ne disposent pas de données historiques suffisantes pour garantir leur stabilité. Les

CRÉER

ANNULER

Requête **REST** ou **LIGNE DE CC**

On pourra voir que notre cluster a bien été créé.

Clusters Kubernetes

[+ CRÉER](#)
[+ DÉPLOYER](#)
[↻ ACTUALISER](#)

APERÇU

OPTIMISATION DES COÛTS BETA

☰

Filtre

Saisissez le nom ou la valeur de la propriété

<input type="checkbox"/>	État	Nom ↑	Zone	Nombre de nœuds	Nombre total de processeurs virtuels	Mémoire totale
<input type="checkbox"/>	✔	k8s-pays-docker	us-central1-c	3	6	12 Go
<input type="checkbox"/>	✔	pays-kubernetes	us-central1-c	3	6	12 Go

On va ensuite se connecter au cluster. On clique sur le cluster et on voit l'ouverture du terminal Google cloud Shell et on exécute la commande qui nous a été donnée pour se connecter au cluster.

```
CLOUD SHELL
Terminal cloudshell x +
Welcome to Cloud Shell! Type "help" to get started.
To set your Cloud Platform project in this session use "gcloud config set project [PROJECT_ID]"
mickail75013@cloudshell:~$ gcloud container clusters get-credentials pays-kubernetes --zone us-central1-c --project macro-raceway-339107
Fetching cluster endpoint and auth data.
kubeconfig entry generated for pays-kubernetes.
```

On va ensuite créer un fichier .yaml dans Spring tools suite 4 à la racine du projet pour configurer le déploiement de notre application.

Nous allons ensuite importer ce fichier d'extension .yaml et nous allons vérifier que ce fichier est bien présent dans notre espace de travail lié au cluster à l'aide de la commande ls.

```
mickail75013@cloudshell:~$ ls
pays-kubernetes.yaml README-cloudshell.txt
```

Puis nous allons taper la commande : `kubectl apply -f pays-kubernetes.yaml` pour créer le déploiement de notre application dans le cluster.

Le déploiement a été créé.

```
mickail75013@cloudshell:~$ kubectl apply -f pays-kubernetes.yaml
deployment.apps/k8s-pays-docker-deployment-service created
mickail75013@cloudshell:~$
```

Nous irons ensuite dans l'onglet Charges de travail. Nous pourrions voir que nous avons une nouvelle charge de travail associée à notre déploiement.

Charges de travail

ACTUALISER DÉPLOYER SUPPRIMER

Cluster Espace de noms RÉINITIALISER SAVE

Les charges de travail sont des unités de calcul déployables qui peuvent être créées et gérées dans un cluster.

Données des clusters manquantes [k8s-pays-docker](#) (indisponible)

APERÇU

OPTIMISATION DES COÛTS BÉTA

Filtre Est un objet système : Faux Filtre les charges de travail

	Nom ↑	État	Type	Pods	Espace de noms	Cluster
<input type="checkbox"/>	k8s-pays-docker-deployment-77c45945db-gvkgg	Terminating	Pod	1/1	default	pays-kubernetes
<input checked="" type="checkbox"/>	k8s-pays-docker-deployment-service	OK	Deployment	1/1	default	pays-kubernetes

Puis nous allons cliquer sur la charge de travail correspondante à notre projet. Une nouvelle fenêtre s'ouvre, nous allons cliquer sur l'onglet Exposer.

✓ k8s-pays-docker-deployment-service

❗ Pour permettre aux autres utilisateurs d'accéder à votre déploiement, vous pouvez l'exposer pour créer un service

EXPOSER

APERÇU

DÉTAILS

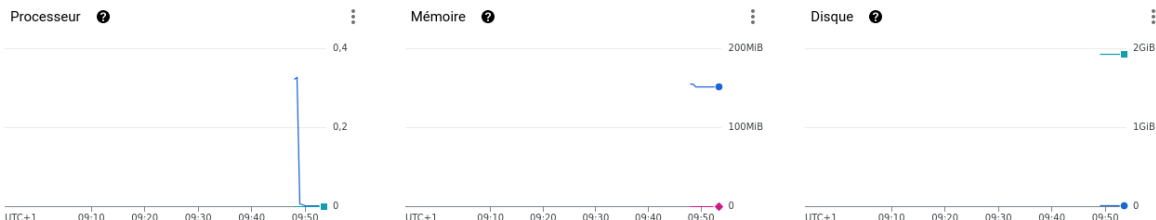
HISTORIQUE DES RÉVISIONS

ÉVÉNEMENTS

JOURNAUX

YAML

✓ 1 heure 6 heures 12 heures 1 jour 2 jours 4 jours 7 jours 14 jours 30 jours Personnalisée ▼



Cluster	pays-kubernetes
Espace de noms	default
Étiquettes	app: pays-kubernetes
Journaux	Container logs , Audit logs
Instances dupliquées	1 mise à jour, 1 prête, 1 disponible, 0 non disponible
Spécifications du pod	Révision 1, conteneurs : pays-kubernetes

Nous allons ici exposer ce déploiement pour créer un service. On choisit pour cela le port 8080 comme port cible et on laisse les autres paramètres par défaut. On clique enfin sur exposer.

← Exposer un déploiement

L'exposition d'un déploiement entraîne la création d'un service Kubernetes. Ce service définit la man

Mappage de port

Port * ? Port cible ? Protocole ?

80 8080 TCP

+ AJOUTER UN MAPPAGE DE PORT

Type de service
Équilibreur de charge

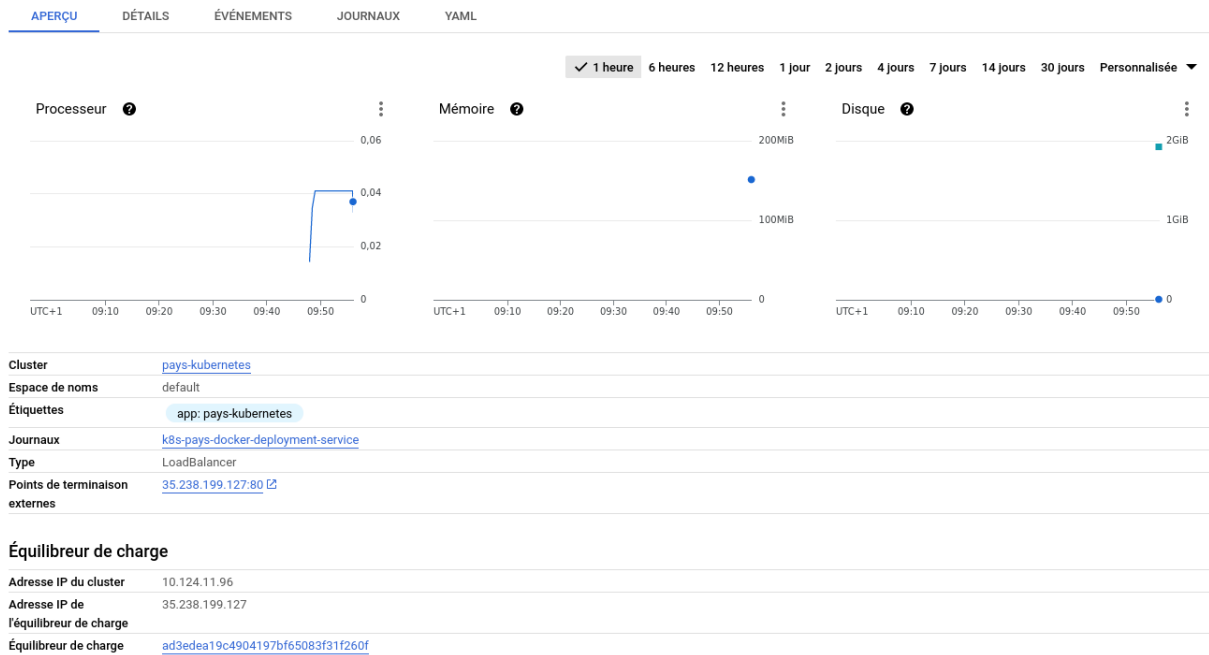
Nom du service
k8s-pays-docker-deployment-service-service

EXPOSER

AFFICHER YAML

Nous avons donc bien exposé notre service et il est désormais accessible à tout d'utilisateur client depuis n'importe quel terminal.

✓ k8s-pays-docker-deployment-service-service



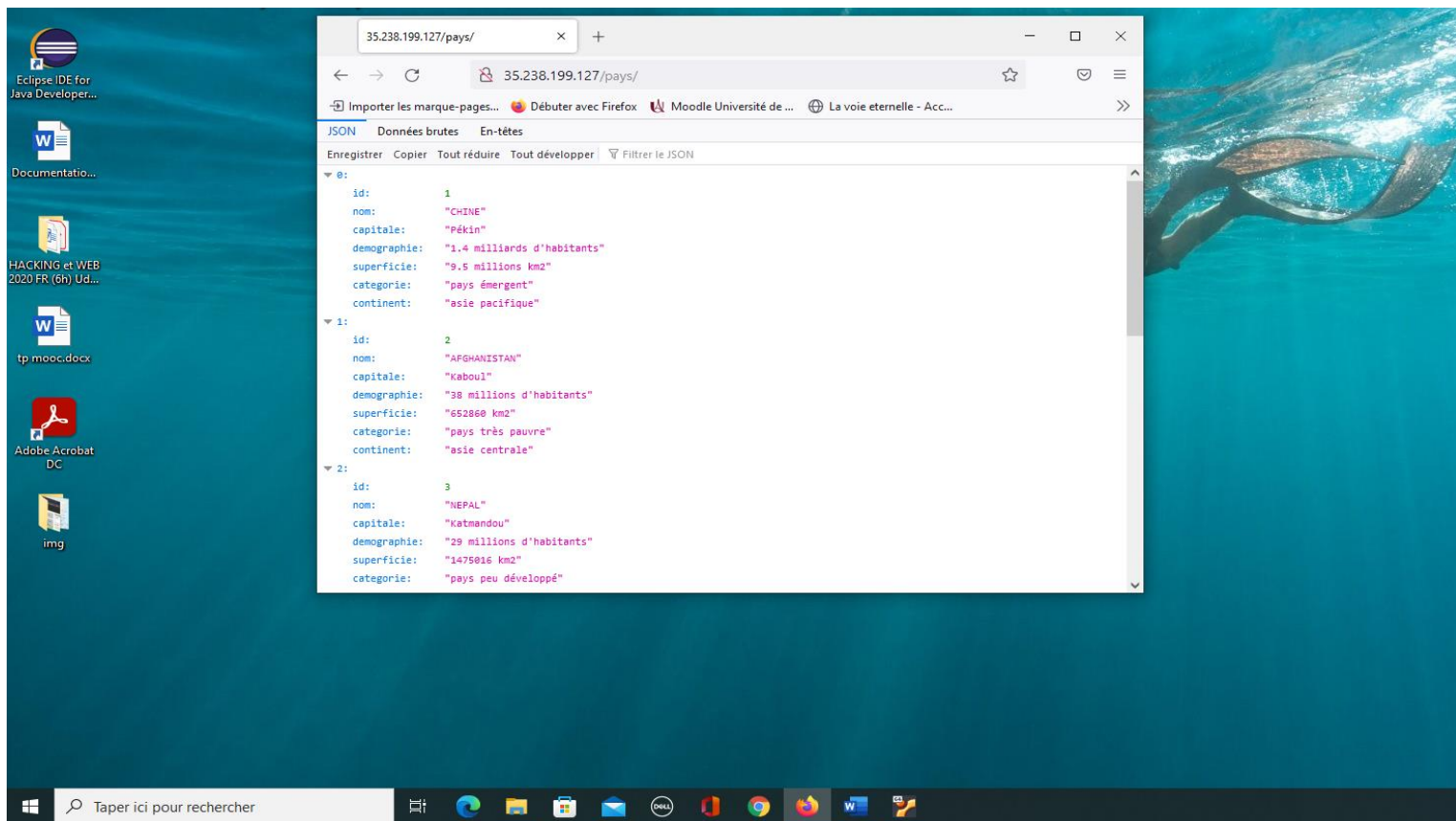
Les points de terminaisons externes indique que notre service est accessible via ce point de terminaison qui est ici une adresse IP suivi d'un numéro de port.

Points de terminaison externes [35.238.199.127:80](#)

On entre dans n'importe quel terminal en tant qu'utilisateur l'adresse suivante.

[35.238.199.127/pays/](#)

Pour vérifier que notre application est accessible à tout utilisateur depuis ce point de terminaison nous allons quitter notre machine virtuelle Linux et on se retrouve sur machine hôte Windows 10 et on tape sur un navigateur l'URL : [35.238.199.127/pays/](#)



Nous avons donc vu que gr\u00e2ce au d\u00e9ploiement de l'application dans un cluster Kubernetes que tout utilisateur pouvait utiliser notre web service depuis n'importe quel terminal et \u00e0 tout moment.

Notre service est rendu disponible \u00e0 tout utilisateur !

Voici les statistiques li\u00e9es \u00e0 l'utilisation de notre web service.

