Architecture micro-service

TP 5 - Premiers micro-services - Météo

Philippe Roussille

3iL INGENIEURS
</les sciences informatiques>

1 Présentation du cadre

> *"Ginette, tu crois qu'il fera beau à Tombouctou ?" "Aucune idée Roger… mais on va devoir le savoir, maintenant qu'on ouvre une succursale là-bas !"*

Après avoir construit un microservice autonome pour les blagues, CanaDuck ouvre deux nouvelles succursales : l'une à **Honolulu**, l'autre à **Tombouctou**. Impossible de gérer correctement les horaires et les conditions d'expédition sans savoir s'il fait beau ou non.

CanaDuck se penche donc sur un **service météo**, capable de répondre aux demandes internes sur les conditions climatiques à différents endroits du monde.. Mais cette fois, pas question de tout coder à la main : on va **interroger un service externe déjà existant** pour récupérer les prévisions !

La météo sera récupérée depuis l'API publique de https://open-meteo.com, et restituée de manière simple depuis un microservice Flask.

2 Fonctionnalités attendues

Le microservice doit :

1. Exposer une route :

```
GET /weather?city=Rodez
```

   - Utiliser un dictionnaire interne pour convertir le nom de la ville en coordonnées (lat, lon)
   - Interroger l'API `https://api.open-meteo.com/v1/forecast?latitude=...&longitude=...&current_weat`
   - Retourner un JSON simplifié avec :

```
{
  "city": "Rodez",
  "temperature": 21.3,
  "windspeed": 15.2,
```

```
  "condition": "partiellement nuageux"
}
```

Si la ville n'est pas supportée :

```
{
  "error": "Ville inconnue"
}
```

2. Exposer une deuxième route :

```
GET /cities
```

qui renvoie la liste des villes disponibles, par exemple :

```
{
  "available_cities": ["Rodez", "Honolulu", "Tombouctou"]
}
```

2.1 Données des villes supportées

Créez un petit dictionnaire Python :

```
known_cities = {
    "Rodez": (44.35, 2.57),
    "Honolulu": (21.30, -157.85),
    "Tombouctou": (16.77, -3.01)
}
```

3 Ça peut vous être utile ?

Héhé, vous avez déjà vu tout le code utile à cette partie… regardez donc les TPs précédents ! ;)

4 Quelques questions de fond

1. Pourquoi ne pas appeler directement open-meteo depuis le navigateur ?
2. Quel est l'avantage de passer par un microservice intermédiaire ?
3. Si le format de réponse de `open-meteo` change, que se passe-t-il ?
4. Que pourrait-on ajouter pour rendre ce service plus complet ou plus robuste ?

5 Dockerisation obligatoire

Pour que ce service soit déployable facilement par l'équipe (et notamment par Paul et Patrice qui refusent d'installer Flask à la main), vous devez le dockeriser.

Pour cela, inspirez vous de ce qui a été fait dans le TP sur le générateur de blagues.

Pour rappel :

```
docker build -t canaduck/weather-service .
docker run -p 5000:5000 canaduck/weather-service
```

Le service doit être accessible à :

— `http://localhost:5000/weather?city=Rodez`

6 Bonus : documentation automatique avec Flasgger

Si vous avez terminé les fonctionnalités principales, ajoutez une documentation interactive avec **Flasgger**, comme dans le TP précédent. Vérifiez alors et accédez à la documentation Swagger sur :

`http://localhost:5000/apidocs`

> *"Je t'avais dit qu'on pouvait prédire le vent avec Python."* — Roger