RT0904 - Rapport de TP

Benjamin HUYNH
Paul MIRGAINE

Introduction

Ce rapport a pour but de décrire le travail réaliser dans le cadre des TP de RT0904 dont l'objectif est la création serverless basé sur des solutions Cloud, existante dans le cadre du TP1 puis dans un cadre à installer et configurer nous même dans le cas du TP2.

Ce rapport est séparer en deux parties correspondant aux TP1 puis TP2.

Contenu

1 TP1

L'objectif de ce TP est de mettre en application deux services cloud appris en cours de RT904, le BaaS et le FaaS.

1.1 Choix techniques

Cloud provider : Nous avons choisi d'utiliser le cloud provider de google. GCloud met à disposition plusieurs services et la mise en relation de celles-ci est simple. GCloud met également à disposition une documentation détaillée.

Application : Pour le développement de l'application web de type BaaS, nous avons choisi le micro framework open-source Flask en python.

Pour le déploiement : le service google app engine.

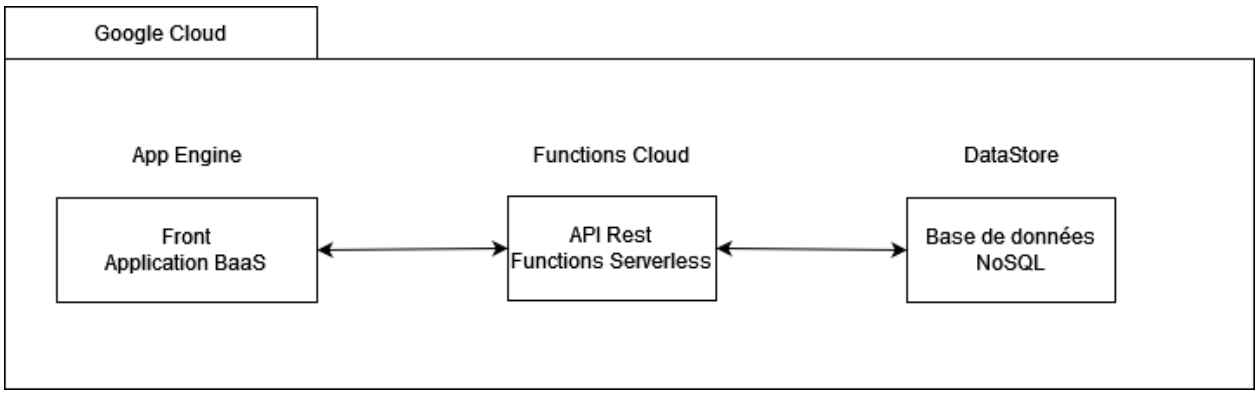
Functions : Pour le développement des fonctions cloud, nous utiliserons python pour sa flexibilité. Pour le déploiement : le service cloud functions.

Database : Pour le stockage des données, nous utiliserons Datastore une base de donnée NoSQL sur GCloud.

Echanges : Pour les échanges de données entre tous les services, nous utiliserons du JSON.

1.2 Architecture

Architecture général :



App Engine :

Structure des fichiers flask :

```
        app/
            app.yaml -> Fichier de configuration pour app
engine
            main.py  -> Code principal
            requirements.txt -> Package python
            static/ -> Assets
                script.js
                style.css
            templates/ -> Fichier HTML
                index.html
```

Endpoint :

/list - Affiche les candidats

/votes - Vote

Cloud functions
Structure des fonctions serverless :
Endpoint :

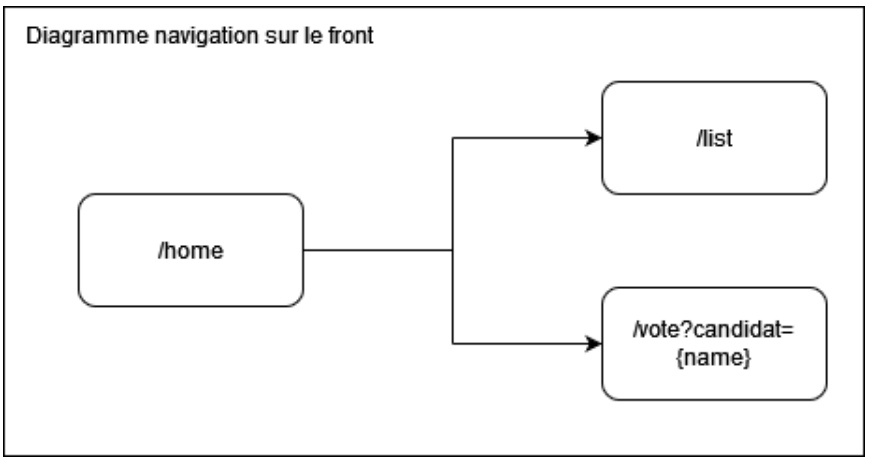
/list - Liste JSON

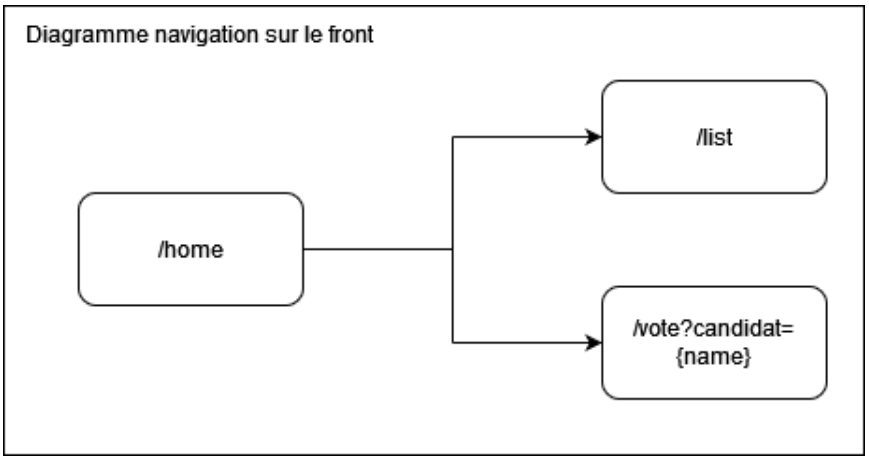
/votes - Enregistre dans la BDD le vote en méthode POST. ex: {"nom" : "Jeannot"}

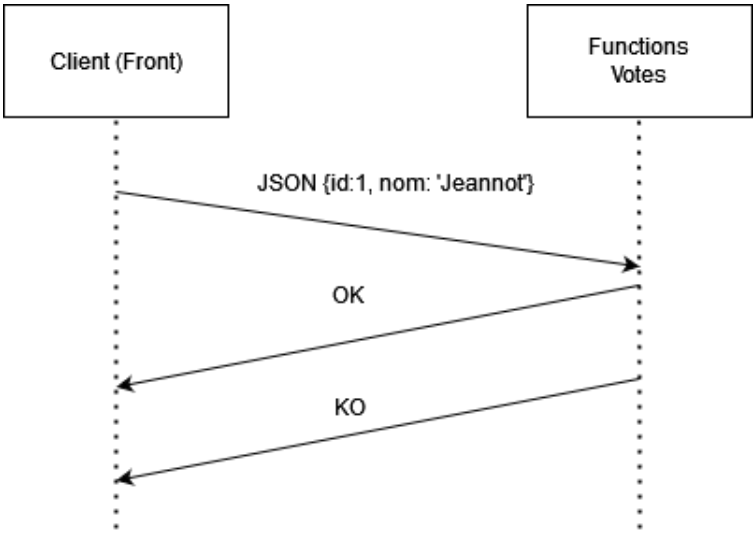
Les triggers pour ces fonctions sont des simples HTTP request.

DataStore
Entity :
Création d'une simple entity "Candidat" avec des propriétés "id", "nom", "vote"

1.3 Diagramme d'échange







1.4 Résultats

Le déploiement est simple avec le CLI de GCloud. Après avoir configurer son projet sur la console Google developper. Il suffit d'utiliser les commandes suivantes pour déployer et nettoyer :

- Pour les fonctions

```
    gcloud functions deploy {function\_name} --trigger-http --
  runtime=python311 --region=europe-west1 --allow-
  unauthenticated

    gcloud functions delete {function\_name} --region=europe-
  west1
```

- Pour le BaaS

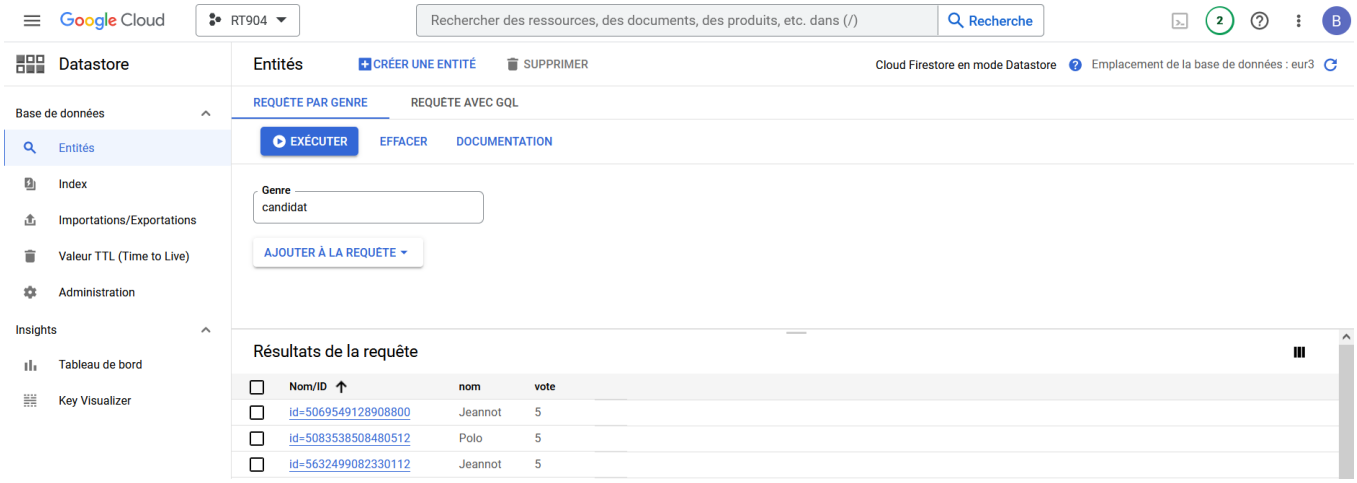
```
    gcloud app deploy
    gcloud app browse -> Pour avoir l'url
```

Console Google - Functions

Filtre Filtrer les fonctions

	Environnement	Nom	Dernier déploiement	Région	Recommandation	Déclencheur	Exécution	Mémoire allouée	Fonction exécutée	Actions
☐	✓	1st gen	add_vote	31 mars 2023, 13:27:59	europe-west1		HTTP	Python 3.11	256 MB	add_vote	⋮
☐	✓	1st gen	list_candidat	31 mars 2023, 13:31:28	europe-west1		HTTP	Python 3.11	256 MB	list_candidat	⋮

DataStore



Ecran d'accueil

TP1

/list pour lister la liste des candidats

/votes pour lister la liste des candidats

Affichage de la liste des candidats

Liste des candidats

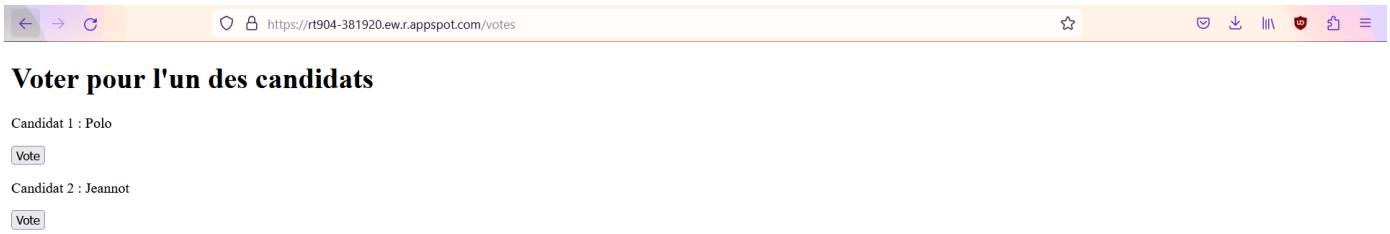
Candidat Polo

Nombre de votes : 5

Candidat Jeannot

Nombre de votes : 3

Vote pour l'un des candidats



Service REST - Liste des candidats sous forme JSON

Ce TP nous a permis d'apprendre les services cloud tels que le BaaS et le FaaS. Nous nous rendons compte de la flexibilité que nous offre les cloud provider sur la scabilité verticale et horizontale.

2 TP2

Le but de ce second TP est la mise en place d'une solution serverless installer à partir d'un environnement cloud local installer à partir de d'Apache Openwhisk ou KNative. L'installation des deux solutions pour comparaison a été effectuer, mais KNative a été retenu pour les développements qui suivent.

2.1 Installation et configuration de l'environnement

2.1.1 OpenWhisk

Les prérequis d'OpenWhisk présument que l'utilisateur possèdent déjà un cluster Kubernetes et que Helm est installé sur leur système.

La première étape de l'installation a été la récupération du dépôt Git **openwhisk-deploy-kube** contenant les fichers nécessaire pour l'installation par helm ainsi que des exemples de manifest pour la configuration du déploiement d'OpenWhisk.

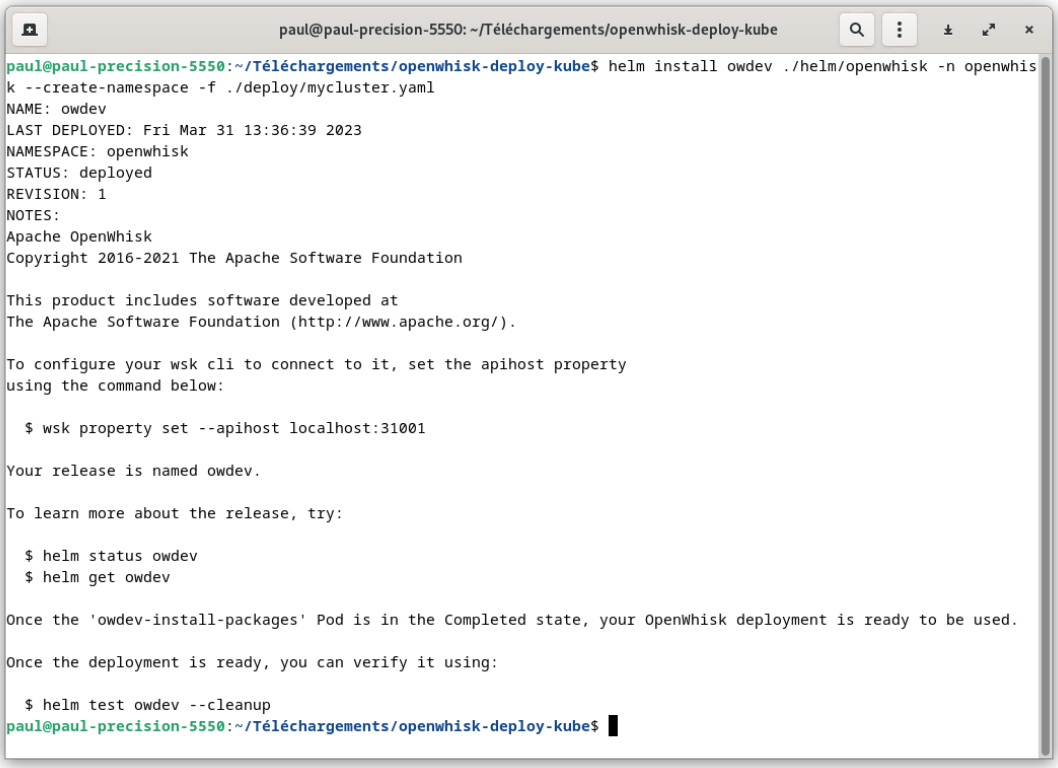
Ce fichier nommée **mycluster.yaml** dans la doc et les exemples possède le contenu suivant :

```
whisk:
  ingress:
    type: NodePort
    apiHostName: localhost
    apiHostPort: 31001
    apiHostProto: "https"
    useInternally: false

nginx:
  httpsNodePort: 31001

# A single node cluster; so disable affinity
affinity:
  enabled: false
toleration:
  enabled: false
invoker:
  options: "-Dwhisk.kubernetes.user-pod-node-affinity.enabled=
   false"
```

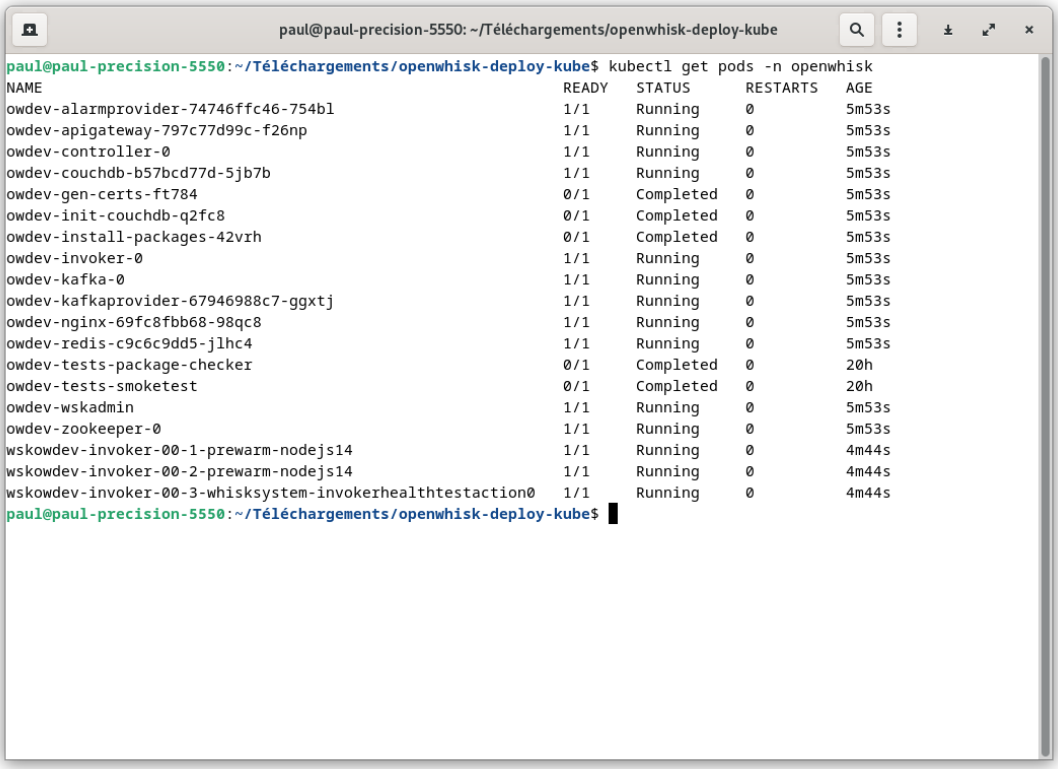
Cette configuration contient les paramètre pour le Ingress généré par OpenWhisk, permettant la connexion au cluster par le CLI d'OpenWhisk.

Nous pouvons maintenant effectué l'installation grâce à Helm :

```
helm install owdev ./helm/openwhisk -n openwhisk --create-
   namespace -f ./deploy/mycluster.yaml
```



Comme indiqué après le lancement de Helm, l'installation sera terminée lorsque le pod **owdev-package-install** aura le statut *complete*. Voici la liste des pods lorsque l'installation est terminée :



La dernière étape de l'installation d'OpenWhisk est de configurer le CLI d'OpenWhisk **wsk**. Il suffit de lancer les commandes suivantes :

```
wsk property set --apihost localhost:31001
wsk property set --auth 23bc46b1-71f6-4ed5-8c54-816aa4f8c502:123
   zO3xZCLrMN6v2BKK1dXYFpXlPkccOFqm12CdAsMgRU4VrNZ9lyGVCGuMDGIwP
```

La chaîne configurer dans auth correspond à l'utilisateur *guest*.

2.1.2 KNative

Le prérequis pour l'installation de KNative est la présence de minikube pour exécuter un cluster Kubernetes. L'installation de KNative se fait principalement grâce à l'installation d'utilitaire binaire. Ces binaires sont : **kn**, **kn-quickstart**, **func**.

kn Utilitaire principal de KNative

kn-quickstart Utilitaire rapide pour KNative.

func Utilitaire pour gérer les fonctions KNative.

Il n'est pas nécessaire de démarrer le cluster minikube à l'avance car l'utilitaire **kn-quickstart** va nous le créer préconfigurer avec un profile *knative* en exécutant la commande suivante :

```
kn-quickstart minikube
```

Le cluster ainsi généré est ainsi prêt à l'emploi.

2.2 Fonctions

2.2.1 Accès REST

La fonction d'accès REST a été créer à l'aide des fonctions KNative.

L'initialisation d'une nouvelle fonction KNative s'effectue à l'aide de l'utilitaire **func** de KNative :

```
func create -l typescript fn-rest
```

L'utilitaire nous a ainsi généré une fonction KNative basique en TypeScript. Celle-ci est exécutable à l'aide de la commande :

```
func run
```

Pour l'accès REST une base de donnée MongoDB Atlas a été utiliser. Celle-ci contient un jeu de donnée d'exemple dont des données météo. Celles-ci se trouve dans la base *sample_weatherdata* dans la collection *data*. Le code de la fonction se trouve dans le fichier **index.ts** dans le dossier *TP2/fn-rest* présent avec ce rapport.

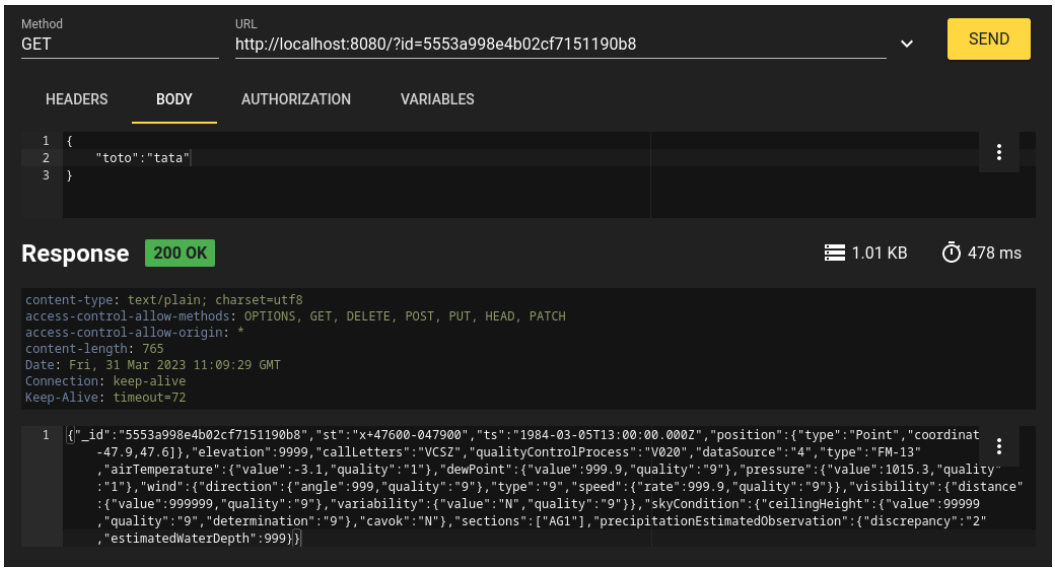


Figure 1: Exemple d'exécution avec la méthode GET

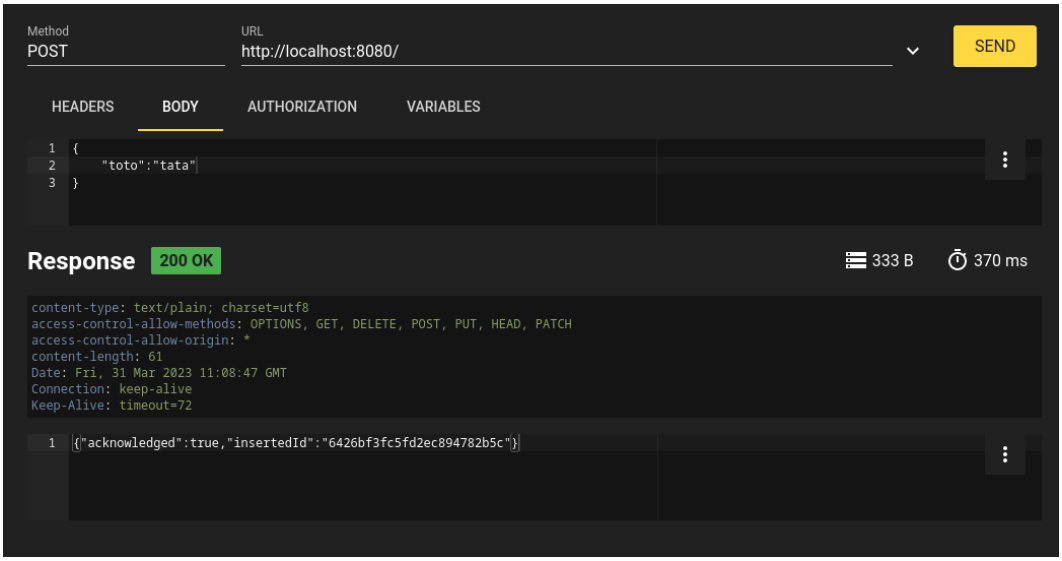


Figure 2: Exemple d'exécution avec la méthode POST