Python Microservices Documentation

Release 0.1

avara1986

Jan 12, 2020

Contents

Python Microservice Scaffold is an example of how to structure a Flask Microservice Project. This Scaffold is build over PyMS package. PyMS is a Microservice chassis pattern like Spring Boot (Java) or Gizmo (Golang). PyMS is a collection of libraries, best practices and recommended ways to build microservices with Python which handles cross-cutting concerns:

- Externalized configuration
- Logging
- Health checks
- Metrics (TODO)
- Distributed tracing

CHAPTER 1

Stack

- **PyMS**
  - Flask
  - connexion
  - Opentracing
- SQLAlchemy
- Flask-SQLAlchemy
- Flask-Script

CHAPTER 2

Content

2.1 Installation

Clone the project

```
git clone https://github.com/python-microservices/microservices-scaffold.git
```

Install your virtualenv

```
virtualenv --python=python[3.6|3.7|3.8] venv
source venv/bin/activate
pip install -r requirements.txt
```

Configure your project and the path of your MS. See *configuration* section.

Configure your setup.py with your project information

2.2 Quickstart

2.2.1 Scaffold

Clone the project

```
git clone https://github.com/python-microservices/microservices-scaffold.git
```

Install your virtualenv

```
virtualenv --python=python[3.6|3.7|3.8] venv
source venv/bin/activate
pip install -r requirements.txt
```

Run the script

```
python manage.py runserver
```

Check the result

Your default endpoint will be in this url:

```
http://127.0.0.1:5000/template/
```

This URL is setted in your *config.yml*:

```
ms:
  DEBUG: false
  TESTING: false
  APP_NAME: Template
  APPLICATION_ROOT : /template # <!---
```

You can acceded to a swagger ui in the next url:

This PATH is setted in your *config.yml*:

2.2.2 Template

You can create your own project from template:

https://github.com/python-microservices/microservices-template

2.3 Structure

You have a project with this structure:

```
Dockerfile
LICENSE
manage.py
config.yml
README.md
requirements.txt
requirements-tests.txt
requirements-docker.txt
service.yaml
setup.py
tests.sh
tox.ini
project
├ __init__.py
├ models
│  ├ __init__.py
│  └ models.py
├ swagger
│  └ swagger.yaml
├ tests
│  └ test_views.py
└views
   ├ __init__.py
   └ views.py
```

(continues on next page)

(continued from previous page)

```
docs/
tests/
```

2.3.1 manager.py

A Django style command line. Use this to start the application like:

```
python manage.py runserver
```

You can set the host and the port with:

```
python manage.py runserver -h 0.0.0.0 -p 8080
```

2.3.2 Common Libraries

py-ms is a library that contains a set of common features for microservices.

2.3.3 Structure of a project

For project configuration see *configuration* section.

2.3.3.1 project/models/init_db.py

Initialize *flask_sqlalchemy.SQLAlchemy object*.

2.3.3.2 project/models/models.py

Project specific models.

2.3.3.3 project/swagger/swagger.yaml

Use to define your rest behaviour, endpoints and routes. See connexion docs to how add new views.

2.3.3.4 project/views

use views.py or create your file.

2.3.3.5 project/__init__.py

This file init the project calling *create_app* method. Initialize the Flask app, register blueprints and initialize all libraries like Swagger, database, trace system, custom logger format, etc.

2.4 Configuration

Project configuration is loaded using PyMS package based on yml or json file. Some example files are config.yml, config-docker.yml and tests/config-tests.yml or see PyMS configuration

2.4.1 Documentation

The Microservice create a URL to inspect the Swagger documentation of the api in:

```
localhost:5000/[APPLICATION_ROOT]/ui/
```

This URL is setted in your *config.yml*:

```
ms:
  DEBUG: false
  TESTING: false
  APP_NAME: Template
  APPLICATION_ROOT : /template # <!---
```

Our API Rest work with connexion. You can see connexion docs or the official Swagger documentation to add the syntax to your APIS and create your Swagger docs

2.5 Code Examples

- See simple examples in https://github.com/python-microservices/pyms/tree/master/examples
- Build a Chat with 3 Microservices and Kubernetes: https://github.com/python-microservices/microservices-chat
- Build a Chat with 4 Microservices and Docker Compose: https://github.com/avara1986/pivoandcode-2019-11-15

2.6 Project

2.7 Run in kubernetes

2.7.1 Use MS with Docker

Install docker

2.7.2 Use MS with Kubernetes localy

We create a extensive example an tutorial about use Python microservice scaffold with Kubernetes in this repository: https://github.com/python-microservices/microservices-chat

In the nexts lines, you can find a simple tutorial tu run this microservice in a simple Kubernetes cluster

- Installing Kubernetes...

```
curl -Lo kubectl https://storage.googleapis.com/kubernetes-release/release/v1.9.0/bin/
↪linux/amd64/kubectl && chmod +x kubectl && sudo mv kubectl /usr/local/bin/
```

- Installing Minikube...

```
curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-
↪linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
```

Start minikube and set the environment of docker

```
minikube start
eval $(minikube docker-env)
kubectl config use-context minikube
```

If a shell isn't your friend. You could use kubernetes web panel to see your pods:

```
minikube dashboard
```

Create your image

```
docker build -t template:v1 .
```

Deploy your images localy:

```
kubectl apply -f service.yaml
minikube service template
```

Clean your environment

```
kubectl delete template
kubectl delete template
docker rmi template:v1 -f
minikube stop
eval $(minikube docker-env -u)
minikube delete
```

Python Module Index

p

Index

P