Build Your Own Microservices in Flask

Last Updated : 25 Sep, 2023

Nowadays Python is everywhere. This is majorly due to its simplicity and wide range of applications. Flask is a lightweight web framework for building web applications in Python. Creating microservices with Flask is straightforward.

In this article, we will start with a basic “hello world” microservice. Following that, we’ll delve into the creation of two distinct microservices and explore how to establish seamless communication between them. So let’s begin with understanding the concept of a Microservice.

What is a microservice?

A microservice is a software architectural style whereby an application is divided into smaller sub-parts. These sub-parts are independent of each other i.e., they are independently deployable and scalable. Each microservice performs a specific function or a single task.

Why do we need microservices?

Imagine an e-commerce app like Amazon or Flipkart. Here the app needs to satisfy many tasks like

Order management: placing orders, tracking status etc.

Cart management: Updating cart, calculating cart value etc.

Customer Profile: Name, email, membership management etc.

As you can see, a single app performs many tasks. However, trying to write code for all these tasks within a single framework can make it complicated and challenging to manage and debug. Hence the concept of microservices came into the picture. Each microservice is designed to perform a single task.

Furthermore, microservices possess the ability to communicate with each other whenever it's required.

Building a simple Microservice

Basic Requirements:

- A computer with internet connection
- Python Programming
- Installing Flask
- A basic idea of Flask
- A basic idea of the Python virtual environment is a plus

Before proceeding, let's create a virtual environment. It is important to create a virtual environment to keep our project dependencies and environment isolated and don't conflict with the other projects in the computer environment. This is a safer process and easy to manage the project environment.

Use the following commands in the terminal to create a virtual environment:

```
python -m venv venv
```

Activate our virtual environment:

In Linux and Mac OS use the below command in the terminal:

```
source venv/bin/activate
```

In Windows, use the below command in the command prompt:

```
venv\Scripts\activate
```

NOTE: Use deactivate keyword to come out of the virtual environment.

The above will activate the virtual environment and you can see (venv) on the left side of the terminal. Now let us create a directory with the name microservices. Create a requirements.txt file in the microservices directory.

Add the following in the requirements.txt. These all modules are required in our project.

```
flask
requests
random
```

Use the following pip command to install all the above modules in our virtual environment:

```
pip install -r requirements.txt
```

Create a Python file named **`microservice.py`** `in the same directory` and add the following code:

Python3

```python
from flask import Flask, jsonify

app = Flask(__name__)

@app.route("/hello", methods=['GET'])
def hello_microservice():
    message = {"message": "Hello from the microservice! This is GeeksForGeeks"}
    return jsonify(message)

if __name__ == "__main__":
    app.run(port=8000)
```

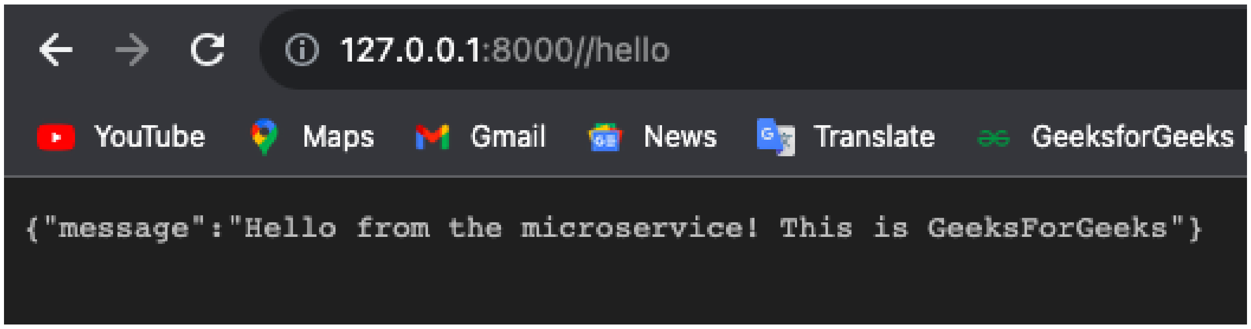
Here, we created our microservice at the route /hello and deployed it on port 8000. So you can access it at http://127.0.0.1:8000/hello. Also, the function hello_microservice() returns the JSON data as output.

Run the **`microservice.py`** using the Python command:

```
python microservice.py
```

Go to http://127.0.0.1:8000/hello in your browser. You can see the JSON output.

Output:



```
{"message":"Hello from the microservice! This is GeeksForGeeks"}
```

First Microservice

NOTE: Please note that if you encounter a "URL not found" error in the Chrome browser, consider trying the following link instead: http://127.0.0.1:8000//hello

You can also use the curl command in the terminal to get the output. You'll receive the same JSON response.

```
curl http://127.0.0.1:8000/hello
```

Output

```
{"message": "Hello from the microservice! This is GeeksForGeeks"}
```

Communication Between Microservices

Let's build two microservices. One microservice will generate a Random Number. The other microservice will check whether the generated random number is Even or Odd.

1. Random Number Generator Microservice:

Create a file named `random_microservice.py in the microservices directory` and add the below code.

Python3

```
from flask import Flask, jsonify
import random

app = Flask(__name__)

@app.route("/generate", methods=['GET'])
def generate_random_number():
    random_number = random.randint(1, 1000)
    return jsonify({"random_number": random_number})

if __name__ == "__main__":
    app.run(port=8001)
```

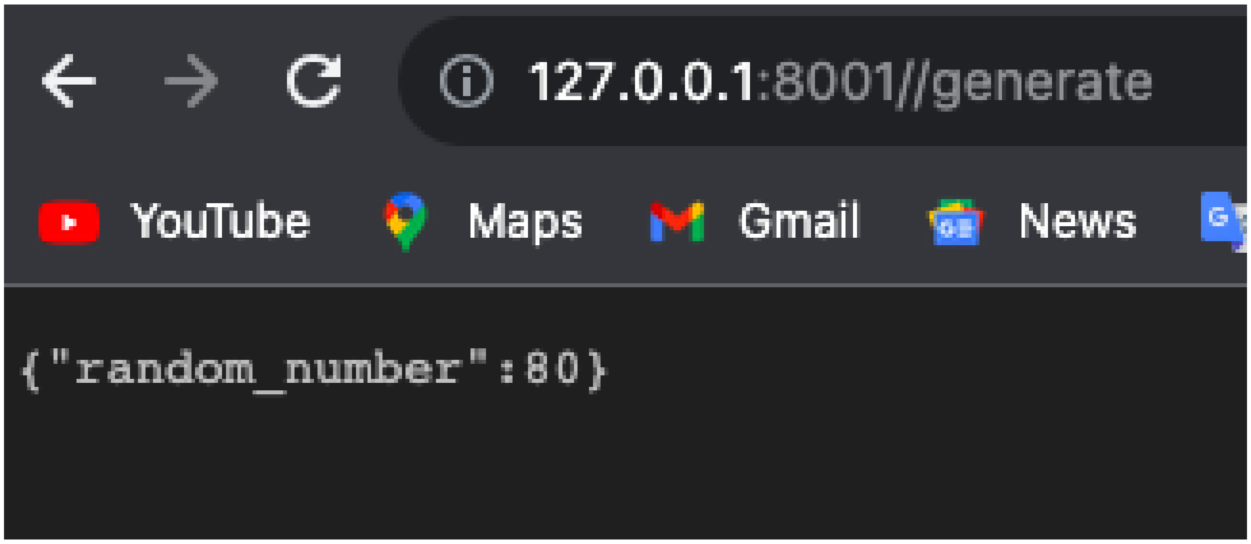
Here, we created a microservice at port 8001 and defined the API route at /generate. Thus the microservice will run at http://127.0.0.1:8001/generate. The generate_random_number() function will return a random number in JSON format. The random.randint(1, 1000) function will give any random number between 1 to 1000.

Run the `random_microservice.py` using the Python command:

```
python random_microservice.py
```

Go to http://127.0.0.1:8001/generate in your browser. You can see the JSON output.

Output:



```
{"random_number":80}
```

Random Number Generator

2. Even or Odd Checker Microservice:

Create a file named `evenodd_microservice.py` in the microservices directory and add the below code.

Python3

```python
from flask import Flask, jsonify
import requests

app = Flask(__name__)

random_microservice_url = "http://127.0.0.1:8001/generate"

# Calling the random number generator microservice
def call_random_microservie():
    response = requests.get(random_microservice_url)
    return response.json().get("random_number")

@app.route("/check", methods=['GET'])
def check_even_odd():
    random_number = call_random_microservie()
    result = "even" if random_number % 2 == 0 else "odd"
    return jsonify({"random_number": random_number, "result": result})

if __name__ == "__main__":
    app.run(port=8002)
```

The second microservice will be running at port 8002. We defined the API route at /check, so the microservice can be called by link http://127.0.0.1:8002/check. In the function check_even_odd(), we are calling our first microservice i.e., random_microservice. We are using the requests function and calling our first microservice through its URL i.e., http://127.0.0.1:8001/generate.

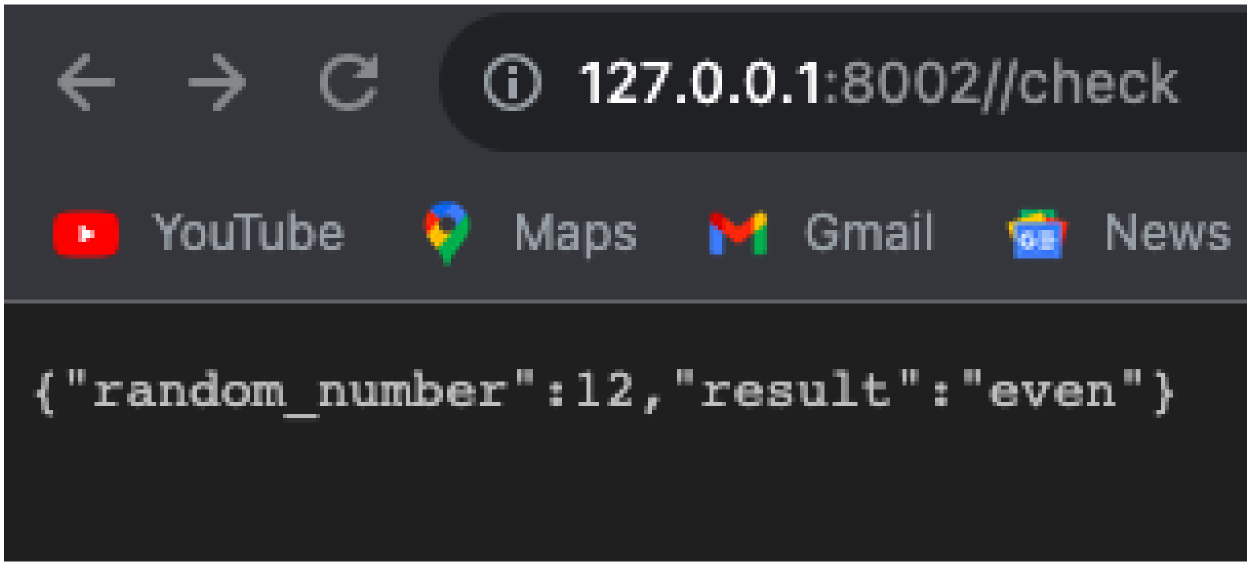
Now that, we have created two microservices. The Even or Odd checker microservice will call the random number generator microservice and get the number and check whether it is even or odd. So make sure you run the `random_microservice.py` first.

After executing `random_microservice.py` in one terminal, proceed to open a new terminal window and run the `evenodd_microservice.py` microservice using the Python command.

```
python evenodd_microservice.py
```

Go to http://127.0.0.1:8002/check in your browser. You can see the JSON output.

Output



```
{"random_number":12,"result":"even"}
```

Even or Odd Checker

Congratulations! Your two microservices have effectively established communication. It's worth noting that in this scenario, we're employing distinct ports (8001 and 8002) for each microservice. This separation ensures that the two microservices operate on separate servers. Keep in mind that there are various approaches to facilitate communication between microservices. In this case, the communication is synchronous, but asynchronous methods can also be employed as an alternative.

B saina... 1

Previous Article **Next Article**

5 Best Java Frameworks For Microservices

Similar Reads

Build Your Own Voice-Activated Calculator Using Python

In this article, you'll be delving into the development of a voice command calculator using Python, taking advantage of libraries like Speech-Recognition...

6 min read

Documenting Flask Endpoint using Flask-Autodoc

Documentation of endpoints is an essential task in web development and being able to apply it in different frameworks is always a utility. This article discusses...

4 min read

How to use Flask-Session in Python Flask ?

Flask Session - Flask-Session is an extension for Flask that supports Server-side Session to your application.The Session is the time between the client logs in to...

4 min read

How to Integrate Flask-Admin and Flask-Login

In order to merge the admin and login pages, we can utilize a short form or any other login method that only requires the username and password. This is know...

8 min read

Minify HTML in Flask using Flask-Minify

Flask offers HTML rendering as output, it is usually desired that the output HTML should be concise and it serves the purpose as well. In this article, we would...

12 min read

Flask URL Helper Function - Flask url_for()

In this article, we are going to learn about the flask url_for() function of the flask URL helper in Python. Flask is a straightforward, speedy, scalable library, used f...

11 min read

How to Build a Simple Android App with Flask Backend?

Flask is an API of Python that allows us to build up web applications. It was developed by Armin Ronacher. Flask's framework is more explicit than Django's...

8 min read

Python | Build a REST API using Flask

Prerequisite: Introduction to Rest API REST stands for REpresentational State Transfer and is an architectural style used in modern web development. It define...

3 min read

Build a Text Translator Web App using Flask and Azure Cognitive Services

We're going to create a website that will translate text into multiple languages using Artificial Intelligence(AI). We'll use Flask framework for the Front end and...

7 min read

Build a Flask application to validate CAPTCHA

In this article, we are going a build a web application that can have a CAPTCHA. CAPTCHA is a tool you can use to differentiate between real users and...

5 min read

Article Tags : Python Flask Projects Microservices Python Flask +1 More

Practice Tags : python

GET IT ON Google Play · Download on the App Store

Company
About Us
Legal
In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program
GeeksforGeeks Community

Languages
Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial
Tutorials Archive

DSA
Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain
All Cheat Sheets

Data Science & ML
Data Science With Python
Data Science For Beginner
Machine Learning
ML Maths
Data Visualisation
Pandas
NumPy
NLP
Deep Learning

Web Technologies
HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Python Tutorial
Python Programming Examples
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question
Django

Computer Science
Operating Systems
Computer Network
Database Management System
Software Engineering

DevOps
Git
Linux
AWS
Docker

Digital Logic Design
Engineering Maths
Software Development
Software Testing

Kubernetes
Azure
GCP
DevOps Roadmap

System Design

High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

Inteview Preparation

Competitive Programming
Top DS or Algo for CP
Company-Wise Recruitment Process
Company-Wise Preparation
Aptitude Preparation
Puzzles

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

GeeksforGeeks Videos

DSA
Python
Java
C++
Web Development
Data Science
CS Subjects