Microservices

Objectives: By the end of this assignment, the student will be able to create, deploy and test Web-service application.

Microservices:

* The way developer worked to build application is changing. In the past, software was built as large monolithic application where a team developers would take months to construct a large application built on a common code base.
* Define an architecture that structures the application as a set of loosely coupled, collaborating services. Each service implements a set of narrowly, related functions.
* For example, an application might consist of services such as the order management service, the customer management service etc.
* Each service has its own database in order to be decoupled from other services. Data consistency between services is maintained .

Microservices are also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are:

* Highly maintainable and testable
* Loosely coupled
* Independently deployable
* Organized around business capabilities.
* Traditional application design is often called “monolithic” because the

whole thing is developed in one piece.

Monolithic vs microservice

* Even if the logic of the application is modular it’s deployed as one group, like a Java application as a JAR file for example.
* This monolith eventually becomes so difficult to manage as the larger applications require longer and longer deployment timeframes.
* In contrast, a team designing a microservices architecture for their application will split all of the major functions of an application into independent services.
* Each independent service is usually packaged as an API so it can interact with the rest of the application elements.

Web framework

* Traditional application design is often called “monolithic” because the

whole thing is developed in one piece.

* Even if the logic of the application is modular it’s deployed as one group, like a Java application as a JAR file for example.
* This monolith eventually becomes so difficult to manage as the larger applications require longer and longer deployment timeframes.
* In contrast, a team designing a microservices architecture for their application will split all of the major functions of an application into independent services.
* Each independent service is usually packaged as an API so it can interact with the rest of the application elements.
* **Flask**
* Flask (source code) is a Python web framework built with a small core and easy-to-extend philosophy. Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine.
* **WSGI**
* Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.
* **Werkzeug**
* It is a WSGI toolkit, which implements requests, response objects, and other utility functions. This enables building a web framework on top of it. The Flask framework uses Werkzeug as one of its bases.

Virtual Environment

* In Python, by default, every project on the system will use the same directories to store and retrieve site packages (third party libraries) and system packages (packages that are part of the standard Python library).
* Consider the a scenario where there are two projects: ProjectA and ProjectB, both have a **dependency on the same library**, ProjectC. The problem becomes apparent when we start requiring **different versions** of ProjectC.

Maybe ProjectA needs v1.0.0, while ProjectB requires the newer v2.0.0, for example.

Since projects are stored in site-packages directory according to their name and can’t differentiate between versions, both projects, ProjectA and ProjectB, would be required to use the same version which is unacceptable in many cases and hence the virtual environment.

Here, we are attempting to develop a microservice based architecture for Movie ticket Booking web application. The services are being implemented using python and JSON is used as for Data Store.

Implementing the solution

**Using virtual Environments:** Install virtualenv for development environment. virtualenv is a virtual Python environment builder. It helps a user to create multiple Python environments side-by-side. Thereby, it can avoid compatibility issues between the different versions of the libraries.

The following command installs virtualenv:

sudo apt-get install virtualenv

* **Flask Module**: Importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of current module ( name ) as argument. The route() function of the Flask class is a decorator, which tells the application which URL should call the associated function.
* **Route Decoder:**
* The route() decorator in Flask is used to bind URL to a function.
* For example −
* @app.route(‘/hello’) def hello\_world():
* return ‘hello world’
* Here, URL ‘/hello’ rule is bound to the hello\_world() function. As a result, if a user visits http://localhost:5000/hello URL, the output of the hello\_world() function will be rendered in the browser.
* **Writing the subroutine for the four microservices:** There are four microservices viz., user, Showtimes, Bookings and Movies for which microservices are to be implemented.

Expected output

* To install the necessary files and create a virtual environment run:

sudo ./setup.sh

* To start the 4 microservices run :

./startup.sh

* To start the command line UI:

python cmdline.py

Conclusion: Due to web services , developer can expose the functionality of existing code over the network. So here we have created and deploy such a micro-service.