Name: Subhasish chatterjee

Class: BE IT

Roll: 02

Computer Laboratory -IX

Assignment 7

Problem Statement:

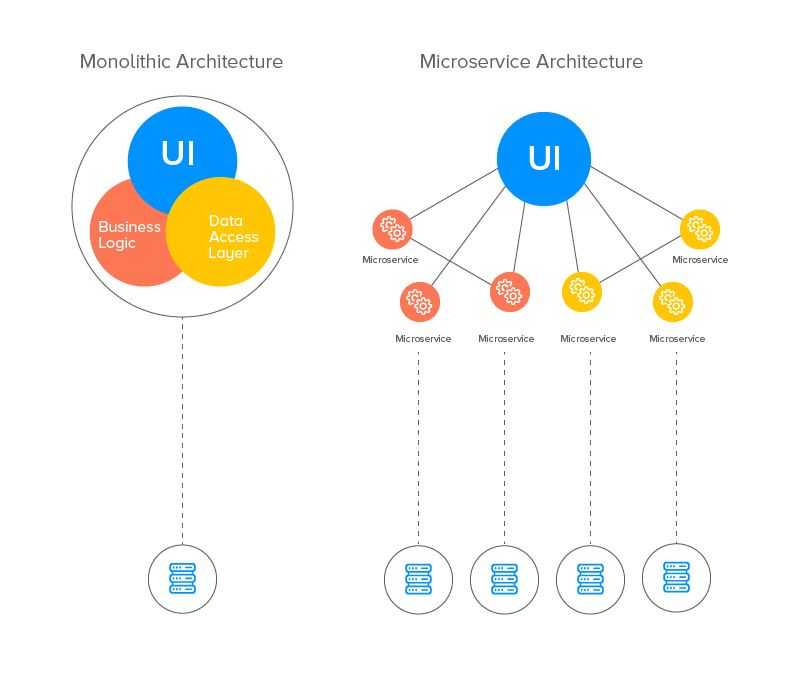
To develop microservices framework based distributed application.

Tools / Environment:

Python 3.6.0 using Flask framework.

Related Theory:

1. Microservices: 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 with the monolith type application, here’s what an app developed with a microservices focus might look like:



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.

Microservices - 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.

The microservice architecture enables the continuous delivery/deployment of large, complex applications. It also enables an organization to evolve its technology stack. 2. Web frameworks encapsulate what developers have learned over the past twenty years while programming sites and applications for the web. Frameworks make it easier to reuse code for common HTTP operations and to structure projects so other developers with knowledge of the framework can quickly build and maintain the application. Common web framework functionality: Frameworks provide functionality in their code or through extensions to perform common operations required to run web applications. These common operations include:

1. URL routing 2. Input form handling and validation 3. HTML, XML, JSON, and other output formats with a templating engine 4. Database connection configuration and persistent data manipulation through an objectrelational mapper (ORM) 5. Web security against Cross-site request forgery (CSRF), SQL Injection, Cross-site Scripting (XSS) and other common malicious attacks 6. Session storage and retrieval.

3. 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.

4. 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.

5. 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.

6. 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 of which 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 just 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. The main purpose of Python virtual environments is to create an isolated environment for Python projects. This means that each project can have its own dependencies, regardless of what dependencies every other project has. There are no limits to the number of environments you can have since they’re just directories containing a few scripts. Plus, they’re easily created using the virtualenv or pyenv command line tools.

Designing the solution:

Here, we are attempting to develop an 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:

1. 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

1. 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 decorator: 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.

1. Writing the subroutine for the four microservices: There are four microservices viz., user, Showtimes, Bookings and Movies for which microservices are to be implemented.

Building and Executing the solution:

1. To install the necessary files and create a virtual environment run: sudo ./setup.sh

2. To start the 4 microservices run : ./startup.sh

3. To start the command line UI: python cmdline.py

Conclusion:

With microservices, modules within software can be independently deployable. In a microservices architecture, each service runs a unique process and usually manages its own database. This not only provides development teams with a more decentralized approach to building software, it also allows each service to be deployed, rebuilt, redeployed and managed independently. Netflix, eBay, Amazon, the UK Government Digital Service, Twitter, PayPal, The Guardian, and many other large-scale websites and applications have all evolved from monolithic to microservices architecture.