**Universidad Distrital Francisco José de Caldas**

**Facultad de Ingeniería**

**Especialización en Ingeniería de Software**

**Asignatura de Informática 1**

**Docente: Alejandro Paolo Daza**

****

|  |  |
| --- | --- |
| Nombre: Angee Paola Ballesteros Maldonado | Código: 20201099027 |
| Nombre: Luigi Santiago Fajardo Toloza | **Código:** 20201099029 |
| Nombre: Jeisson Jair Ariza Pulido | **Código:** 20201099026 |

**Diseño de un aplicativo aritmético mediante el uso de Microservicios con el Framework Flask**

**Requerimiento:** Se plantea establecer una arquitectura orientada a microservicios usando el micro-framework flask para ofrecer la implementación de los servicios de suma, resta, multiplicación y división. En el repositorio se deben exponer las tecnologías seleccionadas para ofrecer los servicios propios de este tipo de arquitecturas teniendo en cuenta que las operaciones se ofrecen con flask. La explicación de despliegue, así como la explicación de las herramientas seleccionadas debe ser clara en el README del proyecto.

**Procedimiento:** Inicialmente debemos conocer que Flask es un micro-framework de Python, por este motivo debemos tener instalado y de manera funcional Python 3 en la maquina en la cual se va a desarrollar el requerimiento. Este procedimiento puede ser consultado en [1], adicional a esto, si se va a trabajar en una maquina con un SO basado en Linux(i.e. Mac OS o las distribuciones de Linux), se sugiere que aplique el cambio de alias para Python.

Para el desarrollo de los servicios se utilizará el IDE Visual Studio Code, dicho IDE permite integrar todos los componentes necesarios de Python para el correcto desarrollo del requerimiento.

Se solicita la creación de 4 servicios para cada una de las operaciones aritméticas(suma, resta, multiplicación y división), adicionalmente se requiere un aplicativo web que permita consumir cada uno de estos servicios cuando el usuario lo solicite.

Antes de iniciar debemos saber el porqué se utiliza Flask como micro-framework de nuestro requerimiento, además debemos mencionar el uso de herramientas adicionales como son Flask-CORS para el intercambio de recursos de origen cruzado, Request Flask para el contenido que un cliente web manda para almacenar al servidor, Waitress para el Web Server Gateway Interface, por ultimo, Docker, utilizado para el despliegue de [aplicaciones](https://es.wikipedia.org/wiki/Aplicaci%C3%B3n_inform%C3%A1tica) dentro de [contenedores de software](https://es.wikipedia.org/wiki/Contenedores_de_software). A continuación, se especifican las características que nos llevaron a la elección de estas herramientas, de igual manera en los casos en los que consideramos necesario se adjuntan estrategias para la implementación de estas herramientas.

1. **Flask:** este micro-framework permite construir aplicaciones web y servicios Restful con Python de una manera simple, de forma tal que utilizando pocas líneas podemos llegar a tener un servicio Restful funcionando correctamente. Se podría decir que su mayor característica consiste en que permite crear rutas web de una forma muy sencilla. Entre sus características principales tenemos que, al ser micro, nos permite que en una sola pagina se pueda alojar una aplicación web.
   1. **Instalación de Flask:** como se mencionaba previamente, es imprescindible contar con una instalación de Python en nuestra maquina de desarrollo, esto debido a que Flask es un micro-framework de Python y además su instalación se realiza mediante el administrador de paquetes de Python (pip). Es por esto que el primer paso y la mayoría de las instalaciones utilizadas en el desarrollo de este requerimiento se hacen mediante pip. Para instalar flask debemos ejecutar la siguiente línea en nuestra terminal **$ pip install Flask**. Por otra parte, es recomendable que cada proyecto se ejecute en un entorno virtual independiente para garantizar que no afecte ni sea afectado por otros proyectos, para esto es necesario instalar el virtualenv con el comando **$ pip install virtualenv** y posteriormente crear un directorio para el proyecto con el comando **$ mkdir miproyecto,** luego debemos dirigirnos al directorio con **$ cd miproyecto** y crear el entorno virtual del proyecto con el comando **$ virtualenv mientornovirtual**, por ultimo se debe activar el entorno virtual con el comando **$ . mientornovirtual/bin/actívate,**  no sin recordar que al terminar nuestro desarrollo es recomendable desactivar este mismo con el comando **$ deactivate.**
2. **Flask-CORS:** extensión de Flask que permite manejar el intercambio de recursos de origen cruzado(CORS), su principal característica radica en su filosofía de manejo simple, esto debido a que cuando el desarrollador desea habilitar CORS, desea habilitarlo para todos los casos de uso en un dominio. Esto significa que no hay que perder el tiempo con diferentes encabezados, métodos permitidos, entre otros, por lo que este paquete expone una extensión Flask que, de una manera predeterminada, habilita el soporte de CORS en todas las rutas, para todos los orígenes y métodos. Adicionalmente, permite la parametrización de todos los encabezados CORS en un nivel por recurso. La instalación se ejecuta mediante el comando **$ pip install -U flask-cors,** mientras que para importar a nuestro desarrollo es necesario ejecutar en nuestro desarrollo el comando **from flask\_cors import CORS, cross\_origin.**
3. **Request Flask:** utilizado para almacenar el contenido que un cliente web envía al servidor del aplicativo en cuestión, existen diferentes tipos de request dependiendo de la petición que haga el cliente al aplicativo, entre los ejemplos se encuentran GET, POST, DELETE, entre otros. Para hacer uso del Request dentro de Flask se debe utilizar el comando **from flask import request** dentro del desarrollo.
4. **Waitress:** los Web Server Gateway Interface son las especificaciones que describen cómo se comunica un servidor web con una aplicación web y cómo se pueden llegar a encadenar diferentes aplicaciones web para procesar una solicitud/petición (o request). Python posee una gran variedad de servidores WSGI puros, pero Waitress esta destinado a ser útil para los autores de marcos web que requieren un amplio soporte de plataforma. Por otra parte, waitress no tiene dependencias, excepto las que viven en la biblioteca estándar de Python. La instalación de waitress se ejecuta mediante el comando **$ pip install waitress.**
5. **Docker:** Es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos, mediante el uso de características de aislamiento de recursos del kernel Linux, tales como cgroups y espacios de nombres, permite que "contenedores" independientes se ejecuten dentro de una sola instancia de Linux, evitando la sobrecarga de iniciar y mantener máquinas virtuales.

Luego de esta breve explicación sobre las características de los recursos a utilizar procedemos a explicar los pasos realizados en el desarrollo del requerimiento:

Primero se deben construir los servicios para cada una de las operaciones aritméticas, cada uno de estos se aloja en una carpeta independiente. Como se puede observar a continuación la estructura del proyecto contiene una carpeta para cada servicio.

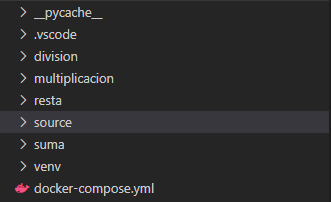


Figura 1: Estructura del Proyecto

Dentro de cada una de las carpetas se tiene el servicio de cada operación aritmética, el archivo docker y los requerimientos necesarios.

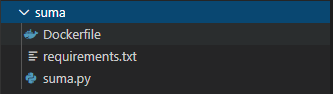


Figura 2: Estructura de los servicios

Cada uno de los servicios debe importar las herramientas necesarias para su correcto funcionamiento en el desarrollo, primero se debe importar Flask, render\_template, request, jsonify, posteriormente flask\_cors y waitress y por último json, requests y traceback.

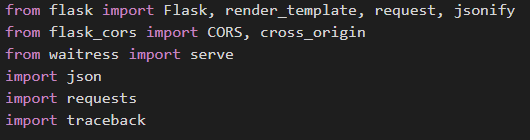


Figura 3: Importación de las herramientas

A continuación, se muestra la lógica utilizada en el llamado de los servicios y adicionalmente la definición del host que alojara la aplicación principal del desarrollo.



Figura 4: Lógica para llamar a los servicios



Figura 5: Definición del Host (waitress)

Posteriormente se define la lógica de cada uno de los servicios, en la siguiente figura se observa la lógica del servicio de suma desarrollado.



Figura 6: Lógica de los servicios (suma)

Luego de desarrollar cada uno de los servicios es necesario crear los contenedores de los mismos, por esta razón, en todos los servicios es necesario crear un archivo Dockerfile que se encarga de crear un orden de las tareas necesarias para que los servicios que se van a ejecutar operen de manera correcta sin necesidad de hacer instalaciones o pasos intermedios.

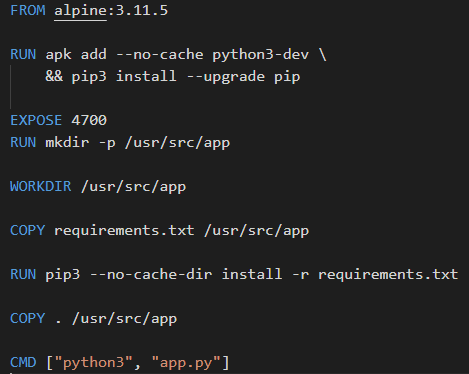


Figura 7: Dockerfile (app)

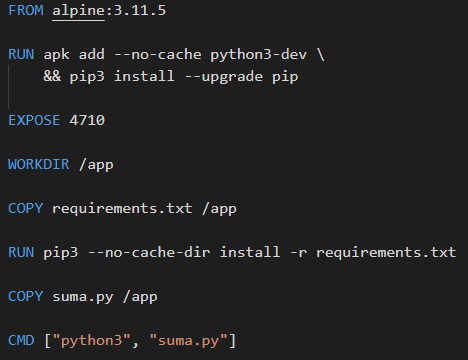


Figura 8: Dockerfile Servicios (suma)

Después de crear cada uno de los archivos Dockerfile para cada servicio se procede a crear el archivo “docker-compose.yml”, dicho archivo es el encargado de realizar de manera automática la creación de las imágenes y dejar en contenedores los servicios listos para su despliegue, en este archivo especificamos los puertos y los nombres de los contenedores para que se puedan comunicar entre ellos.



Figura 9: docker-compose.yml

Posteriormente se valida el correcto funcionamiento de la herramienta Docker Desktop.

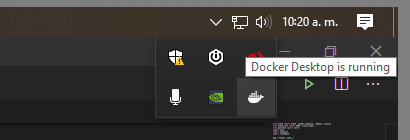


Figura 10: Validación Docker Desktop is Running

Después de ejecutar la instrucción “docker-compose up” podemos ver como los contenedores son desplegados.

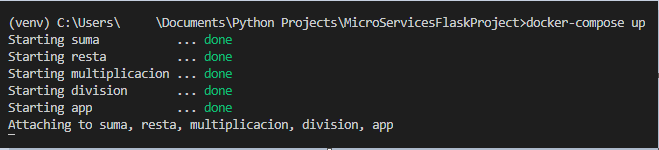


Figura 11: Servicios Iniciados

**Enlaces de interés:**

[1] <https://www.python.org/downloads/>