**Marco teórico de Tecnologías de Microservicios**

A continuación, un breve fundamento teórico de las siguientes tecnologías:

* Flask
* Flask Cors
* Live-Server
* Docker

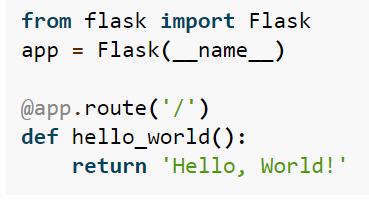
**Flask**

En microframewors, la palabra Micro se refiere a que Flask ayuda a mantener el core de una aplicación web de manera simple pero extensible. Por defecto, flask no incluye una capa de abstracción de bases de datos o formularios de validación. En vez de esto, Flask admite extensiones para adicionar tales funcionalidades a las aplicaciones como si estas fueran implementadas en Flask por sí mismo. Flask puede ser “micro” pero está listo para su uso en producción en una variedad de necesidades.

Flask tienen muchos valores de configuración, con valores por defecto y convenciones, Por convención, plantillas y archivos estáticos sin almacenados en subdirectorios dentro del árbol fuente de Python de la aplicación, con los nombres *template* y *static* respectivamente.

Una vez que Flask esté funcionando, encontraras una variedad de extensiones disponibles en la comunidad para integrar tu proyecto para producción, de la manera personalizada más posible.

Una aplicación mínima en Flask debe parecerse a:



Este código realiza lo siguiente:

1. Se importó una clase Flask.
2. Se creó una instancia de esta clase. El primer argumento es el nombre del módulo de la aplicación o el paquete. Si se importa como aplicación será \_\_name\_\_, si es un módulo será \_\_main\_\_
3. Se usa route() para decir que URL debe desencadenar la función.
4. La función retorna un mensaje a mostrar al usuario.

Una aplicación web de Python basada en WSGI tiene que tener un objeto central llamable (callable) que implementa la actual aplicación. En Flask esta es una instancia de la clase Flask. Cada una de las aplicaciones de Flask tiene que crear una instancia de esta clase por si mismo y lo pasa al nombre del modulo.

Flask usa el sistema de enrutamiento Werkzeug el cual fue diseñado para automáticamente ordenar routes por complejidad. Eso significa que puedes declarar los enrutadores aleatoriamente, y aun así ellos trabajarán como se esperaba. El motor de plantillas sobre el que decide Flask es Jinja2.

Flask es un framework que toma ventajas del trabajo ya realizado por Werkzeug para interfaces apropiadas WSGI.

Flask utiliza objetos locales de hilo para requerimientos, sesiones y un objeto extra que puedes poner en tus propios desarrollos, con el fin de hacer más fácil y rápido escribir la aplicación web.

Flask nunca tendrá una capa de base de datos, o librerías. Flask por sí mismo solo conecta a Werkzeug para implementar una adecuada aplicación WSGI y a Jinja2 para manejar plantillas.

**Flask Cors**

Este paquete tiene una filosofía sencilla, cuando tu deseas habilitar CORS, tu deseas habilitarlo para todos los casos de uso en un dominio. Así no se pierde el tiempo con encabezados, métodos, etc. Por defecto, la sumisión de cookies a través de los dominios es deshabilitado debido a las implicaciones de seguridad.

Este paquete expone una extensión de Flask que por defecto habilita el soporte CORS en todas las rutas, para todos los orígenes y métodos. Permitiendo la parametrización de todos los encabezados CORS en un nivel por recurso. El paquete incluye un decorador, para quienes prefieren ese enfoque.

Se puede inicializar por default o especificar opciones CORS en un nivel granular de recursos y orígenes, al pasar un diccionario como las opciones de recursos.

**Live-Server**

Es un pequeño servidor de desarrollo con capacidad de recarga en vivo. Útil para aplicaciones HTML, JavaScript, CSS, pero no para implementar el sitio final.

Dos principales razones para utilizarlo:

1. Requerimientos AJAX no trabajan con el protocolo file:// debido a restricciones de seguridad, en ese caso, necesitaras un servidor si el sitio obtiene contenido a través de JavaScript.
2. El hacer que la página se recargue automáticamente puede acelerar el desarrollo.

No se requiere instalar ningún complemento del navegador o agregar manualmente fragmentos de código a sus páginas para que la funcionalidad de recarga se ejecute.

El servidor es una simple aplicación nodo que sirve al directorio de trabajo y sus subdirectorios. Observa los archivos para cambios y cuando eso ocurre, envía un mensaje a través de una conexión de socket web al navegador instruyéndolo para recargar.

A fin de que el lado del cliente admita esto, el servidor inyecta un pequeño código de JavaScript a cada archivo HTML solicitado. Este script establece la conexión web socket y escucha a las solicitudes de recarga. Los archivos CSS se pueden actualizar sin una recarga completa de la página buscando las hojas de estilo referenciadas del DOM y engañando al navegador para buscarlas y analizarlas nuevamente.

**Docker**

Su enfoque primario es automatizar el despliegue de aplicaciones dentro de los contenedores de software y la automatización de la virtualización a nivel de sistema operativo en Linux. Es más liviano que los contenedores estándar y se inicia en segundos.

Docker, es un software de TI, es una tecnología de creación de contenedores que permite la creación y el uso de contenedores de Linux.

La comunidad de código abierto Docker trabaja para las tecnologías a fin de beneficiar a los usuarios gratuitamente.

Docker Inc., es una empresa que desarrolla la comunidad Docker, lo hace más seguro y comparte estos avances con el resto de la comunidad. Apoya tecnologías mejoradas para empresas.

Docker usa los contenedores como máquinas virtuales livianas y modulares. Estos contenedores son flexibles, permiten crearlos, implementarlos, copiarlos y moverlos de un entorno a otro, lo que optimiza las aplicaciones en la nube.

La tecnología de Docker usa el kernel de Linux y sus funciones (Cgroups y namespaces) para segregar los procesos, para que se ejecuten de manera independiente. El propósito de los contenedores es esta independencia, es decir, la capacidad para ejecutar varios procesos y aplicaciones por separado para hacer un mejor uso de la infraestructura, y mantener la seguridad de se tuviera con sistemas separados.

Docker se basa en un modelo de implementación basada en imágenes, lo que permite compartir una aplicación o un conjunto de servicios, con todas sus dependencias en varios entornos. Docker automatiza la implementación de la aplicación o el conjunto de procesos que conforman una aplicación, en este entorno de contenedores.

Docker aporta la capacidad de ejecutar contenedores, facilita la creación y diseño de contenedores, envío de imágenes y creación de versiones de imágenes.

Los contenedores de Linux usan un sistema init que gestiona varios procesos, así las aplicaciones completas se pueden ejecutar como una sola. Docker pretende que las aplicaciones se dividan en sus procesos individuales y ofrece las herramientas para hacerlo.

**Ventajas de Docker**

**Modularidad**

Docker tiene la capacidad de tomar una parte de la aplicación, actualizarla o repararla, sin necesidad de tomarla completa. Este enfoque basado en micro servicios, puede compartir procesos entre varias aplicaciones, como funciona la arquitectura orientada al servicio.

**Control de versiones de imágenes y capas**

Cada imagen de Docker se compone de una serie de capas, las capas se combinan en una sola imagen. Una capa se crea cuando la imagen cambia, al ejecutar un comendo de ejecutar o copiar.

Docker reutiliza estas capas para construir nuevos contenedores, lo cual acelera el proceso de construcción. Los cambios intermedios se comparten entre imágenes, mejorando la velocidad e ineficiencia. El control de versiones es inherente a la creación de las capas. Ante un cambio nuevo, se tiene un registro de cambios incorporados, lo que permite tener un control completo de sus imágenes de contenedor.

**Restauración**

Se puede restaurar las capas de una imagen a su versión anterior, lo cual es compatible con el desarrollo ágil y permite hacer realidad la integración e implementación continuas CI/CD desde las herramientas.

**Implementación rápida**

Antes el despliegue de una aplicación demoraba días, Docker reduce el tiempo de implementación en segundos. Al crear un contenedor, puede compartir procesos con nuevas aplicaciones.

La tecnología Docker es un enfoque más granular y controlable, basado en micro servicios, que prioriza la eficiencia.

**Referencias**

RedHat, Capitulo: ¿Que es docker? Tomado de: <https://www.redhat.com/es/topics/containers/what-is-docker>

Npm, Titulo: Live-Server. Tomado de: <https://www.npmjs.com/package/live-server>

Flask, Seccion: Foreword. Tomado de: <https://flask.palletsprojects.com/en/1.1.x/foreword/>

Flask, Seccion: Welcome, User’s Guide. Tomado de: <https://flask.palletsprojects.com/en/1.1.x/>

Flask, Seccion: QuickStart. Tomado de: <https://flask.palletsprojects.com/en/1.1.x/quickstart/#quickstart>