**PARTE 1: INTRO FLASK HTML PLOTLY**

En esta primera parte se va a mostrar cómo desarrollar localmente (en un ambiente de anaconda) una aplicación web enfocada en usos de analítica avanzada / data science utilizando principalmente Python y Html (con un poco de javascript). De Python se van a utilizar librerías de desarrollo de backend como Flask, librería de análisis de datos como pandas y librería de visualización de datos en este caso plotly.

Luego de tener desarrollada localmente la app web se va a proceder a ser subida a cloud para su disponibilización para el usuario a través de empaquetar los códigos en una imagen Docker, posteriormente ser subida a Artifact Registry para finalmente ser deployada en cloud run para el usuario final.

Además se va a mostrar cómo la app web puede interactuar con diferentes servicios de GCP utilizados frecuentemente para analítica avanzada.

Todo este contenido será mostrado en forma gradual en cada uno de estos puntos.

**EJEMPLO 1: Hola Mundo – solo flask**

**DESCRIPCIÓN**

En este ejemplo se muestra:

- crear una app web “hola mundo” utilizando SOLO flask. Flask se encarga de hacer la api/server/backend donde únicamente basta con crear un script de Python “app.py” para tener dicho backend

- probar la vista de la app web localmente trabajando en un ambiente de anaconda

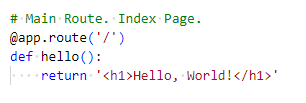
- empaquetar los códigos en una imagen de docker y ser subidos a artifact registry. Desde artifact registry hacer el deploy a cloud run y así tener alojado la app web (desarrollada y probada localmente) en un ambiente cloud

**PASOS**

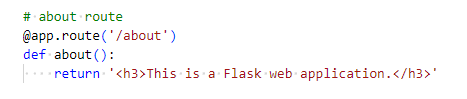
**1.- Crear script “app.py” con el backend de la app web.**

En este primer ejemplo aún no se va a incluir código html para el front por lo que al ver la app solo se va a mostrar el texto que se definió mostrar en el script app.py

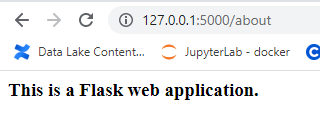
Básicamente cada página creada va a ser a través del decorador “app.route y la url de la web”. A continuación se define una función de Python donde cada función en la route van a retornar una respuesta http a ser mostrada. **IMPORTANTE: LA FUNCIÓN QUE SE DEFINA DESPUÉS DEL DECORATOR ES LA QUE SE VA A UTILIZAR PARA CREAR LA API DE LA ROUTE DEFINA** (es decir, si se escribe un decorator y abajo 2 funciones, la función que esté inmediatamente después del decorator es la que se va a utilizar para la route)



Si se quiere crear otra página se debe proceder de la misma forma agregando la url correspondiente. Por ejemplo (obs: cuando haga del deploy de la app no existe un mecanismo para interactuar y cambiar de página por lo que hay que entrar a ingresar la url manualmente)



Cómo se ve la app web



De forma completa el código flask se vería de la siguiente manera



**2.- Probar localmente la app web**

En este paso se procede a probar localmente la app web y verificar que funciona todo bien.

**Importante: cada vez que se hace un cambio hay que “cerrar el server” y volver a correr flask para poder visualizar en la UI el cambio hecho en los códigos**

Para probar localmente utilizando anaconda se procede a hacer lo siguiente:

- Abrir terminal de anaconda

- Pararse en la carpeta donde está el script “app.py”.En mi caso

- d:

- cd Simple-app-web-for-datascience

- cd 1\_hello\_world

- Correr líneas de códigos para ver la app en local (reemplazar export de linux por set de windows). En mis pruebas solo fue necesario correr la última línea para probar localmente “flask run”

- set FLASK\_APP=app

- set FLASK\_ENV=development

- flask run

- La app corre localmente en la URL. En la consola también indica esta url, solo basta con copiar (control + Y) y pegar en el navegador

**http://127.0.0.1:5000/**

**3.- Subir la app web a cloud run y ver la app online**

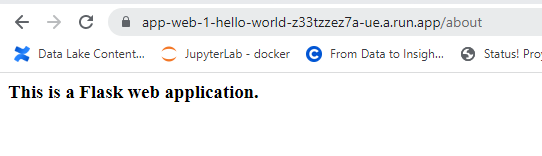
Para poder subir la app a cloud run se necesita empaquetar los códigos en una imagen de Docker. Luego de tener la imagen docker se puede subir esta imagen a artifact registry y posteriormente hacer el deploy a cloud run.

Este proceso se hace mediante códigos que se pueden ejecutar en la consola de GCP. Para este ejemplo más amigable con data scientist familiarizados con el uso de jupyter notebooks se va a utilizar el notebook

“deploy.ipynb”

El cual se va corriendo para hacer los diferentes pasos mencionados anteriormente (obs: en este ejemplo el notebook de deploy contiene dos versiones de deploy la actual utilizando artifact registry y una versión anterior utilizando container registry ya cada vez más deprecated)

Ejemplo web app en cloud run



**Estos pasos son los mismos para cualquier prueba y deploy. Por lo que los ejemplos siguientes profundizar en el diseño e interacciones de la app web pero las pruebas locales y deploy en cloud son estos mismos pasos**

FUENTES:

Flask básico: <https://alan-jones.medium.com/how-to-create-and-run-a-flask-app-533b7b101c86>

Documentación cómo subir app.py a cloud run utilizando cloud build y container registry:

    - cómo crear dockerfile y subirlo a container registry: <https://firebase.google.com/docs/hosting/cloud-run?hl=es-419#python>

    - Mini cursos google: <https://www.cloudskillsboost.google/focuses/10445?parent=catalog>

<https://codelabs.developers.google.com/codelabs/cloud-run-deploy#3>

<https://medium.com/google-cloud/hosting-a-website-on-google-cloud-using-cloud-run-a65343a98fce>

Documentación para integración cloud build con Artifact Registry:

    - integracíon cloud build / artifact registry: <https://cloud.google.com/artifact-registry/docs/configure-cloud-build?hl=es-419>

    - crear repo en artifact registry: <https://cloud.google.com/artifact-registry/docs/repositories/create-repos#gcloud>

Documentación Ejemplo subir app.py a cloud run utilizando cloud build y artifact registry: <https://medium.com/@ThatJenPerson/getting-started-with-artifact-registry-deploying-to-cloud-run-7aa9f2c65d07>

Documentación comando gcloud run deploy (para subir imagen a container registry): <https://cloud.google.com/sdk/gcloud/reference/run/deploy>

Documentación con ideas para profundizar de la autentificación de una web en un cloud run: <https://www.youtube.com/watch?v=5r21CVd6nwo>

**EJEMPLO 2: hola mundo html**

Se va a continuar con el ejemplo anterior, pero a continuación se va crear código HTML para mostrar las dos páginas creadas (index y about).

De esta forma se va a mostrar cómo se crea la API en Flask y el front en HTML y cómo estos interactúan entre si.

**DESCRIPCIÓN:**

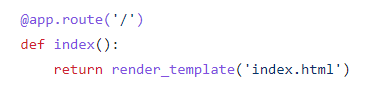
- Los archivos html de las diferentes páginas se guardan en la carpeta **“templates”** que están en el root junto con el script “app.py”

- El nombre de la página se define en route y si en un código html (por ejemplo un índice) se define la url de una página, esta debe de coincidir con los nombres definidos en la API



- Para poder conectar flask y html se utiliza la función **“render\_template”.** La cual va en el return de la función que hace la API de esa página

Es decir, se hace el código de la API de esa página y en el return que es lo que se va a mostrar se utiliza el código render\_template para indicar el código html de esa página web.



Obviamente a esta función “render\_template” se le pueden pasar diferentes variables. Es decir, se pasa en primer lugar la página html y luego las diferentes variables http que se van a utilizar en el código html. 

Por ejemplo, en la imagen se pasa la variable header con el valor header. Donde la variable va a estar con ese nombre en el código html y el valor es el que se calculó en la API

Siguiendo esta lógica, las variables también se pueden definir de la forma



Donde “**table\_html**” es el nombre de la variable y esta variable es el nombre con el que se tiene que llamar en el código html template, mientras que **“df\_html”** es el valor y es una variable que se utiliza en la api de flask

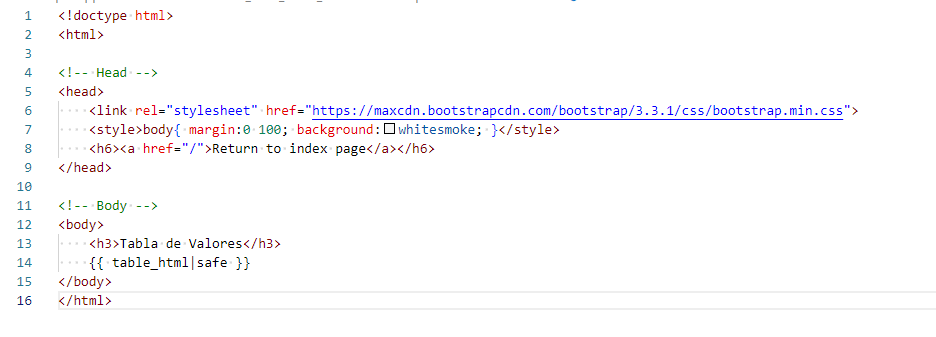
- Para insertar la variable pasada en la API en el HTML de destino se debe de utilizar doble comillas de la forma

**{{**message**}}**



- Para mostrar un dataframe como una tabla en una app web se necesita transformar dicha variable como html se debe de agregar “safe” (porque a diferencia de las variables que son un string y se pueden insertar de forma sencilla en el html, las variable que son un html y que se quieren agregar en otro html se necesita utilizar safe) de la forma:

**{{** variable|safe **}}**



- Cómo agregar un archivo de estilos csv custom. En el post <https://hugobottois.medium.com/simple-kpi-dashboard-877f3068f872> se explica una forma de tener múltiples cajas de kpis solo con html + css custom. Para que se pueda ver en flask se necesita crear la carpeta

static/styles/{archivo.css}

y agregar en el código html que va a utilizar dicho css

<link rel= "stylesheet" type= "text/css" href= "{{ url\_for('static',filename='styles/mainpage.css') }}">

Tal como se muestra: <https://stackoverflow.com/questions/22259847/application-not-picking-up-css-file-flask-python>



- FUENTES

- Multiples recursos html: <https://www.w3schools.com/html/html_tables.asp>

- Tutorial básico flask + html: <https://www.digitalocean.com/community/tutorials/how-to-use-web-forms-in-a-flask-application#step-3-handling-form-requests>

- Tutorial flask + html + plotly: <https://towardsdatascience.com/web-visualization-with-plotly-and-flask-3660abf9c946>

- Ejemplo como hacer reportes analíticos en plotly: <https://plotly.com/python/v3/html-reports/>

- Insertar dataframe html a html principal: <https://stackoverflow.com/questions/71388099/how-do-i-make-a-dataframe-table-to-show-in-html>

- Hacer cajas para valores de KPIS directamente en html + css: <https://hugobottois.medium.com/simple-kpi-dashboard-877f3068f872>

- Cómo agregar archivos css personalizados de estilos en flask: <https://stackoverflow.com/questions/22259847/application-not-picking-up-css-file-flask-python>

FALTA POR HACER – INTERACCIONES CON EL USUARIO

- <https://towardsdatascience.com/an-interactive-web-dashboard-with-plotly-and-flask-c365cdec5e3f>

**EJEMPLO 3: Hola mundo con GCP**

Este ejemplo, toma el ejemplo 2 **específicamente la creación de tablas.**

Realiza una modifcación a ese ejemplo donde en lugar de crear una tabla en la API se va a **BIGQUERY A TRAVÉS DE UNA CONSULTA SQL A OBTENER LOS DATOS A MOSTRAR EN LA APP**

Para poder realizar esto se necesita:

**-** Se crea un dataset y una tabla de BIGQUERY

- Se consulta dicha tabla para generar el dataframe que va a ser mostrado en la app web

- Se necesitan los permisos necesarios para poder que la API interactue con los diferentes recursos de GCP

**DESCRIPCION**

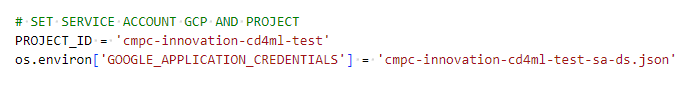
Para poder consultar la tabla de BQ se necesitan los permisos suficientes. Cuando se trabaja en anaconda probablemente ya esté seteada la SA y se tengan los permisos para interactuar con los servicios de BQ.

Pero cuando se sube la app web a cloud run la API no tiene los permisos por lo que deben de hacerse algunos cambios:

- Obtener el json de la SA



- Setear como variable de ambiente el JSON con la SA



- Para hacer deploy (empaquetar los códigos en la imagen docker) en las depencias que se necesitan se deben de agregar:

google-cloud-bigquery

db-dtypes



FUENTES:

- Setear JSON de la SA en variable de ambiente: <https://stackoverflow.com/questions/44328277/how-to-auth-to-google-cloud-using-service-account-in-python>

**EJEMPLO 4: Hola mundo con plotly para data science**

Este ejemplo es LA CONTINUACIÓN DEL EJEMPLO ANTERIOR de tabla dataframe (conectado con BQ) y se agregan páginas con gráficos de plotly donde los datos también se obtiene desde BQ:

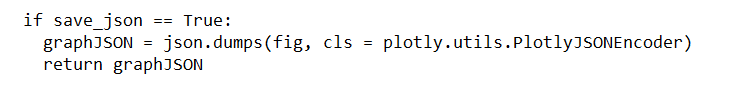
- página con un gráfico de plotly

- página con dos gráficos de plotly

**DESCRIPCION**

Al graficar en plolty para poder obtener el gráfico para agregarlo a html se debe:

- Se debe de obtener el gráfico en formato json utilizando el código



- Luego para insertar el gráfico json en html se debe insertar código de JS para procesar ese json y poder mostrarse en el UI.

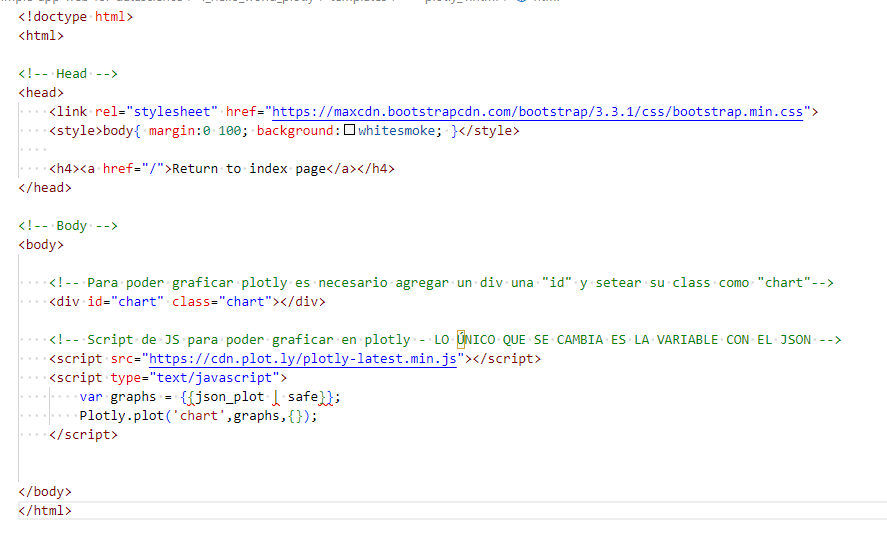




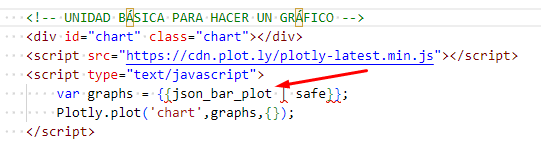
Además antes es necesario definir un div.

<div> element that will contain the Plotly chart (for Plotly to work we need to give this <div> an id and set its class attribute to chart).

Mi Ejemplo con una página con un gráfico



resumido



SE MANTIENE TODO EL CÓDIGO Y SOLO SE DEBE DE CAMBIAR LA VARIABLE QUE CONTIENE EL JSON A GRAFICAR. En el ejemplo “json\_bar\_plot”

- Cuando se quiere graficar múltiples gráficos en la misma página se deben de hacer pequeñas modificaciones a los scripts de JS.

Para graficar 2 o más gráficos de plotly: El “id” debe ser modificado para que sea un id único por gráfico y además cuando se llama el método de plotly que grafica, se le debe de pasar como argumento el mismo valor definido en el id



FUENTES:

La principal fuente es la que indica cómo integrar un grafico de plotly en formato json a un archivo html

<https://towardsdatascience.com/web-visualization-with-plotly-and-flask-3660abf9c946>

Graficar en plotly:

- Multiple axis: <https://plotly.com/python/multiple-axes/>

- grafico scatter: <https://plotly.com/python-api-reference/generated/plotly.graph_objects.Scatter.html>

- propiedad de las líneas: <https://plotly.com/python-api-reference/generated/plotly.graph_objects.scatter.html#plotly.graph_objects.scatter.Line>

- shapes en grafico plotly: <https://plotly.com/python/shapes/>

- como definir las shapes con respecto a un porcentaje del grafico: <https://stackoverflow.com/questions/60907004/plotly-how-to-add-a-horizontal-line-to-a-line-graph>

**EJEMPLO 5: Hola mundo, web que se actualize (refresh)**

En este ejemplo se profundiza el último punto importante para desarrollar una app web que la idea que esté constamente abierta y se actualize el front con datos nuevos.

La actualización del front debe de consistir en volver a ejecutar la API de flask (volver a ejecutar el script app.py para llevar los nuevos valores al front) y así tener la app web actualizada sin tener que volver a refrescar la página

La actualización del front se puede hacer de dos formas:

- Actualizar el front cada X periodo de tiempo. Por ejemplo quiero que se vuelva a ejecutar app.py cada 5 minutos

- Actualizar el front de acuerdo a un trigger. Por ejemplo, utilizar un tópico de pub/sub y cuando llega un mensaje nuevo que gatille la ejecución de app.py

DESCRIPCION:

-     Mostrar la hora actual y en un gráfico de barras de plotly de 3 barras donde se muestran en cada barra hora, minuto y segundo respectivamente (esto para poder validar

    que un gráfico de plotly se pueda refrescar y utilizar un gráfico de barras que muestra la hora actual es un buen ejemplo para representar un backend que se está actualizando

    constamente)

- CADA VEZ QUE SE CARGA LA PÁGINA SE VUELVE A EJECUTAR LA API (app.py) y por lo tanto se actualiza el front cambiando la hora actual.



AHORA, PARA QUE LA APP WEB SE ACTUALIZE CUANDO LE LLEGUE UN MENSAJE A UN TOPICO DE PUBSUB SE DEBE:

- Crear tópico pubsub

- Tener SA con permisos para escuchar pubsub. Y obtener JSON de la SA

EN TEORIA PARA PASARLE UN PUBSUB Y QUE GATILLE LA APP SE DEBE PASAR GET Y POST

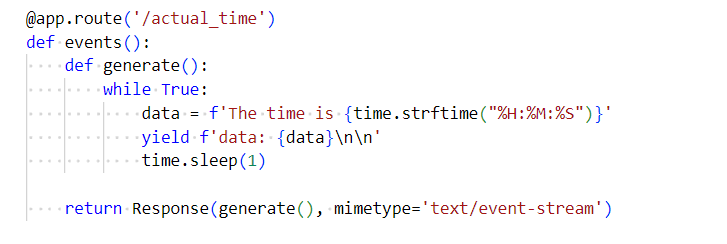
@app.route('/', methods=['GET', 'POST',])  # https://stackoverflow.com/questions/61058754/method-not-allowed-for-requested-url

Y luego en el pubsub que sea un trigger y poner la url del cloud run con la url completa a la página, por ejemplo si es al root, agregarle “/”

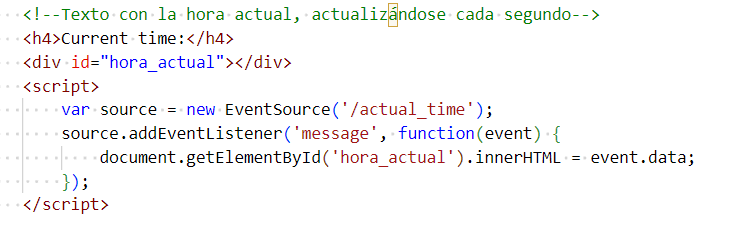
**DIFERENTES REFRESH DE UNA HTML + FLASK**

1. **Actualizar la hora cada segundo.** En este ejemplo se tiene código en flask que está constamente ejecutándose (por ejemplo cada 1 segundo) (que en este ejemplo obtiene la hora actual) y envía un EVENT a html para actualizar el valor de la hora cada vez que ocurre el evento

Flask

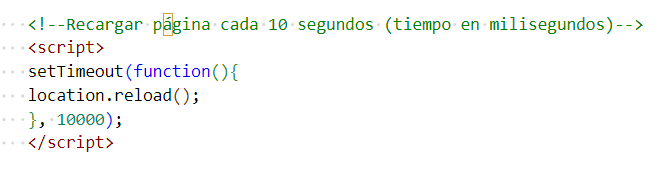


Html



1. **Actualizar TODA la app web cada X tiempo.** En este ejemplo solo basta correr un script de js en el código html que se encarga cada X tiempo de refrescar toda la página web

Html



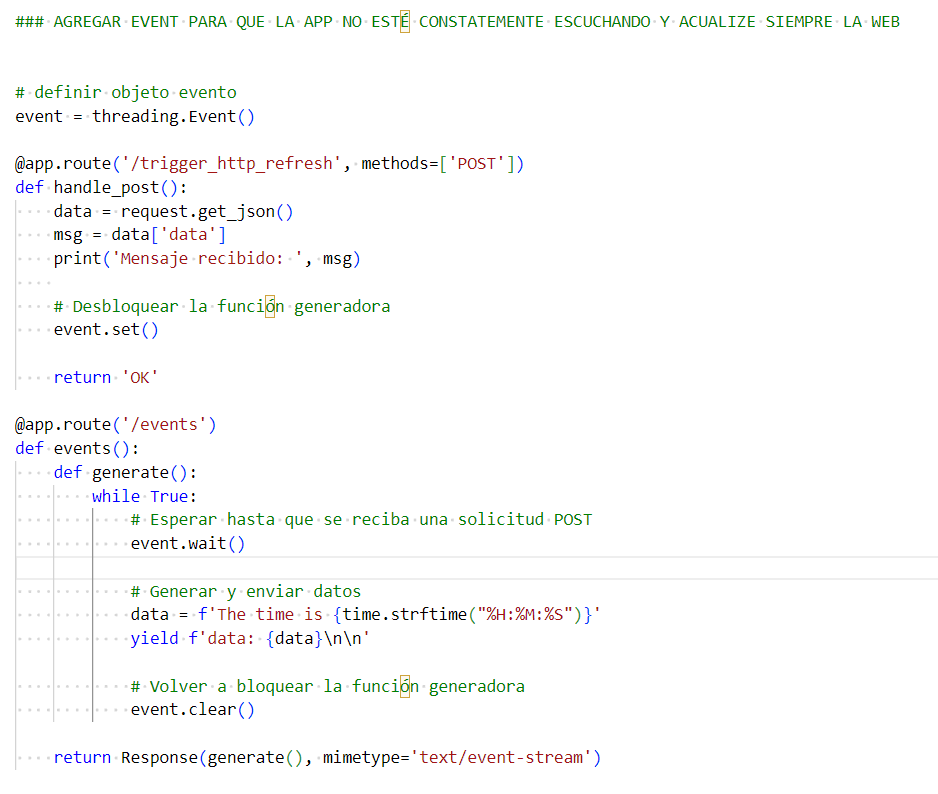
1. Actualizar toda la app web cuando llega un mensaje de pubsub. Esto es útil se trabaja desacoplado en varios recursos de GCP dejando un backend separado y una API flask. Cuando el backend termina su ejecución envía un mensaje de pubsub para actualizar la app web.

Para realizar esto no es forma tan sencilla y una solución es tener una cloud function que se activa por el mensaje de pubsub del backend y esta envía un http request post a una url de la API flask. Y este url envía el evento al front para actualizar toda la app.

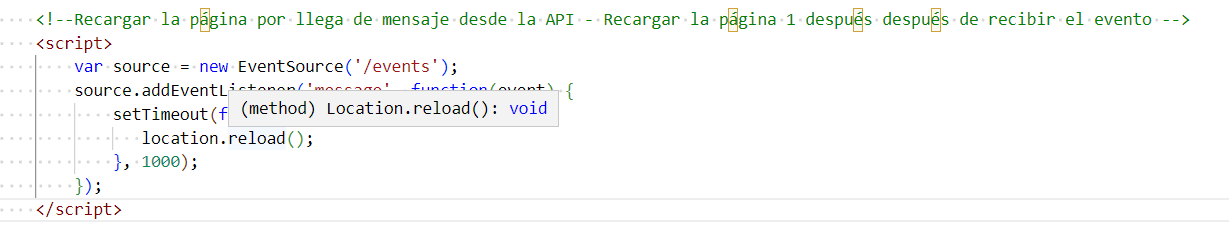
**LA VERSIÓN MÁS SIMPLIFICADA ES SIMPLEMENTE ENVÍAR EL HTTP POST (y evitar crear el tópico de pubsub y la cf que se gatilla por este tópico)**

**VERSIÓN SIMPLIFICADA HTTP POST QUE GATILLA EL EVENTO DE ACTUALIZAR LA APP WEB**

Flask



Html



PROBABLEMENTE FLASK SE PUEDE SIMPLIFICAR MÁS

Importante: las app web en cloud run parecieran que luego de algunas horas dejan de actualizarse. Probablemente sea porque la instancia se termina desactivando.

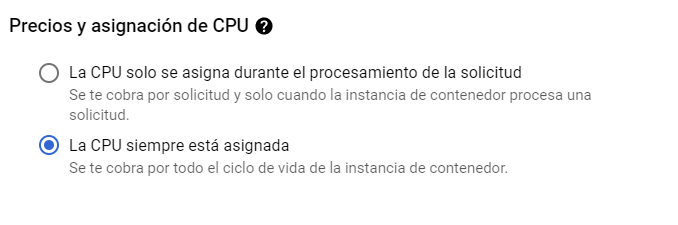
Esto se podría solcionar de 2 formas:

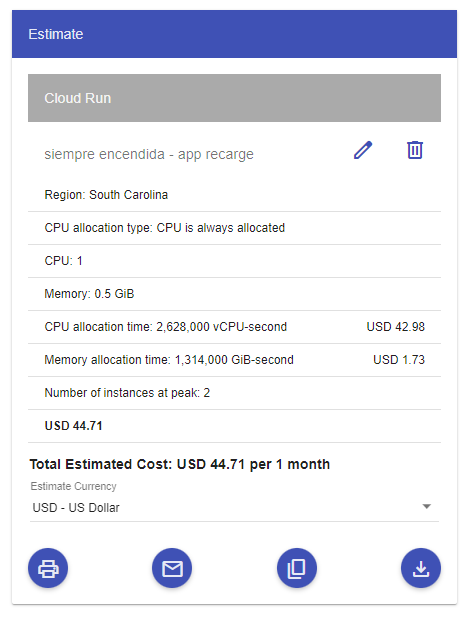
- Primera solución: agregar hora actual. Agregar una interacción constante entre api y front que me muestre la hora actual actualizándose cada segundo. Esto busca que cuando se abre la app web halla una interacción constante y que la instancia de cloud run no se desactive.

- Segunda solución: darle un tiempo de espera mayor. Lo máximo es 1 hora. Si por ejemplo hay un backend que se ejecuta cada 30 minutos, se puede utilizar un cloud run que espere 40 minutos. Luego al mandar un mensaje pasado 30 minutos, el front html se debería de actualizar

- Tecera solución. Migrar de cloud run a app engine. Por lo que entiendo app engine debería funcionar bien y no existiría dicho problema porque en app engine hay un cluster siempre corriendo aunque no se abra la app web (no como en cloud run que es serverless y solo se activa cuando alguien abre el link)

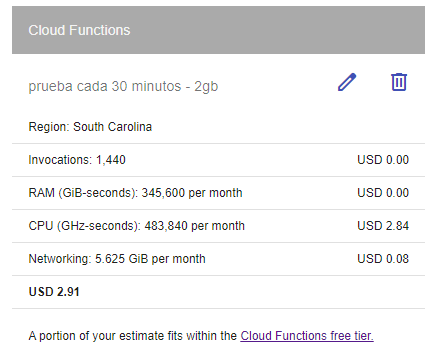
- Cuarta solución. CPU siempre encendida (EN VERDAD NO ES NECESARIO, BASTA CON TENER EL TEMA DE LA HORA ACTUAL SIEMPRE ENCENDIDO PARA QUE NO SE APAGUE LA INSTANCIA, PERO YA NO RECIBE SOLICITUDES DEL USUARIO)



La más barata son 42 dólares al mes, sin considerar un backend que deacuerdo a lo visto ejecutándose cada 30 minutos gastaría 15 dólares. 

<https://cloud.google.com/products/calculator#id=1fd0db44-1d03-48c2-9d4d-0df649edee8c>

Estimación de una cf de 2 gb



Por lo general serían 3 cfs de 2gb, más workflows, más reintentos cuando falla algo -> tiene sentido los 15 dolares que gastan los recomendadores

- Quinta solución: se puede utilizar la misma idea del código que actualiza la hora cada 1 segundo. Sería tomar ese código, que se ejecute cada X tiempo la tabla de ultima actualización y compare si el valor ha cambiado

PRÓXIMAS LECTURAS:

- Cómo hacer un mejor front utilizando react.

- Uitlizar react con flask: <https://towardsdatascience.com/build-deploy-a-react-flask-app-47a89a5d17d9>

**FUENTES – ideas para redactar un documento explicación - post**

FUENTES:

- Documentación cómo crear app en flask: <https://www.digitalocean.com/community/tutorials/how-to-create-your-first-web-application-using-flask-and-python-3>

EJEMPLOS PARA TENER DE GUÍA PARA ESCRIBIR EN MEDIUM SOBRE ESTE TEMA:

- Beneficios cloud run: <https://dev.to/pcraig3/google-cloud-run-the-best-host-platform-for-dynamic-apps-4ma6>

- Cómo alojar una app web en cloud run: <https://medium.com/google-cloud/hosting-a-website-on-google-cloud-using-cloud-run-a65343a98fce>

- Cómo hacer el deploy de una imagen docker en cloud run utilizando artifact registry: <https://medium.com/@ThatJenPerson/getting-started-with-artifact-registry-deploying-to-cloud-run-7aa9f2c65d07>