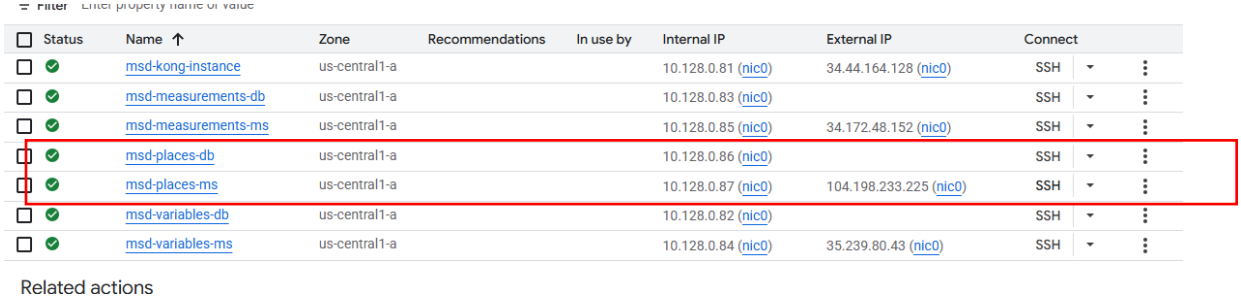
Entregable – Taller Microservicios V4

Brayan Estiven Salcedo Suarez

David Alejandro Calderon Pineda

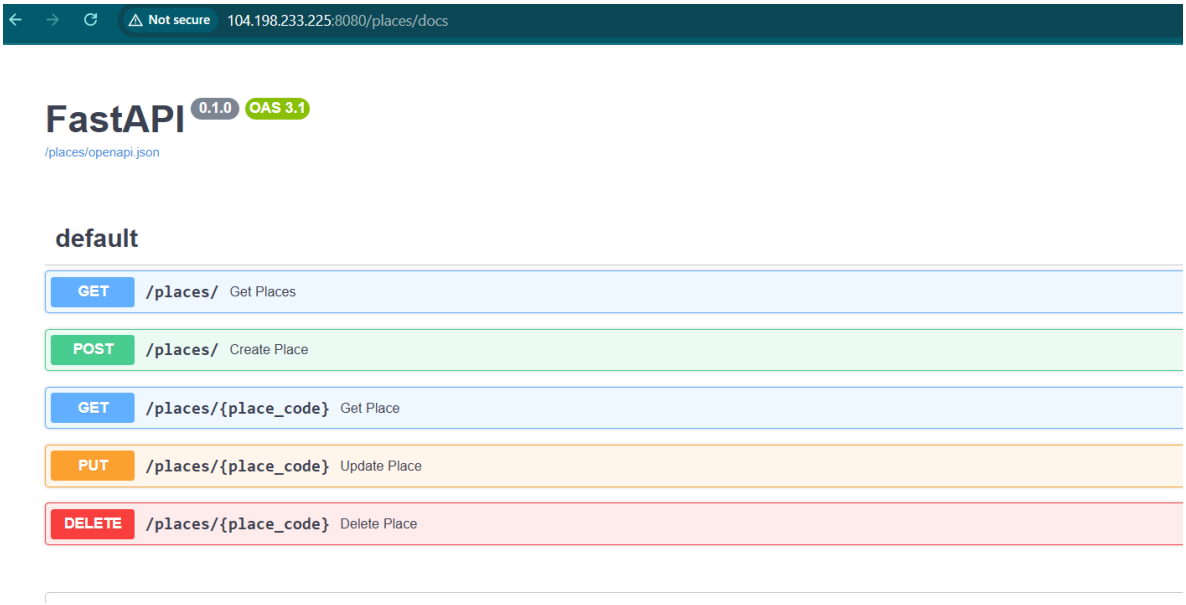
1. **Despliegue del microservicio Places.**
   En esta sección se documenta el proceso de implementación del nuevo microservicio Places, donde se configuró una instancia VM independiente con Ubuntu 18.04 y se desplegó el servicio usando FastAPI. La imagen adjunta muestra la estructura del microservicio conectado a su propia base de datos MongoDB, cumpliendo con el patrón "DB por servicio". Se evidencia la correcta configuración de los endpoints CRUD para gestionar lugares.

A continuación se ve la instancia desplegada de places



Ejecución del microservicio:
En esta sección podemos ver que la aplicación se ejecuto correctamente

```
dav_calderonpi@msd-places-ms:~$ cd /home/labs/ISIS2503-Microservices-AppDjango/places
dav_calderonpi@msd-places-ms:/home/labs/ISIS2503-Microservices-AppDjango/places$ sudo python3.12 main.py
/home/labs/ISIS2503-Microservices-AppDjango/places/main.py:14: DeprecationWarning:
        on_event is deprecated, use lifespan event handlers instead.

        Read more about it in the
        [FastAPI docs for Lifespan Events](https://fastapi.tiangolo.com/advanced/events/).

  @app.on_event("startup")
INFO:     Started server process [3343]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```



2. **Modificación de configuración de Kong**

   Aquí se realizó la actualización del API Gateway Kong para integrar el nuevo microservicio Places al ecosistema existente. La imagen muestra la configuración del archivo YAML donde se agregó el nuevo upstream (places-ms) y se definieron las rutas correspondientes, permitiendo que el gateway enrute correctamente las peticiones al nuevo servicio mientras mantiene el tráfico de los servicios existentes.

```
  - name: measurements_service
    host: measurements_upstream
    protocol: http
    routes:
      - name: measurements
        paths:
          - /measurements
        strip_path: false

  - name: places_service
    host: places_upstream
    protocol: http
    routes:
      - name: places
        paths:
          - /places
        strip_path: true


upstreams:
  - name: variables_upstream
    targets:
      - target: 10.128.0.84:8080
        weight: 100

  - name: measurements_upstream
    targets:
      - target: 10.128.0.85:8080
        weight: 100

  - name: places_upstream
    targets:
      - target: 10.128.0.87:8080
        weight: 100
```

3. Modificación de microservicio Measurements

Esta parte demuestra las adaptaciones necesarias en el servicio Measurements para validar la existencia de lugares antes de crear mediciones. Las imágenes muestran el nuevo código que implementa llamadas HTTP al servicio Places, verificando la integridad referencial distribuida.

```
PLACES_HOST = os.environ.get('PLACES_HOST', '10.128.0.87')
PLACES_PORT = os.environ.get('PLACES_PORT', '8080')
PATH_PLACES = f"http://{PLACES_HOST}:{PLACES_PORT}/places/"

"monitoring/settings.py" 144L, 4079C
```

Se añade la función que verifica el lugar:

```
def check_place(data):
    r = requests.get(settings.PATH_PLACES, headers={"Accept":"application/json"})
    places = r.json()
    for place in places:
        if data["place"] == place["code"]:
            return True
    return False
```

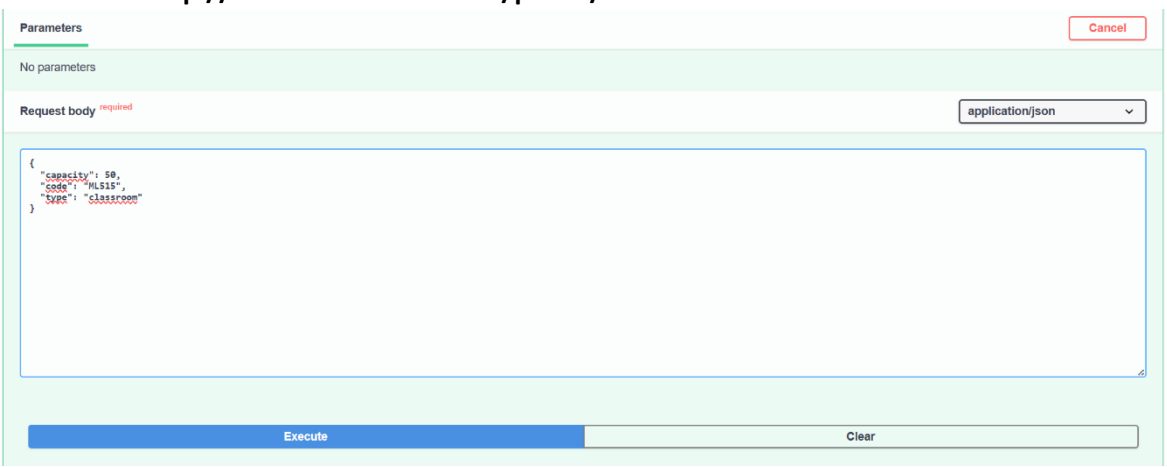
se modifican las funciones de measurement para usar la función

```
def MeasurementCreate(request):
    if request.method == 'POST':
        data = request.body.decode('utf-8')
        data_json = json.loads(data)
        if check_variable(data_json)and heck_place(data_json):
            measurement = Measurement()
            measurement.variable = data_json['variable']
            measurement.value = data_json['value']
            measurement.unit = data_json['unit']
            measurement.place = data_json['place']
            measurement.save()
            return HttpResponse("successfully created measurement")
        else:
            return HttpResponse("unsuccessfully created measurement. Variable or place does not exist")

def MeasurementsCreate(request):
    if request.method == 'POST':
        data = request.body.decode('utf-8')
        data_json = json.loads(data)
        measurement_list = []
        for measurement in data_json:
                    if check_variable(measurement) and check_place(measurement):
                        db_measurement = Measurement()
                        db_measurement.variable = measurement['variable']
                        db_measurement.value = measurement['value']
                        db_measurement.unit = measurement['unit']
                        db_measurement.place = measurement['place']
                        measurement_list.append(db_measurement)
                    else:
                        return HttpResponse("unsuccessfully created measurement. Variable or place does not e
")

        Measurement.objects.bulk_create(measurement_list)
```

4. **Creación de Lugares de Prueba**

   Mediante la interfaz Swagger UI autogenerada por FastAPI, se crearon lugares de prueba para validar el funcionamiento del microservicio. La captura muestra ejemplos de creación exitosa de lugares con diferentes capacidades y tipos (aula, laboratorio, etc.), demostrando la correcta persistencia en MongoDB y la disponibilidad de los datos.

   INGRESO A : http://104.198.233.225:8080/places/docs



Parameters

No parameters

Request body required

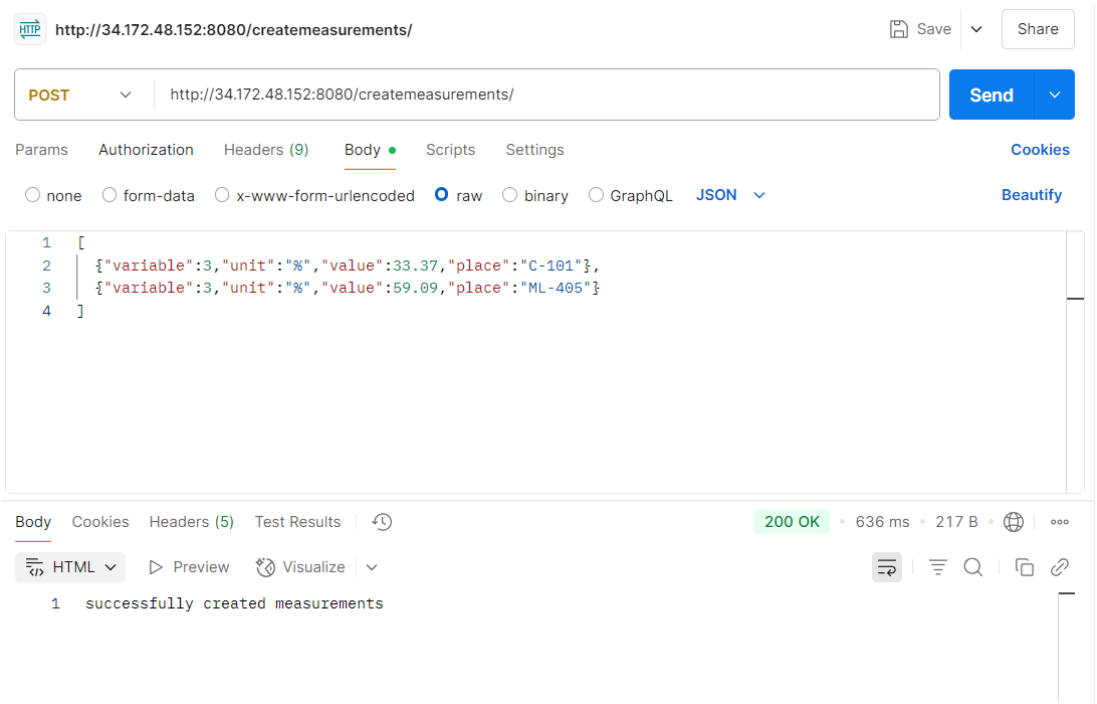
```
{
  "capacity": 80,
  "code": "ML213",
  "type": "classroom"
}
```

Execute

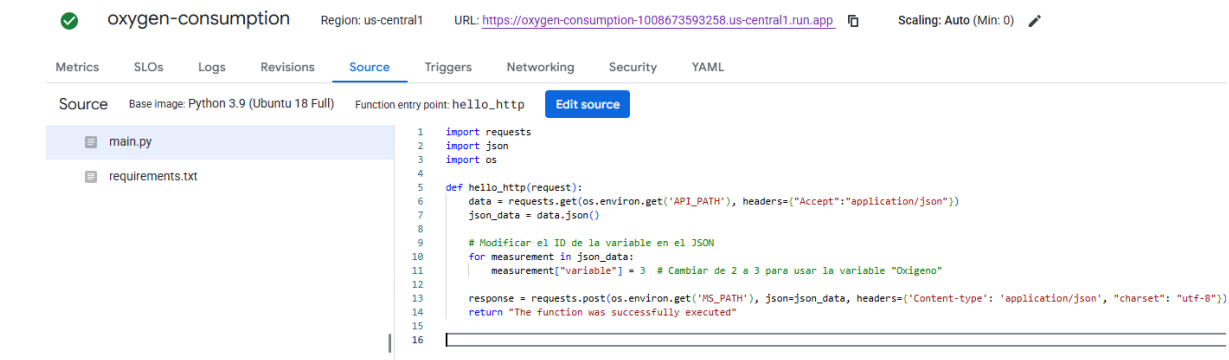
Responses

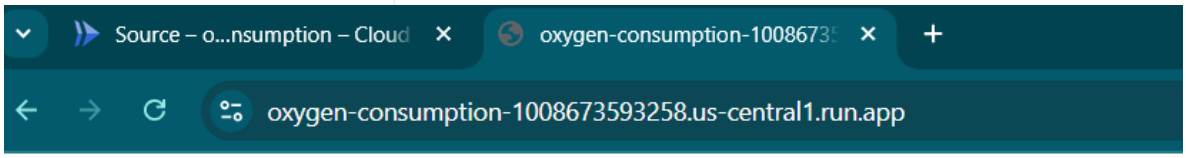
5. Creación de *Cloud Function* para datos de oxígeno

En esta sección se documenta el desarrollo de una nueva función serverless que consume datos de oxígeno de una API externa. Las imágenes muestran la configuración en GCP Cloud Functions, incluyendo variables de entorno y el código Python que realiza las peticiones HTTP tanto al API externo como al servicio Measurements para almacenar los datos.



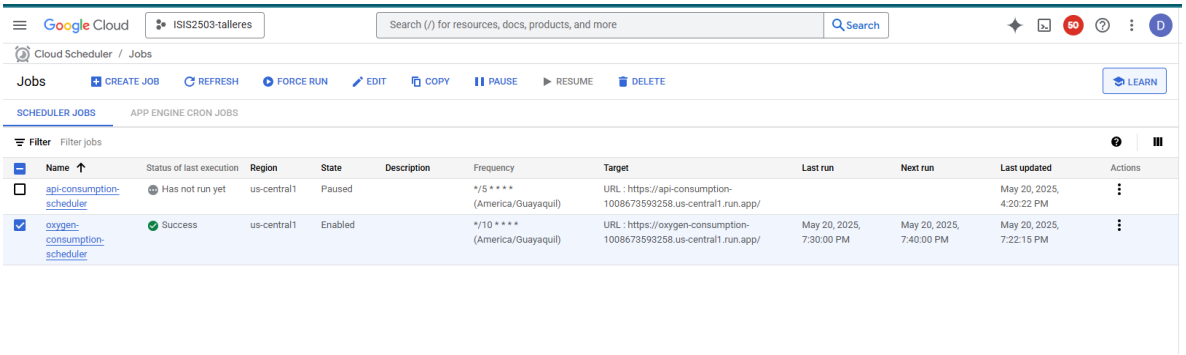
Creación oxyen-consumtion





The function was successfully executed

Posteriormente se realiza una verificación creación del *Cloud Scheduler* para ejecutar la función:



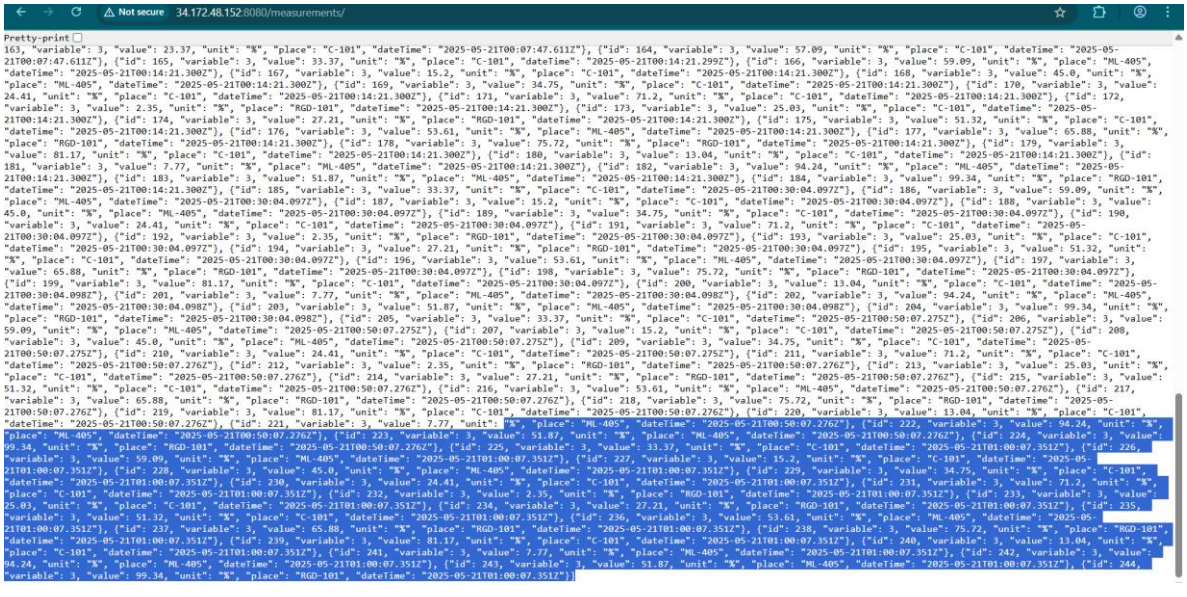
Microservicio Measurements:

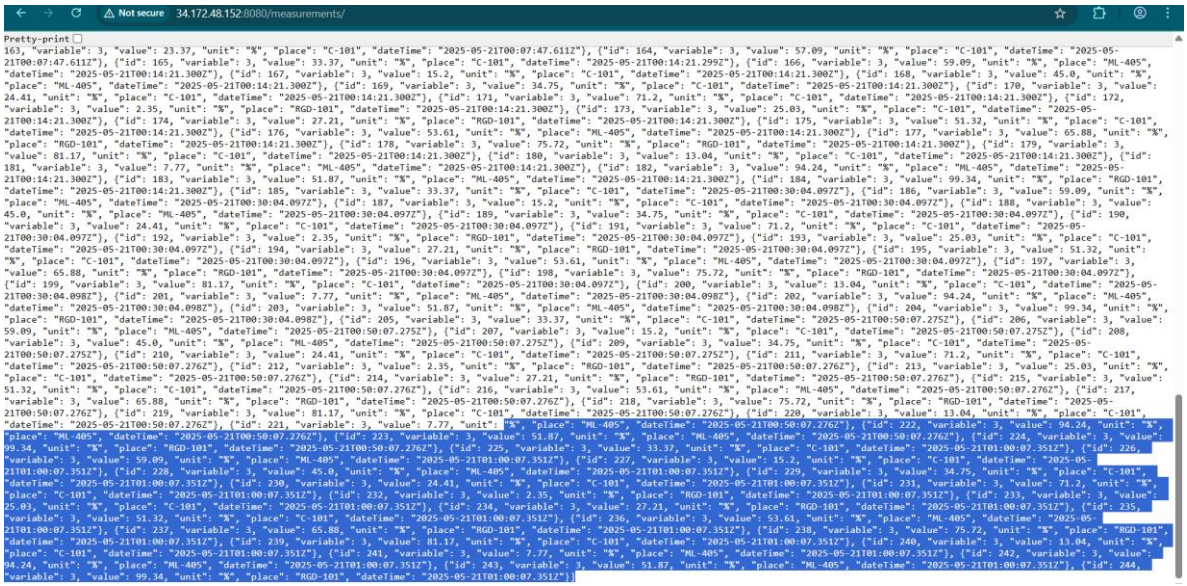
A continuación, se observa, en primer lugar, el resultado del microservicio Measurements accediendo al endpoint /measurements/ desde la dirección IP 104.198.233.225:8000. En esta

primera imagen se muestra una respuesta en formato JSON, en la que cada objeto representa una medición capturada por el sistema. Cada medición contiene atributos como id (identificador único), variable (ID del tipo de variable medida, como temperatura u oxígeno), value (valor numérico registrado), unit (unidad de medida como °C o ppm), place (lugar o sensor que reporta la medición), y dateTime (fecha y hora de captura).

Agregación de datos con el tiempo:

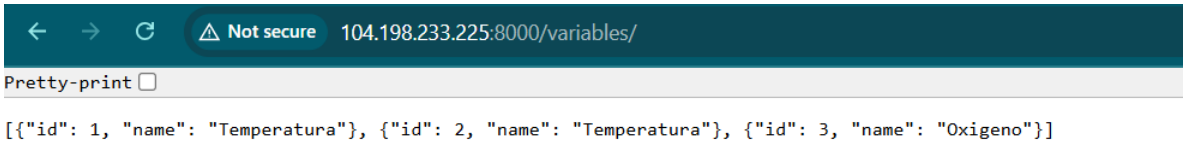
Posteriormente, se observa una segunda imagen correspondiente a la IP 34.172.48.152:8080, donde el mismo endpoint devuelve una respuesta mucho más extensa, lo cual sugiere que este entorno ha recibido un mayor volumen de mediciones. Esto puede ser producto del funcionamiento del Cloud Scheduler y la Cloud Function que inyecta datos automáticamente, demostrando así la integración entre componentes del sistema. Ambas imágenes confirman que el microservicio está correctamente desplegado y operativo, permitiendo la visualización y gestión de mediciones en diferentes entornos de la arquitectura.





Microservicio Variables:

En tercer lugar vemos el funcionamiento de los distintos tipos de variables cada una con su id y nombre



Microservicio Places:

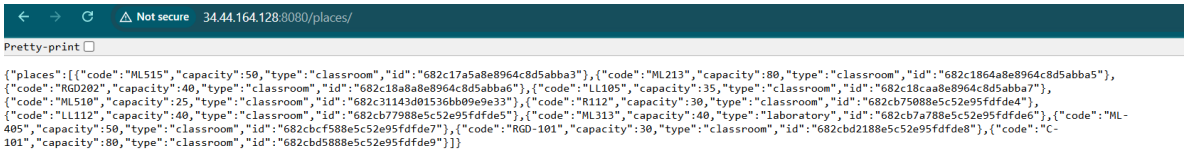
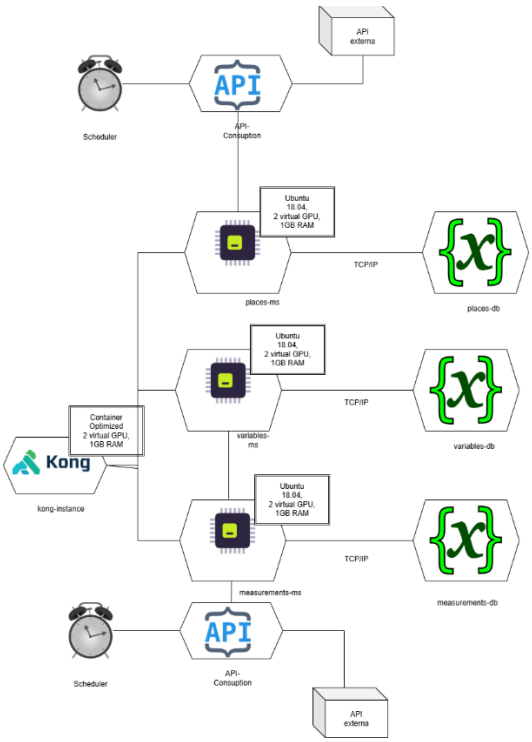


Diagrama de despliegue

El diagrama fue actualizado incorporando un nuevo microservicio llamado places-ms, desplegado sobre una máquina virtual con Ubuntu 18.04, 2 vGPU y 1 GB de RAM, y conectado a su propia base de datos places-db, alojada en una máquina optimizada para contenedores. Además, se añadió una nueva Cloud Function que se comunica directamente con places-ms y está representada con un ícono distintivo. Esta función es invocada de forma periódica por un nuevo Scheduler, el cual también fue agregado al diagrama. Todos los componentes originales,

como kong-instance, variables-ms, measurements-ms y sus respectivas bases de datos, se mantuvieron sin cambios, conservando la estructura inicial del sistema.



Análisis del resultado del experimento

A partir del experimento realizado, se puede concluir que la arquitectura implementada benefició directamente al Atributo de Calidad (ASR) planteado, el cual exige que, una vez desplegado el sistema, la integración de un nuevo servicio no tome más de dos horas. Este objetivo se cumplió gracias al enfoque de microservicios adoptado, el cual permite desarrollar y desplegar servicios de manera independiente, reduciendo significativamente los tiempos de integración. La incorporación del microservicio *Places* y su vinculación con servicios existentes como *Measurements* y la *Cloud Function* se realizó sin requerir modificaciones sustanciales en el sistema base, lo que evidencia el nivel de desacoplamiento alcanzado. Asimismo, el uso de tecnologías como Docker para contenedorización, Kong como API Gateway, y servicios gestionados como Cloud Run y Cloud Scheduler facilitaron una orquestación eficiente y ágil de los componentes. Esta infraestructura estandarizada no solo permitió una integración rápida, sino también garantizó la estabilidad del sistema en su conjunto. En resumen, la arquitectura no solo respondió al requerimiento funcional del ASR, sino que además demostró su capacidad para escalar y adaptarse a nuevas necesidades sin comprometer tiempos ni calidad del servicio.

Créditos:

A continuación, se observan los créditos disponibles en la cuenta al finalizar el laboratorio:

Créditos

Todos los créditos

Consulta y descarga los detalles de crédito aquí. Los descuentos por compromiso de uso activos no están incluidos aquí y se pueden ver en la página Compromisos.

Filtro Filtrar créditos

Nombre del crédito	Estado	Porcentaje restante	Valor restante	Valor original	Tipo	ID de crédito	Permiso
Free Trial	Vence en 8 días	92%	$1,167,087.43	$1,262,802.70	De un solo uso	FreeTrialUpgrad...	Uso específico; consulta la
Free Trial	Vencido	—	$1,262,741.14	$1,263,338.00	De un solo uso	FreeTrial:Credit-...	Uso específico; consulta la