

Tutorial 07 to do in class – Remember to upload the repo link to Teams.

Antes de iniciar:

- Terminar los tutoriales anteriores.
- Este taller muestra como desplegar el proyecto “randomquotes” como un microservicio (utilizando Docker Swarm, Docker service y GCP).
- REPOSITORIO: <https://github.com/Nram94/randomquotes-flask>

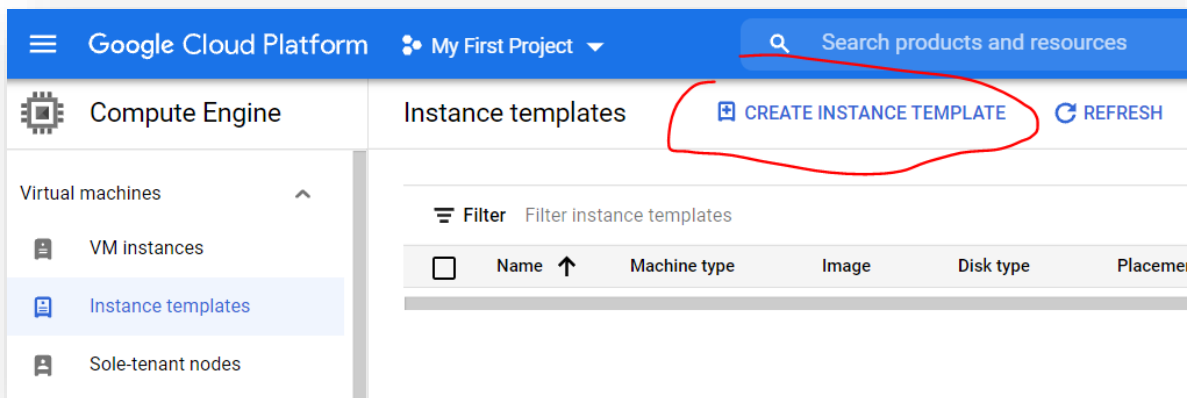
A. Creación de template y lanzamiento de nueva instancia

Sugerencia

Pare o borre todas las instancias que ha creado hasta el momento (menos la de su proyecto si está trabajando en el).

Creación de template

- Entramos en GCP y vamos a “Compute Engine” -> “Instance templates”. Allí seleccionamos “Create Instance Template”.



- Le colocamos un nombre y seleccionamos e2-micro.

← Create an instance template

i You have a draft that wasn't submitted, click Restore to keep working on it Restore

Describe a VM instance once and then use that template to create groups of identical instances [Learn more](#)


Name ?
Name is permanent


Machine configuration


Machine family
General-purpose Compute-optimized Memory-optimized GPU
Machine types for common workloads, optimized for cost and flexibility

Series
E2 ▼
CPU platform selection based on availability

Machine type
e2-micro (2 vCPU, 1 GB memory) ▼

 vCPU
1 shared core

 Memory
1 GB

 GPUs
-

[CPU platform and GPU](#)

These are estimated costs for a VM instance created using this template:
\$7.12 monthly estimate
That's about \$0.01 hourly
Pay for what you use: No upfront costs and per second billing
Show costs for location US ▼
[Details](#)

- En firewall seleccionamos “Allow HTTP traffic” y “Allow HTTPS traffic”.

Firewall ?

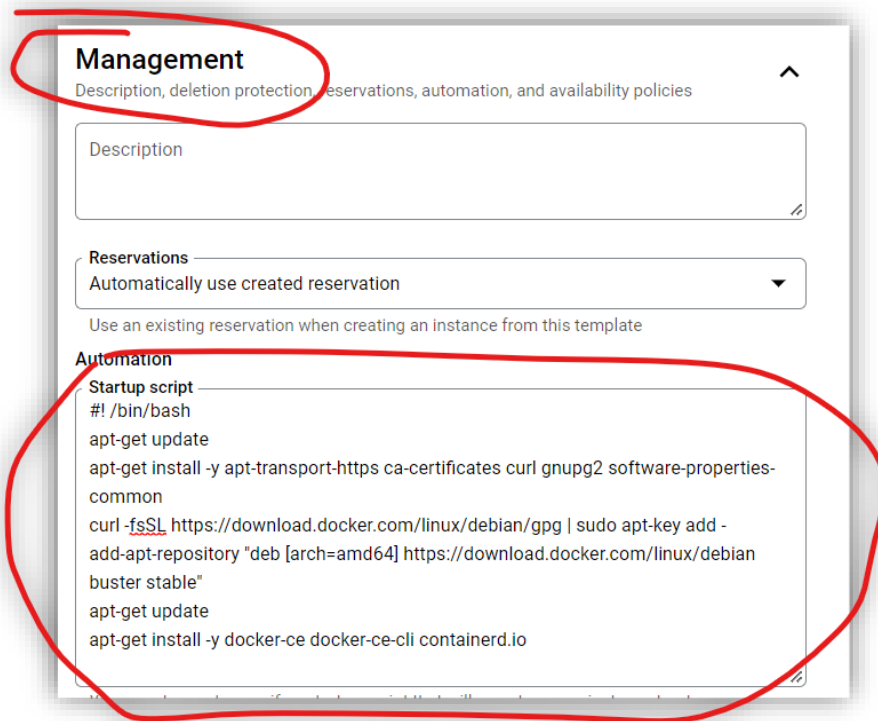
Add tags and firewall rules to allow specific network traffic from the Internet

☒ Allow HTTP traffic

☒ Allow HTTPS traffic

- Luego damos click en “Advance options” (debajo de Firewall). Y en la sección de “Management” -> “Automation” -> “Startup script” copiamos y pegamos el siguiente código (en la imagen siguiente verifique que le queda igual).

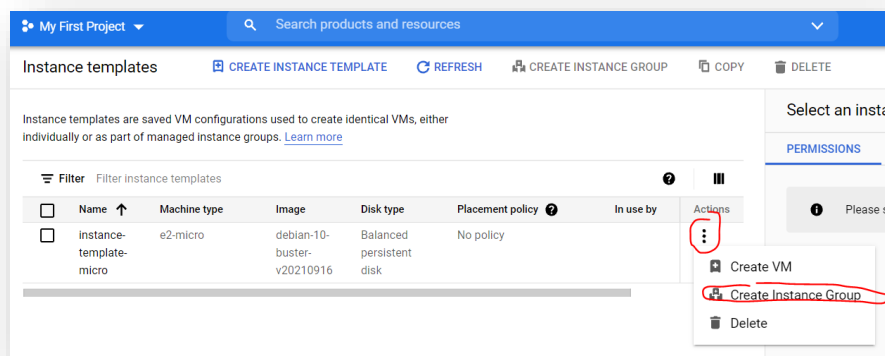
```
#!/bin/bash
apt-get update
apt-get install -y apt-transport-https ca-certificates curl gnupg2 software-properties-common
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian buster stable"
apt-get update
apt-get install -y docker-ce docker-ce-cli containerd.io
```



- Por último, de click en “Create”.

Lanzamiento de dos instancias

- Entramos en GCP y vamos a “Compute Engine” -> “Instance templates”. Seleccionamos la plantilla anterior y damos click en “Create Instance Group”.



- Le colocamos un nombre al grupo “instance-group-micro”. De “instance template”, seleccionamos el template creado anteriormente. Luego en “autoscaling mode” ponemos “Off: do not autoscale” y luego en “Number of instances” colocamos 2 (ver siguiente imagen).

Name *
instance-group-micro
Name is permanent

Description

Instance template *
instance-template-micro

Number of instances *
2

Location

For higher availability, select multiple zones in a region instead of a single zone. [Learn more](#)

☒ Single zone
☐ Multiple zones

Region *
us-central1 (Iowa)

Zone *
us-central1-a

Autoscaling

Use autoscaling to automatically add and remove instances to the group for periods of high and low load. [Learn more](#)

Autoscaling mode
Off: do not autoscale

- Finalmente damos click en “Create”.
- Ahora deberán aparecer 2 instancias (espere un par de minutos). La idea es desplegar el microservicio (proyecto “randomquotes”) en múltiples instancias. Para este taller solo utilizaremos 2, pero ese proyecto se podría replicar y desplegar en 20, 40, 60 instancias.

VM instances [CREATE INSTANCE](#) [IMPORT VM](#) [REFRESH](#) [START / RESUME](#) [STOP](#) [SUSPEND](#) [OPERATIONS](#)

[INSTANCES](#) [INSTANCE SCHEDULE](#)

VM instances are highly configurable virtual machines for running workloads on Google infrastructure. [Learn more](#)

Filter Enter property name or value

<input type="checkbox"/>	Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP
<input type="checkbox"/>	✓	instance-group-micro-2nfs	us-central1-a		instance-group-micro	10.128.0.15 (nic0)	34.122.183.194 ↗
<input type="checkbox"/>	✓	instance-group-micro-hjw2	us-central1-a		instance-group-micro	10.128.0.16 (nic0)	35.232.252.59 ↗

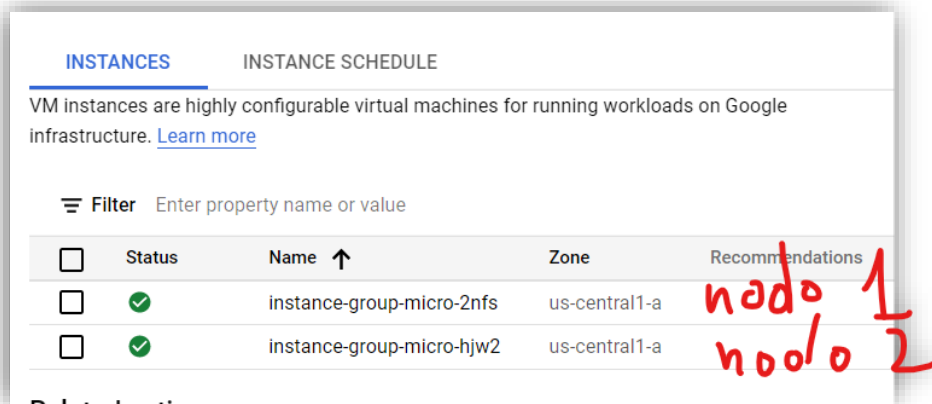
- **Nota:** verifique que las instancias quedaron bien creadas. Conéctese por SSH y ejecute el comando `docker -v` (debe esperar un par de minutos mientras se instala Docker). Debería ver la versión de Docker en ambas instancias.

```
nram_vel94@instance-group-micro-dk2z:~$ docker -v
Docker version 24.0.6, build ed223bc
nram_vel94@instance-group-micro-dk2z:~$
```

Inicialización Docker Swarm

Identificación de instancias

- De ahora en adelante hablaremos de las 2 instancias creadas de la siguiente manera.
 - La primera instancia (tipo instance-group) que le aparezca listada en VM instances, la llamaremos **nodo-01**.
 - La segunda instancia (tipo instance-group) que le aparezca listada en VM instances, la llamaremos **nodo-02**.



Inicializando un cluster de Docker Swarm desde nodo-01

- Conéctese por SSH al nodo-01.
- Ejecute ahora este comando para iniciar Docker Swarm

sudo docker swarm init

```
dcorreab@instance-group-micro-2nfs:~$ sudo docker swarm init
Swarm initialized: current node (ctmuucpj786yl8zh6d6c3lbeq) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-0gr4jvhvelliinvek6khf870jaqtbv3y62pl185cw0iu4i70bl-3e2slaabr9qblux4iiagudulo 10.128.0.15:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

- Ahora ejecutaremos un comando, que permitirá que otros nodos se conecten con el cluster de Docker Swarm que acabamos de iniciar. En este caso como “manager”. Para eso ejecutaremos el siguiente comando (el cual utilizaremos LUEGO en el nodo-02):

sudo docker swarm join-token manager

- Copie y guarde el comando que aparece en rojo.

```
dcorreab@instance-group-micro-2nfs:~$ sudo docker swarm join-token manager
To add a manager to this swarm, run the following command:

docker swarm join --token SWMTKN-1-0gr4jvhvelliinvek6khf870jaqtbv3y62pl185cw0iu4i70bl-acyclucfxhwn0wv41bucekzfn 10.128.0.15:2377
```

- Ejecute el siguiente comando para ver los nodos activos en el cluster (debería aparecer solo 1).

sudo docker node ls

```
dcorreab@instance-group-micro-2nfs:~$ sudo docker node ls
ID                                     HOSTNAME                                STATUS    AVAILABILITY    MANAGER S
TATUS  ENGINE VERSION
ctmuucpj786yl8zh6d6c3lbeq *         instance-group-micro-2nfs             Ready    Active           Leader
20.10.14
```

B. Conectando nodos con un cluster Docker Swarm

Conexión del nodo-02

- Conéctese por SSH al nodo-02.
- Ejecute el comando copiado anteriormente (agréguete *sudo* al inicio), el que incluía el token para unirse como manager. **Verifique que el comando quedo todo en una sola línea, o si no saldrá error (a veces el comando se parte en 2 partes al copiar y pegar).**

```
dcorreab@instance-group-micro-hjw2:~$ sudo docker swarm join --token SWMTKN-1-0gr4jvhvelliinvek6khf870jaqtbv3y62pl185cw0iu4i70bl-acyclucfxhwn0wv41bucekzfn 10.128.0.15:2377
This node joined a swarm as a manager.
```

Verificación desde el nodo-01 (también se puede hacer desde el nodo-02)

- Conéctese por SSH al nodo-01.
- Liste los nodos para verificar si el nodo-02 quedó correctamente enlazado (le deberán aparecer 2 nodos).

sudo docker node ls

```
dcorreab@instance-group-micro-2nfs:~$ sudo docker node ls
ID                                     HOSTNAME                                STATUS    AVAILABILITY    MANAGER S
TATUS  ENGINE VERSION
ctmuucpj786yl8zh6d6c3lbeq *         instance-group-micro-2nfs             Ready    Active           Leader
20.10.14
cvoib6kq0vvywaz0bqabbbhu6          instance-group-micro-hjw2             Ready    Active           Reachable
20.10.14
```

- **Nota:** ese mismo comando lo podría ejecutar desde el nodo-02 dado que es un manager.

C. Modificando el proyecto randomquotes

Controller

- Realice el siguiente cambio en el archivo *index.js*. Luego suba los cambios a GitHub (si no lo ha hecho, cree el repo y suba los cambios).

Modify Bold Code

```
from flask import Flask, jsonify
import os
import random

app = Flask(__name__)

phrases = [
    "Get ready to be inspired...",
    "See rejection as redirection.",
    "There is beauty in simplicity.",
    "You can't be late until you show up.",
    "Maybe life is testing you. Don't give up.",
    "Impossible is just an opinion.",
    "Alone or not you gonna walk forward.",
]

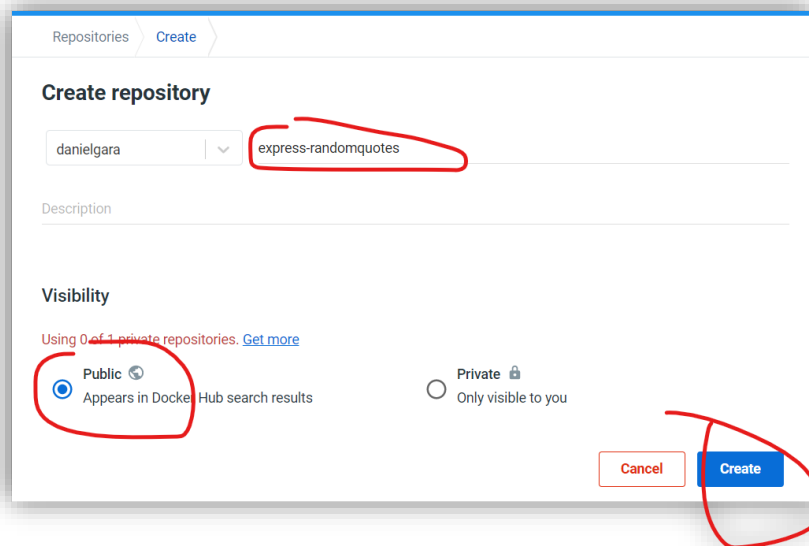
@app.route('/')
def get_random_quote():
    phrase = random.choice(phrases)
    container_id = os.uname()[1]
    return f"{phrase} - Container Id: {container_id}"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)
...
```

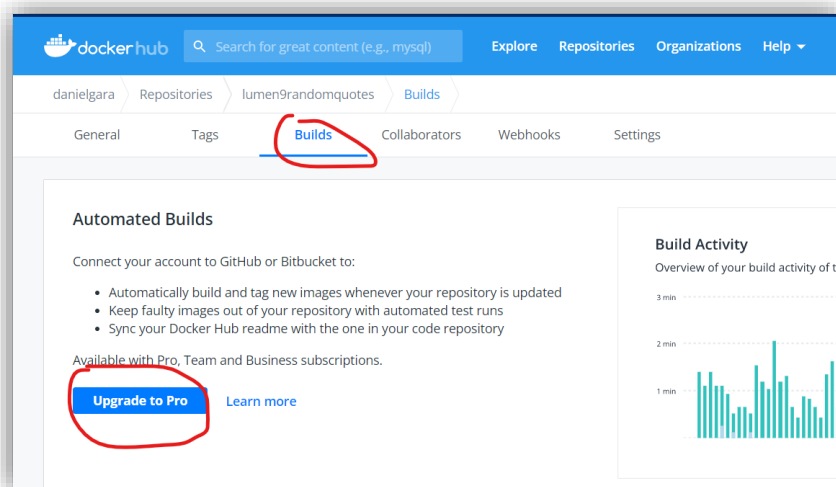

D.Desplegando el proyecto randomquotes en DockerHub

DockerHub

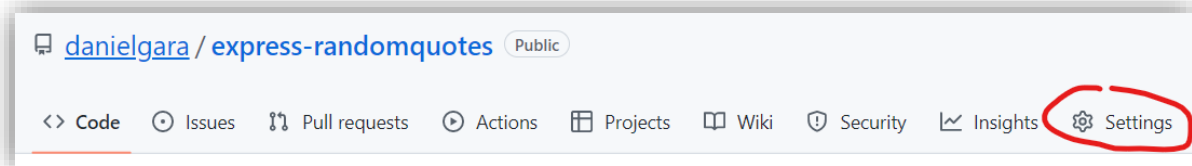
- Visite dockerhub.com, cree una cuenta, y luego cree un repositorio.



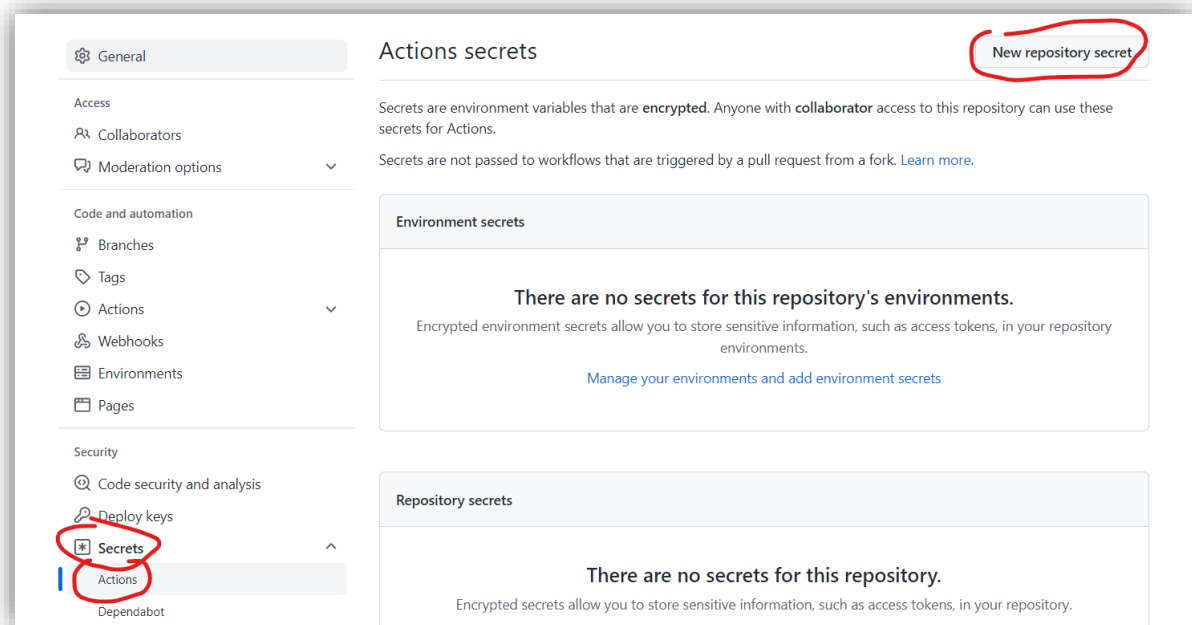
- Anteriormente una vez creado el Repositorio en DockerHub, podíamos conectarlo directamente a nuestro proyecto en GitHub; y crear automáticamente las imágenes de nuestro proyecto. Pero ya no, ya es solo para planes pro (carita triste).



- Veamos cómo solucionar el problema. Vaya a su repositorio en GitHub, y de click en la pestaña "Settings".



- Vaya luego a “Secrets” -> “Actions”, y luego a “New repository secret”.



- Agregue en Name `DOCKERHUB_USER` y en Value coloque su usuario de DockerHub.

A screenshot of the 'Actions secrets / New secret' form. The 'Name' field is filled with 'DOCKERHUB_USER' and the 'Value' field is filled with 'danielgara'. The form is titled 'Actions secrets / New secret'.

- Repita el proceso, pero ahora agregue en Name `DOCKERHUB_PASS` y en Value coloque su contraseña de DockerHub.

Actions secrets / New secret

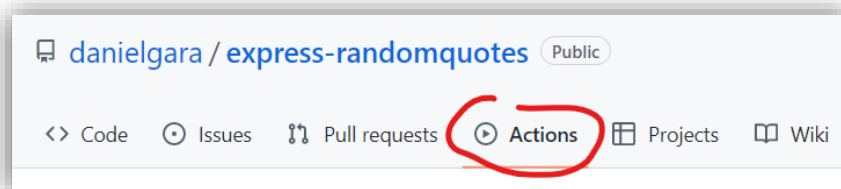
Name

DOCKERHUB_PASS

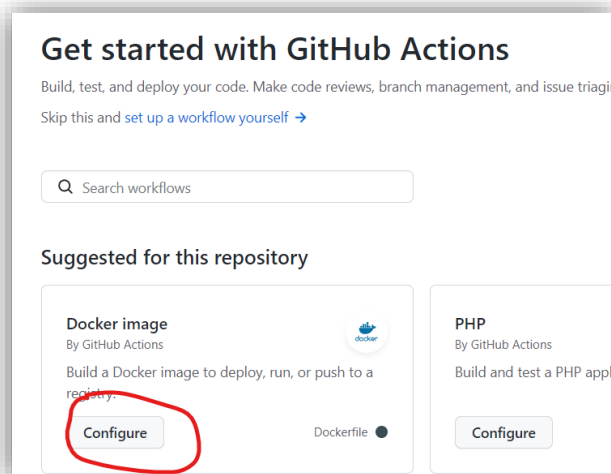
Value

CASICAIGOPERONO

- Ahora, vaya entonces a su repositorio en GitHub, y de click en la pestaña “Actions”.



- Dado que nuestro proyecto tiene un archivo *Dockerfile* en la raíz, entonces GitHub nos va a sugerir crear un Docker image. De click a “Configure”:



- Borre todo lo que le sale por defecto, y reemplázelo por lo siguiente. Ojo, lo que hay en rojo debe reemplazarlo por: el nombre de su usuario en DockerHub / nombre de su repositorio en DockerHub.

name: Docker Image CI

on:

```

push:
  branches: [ master ]
pull_request:
  branches: [ master ]

jobs:

  build:

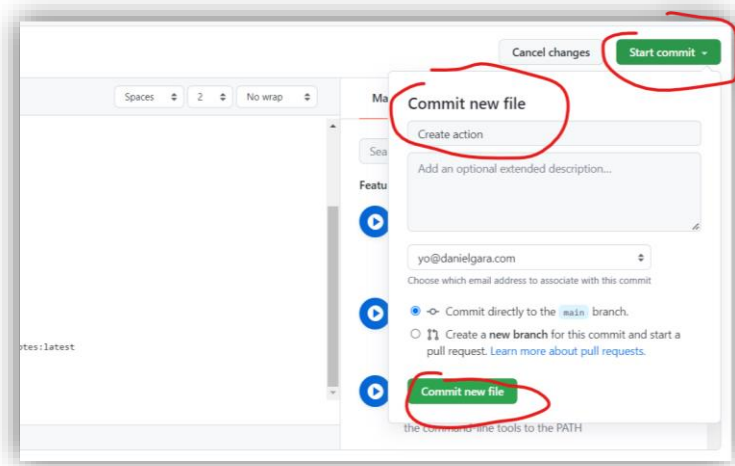
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4
      - name: docker login
        env:
          DOCKER_USER: ${secrets.DOCKERHUB_USER}}
          DOCKER_PASS: ${secrets.DOCKERHUB_PASS}}
        run: |
          docker login -u $DOCKER_USER -p $DOCKER_PASS
      - name: Build the Docker image
        run: docker build . --file Dockerfile --tag danielgara/express-randomquotes:latest

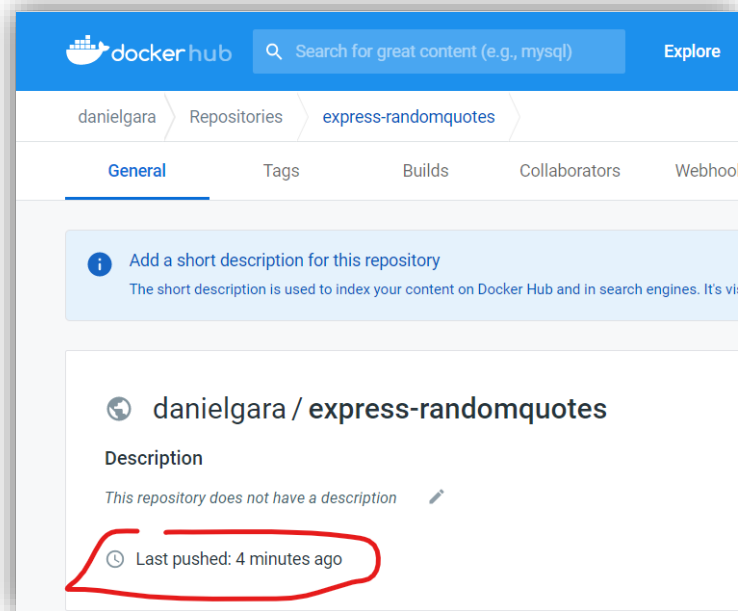
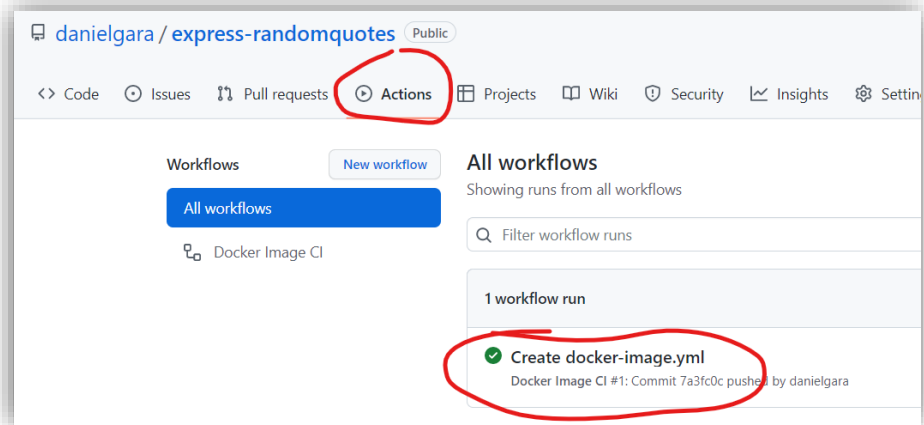
      - name: Docker Push
        run: docker push danielgara/express-randomquotes

```

- Luego de click en “Start commit”, coloque un texto al commit, y luego en “Commit new file”.



- Regrese a GitHub Actions, en un par de minutos le debe salir en verde el workflow ejecutado, lo cual quiere decir que ya debería tener la imagen de su proyecto montada en DockerHub.



- Ya no necesitamos la cuenta pro (carita feliz). Y no solo eso, cada vez que usted haga un push a Github, se va a ejecutar ese GitHub actions, y actualizar su imagen en Docker Hub automáticamente.

E. Desplegando el proyecto con Docker service

Conexión del nodo-01

- Conéctese por SSH al nodo-01.

Despliegue del servicio

- Ejecute el siguiente comando para verificar cuantos servicios están corriendo en este momento:

sudo docker service ls

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Apr 21 16:15:32 2022 from 35.235.245.129
dcorreab@instance-group-micro-2nfs:~$ sudo docker service ls
ID                NAME                MODE                REPLICAS            IMAGE                PORTS
dcorreab@instance-group-micro-2nfs:~$
```

- Cree un servicio llamado *randomquotes* basado en la imagen de su proyecto en DockerHub, y defínale 4 réplicas. Para eso utilice el siguiente comando (reemplace la última palabra en rojo por su repositorio DockerHub -> por ejemplo yo use *danielgara/express-randomquotes*):

sudo docker service create --name randomquotes --replicas 4 -p 80:80 REPO_DOCKER_HUB

```
Randomquotes
ag dcorreab@instance-group-micro-0mlt:~$ sudo docker service create --name randomquotes --r
eplicas 4 -p 80:80 danielgara/express-randomquotes
k2yt93t2jsios159jkebasx3n
overall progress: 4 out of 4 tasks
1/4: running
2/4: running
3/4: running
4/4: running
it's verify: Service converged
dcorreab@instance-group-micro-0mlt:~$
```

- Ahora veamos en que nodos quedaron desplegados las 4 réplicas, con el siguiente comando:

sudo docker service ps randomquotes

```
dcorreab@instance-group-micro-0mlt:~$ sudo docker service ps randomquotes
ID                NAME                IMAGE                ERROR                PORTS                NODE
8az9fp8z4q59     randomquotes.1     danielgara/express-randomquotes:latest    Running 22 seconds ago    Running 22 seconds ago    instance-group-micro-jzld
vnfv4p2754xq     randomquotes.2     danielgara/express-randomquotes:latest    Running 22 seconds ago    Running 22 seconds ago    instance-group-micro-0mlt
83fqilr0zlaw     randomquotes.3     danielgara/express-randomquotes:latest    Running 22 seconds ago    Running 22 seconds ago    instance-group-micro-jzld
m7eqzykbvctj     randomquotes.4     danielgara/express-randomquotes:latest    Running 22 seconds ago    Running 22 seconds ago    instance-group-micro-0mlt
dcorreab@instance-group-micro-0mlt:~$
```

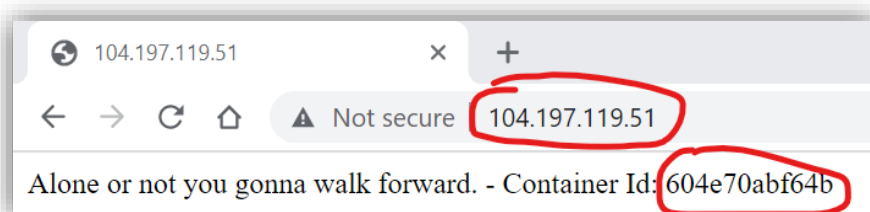
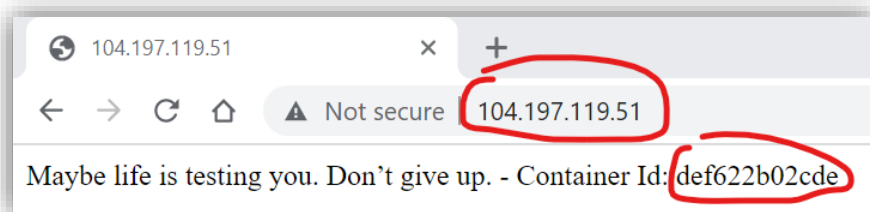
- Como vemos 2 réplicas están corriendo en el nodo-01 y las otras 2 en el nodo-02.
- Para verificar las réplicas, podemos mostrar la lista de contenedores desde el nodo-01 y nodo-02.

sudo docker container ls

```
dcorreab@instance-group-micro-0mlt:~$ sudo docker container ls
CONTAINER ID   IMAGE                                COMMAND                                CREATED
STATUS        PORTS          NAMES
f9dbd57e8aa7   danielgara/express-randomquotes:latest "docker-entrypoint.s..." 59 seconds ago Up 56 seconds 8080/tcp randomquotes.4.m7eqzykbvctjymqc5lpat5avs
def622b02cde   danielgara/express-randomquotes:latest "docker-entrypoint.s..." 59 seconds ago Up 56 seconds 8080/tcp randomquotes.2.vnfv4p9754xq03vcxewlk5kwt
```

```
dcorreab@instance-group-micro-jzld:~$ sudo docker container ls
CONTAINER ID   IMAGE                                COMMAND                                CREATED
STATUS        PORTS          NAMES
f047023dd757   danielgara/express-randomquotes:latest "docker-entrypoint.s..." 2 minutes ago Up About a minute 8080/tcp randomquotes.1.8az9fp8z4g59ojdvdsydsaib2
604e70abf64b   danielgara/express-randomquotes:latest "docker-entrypoint.s..." 2 minutes ago Up About a minute 8080/tcp randomquotes.3.83fqilr0zlawl7cpv622f5una
```

- Si accedemos a desde la IP pública del nodo-01 o desde la IP pública del nodo-02, podremos ver la aplicación corriendo. Recargue la página con un espacio de 3 segundos (o abra en incognito), y podrá ver que el **container id** cambia en múltiples ocasiones. Esto quiere decir que, desde la misma IP, estamos cargando la aplicación desde diferentes contenedores (que podrían o no estar alojados en el mismo servidor). Vea el siguiente ejemplo, desde la IP del nodo-01, está cargando un contenedor alojado en el nodo-01, pero al recargar, carga otro alojado en el nodo-02.



Clusters

- El sistema es tan “inteligente”, que independiente desde la IP o DNS que se acceda, cargará un contenedor disponible del cluster (no importa si pertenece a esa misma IP o DNS desde que se accedió).

Actividad

- Conéctese a el nodo-01, liste los contenedores, elimine un contenedor.
- Espere unos segundos y vuelva a listar los contenedores del nodo-01.
- ¿Entendió que acabó de pasar?

Reto interesante

Hay un reto muy muy muy interesante para aquellos que deseen aplicar lo aprendido en este taller. Es un reto complejo, pero bastante alcanzable. En este taller aprendimos de integración continua, fuimos capaces de generar un proceso que va desde que se hace un commit/push al repo, hasta actualizar el contenedor automáticamente en DockerHub.

Por lo tanto, eso que aprendimos lo podríamos aplicar a nuestro entregable 2. De tal manera, que cada vez que se haga un push, se actualice la imagen en DockerHub. Ojo, en este caso, toca modificar un poco el docker-image.yml, para crear el .env con las respectivas credenciales de base de datos Cloud SQL, y que se cree la imagen valida en DockerHub.

Luego desde la instancia de GCP, creamos el contenedor (run) basados en la imagen de DockerHub (NO desde la local como lo veníamos haciendo), y finalmente, corremos el contenedor de watch-tower (ver final de presentación 15), y ta-ram, tenemos CI/CD listo. Cada vez, que hagamos un push a github, se va a actualizar automáticamente nuestro proyecto en GCP.