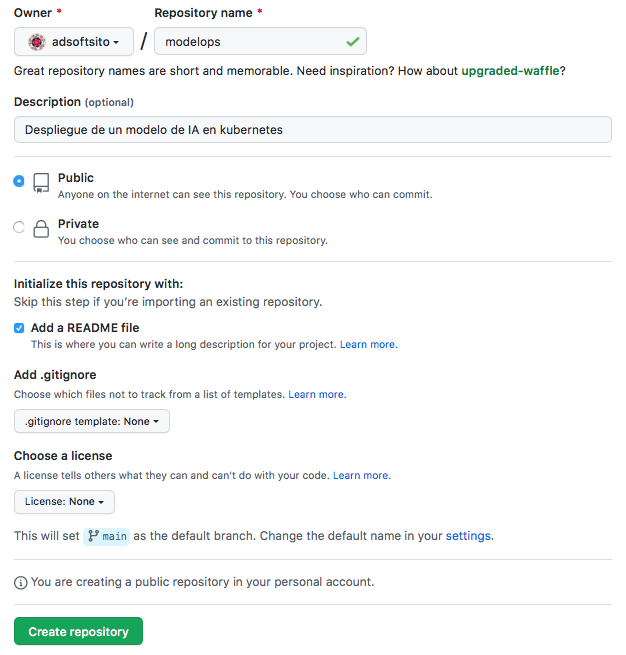
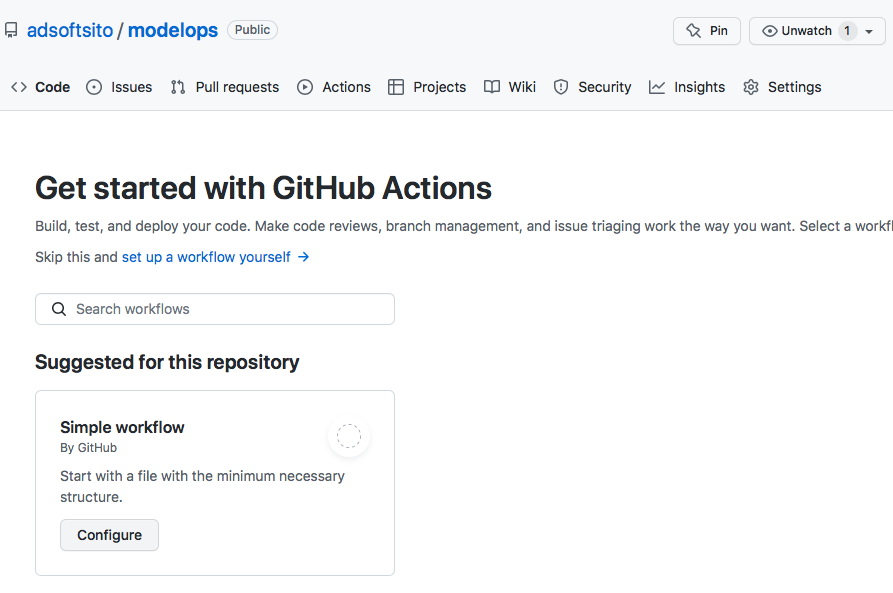
https://hub.docker.com/

Deploy de un Modelo de IA en Cloud Kubernetes

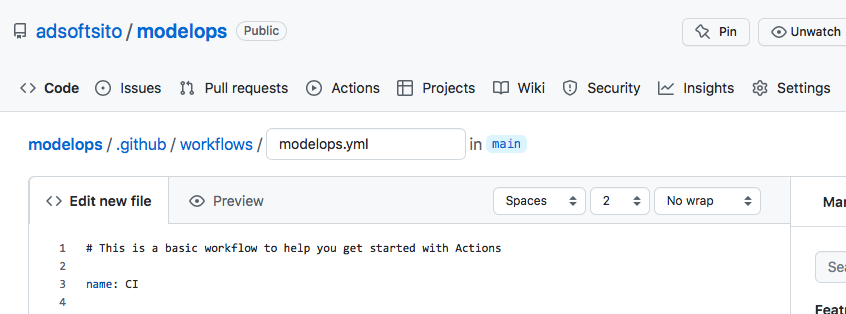
1.- Crear un repositorio en <https://github.com> con las siguientes características



2.- Crear un **Simple workflow** en Github Actions, para automatizar las tareas de despliegue. Click en **Configure**



3.- Renombrar workflow con nombre: **modelops.yml**



4.- Reemplazar el codigo del template, respetando la identacion.

***name: Modelops deploy***

***on:***

***push:***

***branches: [ main ]***

***pull\_request:***

***branches: [ main ]***

***jobs:***

***build:***

***runs-on: ubuntu-latest***

***container: adsoft/aiops:latest***

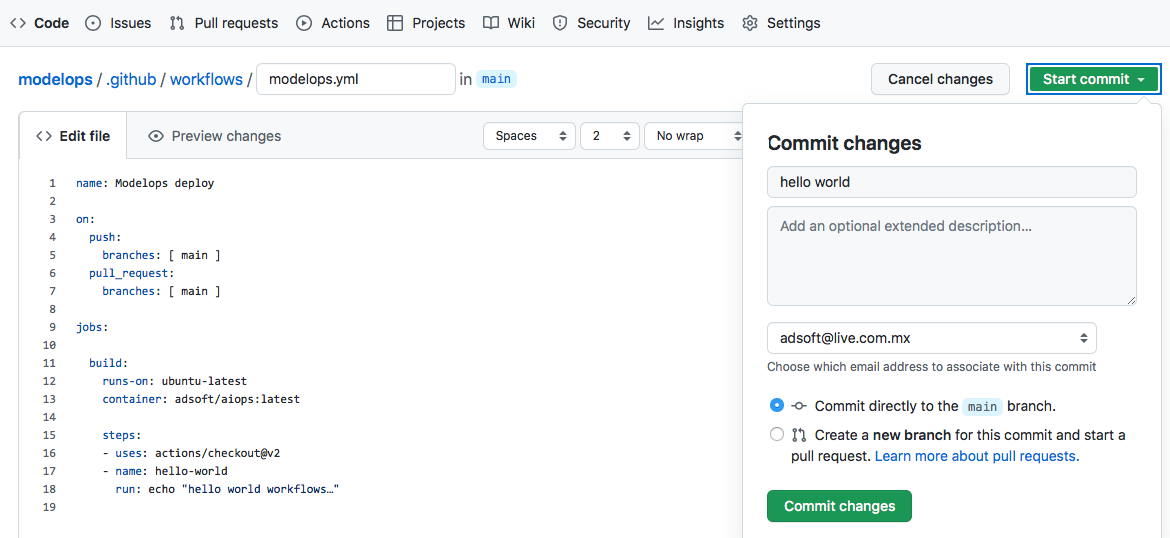
***steps:***

***- uses: actions/checkout@v2***

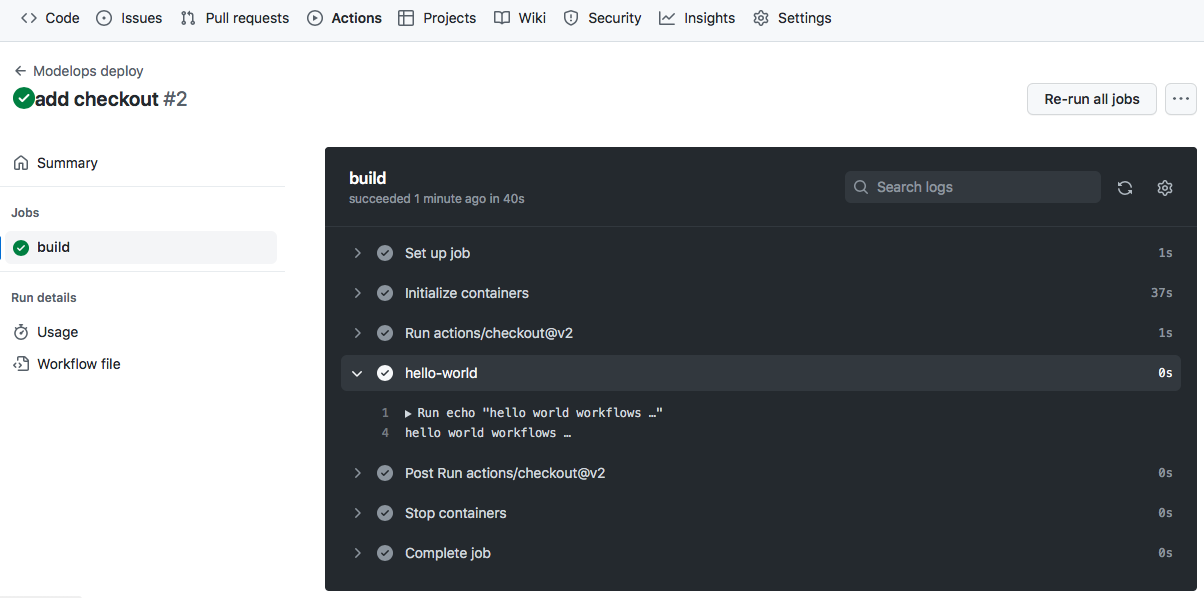
***- name: hello-world***

***run: echo “hello world workflows …”***

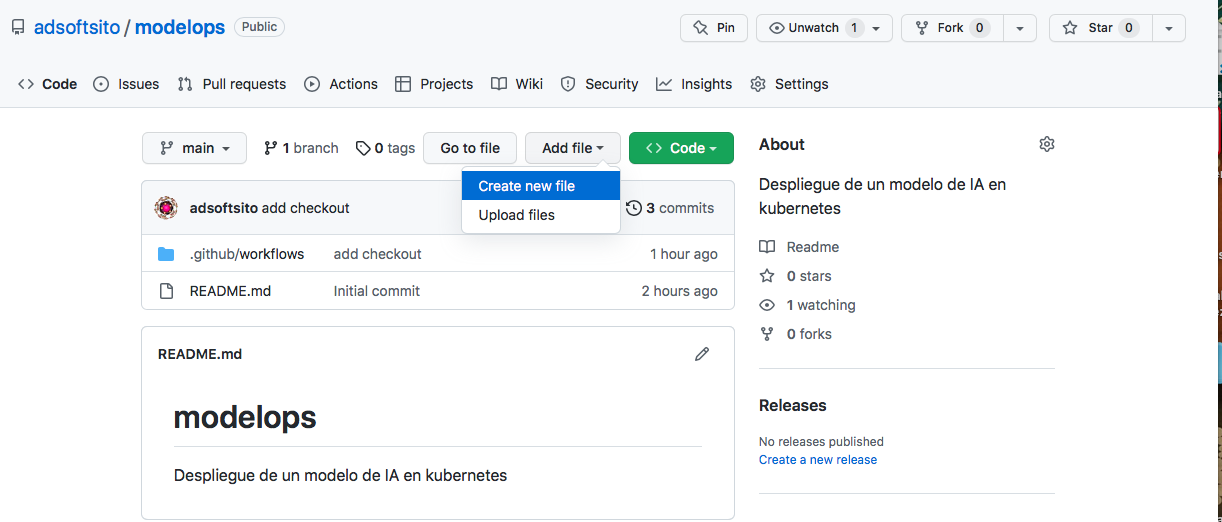
5.- Realizar Start Commit, poner un mensaje para commit y click en **Commit new file**



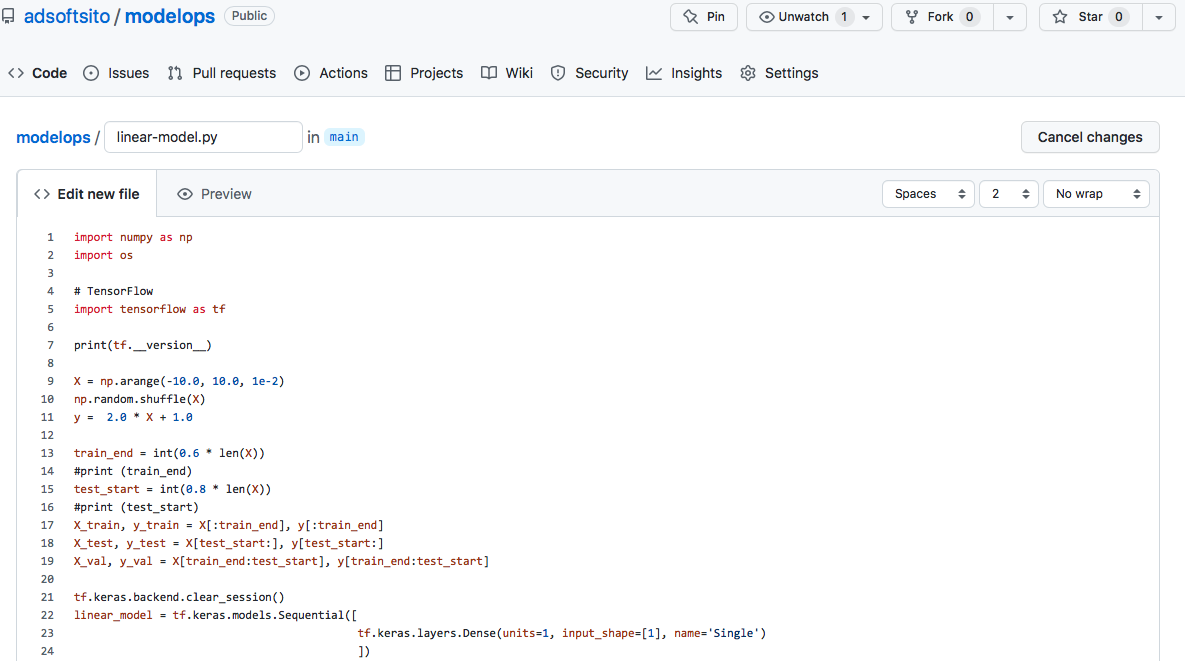
6.- Revisar en Actions que el workflow se ejecute y genere el siguiente resultado.



7.- Crear un archivo nuevo en Code:



7.- Introducir el programa que contenga el modelo a desplegar: **linear-model.py**, con el siguiente código de ejemplo:



**import numpy as np**

**import os**

**# TensorFlow**

**import tensorflow as tf**

**print(tf.\_\_version\_\_)**

**X = np.arange(-10.0, 10.0, 1e-2)**

**np.random.shuffle(X)**

**y = 2.0 \* X + 1.0**

**train\_end = int(0.6 \* len(X))**

**test\_start = int(0.8 \* len(X))**

**X\_train, y\_train = X[:train\_end], y[:train\_end]**

**X\_test, y\_test = X[test\_start:], y[test\_start:]**

**X\_val, y\_val = X[train\_end:test\_start], y[train\_end:test\_start]**

**tf.keras.backend.clear\_session()**

**linear\_model = tf.keras.models.Sequential([**

**tf.keras.layers.Dense(units=1, input\_shape=[1], name='Single')**

**])**

**linear\_model.compile(optimizer=tf.keras.optimizers.SGD(), loss=tf.keras.losses.mean\_squared\_error)**

**print(linear\_model.summary())**

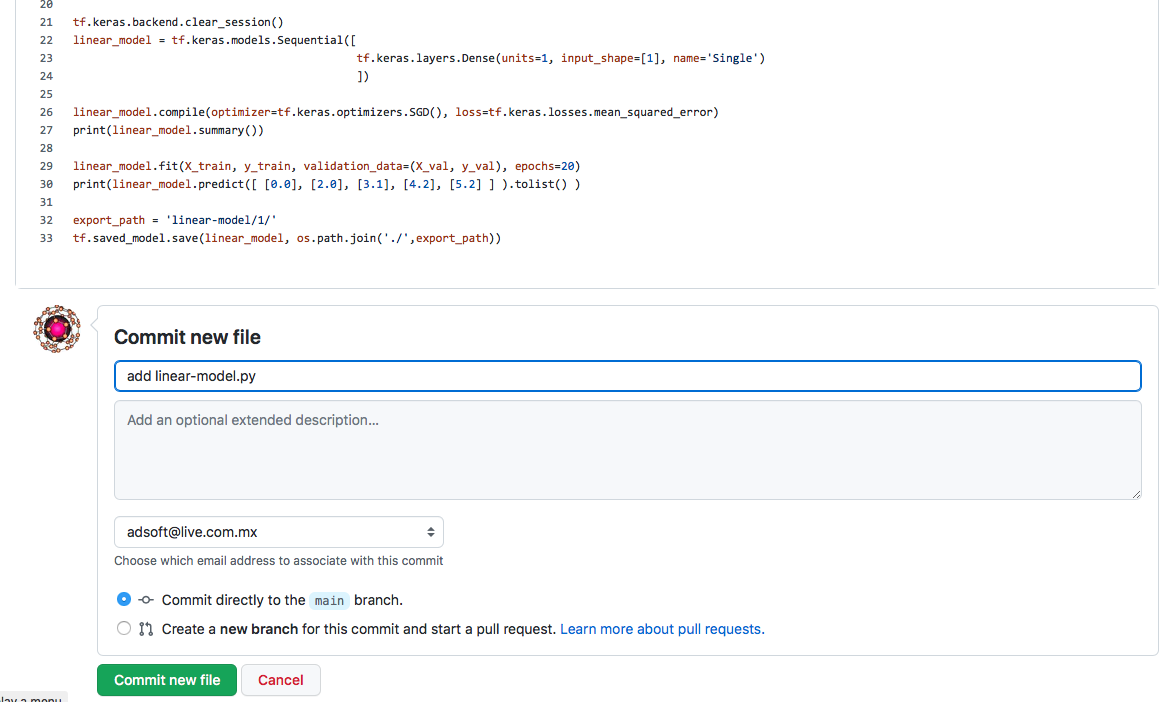
**linear\_model.fit(X\_train, y\_train, validation\_data=(X\_val, y\_val), epochs=20)**

**print(linear\_model.predict([ [0.0], [2.0], [3.1], [4.2], [5.2] ] ).tolist() )**

**export\_path = 'linear-model/1/'**

**tf.saved\_model.save(linear\_model, os.path.join('./',export\_path))**

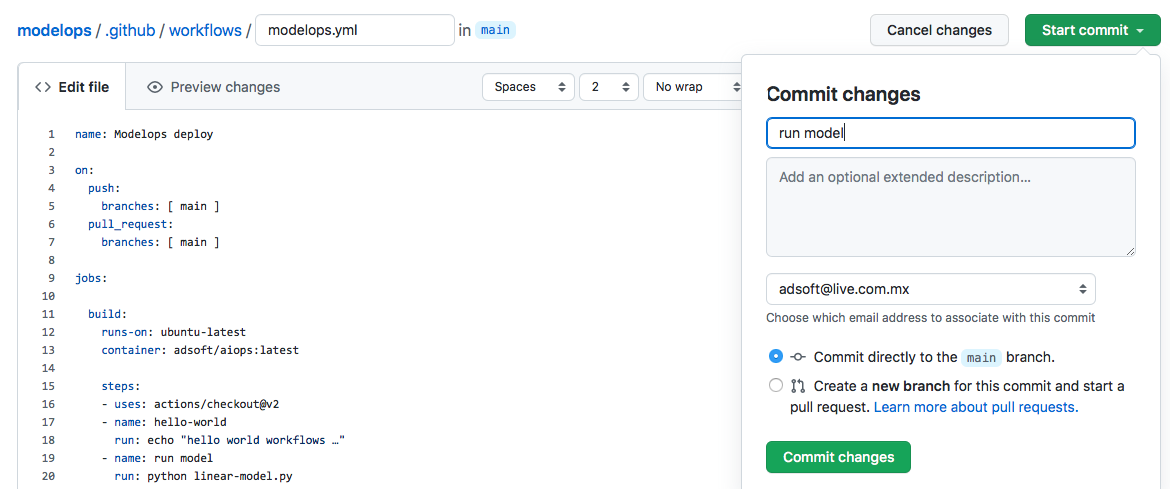
8.- Realizar commit



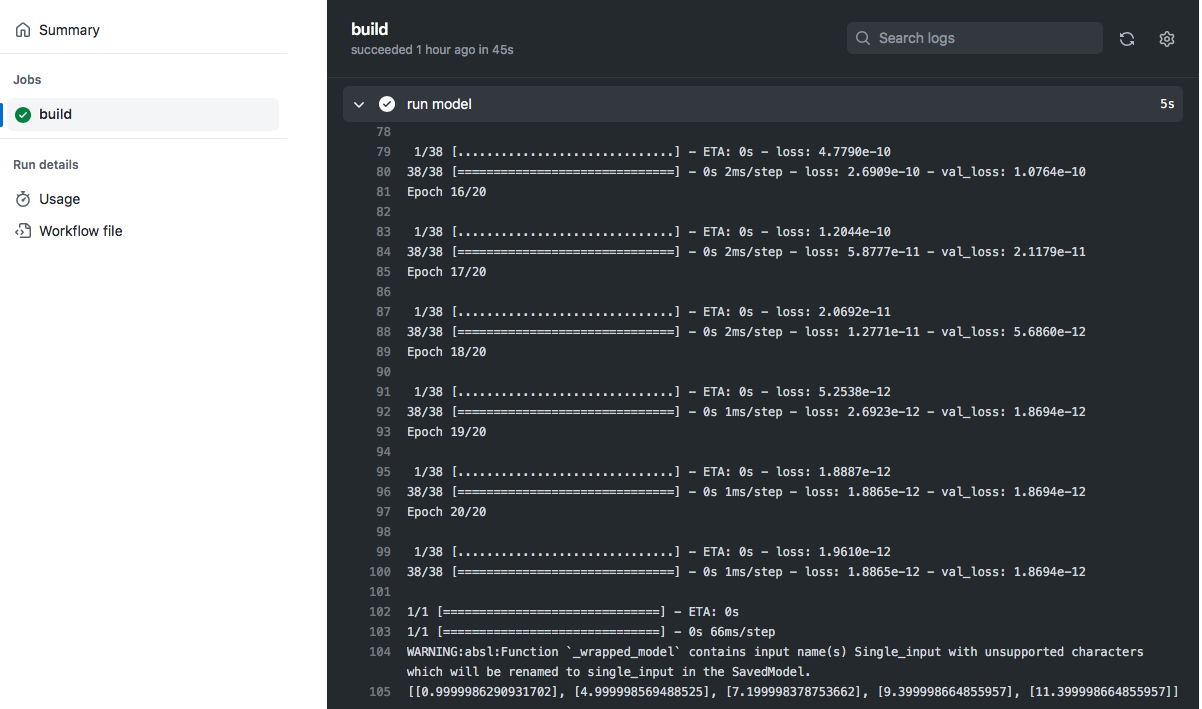
9.- Actualizar workflow para ejecutar el programa, agregar las siguiente líneas y hacer **Start commit.**

**- name: run model**

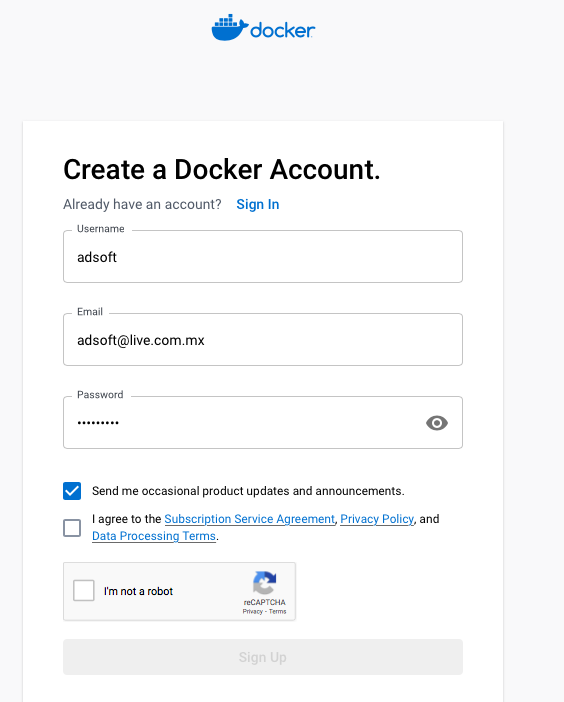
**run: python linear-model.py**



10.- Verificar que el workflow se dispare automáticamente, ejecute el programa y genere la siguiente salida.

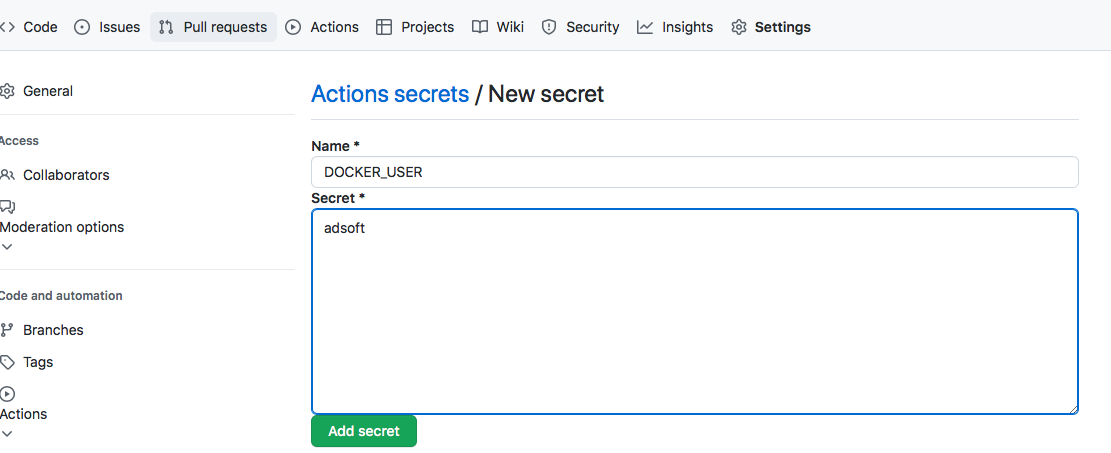


11.- Crear una cuenta en <https://hub.docker.com/signup>, guardar en bloc de notas el nombre de usuario y password.

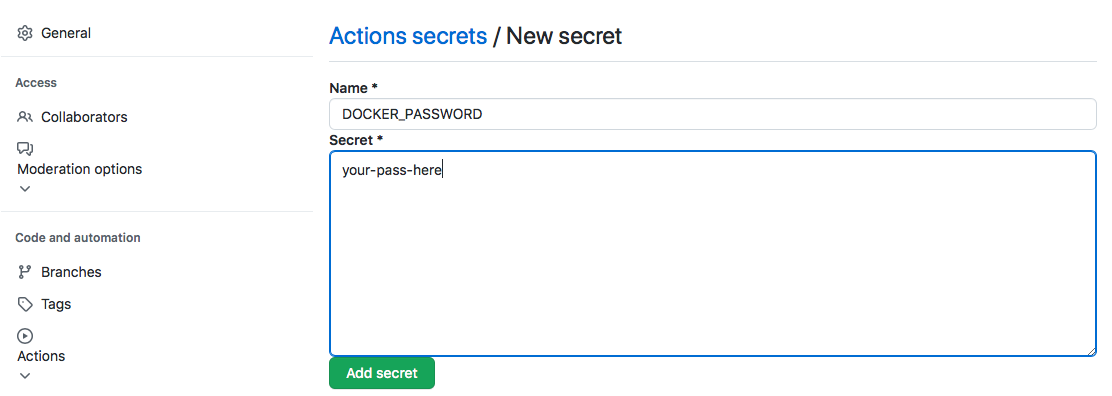


12.- Crear las siguientes secrets en Github en **Settings -> Secrets -> Actions -> New Repository Secret**

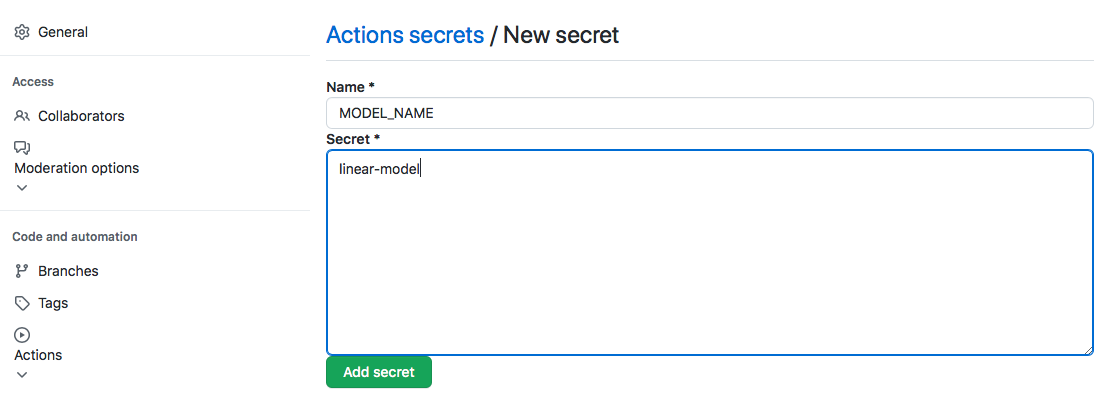
* DOCKER\_USER: AngelAnaya314
* A01741351@tec.mx
* Lionelmessi10\*



* DOCKER\_PASSWORD ( password de docker hub )



* MODEL\_NAME (linear-model)



13.- Actualizar workflow para generar una Docker image que contenga **tensorflow** y nuestro modelo entrenado, hacer commit.

**- name: docker login**

**env:**

**DOCKER\_USER: ${{secrets.DOCKER\_USER}}**

**DOCKER\_PASSWORD: ${{secrets.DOCKER\_PASSWORD}}**

**run: |**

**docker login -u $DOCKER\_USER -p $DOCKER\_PASSWORD**

**- name: Download and run the Docker base image**

**run: docker run -d --name serving\_base tensorflow/serving**

**- name: copy model to the Docker image**

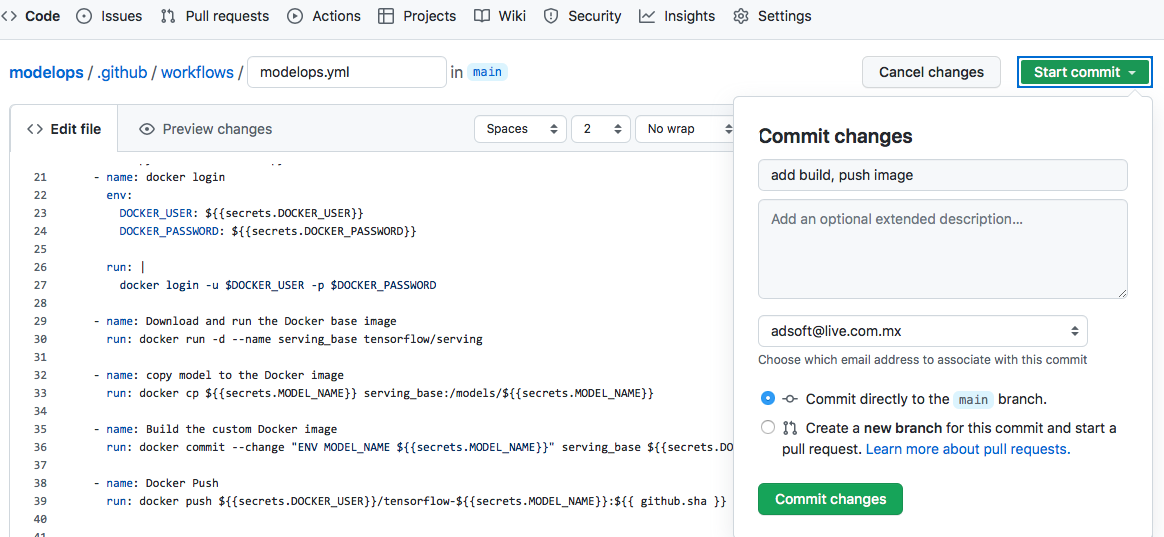
**run: docker cp ${{secrets.MODEL\_NAME}} serving\_base:/models/${{secrets.MODEL\_NAME}}**

**- name: Build the custom Docker image**

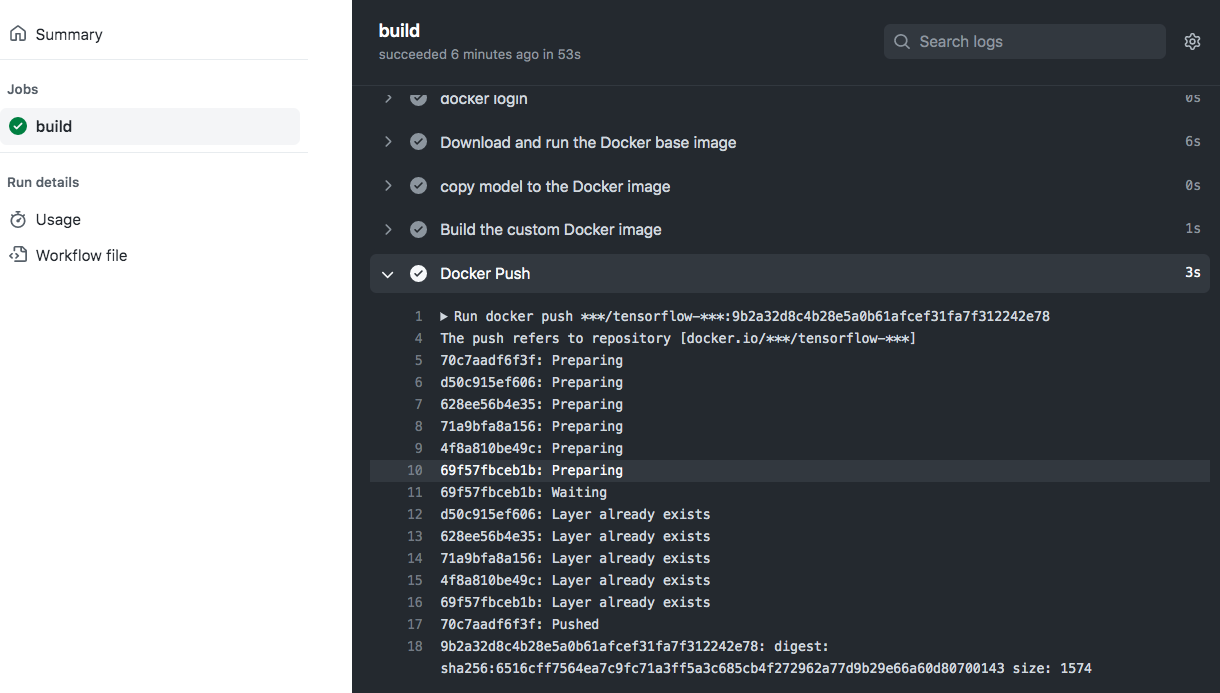
**run: docker commit --change "ENV MODEL\_NAME ${{secrets.MODEL\_NAME}}" serving\_base ${{secrets.DOCKER\_USER}}/tensorflow-${{secrets.MODEL\_NAME}}:${{ github.sha }}**

**- name: Docker Push**

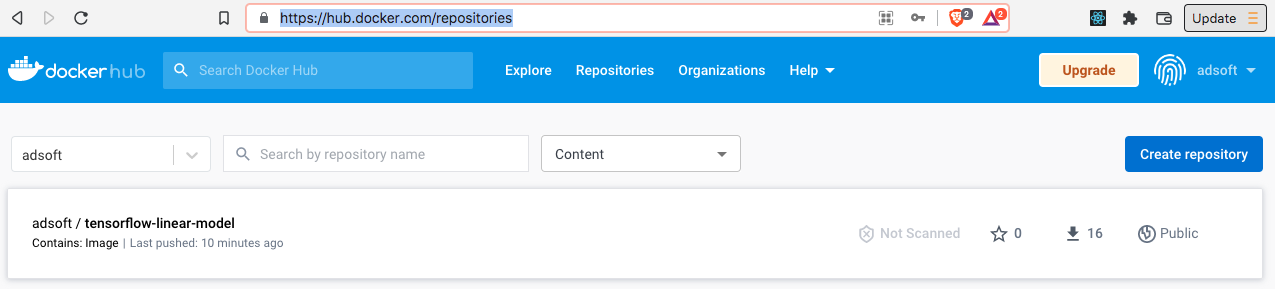
**run: docker push ${{secrets.DOCKER\_USER}}/tensorflow-${{secrets.MODEL\_NAME}}:${{ github.sha }}**



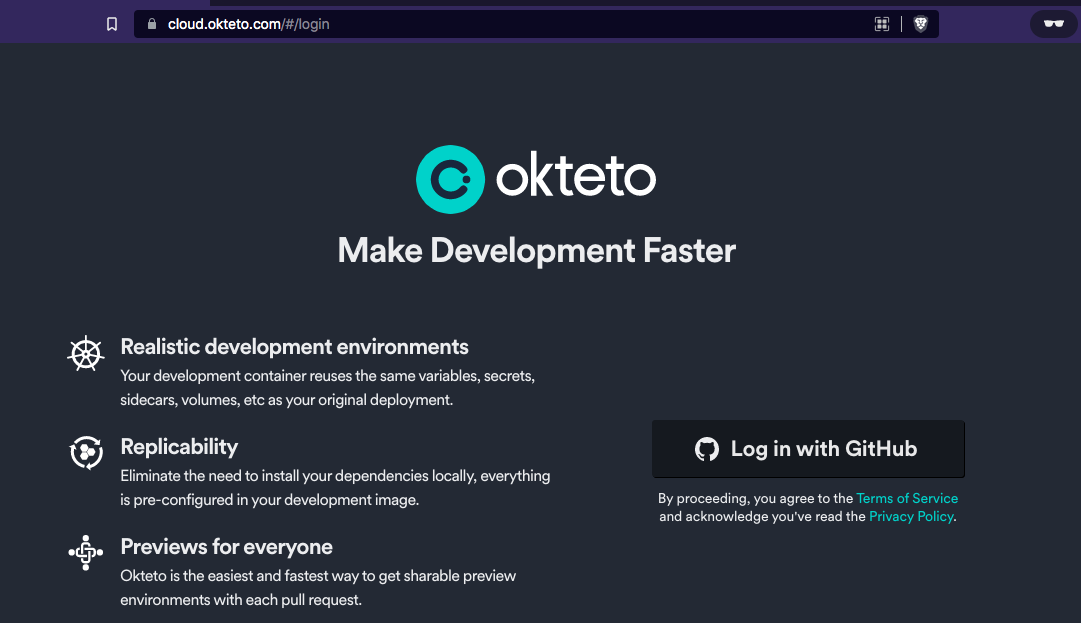
14.- Verificar que el workflow se complete sin errores y genere la siguiente salida



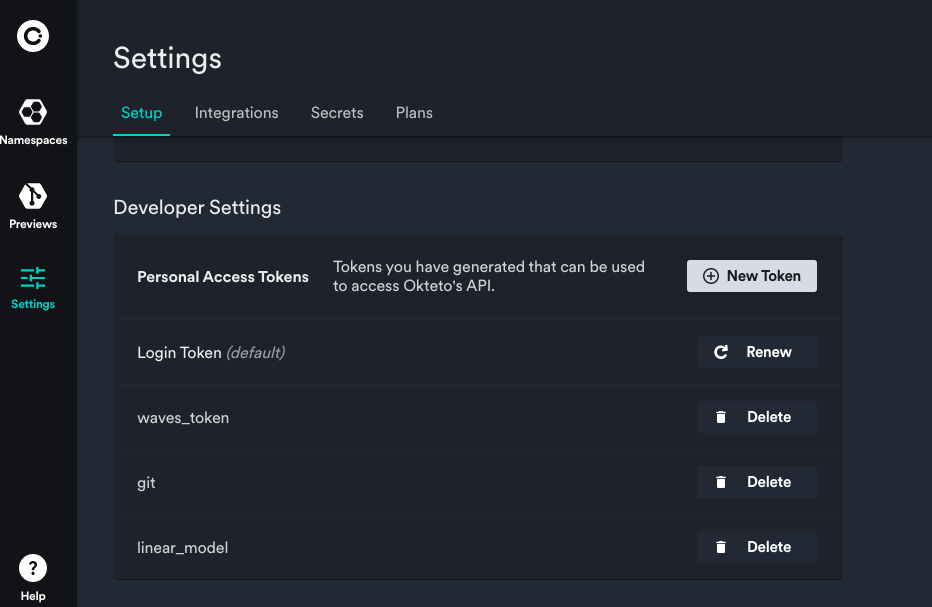
15.- Revisar que la imagen de nuestro Docker sea publicada en <https://hub.docker.com/repositories>



16.- Crear una cuenta en la nube <https://cloud.okteto.com/#/login> usando tu usuario de github.

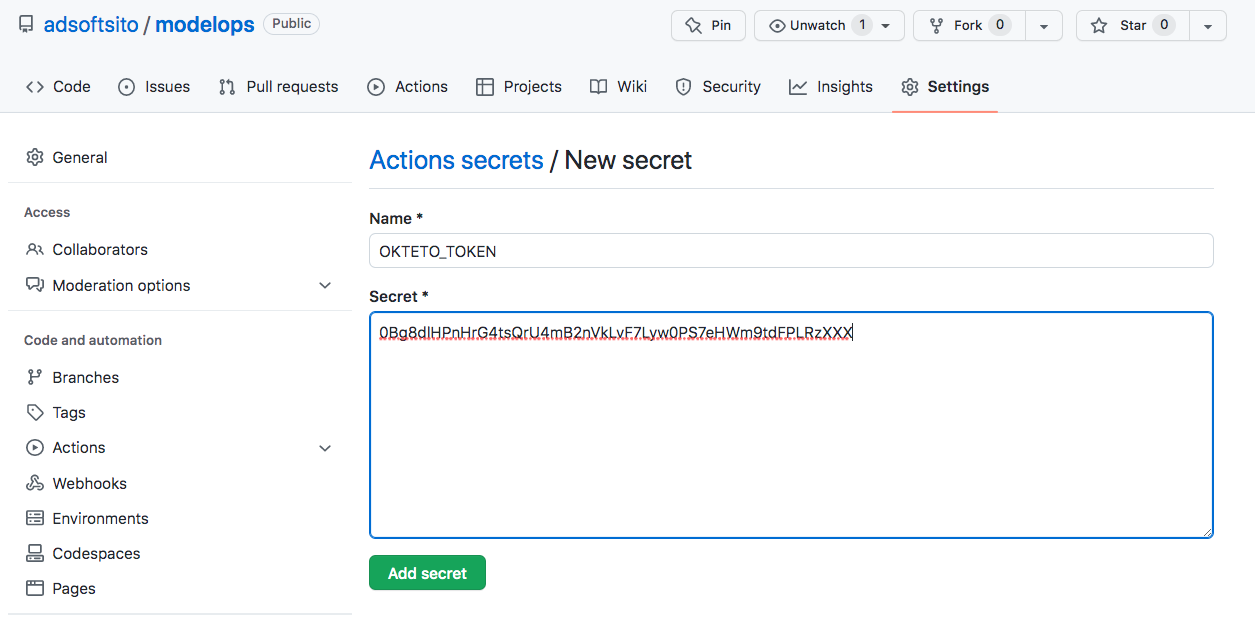


17.- Crear un token de octeto, guardarlo en un block de notas.

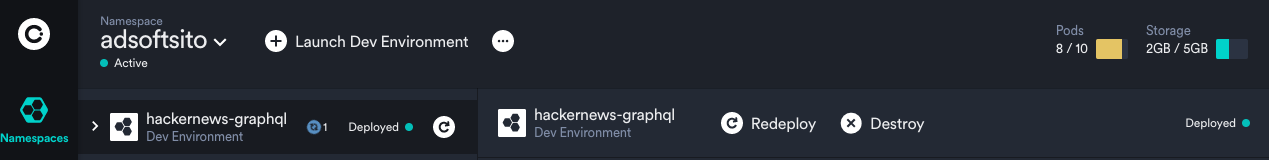


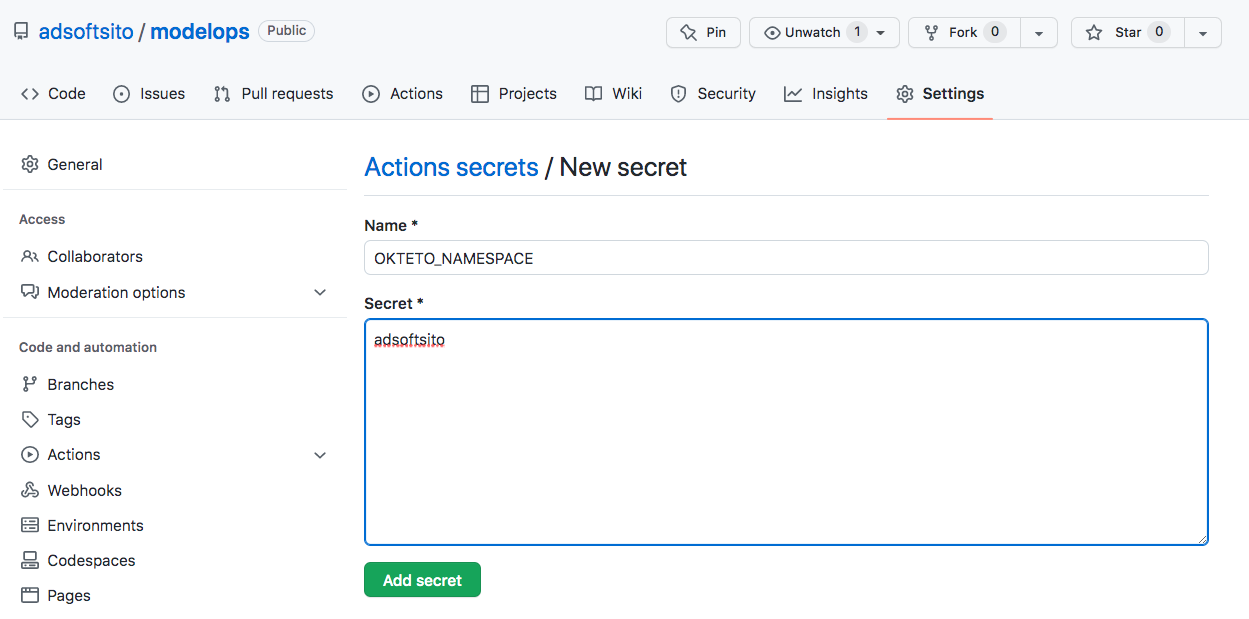
ywBLRCfnlNEz35RXkx1nqvUWQx7mSSHf5xAdb0AIYSOqqpmL

19.- Crear un nuevo secret llamado: OKTETO\_TOKEN y poner como value el token recien creado

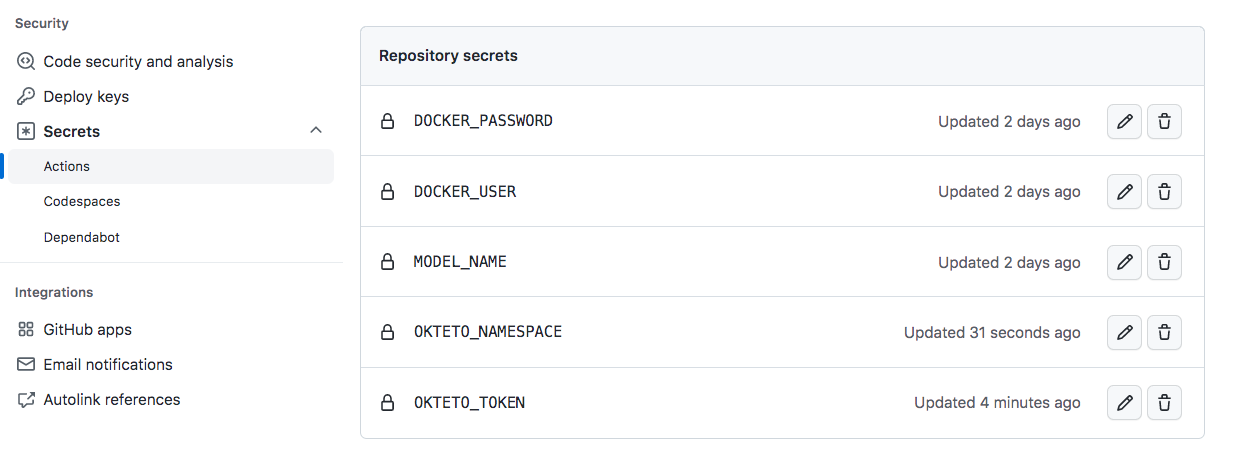


20.- Crear un nuevo secret **OKTETO\_NAMESPACE**, con el nombre de tu namespace de okteto, regularmente es tu nombre de usuario de github en minusculas.





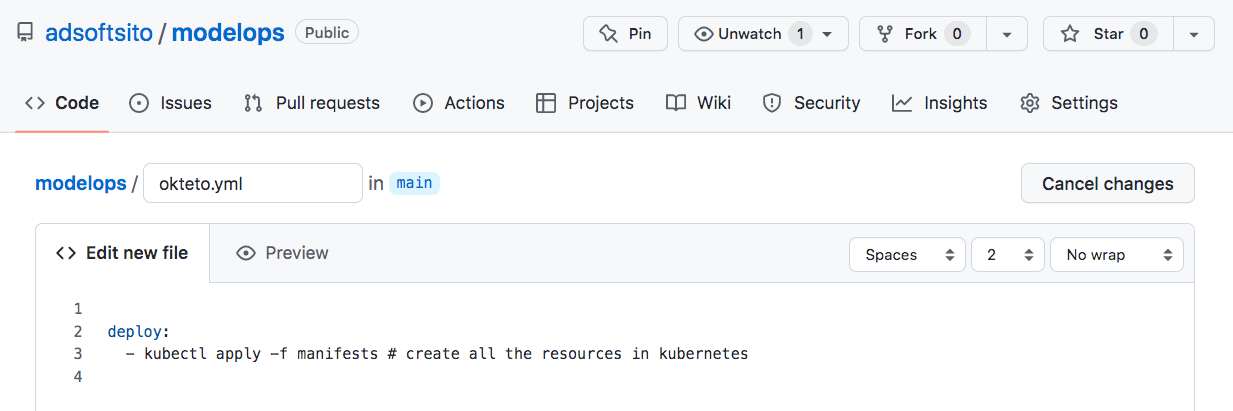
21.- Deberemos contar con 5 secrets



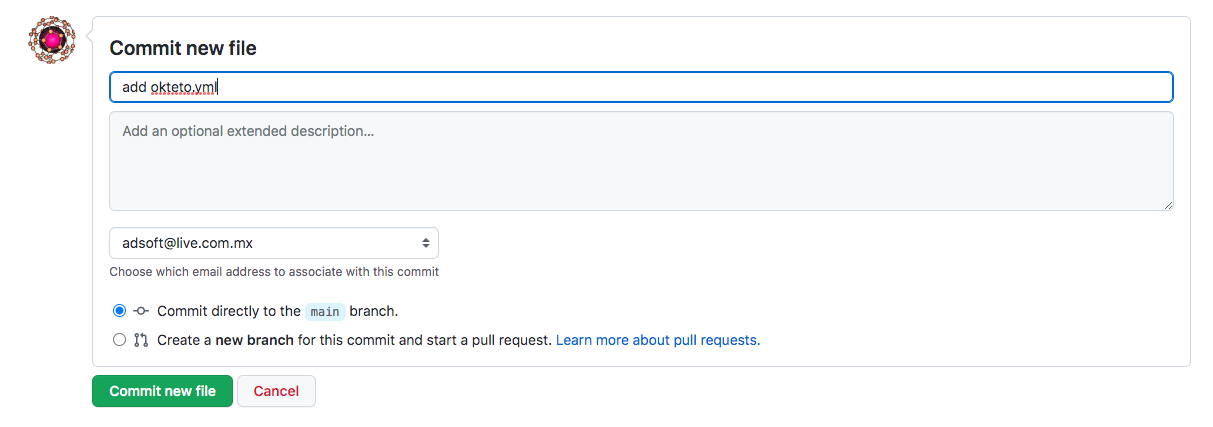
22.- Dentro de Code, crear el archivo **okteto.yml, con el código:**

**deploy:**

**- kubectl apply -f manifests # create all the resources in kubernetes**

****

23.- Guardar okteto.yml con **commit**

****

**24.- Dentro de Code,** crear el archivo **manifests/linear\_model\_k8s.yaml,** con el siguiente codigo.

**#manifest**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: {model\_name}-deployment**

**spec:**

**selector:**

**matchLabels:**

**app: {model\_name}-server**

**replicas: 3**

**template:**

**metadata:**

**labels:**

**app: {model\_name}-server**

**spec:**

**containers:**

**- name: {model\_name}-container**

**#image: gcr.io/tensorflow-serving/resnet**

**image: docker.io/{docker\_user}/tensorflow-{model\_name}:{github.sha}**

**ports:**

**- containerPort: 8501**

**---**

**apiVersion: v1**

**kind: Service**

**metadata:**

**labels:**

**run: {model\_name}-service**

**name: {model\_name}-service**

**spec:**

**ports:**

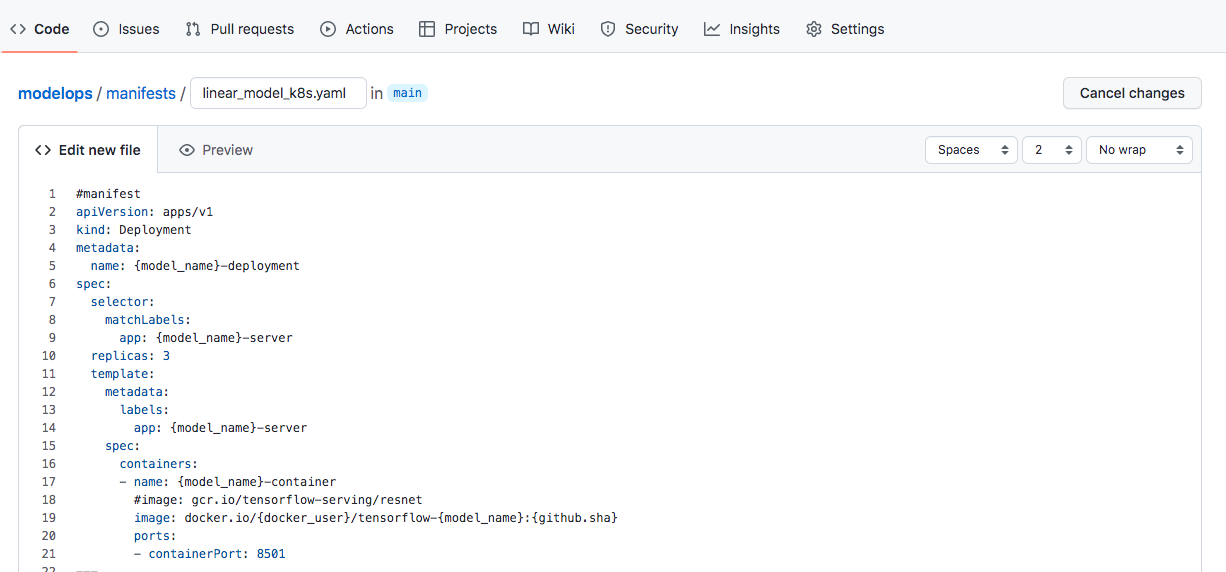
**- port: 8501**

**targetPort: 8501**

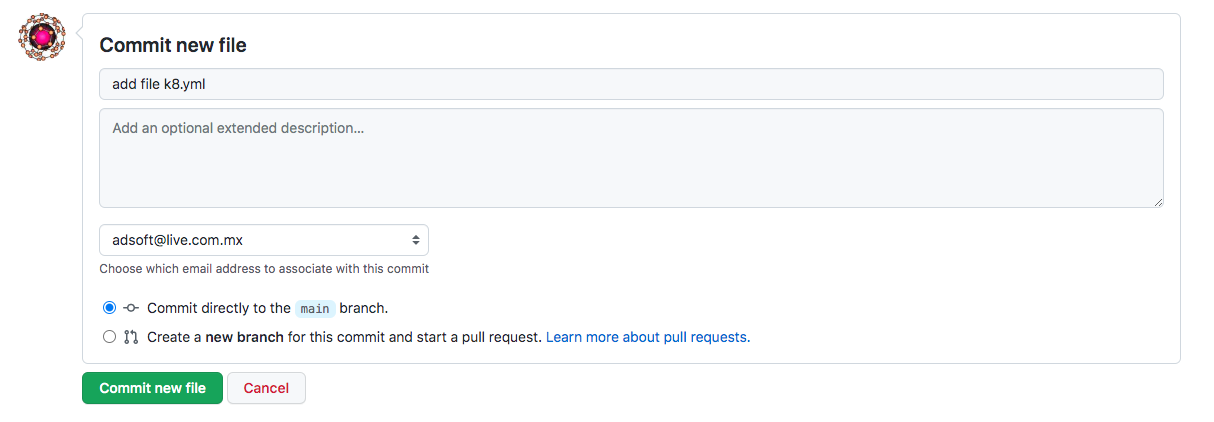
**selector:**

**app: {model\_name}-server**

**type: LoadBalancer**

****

**25.-** Guardar realizando **commit**

****

26.- Actualizar workflow para implementar el despliegue de nuestra Docker image en la nube de okteto.

**- name: update docker user in k8**

**run: sed -i 's/{docker\_user}/${{secrets.DOCKER\_USER}}/g' manifests/linear\_model\_k8s.yaml**

**- name: update model name in k8**

**run: sed -i 's/{model\_name}/${{secrets.MODEL\_NAME}}/g' manifests/linear\_model\_k8s.yaml**

**- name: update \_ by -**

**run: sed -i 's/\_/-/g' manifests/linear\_model\_k8s.yaml**

**- name: update sha in k8**

**run: sed -i 's/{github.sha}/${{ github.sha }}/g' manifests/linear\_model\_k8s.yaml**

**- name: list**

**run: cat manifests/linear\_model\_k8s.yaml**

**- uses: okteto/context@2.7.0**

**with:**

**token: ${{secrets.OKTETO\_TOKEN }}**

**- name: Deploy and Wait**

**uses: okteto/actions/deploy@v1**

**env:**

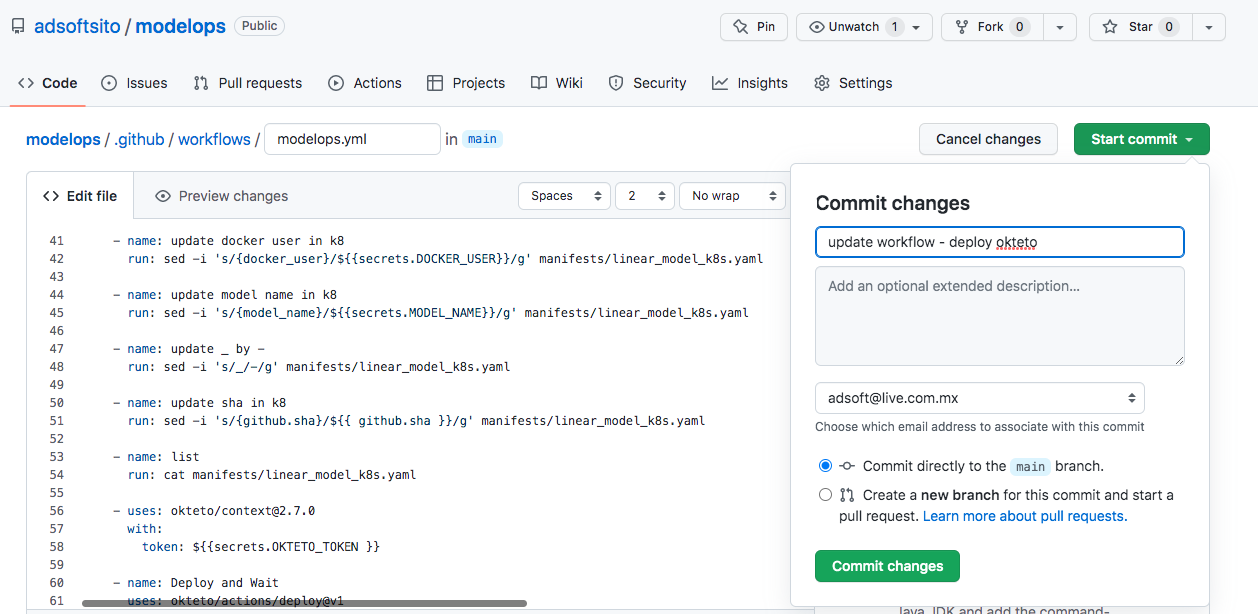
**KUBECONFIG: ${{ steps.namespace.outputs.kubeconfig }}**

**with:**

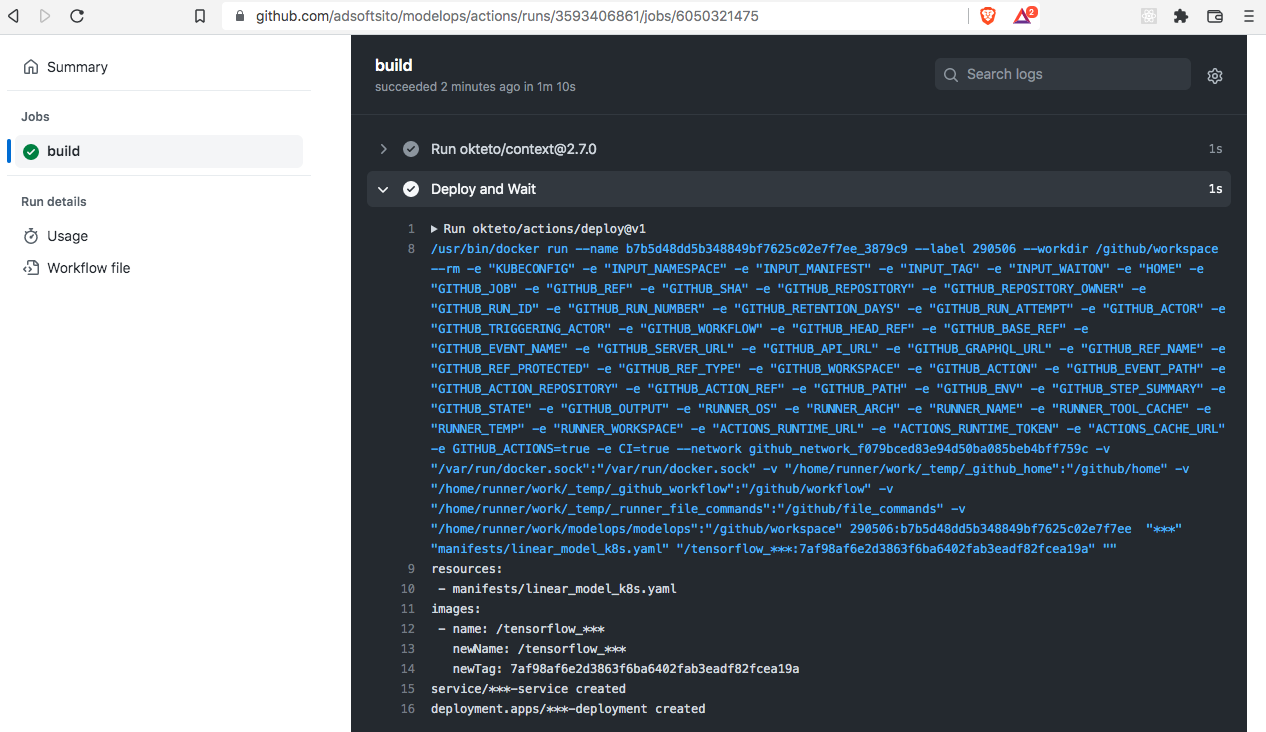
**namespace: ${{secrets.OKTETO\_NAMESPACE }}**

**manifest: manifests/linear\_model\_k8s.yaml**

**tag: ${{ secrets.DOCKER\_USERNAME }}/tensorflow\_${{secrets.MODEL\_NAME}}:${{ github.sha }}**



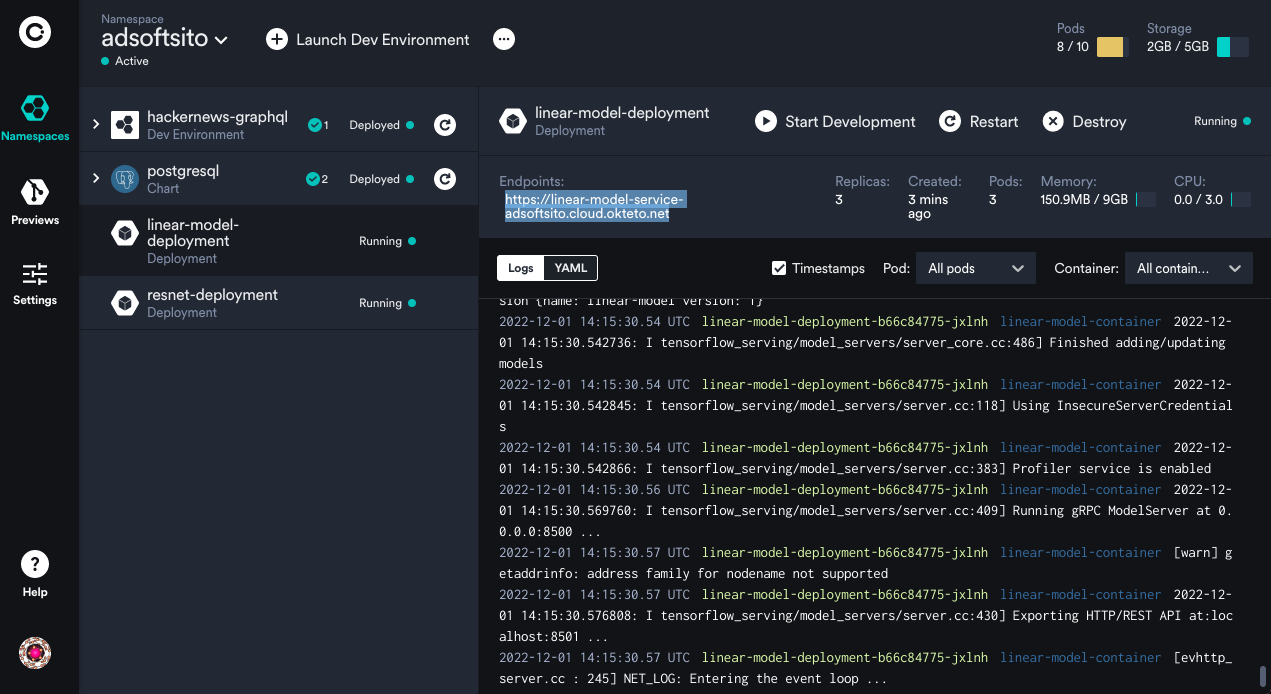
27.- Verificar se despair workflow y genere la siguiente salida:



28.- Revisar la nube de okteto y revisar nuestro modelo este publicado y tengamos el URL de nuestra API, ejemplo: <https://linear-model-service-adsoftsito.cloud.okteto.net/>

NOTA: el URL completo debe contener la versión del modelo, nombre y operación, por tanto para probar nuestro modelo sería:

<https://linear-model-service-adsoftsito.cloud.okteto.net/v1/models/linear-model:predict>



29.- En tu Visual Studio Code local, ingresar el siguiente código para realizar un cliente ( ***model\_client.py*** )que consuma nuestro servicio.

**import json**

**import numpy as np**

**import requests**

**# The server URL specifies the endpoint of your server running the linear\_model**

**# model with the name "linear\_model" and using the predict interface.**

**SERVER\_URL = 'https://linear-model-service-adsoftsito.cloud.okteto.net/v1/models/linear-model:predict'**

**def main():**

**predict\_request = '{"instances" : [ [0.0], [1.0], [2.0] ]}'**

**# Send few actual requests and report average latency.**

**total\_time = 0**

**num\_requests = 10**

**index = 0**

**for \_ in range(num\_requests):**

**response = requests.post(SERVER\_URL, data=predict\_request)**

**response.raise\_for\_status()**

**total\_time += response.elapsed.total\_seconds()**

**prediction = response.json()**

**print (prediction)**

**print('Prediction class: {}, avg latency: {} ms'.format(**

**np.argmax(prediction), (total\_time \* 1000) / num\_requests))**

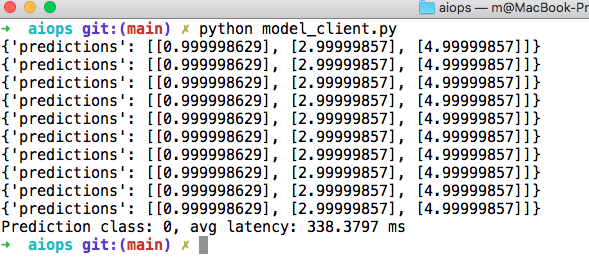
**if \_\_name\_\_ == '\_\_main\_\_':**

**main()**

import jsonimport numpy as npimport requestsimport streamlit as st # The server URL specifies the endpoint of your server running the linear\_model# model with the name "linear\_model" and using the predict interface.SERVER\_URL = 'https://linear-model-service-adsoftsito.cloud.okteto.net/v1/models/linear-model:predict' def callApi(): predict\_request = '{"instances" : [ [20.0], [25.0], [30.0], [35.0] ]}' # Send few actual requests and report average latency. total\_time = 0 num\_requests = 10 index = 0 for \_ in range(num\_requests): response = requests.post(SERVER\_URL, data=predict\_request) response.raise\_for\_status() total\_time += response.elapsed.total\_seconds() prediction = response.json() st.write (prediction) st.title('Api demo')

callApi()

30.- Probar nuestro cliente python.



RETO :

* Cambiar la fórmula de nuestro modelo **y = 2x + 1**, por alguna otra que represente algún cálculo de interés en tu carrera.
* Tomando como base ***model\_client.py****, construir una app Streamlit que permita capturar los valores a predecir y consuma nuestra API.*

[*https://www.postman.com/downloads/*](https://www.postman.com/downloads/)

import numpy as npimport os# TensorFlowimport tensorflow as tf print(tf.\_\_version\_\_)# opcional, leer datos de CSV, SQL server, Oracle, APIX = np.arange(-10.0, 10.0, 1e-2)np.random.shuffle(X)y = 2.0 \* X + 1.0train\_end = int(0.6 \* len(X))test\_start = int(0.8 \* len(X))X\_train, y\_train = X[:train\_end], y[:train\_end]X\_test, y\_test = X[test\_start:], y[test\_start:]X\_val, y\_val = X[train\_end:test\_start], y[train\_end:test\_start]tf.keras.backend.clear\_session()linear\_model = tf.keras.models.Sequential([ tf.keras.layers.Dense(units=1, input\_shape=[1], name='Single') ])linear\_model.compile(optimizer=tf.keras.optimizers.SGD(), loss=tf.keras.losses.mean\_squared\_error)print(linear\_model.summary())linear\_model.fit(X\_train, y\_train, validation\_data=(X\_val, y\_val), epochs=20)print(linear\_model.predict([ [0.0], [2.0], [3.1], [4.2], [5.2] ] ).tolist() ) export\_path = 'linear-model/1/'tf.saved\_model.save(linear\_m

tf.saved\_model.save(linear\_model, os.path.join('./',export\_path))