**BOOTCAMP  
DevOps Engineer**

\_\_\_

**Profesor: Zdenko Hraste**

**Comisión 3 - Grupo 2**

**Integrantes: Alexis Roldán, Florencia Pezzella, Mauro Usay**

**Objetivo**

El objetivo de este desafío es realizar un trabajo que integre lo visto en los distintos módulos de esta fase. El desafío será grupal (grupos de 2 a 5 personas) y tendrán que dividirse las tareas para lograr hacer todo.

**Desafío:**

**Prerrequisitos:**

- Repositorio de código (GitHub, gitlab, bitbucket, etc) a elección del grupo

- Grupo 2 a 5 personas

- Decidir qué aplicación o conjunto de aplicaciones utilizaran para el desafío

**PARTE 1: VAGRANT FILE**

1) Durante esta parte del desafío, tendrán que crear una máquina en Vagrant donde instalarán todas las herramientas utilizadas en la práctica, lo recomendable es que trabajen de forma tal que puedan ir agregando herramientas a medida que las van necesitando (docker, kubectl, minikube, microk8s, kind o cualquier otra herramienta utilizada para crear un cluster de kubernetes).

2) Para configurar la máquina de vagrant podrán usar el provider que más les guste, incluso el de Ansible.

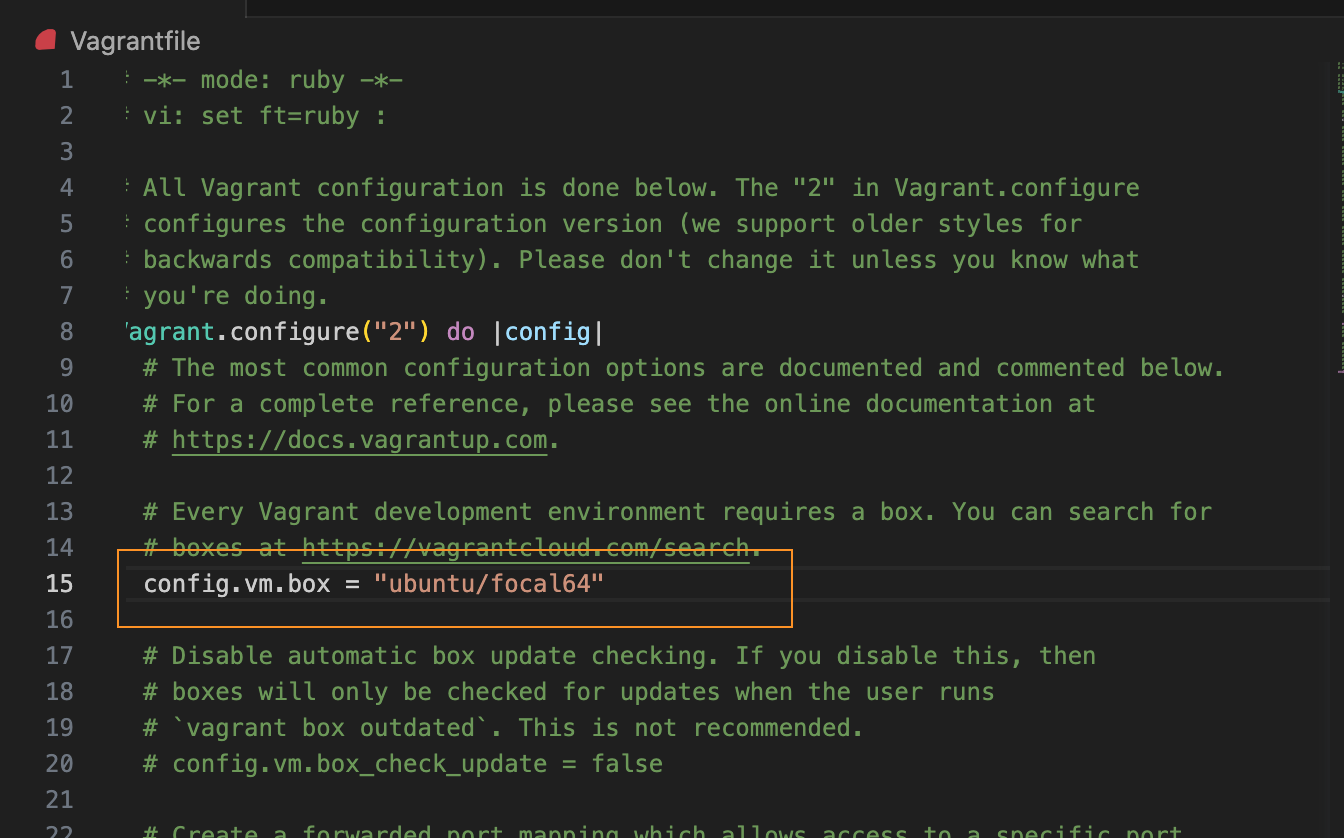
3) Tengan en cuenta que el objetivo de esta máquina virtual es el de poder replicar absolutamente todo el desafío de una forma sencilla a cualquier persona que tenga acceso al repositorio, recomendamos seguir la modalidad utilizada en el módulo 16 para la parte de kubernetes.

**VAGRANTFILE:**

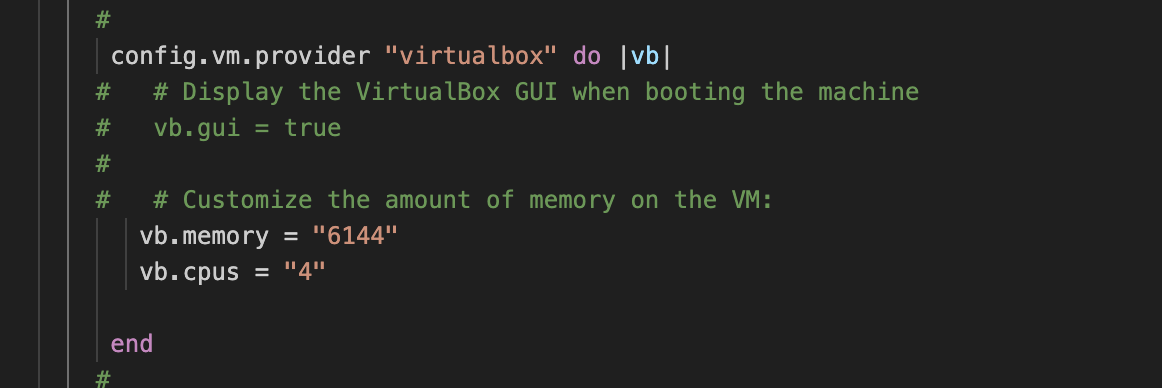
Para poder crear el entorno de Vagrant, primero debe crearse una carpeta dentro la máquina, en este caso, estoy trabajando dentro del sistema operativo MAC.

Para poder instalar la máquina virtual primero se necesita un vagrant file. Para eso se indica el comando:

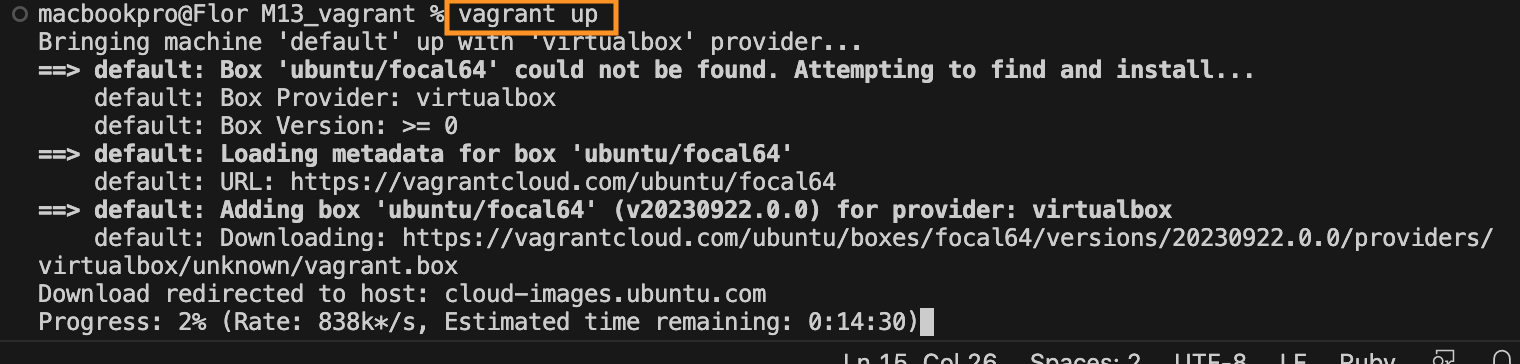
**vagrant init ubuntu/focal64**



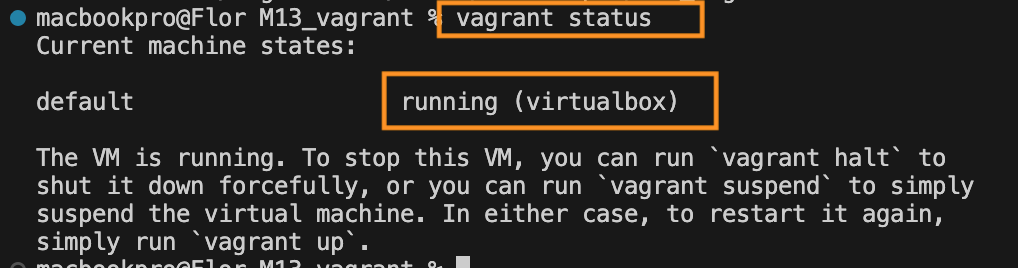
Una vez creado el vagrantfile, se edita el archivo en función de los recursos que necesitemos.



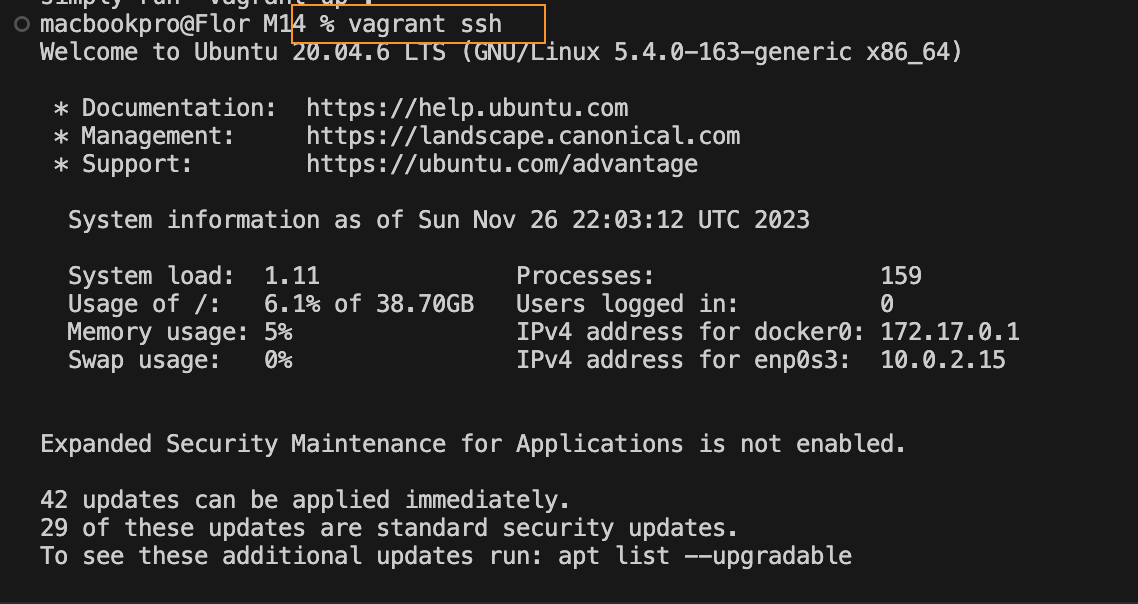
Mediante el comando **vagrant up** se efectúa la instalación del box.



Una vez instalado el box, se consulta el estado mediante el comando **vagrant status**

****

Mediante el comando **vagrant ssh** se accede forma remota a la máquina virtual.



Dentro del script de Shell se instalarán las herramientas que requieren nuestra aplicación

config.vm.provision "shell", path: "bootstrap.sh"

**SCRIPTS DE INSTALACIÓN:**

En el siguiente archivo llamado bootstrap.sh se ejecutan los scripts de instalación:

# Docker install

curl -fsSL https://get.docker.com -o get-docker.sh

sudo sh get-docker.sh

# Use docker without sudo

sudo groupadd docker

sudo usermod -aG docker vagrant

newgrp docker

#Install kubectl

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

#Install minikube

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube

Con el comando **vagrant validate** se valida si es correcto el Vagrantfile.

Luego con el comando **vagrant up** se inicia la máquina virtual.

Finalmente se ejecuta mediante la conexión por SSH con el comando **vagrant ssh**.

**Parte 2: Docker**

1) Durante esta parte del desafío tendrán que instalar las herramientas necesarias en la máquina virtual que les permita Dockerizar aplicaciones.

2) Habiendo elegido la aplicación o conjunto de aplicaciones que van a dockerizar, procederán a crear los Dockerfile y Docker compose necesarios (de forma manual)

**REPOSITORIO EN GITHUB:**

Se crea el repositorio en GitHub: [**https://github.com/alexlpda1420/desafio16-17**](https://github.com/alexlpda1420/desafio16-17)

**DOCKERFILE:**

A continuación, se procederán a crear los Dockerfile y Docker compose necesarios (de forma manual).

Se construye un Dockerfile para ejecutar Plex Media Server. Se trata de un servicio que funciona como un servidor multimedia donde se puede reproducir todo aquello que se encuentre dentro de mi computadora, sería como mi propio Mubi/Netflix/etc.

Para eso se ejecuta el Dockerfile con las siguientes instrucciones:

# Ubuntu image

FROM ubuntu:latest

# Update and install package

RUN apt-get update && \

apt-get install -y curl && \

apt-get clean && \

rm -rf /var/lib/apt/lists/\*

# Download and install Plex Media Server

RUN curl -LO "https://downloads.plex.tv/plex-media-server-new/1.32.6.7557-1cf77d501/debian/plexmediaserver\_1.32.6.7557-1cf77d501\_amd64.deb" && \

dpkg -i plexmediaserver\_1.32.6.7557-1cf77d501\_amd64.deb

# Add port and volumes

EXPOSE 32400

VOLUME /config

VOLUME /media

CMD ["/usr/lib/plexmediaserver/Plex Media Server", "-c", "/usr/sbin/start\_pms", "/config", "/media"]

FROM: Se crea el docker file a partir de la imagen de Ubuntu.

RUN: Actualiza el sistema y corre los paquetes.

Descarga e instala Plex.

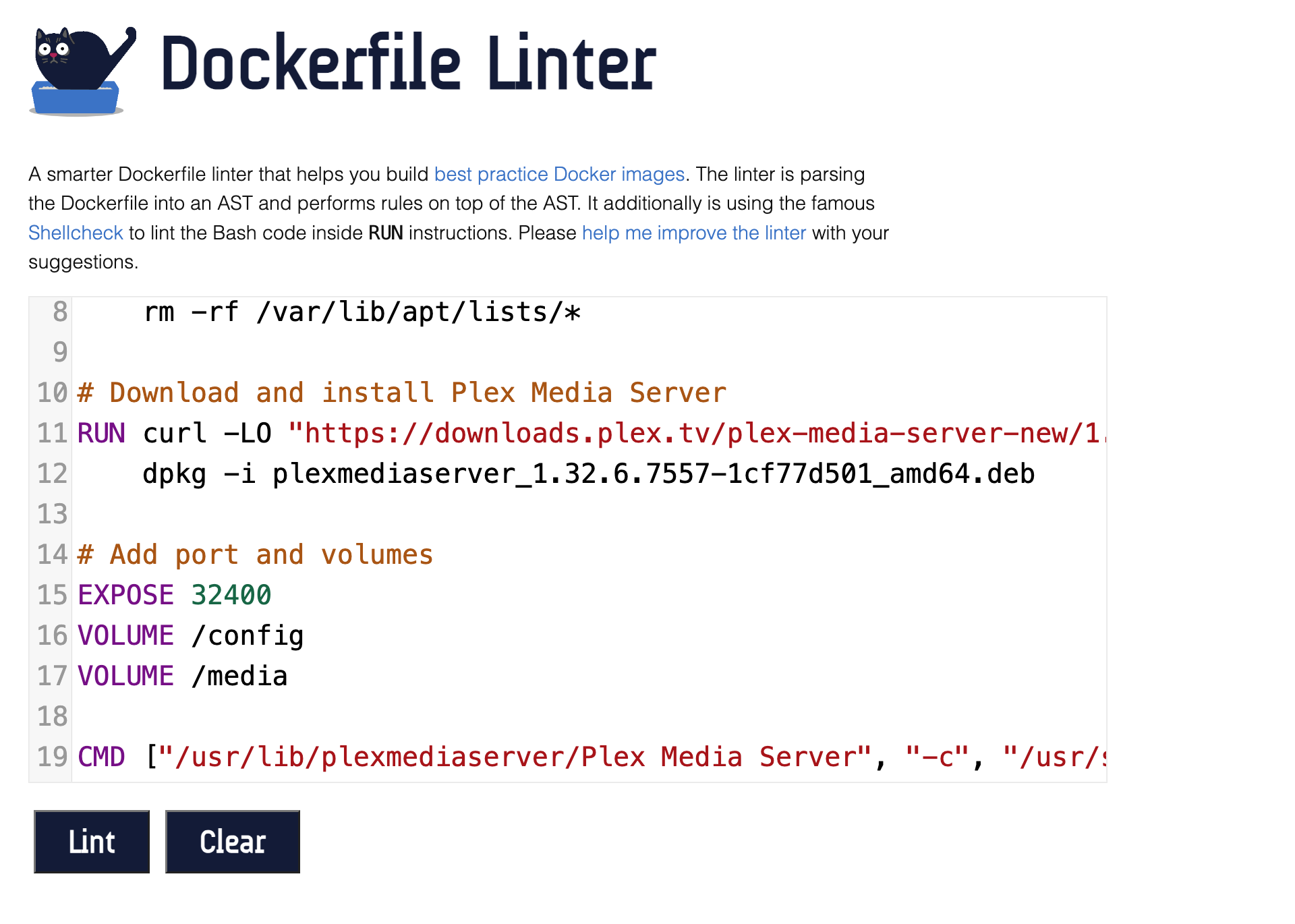
EXPOSE: El puerto que Plex Media Server utiliza para la comunicación es el 32400.

VOLUME: Se definen los volúmenes para guardar la configuración.

CMD: Y se inicia y ejecuta el contenedor.

**REVISIÓN DE VULNERABILIDADES:**

Antes de iniciar la dockerización se verifica el Dockerfile con la herramienta de código abierto llamada Hadolint. La misma fue creada por GitHub.



Una vez verificado, podemos dockerizar nuestra aplicación.

**EJUTANDO DOCKER PARA PLEX MEDIA SERVER:**

Se ejecuta en la terminal el siguiente comando para crear la imagen llamada *docker-on-plex*:

**docker build . -t docker-on-plex**



Es así como la imagen finalmente se construye.

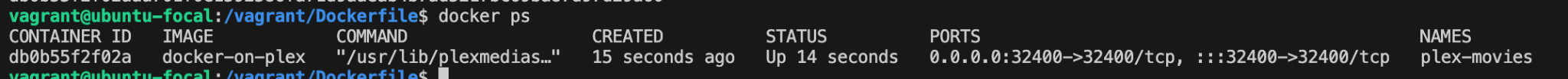
Para consultar si la imagen efectivamente fue creada, se indica el comando:

**docker image ls**

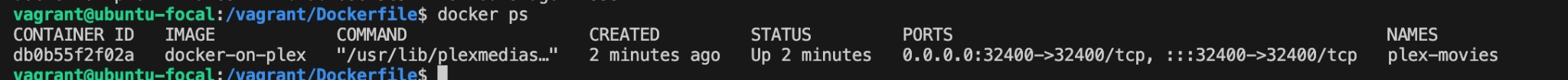


Ahora se crea el contenedor mediante el siguiente comando:

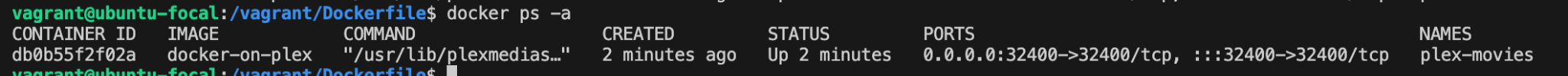
**docker run -d --name plex-movies -p 32400:32400 docker-on-plex**



Y se consulta mediante el comando **docker ps** el listado de contenedores creados y ejecutando en este momento:

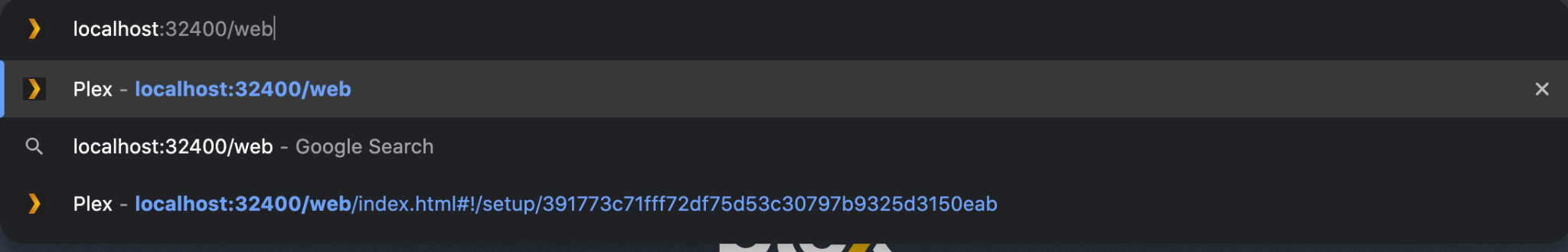


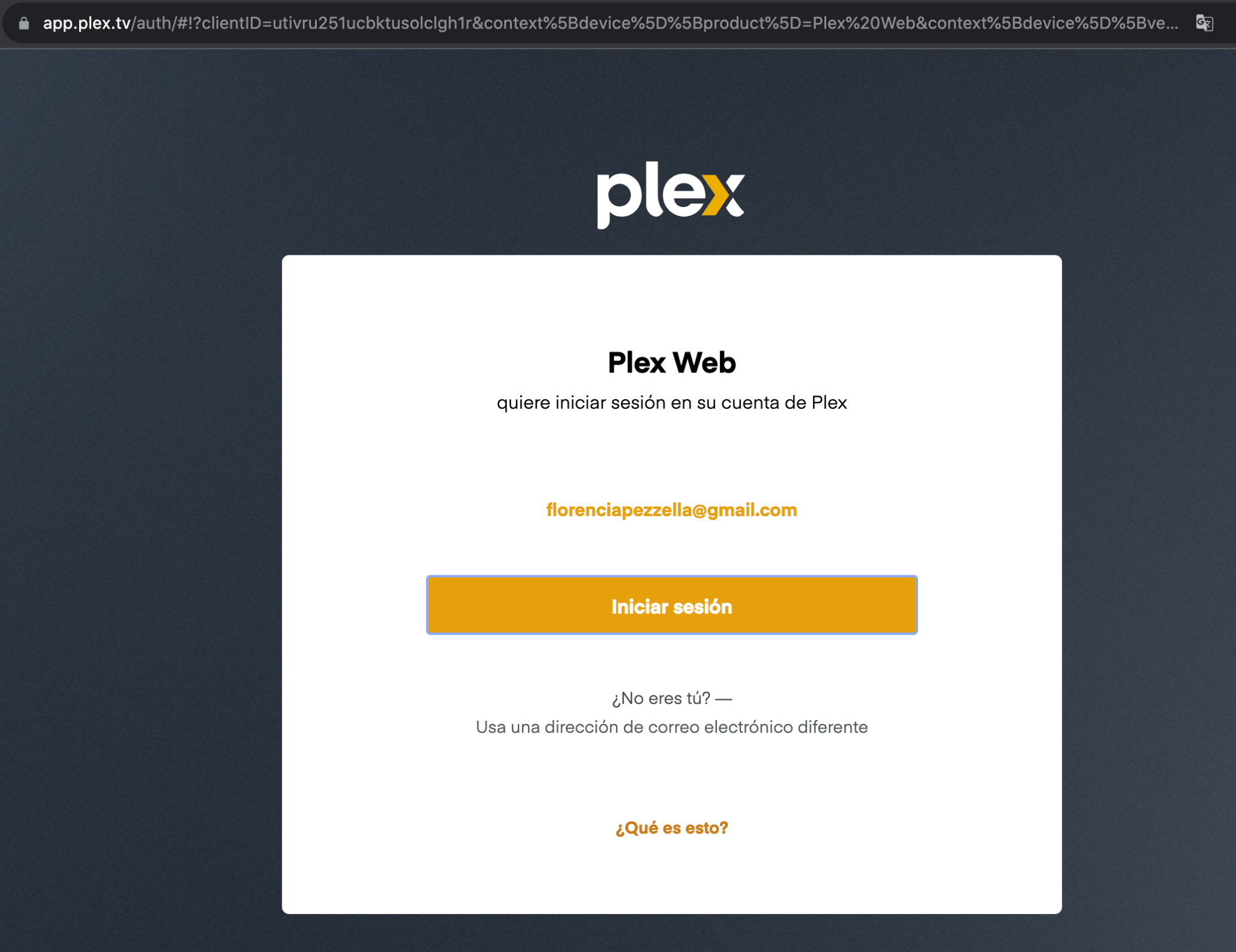
Para visualizar la totalidad de contenedores creados se puede consultar mediante el comando **docker ps -a**



En mi caso, figura solo uno porque es el único contenedor que tengo creado.

Para acceder a Plex a través del contenedor de Docker se ejecuta mediante el navegador web indicando la url: **localhost:32400/web**





**URL DEL REPOSITORIO DE DOCKER:**

**https://hub.docker.com/repository/docker/florpezz/plex\_on\_docker/general**

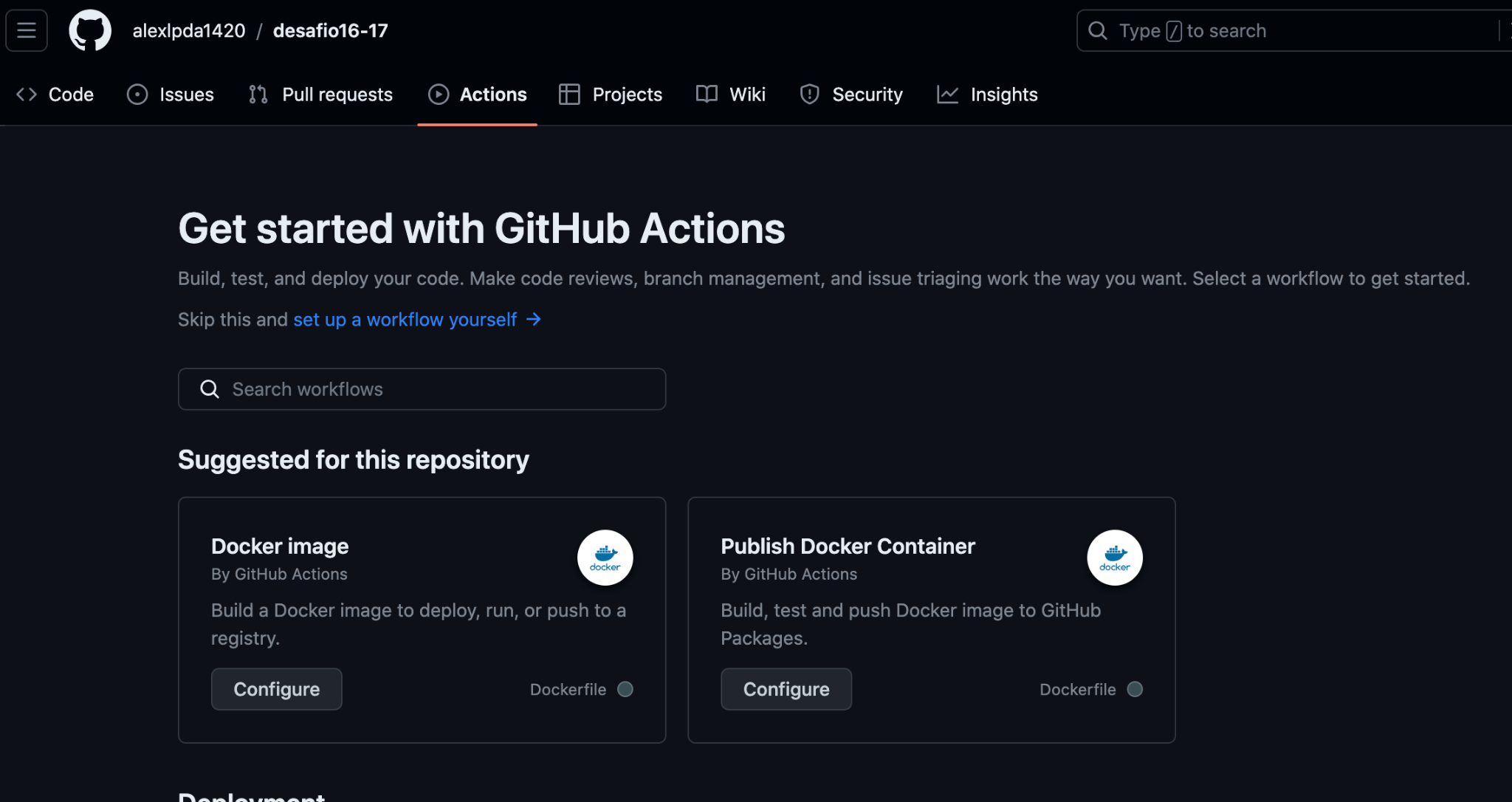
**Parte 3: Automatización de Dockerfile**

1) Deberán automatizar la creación del Dockerfile/Docker compose utilizando alguna herramienta de CICD a elección, el objetivo es que cualquier cambio realizado en el código (ya sea en el Dockerfile/Dockercompose o cambios en el código de la aplicación) active un pipeline que les ejecute un docker build y suba esa imagen a Dockerhub, para esto pueden utilizar Jenkins, Gitlab CI/CD, GitHub Actions, etc. La recomendación es utilizar la herramienta que venga con la herramienta de repositorios que hayan decidido utilizar (por ejemplo, GitHub Actions).

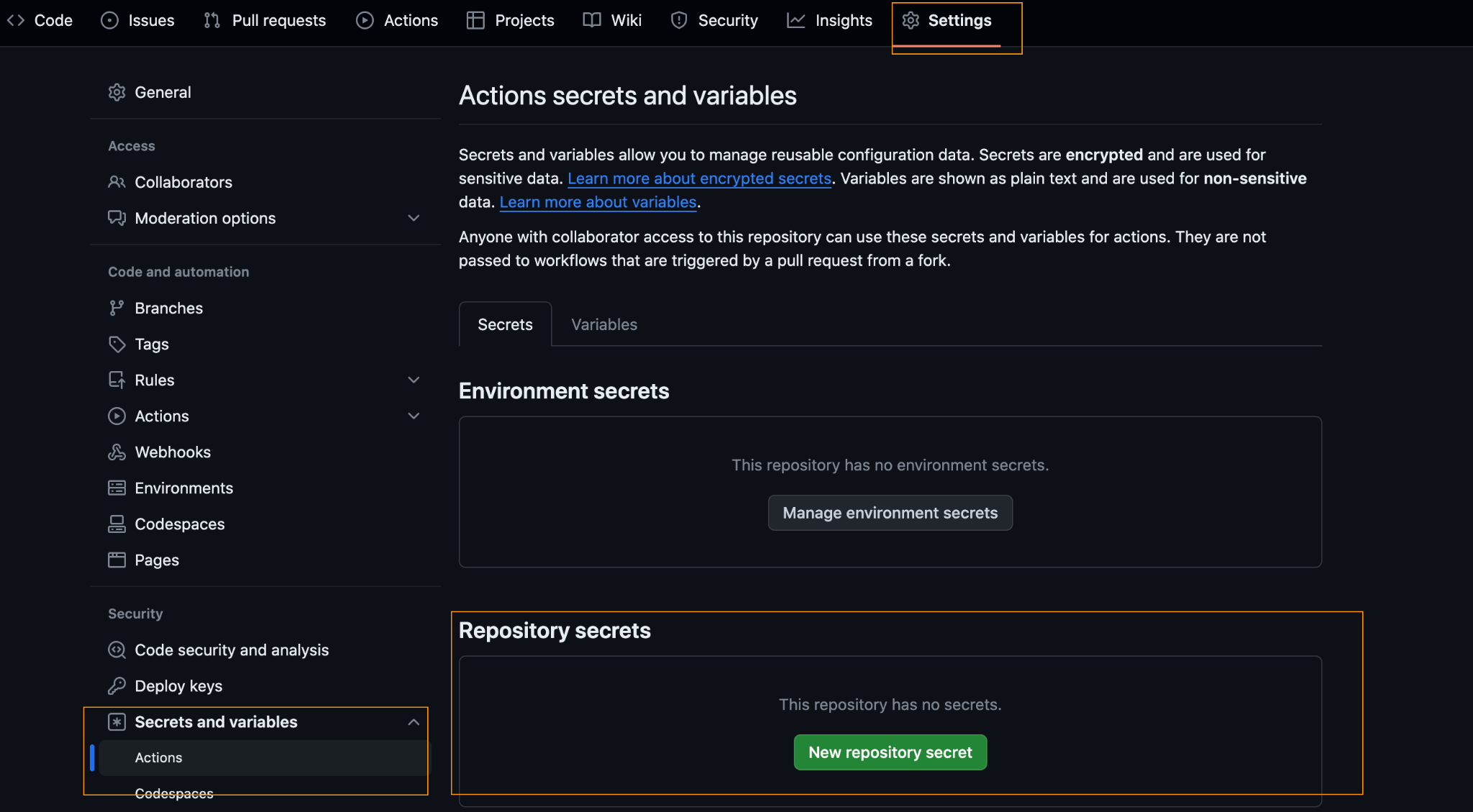
2) Como parte de esta automatización, tendrán que si o si incluir algún tipo de linter, revisión de vulnerabilidades y algún tipo de testeo básico de la aplicación de algún tipo.

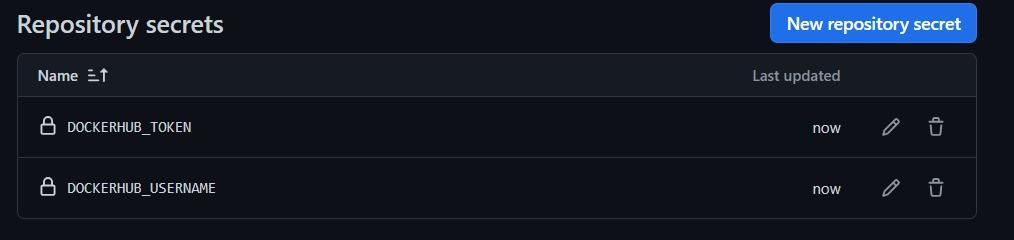
**CREACIÓN DEL WORKFLOW: ci.yaml**

La automatización de la creación del Dockerfile la realizamos con con GitHub Actions.

****

Se almacenan las credenciales de Dockerhub dentro del repositorio:





Se crea el archivo llamado ci.yaml con las configuraciones que se ocuparán de activar el pipeline para ejecutar el docker build y subir esa imagen a Dockerhub.

Se definen los pasos a ejecutar:

Workflow trigger: Se activa el workflow cuando se efectúa un push en la rama main. Y si hay cambios en dentro del directorio de Docker.

Jobs ‘build’: se ejecuta dentro del entorno de Ubuntu, versión latest.

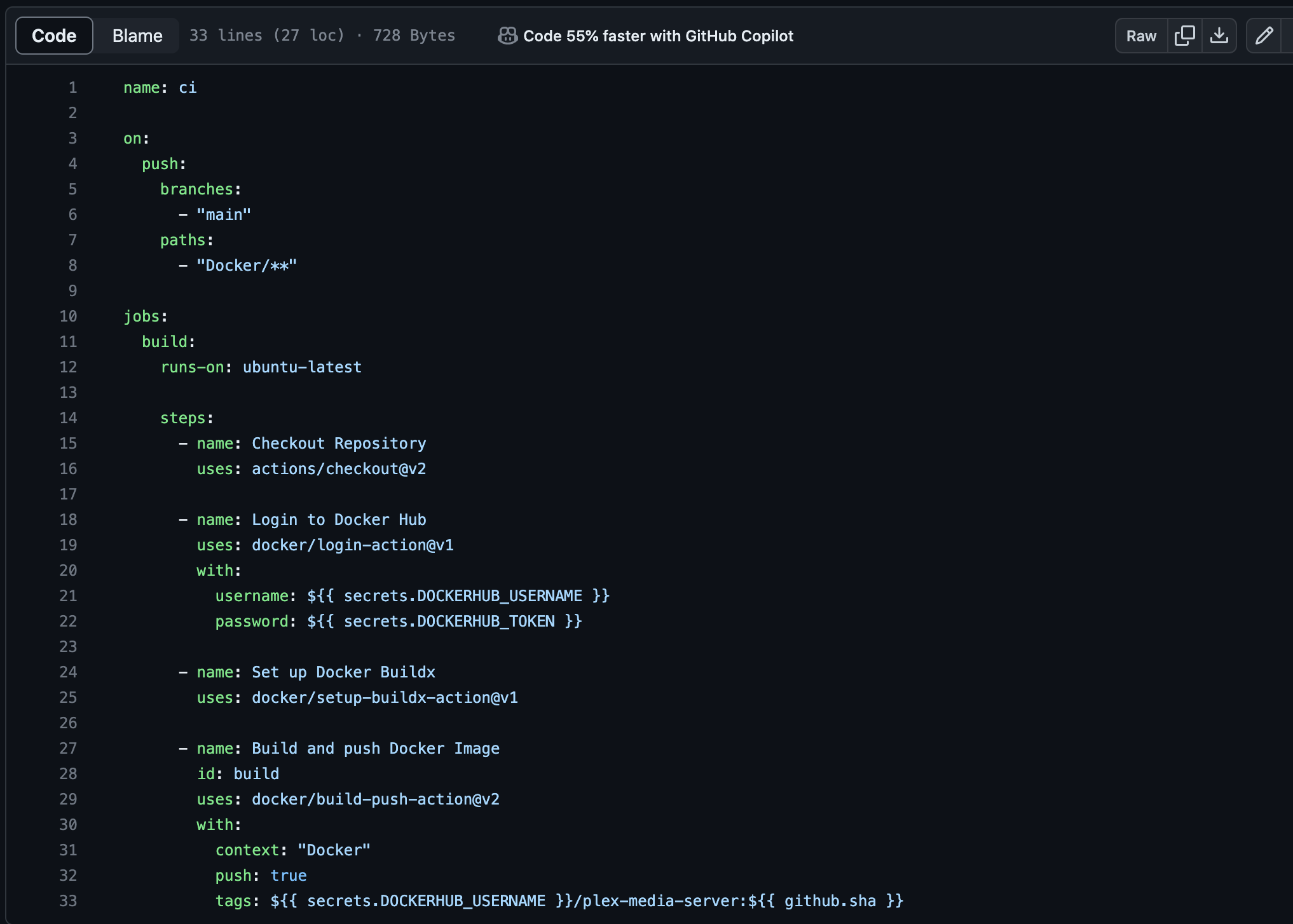
Steps:

Checkout: Descarga el repositorio para que puedan ejecutarse los pasos siguientes.

Login to Docker Hub: inicia sesión.

Set up Docker Buildx: prepara para el proceso de construcción.

Build and push Docker Image: construye y sube la imagen a Docker Hub.

****

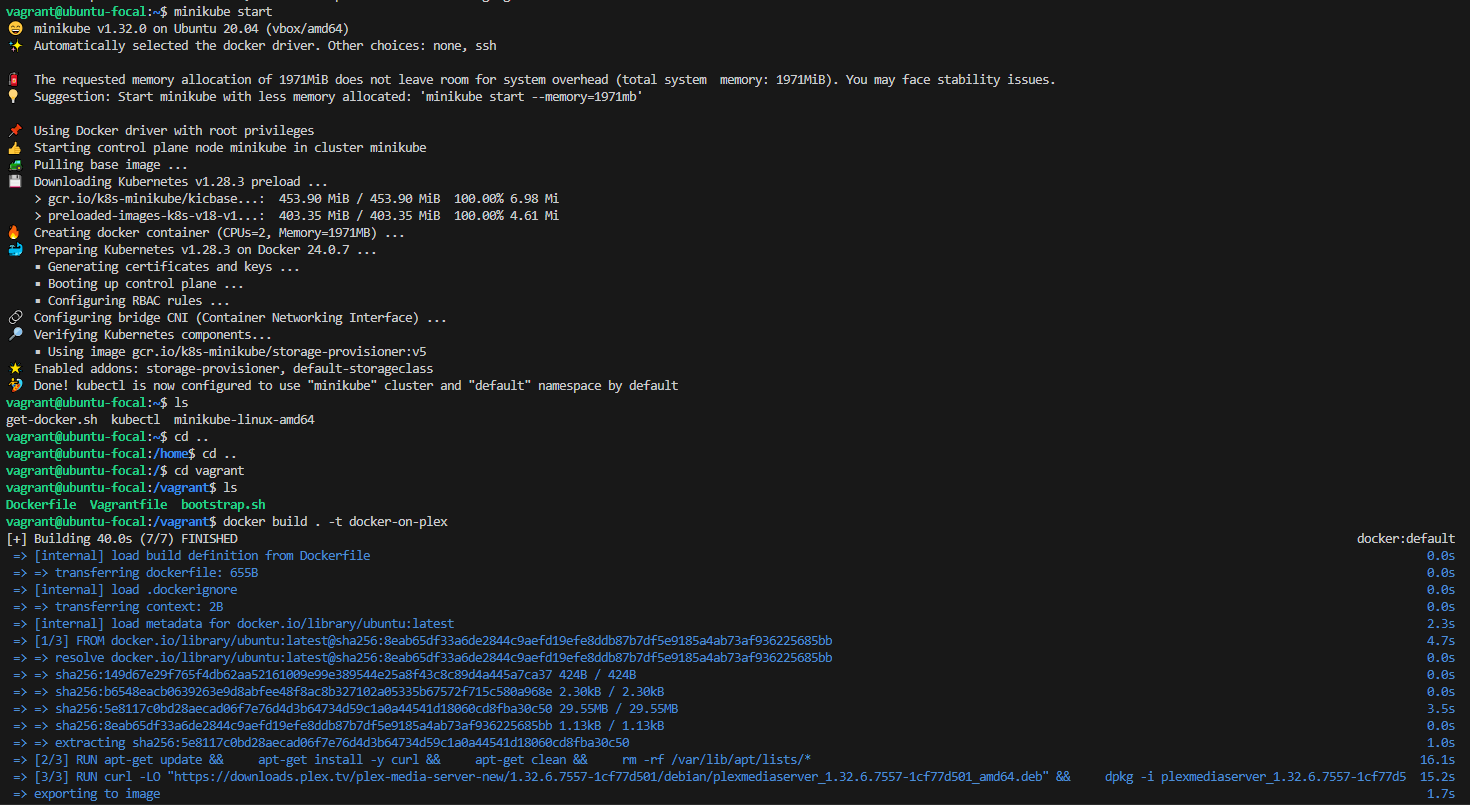
**PARTE 4: KUBERNETES**

1) Ya habiendo contenerizado las distintas partes de la aplicación y automatizado la construcción de la imagen, toca el turno de convertir estos contenedores en recursos de Kubernetes. Los mismos deberán ser deployments + otros recursos adicionales (secretos, volúmenes, configmaps, etc), la configuración del deployment debe permitir que siempre que haya una nueva versión, se cree 1 pod del nuevo replicaset antes de eliminar un pod del replicaset anterior, esto garantizara que solamente una vez que el pod de la nueva versión esté funcionando, se remueva un pod de la versión anterior.

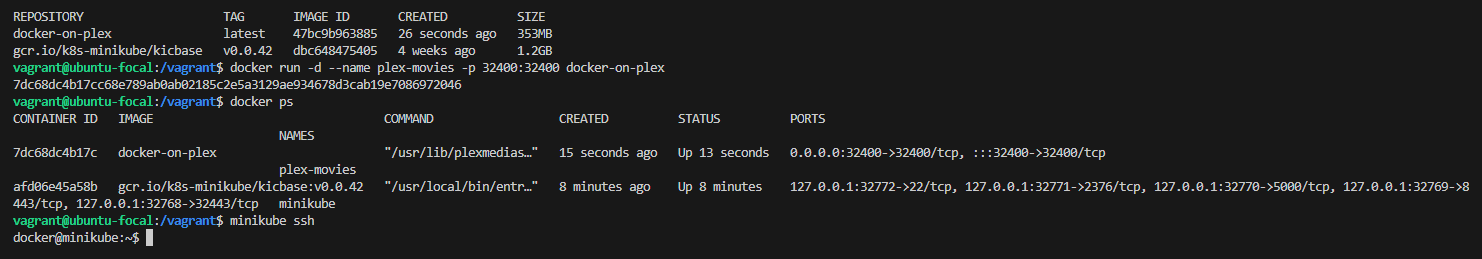
2) Para exponer la aplicación y poder acceder a ella podremos usar el port-forward de kubectl junto al puente de vagrant ssh, aunque pueden buscar otras maneras de exponer la aplicación en caso de requerirlo.

Para convertir los contenedores en recursos de kubernetes debemos instalar minikube en nuestro equipo

Una vez instalado lo iniciamos

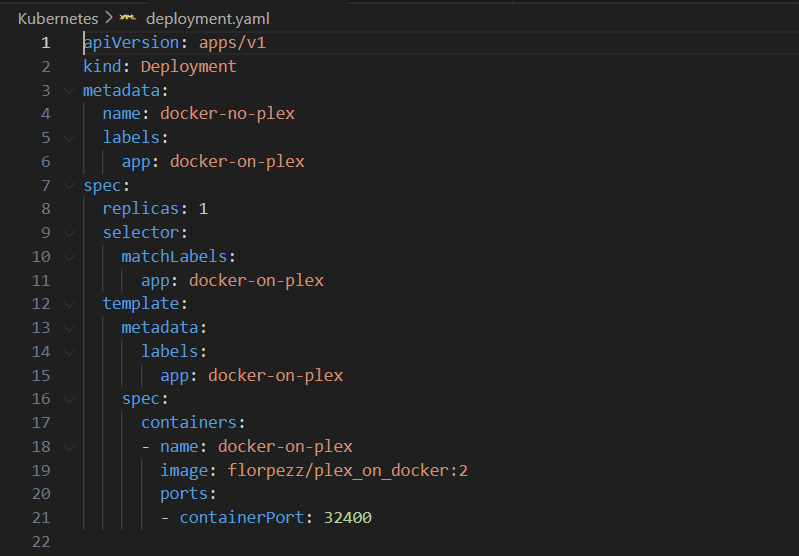


Accedemos a Minikube por ssh



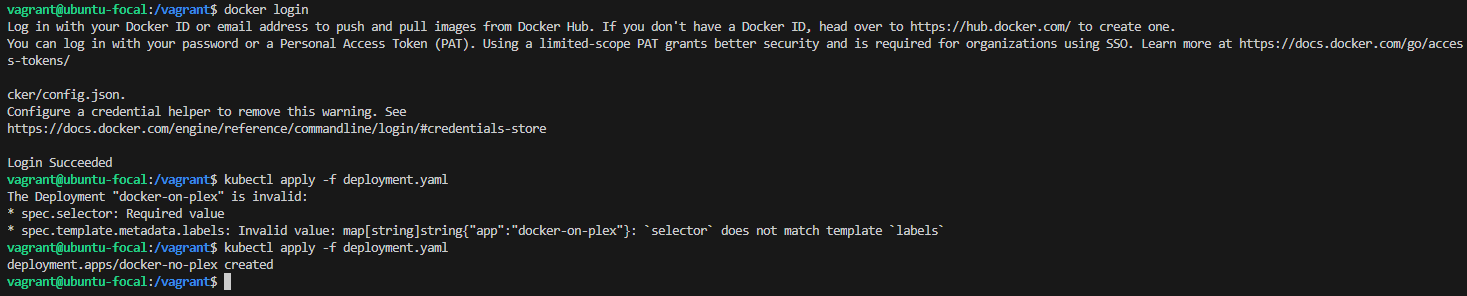
Una vez que inicializamos Minikube procederemos a crear los recursos de kubernetes

Como primer paso debemos crear un archivo YAML denominado deployment.yaml dentro del directorio de nuestro equipo para luego configurarlo con el Deployment.



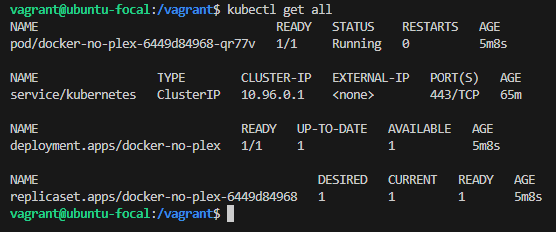
Para aplicar la configuración del Deployment al clúster debemos ejecutar el siguiente comando

kubectl apply -f deployment.yaml



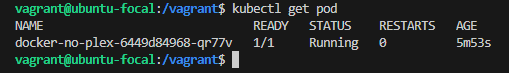
Para obtener todos los recursos en el clúster de Kubernetes ejecutamos

kubectl get all



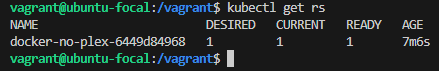
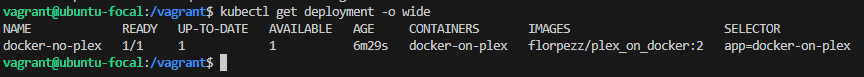
Para verificar el pod

kubectl get pod

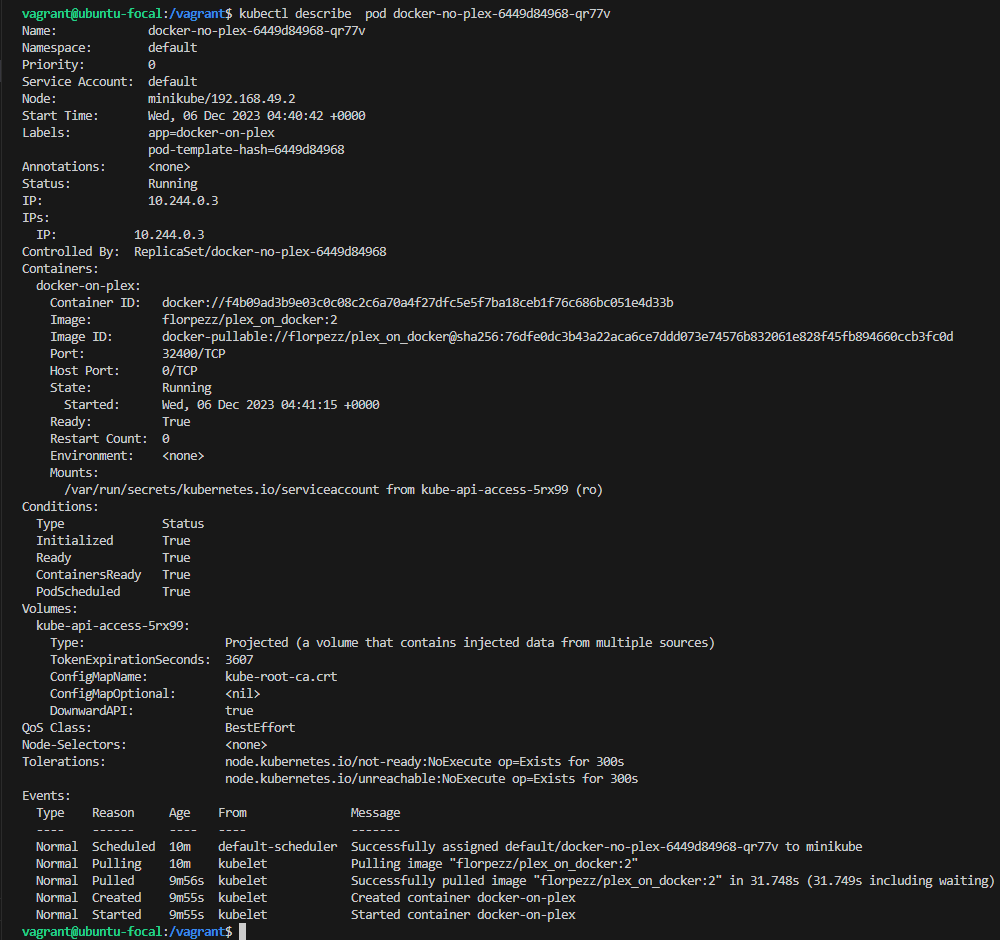


Para ver los detalles de los deployments y replicasets

kubectl get deployments –o wide y kubectl get rs



Para ver la información extendida de los pods



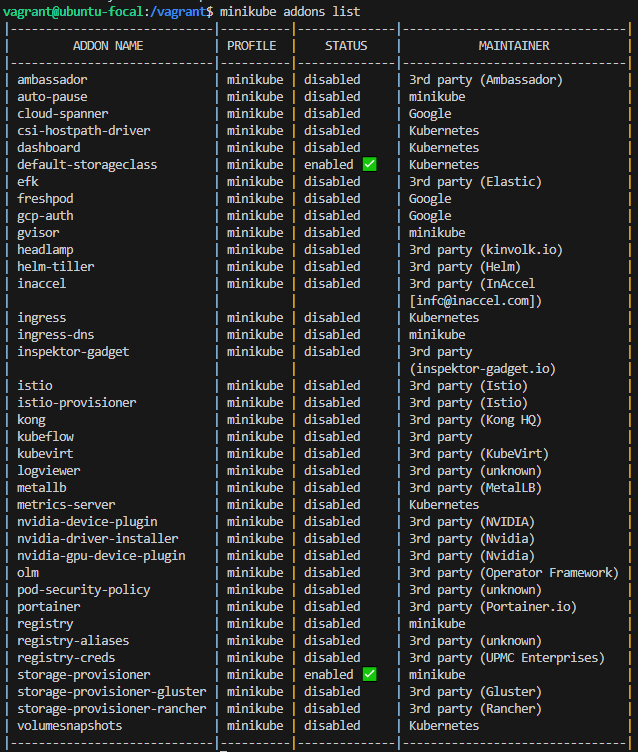
Para ver la información sobre los nodos del clúster

kubectl get nodes



información adicional

minikube addons list



**PARTE 5: Helm y ArgoCD**

1) La primer etapa consiste en convertir los distintos recursos que creamos en

Kubernetes para nuestra aplicación en un paquete de Helm, no solo tendremos que

convertirlo sino que también tendremos que convertir estos recursos en templates

junto a sus variables para poder realizar modificaciones y reutilizar los recursos

definidos en el template.

2) Una vez ya convertida la aplicación en un template de Kubernetes, tendrán que

levantar una instancia de ArgoCD, conectarla a su cluster de Kubernetes y:

a) Levantar una aplicación que apunte a su aplicación convertida en un chart de

Helm, donde le configuraran los valores y tendrán que configurar la

aplicación de Argo para que tenga el Self healing habilitado como así

también el autosync

b) Levantar una aplicación que apunte a su aplicación en recursos de

Kubernetes normales (sin Helm), con sync manual y sin self healing, tengan

en cuenta que para que ambas aplicaciones puedan convivir tendrán que ser

desplegadas en distintos namespaces para evitar repetición de nombres.

**Entregable**

Los entregables serán almacenados en la carpeta compartida que tienen en drive con el

formato (<carpeta con su nombre>/<Fase>/<módulo>/archivo). Por poner un ejemplo, el

instructivo se almacenaría en la carpeta compartida con el nombre Zdenko Hraste, en una

carpeta llamada Fase 3 que dentro tendrá otra carpeta llamada Módulo 16-17 y que dentro

almacenare todos los archivos relevantes a este desafío.

De la misma forma que en el desafío anterior, tendrán que agregar los archivos a su carpeta

en drive dentro de la carpeta Módulo 16-17 dentro de la Fase 3.

Se esperan los archivos:

● Instructivo

● Link al repositorio de código (Documentado en el instructivo)

● Link al repositorio de la imagen (Documentado en el instructivo)

● Una carpeta con una copia del repositorio de código (código-M16-17)

El repositorio debera seguir una estructura similar a las partes del desafio, pueden utilizar

como referencia el siguiente repositorio:

https://github.com/zdenkotraste/workshop-nerdearla/tree/main

Si bien en el instructivo documentaran todo, es importante que trabajen el readme del

repositorio (incluso pueden agregar un readme por carpeta con mayor información para

probar cada parte del mismo). El objetivo es que una persona X pueda agarrar su

repositorio y probarlo de punta a punta (es decir, probar todo en conjunto y también poder

probar cada una de las partes por separado).

Recuerden seguir las instrucciones al pie de la letra para los entregables.