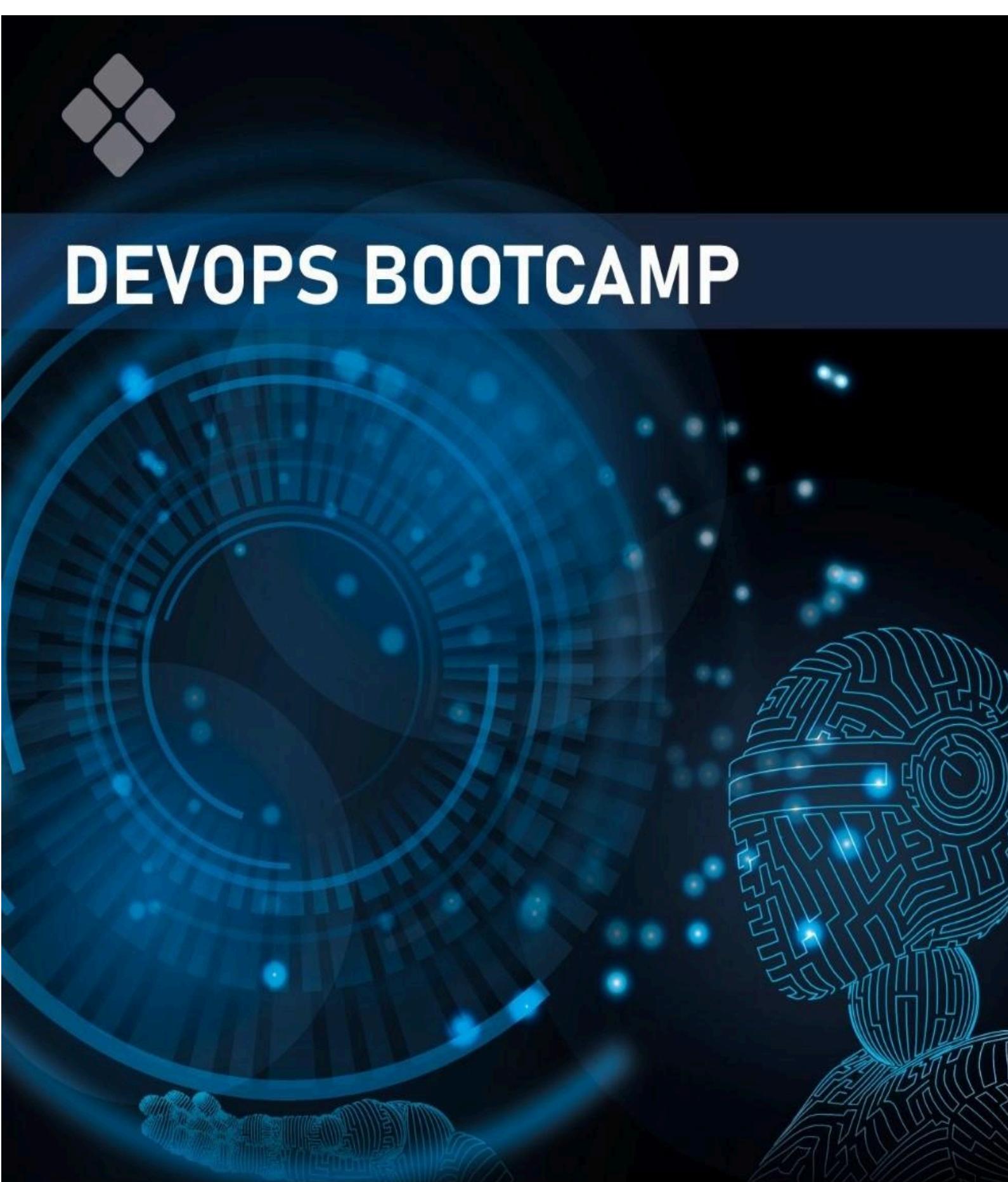




DEVOPS BOOTCAMP



DESAFIOS 16/17

Tipo de documento	INSTRUCTIVO
No. Documento*	001
Última revisión:	07/12/2023

Documento Interno bootcamp	SI	Comentarios:
Clasificación del documento.	INTERNO	

Control de documento

Elaborado por:	Congedo Franco.	Fecha:	01/12/2023
Verificado por:	Gaston Mieulet	Fecha:	29/11/2023
Aprobado por:	Zdenko Hrate	Fecha:	29/11/2023

Índice

1. Vagrant	
1.1. Instalación de Vagrant y Vagrant init	4
1.2. Configuración de Vagrantfile y Vagrant up	5
1.3. Modificación del Vagrantfile y Creación del bootstrap.sh	6
1.4. Agregado de herramientas al bootstrap.sh	7
1.5. Vagrant provision y verificación de instalaciones	8
2. Docker	
2.1. Creación de directorios	
2.2. Configuración del Dockerfile y compose	9
3. CI/CD github actions	10
3.1. Subimos archivos a repositorio y agregamos ci/cd	
3.2. Agregamos secretos al repositorio	11
3.3. Configuración de main.yaml	12
3.4. Ejecución del ci/cd	13/14
3.5. Verificación de imagen en docker hub y testeo de la misma	15
	16/17
4. Kubernetes	
4.1. Iniciamos minikube y creación de directorio	
4.2. Creación de archivos namespace y deployment	18
4.3. Ejecución namespace y deployment	19
4.4. Test utilizando Port-forward	20
	21
5. Helm	
5.1. Creación de chart (paquete predefinido)	22
5.2. Configuración de deployment.yaml	23
5.3. Configuración de values.yaml	24/25/26
5.4. Creación del cluster (con valores por defecto)	27
5.5. Verificación y creación de cluster (Editando parámetros)	28/29/30
6. ArgoCD	
6.1. Instalación de ArgoCD en el cluster de Minikube y configuración del túnel para ingresar a la interfaz gráfica.	31
6.2. Accedemos a la interfaz gráfica y nos logueamos	32
6.3. Creación de la aplicación a través de interfaz gráfica	33
6.4. Configuración de la aplicación (values.yaml)	34/35
6.5. Capturas de aplicación creada y verificación por terminal	36
6.6. Creación de aplicación por terminal	37
6.7. Aplicación sin desplegar y proceso de despliegue y verificación de la misma	38/39/40
7. Links	40

- **Vagrant**

Entorno de Vagrant

Se creará el entorno de Vagrant utilizando el Box Focal64 compuesto por una imagen Linux Ubuntu

Desde la web <https://app.vagrantup.com/ubuntu/boxes/focal64> se obtiene la información para usar el box

Para utilizar Vagrant, es necesario instalarlo en el sistema operativo a utilizar desde la web <https://developer.hashicorp.com/vagrant/downloads>

Primero, creamos una carpeta en el escritorio. Luego abrimos Visual Studio Code (VSC) y abrimos la carpeta recién creada. Por último abrimos una nueva terminal.

Y creamos una subcarpeta Vagrant

Adjunto captura de terminal:

```
● PS E:\Escritorio\desafio-16-17> cd Vagrant
○ PS E:\Escritorio\desafio-16-17\Vagrant> █
```

Luego hacemos un **vagrant init** (con esto creamos el Vagrantfile)

Adjunto captura:



The screenshot shows a Visual Studio Code interface with a terminal window open. The terminal shows the following command sequence:

```
PS E:\Escritorio\desafio-16-17> cd Vagrant
PS E:\Escritorio\desafio-16-17\Vagrant> Vagrant init
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
○ PS E:\Escritorio\desafio-16-17\Vagrant> █
```

Una vez creado el **Vagrantfile**, lo configuramos agregando las siguientes líneas:

Primera línea configuramos el nombre de la vm

Segunda línea le decimos que box vamos a utilizar

```
config.vm.define "desafio" do |desafio|
  desafio.vm.box = "ubuntu/focal64"
end
```

Modificamos los valores de memoria y cpus

```
#   # Customize the amount of memory on the VM:
#   vb.memory = "6144"
#   vb.cpus   = 4
# end
```

Modificamos el nombre

```
config.vm.provider "virtualbox" do |vb|
#   # Display the VirtualBox GUI when booting the machine
#   vb.gui = true
  vb.name = "desafio16-17"
```

Descomentamos esta línea para acceder al contenedor desde '**localhost:8080**'

```
# Create a forwarded port mapping which allows access to a specific port
# within the machine from a port on the host machine. In the example below,
# accessing "localhost:8080" will access port 80 on the guest machine.
# NOTE: This will enable public access to the opened port
config.vm.network "forwarded_port", guest: 80, host: 8080
```

Ejecutamos un **vagrant up**

Adjunto captura:

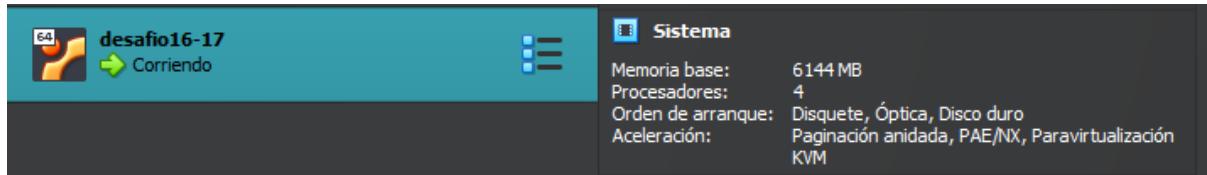
```
PS E:\Escritorio\desafio-16-17\Vagrant> vagrant up
Bringing machine 'desafio' up with 'virtualbox' provider...
==> desafio: Importing base box 'ubuntu/focal64'...
==> desafio: Matching MAC address for NAT networking...
==> desafio: Checking if box 'ubuntu/focal64' version '20230922.0.0' is up to date...
==> desafio: Setting the name of the VM: desafio16-17
==> desafio: Clearing any previously set network interfaces...
```

Ejecutamos un **vagrant status**

- PS E:\Escritorio\desafio-16-17\Vagrant> **vagrant status**
Current machine states:

desafio	running (virtualbox)
----------------	-----------------------------

También verificamos en el virtual box las características de la vm (6gb y 4 núcleos)



Modificamos en el **vagrantfile**, el shell para utilizar un script para la instalación de las distintas herramientas que vamos a utilizar a lo largo del desafío.

```
# Enable provisioning with a shell script. Additional provisioners such as
# Ansible, Chef, Docker, Puppet and Salt are also available. Please see the
# documentation for more information about their specific syntax and use.
config.vm.provision "shell", path: "bootstrap.sh"
```

En la misma carpeta creamos un **bootstrap.sh** con los siguientes comandos:

```
1  #!/bin/bash
2
3  # Update packages
4  sudo apt-get update -y
5
6
7  # Docker Install
8  curl -fsSL https://get.docker.com -o get-docker.sh
9  sudo sh ./get-docker.sh
10
11 # Use Docker without sudo
12 sudo groupadd docker
13 sudo usermod -aG docker vagrant
14 newgrp docker
```

Para la etapa de **Kubernetes** vamos a necesitar Instalar **kubectl** y **minikube**. Entonces agregamos a nuestro bootstrap.sh los siguientes comandos.

```
#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
```

Para la etapa de **Helm** vamos a instalar dicha herramienta. Entonces agregamos a nuestro bootstrap.sh los siguientes comandos.

```
#Install Helm
curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null
sudo apt-get install apt-transport-https -y
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list
sudo apt-get update
sudo apt-get install helm
```

Para la etapa de **argoCD** vamos a instalar dicha herramienta. Entonces agregamos a nuestro bootstrap.sh los siguientes comandos.

```
#Install ArgoCD CLI
curl -sSL -o argocd-linux-amd64 https://github.com/argoproj/argo-cd/releases/latest/download/argocd-linux-amd64
sudo install -m 555 argocd-linux-amd64 /usr/local/bin/argocd
rm argocd-linux-amd64
```

Ejecutamos un **vagrant validate** y un **vagrant provision** (para validar los cambios y realizar instalacion)

```
● PS E:\Escritorio\desafio-16-17\Vagrant> vagrant validate
  Vagrantfile validated successfully.
○ PS E:\Escritorio\desafio-16-17\Vagrant> vagrant provision
  ==> desafio: Running provisioner: shell...
```

Nos volvemos a conectar con un **vagrant ssh**

```
❖ PS E:\Escritorio\desafio-16-17\Vagrant> vagrant ssh
  Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-163-generic x86_64)

    * Documentation:  https://help.ubuntu.com
    * Management:     https://landscape.canonical.com
    * Support:        https://ubuntu.com/advantage
```

Verificamos las instalaciones haciendo un **docker --version**

```
vagrant@ubuntu-focal:~$ docker --version
Docker version 24.0.7, build afdd53b
```

Ahora verificamos haciendo un **kubectl version --client**

```
vagrant@ubuntu-focal:~$ kubectl version --client
Client Version: v1.28.4
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
```

Luego verificamos haciendo un **minikube version**

```
vagrant@ubuntu-focal:~$ minikube version
minikube version: v1.32.0
commit: 8220a6eb95f0a4d75f7f2d7b14cef975f050512d
```

Verificamos haciendo un **helm version**

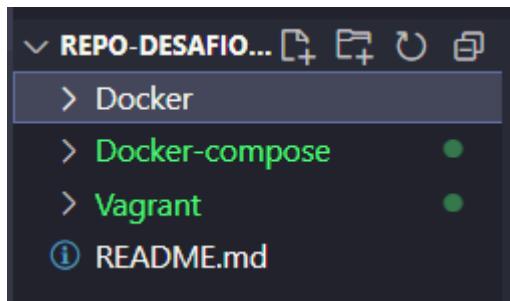
```
vagrant@ubuntu-focal:~$ helm version
version.BuildInfo{Version:"v3.13.2", GitCommit:"2a2fb3b98829f1e0be6fb18af2f6599e0f4e8243", GitTreeState:"clean", GoVersion:"go1.20.10"}
```

Verificamos haciendo un **argocd version**

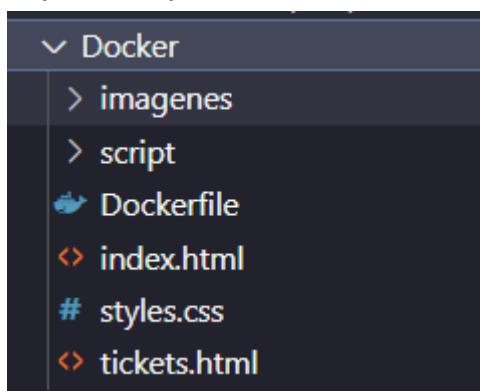
```
vagrant@ubuntu-focal:~$ argocd version
argocd: v2.9.3+6eba5be
  BuildDate: 2023-12-01T23:24:09Z
  GitCommit: 6eba5be864b7e031871ed7698f5233336dfe75c7
```

- **Docker**

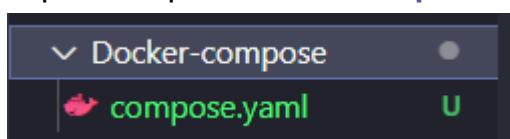
Luego vamos a crear dos directorios, uno llamado **Docker** y otro llamado **Docker-compose** en el VSC donde en la primera vamos a colocar los archivos de la página web y el **Dockerfile** y en la segunda el **Docker-compose**.



Captura carpeta **Docker**:



Captura carpeta **Docker-compose**:



Realizamos la siguiente configuración en el **Dockerfile**:
'From httpd:2.4' (utilizamos la versión 2.4 de apache2)
'COPY ./ /usr/local/apache/htdocs' (copiamos el contenido de la página web al directorio de documentos de apache2)
'EXPOSE 80' (configuramos donde se va a exponer el contenedor, en este caso puerto 80)

```
1 # Utiliza una imagen base de Apache2
2 FROM httpd:2.4
3
4 # Copia solo el contenido de la carpeta "Trabajo-practico-integrador-JS-Codo-a-Codo" al directorio de documentos de Apache2 en el contenedor
5 COPY ./ /usr/local/apache2/htdocs
6
7 # Expone el puerto 80 en el que Apache2 escuchará
8 EXPOSE 80
```

Luego configuramos el **compose.yaml**

Usamos la versión **'3.7 de Docker Compose'**

Nombre de servicio **'mi-web'**

Especificamos la imagen de docker y versión de la misma

Configuramos el nombre del contenedor

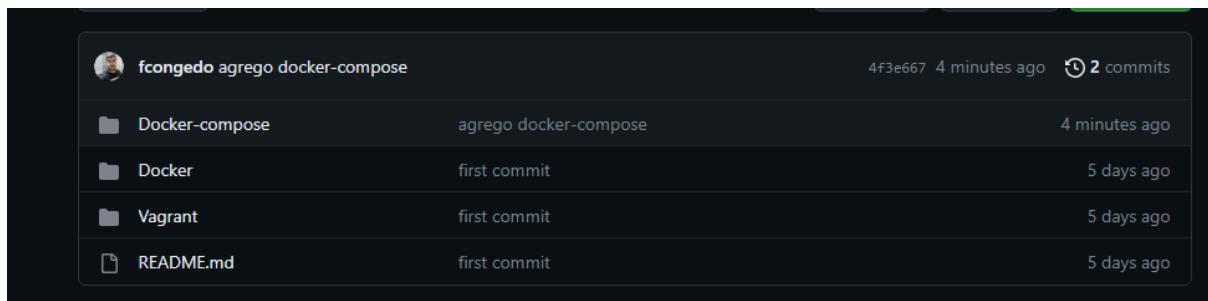
y por último asignamos los puertos entre la máquina host y el contenedor

Adjunto captura:

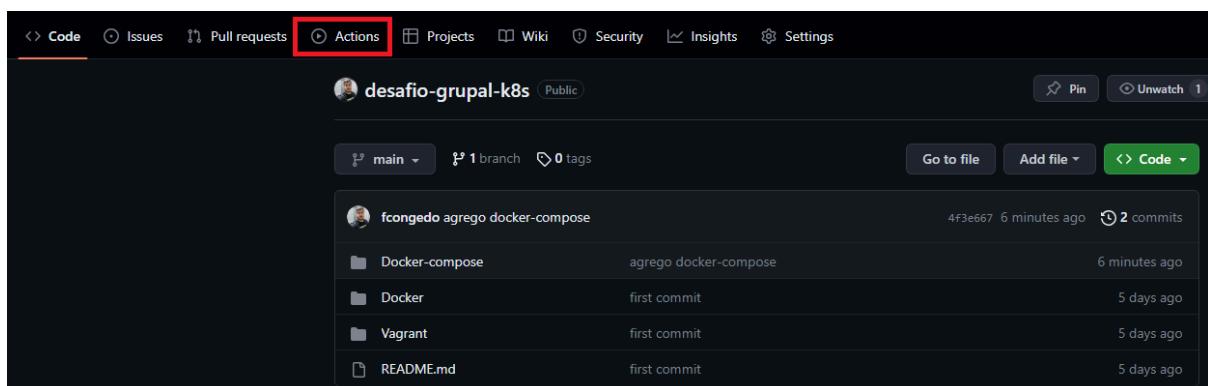
```
Docker-compose > 📄 compose.yaml
1   version: '3.7'
2   services:
3     mi-web:
4       image: fcongedo/mi-web-apache2:1.0
5       container_name: mi-web-compose
6       ports:
7         - "8082:80"
```

- **CI/CD**

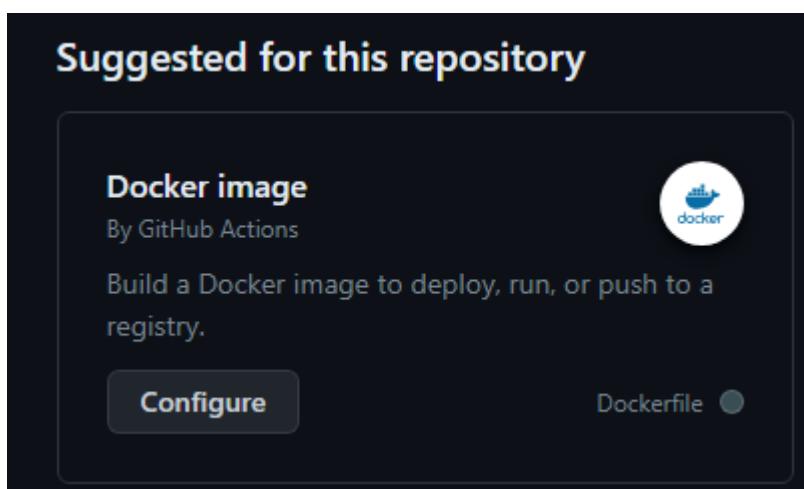
Primero debemos subir los archivos que creamos hasta ahora a un repositorio de github.



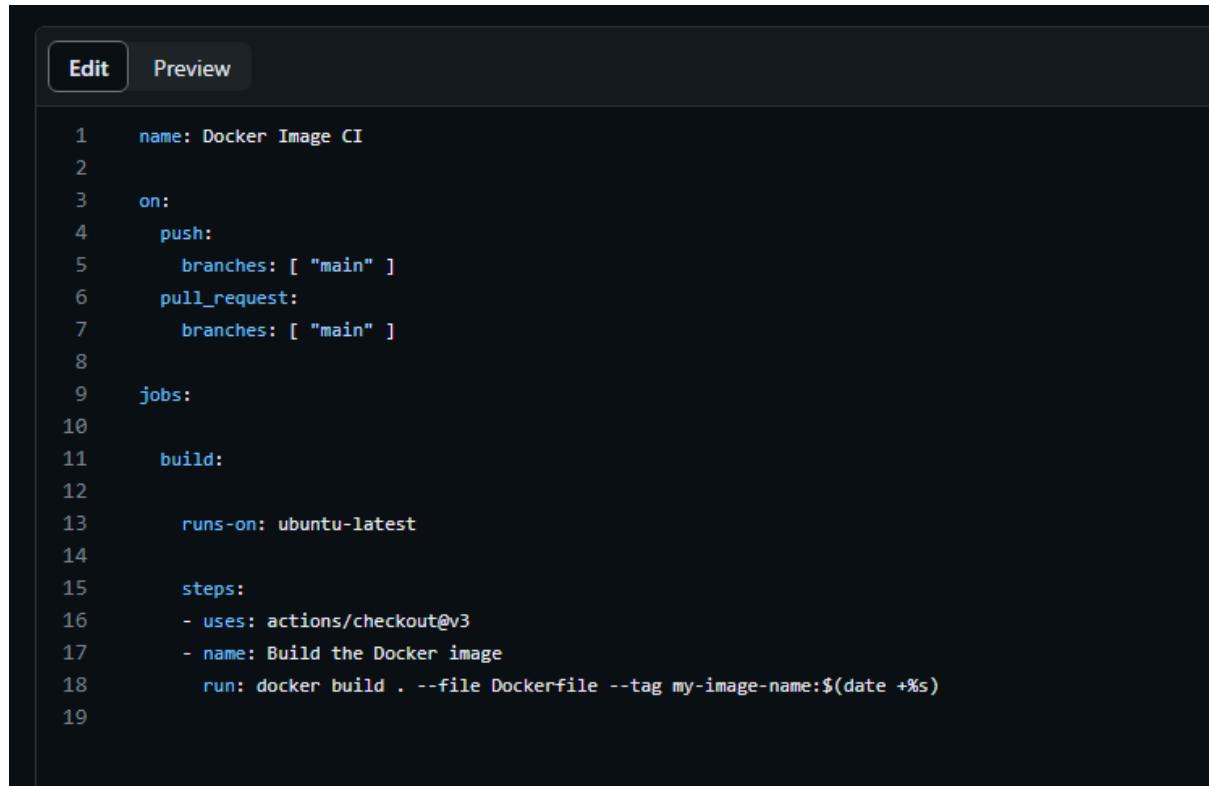
Luego vamos a nuestro repositorio de código y hacemos click en **actions**



Luego elegimos la opción Docker imagen y hacemos click en configure



Nos aparece la siguiente ventana con una base del código de ci/cd

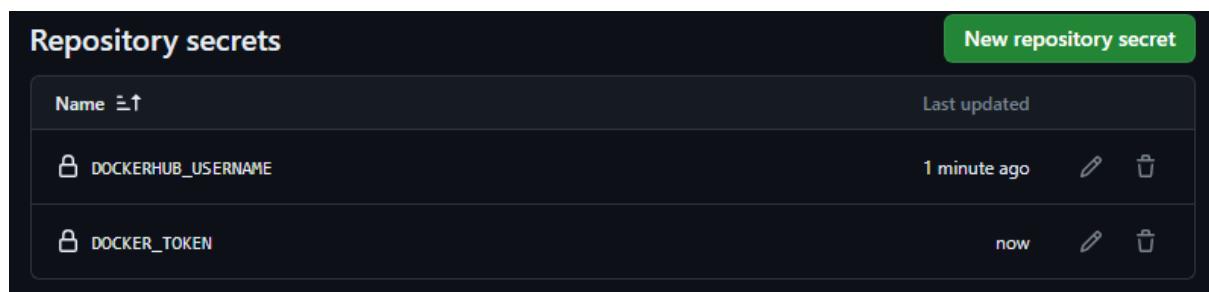


```
1 name: Docker Image CI
2
3 on:
4   push:
5     branches: [ "main" ]
6   pull_request:
7     branches: [ "main" ]
8
9 jobs:
10
11   build:
12
13     runs-on: ubuntu-latest
14
15   steps:
16     - uses: actions/checkout@v3
17     - name: Build the Docker image
18       run: docker build . --file Dockerfile --tag my-image-name:$(date +%s)
19
```

Debemos crear los secretos para el usuario y token de docker hub

Para eso nos dirigimos a nuestro repositorio, luego a **settings**, luego a **secrets and variables** y dentro de esa opción a **Actions**

Por último hacemos click en **new repository secrets** para agregar los secretos.



Name	Last updated	Action
DOCKERHUB_USERNAME	1 minute ago	
DOCKER_TOKEN	now	

Ahora configuraremos el **main.yml** de la siguiente manera

```
name: ci

on:
  push:
    branches:
      - "main"
    paths:
      - "Docker/**"

jobs:
  build-and-scan:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Run Hadolint
        uses: hadolint/hadolint-action@v3.1.0
        with:
          dockerfile: Docker/Dockerfile

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Build Docker image
        run: docker build -t ${ secrets.DOCKER_USERNAME }/mi-web-apache2:${{ github.run_number }} ./Docker

      - name: Scan Docker image with Trivy
        uses: aquasecurity/trivy-action@master
        with:
          image-ref: ${ secrets.DOCKER_USERNAME }/mi-web-apache2:${{ github.run_number }}
          #exit-code-on-severity: "HIGH"

      - name: Login to DockerHub
        uses: docker/login-action@v3
        with:
          username: ${ secrets.DOCKER_USERNAME }
          password: ${ secrets.DOCKERHUB_TOKEN }

      - name: Push Docker image
```

```

        run: docker push ${secrets.DOCKER_USERNAME}
} }/mi-web-apache2:${github.run_number }

      - name: Test Docker image
        run: |
          docker pull ${secrets.DOCKER_USERNAME } }/mi-web-apache2:${github.run_number }
          docker run -d --name test-container -p 8080:80 ${secrets.DOCKER_USERNAME } }/mi-web-apache2:${github.run_number }
          sleep 10 # Time for the container to start properly

          # Perform a curl to the web application page
          curl_output=$(curl -s -o /dev/null -w '%{http_code}' http://localhost:8080/)
          if [ "$curl_output" == "200" ]; then
            echo "Test passed: Application is accessible"
          else
            echo "Test failed: Application is not accessible or did not return a 200 status code"
            exit 1
          fi

          # Stop and remove the test container
          docker stop test-container && docker rm test-container

          # Remove the Docker image used for testing
          docker image rm ${secrets.DOCKER_USERNAME } }/mi-web-apache2:${github.run_number }
    
```

En este caso, primero clonamos el repositorio para obtener el código fuente. Luego, realizamos un análisis del Dockerfile usando [Hadolint](#). A continuación, creamos la imagen Docker a partir de este código. Posteriormente, analizamos exhaustivamente la imagen Docker recién creada con [Trivy](#) para buscar posibles vulnerabilidades o problemas de seguridad.

Después, llevamos a cabo la autenticación en Docker Hub utilizando los secretos de GitHub Actions y subimos la imagen Docker al registro de Docker Hub. Finalmente, ejecutamos una prueba de validación, donde inicializamos la imagen recién creada y verificamos su accesibilidad mediante una solicitud con curl a localhost:8080. Al finalizar esta prueba, detenemos y eliminamos tanto el contenedor como la imagen utilizados en este proceso de validación.

Una vez configurado, hacemos un cambio en algún archivo dentro de el directorio **Docker** (en este caso en el archivo index.html)

```
<div class="navbar-nav">
  <a class="nav-link active" href="#">La conferencia</a>
  <a class="nav-link " href="#">Prueba Bootcamp4</a>
```

(modifique agregando un '**Prueba bootcamp4**')

Luego en el repositorio vemos que se el **ci/cd** se ejecutó correctamente

prueba ci/cd

ci #8: Commit 3ed649d pushed by fcongedo

build-and-scan
succeeded 3 hours ago in 5s

- > ✓ Set up job
- > ✓ Build hadolint/hadolint-action@v3.1.0
- > ✓ Build aquasecurity/trivy-action@master
- > ✓ Checkout
- > ✓ Run Hadolint
- > ✓ Set up Docker Buildx
- > ✓ Build Docker image
- > ✓ Scan Docker image with Trivy
- > ✓ Login to DockerHub
- > ✓ Push Docker image
- > ✓ Test Docker image
- > ✓ Post Login to DockerHub
- > ✓ Post Set up Docker Buildx
- > ✓ Post Checkout
- > ✓ Complete job

Captura del resultado del análisis con **Trivy**

26																				
27	***/mi-web-apache2:5 (debian 12.2)																			
28	=====																			
29	Total: 208 (UNKNOWN: 0, LOW: 152, MEDIUM: 35, HIGH: 20, CRITICAL: 1)																			
30																				
31	<table border="1"> <thead> <tr> <th>Library</th> <th>Vulnerability</th> <th>Severity</th> <th>Status</th> <th>Installed Version</th> <th>Fixed Version</th> <th>Title</th> </tr> </thead> <tbody> <tr> <td>apt</td> <td>CVE-2011-3374</td> <td>LOW</td> <td>affected</td> <td>2.6.1</td> <td></td> <td>It was found that apt-key in apt, all versions, do not correctly...</td> </tr> </tbody> </table>						Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title	apt	CVE-2011-3374	LOW	affected	2.6.1		It was found that apt-key in apt, all versions, do not correctly...
Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title														
apt	CVE-2011-3374	LOW	affected	2.6.1		It was found that apt-key in apt, all versions, do not correctly...														
32																				
33																				
34																				
35																				
36																				

Verificamos en dockerhub si la imagen está subida.

TAG 8 Last pushed 3 hours ago by fcongedo	DIGEST 8a0866bc4827	OS/ARCH linux/amd64	LAST PULL 3 hours ago	COMPRESSED SIZE 63.96 MB
TAG 5 Last pushed 3 days ago by fcongedo	DIGEST 0734cdcaa6d4b	OS/ARCH linux/amd64	LAST PULL 18 hours ago	COMPRESSED SIZE 63.96 MB
TAG 3 Last pushed 3 days ago by fcongedo	DIGEST debdaaa8d9240	OS/ARCH linux/amd64	LAST PULL 3 days ago	COMPRESSED SIZE 63.96 MB

Por último vamos a verificar el correcto funcionamiento de la imagen

Primero en **VSC** nos conectamos a la maquina de vagrant utilizando el comando ya mencionado **vagrant ssh**

Luego vamos a descargar la imagen utilizando el comando
docker pull nombre_de_usuario_en_dockerhub/portfolio-website:etiqueta

En nuestro caso **docker pull fcongedo/mi-web-apache2:5**
verificó la imagen ejecutando **docker images**

Adjunto captura del proceso:

```
vagrant@ubuntu-focal:~$ docker pull fcongedo/mi-web-apache2:5
5: Pulling from fcongedo/mi-web-apache2
1f7ce2fa46ab: Pull complete
424de2a10000: Pull complete
6d9a0131505f: Pull complete
5728e491734b: Pull complete
20d3235e84ad: Pull complete
8450381a0e52: Pull complete
Digest: sha256:0734cdcaa6d4bd3a648660b59793f8ce43aa65bdaabd32a11437859ef315a27d6
Status: Downloaded newer image for fcongedo/mi-web-apache2:5
docker.io/fcongedo/mi-web-apache2:5
vagrant@ubuntu-focal:~$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
fcongedo/mi-web-apache2  5        6f563347f9ff   19 hours ago  171MB
```

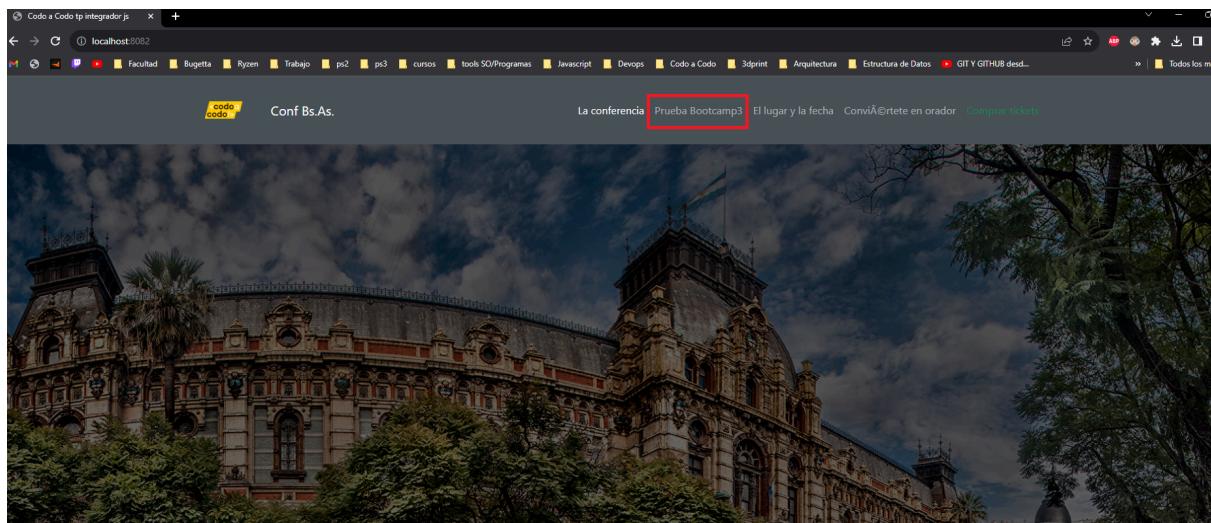
Por último vamos a correr nuestra imagen ejecutando el siguiente comando
docker run -d -p 8082:80 --name website-apache2 fcongedo/mi-web-apache2:5
Luego hacemos un docker ps para ver el contenedor corriendo

Adjunto captura del proceso:

```
vagrant@ubuntu-focal:~$ docker run -d -p 8082:80 --name website-apache2 fcongedo/mi-web-apache2:5
99f7de070630497c020ea5ad0e9544a5b0dad06023e182b140edbc1e691792e0
vagrant@ubuntu-focal:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
99f7de070630 fcongedo/mi-web-apache2:5 "httpd-foreground" 3 seconds ago Up 2 seconds 0.0.0.0:8082->80/tcp, :::8082->80/tcp website-apache2
```

Luego de esto nos conectamos a contenedor del portfolio utilizando el túnel
vagrant ssh -- -L 8082:localhost:8082

Adjunto captura de imagen accediendo de **localhost:8082**



(vemos en el **navbar** que está el cambio que hicimos para ejecutar el ci/cd “**Prueba bootcamp3**”)

● KUBERNETES

En primer medida vamos a conectarnos a la máquina de vagrant ejecutando un **vagrant ssh**

Luego vamos a ejecutar el comando **minikube start** (para desplegar y configurar un cluster de kubernetes en la máquina virtual)

Captura del proceso:

```
vagrant@ubuntu-focal:~$ minikube start
👉 minikube v1.32.0 on Ubuntu 20.04 (vbox/amd64)
👉 Automatically selected the docker driver. Other choices: none, ssh
👉 Using Docker driver with root privileges
👉 Starting control plane node minikube in cluster minikube
👉 Pulling base image ...
💻 Downloading Kubernetes v1.28.3 preload ...
  > preloaded-images-k8s-v18-v1...: 403.35 MiB / 403.35 MiB 100.00% 2.96 Mi
  > gcr.io/k8s-minikube/kicbase...: 453.90 MiB / 453.90 MiB 100.00% 3.07 Mi
🔥 Creating docker container (CPUs=2, Memory=2200MB) ...
🌐 Preparing Kubernetes v1.28.3 on Docker 24.0.7 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
🔗 Configuring bridge CNI (Container Networking Interface) ...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌐 Verifying Kubernetes components...
☀️ Enabled addons: default-storageclass, storage-provisioner
🎉 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Dentro de la máquina con vagrant, en el mismo directorio vagrant creamos una carpeta llamada kubernetes

```
vagrant@ubuntu-focal:/vagrant/kubernetes$ ls
deployment.yaml
```

Primero vamos a crear un **Namespace** (creamos un archivo llamado **ns.yaml** con la siguiente configuración

```
1  apiVersion: v1
2  kind: Namespace
3  metadata:
4    name: prueba-desafiook8s
5    labels:
6      name: prueba-desafiook8s
```

Luego creamos un archivo llamado **deployment.yaml** con la siguiente configuración

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: website-apache2
  namespace: prueba-desafiook8s
  labels:
    app: website-apache2
spec:
  replicas: 3 # El número de réplicas que necesites
  selector:
    matchLabels:
      app: website-apache2
  strategy:
    type: RollingUpdate # Estrategia de actualización
    rollingUpdate:
      maxSurge: 1 # Máximo de pods adicionales durante la actualización
      maxUnavailable: 0 # Máximo de pods no disponibles durante la actualización
  template:
    metadata:
      labels:
        app: website-apache2
    spec:
      containers:
        - name: website
          image: fcongedo/mi-web-apache2:5
          ports:
            - containerPort: 80
```

Primero vamos a crear nuestro namespace. Para luego poder asignarle los pods.

Para eso ejecutamos el comando **kubectl apply -f ns.yaml**

Luego listamos todos los namespaces ejecutando **kubectl get namespaces**

Adjunto captura:

```
vagrant@ubuntu-focal:/vagrant/kubernetes$ kubectl apply -f ns.yaml
namespace/prueba-desafiol8s created
vagrant@ubuntu-focal:/vagrant/kubernetes$ kubectl get namespaces
NAME      STATUS   AGE
default   Active   9h
kube-node-lease   Active   9h
kube-public   Active   9h
kube-system   Active   9h
prueba-desafiol8s   Active   24s
```

Ahora ejecutamos el **deployment.yaml**

Ejecutamos **kubectl apply -f deployment.yaml**

Luego los listamos utilizando **kubectl get pods -n prueba-desafiol8s** (namespace)

Adjunto captura:

```
vagrant@ubuntu-focal:/vagrant/kubernetes$ kubectl apply -f deployment.yaml
deployment.apps/website-apache2 created
vagrant@ubuntu-focal:/vagrant/kubernetes$ kubectl get pods -n prueba-desafiol8s
NAME          READY   STATUS    RESTARTS   AGE
website-apache2-85c4d5689b-hlkx   1/1     Running   0          5s
website-apache2-85c4d5689b-nhlc2   1/1     Running   0          6s
website-apache2-85c4d5689b-zsgp6   1/1     Running   0          5s
```

Ahora para exponer la aplicación y poder acceder a ella podremos usar el **port-forward**

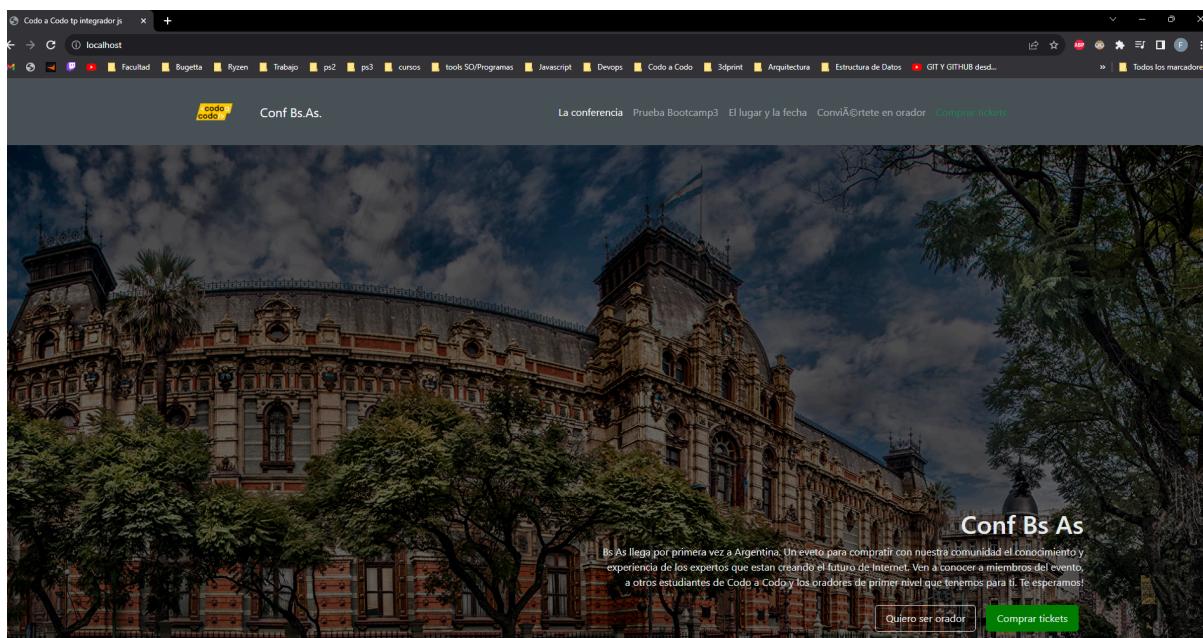
Primero hacemos el port-forward para eso ejecutamos

kubectl port-forward deployment/website-apache2 8080:80 -n prueba-desafiook8s

Luego hacemos el túnel de ssh ejecutando **vagrant ssh -- -L 80:localhost:8080**

Verificamos entrando desde el navegador con un **localhost:80**

Adjunto captura:



Y desde la máquina de vagrant haciendo un **curl localhost:8080**

```
vagrant@ubuntu-focal:~$ curl localhost:8080
<!DOCTYPE html>
<html lang="es">
<head>
  <!-- Meta información del documento -->
  <title>Codo a Codo tp integrador js</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-FwquAoPFOjkml4HiPq3jBDvzsP+ivDlFqjamtJ3z3QkOxW3oO7ZQn=QB83DFGTowi0iMjiWaeVhAn4FJkjQJByhZMI3AhiU" crossorigin="anonymous">
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-/bQdsTh/da6pkI1MST/rWKFNjaCP5gBSY4sEBT380/9RBh9AH40zE0g7Hlq2THRZ"
    crossorigin="anonymous"></script>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <!-- Barra de navegación -->
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <!-- Contenedor de la barra de navegación -->
    <div class="container">
      <!-- Logo de la conferencia -->
      <a class="navbar-brand" href="#">Conf Bs.As.</a>
```

- **HELM**

Primero nos dirigimos a `cd /vagrant` en nuestra máquina virtual y creamos un directorio ejecutando el comando `mkdir helm-website`

Luego hacemos un `cd /helm-website`

Y por ultimo ejecutamos `helm create chart`

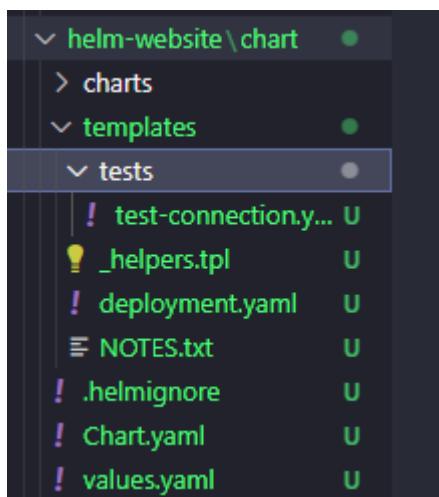
Captura proceso:

```
vagrant@ubuntu-focal:/vagrant$ mkdir helm-website
vagrant@ubuntu-focal:/vagrant$ cd helm-website/
vagrant@ubuntu-focal:/vagrant/helm-website$ helm create chart
```

Esto nos genera un paquete predefinido que contiene una estructura de directorios y archivos que describe cómo desplegar una aplicación en Kubernetes.

Nosotros solamente nos vamos a enfocar en utilizar y modificar el `deployment.yaml` y `values.yaml`

Estos son los archivos que vamos a utilizar (el resto fueron borrados)



Configuración del deployment.yaml

Resaltado con rojo los cambios:

Agregue para pasarle de manera estática en el values.yaml los siguientes parámetros: **namespace**, **replicaCount**(número de réplicas), **repositorio de la imagen**, **tag**, y **port** (puerto en la que se ejecuta)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "chart.fullname" . }}
  namespace: {{ .Values.namespace }}
  labels:
    {{- include "chart.labels" . | nindent 4 }}
spec:
  replicas: {{ .Values.replicaCount }} # El número de réplicas que necesites
  selector:
    matchLabels:
      {{- include "chart.selectorLabels" . | nindent 6 }}
  strategy:
    type: RollingUpdate # Estrategia de actualización
    rollingUpdate:
      maxSurge: 1 # Máximo de pods adicionales durante la actualización
      maxUnavailable: 0 # Máximo de pods no disponibles durante la actualización
  template:
    metadata:
      labels:
        {{- include "chart.selectorLabels" . | nindent 8 }}
    spec:
      containers:
        - name: {{ .Chart.Name }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
          ports:
            - containerPort: {{ .Values.container.port }}
```

Configuración `values.yaml`

Resaltado en **rojo** los valores ingresados estáticamente (valores por defecto)

```
# Default values for helm-website.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

replicaCount: 3

image:
  repository: fcongedo/mi-web-apache2
  pullPolicy: IfNotPresent
  # Overrides the image tag whose default is the chart appVersion.
  tag: "5"

imagePullSecrets: []
nameOverride: ""
fullnameOverride: ""

serviceAccount:
  # Specifies whether a service account should be created
  create: true
  # Automatically mount a ServiceAccount's API credentials?
  automount: true
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the
  fullname template
  name: ""

podAnnotations: {}

namespace: prueba-desafiook8s

container:
  port: 80
```

```
podSecurityContext: {}

# fsGroup: 2000


securityContext: {}
# capabilities:
#   drop:
#     - ALL
# readOnlyRootFilesystem: true
# runAsNonRoot: true
# runAsUser: 1000


service:
type: ClusterIP
port: 80


ingress:
enabled: false
className: ""
annotations: {}
# kubernetes.io/ingress.class: nginx
# kubernetes.io/tls-acme: "true"
hosts:
- host: chart-example.local
  paths:
    - path: /
      pathType: ImplementationSpecific
tls: []
#   - secretName: chart-example-tls
#     hosts:
#       - chart-example.local


resources: {}
# We usually recommend not to specify default resources and to leave
this as a conscious
# choice for the user. This also increases chances charts run on
environments with little
# resources, such as Minikube. If you do want to specify resources,
uncomment the following
# lines, adjust them as necessary, and remove the curly braces after
'resources:'.
# limits:
```

```
#   cpu: 100m
#   memory: 128Mi
# requests:
#   cpu: 100m
#   memory: 128Mi

autoscaling:
  enabled: false
  minReplicas: 1
  maxReplicas: 100
  targetCPUUtilizationPercentage: 80
  # targetMemoryUtilizationPercentage: 80

# Additional volumes on the output Deployment definition.
volumes: []
# - name: foo
#   secret:
#     secretName: mysecret
#     optional: false

# Additional volumeMounts on the output Deployment definition.
volumeMounts: []
# - name: foo
#   mountPath: "/etc/foo"
#   readOnly: true

nodeSelector: {}

tolerations: []

affinity: {}
```

Una vez configurado esto nos situamos en el directorio `/vagrant/helm-website`
 Primero creamos el **namespace** por defecto (en mi caso prueba-desafiook8s)
 Ejecutando `kubectl create namespace prueba-desafiook8s`
 Luego ejecutamos `helm install prueba-2 ./chart`

Captura del proceso:

```
vagrant@ubuntu-focal:/vagrant/helm-website$ kubectl create namespace prueba-desafiook8s
namespace/prueba-desafiook8s created
vagrant@ubuntu-focal:/vagrant/helm-website$ helm install prueba-1 ./chart
Error: INSTALLATION FAILED: cannot re-use a name that is still in use
vagrant@ubuntu-focal:/vagrant/helm-website$ helm install prueba-2 ./chart
NAME: prueba-2
LAST DEPLOYED: Wed Dec 6 01:44:06 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=chart,app.kubernetes.io/instance=prueba-2" -o jsonpath=".items[0].metadata.name")
  export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o jsonpath=".spec.containers[0].ports[0].containerPort")
  echo "Visit http://127.0.0.1:$CONTAINER_PORT to use your application"
  kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT
```

Luego hacemos un `helm status prueba-2 --show-resources`

```
vagrant@ubuntu-focal:/vagrant/helm-website$ helm status prueba-2 --show-resources
NAME: prueba-2
LAST DEPLOYED: Wed Dec 6 01:44:06 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
RESOURCES:
==> v1/Deployment

==> v1/Pod(related)
NAME                      READY   STATUS    RESTARTS   AGE
prueba-2-chart-5c5858869b-4v676  1/1     Running   0          68s
prueba-2-chart-5c5858869b-nwxjx  1/1     Running   0          68s
prueba-2-chart-5c5858869b-wmbrh  1/1     Running   0          68s
```

Por último ejecutamos un **kubectl describe pod prueba-2-chart-5c5858869b-4v676 -n prueba-desafiok8s** (para obtener una descripción detallada del pod)

Con esto verificamos que se creó con los valores por defecto (estáticos)

```
vagrant@ubuntu-focal:/vagrant$ kubectl describe pod prueba-2-chart-5c5858869b-4v676 -n prueba-desafiok8s
Name:           prueba-2-chart-5c5858869b-4v676
Namespace:      prueba-desafiok8s
Priority:       0
Service Account: default
Node:          minikube/192.168.49.2
Start Time:    Wed, 06 Dec 2023 01:44:06 +0000
Labels:         app.kubernetes.io/instance=prueba-2
                app.kubernetes.io/name=chart
                pod-template-hash=5c5858869b
Annotations:   <none>
Status:        Running
IP:            10.244.0.3
IPs:           IP: 10.244.0.3
Controlled By: ReplicaSet/prueba-2-chart-5c5858869b
Containers:
  chart:
    Container ID: docker://8be287be4bbbec737bd3939425dc95ccdf807b4d87872d1e2f15ba30701ef4b8
    Image:         fcongedo/mi-web-apache2:5
    Image ID:     docker-pullable://fcongedo/mi-web-apache2@sha256:0734cdaa6d4bd3a648660b59793f8ce43aa65bdaabd32a11437859ef315a27d6
    Port:          80/TCP
    Host Port:    0/TCP
    State:        Running
    Started:     Wed, 06 Dec 2023 01:44:27 +0000
    Mounts:
```

Ahora para probar que podemos cambiar los valores de las variables voy a crear otro cluster de kubernetes personalizado.

Primero creó el nuevo namespace (prueba-parametros)

Para eso ejecutamos **kubectl create namespace prueba-parametros**

```
vagrant@ubuntu-focal:/vagrant/helm-websites$ kubectl create namespace prueba-parametros
namespace/prueba-parametros created
```

Luego voy a crear un nuevo cluster, pasándole por parámetro el namespace creado, la cantidad de réplicas = 5, y el tag de la imagen = 8

Ejecutando el siguiente comando:

helm install prueba-3 ./chart --set namespace=prueba-parametros --set replicaCount=5 --set image.tag=8

Captura del proceso:

```
vagrant@ubuntu-focal:/vagrant/helm-website$ kubectl create namespace prueba-parametros
namespace/prueba-parametros created
vagrant@ubuntu-focal:/vagrant/helm-website$ helm install prueba-3 ./chart --set namespace=prueba-parametros --set replicaCount=5 --set image.tag=8
NAME: prueba-3
LAST DEPLOYED: Wed Dec 6 02:48:12 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
echo "Visit http://127.0.0.1:8080 to use your application"
kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT
```

Ahora vamos a ejecutar **helm status prueba-3 --show-resources**

```
vagrant@ubuntu-focal:/vagrant/helm-website$ helm status prueba-3 --show-resources
NAME: prueba-3
LAST DEPLOYED: Wed Dec 6 02:48:12 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
RESOURCES:
==> v1/Deployment
NAME READY UP-TO-DATE AVAILABLE AGE
prueba-3-chart 5/5 5 5 25s

==> v1/Pod(related)
NAME READY STATUS RESTARTS AGE
prueba-3-chart-584c95567d-2g5ks 1/1 Running 0 25s
prueba-3-chart-584c95567d-jpg21 1/1 Running 0 25s
prueba-3-chart-584c95567d-krks6 1/1 Running 0 25s
prueba-3-chart-584c95567d-t6jhj 1/1 Running 0 25s
prueba-3-chart-584c95567d-z7vh1 1/1 Running 0 25s
```

(con esto verificamos que se crearon 5 réplicas)

Por último ejecutamos un **kubectl describe pod prueba-3-chart-584c95567d-2g5ks -n prueba-parametros** (verificamos que se creó con la imagen con tag 8)

```
vagrant@ubuntu-focal:/vagrant/helm-website$ kubectl describe pod prueba-3-chart-584c95567d-2g5ks -n prueba-parametros
Name: prueba-3-chart-584c95567d-2g5ks
Namespace: prueba-parametros
Priority: 0
Service Account: default
Node: minikube/192.168.49.2
Start Time: Wed, 06 Dec 2023 02:48:12 +0000
Labels: app.kubernetes.io/instance=prueba-3
        app.kubernetes.io/name=chart
        pod-template-hash=584c95567d
Annotations: <none>
Status: Running
IP: 10.244.0.9
IPs:
  IP: 10.244.0.9
Controlled By: ReplicaSet/prueba-3-chart-584c95567d
Containers:
  chart:
    Container ID: docker://01335b8dc1106637f1ebdd869161c5fbc2dda63f0c6a464c4083b9674928eceb
    Image: fcongedo/mi-web-apache2:8
    Image ID: docker-pullable://fcongedo/mi-web-apache2@sha256:8a0866bc48270544363b1038f3ff6054740d5870bbb4e65bda08e41cda0cd4fe
    Port: 80/TCP
    Host Port: 0/TCP
    State: Running
    Started: Wed, 06 Dec 2023 02:48:20 +0000
    Ready: True
    Restart Count: 0
    Environment: <none>
    Mounts:
```

- **ArgoCD**

El ArgoCD CLI lo instalamos a través del bootstrap.sh al iniciar la máquina de vagrant.

Luego vamos a instalar argoCD en el cluster

Primero vamos a hacer un minikube start

Luego ejecutamos **kubectl create namespace argocd** (creamos el namespace)

Luego ejecutamos **kubectl apply -n argocd -f**

<https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/core-install.yaml>

Captura del proceso:

```
vagrant@ubuntu-focal:~$ kubectl create namespace argocd
namespace/argocd created
vagrant@ubuntu-focal:~$ kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/core-install.yaml
customresourcedefinition.apiextensions.k8s.io/applications.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/applicationsets.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/appprojects.argoproj.io created
serviceaccount/argocd-application-controller created
serviceaccount/argocd-applicationset-controller created
serviceaccount/argocd-redis created
serviceaccount/argocd-repo-server created
```

Accedemos al servicio **kubectl port-forward svc/argocd-server -n argocd 8080:443**

Adjunto captura:

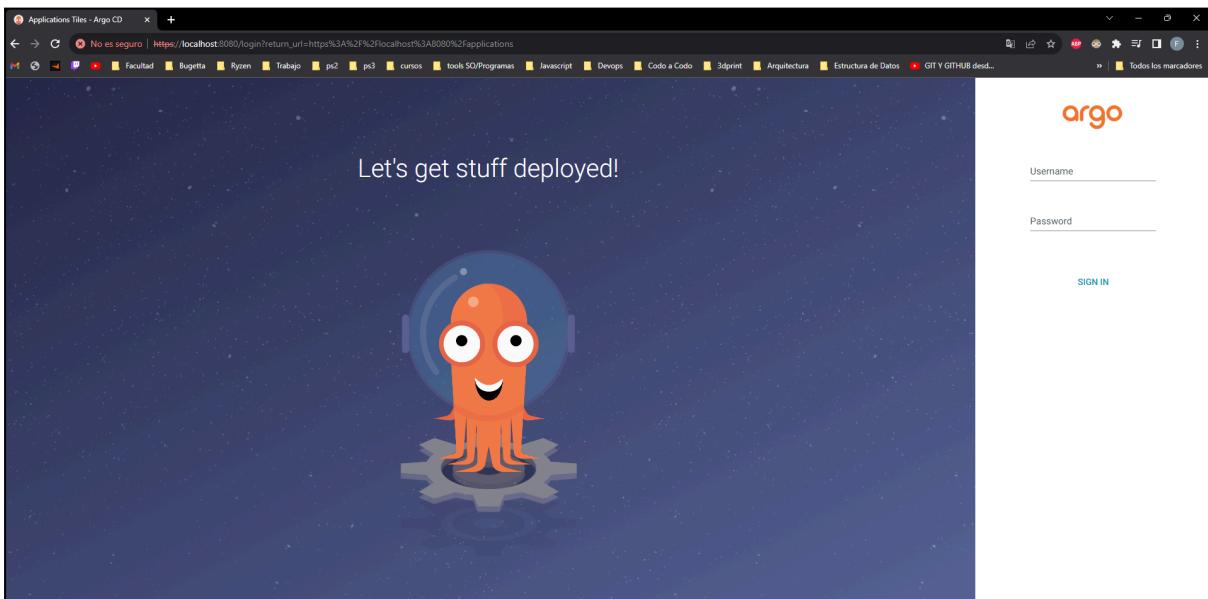
```
vagrant@ubuntu-focal:~$ kubectl port-forward svc/argocd-server -n argocd 8080:443
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

Abrimos otro vagrant ssh y hacemos un túnel de ssh utilizando

vagrant ssh -- -L 8080:localhost:8080

Luego accedemos a la interfaz gráfica de ArgoCD haciendo un **localhost:8080**

Captura proceso:



Para logearnos primero ejecutamos el comando **argocd admin initial-password -n argocd** (para darnos la password por defecto)

Captura proceso:

```
vagrant@ubuntu-focal:~$ argocd admin initial-password -n argocd
MP51U2UC5K9r6RZ

This password must be only used for first time login. We strongly recommend you update the password using `argocd account update-password`.
```

Luego ejecutamos un **argocd login localhost:8080**

Ingresamos el **usuario admin**

Y la **password** por defecto que nos dio anteriormente

Captura proceso:

```
vagrant@ubuntu-focal:~$ argocd login localhost:8080
WARNING: server certificate had error: tls: failed to verify certificate: x509: certificate signed by unknown authority. Proceed insecurely (y/n)? y
Username: admin
Password:

'admin:login' logged in successfully
Context 'localhost:8080' updated
```

Ahora vamos a añadir nuestro cluster

Primero vemos el nombre del contexto

Ejecutamos **kubectl config get-contexts**

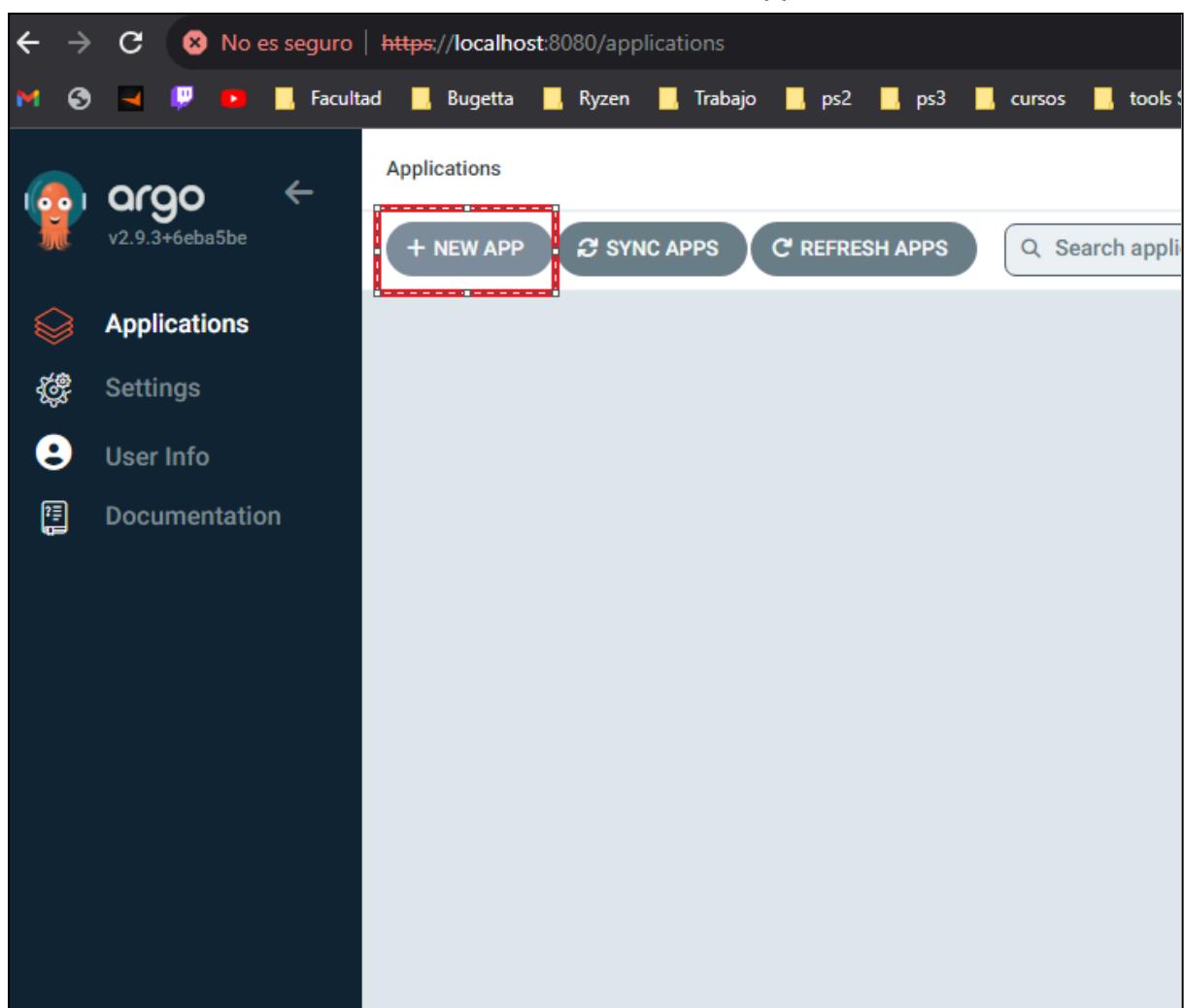
Luego añadimos el cluster ejecutando **argocd cluster add minikube**

Captura proceso:

```
vagrant@ubuntu-focal:~$ kubectl config get-contexts
CURRENT NAME          CLUSTER      AUTHINFO      NAMESPACE
*   minikube   minikube   minikube   default
vagrant@ubuntu-focal:~$ argocd cluster add minikube
WARNING: This will create a service account `argocd-manager` on the cluster referenced by context `minikube` with full cluster level privileges. Do you want to continue [y/N]? y
[INFO] 0002] ServiceAccount "argocd-manager" created in namespace "kube-system"
[INFO] 0002] ClusterRole "argocd-manager-role" created
[INFO] 0002] ClusterRoleBinding "argocd-manager-role-binding" created
[INFO] 0007] Created bearer token secret for ServiceAccount "argocd-manager"
Cluster 'https://192.168.49.2:8443' added
```

Para levantar nuestra aplicación convertida en un chart de Helm (vamos a utilizar el método de interfaz gráfica)

Para eso dentro de la interfaz hacemos click en +new app



Primero configuramos el nombre de la aplicación **website-helm**
 Luego el project name, dejamos el **default**
 Activamos las siguientes opciones **self heal, auto-create namespace y autosync(apply out of sync only)**

The screenshot shows the Helm repository configuration interface. In the 'GENERAL' section, the 'Application Name' is set to 'website-helm' and the 'Project Name' is set to 'default'. In the 'SYNC POLICY' section, 'Automatic' is selected. Under 'Sync Options', 'SELF HEAL' is checked. In the 'Sync Options' section, 'AUTO-CREATE NAMESPACE' and 'APPLY OUT OF SYNC ONLY' are checked.

Agregamos el link del repositorio, dejamos el branch en head, y seleccionamos el path donde está la aplicación

The screenshot shows the Helm repository configuration interface under the 'SOURCE' tab. The 'Repository URL' is set to 'https://github.com/fcongedo/desafio-grupal-k8s'. The 'HEAD' branch is selected. The 'Path' is set to 'helm-website/chart'.

Seleccionamos el cluster default y el namespace

DESTINATION

Cluster URL
https://kubernetes.default.svc

URL ▾

Namespace
prueba-desafiook8s-helm

Por último en la etapa de helm
Modificamos el tag de la imagen
La cantidad de réplicas
Y el namespace

Helm ▾

HELM

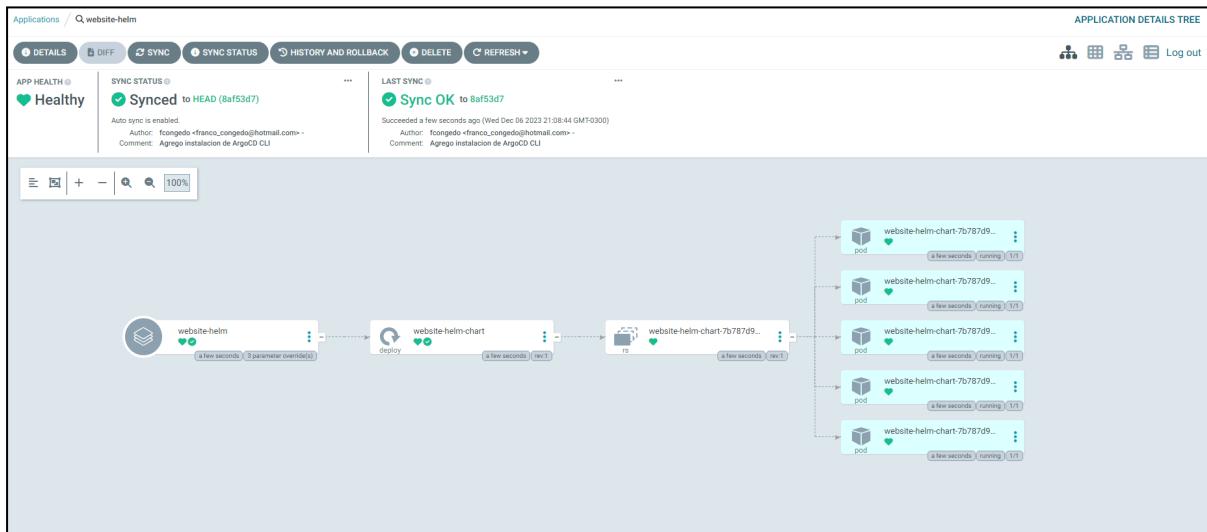
VALUES FILES
values.yaml X

VALUES

PARAMETERS

image.tag	8	Remove override
replicaCount	5	Remove override
namespace	prueba-desafiook8s-helm	Remove override

Aplicación creada con 5 réplicas



Captura de los pods en la terminal

Ejecutando `kubectl get pods -n prueba-desafiook8s-helm`

NAME	READY	STATUS	RESTARTS	AGE
website-helm-chart-7b787d986c-445t5	1/1	Running	0	16m
website-helm-chart-7b787d986c-7m7db	1/1	Running	0	16m
website-helm-chart-7b787d986c-fj5rr	1/1	Running	0	16m
website-helm-chart-7b787d986c-jgc4d	1/1	Running	0	16m
website-helm-chart-7b787d986c-qttxz	1/1	Running	0	16m

Por último describimos un pod ejecutando `kubectl describe pod`

`website-helm-chart-7b787d986c-445t5 -n prueba-desafiook8s-helm`

```
vagrant@ubuntu-focal:~$ kubectl describe pod website-helm-chart-7b787d986c-445t5 -n prueba-desafiook8s-helm
Name:           website-helm-chart-7b787d986c-445t5
Namespace:      prueba-desafiook8s-helm
Priority:      0
Service Account: default
Node:          minikube/192.168.49.2
Start Time:    Thu, 07 Dec 2023 00:08:44 +0000
Labels:        app.kubernetes.io/instance=website-helm
               app.kubernetes.io/name=chart
               pod-template-hash=7b787d986c
Annotations:   <none>
Status:        Running
IP:            10.244.0.11
IPs:
  IP:          10.244.0.11
Controlled By: ReplicaSet/website-helm-chart-7b787d986c
Containers:
  chart:
    Container ID: docker://83fae40c817d13ff7887228a86fcceeacaba3b1a6ed79d8e1e70691e3c04e14d
    Image:         fccongedo/mi-web-apache2:8
    Image ID:     docker-pullable://fccongedo/mi-web-apache2@sha256:8a0866bc48270544363b1038f3ff6054740d5870bbb4e65bda08e41cda0cd4fe
    Port:          80/TCP
    Host Port:    80/TCP
    State:        Running
      Started:   Thu, 07 Dec 2023 00:09:08 +0000
    Ready:        True
    Restart Count: 0
    Environment:  <none>
    Mounts:
```

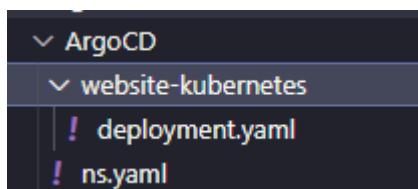
(Observamos que se levantó con la imagen tag:8)

Para levantar nuestra aplicación en recursos de Kubernetes normales (sin Helm)

La vamos a desplegar de manera manual.

Dentro del directorio **ArgoCD**, se encuentra el **ns.yaml** y dentro de website-kubernetes, el **deployment.yaml** (creado en la fase kubernetes)

Captura carpeta:



Primero vamos a crear el namespace ejecutando

kubectl create ns prueba-desafio8s

```
vagrant@ubuntu-focal:~$ kubectl create ns prueba-desafio8s
namespace/prueba-desafio8s created
```

Luego ejecutamos el siguiente comando

argocd app create website-kubernetes --repo

https://github.com/fcongedo/desafio-grupal-k8s --path

ArgoCD/website-kubernetes --dest-server https://kubernetes.default.svc

--dest-namespace prueba-desafio8s

Explicación comando

Primero contiene el nombre de la aplicación: **website-kubernetes**

Segundo le pasamos el **link del repositorio de github**

Tercero le pasamos el **path de donde está alojada la aplicación**

cuarto elegimos el **servidor (default)**

Por último le pasamos el nombre del **namespace** (creado anteriormente)

Captura de proceso:

```
vagrant@ubuntu-focal:vagrant$ argocd app create website-kubernetes --repo https://github.com/fcongedo/desafio-grupal-k8s --path ArgoCD/website-kubernetes -
--dest-server https://kubernetes.default.svc --dest-namespace prueba-desafio8s
application 'website-kubernetes' created
```

Luego en la interfaz gráfica vemos:

Application	Status	Sync Status	Repository	Target Revision	Path	Destination	Namespace	Created At	Last Sync
website-helm	Healthy	Synced	https://github.com/fcongedo/desafio-grupal-k8s	HEAD	helm-website/chart	in-cluster	prueba-desafiook8s-helm	12/06/2023 21:08:40 (2 hours ago)	12/06/2023 21:08:44 (2 hours ago)
website-kubernetes	Missing	OutOfSync	https://github.com/fcongedo/desafio-grupal-k8s	HEAD	ArgoCD/website-kubernetes	in-cluster	prueba-desafiook8s	12/06/2023 23:34:31 (a minute ago)	

(las 2 aplicaciones, la de helm ya corriendo y la de kubernetes esperando sync)

Hacemos click en sobre la aplicación y nos muestra su estado

Luego hacemos click en **sync**

APP HEALTH Missing

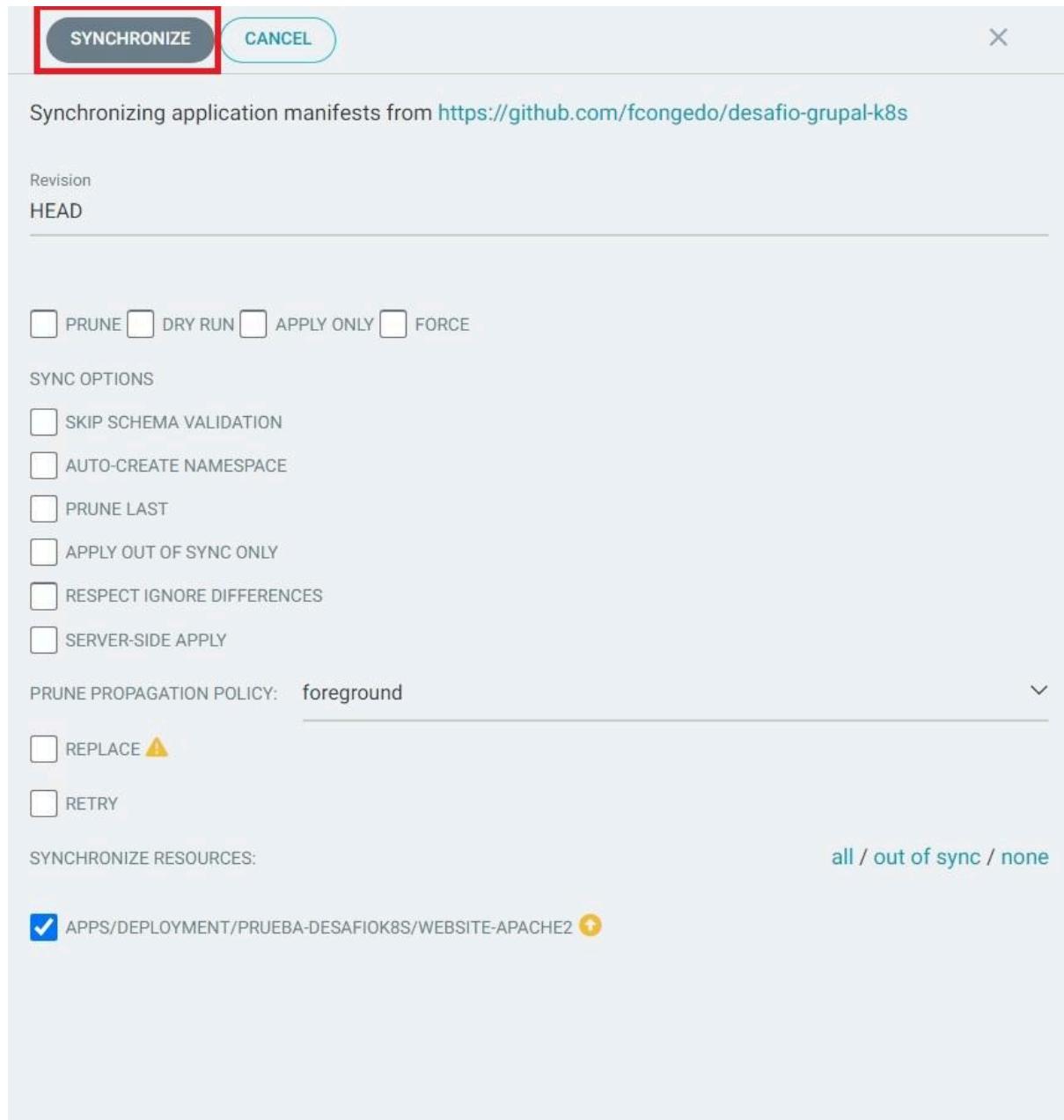
SYNC STATUS OutOfSync from HEAD (4dea85d)

Auto sync is not enabled.
Author: fcongedo <franco_congedo@hotmail.com> -
Comment: Agrego carpeta con app en recursos de kubernetes

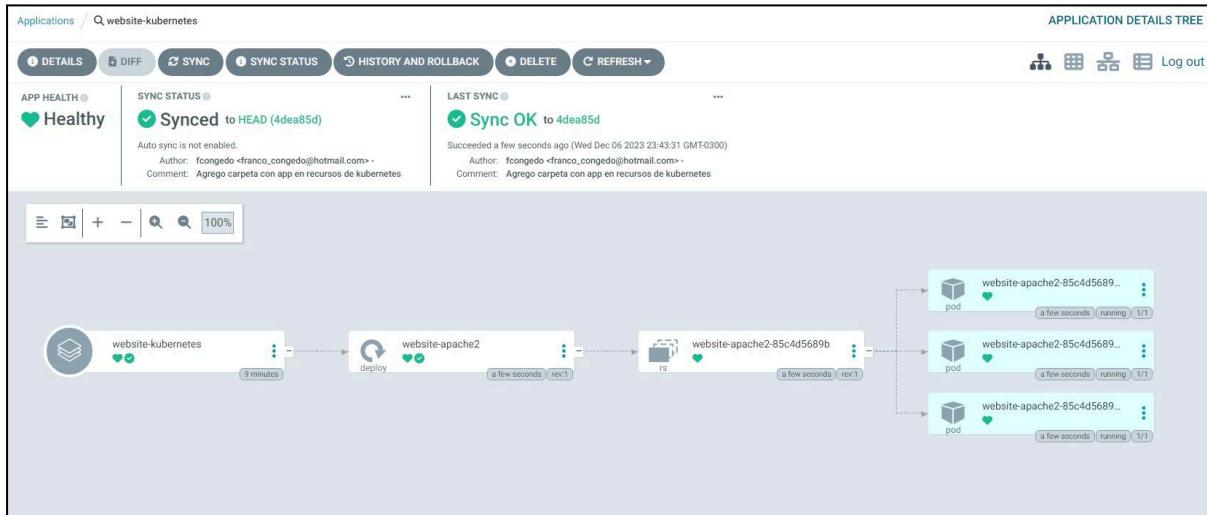
SYNC

website-kubernetes → **website-apache2**

Se nos abre para configurar la aplicación (en este caso no modificamos nada)
Y hacemos click en **synchronize**



Y nos crea los recursos



Captura de las 2 aplicaciones corriendo:

Adjunto captura de los pods en la terminal
Ejecutando **kubectl get pods -n prueba-desafio8s**

```
vagrant@ubuntufocal:/vagrant$ kubectl get pods -n prueba-desafio8s
NAME                               READY   STATUS    RESTARTS   AGE
website-apache2-85c4d5689b-7d59g   1/1     Running   0          23m
website-apache2-85c4d5689b-k6v26   1/1     Running   0          23m
website-apache2-85c4d5689b-rl6fw   1/1     Running   0          23m
```

Por último describimos un pod ejecutando **kubectl describe pod website-apache2-85c4d5689b-7d59g -n prueba-desafiolok8s**

```
vagrant@ubuntu-focal:/vagrant$ kubectl describe pod website-apache2-85c4d5689b-7d59g -n prueba-desafiolok8s
Name:           website-apache2-85c4d5689b-7d59g
Namespace:      prueba-desafiolok8s
Priority:       0
Service Account: default
Node:          minikube/192.168.49.2
Start Time:    Thu, 07 Dec 2023 02:43:32 +0000
Labels:         app=website-apache2
                pod-template-hash=85c4d5689b
Annotations:   <none>
Status:        Running
IP:            10.244.0.17
IPs:
  IP:          10.244.0.17
Controlled By: ReplicaSet/website-apache2-85c4d5689b
Containers:
  website:
    Container ID: docker://f40a48ab3d49b544823d7cb6067bc61541e908c7ba9317d85416803f7be6b122
    Image:          fcongedo/mi-web-apache2:5
    Image ID:      docker-pullable://fcongedo/mi-web-apache2@sha256:0734cdaa6d4bd3a648660b59793f8ce43aa65bdaabd32a11437859ef315a27d6
    Port:          80/TCP
    Host Port:    80/TCP
    State:        Running
    Started:     Thu, 07 Dec 2023 02:43:37 +0000
    Ready:        True
    Restart Count: 0
    Environment:  <none>
    Mounts:
```

(Observamos que se levantó con la imagen tag:5)

- *Links*

Repositorio de código: <https://github.com/fcongedo/desafio-grupal-k8s>

Imagen docker hub:

<https://hub.docker.com/repository/docker/fcongedo/mi-web-apache2/general>