ArgoCD Despliegue de aplicación Hola Mundo

Entorno.

S.O: Windows 10 Pro

RAM: 32 GB

Procesador i7

Instalación.

Minikube

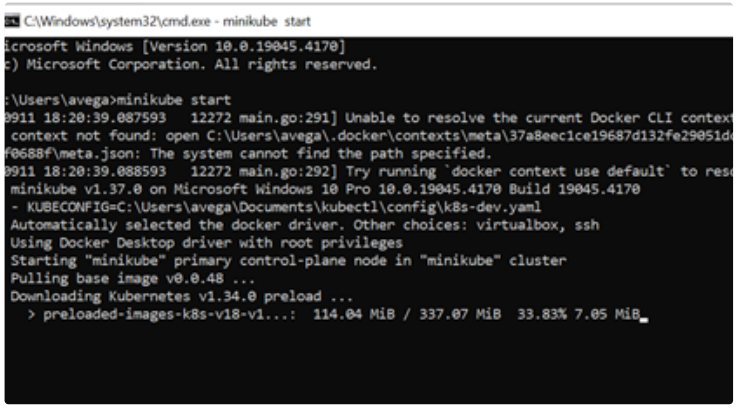
Para realizar la instalación de MiniKube ingresamos a la página oficial para descargar la versión compatible con el sistema operativo.

```
https://minikube.sigs.k8s.io/docs/
```

Seguir las instrucciones y una vez finalizado se valida la version desde la consola con la instrucción.

```
minikube version
```

Después de validar la versión se inicia el clúster con la instrucción “minikube start”.



Para organizar los recursos del clúster se creará el namespace para los objetos K8 de Argo.

Se crea namespace “argo“

```
Kubectl create ns argo
```

```
- Want kubectl v1.34.0? Try 'minikube kubectl -- get pods -A'
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

C:\Users\avega>kubectl create ns argo
namespace/argo created
```

Argo

Ejecución de manifiesto de Argo en cluster Minikube

Se realiza la instalación de Argo Server y UI Argo aplicando el manifiesto que contiene todo lo necesario a desplegar en ese archivo.
URL: https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml.

```
kubectl -n argo -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

```
C:\Users\avega>kubectl apply -n argo -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml

Warning: unrecognized format "int64"
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-dex-server created
```

Una vez finalizado se obtiene la información de los pods en el namespace 'argo'.

```
kubectl -n argo get pods
```

```
C:\Users\avega>kubectl get pods -n argo
NAME                                               READY   STATUS    RESTARTS      AGE
argocd-application-controller-0                    0/1     Running   0             30m
argocd-applicationset-controller-8676b8756b-n9rlv  1/1     Running   0             30m
argocd-dex-server-68b5d79cb9-fplqg                 1/1     Running   1 (19s ago)   30m
argocd-notifications-controller-bfc494cf8-qtwqg    1/1     Running   0             30m
argocd-redis-656fdf7c5d-89hcx                      1/1     Running   0             30m
argocd-repo-server-6bcdd69bbb-qdwtv                1/1     Running   0             30m
argocd-server-648fc5d9df-9sgzr                     0/1     Running   0             30m

C:\Users\avega>
```

Obtener información de los servicios.

```
kubectl -n argo get svc
```

```
C:\Users\avega>kubectl get pods -n argo
NAME                                               READY   STATUS    RESTARTS      AGE
argocd-application-controller-0                    0/1     Running   0             30m
argocd-applicationset-controller-8676b8756b-n9rlv  1/1     Running   0             30m
argocd-dex-server-68b5d79cb9-fplqg                 1/1     Running   1 (19s ago)   30m
argocd-notifications-controller-bfc494cf8-qtwqg    1/1     Running   0             30m
argocd-redis-656fdf7c5d-89hcx                      1/1     Running   0             30m
argocd-repo-server-6bcdd69bbb-qdwtv                1/1     Running   0             30m
argocd-server-648fc5d9df-9sgzr                     0/1     Running   0             30m

C:\Users\avega>
```

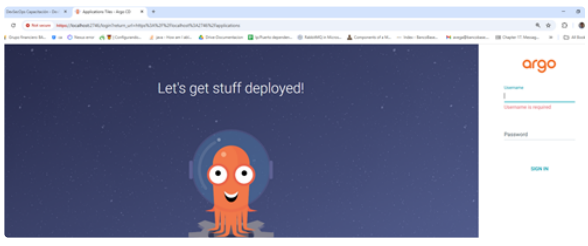
Ingresar Interfaz UI Argo

Para ingresar al service del clúster de minikube desde el equipo local es necesario realizar un túnel de comunicación con la instrucción 'port-forward'.

```
kubectl port-forward svc/argocd-server -n argo 2746:443
```

```
C:\Users\avega\Documents\001_devops>kubectl port-forward svc/argocd-server -n argo 2746:443
Forwarding from 127.0.0.1:2746 -> 8080
Forwarding from [::1]:2746 -> 8080
Handling connection for 2746
Handling connection for 2746
```

Con esta instrucción queda abrierta una comunicación local por el puerto 2746 al service del clúster por el puerto 8080.
Para ingresar a la interfaz se accede desde el navegador con la url: https://localhost:2746/



El usuario default es admin, y la constraseña se encuentra dentro del secreto registrado con el nombre argo argocd-initial-admin-secret.

```
C:\Users\avega>kubectl get secrets -n argo
NAME                          TYPE     DATA   AGE
argocd-initial-admin-secret   Opaque   1      5h29m
argocd-notifications-secret   Opaque   0      5h35m
argocd-redis                  Opaque   1      5h35m
argocd-secret                 Opaque   5      5h35m

C:\Users\avega>
```

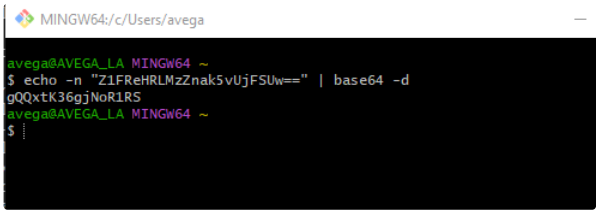
Para obtener el valor codificado se ejecuta la instrcción:

```
Kubectl get secret -n argo argocd-initial-admin-secret -o yaml
```

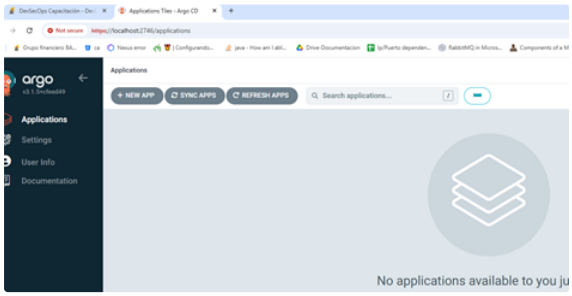
```
C:\Users\avega>kubectl get secret -n argo argocd-initial-admin-secret -o  yaml
apiVersion: v1
data:
  password: Z1FReHRLMzZnak5vUjFSUw==
kind: Secret
metadata:
  creationTimestamp: "2025-09-12T00:56:58Z"
  name: argocd-initial-admin-secret
  namespace: argo
  resourceVersion: "2624"
  uid: 88977cb1-84d1-461d-b329-b23f450352d5
type: Opaque

C:\Users\avega>
```

y se decompila el valor del atributo data.password con la instrucción:

```
echo -n "String codificado" | base64 -d
```



Una vez teniendo la contraseña decompilada se ingresa a la plataforma.



Despliegue de aplicación.

Creación clase HolaMundo.java.

Con finalidad de la práctica se crea una clase en java que solo va a imprimir el texto “Hola Mundo“, para esto se crea la clase con lo siguiente:

```java
public class HolaMundo{

    public static void main (String args[]) throws InterruptedException {
        System.out.println("Hola mundo");
        while(true){                        //Se mantiene un cliclo infinito para que la aplicación no finalice y el pod me mantenga en
            Thread.sleep(1000);
        }
    }
}
```

Creación Dockerfile.

Se crea el archivo Dockerfile que va a contener las instrcciones para crear la imagen Docker donde se ejecutará la clase creada de “HolaMundo.java”.

```
FROM openjdk:17 AS build                                    //Obtiene la imagen openjdk:17 para compilar

WORKDIR /src                                                // Obtiene el directorio de trabajo
COPY HolaMundo.java .
RUN javac HolaMundo.java                                    // Compila el código con javac generando el bytecode

FROM openjdk:17-slim                                        // Obtiene una imagen ligera

WORKDIR /app                                                // Define el directorio /app
COPY --from=build /src/HolaMundo.class .                    // Copia la clase compilada.

CMD ["java", "HolaMundo"]                                   // Ejecuta la clase compilada
```

Registrar imagen en Docker hub.

Para realizar el despliegue se va a ocupar el repositorio publico de Docker hub por lo que será necesario crear una cuenta en la página: https://hub.docker.com/ .
posterior para hacer un push o registrar la imagen será necesario realizar login con la instrucción:

```
docker login -u miusuario -p mipassword
```

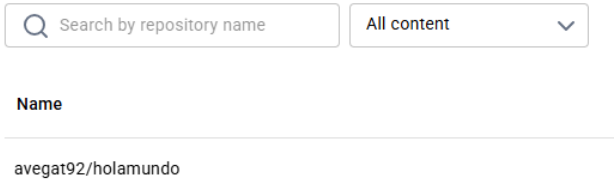
para construir la imagen con un tag se ejecuta la siguiente instrucción:

```
docker build -t usuarioDockerHub/nombreDeImagen:version .
```

y se valida obteniendo las imagenes creadas con la instrucción: “docker images”

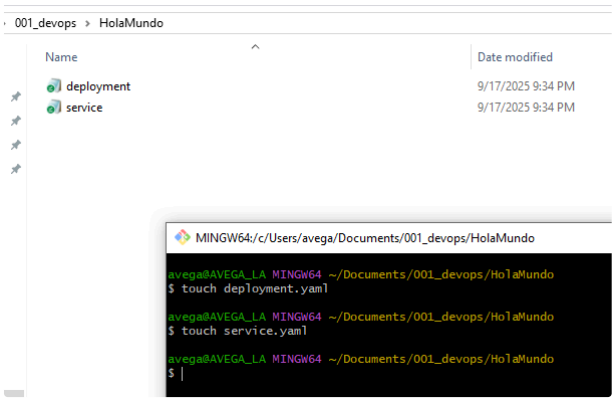
y por último se ejecuta libera la imagen con la instrucción “docker push usuarioDockerHub/nombreDeImagen:version“.

Se confirma en el repositorio remoto.



Creación manifiestos.

Se crean los manifiestos “deployment.yaml” y “service.yaml”.



Deployment.YAML

```
apiVersion: apps/v1
```

```yaml
kind: Deployment
metadata:
  name: gfb-holamundo                           //nombre del despliegue
  namespace: bpo                                //nombre del namespace en el cluster de minikube
spec:
  replicas: 1                                   // numero de pods a desplegar
  revisionHistoryLimit: 5
  selector:
    matchLabels:
      app: gfb-holamundo                        //selector para hacer match con el recurso de service, de esta manera se vincula el obje
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: gfb-holamundo
    spec:
      containers:
        - image: avegat92/holamundo:latest      //imagen a descargar(Pull)
          name: gfb-holamundo
          ports:
            - containerPort: 80                //puesto del contenedor
```

Service.YAML

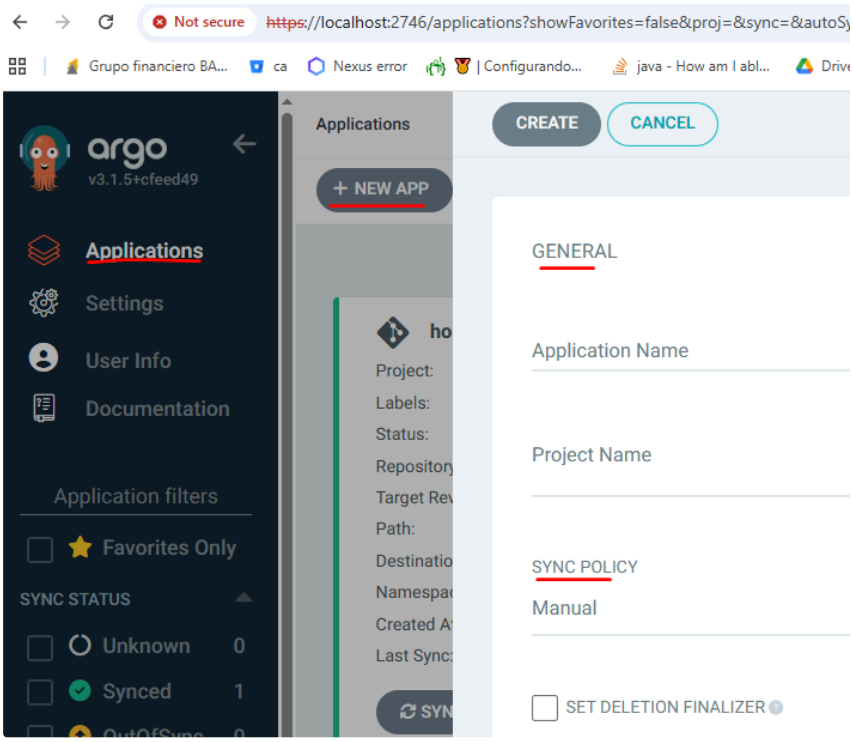
```yaml
apiVersion: v1
kind: Service                                   // Tipo de manifiesto u objeto  Service
metadata:
  name: gfb-holamundo                           // Nombre de service
spec:
  ports:
  - port: 80                                    // puerto de el que estará escuchando el servicio
    targetPort: 80                              // puerto por el que será redirigido
  selector:
    app: gfb-holamundo                          //selector para hacer match con el recurso de deployment, de esta manera se vincula el ob
```

Para que se realice la descarga será necesario subirlo a un repositorio en GitHub, por lo que es necesario tener una cuenta en la plataforma “https://github.com/ “

Creación aplicación en Argo.

La aplicación de Argo se puede crear desde la plataforma o por instrucción desde consola.

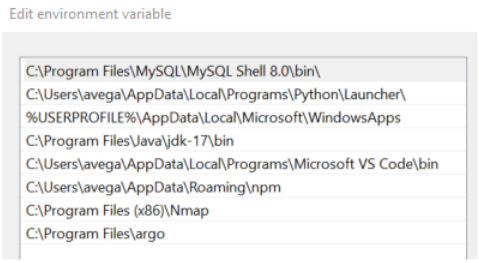
Para crearlo desde consola se accede al Menú Applications⟶ New App y se completa la información solicitada, para que se actualice cuando exista un cambio en los manifiestos se debe agregar la politica de automatizado.
Sync Policy ⟶Automated



Para ejecutar desde local una instrucción argocd se debe descargar el cliente argocd para windows mediante la siguiente instrucción para powershell.

```
Invoke-WebRequest -Uri "https://github.com/argoproj/argo-cd/releases/download/v2.7.1/argocd-windows-amd64.exe" -OutFile "C:\Program Files\
```

Se realizará la copia en la ruta especificada en: -OutFile "C:\Program Files\argocd.exe"
Declaramos la ubicación como variable de entorno para que pueda ser reconocido el ejecutable desde cualquier ubicación.



Por último se realiza un login con el servidor argocd con la siguiente instrucción y manteniendo la comunicación local con port-forward

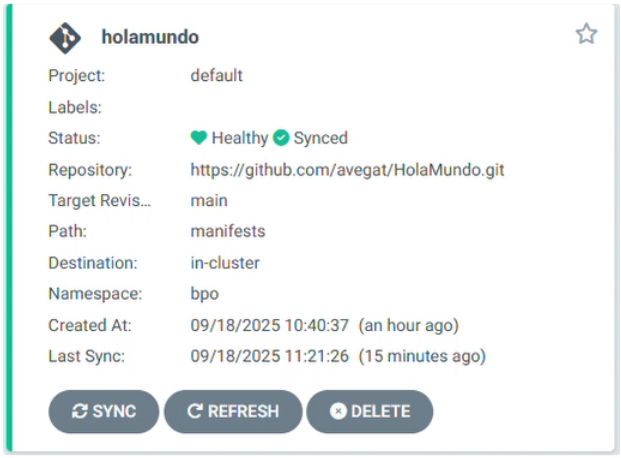
```
argocd login localhost:8080 --username admin --password <PASSWORD>
```

La creación de la aplicación en argo se puede realizar con la instrucción:

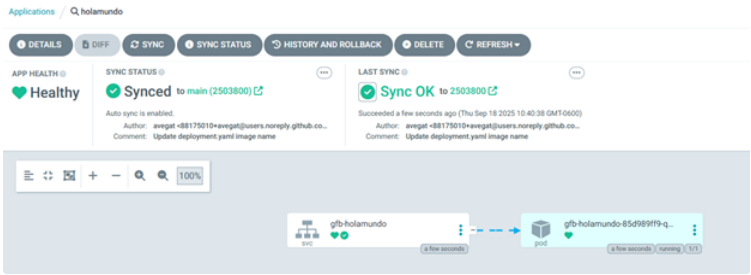
```
argocd app create holamundo  --repo https://github.com/avegat/HolaMundo.git --path manifests  --dest-server https://kubernetes.default.svc
```

--repo //ruta del repositorio
--path //ubicación de los manifiestos
--dest-server //cluster donde se despliega, en este caso es el mismo
--dest-namespace //namespace de despliegue
--sync-policy //sincronización de cambios automatico o manual
--auto-prune //limpia los recursos que no estan en uso
--revision //rama de despliegue

Una vez ejecutado se valida en la plataforma de Argo el despliegue



Estado de pods.



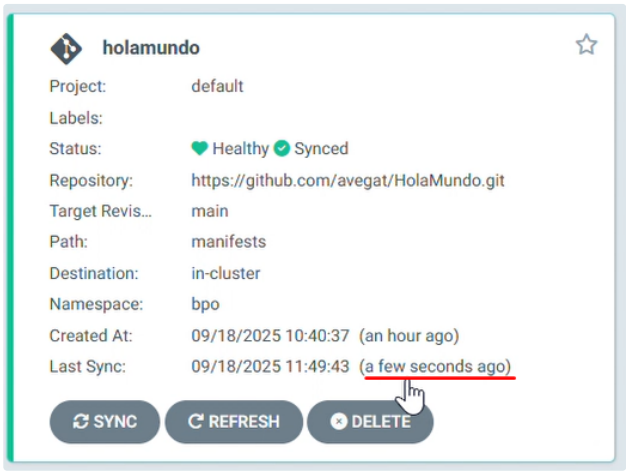
y se puede confirmar los pods localmente con la instrucción ejecutada desde consola: **“kubectl -n bpo get pods”**

Sincronización en despliegue.

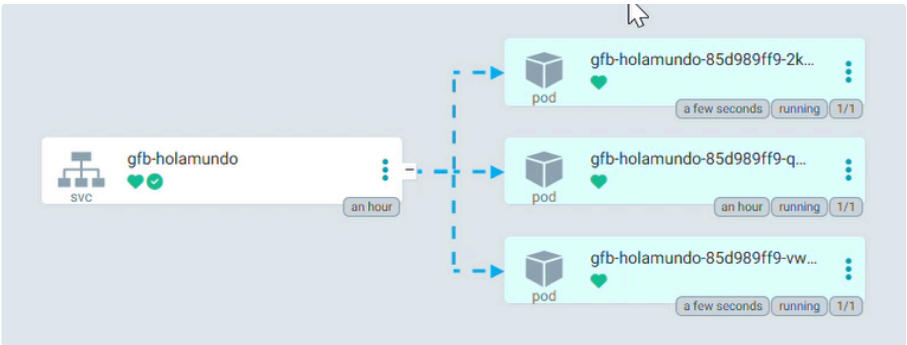
Se actualiza el manifiesto 'deployment.yaml' cambiando el numero de replica a valor 3.

```
spec:
  replicas: 3
```

Cuando se haga el cambio sobre la rama main, la herramienta detectara el cambio y actualizará el despliegue.



Al ingresar al pod se mostrarán las 3 replicas del contenedor en ejecución.



Se valida desde consola los mismos 3 pod

```
C:\Users\avega>kubectl get pods -n bpo
NAME                              READY   STATUS    RESTARTS   AGE
gfb-holamundo-85d989ff9-2kztk   1/1     Running   0          169m
gfb-holamundo-85d989ff9-qwcfl   1/1     Running   0          3h58m
gfb-holamundo-85d989ff9-vwnkm   1/1     Running   0          169m
```

y el log de uno de ellos.

```
C:\Users\avega>kubectl logs gfb-holamundo-85d989ff9-2kztk -n bpo
Hola mundo
```