

Desafíos # 13-14

Realizado por: Joselin Teixeira

Fecha de entrega: 24-09-2024

Se parte de un entorno local de Kubernetes utilizando Minikube como plataforma de desarrollo. El equipo de trabajo ha solicitado la implementación de ArgoCD para gestionar el flujo de despliegues de aplicaciones de manera automatizada. Se utilizarán manifiestos de Kubernetes y se integrará la plataforma con un repositorio Git para la gestión de aplicaciones.

Requisitos:

1. Instalación de Minikube.
2. Instalación de kubectl.
3. Despliegue de ArgoCD en Minikube.
4. Configuración de ArgoCD.
5. Conectar ArgoCD con un repositorio Git.
6. Verificación del despliegue.

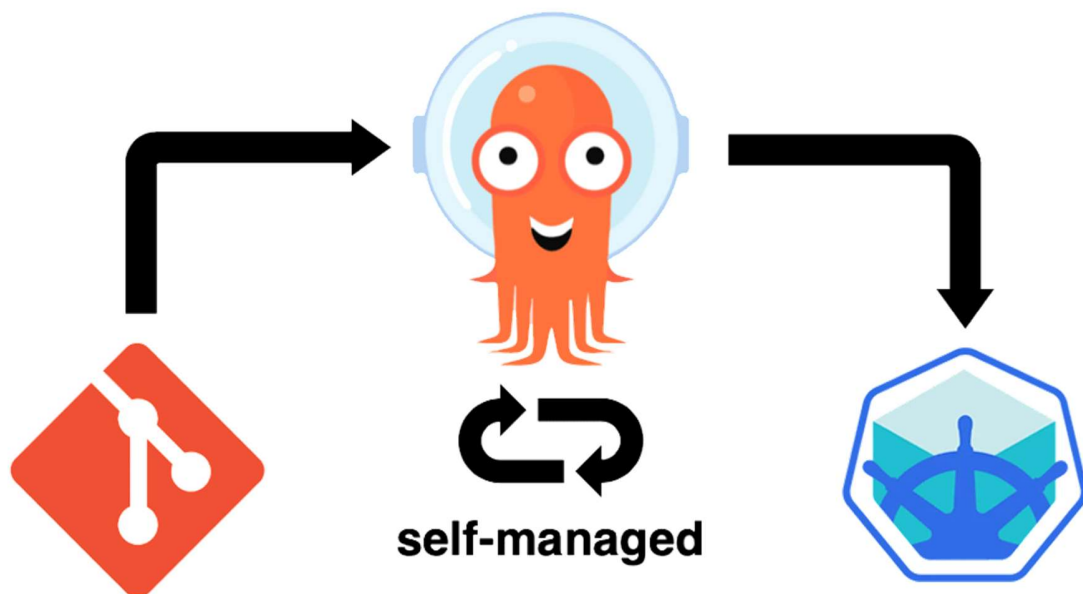
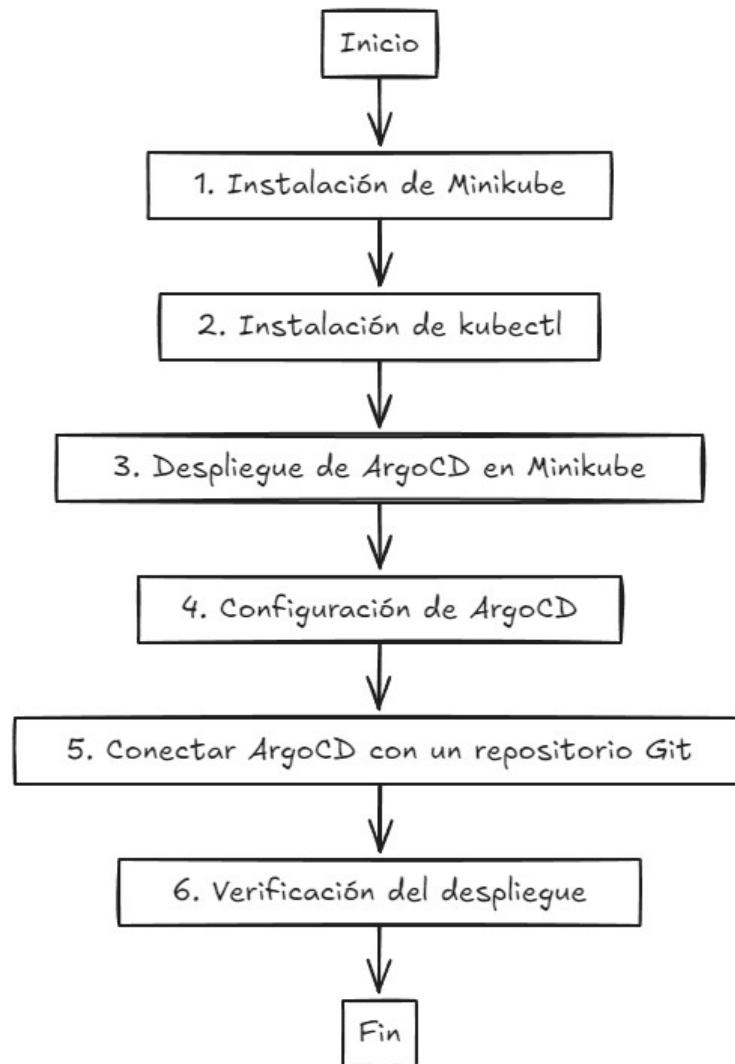


Diagrama de flujo



Referencias:

[MiniKube](#)

[Instalación Minikube](#)

[ArgoCD](#)

[Argo CD Descripción General](#)

[ArgoCD Image Updater](#)

[ArgoCD Example Apps](#)

Desafío 13 - Instalación ArgoCD Minikube

Objetivo:

El objetivo de este desafío es guiar la instalación y configuración de ArgoCD en un entorno de Minikube. ArgoCD es una herramienta de GitOps utilizada para gestionar despliegues en Kubernetes de manera declarativa. A través de este ejercicio, se pondrán en práctica los pasos necesarios para implementar una infraestructura automatizada de despliegue continuo.

1. Instalación de Minikube:

En nuestro entorno local (Windows) ya tenemos instalado Minikube, sin embargo, para realizar la instalación solo requerimos ejecutar desde nuestro terminal con permisos de Administrador el siguiente comando:

```
>winget install Kubernetes.minikube
```

2. Instalación de kubectl:

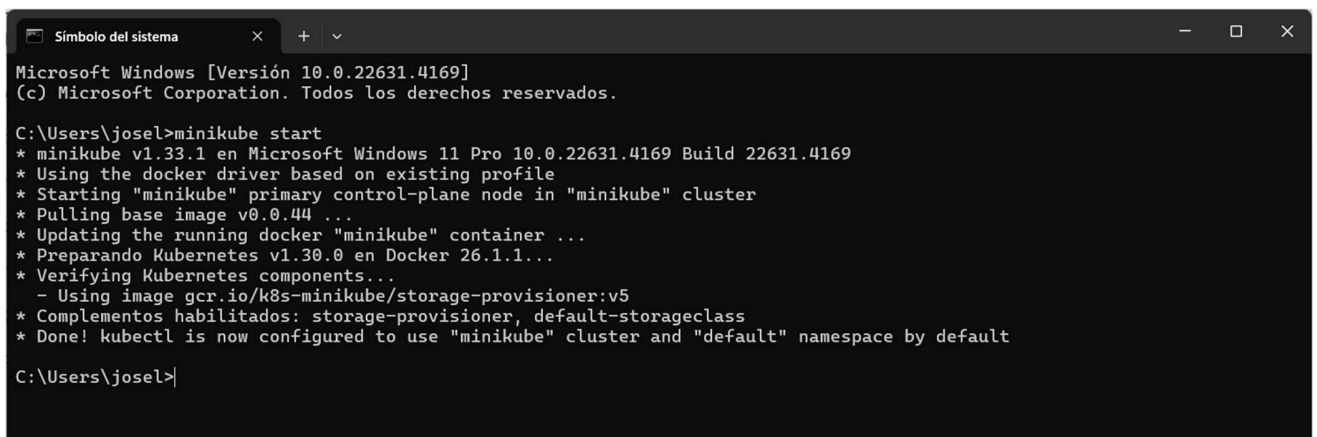
Debemos asegurarnos de tener instalado kubectl para interactuar con el clúster de Kubernetes en Minikube. Guía de instalación: <https://kubernetes.io/docs/tasks/tools/install-kubectl/>. Para nuestro entorno de Windows, ejecutaremos el siguiente comando:

```
>winget install -e --id Kubernetes.kubectl
```

3. Despliegue de ArgoCD en Minikube:

- *Ejecutar Minikube:* Ejecutamos el siguiente comando desde la terminal (Docker desktop debe estar iniciado)

```
>minikube start
```



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.22631.4169]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\josel>minikube start
* minikube v1.33.1 en Microsoft Windows 11 Pro 10.0.22631.4169 Build 22631.4169
* Using the docker driver based on existing profile
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.44 ...
* Updating the running docker "minikube" container ...
* Preparando Kubernetes v1.30.0 en Docker 26.1.1...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Complementos habilitados: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

C:\Users\josel>
```

- *Instalar el namespace de ArgoCD:*

```
>kubectl create namespace argocd
```

```
Simbolo del sistema x + v

C:\Users\josel>minikube start
* minikube v1.33.1 en Microsoft Windows 11 Pro 10.0.22631.4169 Build 22631.4169
* minikube 1.34.0 is available! Download it: https://github.com/kubernetes/minikube/releases/tag/v1.34.0
* To disable this notice, run: 'minikube config set WantUpdateNotification false'

* Using the docker driver based on existing profile
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.44 ...
* Restarting existing docker container for "minikube" ...
* Preparando Kubernetes v1.30.0 en Docker 26.1.1...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Complementos habilitados: default-storageclass, storage-provisioner
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

C:\Users\josel>kubectl create namespace argocd
namespace/argocd created

C:\Users\josel>|
```

- *Desplegar los recursos de ArgoCD en Minikube:* Instalaremos todas las dependencias de Argo

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

```
Simbolo del sistema x + v

C:\Users\josel>kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/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-dex-server created
serviceaccount/argocd-notifications-controller created
serviceaccount/argocd-redis created
serviceaccount/argocd-repo-server created
serviceaccount/argocd-server created
```

4. Configuración de ArgoCD:

- Exponer el servicio ArgoCD Server para acceder al dashboard:

```
>kubectl port-forward svc/argocd-server -n argocd 8080:443
```

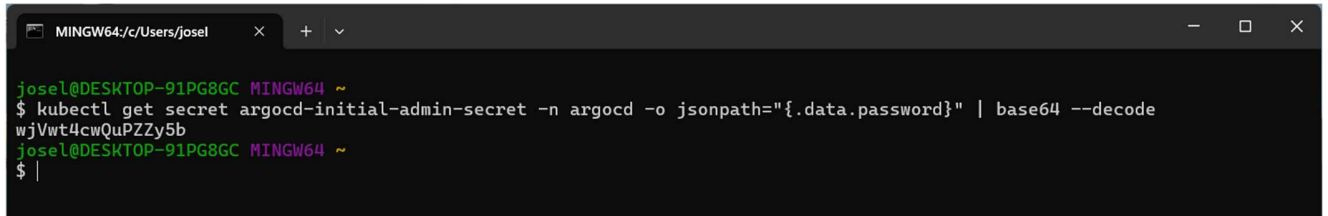
```
Simbolo del sistema - kubectl x + v

service/argocd-notifications-controller-metrics created
service/argocd-redis created
service/argocd-repo-server created
service/argocd-server created
service/argocd-server-metrics created
deployment.apps/argocd-applicationset-controller created
deployment.apps/argocd-dex-server created
deployment.apps/argocd-notifications-controller created
deployment.apps/argocd-redis created
deployment.apps/argocd-repo-server created
deployment.apps/argocd-server created
statefulset.apps/argocd-application-controller created
networkpolicy.networking.k8s.io/argocd-application-controller-network-policy created
networkpolicy.networking.k8s.io/argocd-applicationset-controller-network-policy created
networkpolicy.networking.k8s.io/argocd-dex-server-network-policy created
networkpolicy.networking.k8s.io/argocd-notifications-controller-network-policy created
networkpolicy.networking.k8s.io/argocd-redis-network-policy created
networkpolicy.networking.k8s.io/argocd-repo-server-network-policy created
networkpolicy.networking.k8s.io/argocd-server-network-policy created

C:\Users\josel>kubectl port-forward svc/argocd-server -n argocd 8080:443
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
|
```

- Obtener la contraseña inicial del usuario admin: Iniciamos Git Bash en Windows y ejecutamos el siguiente comando:

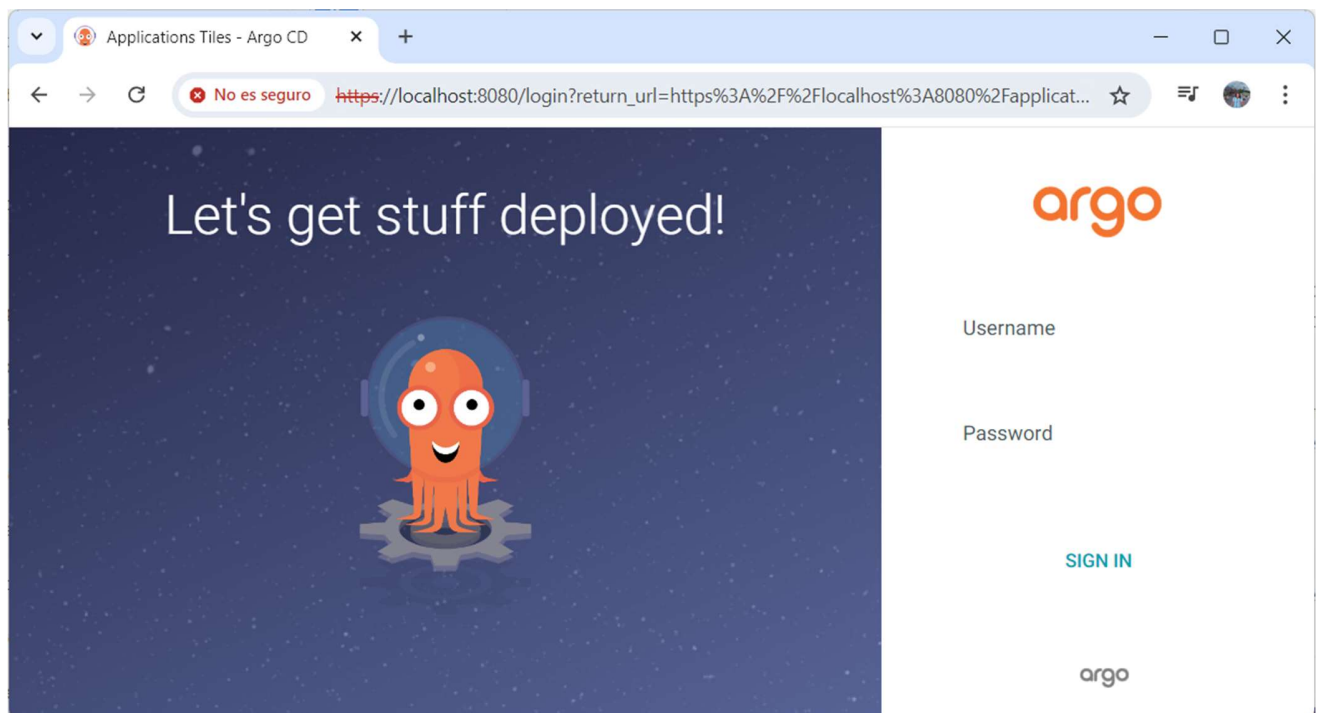
```
$ kubectl get secret argocd-initial-admin-secret -n argocd -o jsonpath="{.data.password}" | base64 --decode
```



```
MINGW64:/c/Users/josel
josel@DESKTOP-91PG8GC MINGW64 ~
$ kubectl get secret argocd-initial-admin-secret -n argocd -o jsonpath="{.data.password}" | base64 --decode
wjVwt4cwQuPZZy5b
josel@DESKTOP-91PG8GC MINGW64 ~
$ |
```

Ahora vamos a nuestro navegador, escribimos la ruta:

<https://localhost:8080/>



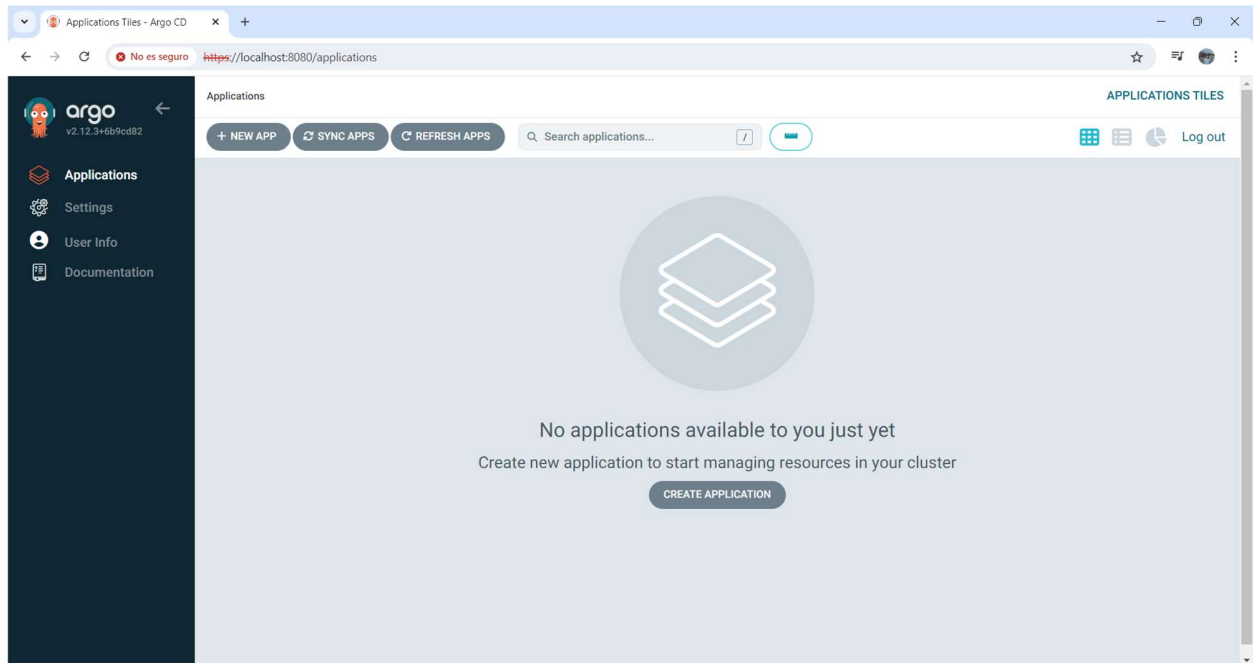
Agregaremos los siguientes valores:

Usuario: Admin

Password: wjVwt4cwQuPZZy5b

Entraremos a la interfaz de Argo y se mostrara la siguiente pantalla:

- Aplicaciones
- Configuraciones
- Info de usuario
- Documentación



5. Conectar ArgoCD con un repositorio Git:

- Configurar una aplicación en ArgoCD conectando un repositorio Git que contenga los manifiestos necesarios para desplegar una aplicación en Minikube.

Ejemplo1: Deployment con un servicio:

Utilizaremos el siguiente archivo YAML que nos servirá como definición de una aplicación en ArgoCD

```
! example-app.yaml X
Desafio13-14 > Desafio13 > ! example-app.yaml > {} spec
  io.argoproj.v1alpha1.Application (v1alpha1@application.json)
  1  apiVersion: argoproj.io/v1alpha1
  2  kind: Application
  3  metadata:
  4    name: guestbook
  5    namespace: argocd
  6  spec:
  7    source:
  8      path: guestbook
  9      repoURL: https://github.com/argoproj/argocd-example-apps.git
10      targetRevision: HEAD
11    destination:
12      server: 'https://kubernetes.default.svc'
13      namespace: kube-system
14      project: default
15
```

Explicación del YAML

1. apiVersion:

- argoproj.io/v1alpha1: Esta línea indica la versión de la API que se está utilizando para definir la aplicación en ArgoCD. En este caso, se está utilizando una versión alpha de la API de ArgoCD.

2. kind:

- **Application:** Este campo especifica el tipo de objeto que se está definiendo, que en este caso es una aplicación de ArgoCD.

3. **metadata:**

- Este bloque contiene información sobre la aplicación.
 - **name:** guestbook : Este es el nombre de la aplicación. En este caso, se llama "guestbook".
 - **namespace:** argocd: Este es el espacio de nombres en Kubernetes donde se encuentra la instancia de ArgoCD. Aquí, se especifica que la aplicación estará en el espacio de nombres "argocd". Es importante que siempre es este nombre.

4. **spec:**

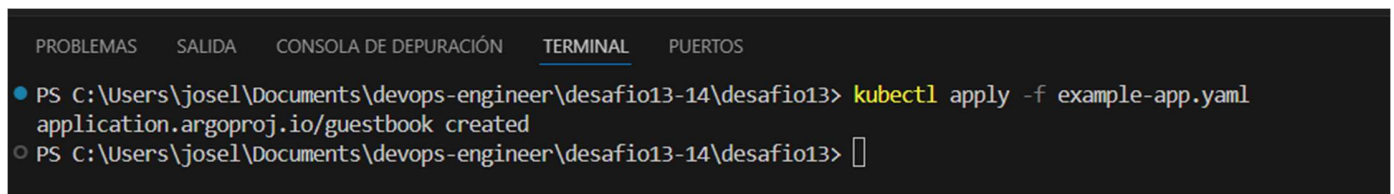
- Este bloque define las especificaciones de la aplicación.
- **source:**
 - Este subbloque define la fuente del código y los manifiestos necesarios para desplegar la aplicación.
 - **path:** guestbook: Indica la ruta dentro del repositorio donde se encuentran los manifiestos de Kubernetes para esta aplicación.
 - **repoURL:** <https://github.com/argoproj/argocd-example-apps.git> : Esta es la URL del repositorio Git donde están almacenados los archivos de configuración de la aplicación. En este caso, apunta a un repositorio público en GitHub que contiene ejemplos de aplicaciones para ArgoCD.
 - **targetRevision:** HEAD: Especifica qué revisión del repositorio se debe utilizar. HEAD significa que se utilizará la última confirmación en la rama principal.
- **destination:**
 - Este subbloque define dónde se desplegará la aplicación.
 - **server:** 'https://kubernetes.default.svc' : Esta es la dirección del servidor API de Kubernetes al que ArgoCD se conectará para desplegar los recursos. La dirección proporcionada es estándar para acceder al servidor API dentro del clúster de Kubernetes.
 - **namespace:** kube-system: Especifica el espacio de nombres en Kubernetes donde se desplegarán los recursos definidos en los manifiestos.
- **project:**
 - **default:** Este campo indica el proyecto dentro de ArgoCD al que pertenece esta aplicación. Los proyectos permiten agrupar aplicaciones y gestionar permisos y configuraciones comunes.

Contexto Adicional

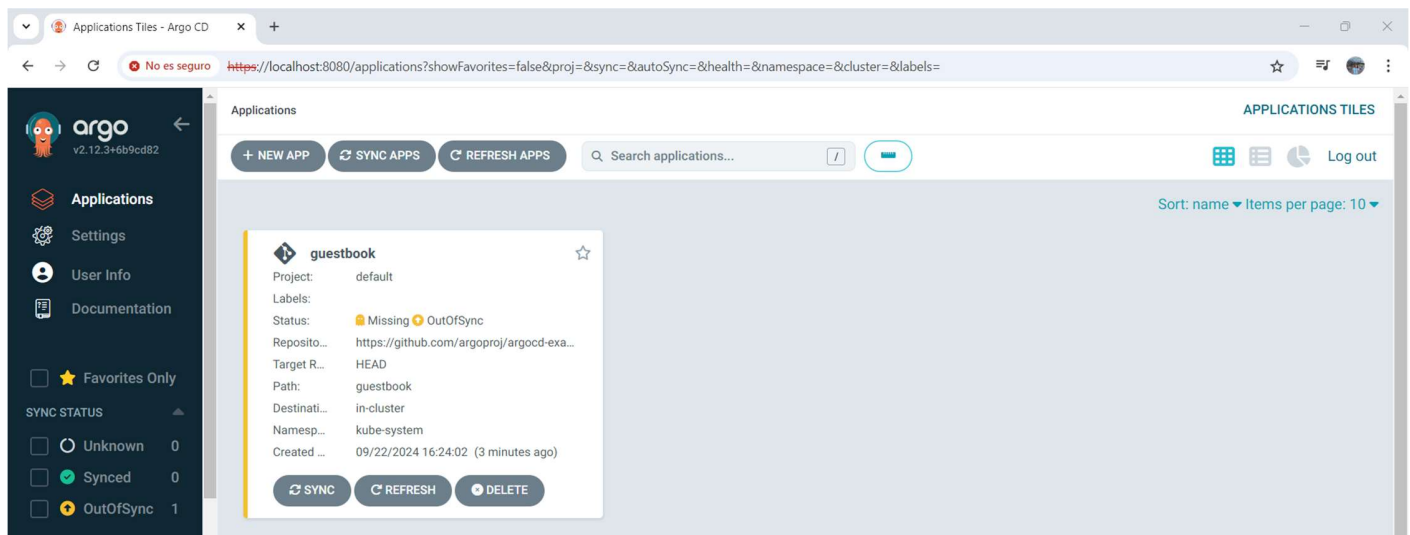
El repositorio mencionado (<https://github.com/argoproj/argocd-example-apps.git>) contiene varios ejemplos de aplicaciones, incluyendo una llamada "guestbook". Esta aplicación es un ejemplo simple que ilustra cómo utilizar ArgoCD para gestionar despliegues en Kubernetes mediante GitOps, donde los cambios en el repositorio Git desencadenan actualizaciones automáticas en el clúster. En resumen, este YAML configura una aplicación llamada "guestbook" que utiliza un conjunto de manifiestos almacenados en un repositorio Git para ser desplegada dentro del clúster Kubernetes gestionado por ArgoCD.

Ahora bien, una vez que hemos entendido la estructura de nuestro archivo, nos ubicamos en el directorio git en donde se encuentra el mismo y ejecutamos el siguiente comando:

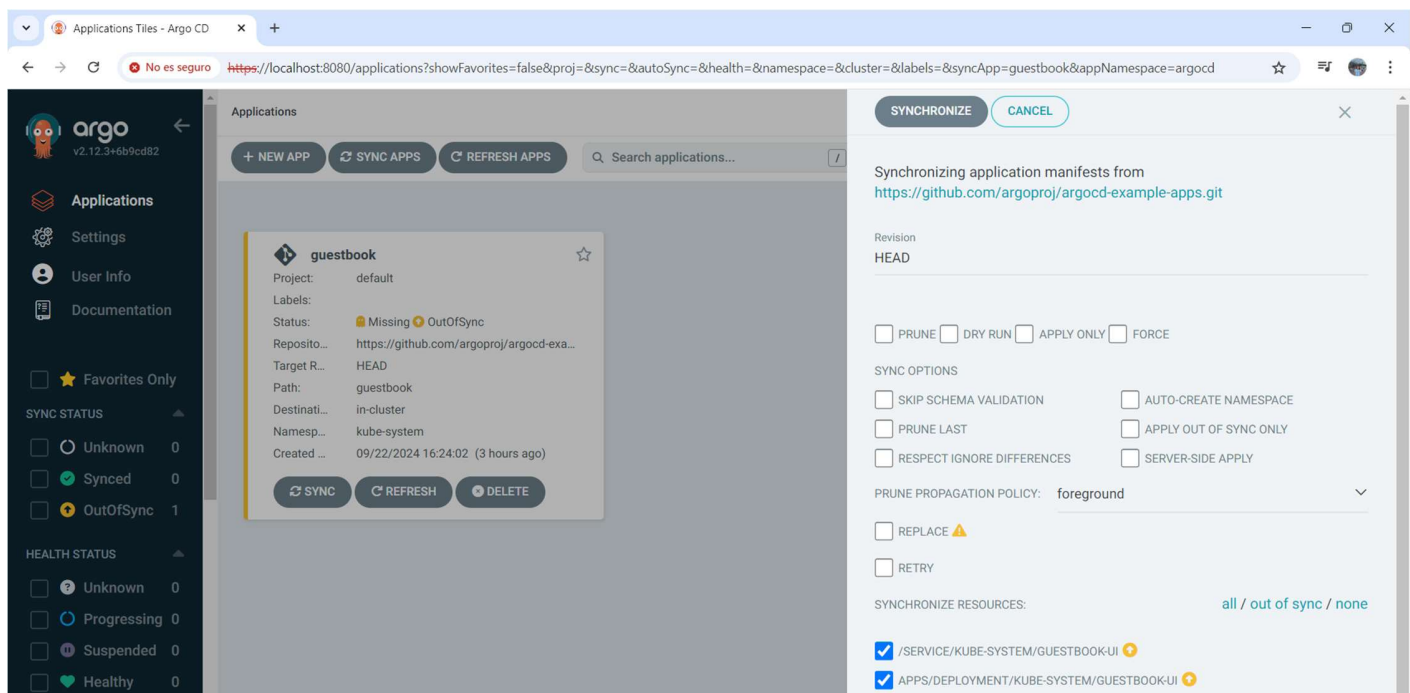

```
>kubectl apply -f example-app.yaml
```



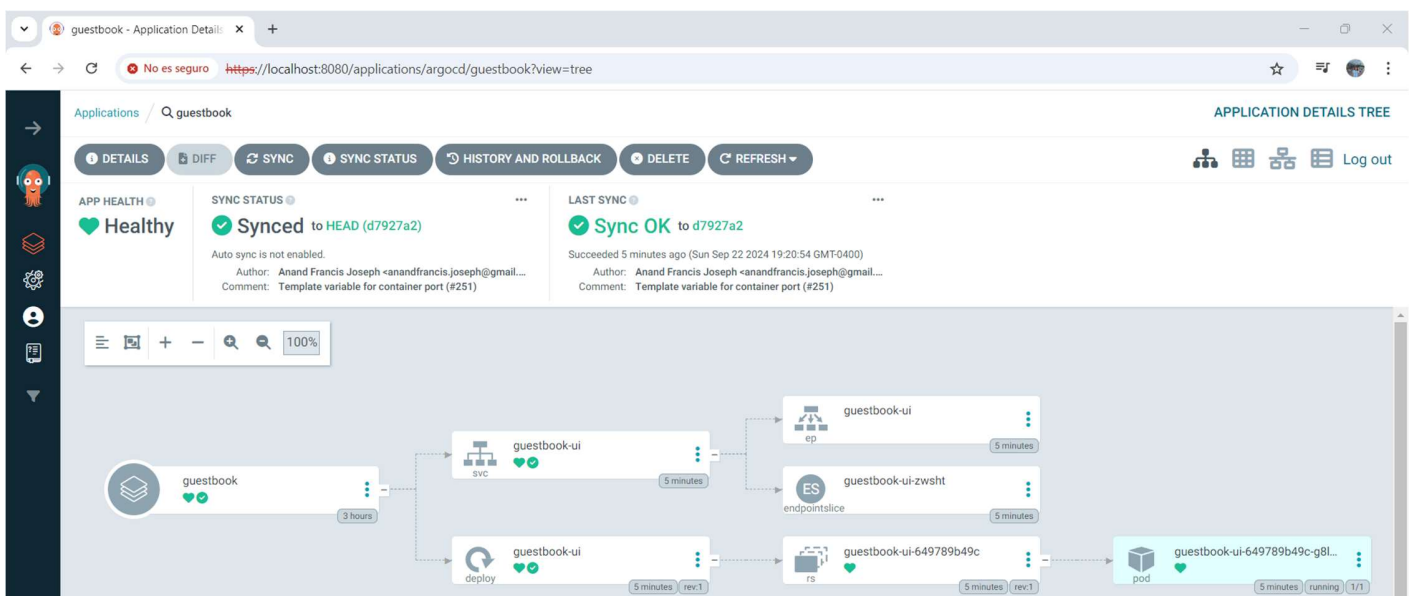
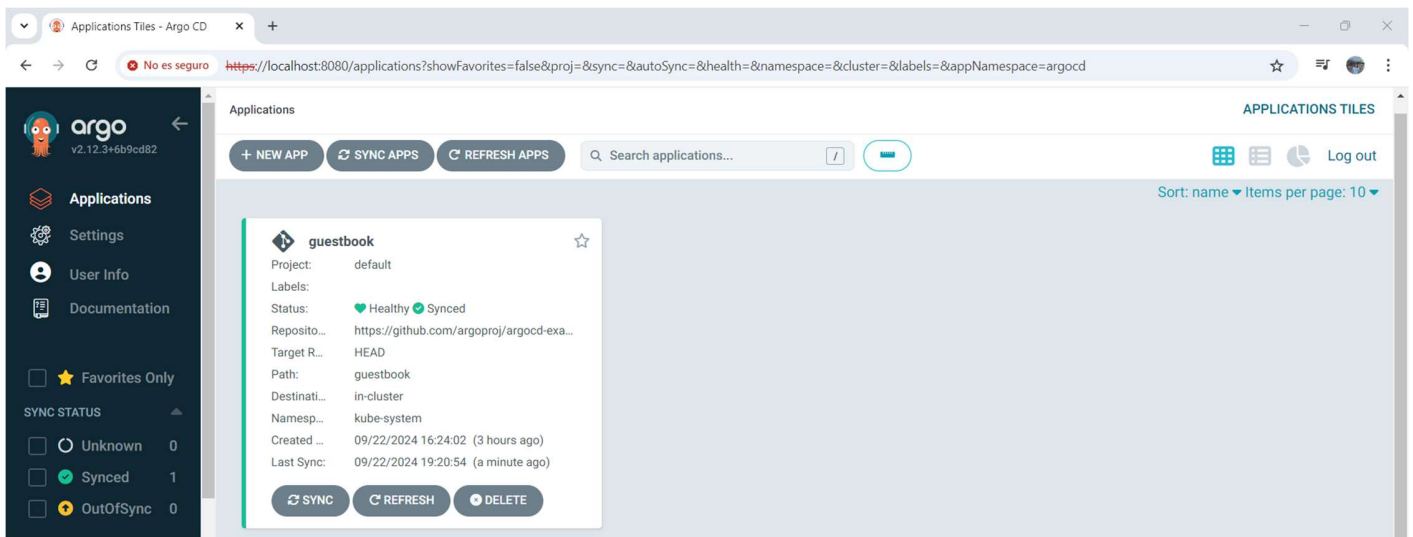
Automáticamente Argo CD ejecuta en su interfaz



Presionamos el Botón SYNC, para que se creen los recursos.



Luego presionamos SYNCHRONIZE (siempre hay que hacerlo)



Veremos los 2 Manifiestos diagramados:

- Servicio
 - o EndPoints
- Deployment
 - o Replica Sets
 - Pod

Ejemplo 2: Crearemos una aplicación sencilla nginx para verificar despliegue

Crearemos una aplicación sencilla nginx (example-nginx)

Los tres archivos YAML proporcionados a continuación son esenciales para desplegar una aplicación NGINX en un clúster de Kubernetes utilizando ArgoCD, permitiendo la creación organizada y eficiente de recursos mediante.

```

PS C:\Users\josel\Documents\devops-engineer\desafio13-14\desafio13\example-nginx> tree /f
Listado de rutas de carpetas
El número de serie del volumen es 8C2E-47A9
C:.\
├── readme.md
└── nginx
    ├── namespace.yaml
    ├── nginx.yaml
    └── service.yaml

```

namespace.yaml

```

! namespace.yaml X
Desafio13-14 > Desafio13 > example-nginx > nginx > ! namespace.yaml > {} metadata > name
1  apiVersion: v1
2  kind: Namespace
3  metadata:
4    name: test

```

- **apiVersion: v1:** Especifica la versión de la API de Kubernetes que se está utilizando.
- **kind: Namespace:** Indica que el objeto que se está definiendo es un espacio de nombres (namespace).
- **metadata:** Contiene información sobre el objeto.
 - **name: test:** Define el nombre del espacio de nombres como "test". Los namespaces son útiles para organizar recursos y evitar conflictos de nombres entre diferentes aplicaciones o entornos.

service.yaml

```

! service.yaml ●
Desafio13-14 > Desafio13 > example-nginx > nginx > ! service.yaml > {} spec > type
io.k8s.api.core.v1.Service (v1@service.json)
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: nginx-service
5    namespace: test
6  spec:
7    selector:
8      app: nginx
9    ports:
10   - name: http
11     port: 80
12     targetPort: 80
13   type: ClusterIP

```

- **apiVersion: v1**: Versión de la API utilizada para definir el servicio.
- **kind: Service**: Define un objeto Service, que permite acceder a los pods mediante una dirección IP estable y un nombre DNS.
- **metadata**:
 - **name: nginx-service**: Nombre del servicio.
 - **namespace: test**: Especifica el namespace donde se creará el servicio.
- **spec**:
 - **selector**:
 - **app: nginx**: Selecciona los pods con la etiqueta "app" igual a "nginx".
 - **ports**:
 - **name: http**: Nombre del puerto del servicio.
 - **port: 80**: Puerto en el cual el servicio estará disponible dentro del clúster.
 - **targetPort: 80**: Puerto en el cual los pods están escuchando (el mismo puerto que el contenedor).
 - **type: ClusterIP**: Tipo de servicio que proporciona una dirección IP interna accesible solo dentro del clúster.

nginx.yaml

```
! nginx.yaml ●
Desafio13-14 > Desafio13 > example-nginx > nginx > ! nginx.yaml > ...
io.k8s.api.apps.v1.Deployment (v1@deployment.json)
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5    namespace: test
6  spec:
7    replicas: 1
8    selector:
9      matchLabels:
10       app: nginx
11  template:
12    metadata:
13      labels:
14       app: nginx
15  spec:
16    containers:
17     - name: nginx
18       image: nginx:latest
19       ports:
20        - containerPort: 80
21       resources:
22         requests:
23           memory: "128Mi" # Solicitud mínima de memoria
24           cpu: "100m" # Solicitud mínima de CPU
25         limits:
26           memory: "256Mi" # Límite máximo de memoria
27           cpu: "200m" # Límite máximo de CPU
28
```

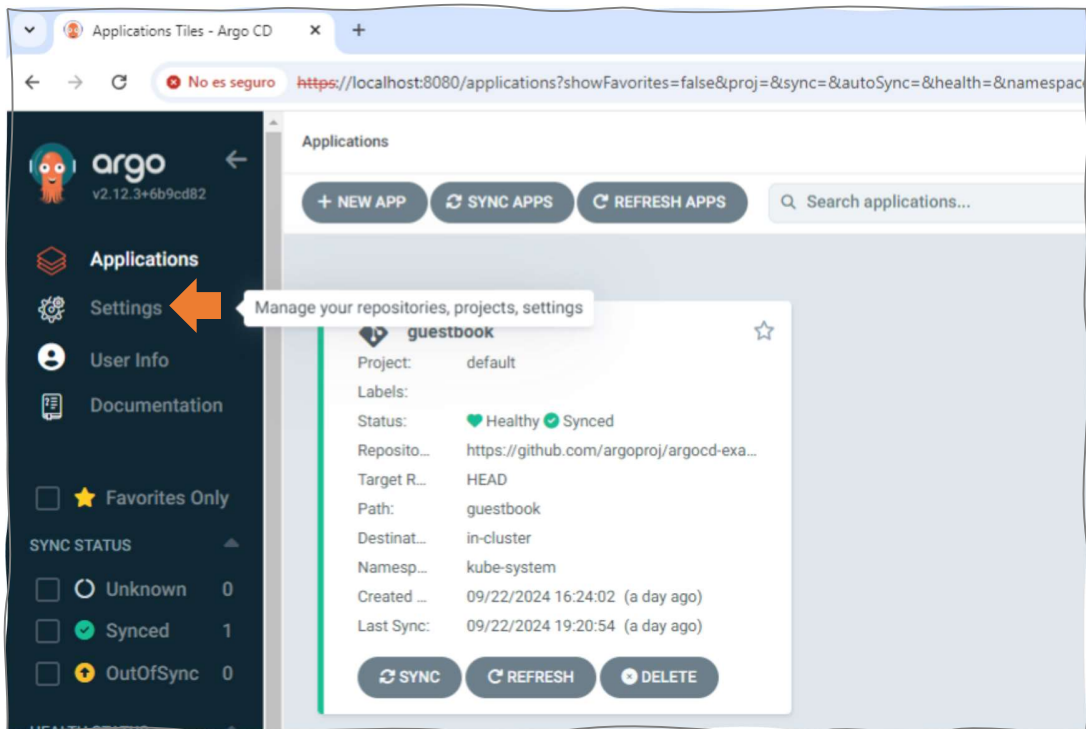
- **apiVersion: apps/v1:** Indica que se está utilizando la versión de la API para aplicaciones.
- **kind: Deployment:** Define un objeto de tipo Deployment, que gestiona la creación y actualización de instancias de pods.
- **metadata:**
 - **name: nginx-deployment:** Nombre del Deployment.
 - **namespace: test:** Especifica el namespace donde se desplegará el Deployment.
- **spec:**
 - **replicas: 1:** Indica que se desea mantener una réplica del pod en ejecución.
 - **selector:**
 - **matchLabels:** Utiliza etiquetas para seleccionar los pods que forman parte de este Deployment.
 - **app: nginx:** La etiqueta utilizada para identificar los pods.
 - **template:**
 - **metadata:**
 - **labels:** Define las etiquetas que se asignarán a los pods creados por este Deployment.
 - **spec:**
 - **containers:**
 - **name: nginx:** Nombre del contenedor.
 - **image: nginx:latest:** Imagen del contenedor, en este caso, la última versión de NGINX.
 - **ports:**
 - **containerPort: 80:** Especifica que el contenedor escuchará en el puerto 80.

Las siguientes configuraciones, mejoran la gestión de recursos en el clúster de Kubernetes, asegurando que la aplicación NGINX funcione correctamente sin interferir con otros procesos

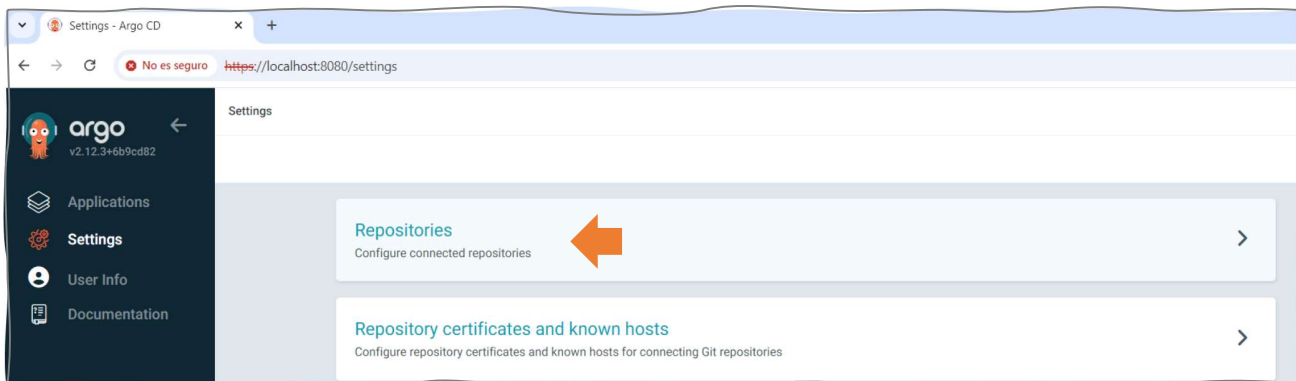
- **resources:** Esta sección se utiliza para definir las solicitudes y límites de recursos para el contenedor.
 - **requests:** Especifica la cantidad mínima de recursos que el contenedor necesita para funcionar. Kubernetes garantiza que estos recursos estén disponibles para el contenedor.
 - **memory:** Aquí se establece una solicitud mínima de memoria de 128 MiB.
 - **cpu:** Se establece una solicitud mínima de CPU de 100 milicpus (0.1 CPU).
 - **limits:** Define la cantidad máxima de recursos que el contenedor puede consumir. Si el contenedor intenta usar más allá de este límite, Kubernetes lo restringirá.
 - **memory:** Se establece un límite máximo de memoria de 256 MiB.
 - **cpu:** Se establece un límite máximo de CPU de 200 milicpus (0.2 CPU).

Una vez que hemos creado los manifiestos y los tengamos guardados en un repositorio de Github, podremos conectarlos a ArgoCD mediante la interfaz gráfica. Para realizar dicha operación, entraremos a la interfaz de ArgoCD y seguiremos los siguientes pasos:

1. Hacemos clic en Settings



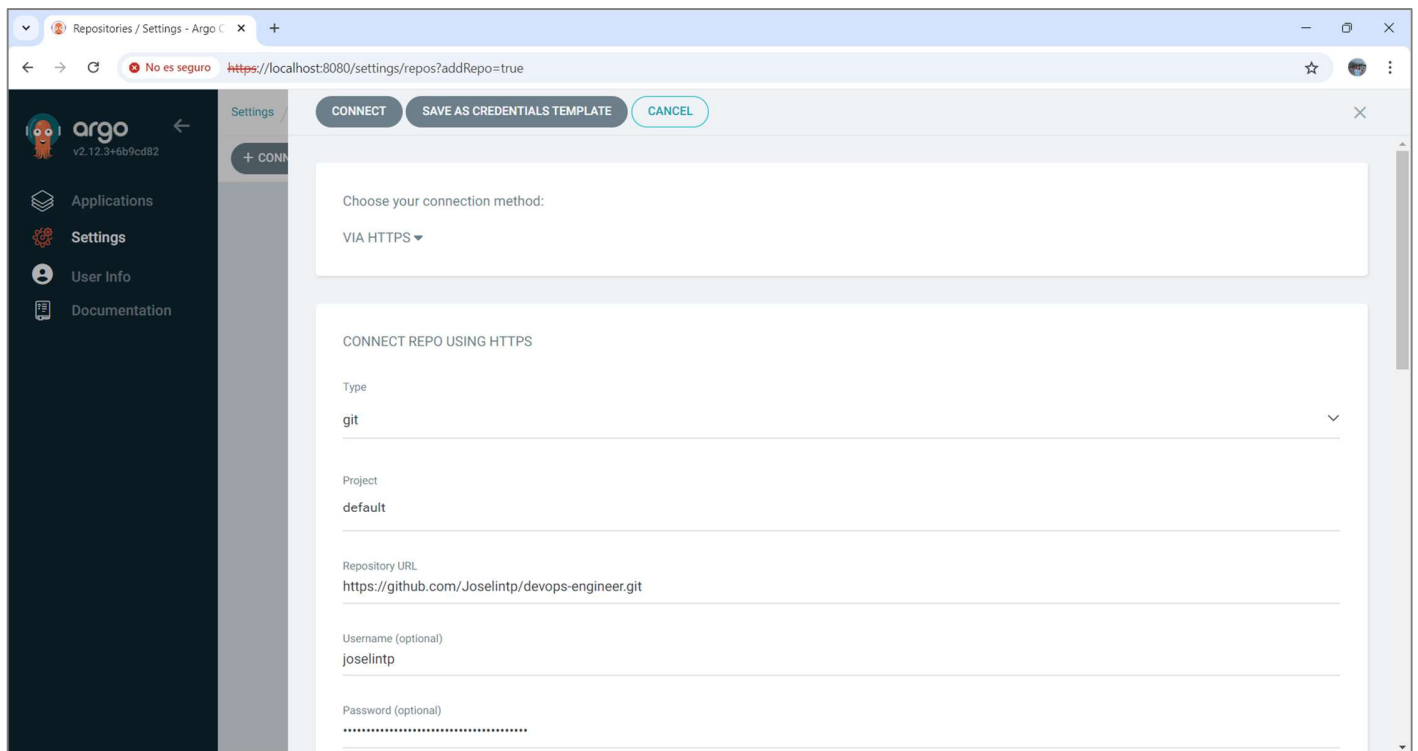
2. Hacemos clic en repositorios



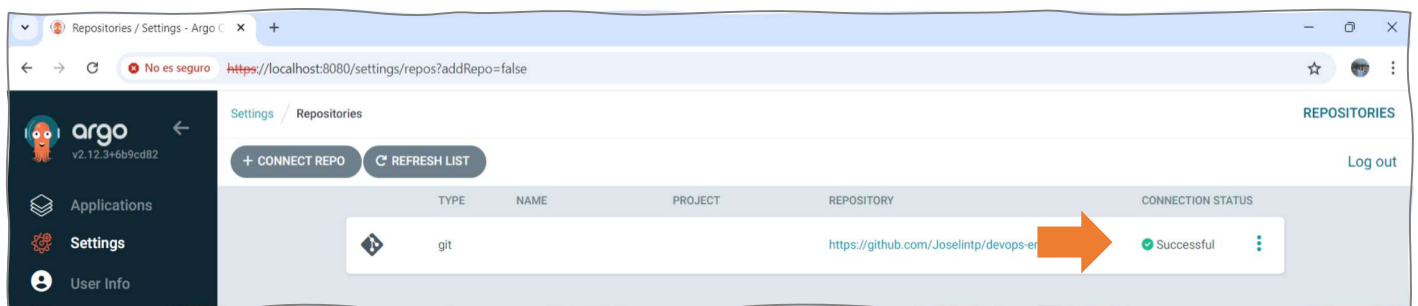
3. Ahora haremos clic en el botón (CONNECT REPO/CONECTAR REPOSITORIO)



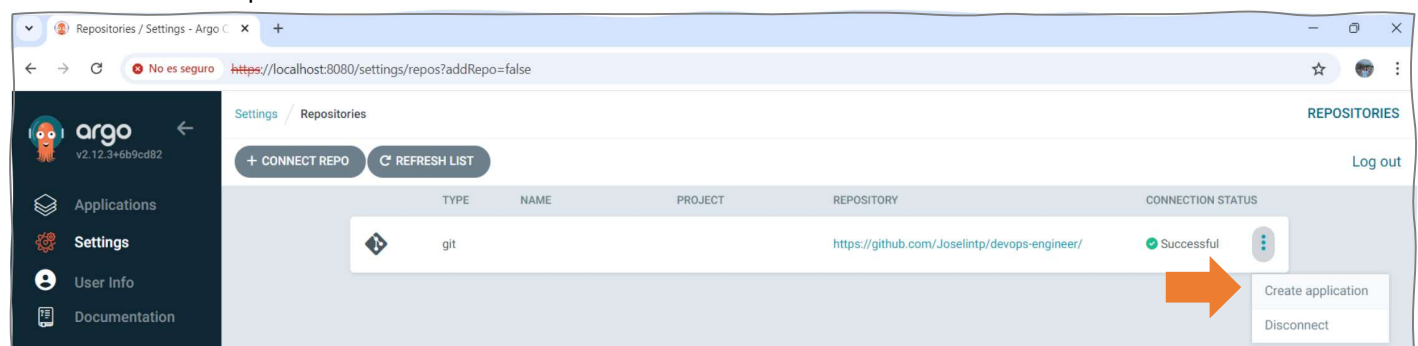
4. En la siguiente pantalla colocaremos para el método de conexión, HTTPS, en Tipo: Git, en Proyecto colocamos predeterminado e ingresamos nuestro repositorio de GitHub como URL del repositorio. Finalmente, presionamos Conectar.



5. Deberíamos visualizar el estado de la conexión como exitosa.

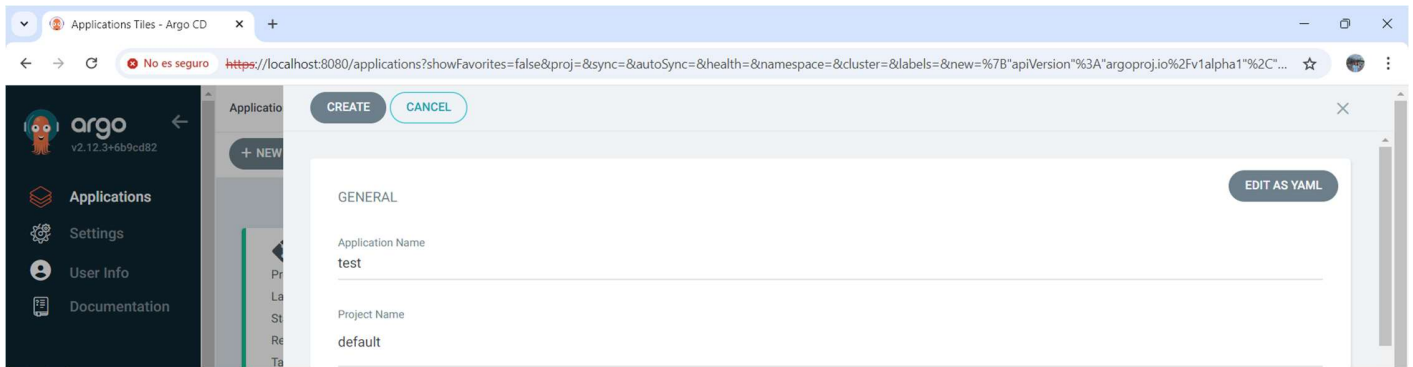


6. Ahora vamos a crear una nueva aplicación desde el repositorio. En el extremo derecho del [Successful] mensaje, hacemos clic en los tres puntos. El menú emergente tendrá una opción [Create application]. Haremos clic también en dicha opción.

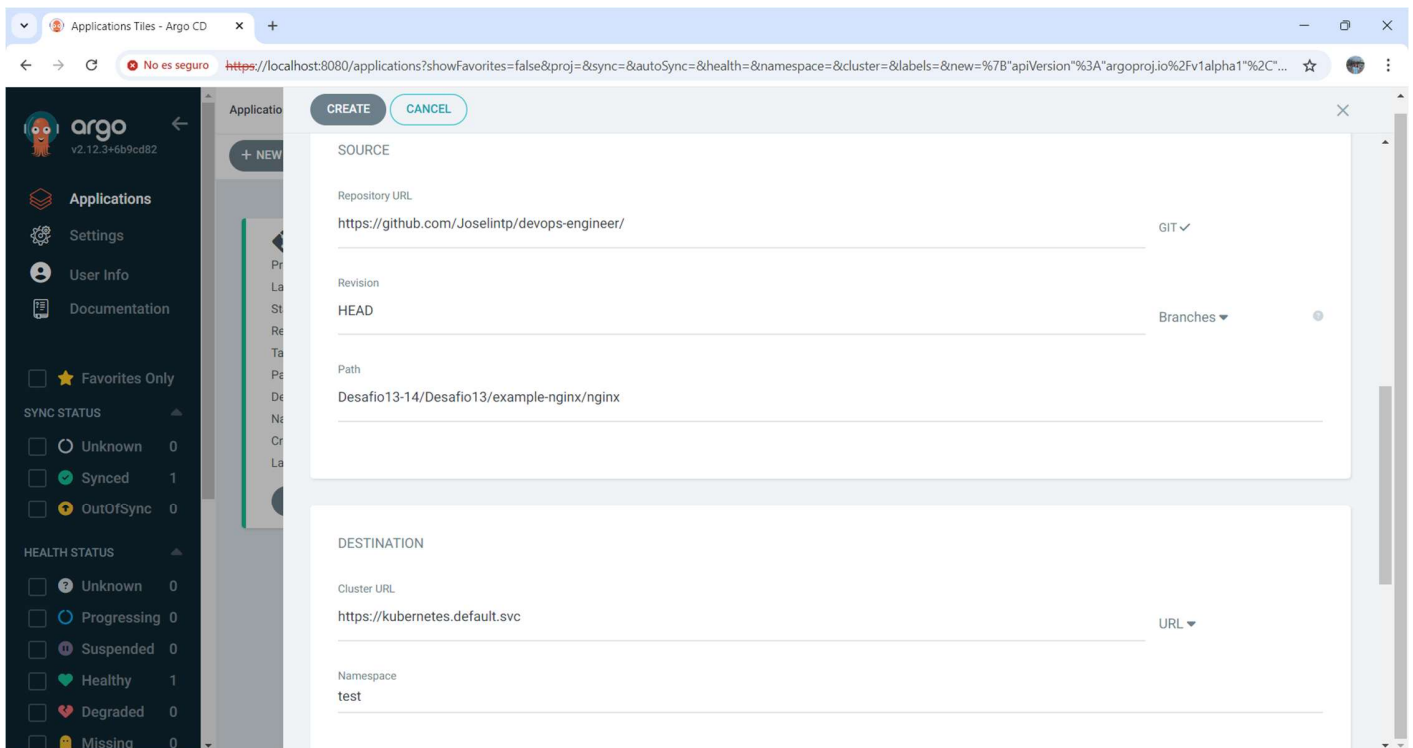


7. Ingresaremos los siguientes datos:

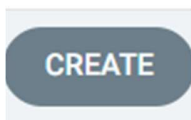
- Application Name: test
- Project Name: Escogemos default

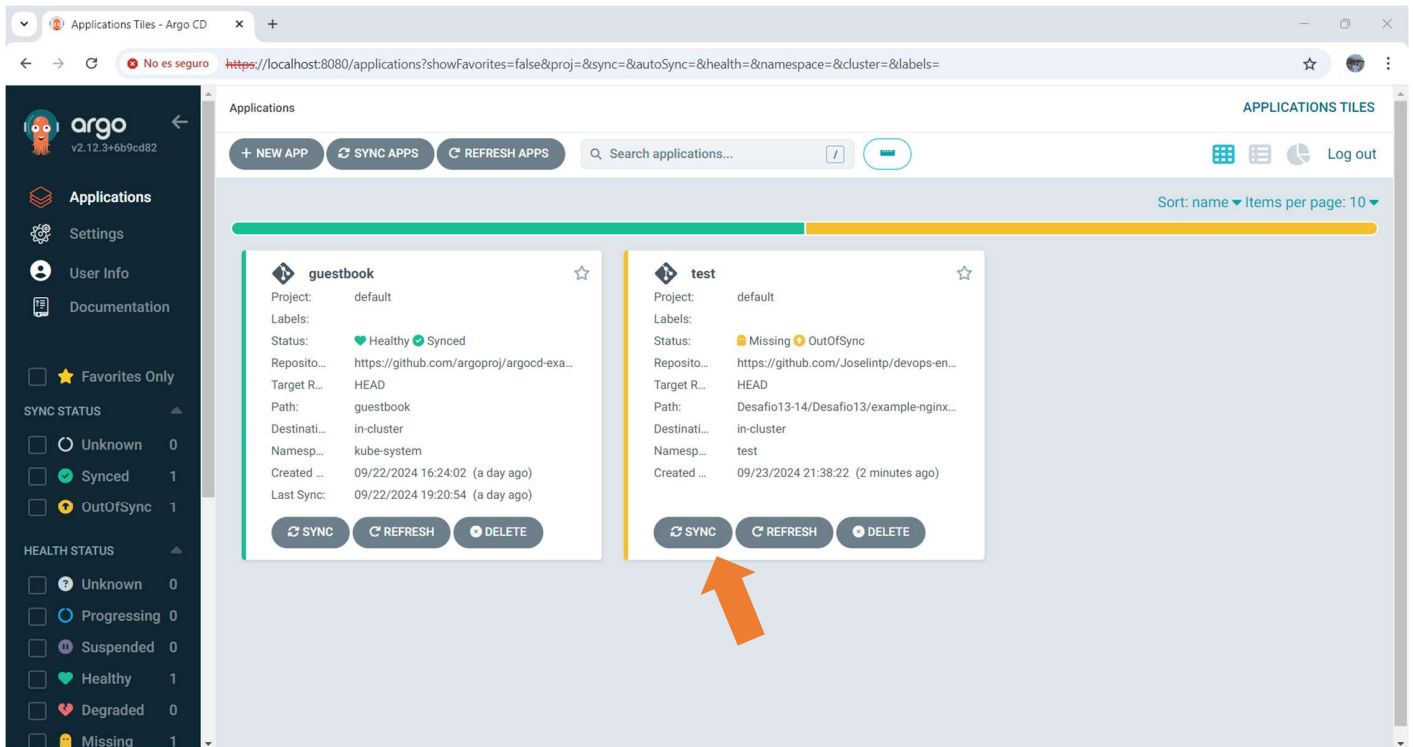


- Repository URL (no es necesario escribirlo, se mostrará): <https://github.com/Joselintp/devops-engineer/>
- Path (Es la ruta donde están los manifiestos): [Desafio13-14/Desafio13/example-nginx/nginx](#)
- Cluster URL, seleccionamos: <https://kubernetes.default.svc>
- Namespace: [test](#)

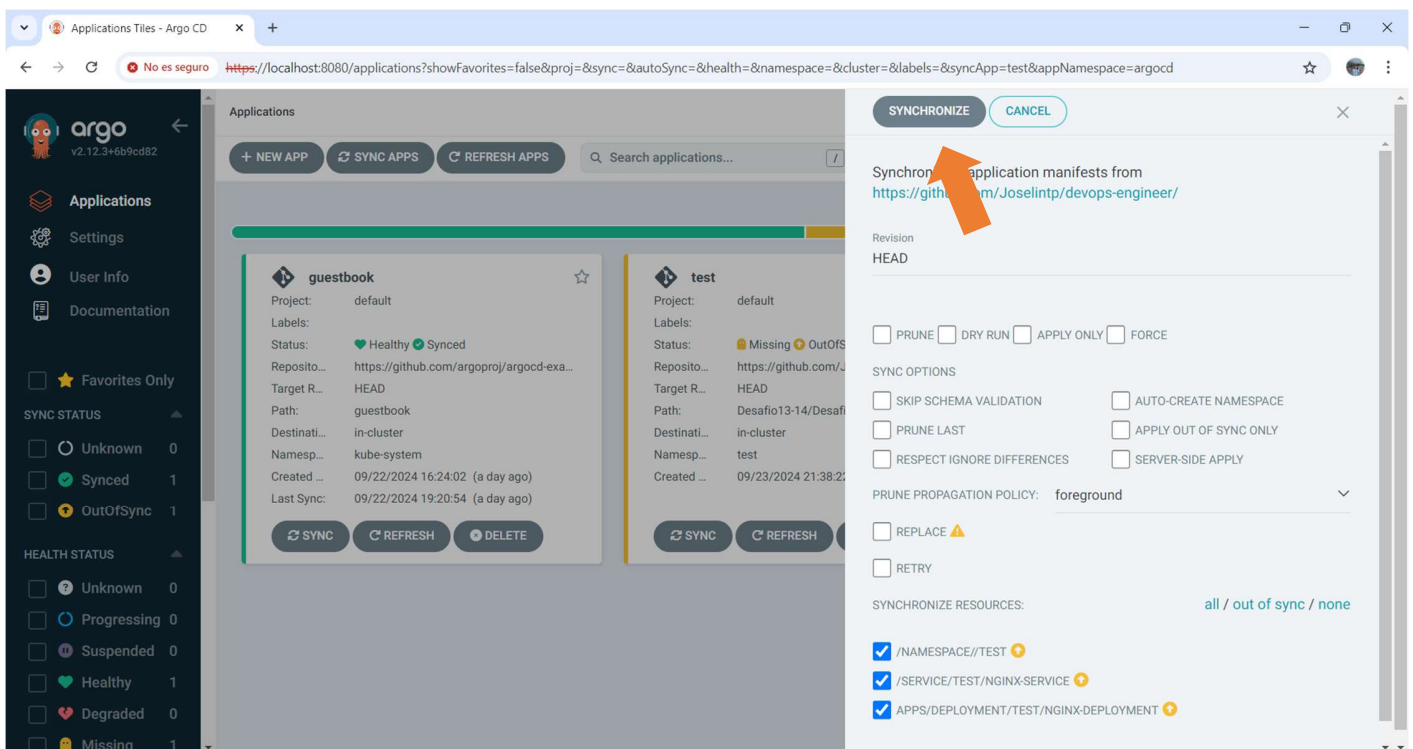


Presionamos el Botón Create.

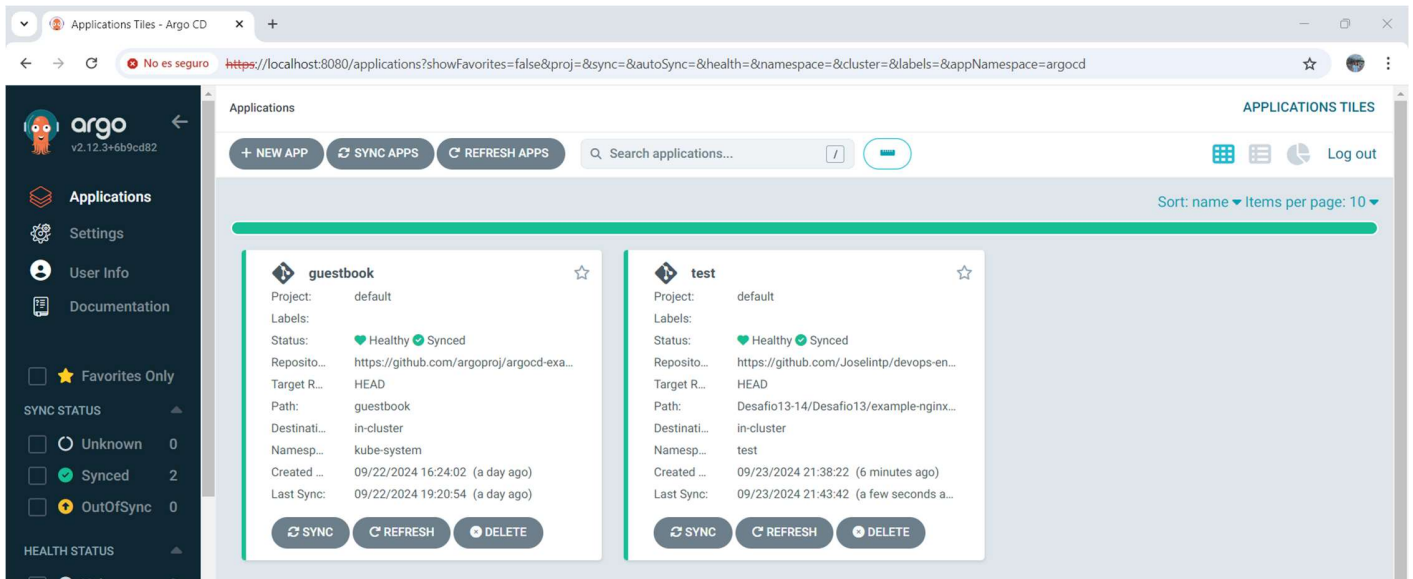




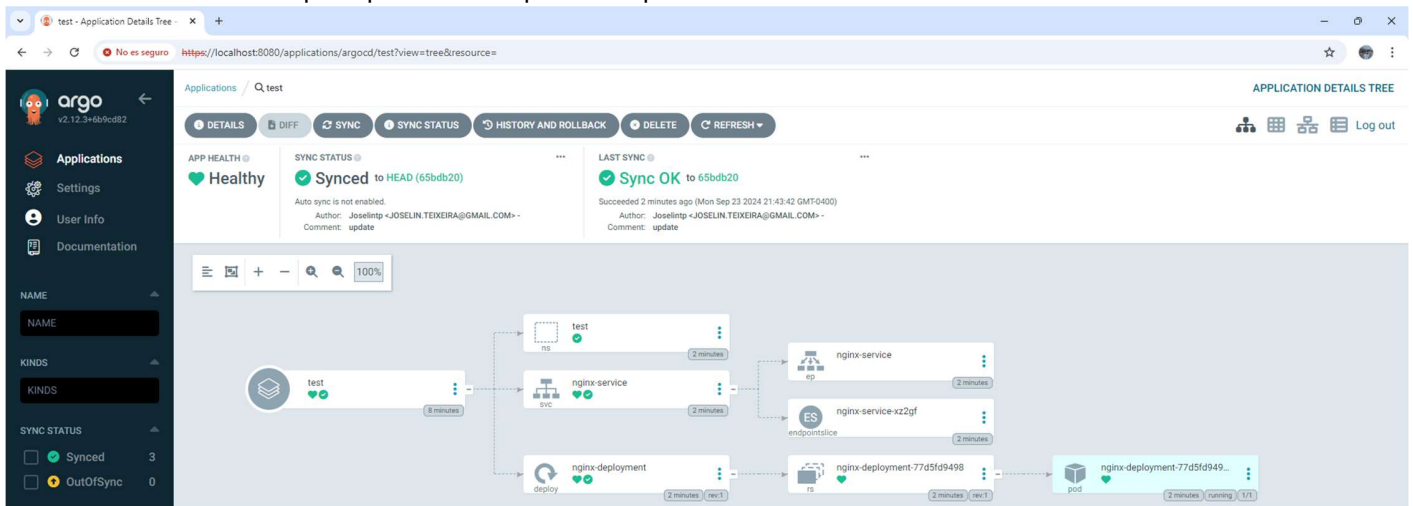
Como especificamos SYNC Manual, debemos presionar dicho botón para sincronizar.



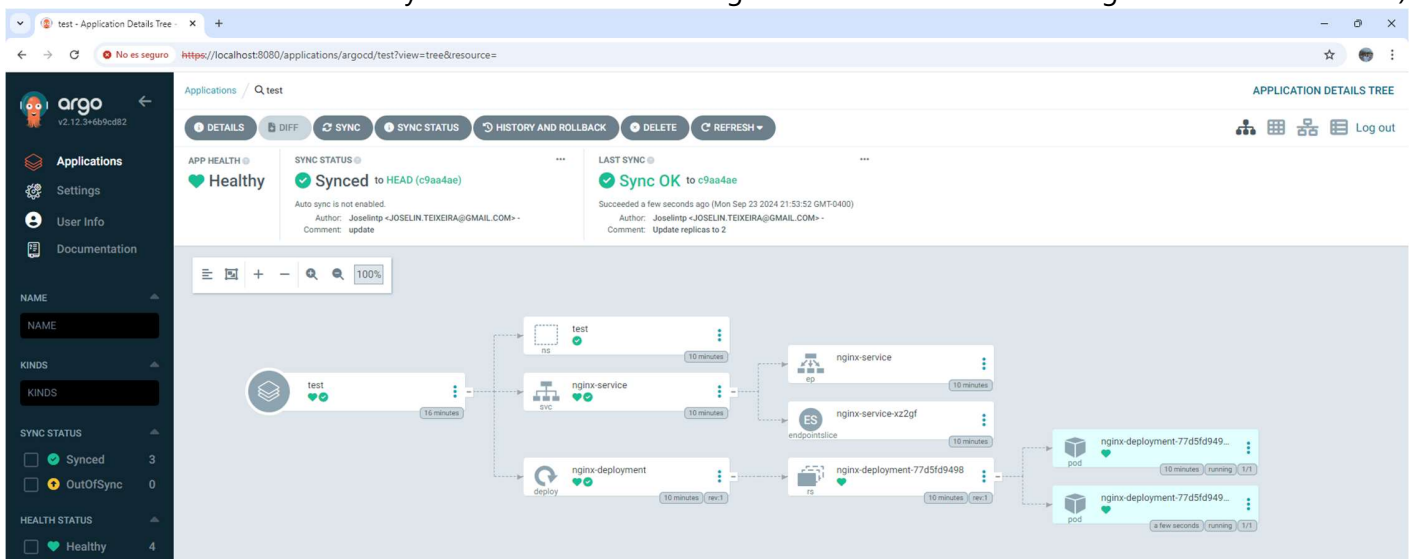
Ya tenemos nuestra aplicación test.



Si hacemos clic en cualquier parte de la aplicación podremos obtener una vista detallada de la misma:



Si editamos nginx.yaml, y colocamos como valor de replicas 2, al sincronizar agregara el 2 pod. (Podemos colocar el valor de SYNC automático y una vez modificado alguno de los manifiestos se generarán los cambios)



Desafío 14 - Despliegue HelmChart ArgoCD

Objetivo:

El objetivo de este desafío es poner en práctica el despliegue del Helm Chart desarrollado en el desafío #12 utilizando ArgoCD. Se debe automatizar la gestión del despliegue de la aplicación y su base de datos MongoDB, siguiendo los principios de GitOps.

1. Configurar ArgoCD:

- Conectar ArgoCD a un repositorio Git que contenga el Helm Chart desarrollado en el desafío #12.
- Configurar una aplicación en ArgoCD para gestionar el despliegue de dicho Helm Chart.

2. Automatización de despliegues:

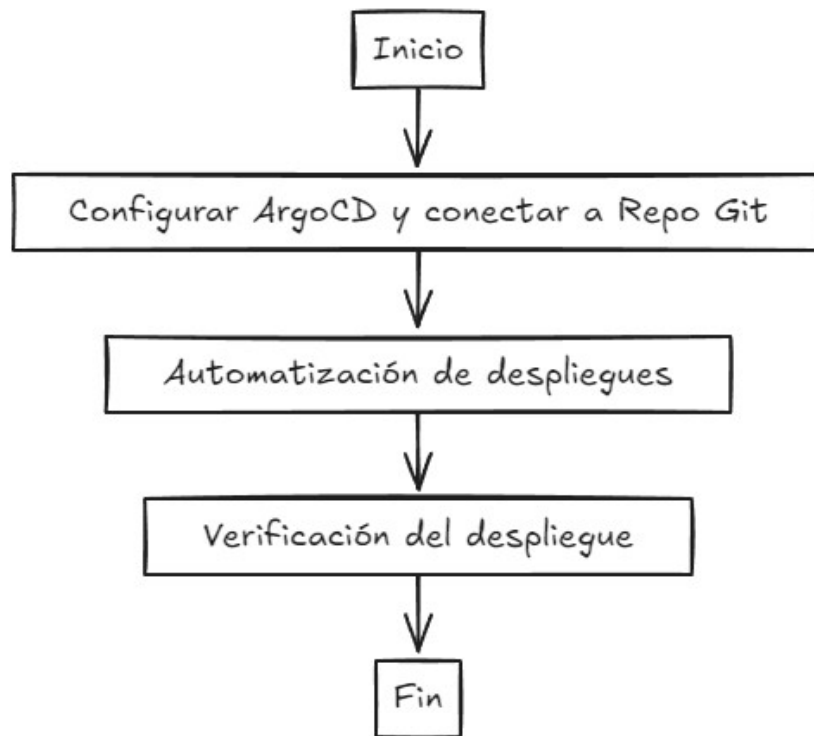
- Implementar la sincronización automática de ArgoCD para que los cambios realizados en el repositorio Git se reflejen automáticamente en el clúster.

3. Verificación del despliegue:

- Validar que el Helm Chart se despliega correctamente utilizando ArgoCD y que la aplicación y MongoDB están operativos.



Diagrama de flujo



1. **Crearemos una nueva aplicación** desde Settings → Repositories → Create application
2. **Completamos los siguientes campos:**
 - **Nombre de la aplicación:** educacionit-app
 - **Proyecto:** default
 - **Política de sincronización:** elegimos **sincronización automática**

A screenshot of the ArgoCD 'Create application' form. The form has a 'CREATE' button and a 'CANCEL' button at the top left. The main section is titled 'GENERAL' and contains two input fields: 'Application Name' with the value 'educacionit-app' and 'Project Name' with the value 'default'. To the right of the 'GENERAL' section is a button labeled 'EDIT AS YAML'. Below the 'GENERAL' section is a section titled 'SYNC POLICY' with the value 'Automatic' and a dropdown arrow.

3. **Configuración de origen:**
 - **URL del repositorio:** proporcionamos la URL del repositorio Git donde se almacenamos nuestro gráfico Helm.
<https://github.com/Joselintp/devops-engineer/>
 - **Ruta:** especificamos la ruta a al gráfico Helm dentro del repositorio.
[Desafio11-12/educacionit-app/educacionit-app-chart](#)
 - **Revisión de destino:** establecemos una rama/etiqueta específica. **HEAD**

4. Configuración de destino:

- **URL del clúster:** Dejamos como: <https://kubernetes.default.svc>
- **Espacio de nombres:** especificamos el espacio de nombres donde deseamos realizar la implementación [kube-system](#)

The screenshot shows a 'CREATE' dialog box with two main sections: 'SOURCE' and 'DESTINATION'. The 'SOURCE' section includes fields for 'Repository URL' (https://github.com/Joselintp/devops-engineer/), 'Revision' (HEAD), and 'Path' (Desafio11-12/educacionit-app/educacionit-app-chart). The 'DESTINATION' section includes fields for 'Cluster URL' (https://kubernetes.default.svc) and 'Namespace' (kube-system). There are 'CREATE' and 'CANCEL' buttons at the top left, and a close button at the top right.

5. Configuración de Helm (automáticamente toma la configuración de Helm)

The screenshot shows a 'Helm' configuration dialog box. It has a 'Helm' dropdown menu at the top left. The main section is titled 'HELM' and contains several fields: 'VALUES FILES', 'VALUES', and 'PARAMETERS'. The 'PARAMETERS' section is a table with the following values:

Parameter	Value
autoscaling.enabled	false
autoscaling.maxReplicas	100
autoscaling.minReplicas	1
autoscaling.targetCPUUtilizationPercentage	80
env.DB_NAME	app-desafio10
env.DB_PASS	Psw*7845!
env.DB_URI	mongodb://db-server:27017
env.DB_USER	root

6. Crear Aplicación: Presionamos el botón (Create) y la aplicación se sincronizará e implementará

Después de crear la aplicación, la veremos listada en el panel de control.

educacionit-app - Application

← → ↻ No es seguro https://localhost:8080/applications/argocd/educacionit-app?view=tree&resource=&conditions=true ☆

→ Applications / educacionit-app

DETAILS

DIFF

SYN

APP HEALTH

Progressing

SYNC

Auto sy

At

Com

educacionit-app

Application conditions

RepeatedResourceWarning

Resource /Pod/kube-system/educacionit-app-educacionit-app-chart-test-connection appeared 2 times among application resources.

a minute ago (Tue Sep 24 2024 21:14:50 GMT-0400)

RepeatedResourceWarning

Resource /Service/kube-system/educacionit-app-educacionit-app-chart appeared 2 times among application resources.

a few seconds ago (Tue Sep 24 2024 21:14:51 GMT-0400)

RepeatedResourceWarning

Resource /ServiceAccount/kube-system/educacionit-app-educacionit-app-chart appeared 2 times among application resources.

a few seconds ago (Tue Sep 24 2024 21:14:51 GMT-0400)

RepeatedResourceWarning

Resource apps/Deployment/kube-system/educacionit-app-educacionit-app-chart appeared 2 times among application resources.

a few seconds ago (Tue Sep 24 2024 21:14:51 GMT-0400)