

---

# Lab 21: Argo CD for Containerized Deployments

---

## Lab overview

In this lab activity, you will set up a GitOps workflow using Argo CD to automate the deployment and management of containerized applications in a Kubernetes cluster. You will organize your infrastructure manifests into a dedicated Git repository, install and expose Argo CD on your Kubernetes cluster, and configure Argo CD to watch your repository. Once set up, you will observe how Argo CD automatically deploys your application and self-heals when manual changes are made to the cluster.

In this lab, you will:

- Create a private GitHub repository for Kubernetes manifests
- Install Argo CD in a Kubernetes cluster and expose its dashboard
- Connect Argo CD to a GitHub repository using a fine-grained access token
- Configure an Argo CD application to automatically deploy from GitHub
- Simulate deployment drift and observe Argo CD's automated self-healing.

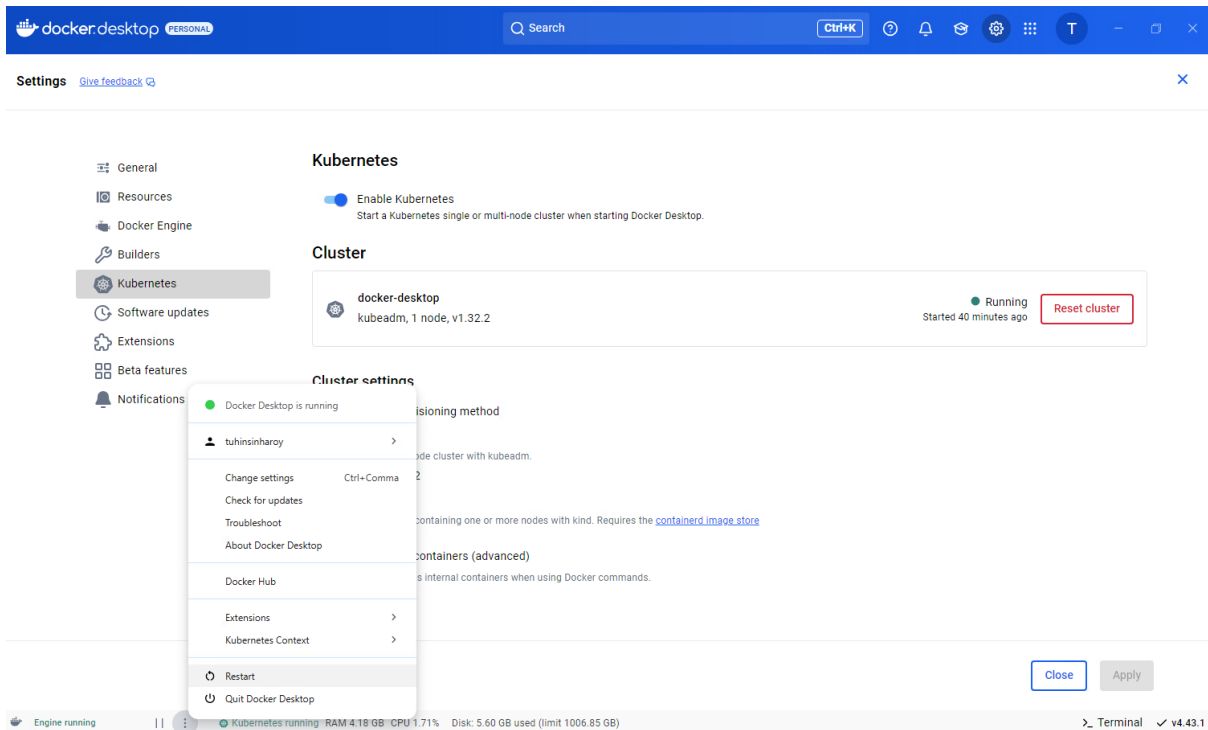
## Estimated completion time

50 minutes

## Task 1: Preparing your code

In this task, you will prepare a repository for infrastructure manifest files. Kubernetes manifests should be kept separate from application code to achieve logical separation between teams.

1. Check that Docker Desktop and the Kubernetes Service are running. If Kubernetes is still in Starting mode, restart the Docker Desktop and re-check.



2. When Kubernetes shows Running, click **Sign In** and enter your Docker credentials.
3. Go to <https://github.com> and create a new **private** repository (**DevOpsLab21**).  
**Do not** initialize a README file, **do not** add a .gitignore, and **do not** choose a license.
4. In your lab environment, create a folder with the name **DevOpsLab21**. Open it with VS Code and then under it, create the following directory structure and files.
  - apps
  - app/devopslab21
    - frontend-deployment.yaml
    - backend-deployment.yaml
    - kustomization.yaml

5. Paste in the following content in `apps/devopslab21/frontend-deployment.yaml`. Replace `<docker_username>` with your Docker Hub username. Notice the image version is not specified.

#### Note

Sample files are in the desktop folder Sample Lab Files/Lab21.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend-app
          image: <docker_username>/frontend
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
```

```
metadata:
  name: frontend-service
spec:
  selector:
    app: frontend
  ports:
    - port: 8080
      targetPort: 80
  type: LoadBalancer
```

6. Paste the following content inside **apps/devopslab21/backend-deployment.yaml**. Replace **<docker\_username>** with your Docker Hub username. Notice the image version is not specified.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
```

```

containers:
  - name: backend-app
    image: <docker_username>/backend
    ports:
      - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app: backend
  ports:
    - port: 80
      targetPort: 80
  type: ClusterIP

```

7. Create a file called **kustomization.yaml** inside **apps/devopslab21/** and add the following code to it. Replace **<docker\_username>** with your Docker Hub username.

```

resources:
  - frontend-deployment.yaml
  - backend-deployment.yaml

images:
  - name: <docker_username>/frontend
    newTag: v1

```

- name: <docker\_username>/backend  
newTag: v1

8. Commit and push these changes to GitHub.
9. From the **DevOpsLab21** folder (**Not** the apps/devopslab21 folder!), in VS Code, open a terminal and run the following.

```
git config --global user.email "<you@example.com>"
git config --global user.name "<Your Name>"
git init
git add .
git commit -m "Initial commit with k8s manifest"
git branch -M main
git remote add origin https://github.com/<username>/DevOpsLab21.git
git push -u origin main
```

10. If asked, sign in with your browser.

## Task 2: Installing Argo CD in Kubernetes

In this task, you will install Argo CD in your Kubernetes Cluster so that it can manage your deployments. Ideally, Argo CD should be installed in a separate **infrastructure** cluster but you will combine the application and Argo CD in the same cluster in this lab.

1. Reconnect to your Kubernetes Cluster and confirm connectivity. From a VS Code terminal, run.

```
kubectl config use-context docker-desktop
```

```
kubectl get nodes
```

2. Create a new namespace for Argo CD. It is better to group all Argo CD resources in a single namespace to prevent accidental tampering.

```
kubectl create namespace argocd
```

3. Install Argo CD.

#### Note

Use the Type Text tool in the Lab 'Instructions' tab to help with command entry.

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

4. Expose Argo CD publicly so that you can access the dashboard. Ideally, such access should be configured from behind a VPN only.

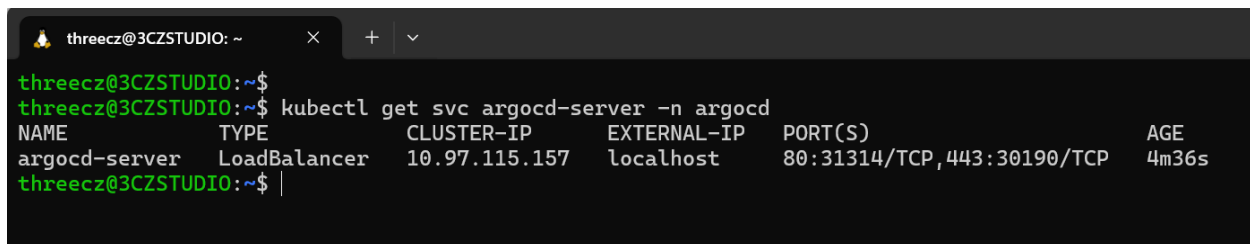
```
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type":
"LoadBalancer"}}'
```

5. Get and copy the default Argo CD dashboard password which is stored as an auto-generated, clear text secret in Kubernetes Secrets. **It might take a little time before it becomes available, wait and retry.**

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

6. Get the public IP address of the Argo CD Load Balancer by typing the following.

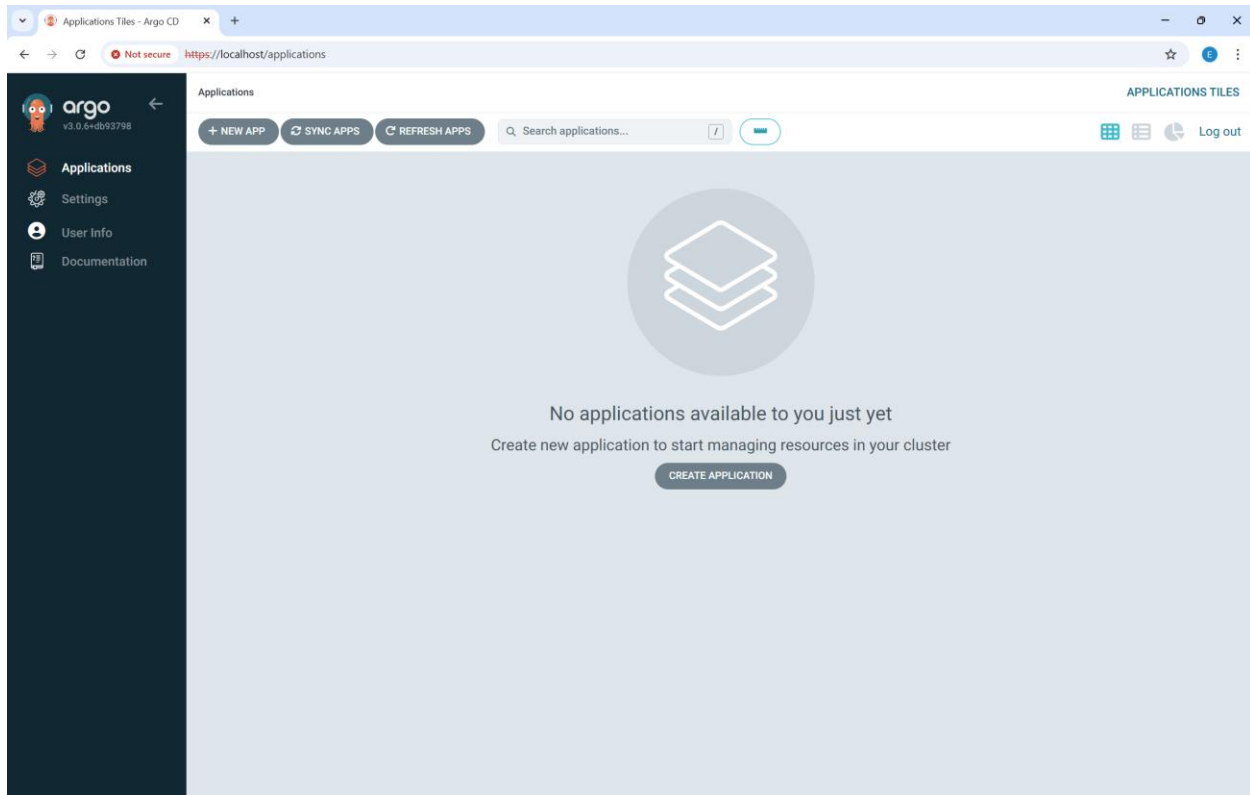
```
kubectl get svc argocd-server -n argocd
```



```
threecz@3CZSTUDIO: ~
threecz@3CZSTUDIO:~$
threecz@3CZSTUDIO:~$ kubectl get svc argocd-server -n argocd
NAME                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
argocd-server       LoadBalancer  10.97.115.157  localhost      80:31314/TCP,443:30190/TCP  4m36s
threecz@3CZSTUDIO:~$ |
```

7. Access **http://localhost** from the browser. Continue past the self-signed certificate warning, and proceed.

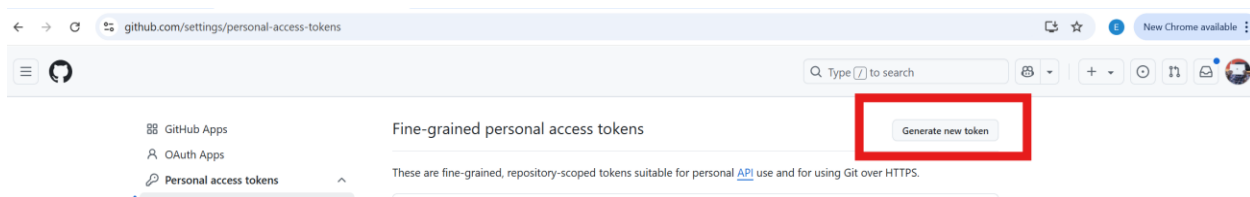
8. Enter **admin** as the username, and the password retrieved from step 5.



### Task 3: Connecting GitHub to Argo CD

In this task, you will connect Argo CD to your private GitHub Repository to allow Argo CD to read manifest files.

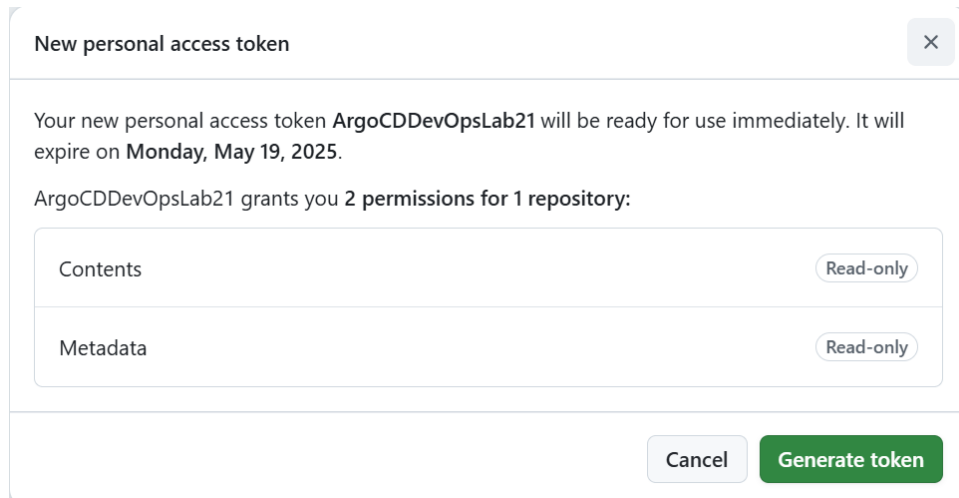
1. Create a GitHub Personal Access token. Go to <https://github.com/settings/personal-access-tokens>.
2. Click **Generate new token**.



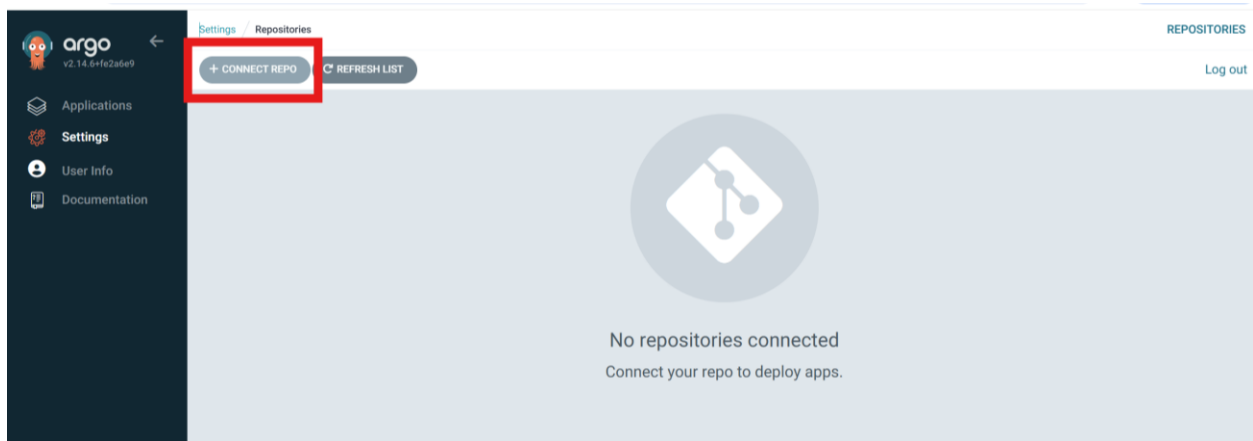
3. Scope the token as follows:
  - Token Name: **ArgoCDDevOpsLab21**
  - Expiration: **30 Days**
  - Repository Access: **Only select repositories > DevOpsLab21**
  - Permissions: **Repository Permissions: > Contents > Read Only**



- Click **Generate token** and then confirm.



- Copy the personal access token and keep it safely.
- Open up the Argo CD Dashboard and click **Settings**.
- Under settings, click **Repositories**.
- Click **+ CONNECT REPO**.



## DevOps Automation Lab Guide

9. Set the following details:

- Connection method: **HTTPS**
- Type: **git**
- Project: **default**
- Repository URL: **https://github.com/<your-github-username>/DevOpsLab21.git**
- Username: **<your-github-username>**
- Password: **<your-ArgoCDDevOpsLab21 -token>**

10. Click **Connect**.

The screenshot shows the Argo CD interface with the 'Settings' sidebar on the left. The main panel displays the 'Connect Repo Using HTTPS' dialog box. At the top, there are buttons for 'CONNECT', 'SAVE AS CREDENTIALS TEMPLATE', and 'CANCEL'. Below these, a dropdown menu shows 'VIA HTTPS'. The form fields are as follows:

- Type:** git
- Name (optional):**
- Project:** default
- Repository URL:** https://github.com/echebukati/DevOpsLab21.git
- Username (optional):** echebukati
- Password (optional):** (masked with dots)
- TLS client certificate (optional):**

11. You will be informed if the connection was successful as follows.

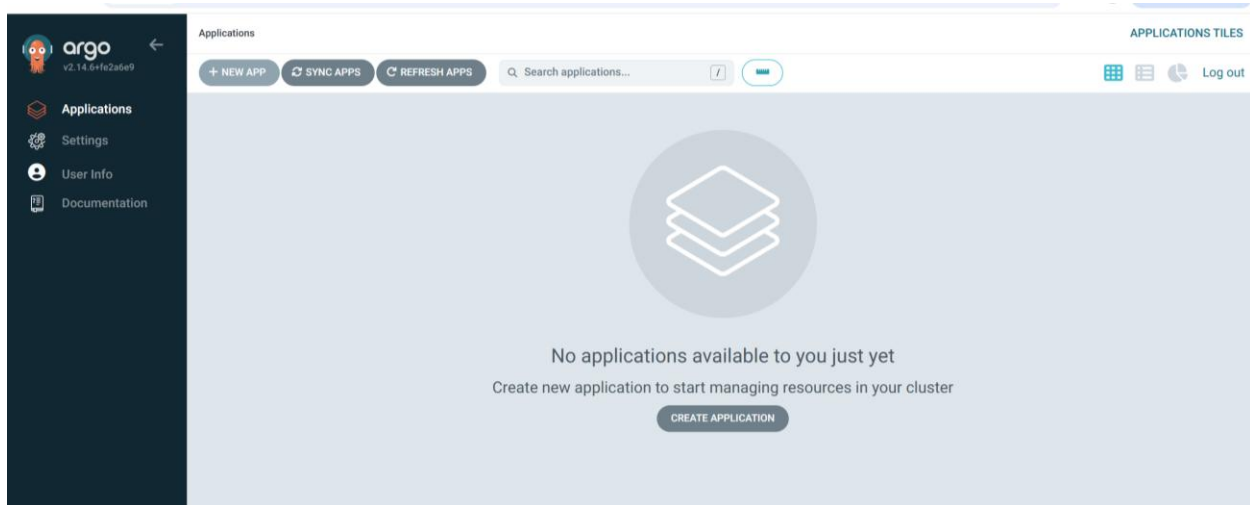
The screenshot shows the Argo CD 'Repositories' page. The sidebar on the left is the same as in the previous image. The main panel has a 'REPOSITORIES' header and a 'Log out' link. Below the header, there are buttons for '+ CONNECT REPO' and 'REFRESH LIST'. A table lists the connected repositories:

| TYPE | NAME | PROJECT | REPOSITORY                                    | CONNECTION STATUS |
|------|------|---------|---|-------------------|
| git  |      | default | https://github.com/echebukati/DevOpsLab21.git | Successful        |

## Task 4: Setting up an application on Argo CD

In this task, you will set up an Argo CD application using Kubernetes Manifest files in GitHub.

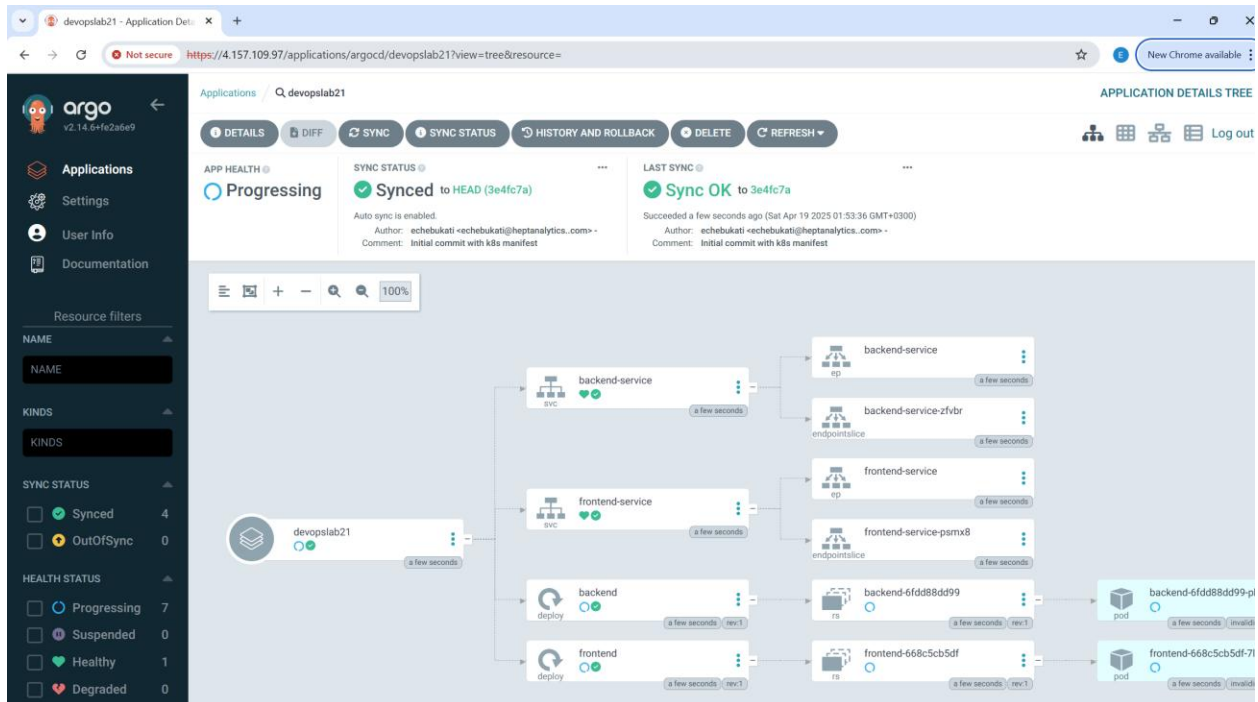
1. Return to **Applications** then click on **+ NEW APP**.



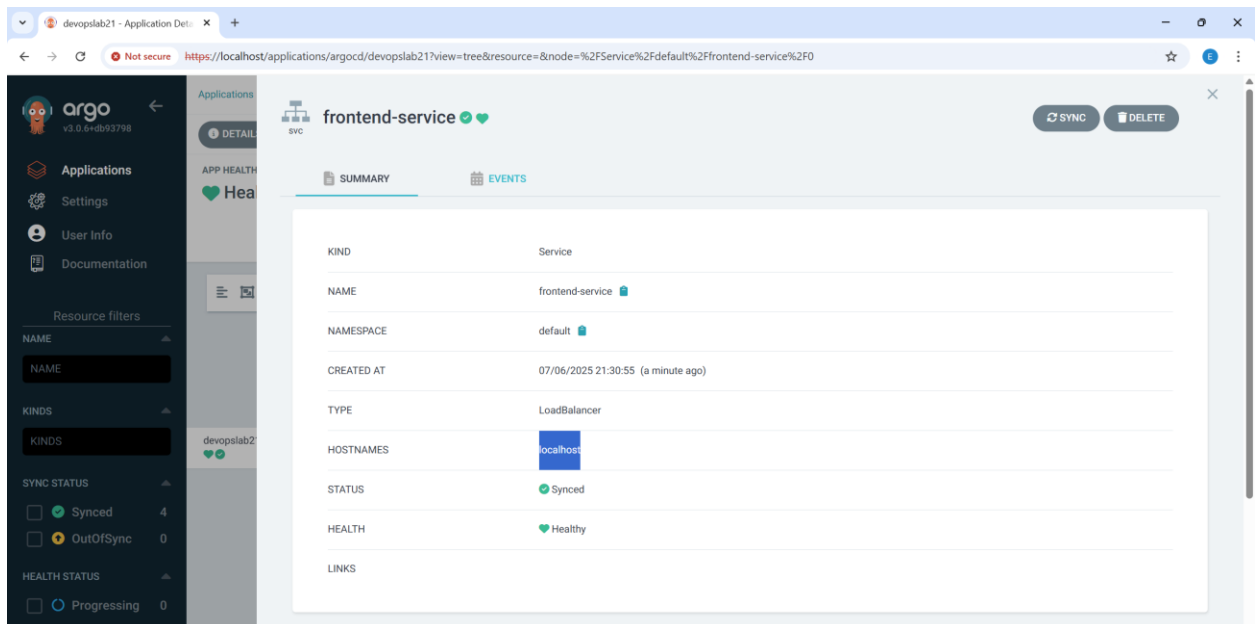
2. Fill in the following information:
  - Application Name: **devopslab21**
  - Project Name: **default**
  - Sync Policy: **Automatic**
    - Prune Resources: checked
    - Self Heal: checked
  - Repository URL: **https://github.com/<your-username>/DevOpsLab21.git**
  - Revision: **HEAD**
  - Path: **apps/devopslab21**
  - Cluster URL: **https://kubernetes.default.svc**
  - Namespace: **default**
3. Click **Create**.

## DevOps Automation Lab Guide

4. Immediately, Argo CD begins to provision the application in the cluster.



5. When the deployment is done, click on the frontend-service and copy the public IP address under hostnames.



- Paste the address, <http://localhost:8080>, into your browser and test that the application is working.



## What is your name?

Name:

## Task 5: Triggering self-healing

In this task, you will “accidentally” delete your frontend deployment and monitor how Argo CD restores it as per the declared state in the repository.

- On your VS Code terminal, run the following command to list your deployments.

```
kubectl get deployment
```

- Then, run the following command to delete the frontend deployment by accident.

```
kubectl delete deployment frontend
```

- Re-run the command to view all deployments.

```
kubectl get deployment
```

- You will notice that a new deployment has been created for you. This is Argo CD working in the background to maintain the state as you defined it in repository.

Expected outcome:

```
threecz@3CZSTUDIO: ~$ kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
backend       1/1     1             1           43m
frontend      1/1     1             1           2m36s
threecz@3CZSTUDIO:~$ kubectl delete deployment frontend
deployment.apps "frontend" deleted
threecz@3CZSTUDIO:~$ kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
backend       1/1     1             1           43m
frontend      1/1     1             1           9s
```

## Lab review

1. What is the point of having separate application and infrastructure repositories?
  - A. To improve the performance of deployments
  - B. To allow independent management of infrastructure and application code
  - C. To reduce storage space in the repository
  - D. To prevent having a public infrastructure repository
2. What does self-heal in Argo CD's sync policy do?
  - A. Rebuilds the Docker image if the container crashes
  - B. Automatically updates the Git repository
  - C. Detects and reverts changes made directly
  - D. Notifies users of new pull requests
3. What does **kustomization.yaml** assist with?
  - A. It defines how to build and publish Docker images to a container registry
  - B. It allows for environment-specific customization of Kubernetes manifests (such as image tags) without modifying the base resources
  - C. It installs applications into multiple Kubernetes clusters using Helm charts
  - D. It manages user access and role-based permissions across namespaces

### STOP

You have successfully completed this lab.

**Do not** delete any resources from this lab.