

GitOps Workflow using ArgoCD on Kubernetes

Objective:

Implemented a GitOps workflow by syncing Kubernetes deployment states directly from a Git repository using ArgoCD. The goal is to automate application deployment and updates using Git as the single source of truth.

Tools Used

- Kubernetes: Minikube (local cluster)
- ArgoCD: GitOps continuous deployment tool
- GitHub: Source control for deployment manifests
- Docker: Containerization of applications

Prerequisites: Install and confirm these tools are available on our machine

1. Docker installed and running (or another Minikube driver).
2. kubectl installed and on your PATH.
3. minikube installed.
4. Git + GitHub account.

Verification

`docker --version && kubectl version --client --short && minikube version && git --version`

```
MINGW64:/c:/Users/umama/OneDrive/Desktop
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ docker version
Client:
 Version:           26.1.4
 API version:       1.45
 Go version:        go1.21.11
 Git commit:        5650f9b
 Built:            Wed Jun  5 11:29:54 2024
 OS/Arch:          windows/amd64
 Context:          desktop-linux

Server: Docker Desktop 4.31.1 (153621)
 Engine:
  Version:          26.1.4
  API version:      1.45 (minimum version 1.24)
  Go version:       go1.21.11
  Git commit:       de5c9cf
  Built:            Wed Jun  5 11:29:22 2024
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.33
  GitCommit:        d2d58213f83a351ca8f528a95fbd145f5654e957
 runc:
  Version:          1.1.12
  GitCommit:        v1.1.12-0-g51d5e94
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0

umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ kubectl version --client
Client Version: v1.29.2
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3

umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ minikube version
minikube version: v1.33.1
commit: 5883c09216182566a63dff4c326a6fc9ed2982ff

umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ git version
git version 2.50.1.windows.1
```

Starting local Kubernetes cluster (Minikube)

Commands: `minikube start --driver=docker && kubectl get nodes`

We can see status of minikube by using `minikube status`

```
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ minikube start --driver=docker
* minikube v1.33.1 on Microsoft windows 11 Home Single Language 10.0.26100.4946 Build 26100.4946
* Using the docker driver based on user configuration
* Using Docker Desktop driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.44 ...
* Creating docker container (CPUs=2, Memory=2200MB) ...
* Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

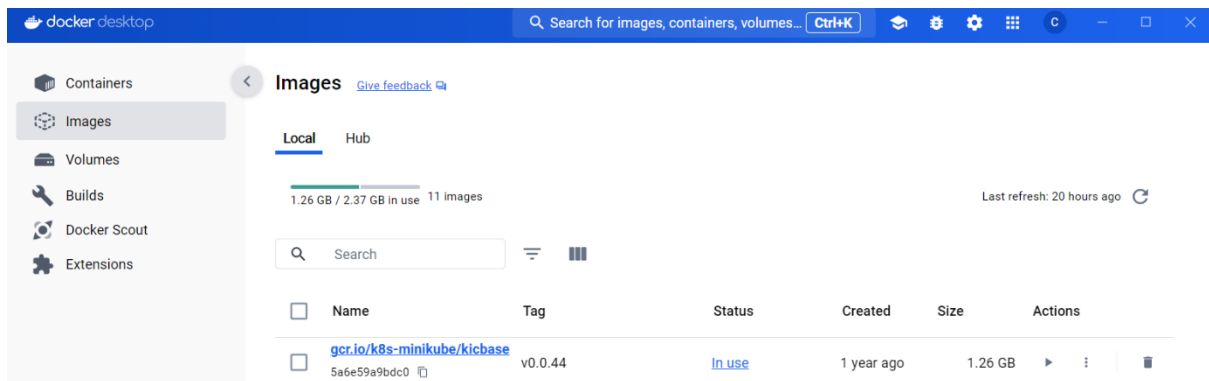
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ kubectl get nodes
NAME        STATUS    ROLES    AGE   VERSION
minikube    Ready     control-plane   28s   v1.30.0

umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

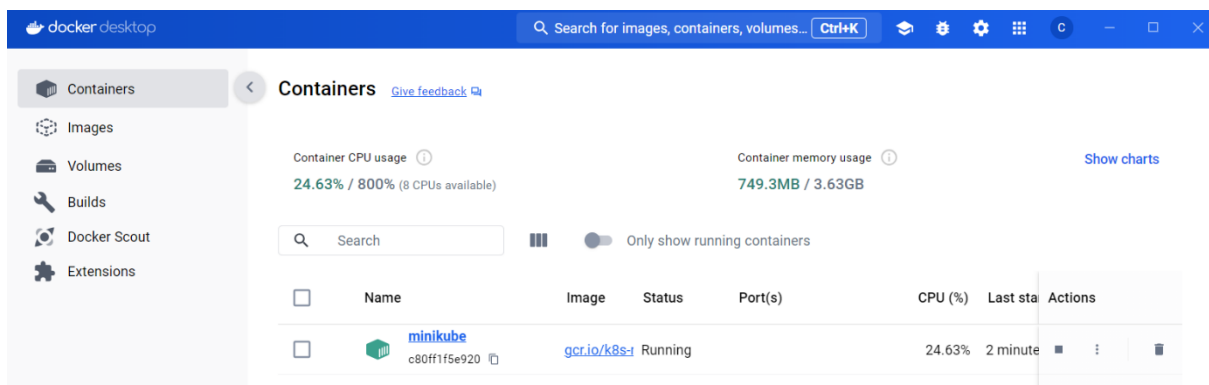
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$
```

We can see the minikube container is running in docker desktop

Image:



Container:



Install Argo CD into the cluster

Create separate namespace: `kubectl create namespace argocd`

We create ArgoCD application by this command:

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

```
MINGW64: c:\Users\umama\OneDrive\Desktop
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ kubectl create namespace argocd
namespace/argocd created

umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
customresourcedefinition.apiextensions.k8s.io/applications.argoproj.io unchanged
customresourcedefinition.apiextensions.k8s.io/applicationsets.argoproj.io unchanged
serviceaccount/argocd-application-controller created
serviceaccount/argocd-dex-server created
serviceaccount/argocd-notifications-controller created
serviceaccount/argocd-repo-server created
serviceaccount/argocd-server created
role.rbac.authorization.k8s.io/argocd-application-controller created
role.rbac.authorization.k8s.io/argocd-applicationset-controller created
role.rbac.authorization.k8s.io/argocd-dex-server created
role.rbac.authorization.k8s.io/argocd-notifications-controller created
role.rbac.authorization.k8s.io/argocd-repo-server created
role.rbac.authorization.k8s.io/argocd-server created
clusterrole.rbac.authorization.k8s.io/argocd-application-controller unchanged
clusterrole.rbac.authorization.k8s.io/argocd-applicationset-controller unchanged
clusterrole.rbac.authorization.k8s.io/argocd-server unchanged
rolebinding.rbac.authorization.k8s.io/argocd-application-controller created
rolebinding.rbac.authorization.k8s.io/argocd-applicationset-controller created
rolebinding.rbac.authorization.k8s.io/argocd-dex-server created
rolebinding.rbac.authorization.k8s.io/argocd-notifications-controller created
rolebinding.rbac.authorization.k8s.io/argocd-repo-server created
rolebinding.rbac.authorization.k8s.io/argocd-server created
clusterrolebinding.rbac.authorization.k8s.io/argocd-application-controller unchanged
clusterrolebinding.rbac.authorization.k8s.io/argocd-applicationset-controller unchanged
clusterrolebinding.rbac.authorization.k8s.io/argocd-server unchanged
configmap/argocd-cm created
configmap/argocd-cmd-params-cm created
configmap/argocd-gpg-keys-cm created
configmap/argocd-notifications-cm created
configmap/argocd-rbac-cm created
configmap/argocd-ssh-known-hosts-cm created
configmap/argocd-tls-certs-cm created
secret/argocd-notifications-secret created
secret/argocd-secret created
service/argocd-applicationset-controller created
```

After applying check whether pods and services are running or not

```
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ kubectl -n argocd get pods
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1     Running   0           5m44s
argocd-applicationset-controller-957474ff5-8k57b  1/1     Running   0           5m46s
argocd-dex-server-bc97497c6-7dfcx   1/1     Running   0           5m46s
argocd-notifications-controller-7d797bf45b-zm6ht  1/1     Running   0           5m45s
argocd-repo-server-74b74f9968-v4swd  1/1     Running   0           5m45s
argocd-repo-server-7c96dfd669-kfc94  1/1     Running   0           5m45s
argocd-server-5b8bb8cd8c-pbpfb      1/1     Running   0           5m44s

umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ kubectl -n argocd get svc
NAME                                TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)                                AGE
argocd-applicationset-controller    ClusterIP      10.110.217.255  <none>           7000/TCP,8080/TCP                     5m55s
argocd-dex-server                  ClusterIP      10.107.72.88    <none>           5556/TCP,5557/TCP,5558/TCP            5m55s
argocd-metrics                     ClusterIP      10.109.176.56   <none>           8082/TCP                               5m55s
argocd-notifications-controller-metrics  ClusterIP      10.98.242.81    <none>           9001/TCP                               5m54s
argocd-repo-server                 ClusterIP      10.97.90.116    <none>           6379/TCP                               5m54s
argocd-repo-server                 ClusterIP      10.100.50.226   <none>           8081/TCP,8084/TCP                     5m54s
argocd-server                      ClusterIP      10.104.81.176   <none>           80/TCP,443/TCP                       5m54s
argocd-server-metrics              ClusterIP      10.100.212.36   <none>           8083/TCP                               5m54s
```

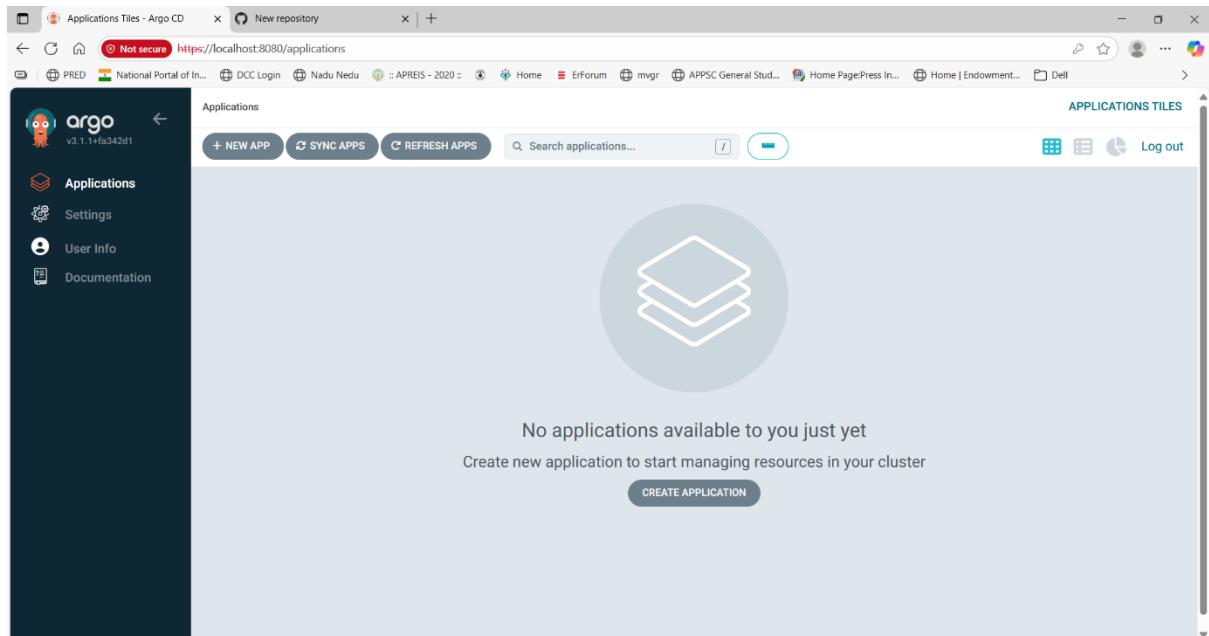
Open the Argo CD UI (port-forward)

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


```
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d && echo
F6ZBAUUMV1jvdv1g
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$
```

After login we should see the Argo CD dashboard.

In UI top-left, click **Applications** — it should be empty for now.



Create the GitHub repository and manifest files

Locally create the folder and files (deployment.yaml and service.yaml)

Deployment.yaml:

```
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-demo
  labels:
    app: nginx-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-demo
  template:
    metadata:
      labels:
        app: nginx-demo
    spec:
      containers:
        - name: nginx
          image: nginx:stable
          ports:
            - containerPort: 80
```

The deployment.yaml defines a **Kubernetes Deployment** for running the Nginx application.

- **apiVersion, kind, metadata:** Specifies this is a Deployment named nginx-demo.
- **replicas: 1:** Runs only one Pod (single instance of Nginx).
- **selector & template labels:** Ensures the Deployment manages Pods with the label app: nginx-demo.
- **containers:** Defines one container named nginx using the **nginx:stable** image.
- **ports:** Exposes port **80** inside the container for HTTP traffic.

Service.yaml

```
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ cat service.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-demo-svc
spec:
  selector:
    app: nginx-demo
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: ClusterIP
```

The service.yaml defines a **Kubernetes Service** to expose the Nginx Deployment inside the cluster.

- **apiVersion, kind, metadata:** Specifies this is a Service named nginx-demo-svc.
- **selector:** Targets Pods with the label app: nginx-demo (from the Deployment).
- **ports:** Maps **port 80** of the Service to **port 80** of the container for HTTP traffic.
- **type: ClusterIP:** Makes the Service accessible **only within the cluster** (not externally).

```
MINGW64/c:/Users/umama/OneDrive/Desktop/argocd-k8s
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ mkdir argocd-k8s
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ cd argocd-k8s/
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s
$ vi deployment.yaml
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s
$ vi service.yaml
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s
$ git init
Initialized empty Git repository in C:/Users/umama/OneDrive/Desktop/argocd-k8s/.git/
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ git add .
warning: in the working copy of 'deployment.yaml', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'service.yaml', LF will be replaced by CRLF the next time Git touches it
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ git commit -m "1st commit"
[main (root-commit) e74f730] 1st commit
2 files changed, 35 insertions(+)
create mode 100644 deployment.yaml
create mode 100644 service.yaml
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ git remote add origin https://github.com/vommidapuchinni/argocd-k8s.git
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 554 bytes | 110.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/vommidapuchinni/argocd-k8s.git
 * [new branch]      main -> main
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$
```

Open github in browser

Right click on + new repository gives name of the repo and give name and keep it public and with initiating readme file click created

Create a new repository
Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository.](#)
Required fields are marked with an asterisk ().*

1 General

Owner * vommidapuchinni

Repository name * argocd-k8s
argocd-k8s is available.

Great repository names are short and memorable. How about **super-duper-winner**?

Description
0 / 350 characters

2 Configuration

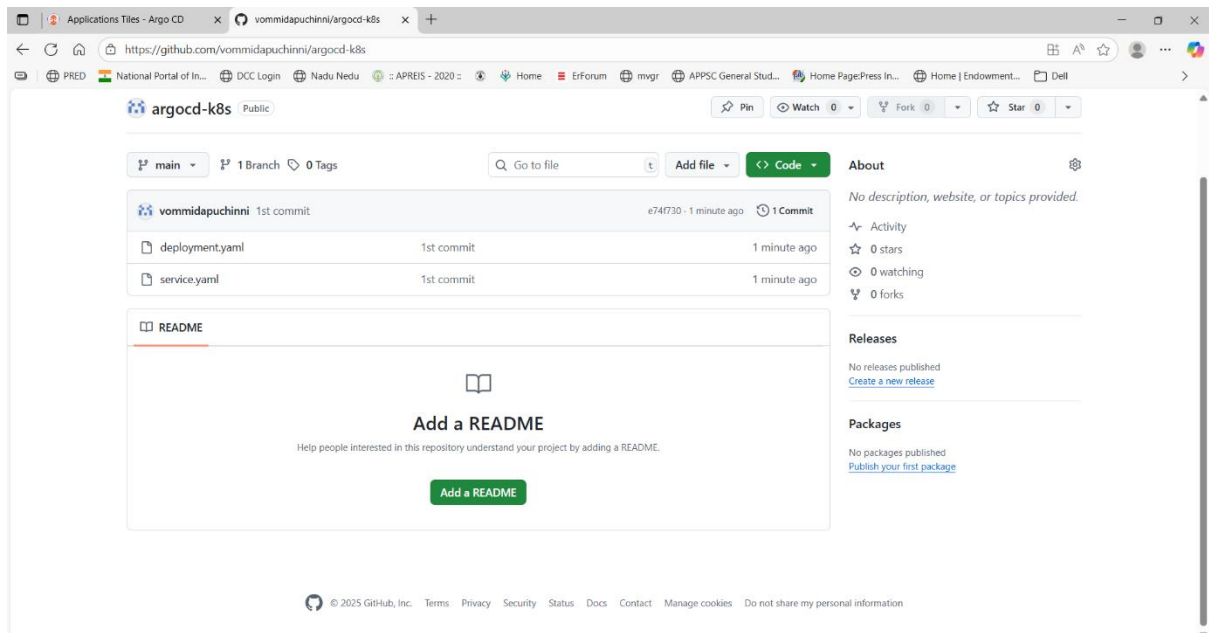
Choose visibility * Public
Choose who can see and commit to this repository

Add README Off
READMEs can be used as longer descriptions. [About READMEs](#)

Add .gitignore No .gitignore
.gitignore tells git which files not to track. [About ignoring files](#)

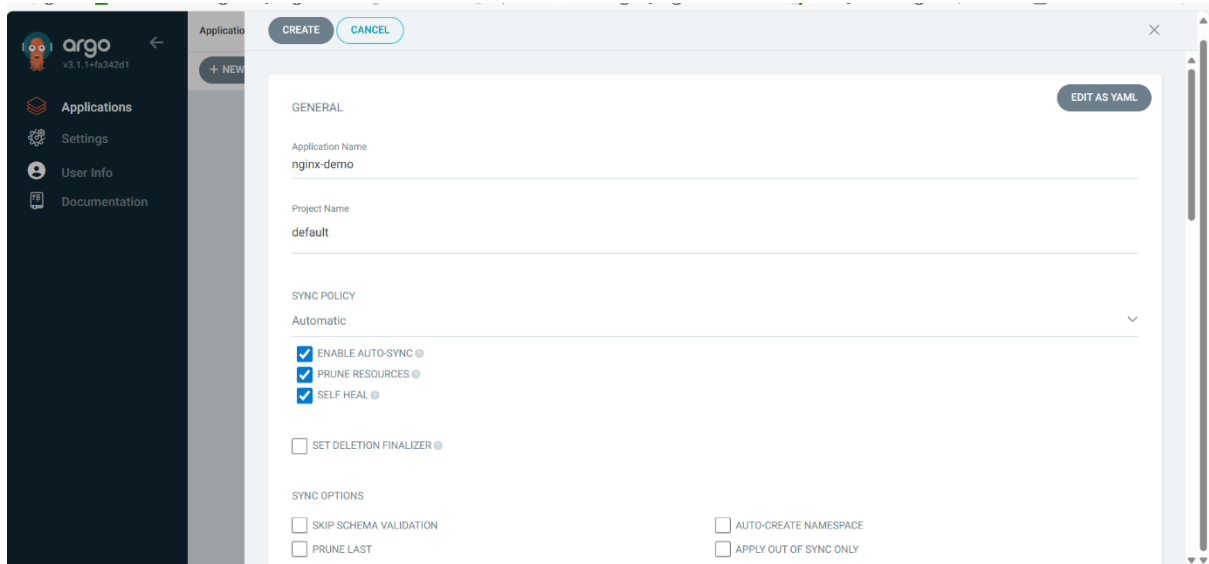
Push files to GitHub

```
MINGW64~/c:/Users/umama/OneDrive/Desktop/argocd-k8s
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ mkdir argocd-k8s
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop
$ cd argocd-k8s/
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s
$ vi deployment.yaml
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s
$ vi service.yaml
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s
$ git init
Initialized empty Git repository in C:/Users/umama/OneDrive/Desktop/argocd-k8s/.git/
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ git add .
warning: in the working copy of 'deployment.yaml', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'service.yaml', LF will be replaced by CRLF the next time Git touches it
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ git commit -m "1st commit"
[main (root-commit) 674f730] 1st commit
2 files changed, 35 insertions(+)
create mode 100644 deployment.yaml
create mode 100644 service.yaml
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ git remote add origin https://github.com/vommidapuchinni/argocd-k8s.git
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 554 bytes | 110.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/vommidapuchinni/argocd-k8s.git
 * [new branch]      main -> main
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$
```

Create the Argo CD Application

- In Argo CD UI left sidebar click Applications.
- Click NEW APP (top-right).
- Fill the form (exact values):
- Application Name: nginx-demo
- Project: default
- Expand Sync Policy and choose Automatic (tick Prune and Self-Heal if shown).



- Repository URL: <https://github.com/vommidapuchinni/my-argocd-demo.git>
- Revision: main
- Path: .

- Destination Cluster: <https://kubernetes.default.svc>
- Destination Namespace: default
- Click Create.

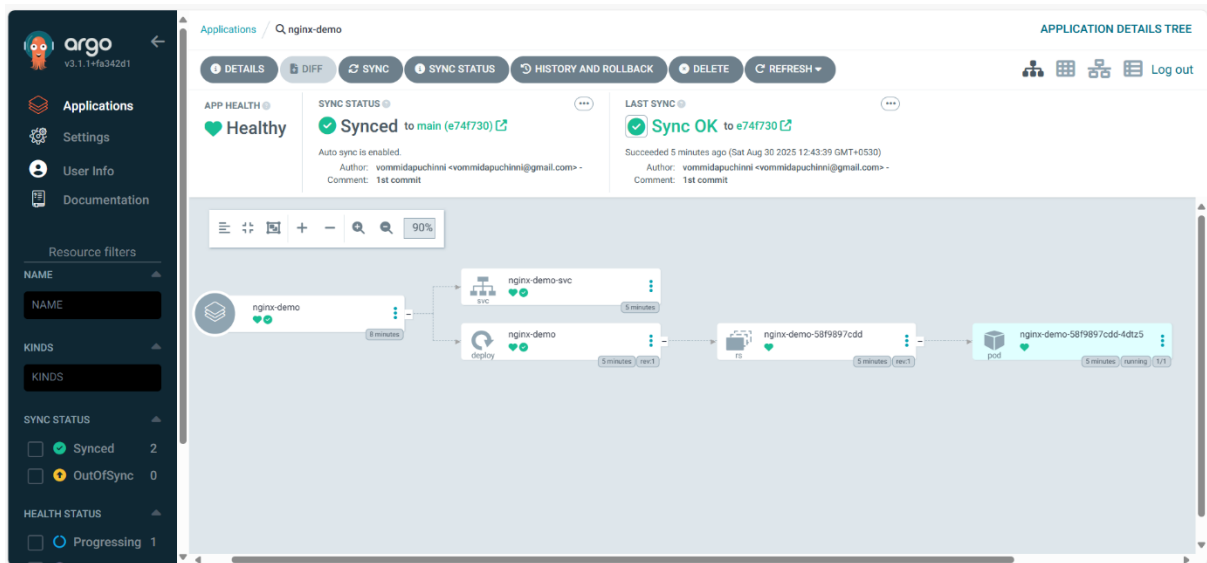
The screenshot shows the 'CREATE' form in the Argo CD web interface. The left sidebar contains the 'argo' logo and navigation links: Applications, Settings, User Info, and Documentation. The main area is titled 'Application' and has a 'CREATE' button and a 'CANCEL' button. The form is divided into two sections: 'SOURCE' and 'DESTINATION'. In the 'SOURCE' section, the 'Repository URL' is 'https://github.com/vommidapuchinni/argocd-k8s.git', the 'Revision' is 'main', and the 'Path' is '-'. In the 'DESTINATION' section, the 'Cluster URL' is 'https://kubernetes.default.svc', and the 'Namespace' is 'default'.

- In the Argo CD **Applications** list the nginx-demo app should appear.
- App status should move from OutOfSync → Synced (if automatic) and **Healthy**.
- In UI click the app → resource tree should show the Deployment and Service

The screenshot shows the 'Applications' list in the Argo CD web interface. The left sidebar contains the 'argo' logo and navigation links: Applications, Settings, User Info, and Documentation. The main area is titled 'Applications' and has a search bar and buttons for '+ NEW APP', 'SYNC APPS', and 'REFRESH APPS'. The 'nginx-demo' application is listed with the following details:

- Project: default
- Labels:
- Status: ♥ Healthy ✔ Synced
- Repository: https://github.com/vommidapuchinni/ar...
- Target R...: main
- Path: -
- Destinati...: in-cluster
- Namesp...: default
- Created ...: 08/30/2025 12:40:36 (8 minutes ago)
- Last Sync: 08/30/2025 12:43:39 (5 minutes ago)

At the bottom of the application details, there are buttons for 'SYNC', 'REFRESH', and 'DELETE'. The left sidebar also shows 'Application filters' and 'SYNC STATUS' (Unknown: 0, Synced: 1, OutOfSync: 0) and 'HEALTH STATUS' (Progressing: 0, Suspended: 0, Healthy: 1, Degraded: 0).



Command-line verification:

```
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ kubectl get deploy nginx-demo -o wide
NAME          READY  UP-TO-DATE  AVAILABLE  AGE    CONTAINERS  IMAGES          SELECTOR
nginx-demo    1/1    1           1           6m9s   nginx       nginx:stable    app=nginx-demo

umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ kubectl get svc nginx-demo-svc
NAME          TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)  AGE
nginx-demo-svc ClusterIP    10.107.132.117 <none>       80/TCP    6m31s

umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ kubectl get pod -l app=nginx-demo
NAME          READY  STATUS   RESTARTS  AGE
nginx-demo-58f9897cdd-4dtz5 1/1    Running   0          6m49s

umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$
```

Test the GitOps flow (make a change in yaml files and push to github and watch Argo CD apply it)

Update the image version to show GitOps flow

- Changed nginx:stable → nginx:1.23

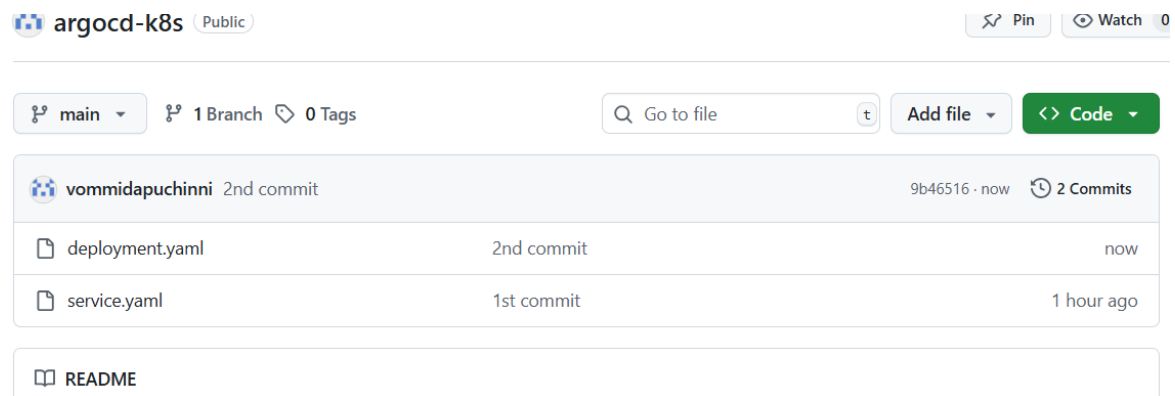
```
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-demo
  labels:
    app: nginx-demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-demo
  template:
    metadata:
      labels:
        app: nginx-demo
    spec:
      containers:
        - name: nginx
          image: nginx:1.23
          ports:
            - containerPort: 80
```

Commit & push

git add deployment.yaml

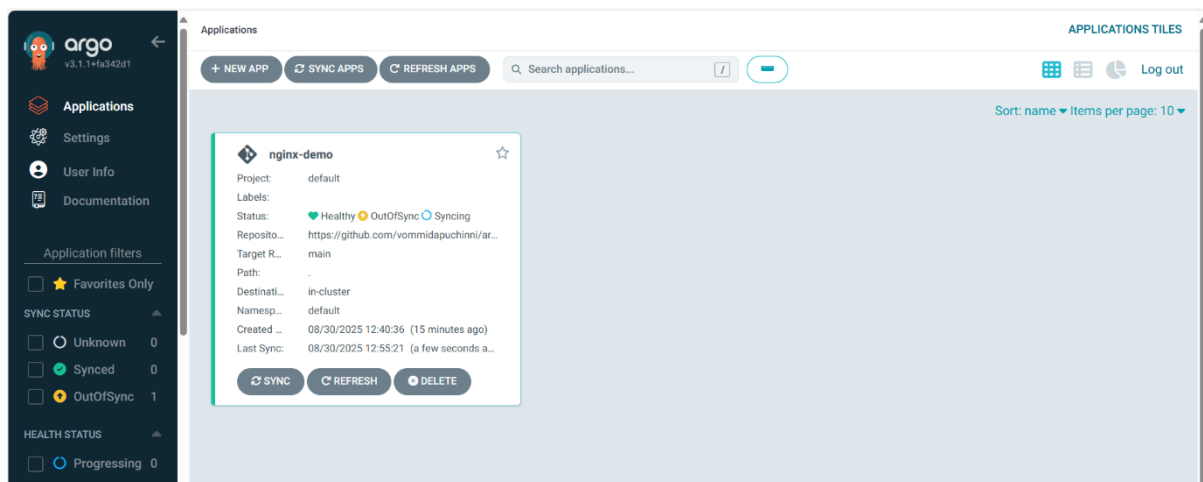
git commit -m "2nd commit"

git push origin main



ArgoCD automatically syncs (since Auto-Sync is enabled)

- UI: Shows OutOfSync → syncing → Synced & Healthy

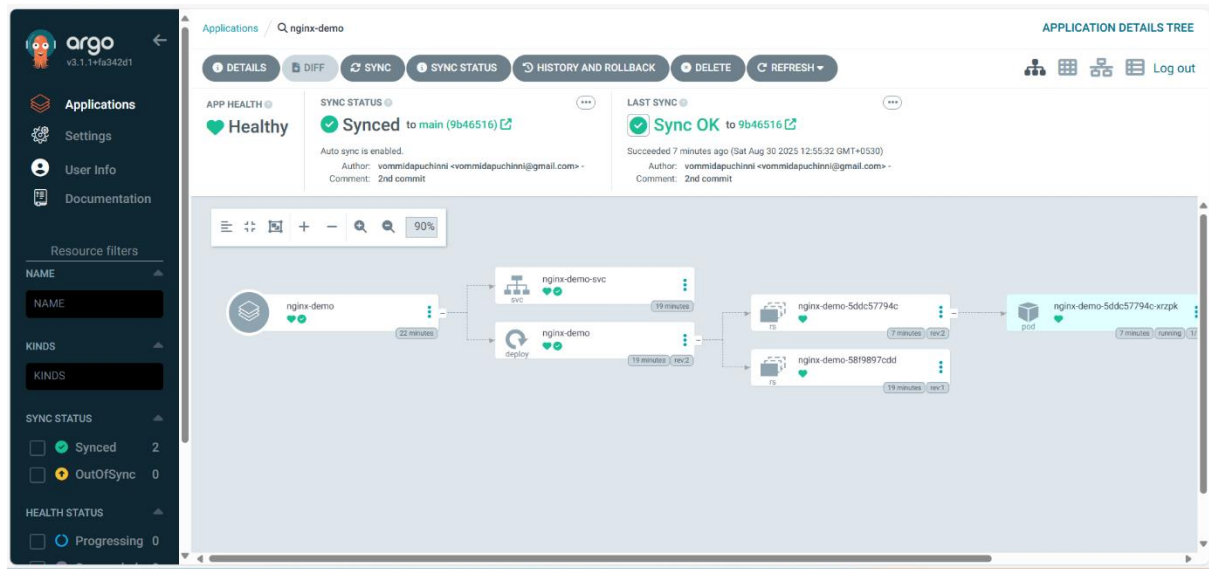


Verify deployment

kubectl get pods -n default

kubectl describe pod <pod-name> -n default





Now I changed the replicas from 1 to 2

```
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-demo
  labels:
    app: nginx-demo
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx-demo
  template:
    metadata:
      labels:
        app: nginx-demo
    spec:
      containers:
      - name: nginx
        image: nginx:1.23
        ports:
        - containerPort: 80
```

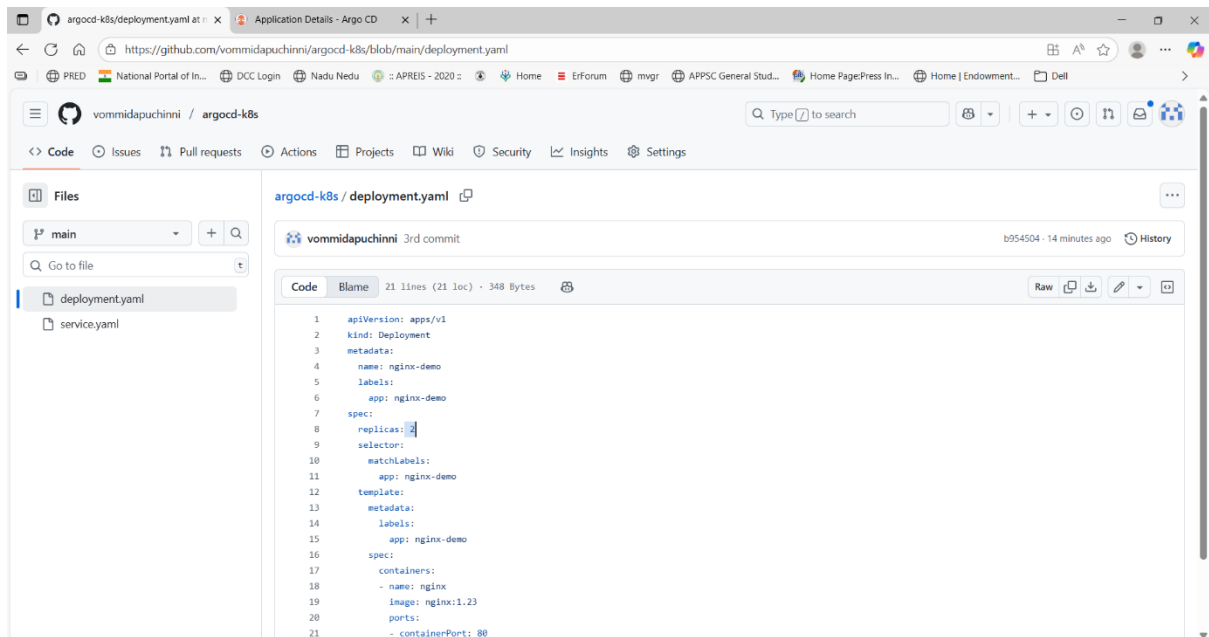
Pushed the edit file to git hub repo

```
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ vi deployment.yaml

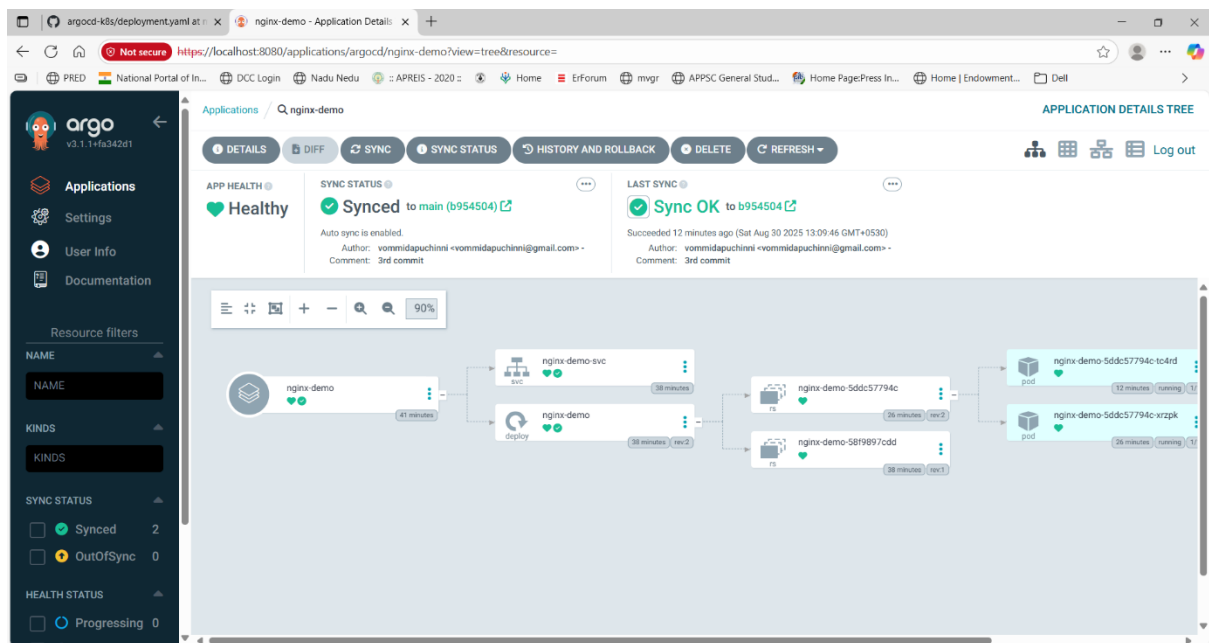
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ git add .
warning: in the working copy of 'deployment.yaml', LF will be replaced by CRLF the next time Git touches it

umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ git commit -m "3rd commit"
[main b954504] 3rd commit
1 file changed, 1 insertion(+), 1 deletion(-)

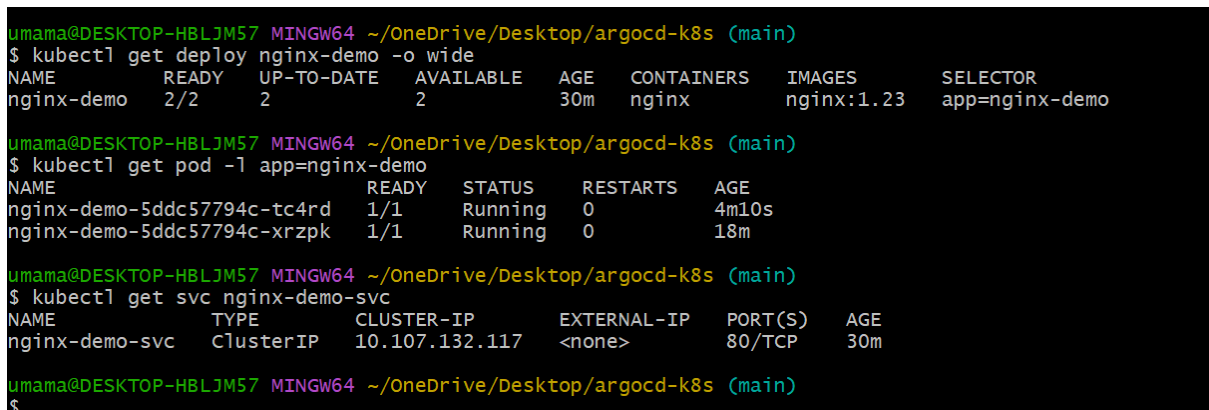
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 309 bytes | 154.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/vommidapuchinni/argocd-k8s.git
9b46516..b954504 main -> main
```



We see auto sync is happened



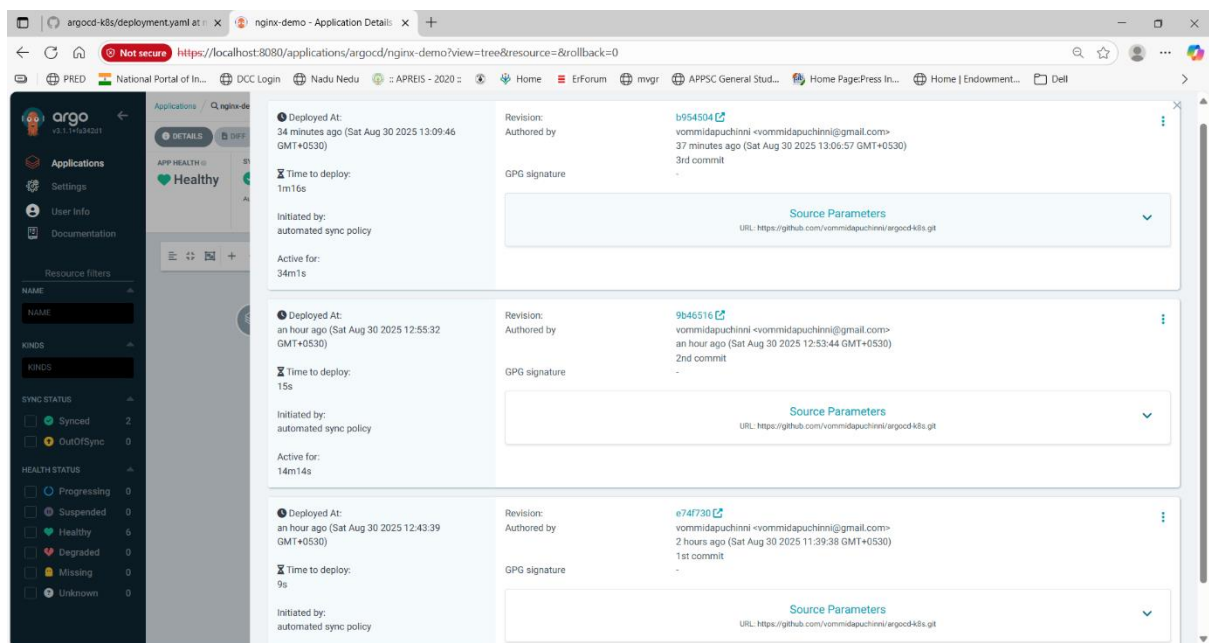
Local verification of replicas



```
umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$ kubectl get applications -n argocd nginx-demo -o yaml | grep path:
  path: .
    path: .
    path: .
    path: .
      path: .
      path: .

umama@DESKTOP-HBLJM57 MINGW64 ~/OneDrive/Desktop/argocd-k8s (main)
$
```

We see history roll back



Applications / nginx-de

DETAILS

DIFF

APP HEALTH

Healthy

OPERATION	Sync
PHASE	Succeeded
MESSAGE	successfully synced (all tasks run)
STARTED AT	36 minutes ago (Sat Aug 30 2025 13:08:30 GMT+05)
DURATION	1m16s
FINISHED AT	35 minutes ago (Sat Aug 30 2025 13:09:46 GMT+05)

The screenshot shows the Argo CD web interface. On the left is a sidebar with navigation links: Applications, Settings, User Info, and Documentation. Below these are resource filters for NAME, KINDS, SYNC STATUS, and HEALTH STATUS. The main panel displays the 'DETAILS' view for an application named 'nginx-demo'. It shows a 'Sync' operation that 'Succeeded' with the message 'successfully synced (all tasks run)'. The operation started 35 minutes ago and finished 35 minutes ago. Below this, a 'RESULT' table lists the resources that were synced:

SYNC WAVE	KIND	NAMESPACE	NAME	STATUS	HEALTH	HOOK	MESSAGE
0	v1/Service	default	nginx-demo-svc	Synced	Healthy		service/nginx-demo-svc unchanged
0	apps/v1/Deployment	default	nginx-demo	Synced	Healthy		deployment apps/nginx-demo configured

Our files repo

The screenshot shows the 'MANIFEST' tab of the Argo CD interface. It displays the YAML configuration for an application. The configuration is as follows:

```

1 project: default
2 source:
3   repoURL: https://github.com/vommidapuchinni/argocd-k8s.git
4   path: .
5   targetRevision: main
6 destination:
7   server: https://kubernetes.default.svc
8   namespace: default
9 syncPolicy:
10  automated:
11    prune: true
12    selfHeal: true
13    enabled: true
14

```

Optional: create the Argo CD Application declaratively (store Application YAML in Git)

If you want Argo CD itself to be configured from Git (pure GitOps), create an `argocd-app.yaml`:

`apiVersion: argoproj.io/v1alpha1`

`kind: Application`

`metadata:`

`name: nginx-demo`

`namespace: argocd`

spec:

project: default

source:

repoURL: 'https://github.com/<you>/my-argocd-demo.git'

targetRevision: main

path: k8s

destination:

server: https://kubernetes.default.svc

namespace: default

syncPolicy:

automated:

prune: true

selfHeal: true

apply by

kubectl apply -f argocd-app.yaml -n argocd

This is advanced but recommended for infra-as-code

Troubles Faced

- ArgoCD UI Access: TLS/self-signed warnings when using port-forward.
- Minikube Issues: Startup failures due to insufficient memory or Docker runtime problems.
- Pod Restarts: Some ArgoCD pods (e.g., repo-server) restarted; needed logs to debug.
- Git Sync Confusion: Changes didn't reflect until YAMLs were pushed to GitHub and auto-sync enabled.
- Service Access: NodePort optional; ClusterIP and port-forward worked after fixing Minikube and Docker.
- Resolution: Followed GitOps workflow properly—YAMLs in Git → ArgoCD Auto-Sync → changes applied automatically.

Key Concepts

- GitOps: Manage Kubernetes deployments declaratively via Git; Git is the single source of truth.
- ArgoCD: Continuous delivery tool for Kubernetes that automatically syncs cluster state with Git.
- Deployment: Defines the desired state of an application (replicas, container image, labels).
- Service: Exposes a deployment inside the cluster (ClusterIP, NodePort, LoadBalancer).
- Auto-Sync: ArgoCD feature to automatically apply changes from Git to the cluster.
- Port-Forwarding: Access cluster services locally for testing UI without exposing NodePorts.
- ClusterIP vs NodePort: ClusterIP is internal access; NodePort exposes service externally.
- Pods & ReplicaSets: Pods are running instances of containers; ReplicaSets ensure desired number of replicas.

Conclusion

In this project, we implemented a GitOps workflow using ArgoCD and Kubernetes. The deployment manifests for the Nginx application were stored in Git, and ArgoCD automatically synced the desired state to the cluster. This approach ensures version-controlled, automated, and reliable deployments, demonstrating the power of GitOps for managing Kubernetes applications efficiently.