Student Guide: GitOps with Argo CD and Helm

---

1. Purpose of This Lab

This lab provides **hands-on, end-to-end understanding of GitOps** using:

- Kubernetes (via Docker Desktop)
- Argo CD for Continuous Delivery
- Helm as a templating mechanism

By completing this lab, students will **experience GitOps behavior directly**, rather than just reading about it.

Flux CD is intentionally excluded to keep the learning focused and clear.

---

2. What You Will Learn

At the end of this lab, you will be able to:

- Understand the GitOps workflow and philosophy
- Deploy applications using Argo CD (not kubectl)
- Use Helm charts with Argo CD
- Observe auto-sync, prune, and self-healing behavior
- Understand why Git is the single source of truth
- Troubleshoot common GitOps mistakes

---

3. GitOps Concept (Very Important)

GitOps = Git is the Source of Truth

- Desired state is defined in Git
- Argo CD continuously watches Git
- Kubernetes is reconciled to match Git
- Manual changes are detected and reverted

If Git and the cluster disagree, **Git always wins**.

---

4. Lab Architecture

```
GitHub Repository
 └── Helm Chart (values.yaml + templates)
        ↓
     Argo CD
        ↓
     Kubernetes (Docker Desktop)
```

---

5. Prerequisites

Before starting:

- Docker Desktop installed and running
- Kubernetes enabled in Docker Desktop
- kubectl configured
- GitHub account
- Git and GitHub CLI authenticated

---

6. Repository Structure Used in This Lab

```
gitops-demo/
├── helm/
│   └── nginx/
│       ├── Chart.yaml
│       ├── values.yaml
│       └── templates/
│           └── deployment.yaml
├── argo-app.yaml
└── README.md
```

---

7. Step 1: Install Argo CD

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

Expose Argo CD UI:

```
kubectl port-forward svc/argocd-server -n argocd 8081:443
```

Access UI:

```
https://localhost:8081
```

Get admin password:

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

---

8. Step 2: Create GitHub Repository

- Create a GitHub repository named `gitops-demo`
- Push the local project files to GitHub

This repository becomes the **single source of truth**.

---

9. Step 3: Create Argo CD Application (GitOps Way)

Create `argo-app.yaml`:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: nginx-demo
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/<YOUR_GITHUB_USERNAME>/gitops-demo
    targetRevision: HEAD
    path: helm/nginx
  destination:
    server: https://kubernetes.default.svc
    namespace: default
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

Apply it:

```
kubectl apply -f argo-app.yaml
```

The Argo CD UI **does not show Edit buttons** because the app is Git-managed.

---

10. Step 4: Helm Chart Creation

Create chart:

```
helm create helm/nginx
```

Important Cleanup (Corrected Step)

Helm creates many optional templates. For this lab, we **remove unused ones**:

```
rm -rf
helm/nginx/templates/{tests,service.yaml,serviceaccount.yaml,ingress.yaml,htt
proute.yaml,hpa.yaml,NOTES.txt}
```

Only keep:

```
helm/nginx/templates/
 └── deployment.yaml
```

---

11. Step 5: Helm Deployment Template

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Release.Name }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: {{ .Release.Name }}
  template:
    metadata:
      labels:
        app: {{ .Release.Name }}
    spec:
      containers:
      - name: nginx
        image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
        ports:
        - containerPort: 80
```

---

12. Step 6: Helm Values File

```
replicaCount: 3

image:
  repository: nginx
  tag: "1.25"
```

---

13. Step 7: Validate Helm Locally (Important)

```
helm template nginx helm/nginx
```

This must render YAML without errors **before** using Argo CD.

---

14. Step 8: Commit and Push to Git

```
git add .
git commit -m "Helm-based GitOps deployment"
git push
```

Step 15: Clean Reload – Commands

1️⃣ Delete the Argo CD Application (with resources)

Option A: Using kubectl (GitOps-safe)

```
kubectl delete application nginx-demo -n argocd
```

Verify all resources are removed:

```
kubectl get deploy -n default
```

Expected:

```
No resources found
```

If any legacy deployment still exists (rare case):

```
kubectl delete deploy nginx -n default
```

2️⃣ Recreate the Application from `argo-app.yaml`

Ensure `argo-app.yaml` points to Helm:

```
path: helm/nginx
```

Recreate the application:

```
kubectl apply -f argo-app.yaml
```

3️⃣ Verify Reconciliation

```
kubectl get applications -n argocd
kubectl get deploy -n default
kubectl get pods -n default
```

Expected:

```
nginx-demo   3/3   Running
```

---

16. Step 10: Observe GitOps in Action

Auto Sync

- App becomes `Synced` automatically

Self-Healing (Pod Level)

```
kubectl delete pod <pod-name>
```

- Pod is recreated automatically
- Argo CD UI shows live transition

Self-Healing (Config Level)

```
kubectl scale deploy nginx-demo --replicas=1
```

- Argo CD detects drift
- Automatically reverts to Git value

---

17. Key Troubleshooting Lessons (From This Lab)

1. Helm vs helm CLI

- `helm list` may show nothing
- Argo CD runs Helm internally

2. Deployment Name Change

- Helm uses application name (`nginx-demo`)

- Do not look for old `nginx` deployment

3. UI Edit Button Missing

- Expected for Git-managed apps
- Changes must be done in Git

4. Prune Required During Migration

- Required once when changing source type

---

18. Final Takeaways

- Git controls deployments
- Argo CD continuously reconciles state
- Helm provides flexibility via values.yaml
- Manual cluster changes never survive

GitOps is not a tool — it is an operating model.

---

19. Completion Checklist

- Argo CD installed
- GitHub repository created
- Helm-based deployment
- Auto-sync verified
- Self-healing observed
- Git as source of truth confirmed
