**🧠 What is GitOps?**

**GitOps = Git + DevOps**

**GitOps is a modern way to manage Kubernetes using Git as the single source of truth.**

It means:

* You store all your **Kubernetes configurations (YAML files)** in a Git repository
* Any change to your app, infra, or configuration is done by committing to Git
* A tool like **Argo CD or Flux** watches that Git repo and **automatically applies** changes to your cluster

**🧱 Core GitOps Principles (Easy Explanation)**

| **Principle** | **Explanation** |
| --- | --- |
| ✅ **Git as the single source of truth** | All your Kubernetes YAMLs (deployments, services, config) live in Git. No changes made manually in the cluster. |
| 📦 **Declarative infrastructure** | Everything you want in your cluster is described in files (e.g., deployment.yaml). Git becomes your blueprint. |
| 🔁 **Automated delivery** | Tools like Argo CD **automatically sync** what’s in Git to your cluster. |
| 🔍 **Version control and audit** | Git tracks every change: who changed what, when, and why. Easy rollback with Git history. |
| 🔐 **Security and compliance** | No one needs direct access to the cluster — all changes go through Git (with review + approval). |
| 🔧 **Self-healing** | If someone makes a manual change in the cluster, GitOps tools **detect drift** and fix it automatically. |

**Why we need GitOps**

**🎯 Why Do We Need GitOps? (With Scenarios)**

| **❌ Problem (Without GitOps)** | **✅ GitOps Solves It By** | **🔍 Real-World Scenario** |
| --- | --- | --- |
| **Manual kubectl apply commands** | **Automated sync from Git** | **A developer forgets to apply the latest YAML to staging; production behaves differently. With GitOps, once it's merged to main, Argo CD applies it — no human error.** |
| **Drift between Git and cluster** | **Reconciles cluster to match Git** | **An engineer manually updates a replica count from 2 to 4 in the cluster. Git still says 2. Argo CD detects the mismatch and rolls back the drift — keeping the cluster in sync with Git.** |
| **No change tracking** | **Every config change is versioned** | **You're debugging a failure — but nobody knows who changed the env variable in the deployment YAML last week. With GitOps, every change is recorded in Git (with commit, author, timestamp).** |
| **Risk of mistakes during live edits** | **Safe, peer-reviewed changes via Git** | **Someone applies a YAML with the wrong image tag in production. Oops! With GitOps, changes go through a Pull Request, team reviews it, and only then it's merged and deployed.** |
| **Difficult rollbacks** | **Rollback with git revert or git checkout** | **A new deployment causes issues. You don’t need to remember the last good YAML or redo changes — just git revert and Argo CD automatically restores the working state.** |
| **No team collaboration** | **Everyone works through Git (pull requests, reviews)** | **DevOps works in kubectl, Devs change YAML in local files, SRE tweaks directly in the cluster. Chaos. With GitOps, everyone works from the same Git repo with PRs and reviews — unified and auditable.** |

**👨‍🏫 How You Can Explain This to Your Team:**

**“Instead of logging into the cluster and applying YAMLs manually, we store everything in Git. Tools like Argo CD keep the cluster updated with whatever is in Git. That way, we get audit, rollback, and automation all in one.”**

**🖼️ Architecture Diagram (Text View)**

Developer

|

| 1. Push YAML to Git (Deployment/Service/etc.)

v

Git Repository (e.g., GitHub)

|

| 2. Argo CD watches this repo

v

Argo CD running in AKS

|

| 3. Reconciles desired state from Git

v

AKS Cluster (actual state updated)

**Step-by-Step GitOps Flow**

**🧑‍💻 Step 1: Developer Pushes YAML to Git**

Developer makes a change:

Pushes it to the main branch of:

**🛰️ Step 2: Argo CD Watches Git Repository**

* Argo CD is **installed in AKS** (via Helm or manifest)
* It has an **Application** object pointing to that Git repo:

**🔁 Step 3: Argo CD Syncs to AKS**

* Argo CD detects the Git commit
* Applies the change to AKS using the Kubernetes API
* Your app is updated (replicas, config, image tag, etc.)

✅ No manual kubectl apply  
✅ Full history in Git  
✅ Cluster reflects Git = **Single Source of Truth**

Perfect! Let's implement a **complete GitOps flow using:**

✅ Git: [https://github.com/Sumanth17-git/argocd-demo](https://github.com/Sumanth17-git/argocd-demo/blob/main/deployment.yaml)  
✅ Argo CD: Installed in AKS  
✅ AKS: Target cluster for deployment

**✅ Step 3: Create Argo CD Application**

We’ll deploy this using an Application YAML pointing to:

* Git repo: https://github.com/Sumanth17-git/argocd-demo.git
* Path: . (root of repo)
* Namespace: default
* Cluster: same AKS cluster

**.**

**🔒 Problem: Argo CD Can't Reach Private AKS Cluster**

**If your target AKS cluster is private (no public API server endpoint), Argo CD in a public VNet or different cloud will fail to connect.**

Solution Strategies for Private Cluster Connectivity

🔁 Option 1: Deploy Argo CD inside the private AKS cluster itself

Best for fully private environments.

* Argo CD is deployed in-cluster on the same private AKS.
* Use Argo CD as a self-managing controller.
* You can still register other private/public clusters that are reachable via VNet peering.

✅ Argo CD has direct access to the API server via https://kubernetes.default.svc.

🌐 Option 2: VNet Peering or VPN/ExpressRoute between Argo CD and Private AKS

If Argo CD is in another cluster or VM in Azure:

* Peer the VNets between:
  + The Argo CD host cluster
  + The private AKS cluster
* Ensure NSGs and Route Tables allow traffic on port 443 to the private API server
* Register the private AKS using its private IP or internal FQDN

bash

CopyEdit

kubectl config set-cluster <name> --server=https://<private-ip>:443

💡 The Argo CD pod must be able to resolve and reach that IP/FQDN.

Option 3: Use Argo CD Agent (Argo CD v2.7+) with Remote Clusters

Argo CD ApplicationSet + Argo CD Agent = Cluster-local GitOps

* Deploy the Argo CD Agent inside the private cluster.
* Agent pulls configuration from the main Argo CD server (reverse proxy model).
* No need for Argo CD to initiate connection to the private cluster.

✅ Works even if cluster is completely private with no inbound access.

**Create Namespace for Argo CD**

kubectl create namespace argocd

helm repo add argo https://argoproj.github.io/argo-helm

helm repo update

**🔐 Step 1: Generate bcrypt hash for admin@123**

sudo apt install apache2-utils # Debian/Ubuntu

**Generate the bcrypt hash:**

htpasswd -nbBC 10 "" admin@123 | tr -d ':\n' | sed 's/$2y/$2a/'

**🛠 Step 2: Install Argo CD with Helm and set the password**

helm install argocd argo/argo-cd \

--namespace argocd \

--create-namespace \

--set server.service.type=LoadBalancer \

--set configs.params.server.insecure=true \

--set-string "configs.secret.argocdServerAdminPassword=\$2a\$10\$9NPKk3cEQczH8uqoN6YcDuwYjEkFZ5I6oncoRIyGjRt4bT47O4hfK"

**🔐 Replace the hashed password with the output from your htpasswd command.**

**kubectl get svc argocd-server -n argocd**

**✅ Step 4: Login**

* **Username: admin**
* **Password: admin@123**

🚀 App 1: manifests (YAML or Kustomize)

1️⃣ Open Argo CD UI  
Login with admin + your password.

2️⃣ Click NEW APP

3️⃣ Fill in the form:

✅ Application Name: springboot-manifests

✅ Project:default

✅ Sync Policy:  
*Leave as Manual for now, or check Auto-Sync if you want automatic deployment.*

4️⃣ In Source:

* Repository URL:

https://github.com/Sumanth17-git/argocd-demo.git

* Revision:main
* Path:manifests

*(This tells Argo CD to look in the manifests folder)*

✅ If you are using plain YAML files:

* Directory recurse: leave default
* Kustomize: leave default

5️⃣ In Destination:

* Cluster URL:https://kubernetes.default.svc
* Namespace: default

6️⃣ Click Create

✅ Done!

* You should see springboot-manifests in the Argo CD UI.
* Click SYNC to deploy.

**argocd-app.yaml**

apiVersion: argoproj.io/v1alpha1

kind: Application

metadata:

name: argocd-demo

namespace: argocd

spec:

project: default

source:

repoURL: https://github.com/Sumanth17-git/argocd-demo.git

targetRevision: main

path: . # since deployment.yaml is in the root

destination:

server: https://kubernetes.default.svc

namespace: default

syncPolicy:

automated:

prune: true

selfHeal: true

**✅ Apply the Application to Argo CD**

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

**✅ Step 4: Verify in Argo CD UI**

* **Open your Argo CD UI (http://<external-ip>)**
* **You’ll see: argocd-demo app**
* **Status: “Synced” and “Healthy”**

**✅ Step 5: Confirm App is Running in AKS**

**kubectl get pods -n default**

**kubectl get deployments -n default**

**⏱️ How Quickly Does Argo CD Detect Git Changes?**

**By default, Argo CD checks for changes in the Git repo every 3 minutes (180 seconds).**

**A screenshot of a phone

AI-generated content may be incorrect.**

**🛠️ How to Make It Faster (Optional)**

You can reduce the Git polling interval by setting the --repo-server.repo.poll.interval and --controller.repoServerTimeoutSeconds flags during Argo CD install.

helm upgrade argocd argo/argo-cd \

--namespace argocd \

--set repoServer.extraArgs="{--repo.poll.interval=30s}" \

--set controller.repoServerTimeoutSeconds=60

**⚡ BEST OPTION: Use Git Webhooks (Instant Detection)**

Set up a Git webhook to notify Argo CD on every commit.

Steps for GitHub:

1. Go to your repo → Settings → Webhooks
2. Add a new webhook:
   * URL: http://<argocd-repo-server-service>:8081/api/webhook
   * Content type: application/json
   * Secret: Optional
3. Trigger on Push events

**Argo CD will instantly fetch and sync the changes.**

**Excellent! Let's walk through a real-time GitOps test using Argo CD and GitHub to validate:**

* **🔄 Auto-Sync: Push changes to Git, and Argo CD applies them to AKS.**
* **🔙 Rollback: Revert the Git commit and Argo CD rolls back the deployment automatically.**

**🔁 STEP 1: Test Auto-Sync**

1️⃣ Go to your GitHub repo → Edit manifests/deployment.yaml

(Or Helm values file if testing springboot-helm)

Change something small, like:

**replicas: 1 # ➡ change to 2**

**2️⃣ Commit to the main branch**

**git add .**

**git commit -m "Increase replicas to 2"**

**git push origin main**

**3️⃣ Go to Argo CD UI**

* Open springboot-manifests
* You’ll see it auto-sync (no manual click)
* Check AKS pods:

**kubectl get pods**

**✅ You should now see 2 pods running.**

**🔙 STEP 2: Test Rollback (Git-based)**

**Let’s now simulate a rollback by reverting the last commit.**

**1️⃣ In Git repo:**

git revert HEAD

git commit -m "Rollback: replicas back to 1"

git push origin main

**2️⃣ Watch Argo CD**

* Argo CD detects the Git change
* It syncs again automatically
* It will scale down to 1 pod

kubectl get deployments springboot-app -o yaml | grep replicas

**Step 3: Bonus Test: Manual Drift Revert**

**kubectl scale deployment springboot-app --replicas=5**

1. Wait 15–30 seconds — Argo CD will detect the drift
2. It will automatically reset it back to Git value (1 or 2)

🧠 This is Self-Heal in action!

**Multi-cluster GitOps setup**

Awesome! You're now ready to take your GitOps setup to the next level by using Argo CD to deploy to multiple AKS clusters from the same Argo CD instance.

Argo CD connects to your AKS cluster the same way kubectl does — using a Kubernetes API token and cluster config.

**When you install Argo CD inside the AKS cluster, it can access the cluster via the internal Kubernetes service:**

[**https://kubernetes.default.svc**](https://kubernetes.default.svc)

**🎯 Goal**

**Use Argo CD (installed in aks-agic) to:**✅ Connect to aksdemo  
✅ Deploy apps from Git to aksdemo  
✅ Manage both clusters from one Argo CD UI

az aks get-credentials --resource-group <rg-agic> --name aks-agic

kubectl config current-context

az aks get-credentials --resource-group <rg-demo> --name aksdemo --context aksdemo

kubectl config use-context aks-agic

argocd cluster add aksdemo

kubectl config use-context aksdemo

kubectl get all -n default

kubectl config use-context aks-agic

**✅ Step-by-Step: Install argocd CLI on Ubuntu (or any Linux) - to interact with ARGOCD running on Kubernetes cluster.(remember Argocd is running Kubernetes cluster , I am installing this argocd cli tool)**

**🛠️ Step 1: Download the argocd binary**

VERSION=$(curl -s https://api.github.com/repos/argoproj/argo-cd/releases/latest | grep tag\_name | cut -d '"' -f 4)

curl -sSL -o argocd "https://github.com/argoproj/argo-cd/releases/download/${VERSION}/argocd-linux-amd64"

chmod +x argocd

sudo mv argocd /usr/local/bin/

argocd version

kubectl config use-context aks-agic # Ensure you're on the Argo CD cluster

argocd login <ARGOCD\_SERVER\_LB\_IP>:<PORT> --username admin --password <your-password>

argocd cluster add aksdemo

 Use your local kubectl credentials for aksdemo

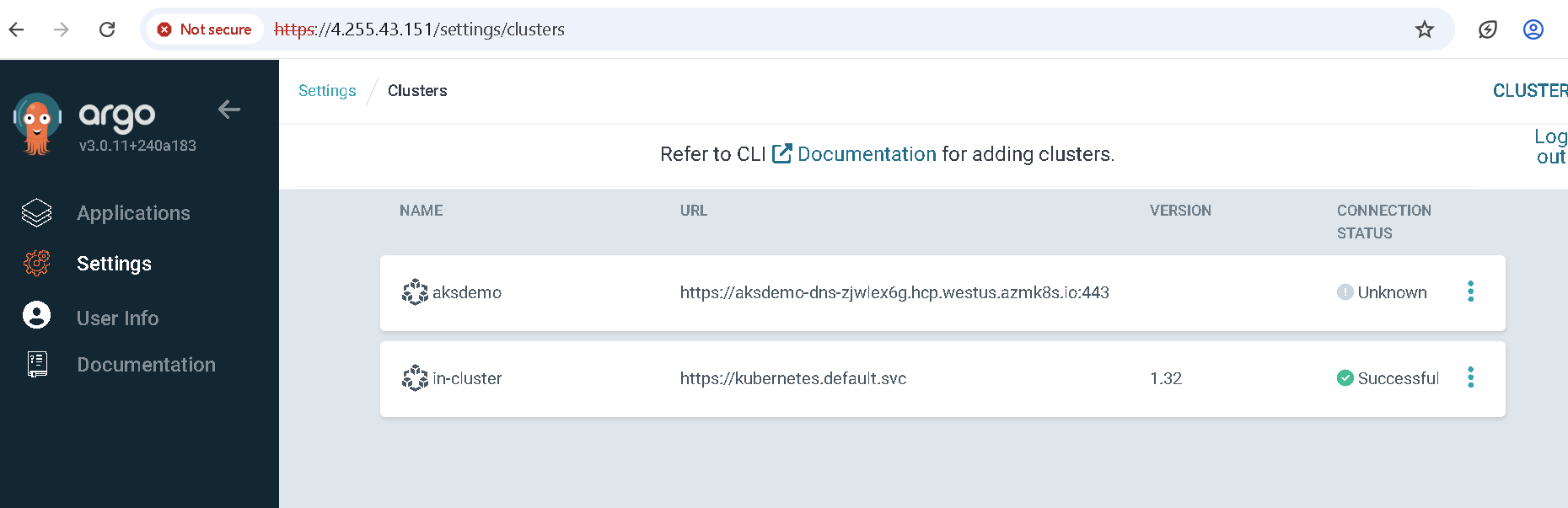
 Create a Secret in argocd namespace in aks-agic

 Register the new cluster for GitOps deployment

**✅ Step 3: Verify the Cluster Was Added**

In your Argo CD UI:

1. Go to **Settings → Clusters**
2. You should see:
   * https://kubernetes.default.svc for aks-agic
   * https://<aksdemo-api> for aksdemo



kubectl config use-context aks-agic

🔐 Step 2: Check the Cluster Secret in Argo CD

kubectl get secrets -n argocd | grep cluster

kubectl run -it curlpod --image=curlimages/curl --rm -- bash

curl <https://aksdemo-dns-xyz.hcp.westus.azmk8s.io:443>

✅ Step 4: Re-add the Cluster (Optional)

argocd cluster rm aksdemo

argocd cluster add aksdemo

To Get Kubernetes API server URL:

kubectl config use-context aksdemo

kubectl config view --minify -o jsonpath='{.clusters[0].cluster.server}'

az aks show --resource-group <resource-group-name> --name aksdemo --query "fqdn" -o tsv

**🛠️ Step 5: Update Argo CD RBAC (if needed)**

If the argocd-application-controller service account does not have permission to manage the remote cluster, Argo CD can't reconcile.

**kubectl get clusterrolebinding | grep argocd**

**Ensure a ClusterRoleBinding exists granting cluster-admin or sufficient roles to Argo CD.**

**From a pod in the aks-agic cluster (where Argo CD is installed), run:**

kubectl run curltest --image=curlimages/curl -it --rm – bash

curl -k <https://aksdemo-dns-zjwlex6g.hcp.westus.azmk8s.io:443>

**✅ 3. RBAC Check (Optional)**

Ensure Argo CD's controller can manage remote resources on aksdemo:

kubectl config use-context aksdemo

kubectl get clusterrolebinding | grep argocd

# argocd-clusterrolebinding.yaml

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: argocd-manager-binding

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: argocd-application-controller

namespace: argocd

**kubectl apply -f argocd-clusterrolebinding.yaml --context aksdemo**

**kubectl config use-context aks-agic**

**kubectl get secrets -n argocd | grep cluster**

**kubectl describe secret <secret-name-for-aksdemo> -n argocd**

**✅ 2. Try Re-adding the Cluster Cleanly**

**Sometimes the connection becomes stale or misconfigured. To refresh it:**

**argocd cluster rm aksdemo**

**argocd cluster add aksdemo**