**Argo CD Comprehensive Guide — For Shashank**

# 1. Introduction to Argo CD

**Argo CD** is a declarative, GitOps continuous delivery tool for Kubernetes. It automates the deployment and synchronization of Kubernetes applications using Git as the single source of truth.

### Key Concepts:

* **GitOps:** Git repository defines the desired state of your applications and environment.
* **Declarative:** Desired state is described in manifests (YAML/Helm/Kustomize).
* **Continuous Delivery:** Changes in Git are automatically applied to the Kubernetes cluster.

# 2. Argo CD Components

| Component | Purpose |
| --- | --- |
| argocd-server | API server + Web UI for managing apps |
| argocd-repo-server | Clones Git repos and generates manifests |
| argocd-application-controller | Watches Application CRDs and syncs apps |
| argocd-dex-server | Handles authentication (SSO/OIDC) |
| argocd-redis | Internal caching |

# 3. Minikube Setup & Argo CD Installation

### Install Minikube & Kubectl

sudo apt install -y kubectl  
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64  
sudo install minikube-linux-amd64 /usr/local/bin/minikube

### Start Minikube

minikube start --memory=4096 --cpus=2 --driver=docker  
kubectl get nodes

### Install Argo CD

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

### Install CLI

sudo curl -sSL -o /usr/local/bin/argocd https://github.com/argoproj/argo-cd/releases/latest/download/argocd-linux-amd64  
sudo chmod +x /usr/local/bin/argocd  
argocd version --client

### Access UI & Login

kubectl port-forward svc/argocd-server -n argocd 8080:443  
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d && echo  
argocd login localhost:8080 --username admin --password <password> --insecure

# 4. Creating a Simple Node.js App

### Folder Structure

argo-node-demo/  
 ├── app.js  
 ├── package.json  
 ├── Dockerfile  
 ├── kubernetes/  
 │ ├── deployment.yaml  
 │ ├── service.yaml  
 └── .gitignore

### app.js

const express = require('express');  
const app = express();  
const port = 3000;  
app.get('/', (req, res) => res.send('Hello World from Argo CD Demo!'));  
app.listen(port, () => console.log(`App running on port ${port}`));

### Dockerfile

FROM node:18  
WORKDIR /app  
COPY package\*.json ./  
RUN npm install  
COPY . .  
EXPOSE 3000  
CMD ["node", "app.js"]

### Kubernetes Manifests

#### deployment.yaml

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: argo-node-demo  
 labels:  
 app: argo-node-demo  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: argo-node-demo  
 template:  
 metadata:  
 labels:  
 app: argo-node-demo  
 spec:  
 containers:  
 - name: argo-node-demo  
 image: <dockerhub-username>/argo-node-demo:v1  
 ports:  
 - containerPort: 3000

#### service.yaml

apiVersion: v1  
kind: Service  
metadata:  
 name: argo-node-demo  
spec:  
 selector:  
 app: argo-node-demo  
 ports:  
 - port: 80  
 targetPort: 3000  
 type: NodePort

### Git Initialization & Push

git init  
git add .  
git commit -m "Initial commit - Node.js ArgoCD demo app"  
git remote add origin https://github.com/<username>/argo-node-demo.git  
git branch -M main  
git push -u origin main

# 5. Deploying with Argo CD

### Create Application via UI or CLI

argocd app create argo-node-demo \  
 --repo https://github.com/<username>/argo-node-demo.git \  
 --path kubernetes \  
 --dest-server https://kubernetes.default.svc \  
 --dest-namespace default \  
 --sync-policy automated \  
 --self-heal \  
 --prune

### Sync Application

argocd app sync argo-node-demo  
argocd app get argo-node-demo

### Verify Service

kubectl get all  
minikube service argo-node-demo --url

# 6. Behind the Scenes (Flow)

1. Developer pushes code + manifests to GitHub.
2. Argo CD Application CR in cluster watches Git repo.
3. Repo server clones repo and generates manifests.
4. Controller compares desired state (Git) vs actual state (cluster).
5. If out-of-sync → controller applies manifests automatically (or manually if required).
6. Cluster updated → status turns Synced & Healthy.

# 7. Sync Policies

| Mode | Description |
| --- | --- |
| Manual | You click sync in UI or CLI |
| Auto-Sync | Detects Git changes automatically |
| Auto-Sync with prune & self-heal | Removes deleted resources & corrects drift |

Configure Auto-Sync via CLI:

argocd app set argo-node-demo --sync-policy automated --self-heal --prune

# 8. Application CR vs Deployment/Service Manifests

| Type | Purpose | Location |
| --- | --- | --- |
| Deployment/Service YAML | Define app desired state | Git repo /kubernetes folder |
| Application YAML (CR) | Tells Argo CD how to find & sync app | Argo CD cluster (argocd namespace) |

* Deployment/Service = actual app definition
* Application CR = Argo CD instructions to manage the app

# 9. Real-World Organization Setup

* **Multi-repo pattern**: separate repo for application code & GitOps manifests
* **RBAC**: developers push code, DevOps manages Argo CD
* **Multi-environment deployments**: dev, staging, prod each with own Application CR
* **Audit & rollback**: all changes tracked in Git, easy to revert

# 10. Key Learning Summary

* Argo CD implements GitOps for Kubernetes
* Git = single source of truth, no manual kubectl in production
* Application CR defines the connection between Git repo and cluster
* Deployment & Service YAMLs = define your app
* Auto-sync keeps cluster in desired state
* Useful for production multi-cluster, multi-team setups

*This document covers all foundational concepts, setup steps, hands-on commands, and organizational best practices for Argo CD.*