GitOps Stack Blueprint

Docker, Kubernetes, Ansible, Debian, Terraform, NGINX Ingress, Argo CD,
OPA/Gatekeeper, Cloud Custodian, GitHub Actions & GitHub Advanced Security

October 23, 2025

Contents

1 Executive Summary 2

2 System Roles & Flow 2

3 Repository Layout (Suggested) 2

4 Minimal Working Snippets 3

4.1 Terraform: Cluster Provisioning (EKS placeholder) . . . 3

4.2 Ansible: Node Bootstrap . . . 3

4.3 Argo CD: ApplicationSet (GitOps Multi-Env) . . . 3

4.4 NGINX Ingress for Sample App . . . 4

4.5 Gatekeeper: Disallow Privileged Containers . . . 4

4.6 Cloud Custodian: Require S3 Encryption . . . 5

4.7 GitHub Actions: CI Build & GHAS . . . 5

5 Quickstart (Happy Path) 6

6 Security & Governance Notes 6

7 Operations (Day 2) 7

8 Next Steps 7

1 Executive Summary

This document provides an opinionated, end-to-end blueprint for a modern GitOps platform. It compiles roles, flow, and minimal working snippets for the following stack:

- **Provisioning:** Terraform (cloud primitives, cluster), Debian (base OS on nodes where applicable), Ansible (node bootstrap and hardening).
- **Runtime:** Docker (image build), Kubernetes (orchestrator), NGINX Ingress (L7 entry).
- **Delivery:** Argo CD for continuous delivery and drift reconciliation from Git.
- **Policy & Governance:** OPA/Gatekeeper (admission control), Cloud Custodian (cloud governance & remediation).
- **CI & Security:** GitHub Actions for CI and GitHub Advanced Security (CodeQL, secrets, dependency review).

2 System Roles & Flow

Step 1. **Terraform** provisions cloud networking, registry access, and a managed Kubernetes control plane (or bare metal/VM where desired). Debian may be used for node OS images.

Step 2. **Ansible** bootstraps nodes (Docker/CRI, kube prerequisites) and applies baseline hardening.

Step 3. **Docker** builds application images; images are pushed to a registry (e.g., GHCR/ECR/GCR).

Step 4. **Kubernetes** runs workloads; **NGINX Ingress** exposes services.

Step 5. **Argo CD** continuously syncs cluster state from Git; detects and heals drift.

Step 6. **OPA/Gatekeeper** enforces admission-time policies (e.g., disallow privileged pods).

Step 7. **Cloud Custodian** enforces cloud-level governance and can auto-remediate violations.

Step 8. **GitHub Actions & GHAS** run tests, build/push images, and scan code/secrets/dependencies.

3 Repository Layout (Suggested)

```
infra/
  terraform/           # VPC/cluster/IRSA/registries
  policies/cloudcustodian/
platform/
  ansible/             # roles: docker, k8s-prereqs, hardening
  k8s/                 # base namespaces, rbac, ingress-nginx, argocd/
  gatekeeper/          # constraints + templates
apps/
  sample-web/
    Dockerfile
    k8s/               # deployment, service, ingress
  appset/               # ArgoCD ApplicationSet
.github/
  workflows/            # ci.yml, deploy.yml, security.yml
```

Listing 1: Top-level repository structure

4 Minimal Working Snippets

4.1 Terraform: Cluster Provisioning (EKS placeholder)

```
terraform { required_version = ">= 1.6.0" }

provider "aws" { region = var.region }

module "cluster" {
  source      = "terraform-aws-modules/eks/aws"
  cluster_name = var.name
  vpc_id       = var.vpc_id
  subnet_ids   = var.subnet_ids
}
# Optionally: Helm releases for ingress-nginx and argocd via Terraform.
```

Listing 2: infra/terraform/main.tf

4.2 Ansible: Node Bootstrap

```
- hosts: workers
  become: true
  roles:
    - docker
    - k8s_prereqs
    - hardening
```

Listing 3: platform/ansible/playbooks/bootstrap.yaml

4.3 Argo CD: ApplicationSet (GitOps Multi-Env)

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata: { name: apps }
spec:
  generators:
    - list:
        elements:
          - { name: sample-web, namespace: web, path: apps/sample-web/k8s,
              env: dev }
  template:
    metadata: { name: '{{name}}-{{env}}' }
    spec:
      project: default
      source:
        repoURL: https://github.com/your/org.git
        targetRevision: main
        path: ’{{path}}’
      destination:
        server: https://kubernetes.default.svc
        namespace: ’{{namespace}}’
      syncPolicy: { automated: { prune: true, selfHeal: true } }
```

Listing 4: apps/appset/apps.yaml

4.4 NGINX Ingress for Sample App

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: sample-web
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
    - host: sample.local
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: sample-web
                port:
                  number: 80
```

Listing 5: apps/sample-web/k8s/ingress.yaml

4.5 Gatekeeper: Disallow Privileged Containers

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata: { name: k8spspprivilegedcontainer }
spec:
  crd:
    spec:
      names: { kind: K8sPSPPrivilegedContainer }
  targets:
  - target: admission.k8s.gatekeeper.sh
    rego: |
      package k8spspprivileged
      violation[{”msg”: msg}] {
        input.review.object.spec.containers[_].securityContext.privileged == true
        msg := ”Privileged containers are not allowed”
      }
---
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sPSPPrivilegedContainer
metadata: { name: disallow-privileged }
spec: {}
```

Listing 6: platform/gatekeeper/templates/k8spsp-privileged.yaml

4.6 Cloud Custodian: Require S3 Encryption

```
policies:
  - name: s3-require-encryption
    resource: s3
    filters:
      - type: bucket-encryption
        state: false
    actions:
      - type: set-bucket-encryption
        crypto: aws:kms
```

Listing 7: infra/policies/cloudcustodian/enforce-bucket-encryption.yml

4.7 GitHub Actions: CI Build & GHAS

```
name: ci
on: [push]

jobs:
  build-test-scan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-node@v4
      - run: npm ci && npm test

      - name: Build image
        run: docker build -t ghcr.io/your/app:${{ github.sha }} apps/sample-web

      - name: Push image
        run: |
          echo "${{ secrets.CR_PAT }}" | docker login ghcr.io \
            -u ${{ github.actor }} --password-stdin
          docker push ghcr.io/your/app:${{ github.sha }}

      - name: CodeQL
        uses: github/codeql-action/analyze@v3

      - name: Secret scanning (GHAS native)
        run: echo "Enable at repo settings"

      - name: Update K8s manifest image
        run: |
          sed -i "s#image: .*#image: ghcr.io/your/app:${GITHUB_SHA}#" \
            apps/sample-web/k8s/deployment.yaml
          git config user.name bot
          git config user.email bot@users.noreply.github.com
          git commit -am "bump image"
          git push
```

Listing 8: .github/workflows/ci.yml

5 Quickstart (Happy Path)

Q1. Bootstrap infra with Terraform. Create VPC/subnets, registry, and cluster. Optionally manage Helm releases for ingress-nginx and Argo CD.

Q2. Harden nodes with Ansible. Install CRI/Docker, kube prerequisites, and security baselines.

Q3. Install Argo CD. Point to the Git repo; commit the base platform manifests (namespaces, RBAC, ingress, Gatekeeper).

Q4. Apply policies. Gatekeeper constraints and Cloud Custodian policies for guardrails and remediation.

Q5. Wire CI. GitHub Actions builds/tests, pushes images, scans with GHAS, and bumps manifest tags so Argo CD deploys.

6 Security & Governance Notes

- Enforce least-privilege via IRSA (AWS) or workload identity (GCP) and namespace-scoped RBAC.
- Use Gatekeeper libraries for common controls (privileged, hostPath, runAsNonRoot, image provenance).
- Enable GHAS features: CodeQL, secret scanning, Dependabot security updates.
- Apply Cloud Custodian for encryption at rest, public exposure checks, and lifecycle policies.

7 Operations (Day 2)

- **Drift & Sync:** Use Argo CD automated sync and health checks; protect prod with PR + manual sync if desired.
- **Environments:** Promote via Git branches or ApplicationSet generators (folder-per-env); avoid kubectl-by-hand.
- **Observability:** Add metrics/logs/traces; gate production with SLOs and rollout strategies (e.g., canary).

8 Next Steps

- Add Helm/Kustomize overlays per environment.
- Expand Gatekeeper constraints and Custodian policies to your regulatory profile.
- Introduce image signing (Sigstore/cosign) and policy checks (Conftest/OPA) in CI.

This document is a practical compilation intended as a quick-start blueprint. Adapt module versions, cloud providers, and security baselines to your environment.