Contents

Confluent Platform GitOps with ArgoCD

Complete Guide to Declarative Kafka Management on Kubernetes

Date: February 2, 2026
Author: Confluent Federal
Version: 2.0

Table of Contents

Executive Summary

This white paper documents a production-ready GitOps implementation for Confluent Platform on Kubernetes using ArgoCD. The solution provides:

- **Declarative Infrastructure** - All components defined as Helm charts
- **GitOps Deployment** - Git as the single source of truth
- **Modular Architecture** - Independent lifecycle for each application
- **Self-Healing** - ArgoCD automatically reverts configuration drift
- **Multi-Environment Support** - Branch and values-based configuration

Key Technologies

Technology	Purpose
Confluent Platform	Event streaming platform
CFK Operator	Kubernetes operator for Confluent
ArgoCD	GitOps continuous delivery
Helm	Package management and templating
Flink	Stream processing
Kafka Streams	Stateful stream processing

What This Guide Covers

- Installing and configuring ArgoCD
- Deploying Confluent Platform components
- Managing Kafka Streams applications (content routers)
- Managing Flink applications
- Environment-specific configurations
- Monitoring, troubleshooting, and operations

Architecture Overview

```
 Git Repo             ArgoCD            Kubernetes
 (Source of             (GitOps               Cluster
  Truth)                Controller)




 Helm Charts            Auto-Sync             Running Apps:
 - confluent-platform   - Detect changes      - Kafka (KRaft)
```

```
   - content-router        - Apply manifests      - Schema Registry
   - flink-application     - Self-heal drift      - Kafka Connect
   - syslog-reconstruction                        - ksqlDB
   - datagen-connectors                           - Control Center
                                                  - Flink Jobs
                                                  - KStreams Apps
```

Repository Structure

```
cfk-argocd-demo/
  argocd/
     applications/                   # ArgoCD Application manifests
        confluent-platform-prod.yaml
        confluent-platform-dev.yaml
        content-router-prod.yaml
        content-router-syslog.yaml
        content-router-akamai.yaml
        flink-state-machine.yaml
        flink-kafka-streaming.yaml
        flink-hostname-enrichment.yaml
        syslog-reconstruction.yaml
        datagen-connectors-prod.yaml
     project.yaml                    # ArgoCD Project definition
  charts/
     confluent-platform/             # Core platform chart
        Chart.yaml
        values.yaml                 # Default values
        values-dev.yaml             # Dev overrides
        values-prod.yaml            # Prod overrides
        templates/
           kafka.yaml
           kraftcontroller.yaml
           schemaregistry.yaml
           connect.yaml
           ksqldb.yaml
           controlcenter.yaml
           flink.yaml               # Flink environment
     content-router/                 # KStreams routing app
        Chart.yaml
        values.yaml
        values-syslog.yaml
        values-akamai.yaml
        templates/
     flink-application/              # Flink job chart
        Chart.yaml
        values.yaml
        values-state-machine.yaml
        values-kafka-streaming.yaml
```

```
            values-hostname-enrichment.yaml
            templates/
        syslog-reconstruction/          # Specialized KStreams
            Chart.yaml
            values.yaml
            templates/
        datagen-connectors/             # Connector definitions
            Chart.yaml
            values.yaml
            templates/
    docs/
        *.md                            # Documentation
```

Prerequisites

Required Tools

```
# Kubernetes CLI
kubectl version --client

# Helm
helm version

# Git
git --version

# GitHub CLI (optional, for repo management)
gh --version
```

Kubernetes Cluster Requirements

- Kubernetes 1.25+
- Sufficient resources (minimum 3 nodes, 4 CPU / 16GB RAM each)
- Storage class configured (e.g., gp3, standard)
- CFK Operator installed

CFK Operator Installation

```
# Add Confluent Helm repository
helm repo add confluentinc https://packages.confluent.io/helm
helm repo update

# Create namespace
kubectl create namespace confluent-operator

# Install CFK Operator
helm upgrade --install confluent-operator \
  confluentinc/confluent-for-kubernetes \
  --namespace confluent-operator \
  --set namespaced=false \
  --wait

# Verify CRDs
kubectl get crds | grep confluent
```

ArgoCD Installation & Configuration

Install ArgoCD

```
# Create namespace
kubectl create namespace argocd

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

# Wait for ArgoCD to be ready
kubectl wait --for=condition=ready pod \
  -l app.kubernetes.io/name=argocd-server \
  -n argocd \
  --timeout=300s
```

Expose ArgoCD UI

```
# Expose via NodePort
kubectl patch svc argocd-server -n argocd \
  -p '{"spec": {"type": "NodePort"}}'

# Get access details
ARGOCD_PORT=$(kubectl get svc argocd-server -n argocd \
  -o jsonpath='{.spec.ports[0].nodePort}')

NODE_IP=$(kubectl get nodes \
  -o jsonpath='{.items[0].status.addresses[?(@.type=="ExternalIP")].address}')

ARGOCD_PASSWORD=$(kubectl -n argocd get secret argocd-initial-admin-secret \
  -o jsonpath="{.data.password}" | base64 -d)

echo "URL: https://$NODE_IP:$ARGOCD_PORT"
echo "Username: admin"
echo "Password: $ARGOCD_PASSWORD"
```

Connect Repository

Via ArgoCD UI: 1. **Settings** → **Repositories** → **+ Connect Repo** 2. Choose **VIA HTTPS** 3. Enter repository URL and GitHub credentials (PAT) 4. Click **Connect**

Via kubectl:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: repo-cfk-argocd-demo
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://github.com/confluentfederal/cfk-argocd-demo.git
  username: <github-username>
  password: <github-pat>
EOF
```

Create ArgoCD Project

```yaml
# argocd/project.yaml
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: confluent
  namespace: argocd
spec:
  description: Confluent Platform GitOps Project
  sourceRepos:
    - https://github.com/confluentfederal/cfk-argocd-demo.git
  destinations:
    - namespace: confluent
      server: https://kubernetes.default.svc
    - namespace: argocd
      server: https://kubernetes.default.svc
  clusterResourceWhitelist:
    - group: '*'
      kind: '*'
```

```
kubectl apply -f argocd/project.yaml
```

Helm Charts Structure

Chart Pattern

Each application follows a consistent Helm chart structure:

```
charts/<app-name>/
  Chart.yaml               # Chart metadata
  values.yaml              # Default configuration
  values-<env>.yaml       # Environment-specific overrides
  templates/
      _helpers.tpl        # Template helpers
      deployment.yaml     # Kubernetes Deployment
      configmap.yaml      # ConfigMap for app config
      ...                 # Additional resources
```

ArgoCD Application Pattern

Each ArgoCD Application references a Helm chart with specific values:

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: <app-name>
  namespace: argocd
spec:
  project: confluent
  source:
    repoURL: https://github.com/confluentfederal/cfk-argocd-demo.git
    targetRevision: main
    path: charts/<chart-name>
    helm:
      releaseName: <release-name>
      valueFiles:
        - values-<env>.yaml
  destination:
    server: https://kubernetes.default.svc
    namespace: confluent
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

Deploying Confluent Platform

Chart: confluent-platform

The core platform chart deploys: - KRaft Controllers - Kafka Brokers - Schema Registry - Kafka Connect - ksqlDB - Control Center - Flink Environment (CMF + Operator)

Deploy Production Environment

```bash
kubectl apply -f argocd/applications/confluent-platform-prod.yaml
```

ArgoCD Application:

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
```

```yaml
metadata:
  name: confluent-platform-prod
  namespace: argocd
spec:
  project: confluent
  source:
    repoURL: https://github.com/confluentfederal/cfk-argocd-demo.git
    targetRevision: helm-migration
    path: charts/confluent-platform
    helm:
      releaseName: confluent
      valueFiles:
        - values.yaml
        - values-prod.yaml
  destination:
    server: https://kubernetes.default.svc
    namespace: confluent
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

Verify Deployment

```bash
# Check ArgoCD status
kubectl get application confluent-platform-prod -n argocd

# Check Confluent components
kubectl get kafka,connect,schemaregistry,ksqldb,controlcenter -n confluent

# Check pods
kubectl get pods -n confluent
```

Deploying Kafka Streams Applications

Chart: content-router

A reusable chart for Kafka Streams content routing applications.

Pattern: Multiple Instances from One Chart

Each content router instance uses the same chart with different values:

Instance	Values File	Purpose
content-router-prod	values.yaml	Default routing
content-router-syslog	values-syslog.yaml	Syslog routing
content-router-akamai	values-akamai.yaml	Akamai CDN routing

Deploy a Content Router

```
kubectl apply -f argocd/applications/content-router-syslog.yaml
```

ArgoCD Application:

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: content-router-syslog
  namespace: argocd
spec:
  project: confluent
  source:
    repoURL: https://github.com/confluentfederal/cfk-argocd-demo.git
    targetRevision: helm-migration
    path: charts/content-router
    helm:
      releaseName: content-router-syslog
      valueFiles:
        - values-syslog.yaml
  destination:
    server: https://kubernetes.default.svc
    namespace: confluent
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

Adding a New Content Router Instance

1. Create values file: `charts/content-router/values-<name>.yaml`
2. Create ArgoCD application: `argocd/applications/content-router-<name>.yaml`
3. Commit and push to Git
4. Apply the ArgoCD application

Deploying Flink Applications

Chart: flink-application

A reusable chart for deploying Flink jobs via FlinkApplication CRD.

Prerequisites

The Confluent Platform chart creates: - CMFRestClass (Confluent Manager for Flink) - FlinkEnvironment (`flink-env`)

Deploy a Flink Application

```
kubectl apply -f argocd/applications/flink-kafka-streaming.yaml
```

Values file example:

```yaml
# charts/flink-application/values-kafka-streaming.yaml
namespace: confluent
flinkEnvironment: flink-env

image:
  repository: cstevenson954/java-flink
  tag: "0.0.3"

flinkVersion: v1_20
serviceAccount: flink

flinkConfiguration:
  taskmanager.numberOfTaskSlots: "2"

jobManager:
  resource:
    memory: "1024m"
    cpu: "0.5"

taskManager:
  resource:
    memory: "1024m"
    cpu: "0.5"

job:
  jarURI: local:///opt/flink/usrlib/java-flink-0.0.1.jar
  entryClass: com.example.javaflink.StreamingJob
  state: running
  parallelism: 1
  upgradeMode: stateless
  args:
    - "--bootstrap.servers=kafka.confluent.svc.cluster.local:9071"
    - "--input.topic=flink-input"
    - "--output.topic=flink-output"
```

Verify Flink Deployment

```bash
# Check FlinkApplication
kubectl get flinkapplication -n confluent

# Check FlinkDeployment (created by CMF)
kubectl get flinkdeployments -n confluent

# Check pods
kubectl get pods -n confluent | grep flink
```

Configuration Management

Values File Hierarchy

1. `values.yaml` - Base defaults
2. `values-<env>.yaml` - Environment overrides (merged on top)

Example: Environment-Specific Resources

values.yaml (defaults):

```yaml
kafka:
  replicas: 3
  resources:
    requests:
      cpu: "1"
      memory: "4Gi"
```

values-dev.yaml (dev overrides):

```yaml
kafka:
  replicas: 1
  resources:
    requests:
      cpu: "500m"
      memory: "2Gi"
```

values-prod.yaml (prod overrides):

```yaml
kafka:
  replicas: 3
  resources:
    requests:
      cpu: "2"
      memory: "8Gi"
```

Applying Configuration Changes

1. Edit the values file in Git
2. Commit and push
3. ArgoCD auto-syncs (or force refresh):

```bash
kubectl annotate application <app-name> -n argocd \
  argocd.argoproj.io/refresh=hard --overwrite
```

Branch-Based Deployments

Strategy: targetRevision

Use Git branches to manage different environments or feature deployments.

Branch	Purpose
`main`	Production-ready code
`develop`	Integration testing
`feature/*`	Feature development
`helm-migration`	Helm chart development

Switching Branches

Via ArgoCD UI: 1. Click application → **App Details** → **Edit** 2. Change **Target Revision** to desired branch 3. Save

Via kubectl:

```
kubectl patch application confluent-platform-prod -n argocd \
  --type merge -p '{"spec":{"source":{"targetRevision":"develop"}}}'
```

Multi-Environment Setup

Deploy same chart to different environments using different ArgoCD Applications:

```
# Dev environment
spec:
  source:
    targetRevision: develop
    helm:
      valueFiles:
        - values-dev.yaml

# Prod environment
spec:
  source:
    targetRevision: main
    helm:
      valueFiles:
        - values-prod.yaml
```

Scaling Applications

Scaling Kafka Streams Apps

Update `replicas` in values file:

```
# charts/content-router/values-syslog.yaml
replicas: 4  # Scale from 2 to 4
```

Commit, push, and ArgoCD applies the change.

Scaling Flink Jobs

Update `parallelism` and resources:

```
# charts/flink-application/values-kafka-streaming.yaml
job:
  parallelism: 4

taskManager:
  resource:
    memory: "2048m"
    cpu: "1"
```

Scaling Kafka Brokers

Update broker count in values:

```
# charts/confluent-platform/values-prod.yaml
kafka:
  replicas: 5
```

Note: Scaling down Kafka requires partition reassignment.

Secrets Management

Option 1: Kubernetes Secrets (Basic)

Create secrets manually, reference in values:

```
kubectl create secret generic kafka-credentials \
  -n confluent \
  --from-literal=username=admin \
  --from-literal=password=secret
```

```
# In values.yaml
kafka:
  authentication:
    secretRef: kafka-credentials
```

Option 2: External Secrets Operator

Integrate with AWS Secrets Manager, HashiCorp Vault, etc.:

```
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: kafka-credentials
  namespace: confluent
spec:
  secretStoreRef:
    name: aws-secrets-manager
    kind: ClusterSecretStore
  target:
    name: kafka-credentials
  data:
    - secretKey: username
      remoteRef:
        key: confluent/kafka
        property: username
    - secretKey: password
      remoteRef:
        key: confluent/kafka
        property: password
```

Option 3: Sealed Secrets

Encrypt secrets in Git:

```
# Install kubeseal CLI
# Encrypt secret
kubeseal --format yaml < secret.yaml > sealed-secret.yaml
```

Store `sealed-secret.yaml` in Git; SealedSecrets controller decrypts at runtime.

CI/CD Integration

GitHub Actions Example

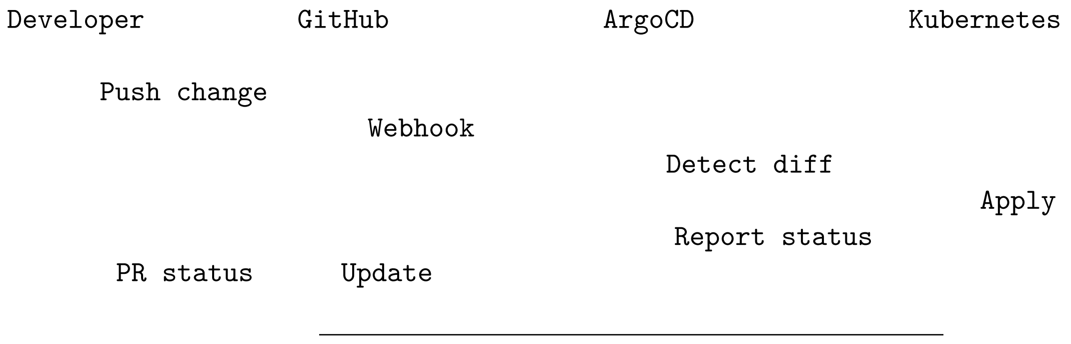
```
# .github/workflows/deploy.yml
name: Deploy to ArgoCD

on:
  push:
    branches: [main]
    paths:
      - 'charts/**'

jobs:
  sync:
    runs-on: ubuntu-latest
    steps:
      - name: Trigger ArgoCD Sync
        run: |
          curl -X POST \
            -H "Authorization: Bearer ${{ secrets.ARGOCD_TOKEN }}" \
            "https://${{ secrets.ARGOCD_SERVER }}/api/v1/applications/confluent-platform-prod/s
```

ArgoCD Webhook Integration

Configure GitHub webhook to notify ArgoCD of changes:

1. ArgoCD Settings → Repositories → Configure webhook
2. Add webhook URL to GitHub repository settings
3. ArgoCD immediately syncs on push

GitOps Workflow



Monitoring & Observability

ArgoCD Status Monitoring

```
# List all applications with status
kubectl get applications -n argocd

# Watch application status
kubectl get application <name> -n argocd -w

# Get sync details
kubectl get application <name> -n argocd \
  -o jsonpath='{.status.sync.status} / {.status.health.status}'
```

Control Center Access

```
# Get Control Center URL
NODE_IP=$(kubectl get nodes -o jsonpath='{.items[0].status.addresses[?(@.type=="ExternalIP")].a
CC_PORT=$(kubectl get svc controlcenter -n confluent -o jsonpath='{.spec.ports[0].nodePort}')
echo "http://$NODE_IP:$CC_PORT"
```

Application Logs

```
# Kafka Streams app logs
kubectl logs -l app=content-router-syslog -n confluent --tail=100

# Flink job manager logs
kubectl logs -l app=kafka-streaming-job,component=jobmanager -n confluent

# Flink task manager logs
kubectl logs -l app=kafka-streaming-job,component=taskmanager -n confluent
```

```
# Connect worker logs
kubectl logs -l app=connect -n confluent --tail=100
```

Flink Dashboard

```
# Port-forward to Flink UI
kubectl port-forward svc/kafka-streaming-job-rest 8081:8081 -n confluent
# Access at http://localhost:8081
```

Metrics & Alerting

For production, integrate with: - **Prometheus** - Metrics collection - **Grafana** - Dashboards - **AlertManager** - Alerting

CFK exposes metrics endpoints on all components.

Operational Commands Reference

ArgoCD Commands

Command	Description
`kubectl get applications -n argocd`	List all applications
`kubectl get application <name> -n argocd -o yaml`	View application details
`kubectl annotate application <name> -n argocd argocd.argoproj.io/refresh=hard`	Force sync
`kubectl delete application <name> -n argocd`	Delete application

Confluent Platform Commands

Command	Description
`kubectl get kafka,connect,schemaregistry -n confluent`	List CP components
`kubectl get connector -n confluent`	List connectors
`kubectl get flinkapplication -n confluent`	List Flink apps
`kubectl get flinkdeployments -n confluent`	List Flink deployments

Kafka Topic Management

Command	Description
`kubectl exec kafka-0 -n confluent -- kafka-topics --list --bootstrap-server localhost:9092`	List topics
`kubectl exec kafka-0 -n confluent -- kafka-topics --create --topic <name> --partitions 3 --replication-factor 3 --bootstrap-server localhost:9092`	Create topic
`kubectl exec kafka-0 -n confluent -- kafka-console-consumer --topic <name> --bootstrap-server localhost:9092 --from-beginning`	Consume messages

Troubleshooting Guide

ArgoCD Sync Issues

Application stuck in "Syncing":

```
# Force refresh
kubectl annotate application <name> -n argocd \
  argocd.argoproj.io/refresh=hard --overwrite

# Clear stuck operation
kubectl patch application <name> -n argocd \
  --type merge -p '{"operation": null}'
```

Application "OutOfSync":

```
# Check diff
kubectl get application <name> -n argocd -o jsonpath='{.status.sync.revision}'

# Force sync
kubectl patch application <name> -n argocd \
  --type merge -p '{"operation": {"initiatedBy": {"username": "admin"}, "sync": {}}}'
```

Flink Application Issues

FlinkApplication stuck in CREATED:

```
# Check CMF logs
kubectl logs deployment/confluent-manager-for-apache-flink -n confluent --tail=50

# Restart CMF
kubectl rollout restart deployment/confluent-manager-for-apache-flink -n confluent
```

FlinkDeployment FAILED:

```
# Get error details
kubectl describe flinkdeployment <name> -n confluent | grep -A10 "Error:"

# Check job manager logs
kubectl logs -l app=<name>,component=jobmanager -n confluent
```

Common Flink Errors:

Error	Cause	Fix
`NoClassDefFoundError`	Missing dependency in JAR	Fix Maven shade plugin order
`Insufficient cpu`	Cluster resource limits	Reduce resource requests
`Memory configuration failed`	Memory too low	Increase to minimum 1024m

Kafka Streams Issues

Pod CrashLoopBackOff:

```
# Check logs
kubectl logs <pod-name> -n confluent --previous

# Common causes:
# - Missing Kafka topics
# - Invalid bootstrap servers
# - Configuration errors
```

Create missing topics:

```
kubectl exec kafka-0 -n confluent -- kafka-topics \
  --create --topic <name> \
  --partitions 3 --replication-factor 3 \
  --bootstrap-server localhost:9092
```

Resource Issues

Pods Pending (Insufficient CPU/Memory):

```
# Check node resources
kubectl describe nodes | grep -A 10 "Allocated resources"

# Reduce application resources in values.yaml
```

```yaml
resources:
  requests:
    cpu: "100m"
    memory: "512Mi"
```

Summary

This GitOps implementation provides:

Benefit	Description
Single Source of Truth	All configuration in Git
Audit Trail	Git history tracks all changes
Rollback	Revert to any previous state
Self-Healing	ArgoCD reverts manual drift
Multi-Environment	Values files and branches
Modular	Independent app lifecycles

Current Deployments

Application Type	Count	Chart
Confluent Platform	1	confluent-platform
Content Routers	3	content-router
Flink Applications	4	flink-application
Syslog Reconstruction	1	syslog-reconstruction

Resources

- **Repository:** https://github.com/confluentfederal/cfk-argocd-demo
- **CFK Documentation:** https://docs.confluent.io/operator/current/overview.html
- **ArgoCD Documentation:** https://argo-cd.readthedocs.io/
- **Flink on Confluent:** https://docs.confluent.io/platform/current/flink/overview.html

---

Document Version: 2.0 | Last Updated: February 2, 2026