Contents

Confluent Platform GitOps with ArgoCD

Complete Guide to Declarative Kafka Management on Kubernetes

Date: January 28, 2026
Author: Confluent Federal
Version: 1.0

Table of Contents

13. Cleanup

Executive Summary

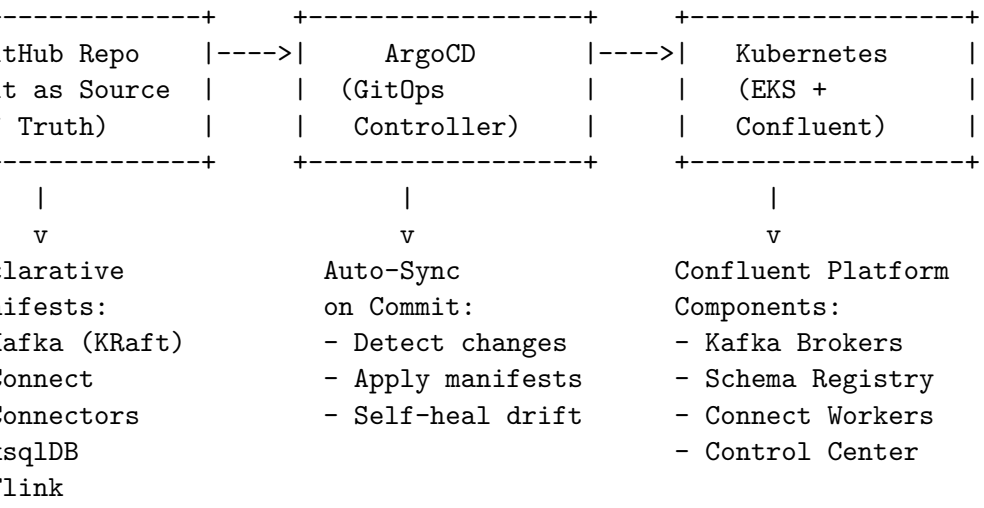
This white paper documents the complete implementation of a GitOps-based Confluent Platform deployment on AWS EKS using ArgoCD. The solution demonstrates:

- **Infrastructure as Code** - EKS cluster defined in YAML
- **GitOps Deployment** - All Confluent components managed via Git
- **Declarative Connectors** - Kafka Connect connectors defined as Kubernetes CRDs
- **Branch-Based Configuration** - Different environments/configs via Git branches
- **Self-Healing Infrastructure** - ArgoCD automatically reverts drift

Key Technologies

Technology	Version	Purpose
AWS EKS	1.31	Kubernetes cluster
Confluent Platform	8.1	Event streaming platform
CFK Operator	3.1	Kubernetes operator for CP
ArgoCD	Latest stable	GitOps continuous delivery
Kustomize	Built-in	Configuration management

Architecture Overview



Repository Structure

```
cfk-argocd-demo/
|-- argocd-application.yaml      # ArgoCD Application manifest
|-- eks-cluster-config.yaml      # EKS cluster definition
|-- base/
```

```
|   |-- kustomization.yaml       # Base kustomization
|   |-- confluent-platform.yaml  # Core CP components
|   |-- connectors.yaml          # Declarative connectors
|   |-- ksqldb-queries.yaml      # ksqlDB queries as code
|   |-- kstreams-app.yaml        # Kafka Streams app
|   +-- flink.yaml               # Flink resources (optional)
|-- overlays/
|   |-- dev/                     # Development overrides
|   +-- prod/                    # Production overrides
+-- docs/
    +-- CONFLUENT-PLATFORM-GITOPS-WHITEPAPER.md
```

Prerequisites

Required Tools

```bash
# AWS CLI
aws --version

# eksctl
eksctl version

# kubectl
kubectl version --client

# Helm
helm version

# GitHub CLI
gh --version
```

AWS Permissions

- EKS cluster creation
- EC2 instance management
- EBS volume provisioning
- IAM role management

Part 1: EKS Cluster Setup

1.1 Create EKS Cluster Configuration

Create `eks-cluster-config.yaml`:

```yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
```

```yaml
metadata:
  name: cfk-argocd-demo
  region: us-east-1
  version: "1.31"
  tags:
    owner_email: your-email@company.com
    environment: demo

vpc:
  nat:
    gateway: Disable

managedNodeGroups:
  - name: cfk-argocd-nodes
    instanceType: m5.xlarge
    desiredCapacity: 3
    minSize: 3
    maxSize: 6
    privateNetworking: false
    volumeSize: 100
    labels:
      role: confluent
```

1.2 Create the Cluster

```
# Create EKS cluster (takes 15-20 minutes)
eksctl create cluster -f eks-cluster-config.yaml
```

1.3 Install EBS CSI Driver

```
# Install EBS CSI addon
eksctl create addon \
  --name aws-ebs-csi-driver \
  --cluster cfk-argocd-demo \
  --region us-east-1

# Get node role name
ROLE_NAME=$(aws iam list-roles \
  --query "Roles[?contains(RoleName, 'eksctl-cfk-argocd-demo-nodegroup')].RoleName" \
  --output text)

# Attach EC2 permissions
aws iam attach-role-policy \
  --role-name "$ROLE_NAME" \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2FullAccess
```

1.4 Create Storage Class

```
kubectl apply -f - <<EOF
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gp3
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.aws.com
parameters:
  type: gp3
  encrypted: "true"
volumeBindingMode: WaitForFirstConsumer
EOF
```

1.5 Verify Cluster

```
# Check nodes
kubectl get nodes

# Expected output:
# NAME                          STATUS   ROLES    AGE   VERSION
# ip-xxx-xxx-xxx-xxx...         Ready    <none>   5m    v1.31.x
# ip-xxx-xxx-xxx-xxx...         Ready    <none>   5m    v1.31.x
# ip-xxx-xxx-xxx-xxx...         Ready    <none>   5m    v1.31.x
```

Part 2: Install CFK Operator

2.1 Add Confluent Helm Repository

```
helm repo add confluentinc https://packages.confluent.io/helm
helm repo update
```

2.2 Create Operator Namespace

```
kubectl create namespace confluent-operator
```

2.3 Install CFK Operator

```
helm upgrade --install confluent-operator \
  confluentinc/confluent-for-kubernetes \
  --namespace confluent-operator \
  --set namespaced=false \
  --wait
```

2.4 Verify Installation

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

# Check CRDs installed
kubectl get crds | grep confluent
```

Expected CRDs: - kafkas.platform.confluent.io - kafkarestclasses.platform.confluent.io - connects.platform.confluent.io - connectors.platform.confluent.io - schemaregistries.platform.con - ksqldbs.platform.confluent.io - controlcenters.platform.confluent.io

---

Part 3: Install and Configure ArgoCD

3.1 Create ArgoCD Namespace

```
kubectl create namespace argocd
```

3.2 Install ArgoCD

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

3.3 Wait for ArgoCD to be Ready

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

3.4 Expose ArgoCD UI (NodePort)

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

3.5 Get ArgoCD Access Details

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

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

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

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

3.6 Access ArgoCD

Open the URL in your browser and login with the credentials above.

Part 4: Deploy Confluent Platform via GitOps

4.1 Create ArgoCD Application Manifest

Create `argocd-application.yaml`:

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: confluent-platform
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/confluentfederal/cfk-argocd-demo.git
    targetRevision: HEAD
    path: overlays/prod
  destination:
    server: https://kubernetes.default.svc
    namespace: confluent
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true
      - PrunePropagationPolicy=foreground
    retry:
      limit: 5
      backoff:
        duration: 5s
        factor: 2
        maxDuration: 3m
```

4.2 Deploy the Application

```
kubectl apply -f argocd-application.yaml
```

4.3 Monitor Deployment

```
# Watch ArgoCD application status
kubectl get application confluent-platform -n argocd -w

# Watch pods coming up
kubectl get pods -n confluent -w
```

4.4 Verify Deployment

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

Expected Components: - Kafka (3 brokers in KRaft mode) - Schema Registry - Kafka Connect - ksqlDB - Control Center

---

Part 5: Expose Services for External Access

5.1 Expose Control Center via NodePort

Update `base/confluent-platform.yaml` to add external access:

```
apiVersion: platform.confluent.io/v1beta1
kind: ControlCenter
metadata:
  name: controlcenter
  namespace: confluent
spec:
  # ... existing config ...
  externalAccess:
    type: nodePort
    nodePort:
      host: <NODE_EXTERNAL_IP>
      nodePortOffset: 30021
```

5.2 Get Node IP and Update Config

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

echo "Use this IP in externalAccess.nodePort.host: $NODE_IP"
```

5.3 Commit and Push Changes

```
cd /path/to/cfk-argocd-demo
git add base/confluent-platform.yaml
git commit -m "Add external access for Control Center"
git push origin main
```

5.4 ArgoCD Auto-Syncs

ArgoCD detects the Git change and automatically applies it. Monitor in UI or:

```
kubectl get application confluent-platform -n argocd \
  -o jsonpath='{.status.sync.status}'
```

5.5 Access Control Center

```
# Get the NodePort
CC_PORT=$(kubectl get svc controlcenter-bootstrap -n confluent \
  -o jsonpath='{.spec.ports[0].nodePort}')

echo "Control Center: http://$NODE_IP:$CC_PORT"
```

Part 6: Connector GitOps Demo

This section demonstrates managing Kafka Connect connectors via Git with branch-based configuration.

6.1 Create Connector Demo Repository

```
# Clone baseline connector code
git clone https://github.com/confluentfederal/kafka-connect-netty-source.git \
  kafka-connect-netty-source-argo-example

cd kafka-connect-netty-source-argo-example

# Initialize fresh git history
rm -rf .git
git init
git add .
git commit -m "Initial commit - Netty Source Connector baseline"

# Create GitHub repo and push
gh repo create confluentfederal/kafka-connect-netty-source-argo-example \
  --public --source=. --remote=origin --push
```

6.2 Add Kubernetes Deployment Manifests

Create `deploy/base/connector.yaml`:

```
apiVersion: platform.confluent.io/v1beta1
kind: Connector
metadata:
  name: datagen-gitops-demo
  namespace: confluent
spec:
  class: io.confluent.kafka.connect.datagen.DatagenConnector
```

```
  taskMax: 1
  connectClusterRef:
    name: connect
  configs:
    kafka.topic: "gitops-pageviews"
    quickstart: "pageviews"
    key.converter: "org.apache.kafka.connect.storage.StringConverter"
    value.converter: "io.confluent.connect.avro.AvroConverter"
    value.converter.schema.registry.url: "http://schemaregistry.confluent.svc.cluster.local:808
    max.interval: "1000"
    iterations: "10000000"
```

Create `deploy/base/kustomization.yaml`:

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
  - connector.yaml

commonLabels:
  app.kubernetes.io/name: datagen-gitops-demo
  app.kubernetes.io/managed-by: argocd
```

Create `deploy/argocd-application.yaml`:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: netty-connector
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  project: default
  source:
    repoURL: https://github.com/confluentfederal/kafka-connect-netty-source-argo-example.git
    targetRevision: main
    path: deploy/base
  destination:
    server: https://kubernetes.default.svc
    namespace: confluent
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

6.3 Commit and Push

```
git add deploy/
git commit -m "Add Kubernetes deploy manifests for ArgoCD"
git push origin main
```

6.4 Deploy to ArgoCD

```
kubectl apply -f deploy/argocd-application.yaml
```

6.5 Register Repository for Branch Discovery

To enable branch dropdown in ArgoCD UI:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: repo-netty-connector
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://github.com/confluentfederal/kafka-connect-netty-source-argo-example.git
EOF
```

6.6 Create Topic

```
kubectl exec -n confluent kafka-0 -- kafka-topics \
  --bootstrap-server localhost:9092 \
  --create --topic gitops-pageviews \
  --partitions 3 --replication-factor 1
```

6.7 Verify Connector

```
kubectl get connector datagen-gitops-demo -n confluent
```

Part 7: Branch-Based Configuration Management

7.1 Create Feature Branches

Users Data Branch

```
git checkout -b feature/users-data

# Update deploy/base/connector.yaml
cat > deploy/base/connector.yaml <<EOF
apiVersion: platform.confluent.io/v1beta1
kind: Connector
```

```
metadata:
  name: datagen-gitops-demo
  namespace: confluent
spec:
  class: io.confluent.kafka.connect.datagen.DatagenConnector
  taskMax: 2
  connectClusterRef:
    name: connect
  configs:
    kafka.topic: "gitops-users"
    quickstart: "users"
    key.converter: "org.apache.kafka.connect.storage.StringConverter"
    value.converter: "io.confluent.connect.avro.AvroConverter"
    value.converter.schema.registry.url: "http://schemaregistry.confluent.svc.cluster.local:808
    max.interval: "2000"
    iterations: "10000000"
EOF

git add .
git commit -m "Generate users data instead of pageviews"
git push -u origin feature/users-data
```

High-Throughput Orders Branch

```
git checkout main
git checkout -b feature/orders-high-throughput

# Update deploy/base/connector.yaml
cat > deploy/base/connector.yaml <<EOF
apiVersion: platform.confluent.io/v1beta1
kind: Connector
metadata:
  name: datagen-gitops-demo
  namespace: confluent
spec:
  class: io.confluent.kafka.connect.datagen.DatagenConnector
  taskMax: 4
  connectClusterRef:
    name: connect
  configs:
    kafka.topic: "gitops-orders"
    quickstart: "orders"
    key.converter: "org.apache.kafka.connect.storage.StringConverter"
    value.converter: "io.confluent.connect.avro.AvroConverter"
    value.converter.schema.registry.url: "http://schemaregistry.confluent.svc.cluster.local:808
    max.interval: "100"
    iterations: "10000000"
EOF
```

```
git add .
git commit -m "Generate high-volume orders data"
git push -u origin feature/orders-high-throughput
git checkout main
```

7.2 Branch Configuration Summary

Branch	Topic	Data Type	Tasks	Rate
`main`	gitops-pageviews	Pageviews	1	1s
`feature/users-data`	gitops-users	Users	2	2s
`feature/orders-high-throughput`	gitops-orders	Orders	4	100ms

7.3 Switch Branches via ArgoCD UI

1. Open ArgoCD UI
2. Click on `netty-connector` application
3. Click **APP DETAILS** then **EDIT**
4. Change **TARGET REVISION** dropdown to desired branch
5. Click **SAVE**
6. ArgoCD auto-syncs the new configuration

7.4 Switch Branches via Command Line

```
# Switch to users data
kubectl patch application netty-connector -n argocd \
  --type merge -p '{"spec":{"source":{"targetRevision":"feature/users-data"}}}'

# Switch to high-throughput orders
kubectl patch application netty-connector -n argocd \
  --type merge -p '{"spec":{"source":{"targetRevision":"feature/orders-high-throughput"}}}'

# Switch back to main
kubectl patch application netty-connector -n argocd \
  --type merge -p '{"spec":{"source":{"targetRevision":"main"}}}'
```

7.5 Verify Branch Switch

```
# Check current branch
kubectl get application netty-connector -n argocd \
  -o jsonpath='{.spec.source.targetRevision}'

# Check connector configuration
kubectl get connector datagen-gitops-demo -n confluent \
  -o jsonpath='Topic: {.spec.configs.kafka\.topic}, Tasks: {.spec.taskMax}'
```

Operational Commands Reference

ArgoCD Commands

Command	Description
`kubectl get application -n argocd`	List all applications
`kubectl get application <name> -n argocd -o yaml`	View application details
`kubectl patch application <name> -n argocd --type merge -p '{...}'`	Update application
`kubectl annotate application <name> -n argocd argocd.argoproj.io/refresh=hard`	Force refresh
`kubectl patch application <name> -n argocd --type merge -p '{"operation": null}'`	Clear stuck operation

Confluent Platform Commands

Command	Description
`kubectl get kafka,connect,schemaregistry -n confluent`	List CP components
`kubectl get connector -n confluent`	List connectors
`kubectl describe connector <name> -n confluent`	Connector details
`kubectl logs -l app=connect -n confluent`	Connect worker logs

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> --bootstrap-server localhost:9092`	Create topic

Command	Description
`kubectl exec kafka-0 -n confluent -- kafka-console-consumer --topic <name> --bootstrap-server localhost:9092 --from-beginning`	Consume messages

Troubleshooting Guide

ArgoCD Not Syncing

```
# Force hard refresh
kubectl annotate application confluent-platform -n argocd \
  argocd.argoproj.io/refresh=hard --overwrite

# Clear stuck operation
kubectl patch application confluent-platform -n argocd \
  --type merge -p '{"operation": null}'

# Check sync status
kubectl get application confluent-platform -n argocd \
  -o jsonpath='{.status.sync.status} {.status.health.status}'
```

Connector in ERROR State

```
# Get error message
kubectl get connector <name> -n confluent \
  -o jsonpath='{.status.conditions[*].message}'

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

Pods Not Starting

```
# Describe pod for events
kubectl describe pod <pod-name> -n confluent

# Check PVC binding
kubectl get pvc -n confluent

# Check resource limits
kubectl describe nodes | grep -A 5 "Allocated resources"
```

Topic Not Created

```
# Manually create topic
kubectl exec kafka-0 -n confluent -- kafka-topics \
```

```
  --bootstrap-server localhost:9092 \
  --create --topic <topic-name> \
  --partitions 3 --replication-factor 1
```

Cleanup

Delete Connector Demo

```
kubectl delete application netty-connector -n argocd
kubectl delete secret repo-netty-connector -n argocd
```

Delete Confluent Platform

```
kubectl delete application confluent-platform -n argocd
```

Delete ArgoCD

```
kubectl delete namespace argocd
```

Delete EKS Cluster

```
eksctl delete cluster --name cfk-argocd-demo --region us-east-1
```

Summary

This implementation demonstrates a complete GitOps workflow for Confluent Platform:

1. **Infrastructure** - EKS cluster created via eksctl
2. **Operators** - CFK and ArgoCD installed via Helm/kubectl
3. **Platform** - Confluent Platform deployed via ArgoCD from Git
4. **Connectors** - Declarative connector management with branch-based configs
5. **Operations** - Self-healing, audit trail, rollback via Git

Key Benefits

- **Single Source of Truth** - All configuration in Git
- **Audit Trail** - Git history shows all changes
- **Rollback** - Revert to any previous state via Git
- **Self-Healing** - ArgoCD reverts manual changes
- **Multi-Environment** - Branches for dev/staging/prod

Resources

- **Main Repository:** https://github.com/confluentfederal/cfk-argocd-demo
- **Connector Demo:** https://github.com/confluentfederal/kafka-connect-netty-source-argo-example
- **CFK Documentation:** https://docs.confluent.io/operator/current/overview.html

- **ArgoCD Documentation:** https://argo-cd.readthedocs.io/

Document generated: January 28, 2026