Configuring Flux with KinD and Azure Container Registry (OCI)

This guide walks you through setting up Flux in a KinD (Kubernetes in Docker) cluster to sync OCI artifacts from Azure Container Registry (ACR).

Prerequisites

Before starting, ensure you have the following tools installed:

- Docker
- KinD
- kubectl
- Flux CLI
- Azure CLI

Step 1: Create KinD Cluster

Create a KinD cluster with a custom configuration to enable proper networking:

```
# Create kind-config.yaml
cat <<EOF > kind-config.yaml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
name: flux-cluster
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
    protocol: TCP
  - containerPort: 443
    hostPort: 443
    protocol: TCP
EOF

# Create the cluster
kind create cluster --config=kind-config.yaml
```

Verify the cluster is running:

```
kubectl cluster-info --context kind-flux-cluster
kubectl get nodes
```

Step 2: Set Up Azure Container Registry

Create Azure Container Registry

```
# Login to Azure
az login

# Set variables
RESOURCE_GROUP="flux-demo-rg"
ACR_NAME="fluxdemoacr$(date +%s)"  # Unique name
LOCATION="eastus"

# Create resource group
az group create --name $RESOURCE_GROUP --location $LOCATION

# Create ACR
az acr create --resource-group $RESOURCE_GROUP --name $ACR_NAME --sku
Basic

# Get ACR login server
ACR_LOGIN_SERVER=$(az acr show --name $ACR_NAME --resource-group
$RESOURCE_GROUP --query "loginServer" --output tsv)
echo "ACR Login Server: $ACR_LOGIN_SERVER"
```

Create Service Principal for Flux Authentication

```
# Create service principal
SP_NAME="flux-acr-sp"
ACR_REGISTRY_ID=$(az acr show --name $ACR_NAME --resource-group
$RESOURCE_GROUP --query "id" --output tsv)

SP_PASSWD=$(az ad sp create-for-rbac --name $SP_NAME --scopes
$ACR_REGISTRY_ID --role acrpull --query "password" --output tsv)
SP_APP_ID=$(az ad sp list --display-name $SP_NAME --query "[].appId" --
output tsv)

echo "Service Principal App ID: $SP_APP_ID"
echo "Service Principal Password: $SP_PASSWD"
```

Step 3: Install Flux

Bootstrap Flux in the KinD Cluster

```
# Install Flux components
flux install

# Verify Flux installation
kubectl get pods -n flux-system
```

Wait for all Flux pods to be running before proceeding.

Step 4: Create Kubernetes Secret for ACR Authentication

```
# Create namespace for your application (if needed)
kubectl create namespace demo

# Create Docker registry secret for ACR
kubectl create secret docker-registry acr-secret \
  --namespace=flux-system \
  --docker-server=$ACR_LOGIN_SERVER \
  --docker-username=$SP_APP_ID \
  --docker-password=$SP_PASSWD
```

Step 5: Push OCI Artifacts to ACR

Example: Push a Kustomization to ACR as OCI Artifact

First, create a sample Kustomization:

```
# Create a sample application directory
mkdir -p sample-app
cd sample-app

# Create a simple deployment
cat <<EOF > deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: demo
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
```

```
      - name: nginx
        image: nginx:1.21
        ports:
        - containerPort: 80
EOF

# Create kustomization.yaml
cat <<EOF > kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- deployment.yaml
EOF

cd ..
```

Push to ACR using Flux CLI

```
# Login to ACR
az acr login --name $ACR_NAME

# Push the kustomization as OCI artifact
flux push artifact oci://$ACR_LOGIN_SERVER/flux/sample-app:v1.0.0 \
  --path="./sample-app" \
  --source="$(git config --get remote.origin.url)" \
  --revision="$(git rev-parse HEAD)"
```

Step 6: Configure Flux OCIRepository

Create an OCIRepository resource to connect Flux to your ACR:

```
# Create oci-repository.yaml
cat <<EOF > oci-repository.yaml
apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: OCIRepository
metadata:
  name: sample-app-oci
  namespace: flux-system
spec:
  interval: 5m
  url: oci://$ACR_LOGIN_SERVER/flux/sample-app
  ref:
    tag: v1.0.0
  secretRef:
    name: acr-secret
EOF
```

Apply the OCIRepository:

```
kubectl apply -f oci-repository.yaml
```

Step 7: Create Kustomization for Deployment

Create a Kustomization resource that references the OCI artifact:

```
# Create kustomization-deploy.yaml
cat <<EOF > kustomization-deploy.yaml
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata:
  name: sample-app
  namespace: flux-system
spec:
  interval: 5m
  targetNamespace: demo
  sourceRef:
    kind: OCIRepository
    name: sample-app-oci
  path: "./"
  prune: true
  wait: true
  timeout: 2m
EOF
```

Apply the Kustomization:

```
kubectl apply -f kustomization-deploy.yaml
```

Step 8: Verify Deployment

Check that Flux has successfully deployed your application:

```
# Check Flux sources
flux get sources oci

# Check Kustomizations
flux get kustomizations

# Check if the demo namespace was created
kubectl get namespaces

# Check if the nginx deployment was created
kubectl get deployments -n demo
```

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

Step 9: Monitor and Troubleshoot

View Flux Events and Logs

```
# Get events from Flux
flux events

# Check OCIRepository status
kubectl describe ocirepository sample-app-oci -n flux-system

# Check Kustomization status
kubectl describe kustomization sample-app -n flux-system

# View controller logs
kubectl logs -n flux-system deployment/source-controller
kubectl logs -n flux-system deployment/kustomize-controller
```

Common Issues and Solutions

1. **Authentication Issues**

```
# Verify secret exists
kubectl get secret acr-secret -n flux-system

# Check secret content
kubectl get secret acr-secret -n flux-system -o yaml
```

2. **OCI Repository Not Found**

```
# List artifacts in ACR
az acr repository list --name $ACR_NAME

# Check specific artifact
az acr repository show-tags --name $ACR_NAME --repository flux/sample-
app
```

3. **Network Issues in KinD**

```
# Test DNS resolution from within cluster
kubectl run test-pod --image=busybox -it --rm -- nslookup
$ACR_LOGIN_SERVER
```

Step 10: Updating OCI Artifacts

To update your application, push a new version and update the OCIRepository:

```
# Make changes to your application
# Push new version
flux push artifact oci://$ACR_LOGIN_SERVER/flux/sample-app:v1.1.0 \
  --path="./sample-app" \
  --source="$(git config --get remote.origin.url)" \
  --revision="$(git rev-parse HEAD)"

# Update OCIRepository to use new tag
kubectl patch ocirepository sample-app-oci -n flux-system \
  --type='merge' -p='{"spec":{"ref":{"tag":"v1.1.0"}}}'
```

Cleanup

To clean up the resources:

```
# Delete the KinD cluster
kind delete cluster --name flux-cluster

# Delete Azure resources
az group delete --name $RESOURCE_GROUP --yes --no-wait

# Delete service principal
az ad sp delete --id $SP_APP_ID
```

Additional Resources

- Flux OCI Documentation
- Azure Container Registry Documentation
- KinD Documentation
- Flux GitHub Repository

Notes

- Replace placeholder values (like `$ACR_NAME`, `$RESOURCE_GROUP`) with your actual values
- Ensure your Azure subscription has sufficient permissions to create ACR and service principals
- The service principal needs `acrpull` permissions to access the registry
- OCI artifacts in ACR are billed as storage, so monitor usage in production environments