HLF - Operator

Another way to launch a hyperledger fabric network in the blockchain is by using the hlf-operator. It is a Kubernetes plugin that provides a declarative way of creating hyperledger fabric components. The operator has a wide variety of features that helps in the end-to-end deployment and management of hyperledger fabric network components. The hlf-operator has particular, abstract, and imperative commands, saves a lot of initial bootstrapping, and makes the fabric component deployment task easier.

HLF Operator is a Kubernetes Operator built with the operator sdk to manage the Hyperledger Fabric components:

- Peer
- Ordering service nodes(OSN)
- Certificate authorities

Features

- Create certificates authorities (CA)
- Create peers
- Create ordering services
- Create resources without manual provisioning of cryptographic material
- Domain routing with SNI using Istio
- Run chaincode as external chaincode in Kubernetes
- Support Hyperledger Fabric 2.3+
- Managed genesis for Ordering services
- E2E testing including the execution of chaincodes in KIND
- Renewal of certificates

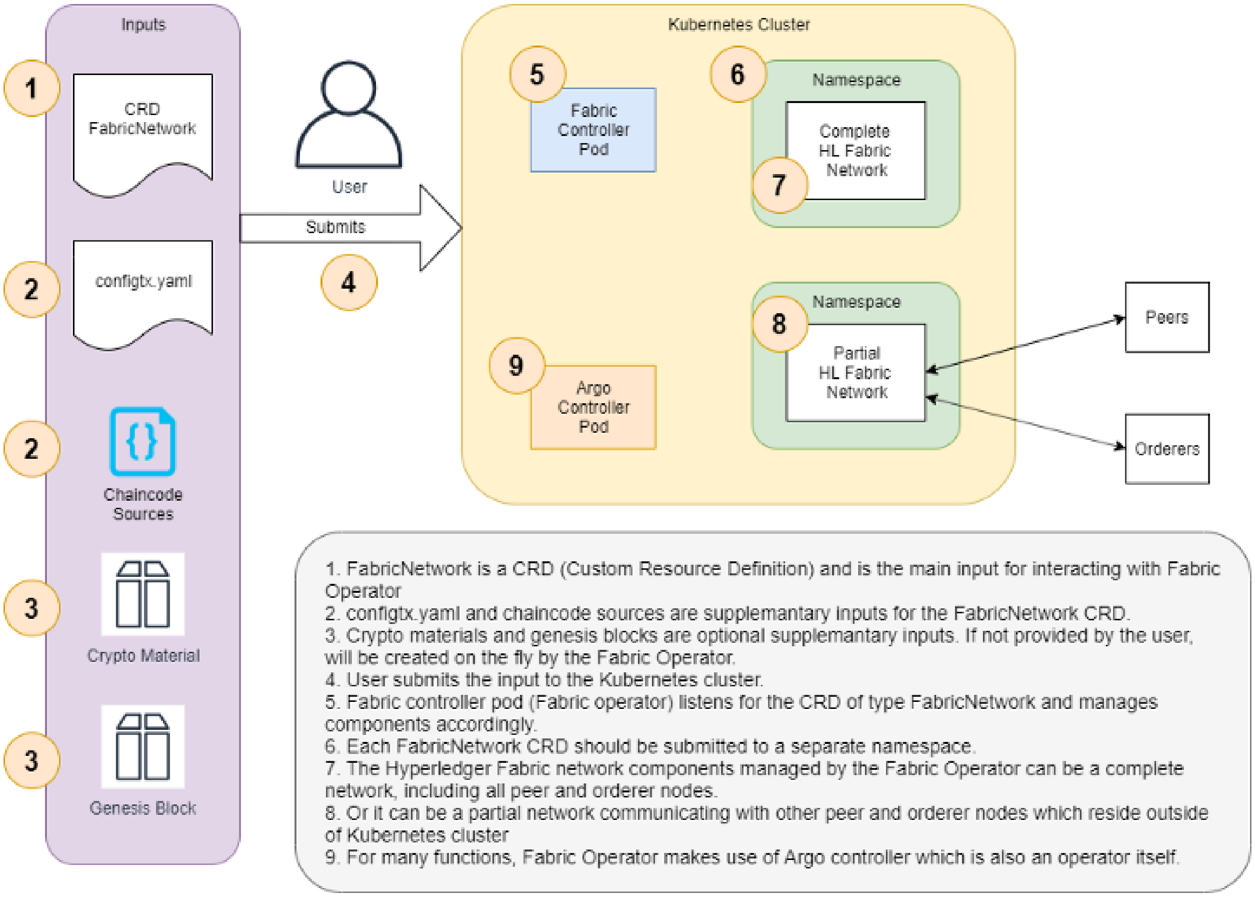
Documentation : https://hyperledger.github.io/bevel-operator-fabric/docs/getting-started

Github repo : https://github.com/hyperledger/bevel-operator-fabric

Adithya Joshi repo for operator :
https://github.com/adityajoshi12/hyperledger-fabric-on-kubernetes

Youtube tutorial -
https://www.youtube.com/playlist?list=PLuAZTZDgj0csRQuNMY8wbYqOCpzggAuMo

Diagram for HL Fabric Operator



1. FabricNetwork is a CRD (Custom Resource Definition) and is the main input for interacting with Fabric Operator
2. configtx.yaml and chaincode sources are supplemantary inputs for the FabricNetwork CRD.
3. Crypto materials and genesis blocks are optional supplemantary inputs. If not provided by the user, will be created on the fly by the Fabric Operator.
4. User submits the input to the Kubernetes cluster.
5. Fabric controller pod (Fabric operator) listens for the CRD of type FabricNetwork and manages components accordingly.
6. Each FabricNetwork CRD should be submitted to a separate namespace.
7. The Hyperledger Fabric network components managed by the Fabric Operator can be a complete network, including all peer and orderer nodes.
8. Or it can be a partial network communicating with other peer and orderer nodes which reside outside of Kubernetes cluster
9. For many functions, Fabric Operator makes use of Argo controller which is also an operator itself.

Helm

Helm is a package manager for Kubernetes, which helps to simplify the installation, configuration, and deployment of applications and services on a Kubernetes cluster. It provides a templating engine that allows users to define the desired state of their application or service as a set of parameters, and then generate Kubernetes manifests to deploy those resources.

With Helm, users can create reusable packages called "charts" that can be shared and versioned, making it easy to distribute and manage applications on Kubernetes. Charts are essentially a collection of YAML files that define Kubernetes resources like deployments, services, configmaps, and more, as well as optional values that can be customized during installation.

Helm also provides a robust ecosystem of tools and plugins that extend its functionality, including tools for managing chart dependencies, managing releases, and even creating charts from existing Kubernetes resources. Overall, Helm is a powerful tool that simplifies the process of deploying applications and services on Kubernetes.

Krew

Krew is a package manager for Kubernetes that helps you discover and install kubectl plugins. It is a command-line tool that simplifies the installation and management of kubectl plugins, making it easy to find, install, and update them. With Krew, you can easily discover new tools and plugins to extend your kubectl experience and improve your productivity. It is similar to other package managers such as apt, yum, and Homebrew, but is specifically designed for Kubernetes. Krew is an open-source project and can be used on any operating system that supports kubectl.

DEMO

Note : When encountering errors first check errors and solution section below

Create kubernetes cluster
Create cloud kubernetes cluster with 3 nodes with 2 cpus and 8gb memory.

Note : Might not need this much memory and nodes, try with less but manage storage allocation in the below commands accordingly.

Installations

Install helm

```
sudo snap install helm --classic
```

Add helm repo for hlf-operator in cluster

```
helm repo add kfs https://kfsoftware.github.io/hlf-helm-charts --force-update
```

Install hlf-operator sdk in cluster

```
helm install hlf-operator --version=1.6.0 kfs/hlf-operator
```

Check if operator pod created

```
kubectl get pods
```

Installing krew

```
(
  set -x; cd "$(mktemp -d)" &&
  OS="$(uname | tr '[:upper:]' '[:lower:]')" &&
  ARCH="$(uname -m | sed -e 's/x86_64/amd64/' -e 's/\(arm\)\(64\)\?.*/\1\2/' -e 's/aarch64$/arm64/')" &&
  KREW="krew-${OS}_${ARCH}" &&
  curl -fsSLO "https://github.com/kubernetes-sigs/krew/releases/latest/download/${KREW}.tar.gz" &&
  tar zxvf "${KREW}.tar.gz" &&
  ./"${KREW}" install krew
)
export PATH="${KREW_ROOT:-$HOME/.krew}/bin:$PATH"

//open new terminal
kubectl krew update
```

Install hlf plugin

```
kubectl krew install hlf
```

See storage class

```
kubectl get sc
```

Export storage class

```
export SC=$(kubectl get sc -o=jsonpath='{.items[0].metadata.name}')
echo $SC
```

Create namespace

```
kubectl create ns fabric
kubectl get ns
```

Create ca

for org1

```
kubectl hlf ca create --storage-class=$SC --capacity=2Gi --name=org1-ca --enroll-id=enroll
--enroll-pw=enrollpw --namespace=fabric
```

for org2

```
kubectl hlf ca create --storage-class=$SC --capacity=2Gi --name=org2-ca --enroll-id=enroll
--enroll-pw=enrollpw --namespace=fabric
```

for orderer

```
kubectl hlf ca create --storage-class=$SC --capacity=2Gi --name=ord-ca --enroll-id=enroll
--enroll-pw=enrollpw --namespace=fabric
```

Check ca available in fabric ns

```
kubectl get pods -n fabric
```

See all pv for ca

```
kubectl get pvc -A  //-A is for all namespaces
```

ca setup complete

Export env variables

```
export PEER_IMAGE=hyperledger/fabric-peer
export PEER_VERSION=2.4.1
export ORDERER_IMAGE=hyperledger/fabric-orderer
export ORDERER_VERSION=2.4.1
```

Registering and generating identities for peer

for org1-peer1

```
kubectl hlf ca register --name=org1-ca --user=org1-peer1 --secret=peerpw --type=peer --enroll-id enroll
--enroll-secret=enrollpw --mspid=Org1MSP --namespace=fabric
```

for org1-peer2

```
kubectl hlf ca register --name=org1-ca --user=org1-peer2 --secret=peerpw --type=peer --enroll-id enroll
--enroll-secret=enrollpw --mspid=Org1MSP --namespace=fabric
```

for org2-peer1

```
kubectl hlf ca register --name=org2-ca --user=org2-peer1 --secret=peerpw --type=peer --enroll-id enroll
--enroll-secret=enrollpw --mspid=Org2MSP --namespace=fabric
```

for org2-peer2

```
kubectl hlf ca register --name=org2-ca --user=org2-peer2 --secret=peerpw --type=peer --enroll-id enroll
--enroll-secret=enrollpw --mspid=Org2MSP --namespace=fabric
```

Creating Peers

```
kubectl hlf peer create --storage-class=$SC --enroll-id=org1-peer1 --mspid=Org1MSP
--enroll-pw=peerpw --capacity=5Gi --name=org1-peer1 --ca-name=org1-ca.fabric --namespace=fabric
--statedb=couchdb --image=$PEER_IMAGE --version=$PEER_VERSION
```

```
kubectl hlf peer create --storage-class=$SC --enroll-id=org1-peer2 --mspid=Org1MSP
--enroll-pw=peerpw --capacity=5Gi --name=org1-peer2 --ca-name=org1-ca.fabric --namespace=fabric
--statedb=couchdb --image=$PEER_IMAGE --version=$PEER_VERSION
```

```
kubectl hlf peer create --storage-class=$SC --enroll-id=org2-peer1 --mspid=Org2MSP
--enroll-pw=peerpw --capacity=5Gi --name=org2-peer1 --ca-name=org2-ca.fabric --namespace=fabric
--statedb=couchdb --image=$PEER_IMAGE --version=$PEER_VERSION
```

```
kubectl hlf peer create --storage-class=$SC --enroll-id=org2-peer2 --mspid=Org2MSP
--enroll-pw=peerpw --capacity=5Gi --name=org2-peer2 --ca-name=org2-ca.fabric --namespace=fabric
--statedb=couchdb --image=$PEER_IMAGE --version=$PEER_VERSION
```

NOTE: If u want to get all peer details in an output file add --output > org1.peer1.yaml at end of command. Running with this won't create peer in cluster

Register and Enroll org admin (Admin Certs)

```
kubectl hlf ca register --name=org1-ca --user=admin --secret=adminpw --type=admin --enroll-id enroll
--enroll-secret=enrollpw --mspid=Org1MSP --namespace=fabric
```

```
kubectl hlf ca enroll --name=org1-ca --user=admin --secret=adminpw --ca-name ca  --output
org1-peer.yaml --mspid=Org1MSP --namespace=fabric
```

```
kubectl hlf ca register --name=org2-ca --user=admin --secret=adminpw --type=admin --enroll-id enroll
--enroll-secret=enrollpw --mspid=Org2MSP --namespace=fabric
```

```
kubectl hlf ca enroll --name=org2-ca --user=admin --secret=adminpw --ca-name ca  --output
org2-peer.yaml --mspid=Org2MSP --namespace=fabric
```

Orderer

Register orderer identity

```
kubectl hlf ca register --name=ord-ca --user=admin --secret=adminpw --type=admin --enroll-id enroll
--enroll-secret=enrollpw --mspid=OrdererMSP --namespace=fabric
```

Create Orderer node

```
kubectl hlf ordnode create  --storage-class=$SC --enroll-id=orderer --mspid=OrdererMSP
--enroll-pw=ordererpw --capacity=2Gi --name=ord-node1 --ca-name=ord-ca.fabric --namespace=fabric
--image=$ORDERER_IMAGE --version=$ORDERER_VERSION
```

Register orderer admin

```
kubectl hlf ca register --name=ord-ca --user=admin --secret=adminpw --type=admin --enroll-id enroll
--enroll-secret=enrollpw --mspid=OrdererMSP --namespace=fabric
```

Enroll orderer admin ca and tls certs

```
kubectl hlf ca enroll --name=ord-ca --user=admin --secret=adminpw --mspid=OrdererMSP --ca-name ca
--output admin-ordservice.yaml --namespace=fabric
```

```
kubectl hlf ca enroll --name=ord-ca --user=admin --secret=adminpw --mspid=OrdererMSP --ca-name
tlsca  --output admin-tls-ordservice.yaml --namespace=fabric
```

Connection Profile

```
kubectl hlf inspect --output networkConfig.yaml -o Org1MSP -o OrdererMSP -o Org2MSP
```

Add admin users to connection profile

```
kubectl hlf utils adduser --userPath=org1-peer.yaml --config=networkConfig.yaml --username=admin
--mspid=Org1MSP
```

```
kubectl hlf utils adduser --userPath=org2-peer.yaml --config=networkConfig.yaml --username=admin
--mspid=Org2MSP
```

Channel

```
kubectl hlf channel generate --output=mychannel.block --name=mychannel --organizations Org1MSP
--organizations Org2MSP --ordererOrganizations OrdererMSP
```

For orderer to join channel

```
kubectl hlf ordnode join --block=mychannel.block --name=ord-node1 --namespace=fabric
--identity=admin-tls-ordservice.yaml --namespace=fabric
```

For peers to join channel

```
kubectl hlf channel join --name=mychannel --config=networkConfig.yaml --user=admin
-p=org1-peer1.fabric
```

```
kubectl hlf channel join --name=mychannel --config=networkConfig.yaml --user=admin
-p=org1-peer2.fabric
```

```
kubectl hlf channel join --name=mychannel --config=networkConfig.yaml --user=admin
-p=org2-peer1.fabric
```

```
kubectl hlf channel join --name=mychannel --config=networkConfig.yaml --user=admin
-p=org2-peer2.fabric
```

Add Anchor Peers

```
kubectl hlf channel addanchorpeer --channel=mychannel --config=networkConfig.yaml --user=admin
--peer=org1-peer1.fabric
```

```
kubectl hlf channel addanchorpeer --channel=mychannel --config=networkConfig.yaml --user=admin
--peer=org2-peer1.fabric
```

Chaincode

```
CC_NAME=mycc
```

Create metadata.json

```
cat <<METADATA-EOF >"metadata.json"
   {
      "type": "ccaas",
      "label": "${CC_NAME}"
    }
METADATA-EOF
```

Create connection.json

```
cat <<CONN_EOF >"connection.json"
   {
   "address": "${CC_NAME}:7052",
   "dial_timeout": "10s",
   "tls_required": false
   }
CONN_EOF
```

```
tar cfz code.tar.gz connection.json

tar cfz ${CC_NAME}-external.tgz metadata.json code.tar.gz
```

```
PACKAGE_ID=$(kubectl-hlf chaincode calculatepackageid --path=$CC_NAME-external.tgz
--language=node --label=$CC_NAME)

echo "PACKAGE_ID=$PACKAGE_ID"
```

Installing Chaincode

```
kubectl hlf chaincode install --path=./${CC_NAME}-external.tgz --config=networkConfig.yaml
--language=node --label=$CC_NAME --user=admin --peer=org1-peer1.fabric
```

```
kubectl hlf chaincode install --path=./${CC_NAME}-external.tgz --config=networkConfig.yaml
--language=node --label=$CC_NAME --user=admin --peer=org2-peer1.fabric
```

Deploying Chaincode

```
kubectl hlf externalchaincode sync --image=adityajoshi12/hlf-nodejs-external-cc:latest
--name=$CC_NAME --namespace=fabric --package-id=$PACKAGE_ID --tls-required=false --replicas=1
```

Approve Chaincode

```
kubectl hlf chaincode approveformyorg --config=networkConfig.yaml --user=admin
--peer=org1-peer1.fabric --package-id=$PACKAGE_ID --version 1.0 --sequence 1 --name=$CC_NAME
--policy="OR('Org1MSP.member','Org2MSP.member')" --channel=mychannel
```

```
kubectl hlf chaincode approveformyorg --config=networkConfig.yaml --user=admin
--peer=org2-peer1.fabric --package-id=$PACKAGE_ID --version 1.0 --sequence 1 --name=$CC_NAME
--policy="OR('Org1MSP.member','Org2MSP.member')" --channel=mychannel
```

Commit Chaincode

```
kubectl hlf chaincode commit --config=networkConfig.yaml --mspid=Org1MSP --user=admin --version
1.0 --sequence 1 --name=$CC_NAME --policy="OR('Org1MSP.member','Org2MSP.member')"
--channel=mychannel
```

Invoke/Query

```
kubectl hlf chaincode invoke --config=networkConfig.yaml --user=admin --peer=org1-peer1.fabric
--chaincode=$CC_NAME --channel=mychannel --fcn=createCar -a "car1" -a "ford" -a "mustang" -a
"black" -a "abhi"
```

```
kubectl hlf chaincode query --config=networkConfig.yaml --user=admin --peer=org1-peer1.fabric
--chaincode=$CC_NAME --channel=mychannel --fcn=queryAllCars -a ''
```

```
kubectl hlf chaincode query --config=networkConfig.yaml --user=admin --peer=org1-peer1.fabric
--chaincode=$CC_NAME --channel=mychannel --fcn=queryCar -a 'car1'
```

Other Commands

For seeing ledger height
ie, peers with the no of blocks they have

```
kubectl hlf channel top --channel=mychannel --config=networkConfig.yaml --user=admin
-p=org1-peer1.fabric
```

Get all channel details in a .json file

```
kubectl hlf channel inspect --channel=mychannel --config=networkConfig.yaml --user=admin
-p=org1-peer1.fabric > mychannel.json
```

Open Couchdb
Go to lens IDE
Check details of org1 peer1 pod couchdb
There the ports will be mentioned ,
username - couchdb
Password - couchdb

Possible Errors

Error: unknown command "hlf" for "kubectl"
And if sometimes krew is not working in new terminal

Solution: in terminal -
```
export PATH="${KREW_ROOT:-$HOME/.krew}/bin:$PATH"
```

Error -
```
Error: enroll failed: enroll failed: POST failure of request: POST
https://34.93.5.241:30745/enroll {"hosts":null,"certificate_request":"-----BEGIN CERTIFICATE
REQUEST-----\nMIHvMIGWAgEAMBExDzANBgNVBAMTBmVucm9sbDBZMBMGByqGSM49AgE
GCCqGSM49\nAwEHA0IABG4US2LmSuZa7aX4f3tudWBZEN27xPYhzuth2Mw5rpbAISD6vZ9LG3g
c\nZKDC4/1Aom7t6nT00AVwdTd/uxTQuXegIzAhBgkqhkiG9w0BCQ4xFDASMBAGA1Ud\nEQQJM
AeCBWh5ZHJhMAoGCCqGSM49BAMCA0gAMEUCIQCkEzzXJwYktQ8qGYX0Kadp\noON+c98zV
CcwtEAnl3DcSAIgeGblreYpBRsbxZHbFeJStDMrH3uE32bf6JVdY7/5\nCKI=\n-----END
CERTIFICATE
REQUEST-----\n","profile":"","crl_override":"","label":"","NotBefore":"0001-01-01T00:00:00Z","NotAft
er":"0001-01-01T00:00:00Z","ReturnPrecert":false,"CAName":""}: Post
"https://34.93.5.241:30745/enroll": dial tcp 34.93.5.241:30745: connect: connection timed out
```

Solution : the error is due to a firewall in gcp . go to cloud dashboard -> compute engine -> any of the VMs choose “more actions” button -> view network details -> firewall -> create firewall rule -> add the port there
Refer this video - https://youtu.be/-RjDWwTZUnc

Note: this error can occur while using aws, gcp, azure etc and also while creating orderer

Error : 'InstallChaincode': could not build chaincode: docker build failed: docker build is disabled

Solution : This is caused when using a fabric version lower than 2.4.1 , change peer , orderer version to 2.4.1 or above

Error: ChaincodeID and Fcn are required

Solution : in terminal **-**

```
CC_NAME=mycc

PACKAGE_ID=$(kubectl-hlf chaincode calculatepackageid --path=$CC_NAME-external.tgz
--language=node --label=$CC_NAME)
```

Other Errors : export all the environment files you exported earlier and try command again. This may solve the error.

```
export SC=$(kubectl get sc -o=jsonpath='{.items[0].metadata.name}')
export PEER_IMAGE=hyperledger/fabric-peer
export PEER_VERSION=2.4.1
export ORDERER_IMAGE=hyperledger/fabric-orderer
export ORDERER_VERSION=2.4.1
```