# Introduction

This git repository contains all the artifacts required to deploy Hyperledger Fabric setup on Kubernetes Single Node Cluster

# K8S Environment Details

This setup was tested on a single node kubernetes cluster i.e. a master node + 1 worker node. Kubernetes was installed on ubuntu 16.04 using kubeadm and kubectl.

K8S version: 1.15.3

Refer [this link](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#installing-kubeadm-kubelet-and-kubectl) for k8s installation.

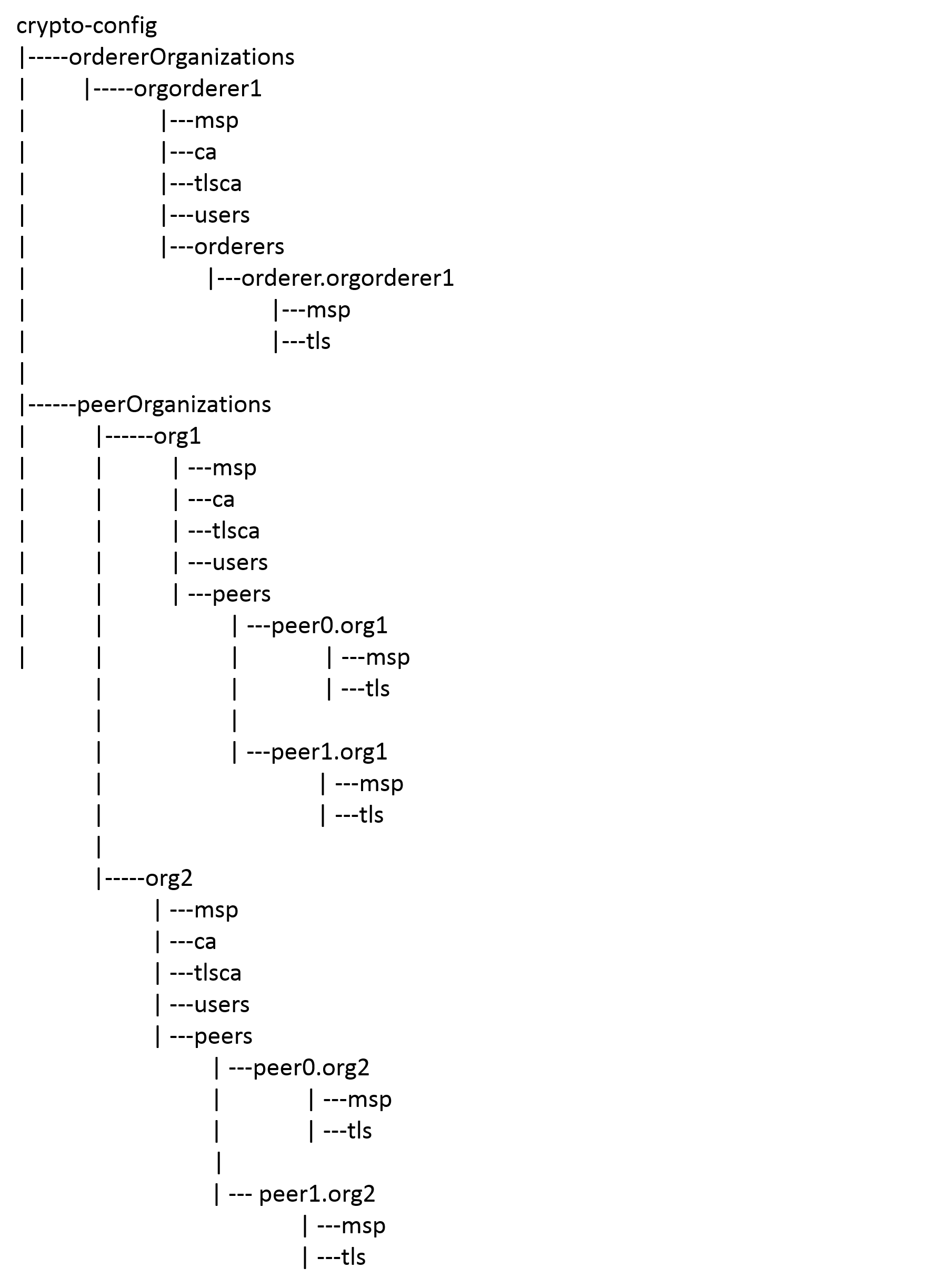
# Hyperledger Fabric Details

Hyperledger fabric version: 1.4

Following is the Org Structure:

* 2 orgs with 2 peers each and 1 cli container per org. Peers use couchdb as state db.
  + Org1:
    - peer0
    - peer1
  + Org2:
    - peer0
    - peer1
* 1 solo orderer with org: orgorderer1
* 1 cli container per org

The configuration of the network looks like this



# Hyperledger Fabric on K8S

A picture containing outdoor, black

Description automatically generated

The HF setup on K8S single node cluster looks as shown above.

As there is only 1 worker node, all pods would reside on that same node.

In Kubernetes, namespace is an important concept. It is used to divide cluster resources between multiple users. In the case of Fabric, organizations can be mapped into namespaces so that they have their dedicated resource. After this mapping, peers of each organization can be distinguished by domain name.

For our setup there would be 2 namespaces org1 and org2 for peer orgs and 1 namespace for orderer org called orgorderer1.

As shown in above diagram, org1 and org2 will have following PODs:

1. Peer POD: Every peer would have a separate pod. As there are 2 peers for an org, there would be 2 pods for peer0 and peer1 respectively. Every peer pod would have a peer container and a couchdb(statedb) container.
2. CLI POD: Provides an environment for command-line tools to manipulate the nodes of the organization. Fabric’s peer environment variables are configured in this pod.

As the setup contains 1 solo orderer, it would have a separate namespace named orgorderer1 which would have 1 POD containing a solo orderer container

## Volume hostPath

Fabric containers in the PODs need to refer to configuration data and keys/certificates data from file system. As this is a single node cluster, this data is accessed from the k8s volume type called “hostPath” which points to the directory/file on worker node.

The referred configuration files and keys/certificates are copied to the specific directory on worker node and are mapped to the container directory in volumeMounts stanza of deployment yaml.

## Communication between Fabric components

When fabric components are placed into k8s pods, we need to consider connectivity between pods.Each POD in k8s has IP address, but it is hard to use IP and port to communicate between pods as k8s pods are ephemeral to the POD. When POD gets restarted, its IP address gets changed too. Therefore it is necessary to create services in k8s for pods so that they can talk to each other through service name. The name of the service is: *<service name>.<namespace>.* The name of the service and namespace should follow below rules:

* The namespace of the service and the POD should be consistent
* The name of the service should be consistent with the id of the container within the pod.

E.g. Fabrics peer0 of org1 is mapped to pod named peer0 under namespace org1. The service binding to it should be named as peer0.org1 where peer0 is name of the service and org1 is the namespace of the service. Other pods can connect to peer0 of org1 by service name peer0.org1 which appears as peer0’s hostname.

## Workaround for Chaincode instantiate issue

When fabric peer instantiates the Chaincode it creates a docker container in which Chaincode runs. The docker API it invokes to create this Chaincode is unix::///var/run/docker.sock. This works well as long as the peer container and the Chaincode container are managed by same docker engine.However, in k8s Chaincode container is created by peer without notifying k8s. Hence the Chaincode and the peer container can not communicate with each other which results in failure when instantiating Chaincode.

To work around this problem, we need to add IP address of kube\_dns service in each worker node’s docker engine. This ensures the Chaincode container can resolve the peer’s hostname (service name) correctly by using kube\_dns service.

Follow below steps on worker node to achieve this:

1. systemctl edit docker.service
2. Add following lines

[Service]

ExecStart=

ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --*dns=10.244.0.7 --dns=10.244.0.8* ***--dns=10.244.0.1*** --dns-search \

default.svc.cluster.local --dns-search \

svc.cluster.local --dns-opt ndots:2 --dns-opt \

timeout:2 --dns-opt attempts:2

where 10.244.0.1 is flannel addOn’s subnet IP and 10.244.0.7 and 10.244.0.8 are IPs of core-dns PODs

1. Reload systemctl

*systemctl daemon-reload*

1. Restart docker

*systemctl restart docker*

To verify if changes have taken effect:

1. root@v-pt-mskhf01:/home/admin1# ps faux | grep dockerd

*root 25754 0.0 0.0 12952 936 pts/0 S+ 11:36 0:00 \\_ grep --color=auto dockerd*

*root 24410 5.3 1.1 809948 92336 ? Ssl 11:36 0:01 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --dns=10.244.0.7 --dns=10.244.0.8 --dns=10.244.0.1 --dns-search default.svc.cluster.local --dns-search svc.cluster.local --dns-opt ndots:2 --dns-opt timeout:2 --dns-opt attempts:2*

*root@v-pt-mskhf01:/home/admin1#*

1. root@v-pt-mskhf01:/home/admin1# systemctl status docker.service | grep dns

*ââ24410 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --dns=10.244.0.7 --dns=10.244.0.1 --dns-search default.svc.cluster.local --dns-search svc.cluster.local --dns-opt ndots:2 --dns-opt timeout:2 --dns-opt attempts:2*

# Setting up HF1.4 on K8S cluster

Perform following steps in exact same order:

1. On K8S **worker** node:
2. Login as a root/sudo root user
3. Make sure you have proxy less access to VM.
4. Start off in your home directory and clone following git repository:

git clone <https://github.com/medhak19/HF-On-K8S>

1. CD to SingleNodeSetup directory
2. Generate crypto-material, channel-artifacts and deployment yaml files by running following command:
   1. For **solo** orderer:

*chmod a+x generateAll.sh*

*./generateAll.sh*

* 1. For **etcdraft** orderer:

chmod a+x generateAll.sh

*./generateAll.sh* ***-o etcdraft***

1. Modify scripts/generate.py as follows: Modify value of WORKERNODE\_BASE\_DIR to the worker node base directory which contains crypto-config and channel-artifacts directories. E.g. If you have cloned git repo in directory /home/admin1/k8s, then WORKERNODE\_BASE\_DIR would be */home/admin1/k8s/HF-On-K8S/SingleNodeSetup*
2. Run python script to generate deployment yamls

Command: ***pyton3.5 scripts/generate.py***

1. On K8S **master** node:
2. Login to K8S master node as a root or sudo root user
3. Make sure you have proxy less access to VM.
4. Start off in your home directory and copy HF-On-K8S directory from worker node which was cloned from git repo in section A) above.
5. CD to HF-On-K8S/SingleNodeSetup directory
6. Run python script to build the K8S PODs for HF. All the pods should now be created.

Command: ***python3.5 scripts/run.py***

1. Run following command to verify this:

kubectl get pods --all-namespaces

# Verify HF setup using cli container

1. Login to K8S **master** as a root or sudo root
2. Get ID of cli pod of org1 by running command: kubectl get pods -n org1 | grep cli
3. Get Shell to the running containing of org1 cli pod: *kubectl exec <CLI POD> -c cli -n org1 -it bash*
4. Run command: export CHANNEL\_NAME=mychannel
5. Create Channel:

*peer channel create -o orderer.orgorderer1:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/mychannel.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/orgorderer1/orderers/orderer.orgorderer1/msp/tlscacerts/tlsca.orgorderer1-cert.pem*

1. peer0.org1 joins channel

*peer channel join -b mychannel.block*

1. Update anchor peer for org1

peer channel update -o orderer.orgorderer1:7050 -c mychannel -f ./channel-artifacts/Org1MSPanchors.tx

1. peer0.org2 joins the channel

*CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2/users/Admin@org2/msp CORE\_PEER\_ADDRESS=peer0.org2:7051 CORE\_PEER\_LOCALMSPID="Org2MSP" CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2/peers/peer0.org2/tls/ca.crt peer channel join -b mychannel.block*

1. Update anchor peer for org2

*CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2/users/Admin@org2/msp CORE\_PEER\_ADDRESS=peer0.org2:7051 CORE\_PEER\_LOCALMSPID="Org2MSP" CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2/peers/peer0.org2/tls/ca.crt* peer channel update -o orderer.orgorderer1:7050 -c mychannel -f ./channel-artifacts/Org2MSPanchors.tx *--tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/orgorderer1/orderers/orderer.orgorderer1/msp/tlscacerts/tlsca.orgorderer1-cert.pem*

1. Install the Chaincode on peer0.org1

*peer chaincode install -n mycc -v 1.0 -p github.com/chaincode/chaincode\_example02/go/*

1. Install chaincode on peer0.org2

*CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2/users/Admin@org2/msp CORE\_PEER\_ADDRESS=peer0.org2:7051 CORE\_PEER\_LOCALMSPID="Org2MSP" CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2/peers/peer0.org2/tls/ca.crt peer chaincode install -n mycc -v 1.0 -p github.com/chaincode/chaincode\_example02/go/*

1. Instantiate Chaincode on channel

*peer chaincode instantiate -o orderer.orgorderer1:7050 --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/orgorderer1/orderers/orderer.orgorderer1/msp/tlscacerts/tlsca.orgorderer1-cert.pem -C $CHANNEL\_NAME -n mycc -v 1.0 -c '{"Args":["init","a", "100", "b","200"]}'* *-P "AND ('Org1MSP.peer','Org2MSP.peer')"*

1. Query Chaincode

peer chaincode query -C $CHANNEL\_NAME -n mycc -c '{"Args":["query","a"]}'

1. Invoke Chaincode

peer chaincode invoke -o orderer.orgorderer1:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/orgorderer1/orderers/orderer.orgorderer1/msp/tlscacerts/tlsca.orgorderer1-cert.pem -C $CHANNEL\_NAME -n mycc --peerAddresses peer0.org1:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1/peers/peer0.org1/tls/ca.crt --peerAddresses peer0.org2:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2/peers/peer0.org2/tls/ca.crt -C $CHANNEL\_NAME -n mycc -c '{"Args":["invoke","a","b","20"]}'

1. Again query the chaincode and verify the value of a is now 90

peer chaincode query -C $CHANNEL\_NAME -n mycc -c '{"Args":["query","a"]}'

# Cleanup

To cleanup the setup,

1. Run following script on **k8s master** node to delete all the k8s pods, services and namespaces

***./cleanup.sh***

1. To remove the crypto material, channel artifacts, volume dir and deployment yamls, run following script on k8s **worker** node:

cd <BASE DIR>/HF-On-K8S/SingleNodeSetup

***./cleanup.sh rm\_dirs***