# Introduction

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

# K8S Environment Details

This setup was tested on a 2 node kubernetes cluster i.e. a master node + 2 worker nodes. 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.

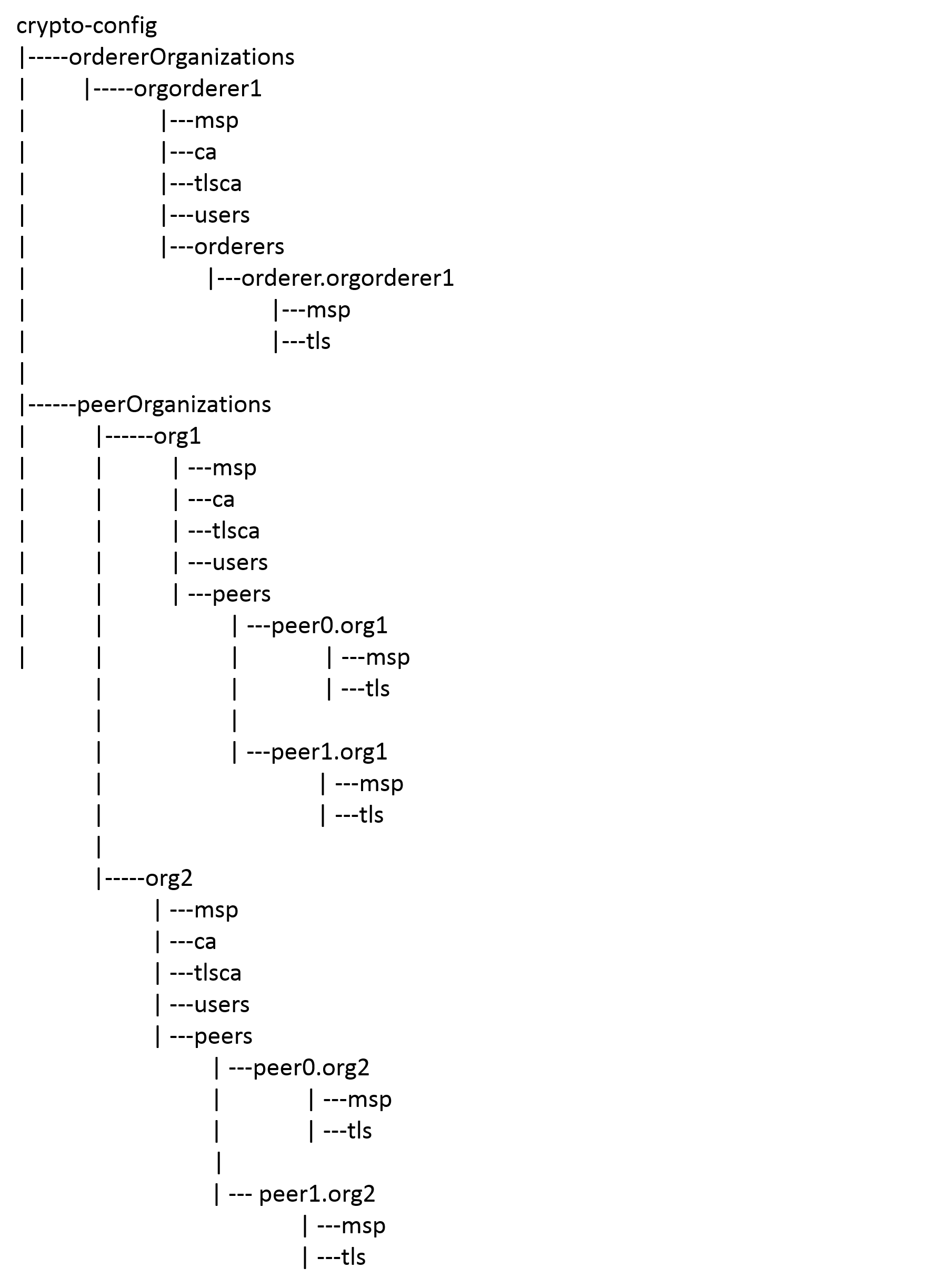
# Hyperledger Fabric Details

Hyperledger fabric version: 1.4

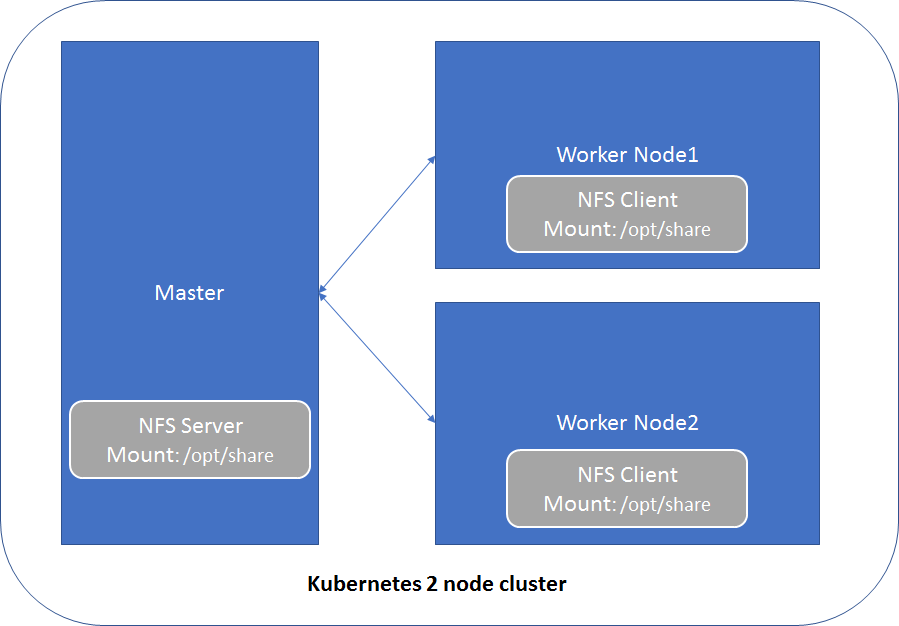
Following is the Org Structure:

* 2 orgs with 2 peers each and 1 cli container per org. Peers use couchdb as state db.
  + Org1:
    - peer0
    - peer1
  + Org2:
    - peer0
    - peer1
* Orderer Setup:
  + 1 solo orderer with org: orgorderer1 OR
  + RAFT ordering service with 5 orderer nodes
* 1 cli container per org

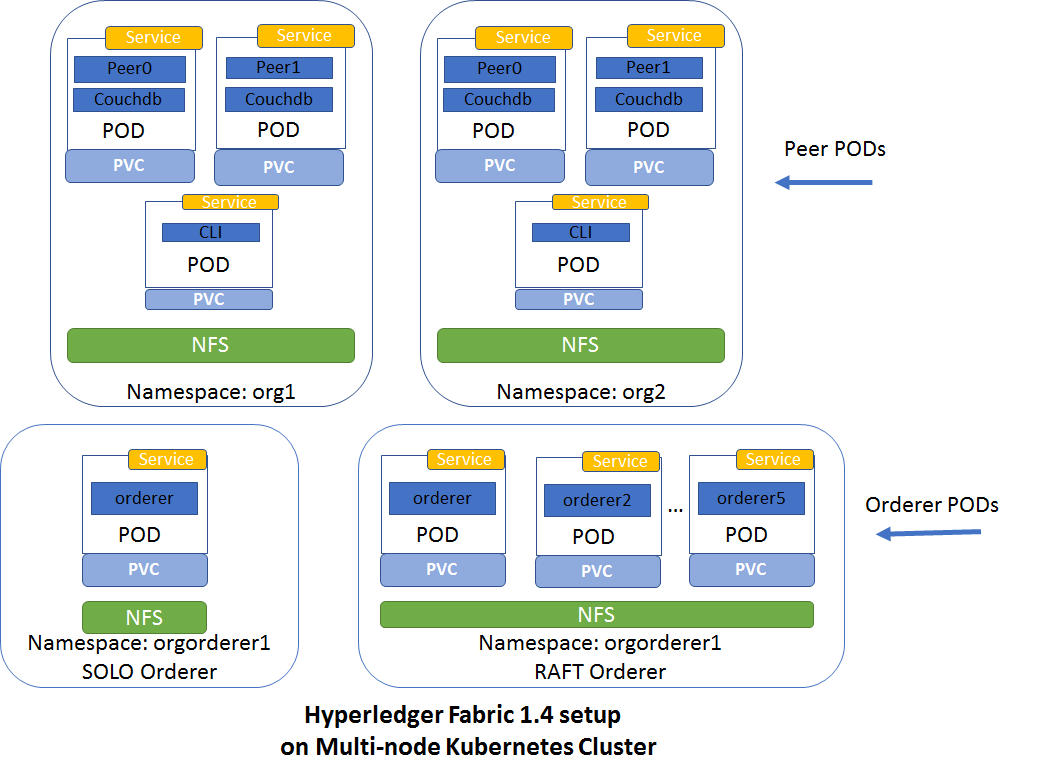
The configuration of the network looks like this(The diagram shows solo orderer config)



# Hyperledger Fabric on K8S



*Figure1: Kubernetes 2 node cluster*



*Figure2: HF Setup on K8S multi-node cluster*

Figure1 shows 2 node kubernetes cluster used for HF setup.

Figure2 shows how HF is setup on 2 node kubernetes cluster.

As there are 2 worker nodes in a k8s cluster, HF pods would be created on any of these 2 nodes by kubernetes.

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 in their respective namespace:

1. Peer POD: Every peer would have a separate pod. As there are 2 peers for an org, there would be 2 pods 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. There is 1 cli POD per org.

The setup supports both solo orderer and raft orderer.

For solo orderer, 1 POD would be created in namespace: orgorderer1

For RAFT orderer, 5 PODs would be created in namespace orgorderer1

## NFS Volume

Fabric containers in the PODs need to refer to configuration data and keys/certificates data from file system. This data is accessed from the k8s PersistentVolume mounted as NFS.

Follow below steps to setup NFS

1. On Master node, install NFS Server: apt-get install nfs-kernel-server
2. On both the worker nodes, install NFS client: apt-get install nfs-common
3. On master node,
   1. make a directory /opt/share and remove restrictive permission to that directory so that it can be viewed by anybody

mkdir -p /opt/share

chown nobody:nobody /opt/share

chmod 777 /opt/share

* 1. Add entry to nfs exports files so that clients can access it. Open /etc/exports and add following line:

*/opt/share \*(rw,sync,no\_subtree\_check,no\_root\_squash)*

* 1. Save the file. Now export the shares and restart the server:

exportfs -a

systemctl restart nfs-kernel-server

1. On both the worker nodes,
   1. Mount nfs using following command:

mount <IP address of master node i.e. nfs kernel server>:/opt/share /opt/share

* 1. To automatically mount it every time, add the following to /etc/fstab:

<IP address of master node i.e. nfs kernel server>:/opt/share /opt/share auto,nofail,noatime,nolock,intr,tcp,actimeo=1800 0 0

Save that file and you’re good. Try making a test file in the NFS folder and make sure you can see it on both systems.

## 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 **master** 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 **MultiNodeSetup** 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. 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

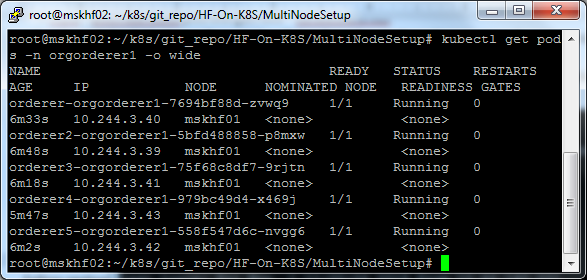
peer chaincode query -C $CHANNEL\_NAME -n mycc -c '{"Args":["query","a"]}'

# Test scenarios:

## Orderer mode: *etcdraft*

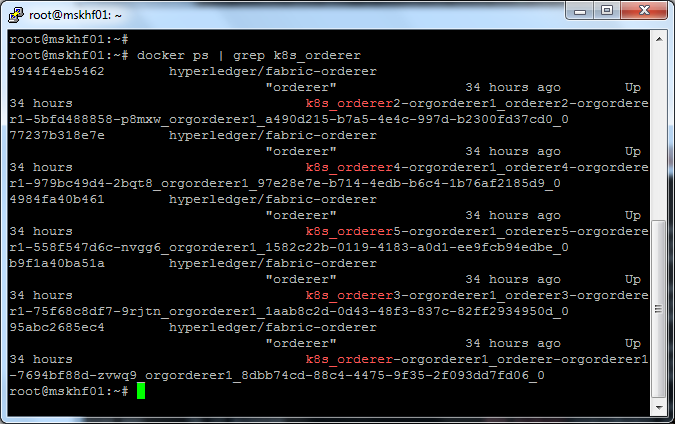
Follow section 5 & 6 for HF setup with etcdraft orderer. To verify the fault tolerance, follow below steps:

1. To identify the node on which orderer pods are created, perform following steps on K8S master node:
   1. Run command: kubectl get pods -n orgorderer1 -o wide

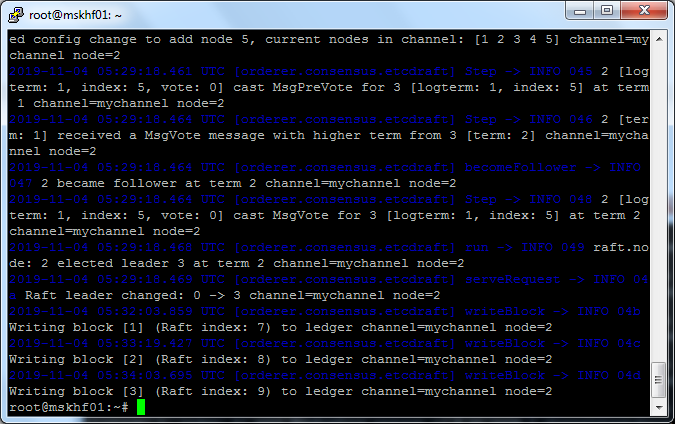


As seen in above screenshot, all orderer pods are created on node **mskhf01.**

1. Now perform following steps on mskhf01 node which has orderer pods and containers.
   1. To get docker container names for orderer, run command: docker ps | grep k8s\_orderer



* 1. Identify the RAFT leader by checking logs of any of the orderer container.

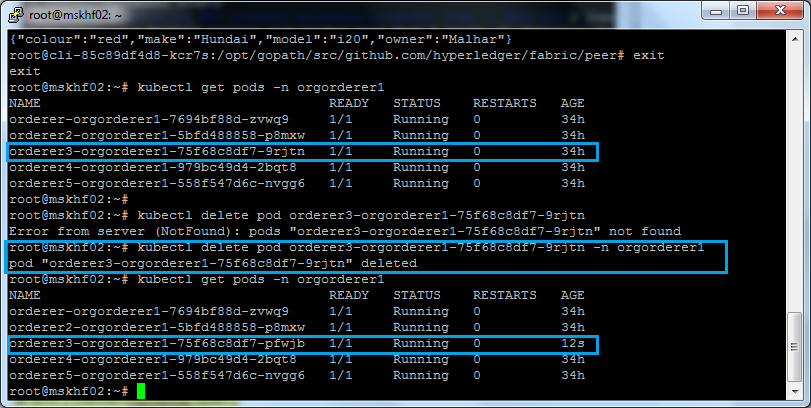


Notice the line:

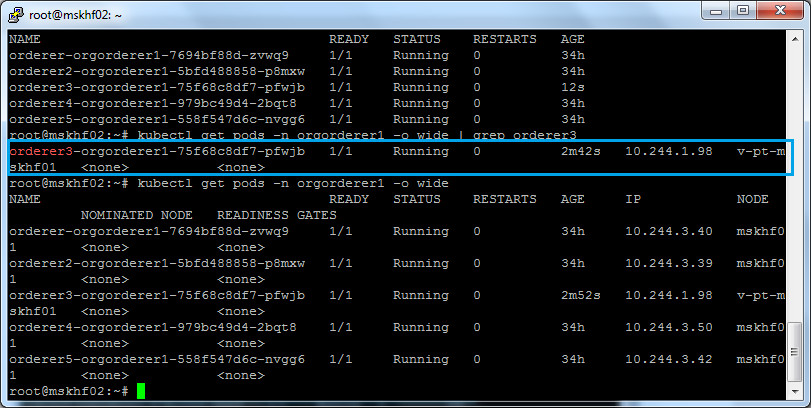
*2019-11-04 05:29:18.468 UTC [orderer.consensus.etcdraft] run -> INFO 049 raft.node: 2 elected* ***leader 3 at*** *term 2 channel=mychannel node=2*

As can be seen above, orderer3 is a RAFT leader currently. Hence to verify leader re-election and POD auto-creation, delete POD ***orderer3-orgorderer1-75f68c8df7-9rjtn***.

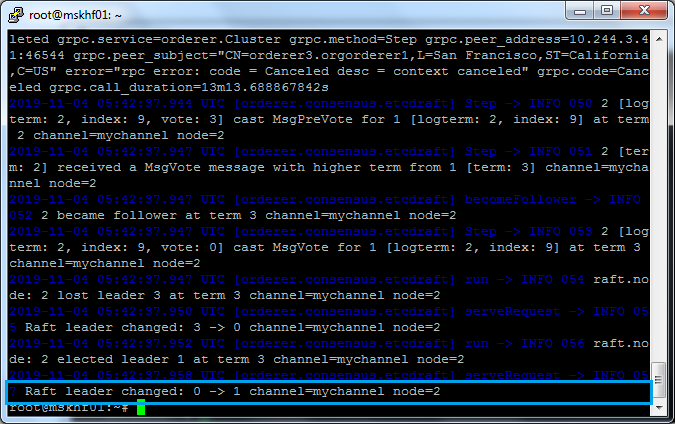
Command: ***kubectl delete pod orderer4-orgorderer1-979bc49d4-x469j -n orgorderer1***

******

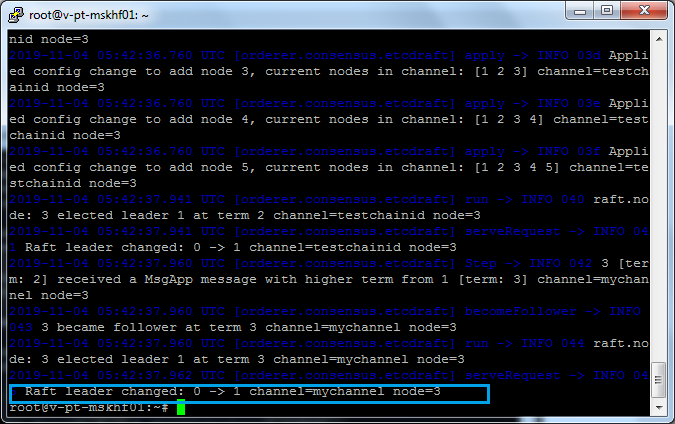
It can be seen that K8S has re-created the orderer3 POD. Now check the node where this POD is created



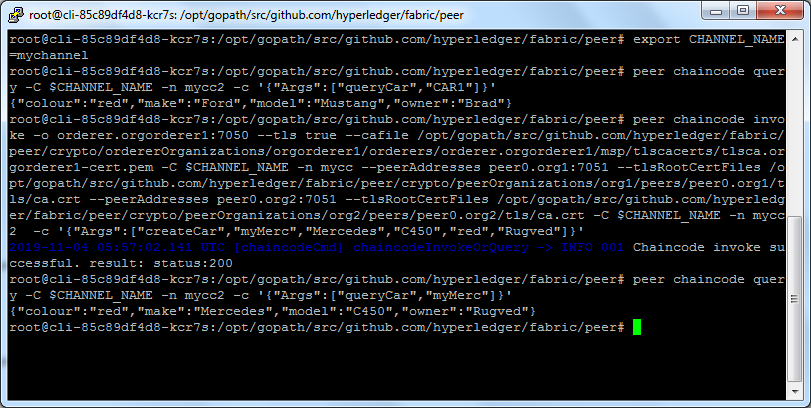
As can be seen, this POD is created on v-pt-mskhf01. Now verify that the new leader is elected by checking logs of any of the orderer.

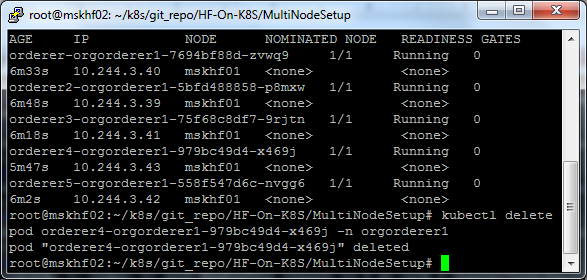


Also verify this, by checking logs of newly created orderer3 container on v-pt-mskhf01:



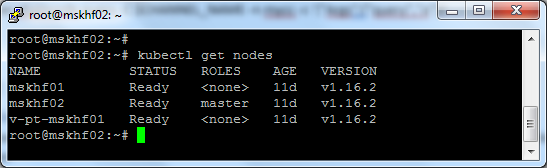
Verify the setup by executing query Chaincode





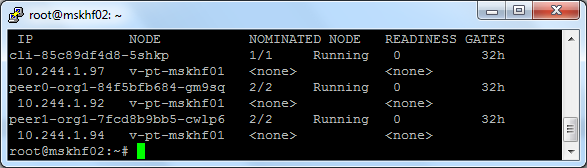
## Failover scenario

1. Verify that K8S cluster has 2 worker nodes and both are in “Ready” state

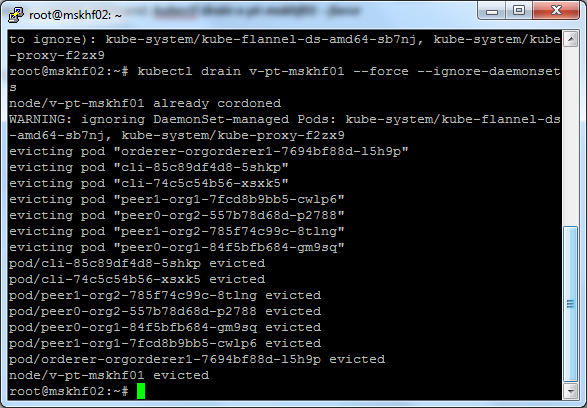


1. Check which node has PODs for say Org1 peers by running following command:

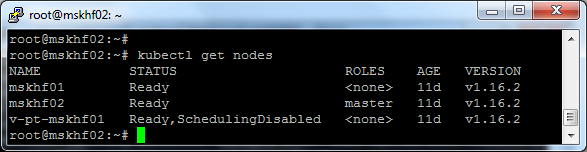
kubectl get pods -n org1 -o wide



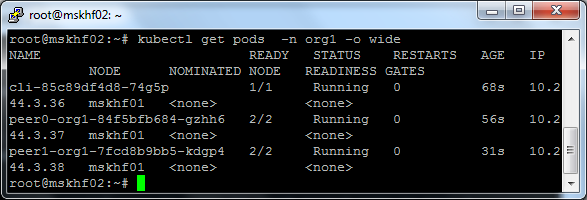
As seen in above screenshot, v-pt-mskhf01 has PODs for org1 peers, hence to test failover, drain this node by running command: ***kubectl drain v-pt-mskhf01 –force –ignore-daemonset***



Verify that the node v-pt-mskhf01 is drained i.e. the status of this node is as shown below



Again run the command ***kubectl get pods -n org1 -o wide*** to verify that K8s has re-created the PODS for org1 peers on the other worker node i.e. mskhf01.



Test the HF setup by running invoke and query commands and verify that state DB is up-to-date

To bring back the node, run command: ***kubectl uncordon <Node name>***

# Cleanup

To cleanup the setup, run **cleanup.sh** script on **K8S master** as described below:

1. Run following to delete all the k8s pods, services and namespaces
   1. *For solo orderer:* ***./cleanup.sh k8s\_deploy solo***
   2. *For etcdraft orderer:* ***./cleanup.sh k8s\_deploy etcdraft***
2. To remove the crypto material, channel artifacts, volume dir and deployment yamls, run following script:

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

1. To cleanup everything(1 and 2 above):
   1. For solo orderer: ***./cleanup.sh all solo***
   2. For etcdraft orderer: ***./cleanup.sh all etcdraft***