Abdul Wahab Follow
Blockchain Enthusiast. I have a passion for understanding things at a fundamental level and sharing it as clearly as possible.
Feb 14 · 10 min read

Hyperledger Fabric on Multiple Hosts

Hyperledger Fabric is a business Blockchain project hosted by Linux Foundation. It is a

> “platform for distributed ledger solutions, underpinned by a modular architecture delivering high degrees of confidentiality, resiliency, flexibility and scalability. It is designed to support pluggable implementations of different components, and accommodate the complexity and intricacies that exist across the economic ecosystem.”

I started working on this technology as a part of a PoC to explore blockchain solutions. Hyperledger Fabric is well documented and has detailed tutorials on getting started. However, it missed the most important guide on a distributed system implementation (deploying on multiple host), which is the essence of a distributed system.

Yes, there is no tutorial or guide that would help you to deploy the network across multiple hosts. Many people are struggling around different forums on the same issue. I decided to study the existing samples first to understand how different components of the hyperledger network are able to communicate on a single host.

Prerequisites

This tutorial requires you to first follow Build You First Network from the Hyperledger tutorials. It is an essential step to learn how it wires up and works in a single host mode. Also, it installs all the necessary prerequisites and dependencies that are important for this tutorial as well.

If you want to skip the above tutorial and want to try this regardless, you must install all the prerequisite as stated below:

> *Before we begin, if you haven’t already done so, you may wish to check that you have all the Prerequisites installed on the platform(s) on which you’ll be developing blockchain applications and/or operating Hyperledger Fabric.*

> *You will also need to download and install the Hyperledger Fabric Samples. You will notice that there are a number of samples included in the* `fabric-samples` *repository. We will be using the* `first-network` *sample. Let's open that sub-directory now.*

After you have installed the above pre-requisites and before moving forward, let's understand how the network works on single host . . .

How it works on single host…

Hyperledger Fabric relies on docker based architecture, and all the components of your hyperledger network run in separate containers with no visibility of the neighboring containers. To make them communicate with each other, they create a network and each container attaches itself to it. You can find it in the docker-compose-cli.yml under "First Network" in fabric-samples repo. You will find that at start of the **docker-compse-cli.yml** file, the network 'byfn' is created, and then all the containers attaches itself to the just created network.

> *By default Compose sets up a single network for your app. Each container for a service joins the default network and is both* reachable *by other containers on that network, and* discoverable *by them at a hostname identical to the container name.*

How will it work for multi host…

But when we are working across multiple hosts, the containers can't communicate with one another. In short, we need to find a way to share this network (to which all the containers attach to) across multiple hosts(PCs). I had a guess that it could be achieved by using docker swarm. and I found :

> *The* `overlay` *network driver creates a distributed network among multiple Docker daemon hosts. This network sits on top of (overlays) the host-specific networks allows containers connected to it (including swarm service containers) to communicate securely. Docker transparently handles routing of each packet to and from the correct Docker daemon host and the correct destination container.*

and then I landed here.

I started merging the pieces and with some trial and error, I was able to pull it off. To do the same, you will need to

1. Initialize docker swarm mode first (let's say on PC 1)
2. and make all the other hosts join that swarm as "manager" (let's say PC 2, PC3 ..)
3. create an overly network so that it could be shared across all other hosts. (The reason other hosts have visibility of network is that they are part of the swarm)

Since this is a distributed system, you will need more than one host (two computers in our case) to verify the distributed nature of the technology. Let's say you have two PCs i.e. **PC1** and **PC2**

This tutorial will only work on **Linux**….why ?

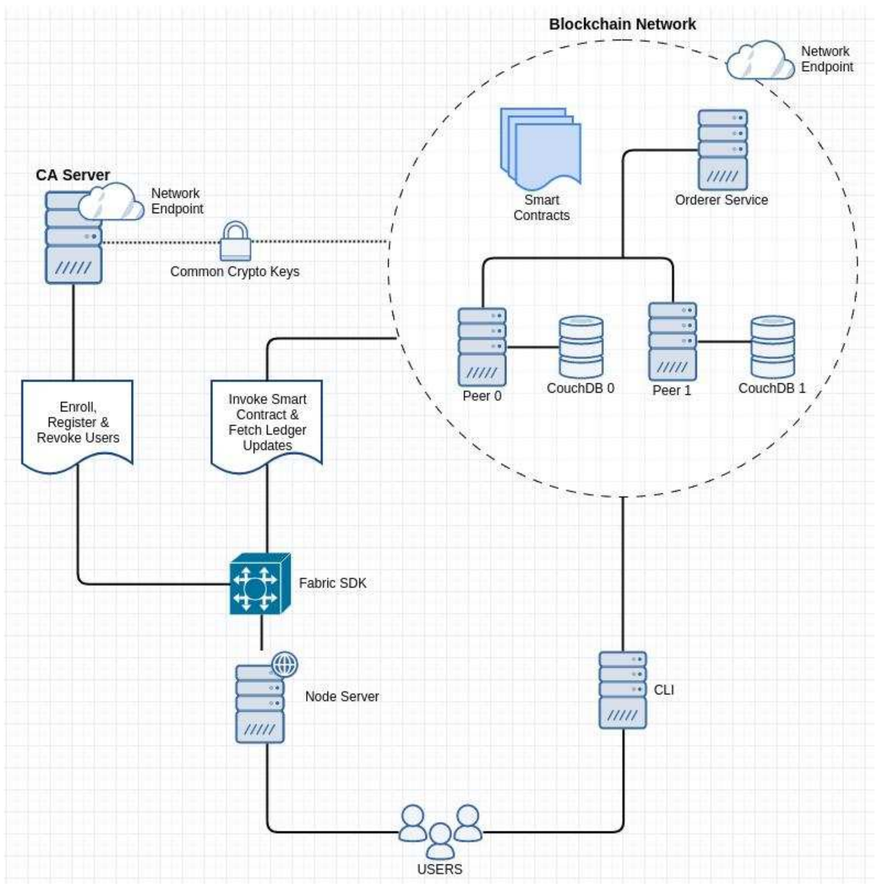
> *Currently, you cannot use Docker for Mac or Docker for Windows alone to test a* multi-node *swarm.*

Network Topology

So the network that we are going to build will have the following below components. For this example we are using two PCs lets say (**PC1 and PC2**):

1. A Certificate Authority (CA)**—PC1**
2. An Orderer—**PC1**
3. 1 PEER (peer0) on—**PC1**
4. 1 PEER (peer1) on—**PC2**
5. CLI on—**PC2**

in the picture …



Before You Start

- Initialize a swarm: (docker swarm documentation for more information)

```
$ docker swarm init
```

- Join the swarm with the other host as a manager (**PC1** will create swarm and **PC2** will join it)

PC1:

```
$ docker swarm join-token manager
```

It will output something like this

```
docker swarm join — token SWMTKN-1-
3as8cvf3yxk8e7zj98954jhjza3w75mngmxh543llgpo0c8k7z-
61zyibtaqjjimkqj8p6t9lwgu 172.16.0.153:2377
```

We will copy it(the one on your terminal, not the one above) and execute it on **PC2** terminal to make it join PC1

use the output command of the previous command on PC2

- Create a network ("my-net" in my case)**—PC1**

```
$ docker network create --attachable --driver overlay
my-net
```

- Clone this repo on both the PCs i.e PC1 and PC2.

```
$ git clone https://github.com/wahabjawed/Build-Multi-
Host-Network-Hyperledger.git
```

- Generate Network Artifacts (Crypto Material)**—PC1**

```
$ cd Build-Multi-Host-Network-Hyperledger/

$ ./bmhn.sh
```

This will generate network artifacts for you in 'crypto-config' and 'channel-artifacts' folder. You must copy these folders on PC2 so that both the projects will have same crypto material. **It is important that both the PCs must have same crypto material otherwise the network will not communicate.**

Setting up the Network

On PC1 :

The below scripts will run on PC1. Execute each command in a **separate terminal**.

Also make sure that you are in "Build-Multi-Host-Network-Hyperledger/" folder before executing any of the script. The scripts utilizes the files in the **"Build-Multi-Host-Network-Hyperledger"** folder and will throw error if it can't locate it.

1. CA Server:

You will execute this command on PC1. before you do so, replace **{put the name of secret key}** with the name of the secret key. You can find it under **'/crypto-config/peerOrganizations/org1.example.com/ca/'**.

```
docker run --rm -it --network="my-net" --name
ca.example.com -p 7054:7054 -e
FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server -e
FABRIC_CA_SERVER_CA_NAME=ca.example.com -e
FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-
ca-server-config/ca.org1.example.com-cert.pem -e
FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-
ca-server-config/{put the name of secret key} -v
$(pwd)/crypto-
config/peerOrganizations/org1.example.com/ca/:/etc/hyp
erledger/fabric-ca-server-config -e
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=hyp-net
hyperledger/fabric-ca sh -c 'fabric-ca-server start -b
admin:adminpw -d'
```

2. Orderer

Execute this command to spawn up the orderer on PC1

```
docker run --rm -it --network="my-net" --name
orderer.example.com -p 7050:7050 -e
ORDERER_GENERAL_LOGLEVEL=debug -e
ORDERER_GENERAL_LISTENADDRESS=0.0.0.0 -e
ORDERER_GENERAL_LISTENPORT=7050 -e
ORDERER_GENERAL_GENESISMETHOD=file -e
ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/o
rderer.genesis.block -e
ORDERER_GENERAL_LOCALMSPID=OrdererMSP -e
ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/m
sp -e ORDERER_GENERAL_TLS_ENABLED=false -e
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=my-net -v
$(pwd)/channel-
artifacts/genesis.block:/var/hyperledger/orderer/order
er.genesis.block -v $(pwd)/crypto-
```

```
config/ordererOrganizations/example.com/orderers/order
er.example.com/msp:/var/hyperledger/orderer/msp -w
/opt/gopath/src/github.com/hyperledger/fabric
hyperledger/fabric-orderer orderer
```

3. CouchDB 0 — for Peer 0

This command will spawn a couchDB instance that will be used by peer0 for storing peer ledger.

```
docker run --rm -it --network="my-net" --name couchdb0
-p 5984:5984 -e COUCHDB_USER= -e COUCHDB_PASSWORD= -e
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=my-net
hyperledger/fabric-couchdb
```

4. Peer 0

And now we execute this command to spawn peer0

```
docker run --rm -it --link
orderer.example.com:orderer.example.com --network="my-
net" --name peer0.org1.example.com -p 8051:7051 -p
8053:7053 -e CORE_LEDGER_STATE_STATEDATABASE=CouchDB -
e
CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb
0:5984 -e CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME= -e
CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD= -e
CORE_PEER_ADDRESSAUTODETECT=true -e
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock -e
CORE_LOGGING_LEVEL=DEBUG -e
CORE_PEER_NETWORKID=peer0.org1.example.com -e
CORE_NEXT=true -e CORE_PEER_ENDORSER_ENABLED=true -e
CORE_PEER_ID=peer0.org1.example.com -e
CORE_PEER_PROFILE_ENABLED=true -e
CORE_PEER_COMMITTER_LEDGER_ORDERER=orderer.example.com
:7050 -e CORE_PEER_GOSSIP_IGNORESECURITY=true -e
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=my-net -e
CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org1.example.c
om:7051 -e CORE_PEER_TLS_ENABLED=false -e
CORE_PEER_GOSSIP_USELEADERELECTION=false -e
CORE_PEER_GOSSIP_ORGLEADER=true -e
CORE_PEER_LOCALMSPID=Org1MSP -v
/var/run/:/host/var/run/ -v $(pwd)/crypto-
config/peerOrganizations/org1.example.com/peers/peer0.
org1.example.com/msp:/etc/hyperledger/fabric/msp -w
/opt/gopath/src/github.com/hyperledger/fabric/peer
hyperledger/fabric-peer peer node start
```

On PC2:

The following below command will be executed on PC2.

Make sure that you are in "Build-Multi-Host-Network-Hyperledger" folder before executing any of the script. The scripts utilizes the files in the **"Build-Multi-Host-Network-Hyperledger"** folder and will throw error if it can't locate it.

5. CouchDB 1—for Peer 1

This command will spawn a couchDB instance that will be used by peer1 for storing peer ledger. We will execute this in a separate terminal on PC2

```
docker run --rm -it --network="my-net" --name couchdb1
-p 6984:5984 -e COUCHDB_USER= -e COUCHDB_PASSWORD= -e
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=my-net
hyperledger/fabric-couchdb
```

6. Peer 1

We will execute this in a separate terminal on PC2 to spawn peer1.

```
docker run --rm -it --network="my-net" --link
orderer.example.com:orderer.example.com --link
peer0.org1.example.com:peer0.org1.example.com --name
peer1.org1.example.com -p 9051:7051 -p 9053:7053 -e
CORE_LEDGER_STATE_STATEDATABASE=CouchDB -e
CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb
1:5984 -e CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME= -e
CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD= -e
CORE_PEER_ADDRESSAUTODETECT=true -e
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock -e
CORE_LOGGING_LEVEL=DEBUG -e
CORE_PEER_NETWORKID=peer1.org1.example.com -e
CORE_NEXT=true -e CORE_PEER_ENDORSER_ENABLED=true -e
CORE_PEER_ID=peer1.org1.example.com -e
CORE_PEER_PROFILE_ENABLED=true -e
CORE_PEER_COMMITTER_LEDGER_ORDERER=orderer.example.com
:7050 -e CORE_PEER_GOSSIP_ORGLEADER=true -e
CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1.org1.example.c
om:7051 -e CORE_PEER_GOSSIP_IGNORESECURITY=true -e
CORE_PEER_LOCALMSPID=Org1MSP -e
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=my-net -e
CORE_PEER_GOSSIP_BOOTSTRAP=peer0.org1.example.com:7051
-e CORE_PEER_GOSSIP_USELEADERELECTION=false -e
CORE_PEER_TLS_ENABLED=false -v
/var/run/:/host/var/run/ -v $(pwd)/crypto-
config/peerOrganizations/org1.example.com/peers/peer1.
```

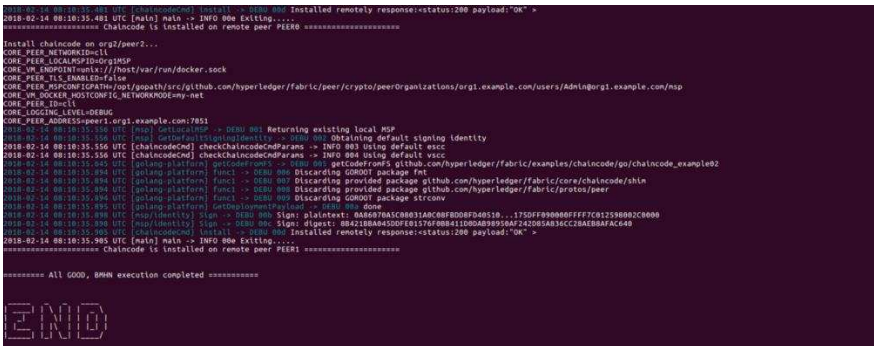
```
org1.example.com/msp:/etc/hyperledger/fabric/msp -w
/opt/gopath/src/github.com/hyperledger/fabric/peer
hyperledger/fabric-peer peer node start
```

7. CLI

Execute the below script in a different terminal on PC2 to spawn CLI.

```
docker run --rm -it --network="my-net" --name cli --
link orderer.example.com:orderer.example.com --link
peer0.org1.example.com:peer0.org1.example.com --link
peer1.org1.example.com:peer1.org1.example.com -p
12051:7051 -p 12053:7053 -e GOPATH=/opt/gopath -e
CORE_PEER_LOCALMSPID=Org1MSP -e
CORE_PEER_TLS_ENABLED=false -e
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock -e
CORE_LOGGING_LEVEL=DEBUG -e CORE_PEER_ID=cli -e
CORE_PEER_ADDRESS=peer0.org1.example.com:7051 -e
CORE_PEER_NETWORKID=cli -e
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyp
erledger/fabric/peer/crypto/peerOrganizations/org1.exa
mple.com/users/Admin@org1.example.com/msp -e
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=my-net  -v
/var/run/:/host/var/run/ -v
$(pwd)/chaincode/:/opt/gopath/src/github.com/hyperledg
er/fabric/examples/chaincode/go -v $(pwd)/crypto-
config:/opt/gopath/src/github.com/hyperledger/fabric/p
eer/crypto/ -v
$(pwd)/scripts:/opt/gopath/src/github.com/hyperledger/
fabric/peer/scripts/ -v $(pwd)/channel-
artifacts:/opt/gopath/src/github.com/hyperledger/fabri
c/peer/channel-artifacts -w
/opt/gopath/src/github.com/hyperledger/fabric/peer
hyperledger/fabric-tools /bin/bash -c
'./scripts/script.sh'
```

If you see this, it means that the script has been executed



```
2018-02-14 08:10:35.481 UTC [chaincodeCmd] install -> DEBU 00d Installed remotely response:<status:200 payload:"OK" >
2018-02-14 08:10:35.481 UTC [main] main -> INFO 00e Exiting.....
===================== Chaincode is installed on remote peer PEER0 =====================

Install chaincode on org2/peer2...
CORE_PEER_NETWORKID=cli
CORE_PEER_LOCALMSPID=Org1MSP
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
CORE_PEER_TLS_ENABLED=false
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=my-net
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer1.org1.example.com:7051
2018-02-14 08:10:35.556 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-02-14 08:10:35.556 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-02-14 08:10:35.556 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 003 Using default escc
2018-02-14 08:10:35.556 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 004 Using default vscc
2018-02-14 08:10:35.645 UTC [golang-platform] getCodeFromFS -> DEBU 005 getCodeFromFS github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02
2018-02-14 08:10:35.894 UTC [golang-platform] func1 -> DEBU 006 Discarding GOROOT package fmt
2018-02-14 08:10:35.894 UTC [golang-platform] func1 -> DEBU 007 Discarding provided package github.com/hyperledger/fabric/core/chaincode/shim
2018-02-14 08:10:35.894 UTC [golang-platform] func1 -> DEBU 008 Discarding provided package github.com/hyperledger/fabric/protos/peer
2018-02-14 08:10:35.894 UTC [golang-platform] func1 -> DEBU 009 Discarding GOROOT package strconv
2018-02-14 08:10:35.895 UTC [golang-platform] GetDeploymentPayload -> DEBU 00a done
2018-02-14 08:10:35.898 UTC [msp/identity] Sign -> DEBU 00b Sign: plaintext: 0A86070A5C08031A0C08FBDD8FD40510...175DFF090000FFFF7C012598002C0000
2018-02-14 08:10:35.898 UTC [msp/identity] Sign -> DEBU 00c Sign: digest: 8B421BBA045DDFE01576F0BB411D0DAB98950AF242D85A836CC28AEB8AFAC640
2018-02-14 08:10:35.905 UTC [chaincodeCmd] install -> DEBU 00d Installed remotely response:<status:200 payload:"OK" >
2018-02-14 08:10:35.905 UTC [main] main -> INFO 00e Exiting.....
===================== Chaincode is installed on remote peer PEER1 =====================

========= All GOOD, BMHN execution completed ===========
```

This will install the CLI container and will execute the script :

```
'./scripts/script.sh'
```

The script will:

- Create channel; **mychannel** in our case
- Make peer0 and peer1 join the channel.
- Upon successful joining of the channel, the script will update the anchor peer (peer0 in our case).
- Install the chaincode on both peers

Now our network is up and running, let's test it out. Now we will invoke and query chaincode on both peers from PC2.

Let's do it

Testing the Network

Step 1. Bin/Bash CLI—PC2

We will again spawn the cli container on PC2, but this time we will exec into it

```
docker run --rm -it --network="my-net" --name cli --
link orderer.example.com:orderer.example.com --link
peer0.org1.example.com:peer0.org1.example.com --link
peer1.org1.example.com:peer1.org1.example.com -p
12051:7051 -p 12053:7053 -e GOPATH=/opt/gopath -e
CORE_PEER_LOCALMSPID=Org1MSP -e
CORE_PEER_TLS_ENABLED=false -e
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock -e
CORE_LOGGING_LEVEL=DEBUG -e CORE_PEER_ID=cli -e
CORE_PEER_ADDRESS=peer0.org1.example.com:7051 -e
CORE_PEER_NETWORKID=cli -e
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyp
erledger/fabric/peer/crypto/peerOrganizations/org1.exa
mple.com/users/Admin@org1.example.com/msp -e
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=my-net  -v
/var/run/:/host/var/run/ -v
$(pwd)/chaincode/:/opt/gopath/src/github.com/hyperledg
er/fabric/examples/chaincode/go -v $(pwd)/crypto-
config:/opt/gopath/src/github.com/hyperledger/fabric/p
eer/crypto/ -v
$(pwd)/scripts:/opt/gopath/src/github.com/hyperledger/
fabric/peer/scripts/ -v $(pwd)/channel-
artifacts:/opt/gopath/src/github.com/hyperledger/fabri
```

```
c/peer/channel-artifacts -w
/opt/gopath/src/github.com/hyperledger/fabric/peer
hyperledger/fabric-tools /bin/bash
```

You must see this after executing the command.

```
root@a14d67c2dbb5:/opt/gopath/src/github.com/hyperledg
er/fabric/peer#
```

Now that you have entered into CLI container, we will execute the commands to instantiate, invoke and query the chaincode in this container.

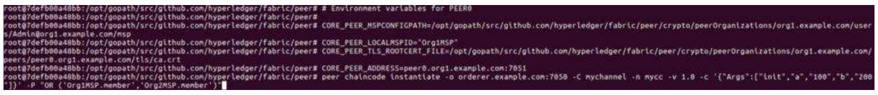
Step 2. Instantiate Chaincode on Peer0

To instantiate the chaincode on peer0 we will need to set few environment variables first. Paste the below line in the cli terminal.

```
# Environment variables for PEER0

CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyp
erledger/fabric/peer/crypto/peerOrganizations/org1.exa
mple.com/users/Admin@org1.example.com/msp

CORE_PEER_LOCALMSPID="Org1MSP"

CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com
/hyperledger/fabric/peer/crypto/peerOrganizations/org1
.example.com/peers/peer0.org1.example.com/tls/ca.crt

CORE_PEER_ADDRESS=peer0.org1.example.com:7051
```

after that we will initialize chaincode. Execute the below command to instantiate the chaincode that was installed as a part of step 1.

```
$ peer chaincode instantiate -o
orderer.example.com:7050 -C mychannel -n mycc -v 1.0 -
c '{"Args":["init","a","100","b","200"]}' -P "OR
('Org1MSP.member','Org2MSP.member')"
```



```
root@7defb00a48bb:/opt/gopath/src/github.com/hyperledger/fabric/peer# # Environment variables for PEER0
root@7defb00a48bb:/opt/gopath/src/github.com/hyperledger/fabric/peer#
root@7defb00a48bb:/opt/gopath/src/github.com/hyperledger/fabric/peer# CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/user
s/Admin@org1.example.com/msp
root@7defb00a48bb:/opt/gopath/src/github.com/hyperledger/fabric/peer# CORE_PEER_LOCALMSPID="Org1MSP"
root@7defb00a48bb:/opt/gopath/src/github.com/hyperledger/fabric/peer# CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/
peers/peer0.org1.example.com/tls/ca.crt
root@7defb00a48bb:/opt/gopath/src/github.com/hyperledger/fabric/peer# CORE_PEER_ADDRESS=peer0.org1.example.com:7051
root@7defb00a48bb:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode instantiate -o orderer.example.com:7050 -C mychannel -n mycc -v 1.0 -c '{"Args":["init","a","100","b","200
"]}' -P "OR ('Org1MSP.member','Org2MSP.member')"
```

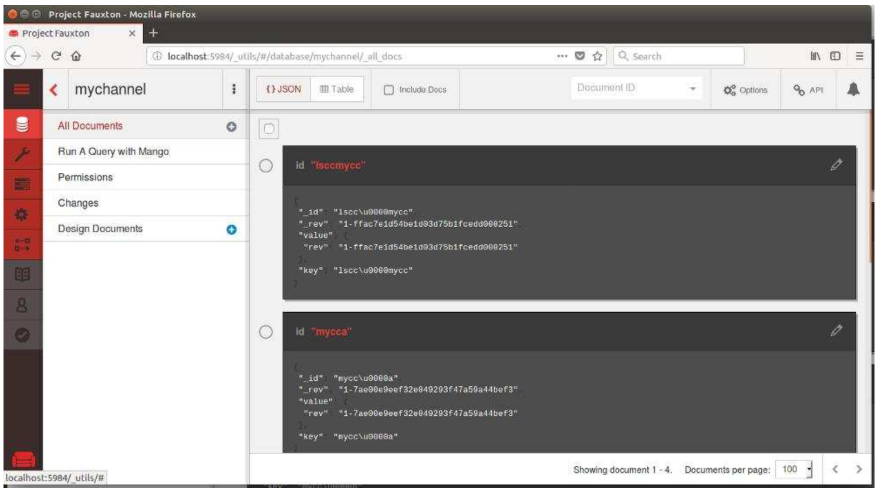
This will instantiate the chiancode and populate it with a =100 and b = 200.

At this point, as your ledger is populated, you can view the transactions at **(open it on browser on PC1)**

Peer 0 (PC 1):
http://localhost:5984/_utils/#/database/mychannel/_all_docs

Peer1 (PC 2):
http://localhost:6984/_utils/#/database/mychannel/_all_docs

The above are the couchDB web interfaces endpoints. Since the data is saved in binary, you won't find exact values(instead you will find hashes) but will see the records having key containing "myacc".



```
id "lsccmycc"
{
 "_id": "lscc\u0000mycc",
 "_rev": "1-ffac7e1d54be1d03d75b1fcedd008251",
 "value": {
  "rev": "1-ffac7e1d54be1d03d75b1fcedd008251"
 },
 "key": "lscc\u0000mycc"
}

id "mycca"
{
 "_id": "mycc\u0000a",
 "_rev": "1-7ae00e9eef32e049293f47a59a44bef3",
 "value": {
  "rev": "1-7ae00e9eef32e049293f47a59a44bef3"
 },
 "key": "mycc\u0000a"
}
```

Showing document 1 - 4. Documents per page: 100

OR

Let's Query it and see the results. We will query it on **peer1**

Step 3. Query the Chaincode on Peer1

To query the chaincode on peer1 we will need to set few environment variables first. Paste the below line in the cli terminal on PC2

```
# Environment variables for PEER1

CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyp
erledger/fabric/peer/crypto/peerOrganizations/org1.exa
```

```
mple.com/users/Admin@org1.example.com/msp

CORE_PEER_LOCALMSPID="Org1MSP"

CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com
/hyperledger/fabric/peer/crypto/peerOrganizations/org1
.example.com/peers/peer0.org1.example.com/tls/ca.crt

CORE_PEER_ADDRESS=peer1.org1.example.com:7051
```

Let's query for the value of `a` to make sure the chaincode was properly instantiated and the couch DB was populated. The syntax for query is as follows: (execute in cli terminal) and wait for a while

```
$ peer chaincode query -C mychannel -n mycc -c
'{"Args":["query","a"]}'
```

it will bring

```
Query Result: 100
```

Step 4. Invoke the Chaincode on Peer0

To invoke the chaincode on peer0 we will need to set few environment variables first. Paste the below line in the cli terminal on PC2

```
# Environment variables for PEER0

CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyp
erledger/fabric/peer/crypto/peerOrganizations/org1.exa
mple.com/users/Admin@org1.example.com/msp

CORE_PEER_LOCALMSPID="Org1MSP"

CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com
/hyperledger/fabric/peer/crypto/peerOrganizations/org1
.example.com/peers/peer0.org1.example.com/tls/ca.crt

CORE_PEER_ADDRESS=peer0.org1.example.com:7051
```

Now let's move `10` from `a` to `b`. This transaction will cut a new block and update the couch DB. The syntax for invoke is as follows: (execute in cli terminal on PC2)

```
$ peer chaincode invoke -o orderer.example.com:7050 -C
mychannel -n mycc -c '{"Args":
["invoke","a","b","10"]}'
```

Step 5. Query the Chaincode

Let's confirm that our previous invocation executed properly. We initialized the key `a` with a value of `100` and just removed `10` with our previous invocation. Therefore, a query against `a` should reveal `90`. The syntax for query is as follows. (we are querying on peer0 so no need to change the environment variables)

```
# be sure to set the -C and -n flags appropriately

peer chaincode query -C mychannel -n mycc -c '{"Args":
["query","a"]}'
```

We should see the following:

```
Query Result: 90
```

Feel free to start over and manipulate the key value pairs and subsequent invocations.

Whats Next….

The implementation of chaincode execution on hyperledger fabric tutorials are still in development mode, that is why I have been playing around with load balancing chaincode execution using HProxy to make it production ready.

Also, I have been working around with blockchain explorer to monitor the different matrices of the network in action i.e. blocks, transaction, throughput, active nodes and etc.

I will post them soon.

Kindly try this tutorial and feel free to feedback if you face any issue in the implementation. I will be more than happy to assist you.

Happy Blockchaining :)