Hyperledger fabric first network tutorial

Installing Prerequisites for Hyperledger Fabric:

- **CURL** : curl is a command line tool to transfer data to or from a server, using any of the supported protocols (HTTP, FTP, IMAP, POP3, SCP, SFTP, SMTP, TFTP, TELNET, LDAP or FILE). curl can transfer multiple file at once.
- **GO Programming Language with setup of path variable**: GoLang is a very powerful programming language developed by Google. It is a compiled programming language. It means, Go source codes are converted to machine code or commonly known as executable file. Then you can run these executable files on other computers. Unlike Java that converts source code to byte code, then runs these byte codes using JVM (Java Virtual Machine), Go does not use any VM (Virtual Machines).
- **Docker & Docker Compose**: Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration. ... Run docker-compose up and Compose starts and runs your entire app
- **Node.js Runtime & NPM :** js runtime is basically what will understand your javascript code and execute it to produce a result. Npm package manager is a tool which will allow you to install third party libraries (other people's code) by using the command line. npm install express**.**
- **PYTHON**

To install all these prerequisites run following commands in terminal window one by one, everything should go smooth.

```
sudo apt-get install curl
sudo apt-get install golang-go
export GOPATH=$HOME/go
export PATH=$PATH:$GOPATH/bin
sudo apt-get install nodejs
sudo apt-get install npm
sudo apt-get install python
sudo apt-get install docker
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get update
apt-cache policy docker-ce
sudo apt-get install -y docker-ce
sudo apt-get install docker-compose
sudo apt-get upgrade
```

Till now the environment is ready to install Hyperledger Fabric, Now we will download fabric samples. To do that run following commands in terminal one by one…

```
sudo curl -sSL https://goo.gl/6wtTN5 | sudo bash -s 1.1.0
sudo chmod 777 -R fabric-samples
```

Now go into first-network directory which is inside fabric-samples folder and then we will run generate script that will create certificates and keys for the entities on our first blockchain network. This will also create genesis block (first block on blockchain)

```
cd fabric-samples/first-network
sudo ./byfn.sh generate
```

Bring your first network up by running following command. ***byfn*** stands for "**B**uild **Y**our **F**irst **N**etwork"

```
sudo ./byfn.sh up
```

If everything works fine you should see start screen of fabric network. To bring network down following command is used.

```
sudo ./byfn.sh down
```

List of files in BYFN

In this part 1 tutorial, I am going to introduce files in BYFN. I plan to introduce all files in 2 parts. Then, we continue some practical steps.

Let's switch to BYFN directory (*supposed that you finish the prerequisite part, you should have all needed files and directories):

```
cd fabric-samples/first-network
```

This is the file or directory list that we will study:

1. ***docker-compose-cli.yaml (Part 1)***
2. base/docker-compose-base.yaml
3. base/peer-base.yaml
4. channel-artifacts/
5. crypto-config.yaml
6. configtx.yaml

7. byfn.sh
8. scripts/script.sh
9. script/utils.sh

There are other files in fabric-samples/first-network/, but they are not directly related to BYFN, we will ignore them.

docker-compose-cli.yaml

In order to develop Blockchain application, we need to have a Blockchain network first, right? This is a file related to setting up the network.

This is a Docker compose file, which defines your (virtual) ***Fabric network***, such as what nodes are in the network, their internal use domain names, etc.

Below is `docker-compose-cli.yaml`

```
1  networks:
2    byfn:
3
4  services:
5    orderer.example.com:
6      extends:
7        file:   base/docker-compose-base.yaml
8        service: orderer.example.com
9      container_name: orderer.example.com
10     networks:
11       - byfn
12
13   peer0.org1.example.com:
14     container_name: peer0.org1.example.com
15     extends:
16       file:  base/docker-compose-base.yaml
17       service: peer0.org1.example.com
18     networks:
19       - byfn
20
21   peer1.org1.example.com:
22     container_name: peer1.org1.example.com
23     extends:
24       file:  base/docker-compose-base.yaml
25       service: peer1.org1.example.com
26     networks:
27       - byfn
28
29   peer0.org2.example.com:
30     container_name: peer0.org2.example.com
31     extends:
32       file:  base/docker-compose-base.yaml
33       service: peer0.org2.example.com
34     networks:
35       - byfn
36
37   peer1.org2.example.com:
38     container_name: peer1.org2.example.com
```

Line 1 defines our ***Docker network*** name, such that our nodes could have a “place” to talk to each other. And this “place” is named ***byfn***.

Line 4 defines a list of nodes in the network.

Line 5 defines a node in our network — ***Orderer***, which helps ***decide how transactions and blocks are ordered*** in the Blockchain network.

You see in line 6, there is an ***extends***, which is to re-use another ***base*** Docker compose file. I could smell the odor of “Inheritance” in programming.

Line 9 defines this node’s name in the network. Finally, line 10 defines which network this node belongs to.

base/docker-compose-base.yaml

In part 1, we have introduced***docker-compose-cli.yaml,*** which is to define our (virtual) Fabric network. We review a part of `docker-compose-cli.yaml`

```
1  # Part of docker-compose-cli.yaml
2  networks:
3    byfn:
4
5  services:
6    orderer.example.com:
7      extends: # notice here!
8        file:   base/docker-compose-base.yaml
9        service: orderer.example.com
10     container_name: orderer.example.com
11     networks:
12       - byfn
```

Notice that there is an ***extends*** keyword. As discussed in part 1, it is to ***re-use*** another ***base*** Docker compose file. The purpose of this is to make things ***cleaner*** — by grouping the shared configurations in one base file, such that other Docker compose files can ***extend*** it. It is about good practice — ***reusability***, but this is not mandatory.

Then, the ***file*** keyword in line 8 specifies which base file to extend, and ***service*** keyword specifies which item (service) in the base file to extend.

Now, we look at ***`base/docker-compose-base.yaml`***:

The base file is similar to ***`docker-compose-cli.yaml`***.

Lines 3 to 28 define a template node called ***orderer.example.com***, and ***`docker-compose-cli.yaml`*** extends it and add some extra configurations based on this template.

Line 5 specifies which image this node uses. In our case, the node uses an image from Hyperledger official — ***hyperledger/fabric-orderer***. That ***$IMAGE_TAG*** is the image version, such as ***1.4***.

Other keywords and configurations are similar to ***`docker-compose-cli.yaml`***. The point of having a base file is just to group common configurations to a base file to make them reusable.

Notice that in line 32, there is also an ***extends*** keyword, that is, again, to extend or reuse another base file,***`base/peer-base.yaml`***, which we will quickly take a look.

```
services:

  orderer.example.com:
    container_name: orderer.example.com
    image: hyperledger/fabric-orderer:$IMAGE_TAG
    environment:
      - FABRIC_LOGGING_SPEC=INFO
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_GENESISMETHOD=file
      - ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block
      - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
      - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
      # enabled TLS
      - ORDERER_GENERAL_TLS_ENABLED=true
      - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
      - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
      - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
      - ORDERER_KAFKA_TOPIC_REPLICATIONFACTOR=1
      - ORDERER_KAFKA_VERBOSE=true
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric
    command: orderer
    volumes:
    - ../channel-artifacts/genesis.block:/var/hyperledger/orderer/orderer.genesis.block
    - ../crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/msp:/
    - ../crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/:
    - orderer.example.com:/var/hyperledger/production/orderer
    ports:
      - 7050:7050

  peer0.org1.example.com:
    container_name: peer0.org1.example.com
    extends:
      file: peer-base.yaml
```

channel-artifacts/

Notice that this is an empty directory currently. This directory ***will*** be used to store some configuration transactions and a genesis block. More details later.

crypto-config.yaml

This is a file to ***define certifications and keys to be generated and used in the network***.

In Hyperledger Fabric, we have a set of certifications and keys for users and nodes. For example, Peer needs to have a set of certificates and keys to perform endorsement, prove itself as a member in the Blockchain network, do signing, etc.

byfn.sh

This is a convenient script file by Hyperledger Fabric to start the network.

We use it this way to bring up the network (we will run together later):

```
./byfn.sh up
```

Basically, it does the followings:

1. Generate certificates and keys based on ***crypto-config.yaml***
2. Generate channel artifacts based on ***configtx.yaml***, outputs will be stored in ***channel-artifacts/***
3. Bring up the (virtual) Fabric network based on ***docker-compose-cli.yaml***
4. Create Hyperledger Fabric channel, ***mychannel***
5. Join Peer nodes into the channel
6. Install the Smart Contract (Chaincode) in Peer nodes
7. Instantiate the Smart Contract (Chaincode) in one of the Peer nodes

Channel in Hyperledger Fabric is a private group in a Blockchain network. And each Blockchain network could contain multiple Channels, where each Channel is independent to other Channels, has its own ledger, and contains (multiple) organisation(s).

Run the following script. This script will clone the pre-packaged Hyperledger Fabric samples and download the binaries you need. Then navigate to the `first-network` directory we'll work out of with the `cd` command below.

```
> curl -sSL https://goo.gl/6wtTN5 | bash -s 1.1.0> cd fabric-samples/first-network
```

Set an environment variable with a path to your binaries so Fabric knows where to find them. Replace the `< >` part below with the full path of the directory where the `bin` folder is found. You can find that by typing `pwd` in your terminal.

```
> export PATH=<replace this with your path>/bin:$PATH
```

Great! We have everything we need to proceed with getting our network set up.

Create Network Entities

We need to now create the participants in our network. Since they are permissioned participants, we need to give them all unique, secure IDs. Here is how we do it.

We'll run the commands first and explain what's happening after.

```
> ../bin/cryptogen generate --config=./crypto-config.yaml
```

You should see:

```
org1.example.com
org2.example.com
> export FABRIC_CFG_PATH=$PWD> ../bin/configtxgen -profile
TwoOrgsOrdererGenesis -outputBlock ./channel-artifacts/genesis.block
```

You should see:

```
2018-06-14 19:59:35.081 PDT [common/tools/configtxgen] main -> INFO 001 Loading configuration
2018-06-14 19:59:35.095 PDT [msp] getMspConfig -> INFO 002 Loading NodeOUs
2018-06-14 19:59:35.095 PDT [msp] getMspConfig -> INFO 003 Loading NodeOUs
2018-06-14 19:59:35.096 PDT [common/tools/configtxgen] doOutputBlock -> INFO 004 Generating
genesis block
2018-06-14 19:59:35.096 PDT [common/tools/configtxgen] doOutputBlock -> INFO 005 Writing
genesis block
```

So what happened here? Simply,

- We created 2 organizations
- We created 2 peers per organization
- We created certificates for each of the above, so each transaction can be signed by them and we know who created and signed the transactions
- We created a genesis block

Next we need to create channels where our peers can interact and create transactions. We'll call this **mychannel** but feel free to change it to whatever you want.

```
> export CHANNEL_NAME=mychannel  && ../bin/configtxgen -profile
TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -
channelID $CHANNEL_NAME> ../bin/configtxgen -profile TwoOrgsChannel -
outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID
$CHANNEL_NAME -asOrg Org1MSP> ../bin/configtxgen -profile TwoOrgsChannel -
outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID
$CHANNEL_NAME -asOrg Org2MSP
```

The last two lines are important. **Anchor Peers** are created so that new participants who join the network can talk to it and find out who the other participants are in the channel.

Enough setup! Let's start our network!

We'll use Docker to bring up our network.

```
> docker-compose -f docker-compose-cli.yaml up -d
```

You'll see this:

```
Creating network "net_byfn" with the default driver
Creating volume "net_orderer.example.com" with default driver
Creating volume "net_peer0.org1.example.com" with default driver
Creating volume "net_peer1.org1.example.com" with default driver
Creating volume "net_peer0.org2.example.com" with default driver
Creating volume "net_peer1.org2.example.com" with default driver
Creating peer0.org1.example.com ... done
Creating peer0.org2.example.com ... done
Creating peer1.org1.example.com ... done
Creating orderer.example.com    ... done
Creating peer1.org2.example.com ... done
Creating cli                    ... done
```

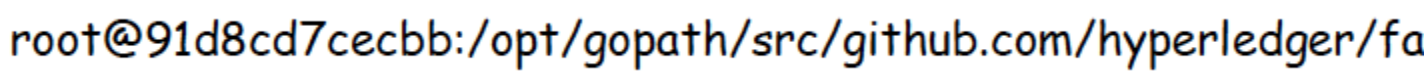
Let's also set up our Docker command line interface so we can enter it and execute our transaction commands.

```
> docker start cli
```

Now let's enter our Docker container.

```
> docker exec -it cli bash
```

Voila! We're now in our container:



We'll now pass in the channel configuration we created earlier so our container can start the channel.

```
> export CHANNEL_NAME=mychannel> peer channel create -o
orderer.example.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/channel.tx -
-tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganization
s/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-
cert.pem
```

Let's now add our peers to the channel. We'll run through the commands first then explain them.

```
> peer channel join -b
mychannel.block>CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledge
r/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.exam
ple.com/msp CORE_PEER_ADDRESS=peer0.org2.example.com:7051
CORE_PEER_LOCALMSPID="Org2MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/pee
r/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/
ca.crt peer channel join -b mychannel.block> peer channel update -o
orderer.example.com:7050 -c $CHANNEL_NAME -f ./channel-
artifacts/Org1MSPanchors.tx --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganization
s/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-
cert.pem>
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/cr
ypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_ADDRESS=peer0.org2.example.com:7051 CORE_PEER_LOCALMSPID="Org2MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/pee
r/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/
ca.crt peer channel update -o orderer.example.com:7050 -c $CHANNEL_NAME -f
./channel-artifacts/Org2MSPanchors.tx --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganization
s/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-
cert.pem
```

What did we just do?

- We joined the first peer of our first organization
- We joined the first peer of our second organization and updated the environment variables accordingly to recognize it
- We made these two peers the Anchor Peers of each organization so new peers can talk to them and learn about other peers

Install our Smart Contract

Remember, smart contracts are referred to as “chaincode” in Fabric. Here, we will pull some pre-packaged chaincode that Fabric provides and install it to our network.

```
> peer chaincode install -n mycc -v 1.0 -p
github.com/chaincode/chaincode_example02/go/> peer chaincode instantiate -o
orderer.example.com:7050 --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganization
s/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-
cert.pem -C $CHANNEL_NAME -n mycc -v 1.0 -c '{"Args":["init","a", "100",
"b","200"]}' -P "OR ('Org1MSP.peer','Org2MSP.peer')"
```

You’ll see:

```
2018-06-15 02:57:28.164 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using
default escc
2018-06-15 02:57:28.164 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using
default vscc
2018-06-15 02:57:50.860 UTC [main] main -> INFO 003 Exiting.....
=================== Chaincode Instantiation on peer0.org2 on channel 'mychannel' is successful
===================
```

What did we do?

- We pulled the chaincode from Github
- We instantiated it then set asset balances (you can think of these as token balances for simplicity). We set “a”, or the first peer we created as having 100 tokens, and “b”, the second peer as having 200 tokens.

The fun stuff!

We’ve got our Fabric network up and running with a couple peers with starting token balances! Let’s play around with it and send some tokens around.

Let’s double check and see how many tokens “a” has

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

```
Query Result: 100
2018-06-15 02:58:14.066 UTC [main] main -> INFO 003 Exiting.....
==================== Query on peer0.org1 on channel 'mychannel' is successful
====================
```

As expected, we see that “a” has 100 tokens. Now let’s send 10 tokens from “a” to “b”

```
> peer chaincode invoke -o orderer.example.com:7050  --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganization
s/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-
cert.pem  -C $CHANNEL_NAME -n mycc -c '{"Args":["invoke","a","b","10"]}'
```

Now let’s try the same query from before. In theory, “a” should now have 90 tokens.

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

Lo and behold!

```
Query Result: 90
2018-06-15 02:58:36.250 UTC [main] main -> INFO 003 Exiting.....
==================== Query on peer1.org2 on channel 'mychannel' is successful
====================
```

Congratulations!

You’ve just got an entire Hyperledger Fabric network up and running, installed chaincode on it and created transactions between peers. This is no small feat. You’ve just run through the core steps of starting a blockchain enterprise application.

Our mission in this tutorial was to give you a simple way to run through the Fabric documentation without being encumbered by all the unnecessary minutia they give, particularly when you just want to try Hyperledger for yourself. If you want the full details of all the steps above, feel free to refer to their more detailed documentation.

Next Steps

You already have all the tools you need to run your own Fabric blockchain and add transactions. **In the next tutorial, we'll be taking a deeper dive at chaincode and showing you how to program your own smart contracts on Fabric with Go.**

You've successfully completed your first step in becoming an enterprise blockchain developer!

Also, be sure to read our previous posts. If you were curious about the basic concepts about the blockchain that supported this tutorial, these articles are the perfect place to start:

- Code your own blockchain
- Networking
- Proof of Work
- Proof of Stake
- IPFS
- P2P
- Advanced Blockchain Concepts
- Build a DApp on Hyperledger

Until next time, happy blockchain programming!