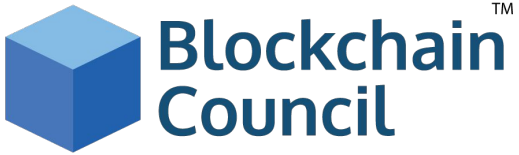

Certified Blockchain Expert

Hyperledger Fabric Building Your First Network

Download and Install

Before we start, you need to make sure correct prerequisites are installed at your operating system.

You will also need to Install Samples, Binaries and Docker Images.

All the binaries and samples are included in the repository. We will be using the first-network sample to see the Hyperledger Fabric in action.

Generate your Network - Automatically

If you go to the first-network directory, there is script file named `byfn.sh` which helps you build the first network automatically.

If you open this file, you may notice that it helps you create 4 peers representing two different organizations, and an orderer node. It will also launch a container to run a script execution that will join peers to a channel, deploy and instantiate chaincode and drive execution of transactions against the deployed chaincode.

To generate all the artifacts mentioned in the script use the following command:

```
./byfn.sh generate
```

Start your Network - Automatically

The next step is to bring up your network.

By bringing up the network we would compile Golang chaincode images and spin up the corresponding containers. Go is the default chaincode language, however there is also support for Node.js chaincode.

You can start your network by running the following command:

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

Stop your Network - Automatically

Finally, let's bring it all down. This will kill your containers, remove the crypto material and four artifacts, and delete the chaincode images from your Docker Registry. Following command will bring your network down.

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

Behind the Scenes

- When we run *byfn.sh*, it runs a *script.sh* that performs all the magic. We can create our own script for the same.
- *script.sh* is in cli container that drives the command createChannel against the supplied channel name. It uses the *channel.tx* file for channel configuration.
- The output of *createChannel* is a genesis block - *<your_channel_name>.block* - which gets stored on the peers' file systems and contains the channel configuration specified from channel.tx.
- The *joinChannel* command is exercised for all four peers, which takes as input the previously generated genesis block. This command instructs the peers to join *<your_channel_name>* and create a chain starting with *<your_channel_name>.block*.

Behind the Scenes

- Now we have a channel consisting of four peers, and two organizations. This is our `TwoOrgsChannel` profile.
- peer0.org1.example.com and peer1.org1.example.com belong to Org1 & peer0.org2.example.com and peer1.org2.example.com belong to Org2.
- These relationships are defined through the `crypto-config.yaml` and the MSP path is specified in our docker compose.
- The anchor peers for Org1MSP (peer0.org1.example.com) and Org2MSP (peer0.org2.example.com) are then updated. We do this by passing the `Org1MSPanchors.tx` and `Org2MSPanchors.tx` artifacts to the ordering service along with the name of our channel.
- An example chaincode is installed on peer0.org1.example.com and peer0.org2.example.com.

Behind the Scenes

- The chaincode is then “instantiated” on peer0.org2.example.com. Instantiation adds the chaincode to the channel, starts the container for the target peer, and initializes the key value pairs associated with the chaincode. The initial values for this example are `["a","100" "b","200"]`. This “instantiation” results in a container by the name of dev-peer0.org2.example.com-mycc-1.0 starting.
- The instantiation also passes in an argument for the endorsement policy. The policy is defined as *`-P "OR ('Org1MSP.peer','Org2MSP.peer')`*", meaning that any transaction must be endorsed by a peer tied to Org1 or Org2.
- A query against the value of “a” is issued to peer0.org1.example.com. The chaincode was previously installed on peer0.org1.example.com, so this will start a container for Org1 peer0 by the name of *`dev-peer0.org1.example.com-mycc-1.0`*. The result of the query is also returned. No write operations have occurred, so a query against “a” will still return a value of “100”.

Behind the Scenes

- An invoke request is sent to peer0.org1.example.com to move “10” from “a” to “b”
- The chaincode is then installed on peer1.org2.example.com
- A query is sent to peer1.org2.example.com for the value of “a”. This starts a third chaincode container by the name of `dev-peer1.org2.example.com-mycc-1.0`. A value of 90 is returned, correctly reflecting the previous transaction during which the value for key “a” was modified by 10.

Things to Notice

- Chaincode must be installed on a peer in order for it to successfully perform read/write operations against the ledger.
- All peers in a channel maintain an exact copy of the ledger which comprises the blockchain to store the immutable, sequenced record in blocks, as well as a state database to maintain a snapshot of the current state.
- The chaincode is only accessible after it is installed because it has already been instantiated.

THANK YOU!

Any questions?

You can mail us at

hello@blockchain-council.org