**Hyperledger Fabric Technical Guideline for Intermediate Developers**



|  |  |
| --- | --- |
| Version | 1.4 |
| Date | 19 Sept 2019 |

Contents

[1. Version Control 3](#_Toc19807532)

[2. Introduction 4](#_Toc19807533)

[3. Deploy a hyperledger network based on balance transfer sample in single machine 5](#_Toc19807534)

[4. Add peers in both organizations 7](#_Toc19807535)

[5. Deploy a kafka-based multi-orderer network in single machine 12](#_Toc19807536)

[6. Network backup and recovery 17](#_Toc19807537)

[7. Deploy CouchDB as state database with rich query functionality 20](#_Toc19807538)

[8. Deploy MongoDB for storage of user credentials 21](#_Toc19807539)

[9. Deploy Hyperledger Explorer 22](#_Toc19807540)

[10. Deploy a cross-machine kafka-based multi-orderer network with Docker Swarm 23](#_Toc19807541)

[11. Deploy a cross-machine kafka-based multi-orderer network with hosts file 27](#_Toc19807542)

[12. Remarks on high-availability of the Fabric network 30](#_Toc19807543)

[13. Further study 31](#_Toc19807544)

# Version Control

## Updated Sections

* 9

# Introduction

## Before you start

* This guideline is intended for implementation. For concepts and high-level architecture, please refer to <https://hyperledger-fabric.readthedocs.io/en/release-1.1/key_concepts.html>
* A hyperledger network based on "Build your first network" tutorial is successfully deployed.
  + If you are using Microsoft Windows, you may need to download Virtual Box and Ubuntu OS version 18.04.3 to run the network.
  + <https://hyperledger-fabric.readthedocs.io/en/release-1.1/build_network.html>
* This sample is based on hyperledger fabric version 1.1. Later version may also works with some amendments
* Codebase can be found in [https://github.com/whchengaa/hyperledger-fabric-technical-tutorial](https://github.com/whchengaa/hyperledger-fabric-tutorial) which is based on <https://github.com/hyperledger/fabric-samples/tree/release-1.1>
* The colouring of the code samples below are based on the Visual Studio Code IDE and Xshell terminal
* Do not include **$** in the command

# Deploy a hyperledger network based on balance transfer sample in single machine

## Initial Setup

Clone the Github repository from [https://github.com/whchengaa/hyperledger-fabric-technical-tutorial](https://github.com/whchengaa/hyperledger-fabric-tutorial) by $ git clone https://github.com/whchengaa/hyperledger-fabric-technical-tutorial

Switch to a new branch by $ git checkout -b test-release-1.1 and refer to the instructions of option 2 in hyperledger-fabric-tutorial/balance-transfer/README.md "Sample REST APIs Requests" section can be skipped for this stage.

Open another terminal to run command. Do not close the terminal where the node app is running.

Unless otherwise specified, all terminal commands are run under hyperledger-fabric-tutorial/balance-transfer directory.

## Troubleshooting

After running the $ ./testAPIs.sh -l node, the following error may appear if the machine's processing power is not enough.

|  |
| --- |
| error: [Peer.js]: sendProposal - timed out after:60000 error: [Peer.js]: sendProposal - timed out after:60000 error: [client-utils.js]: sendPeersProposal - Promise is rejected: Error: REQUEST\_TIMEOUT  at Timeout.\_onTimeout (/home/ubuntu/hyperledger-fabric-tutorial/balance-transfer/node\_modules/fabric-client/lib/Peer.js:124:19)  at ontimeout (timers.js:469:11)  at tryOnTimeout (timers.js:304:5)  at Timer.listOnTimeout (timers.js:264:5) error: [client-utils.js]: sendPeersProposal - Promise is rejected: Error: REQUEST\_TIMEOUT  at Timeout.\_onTimeout (/home/ubuntu/hyperledger-fabric-tutorial/balance-transfer/node\_modules/fabric-client/lib/Peer.js:124:19)  at ontimeout (timers.js:469:11)  at tryOnTimeout (timers.js:304:5)  at Timer.listOnTimeout (timers.js:264:5) [2019-06-24 10:48:48.541] [ERROR] instantiate-chaincode - instantiate proposal was bad [2019-06-24 10:48:48.541] [ERROR] instantiate-chaincode - instantiate proposal was bad [2019-06-24 10:48:48.541] [DEBUG] instantiate-chaincode - Failed to send Proposal and receive all good ProposalResponse [2019-06-24 10:48:48.541] [ERROR] instantiate-chaincode - Failed to instantiate. cause:Failed to send Proposal and receive all good ProposalResponse (node:5450) UnhandledPromiseRejectionWarning: Unhandled promise rejection (rejection id: 1): Error: Failed to instantiate. cause:Failed to send Proposal and receive all good ProposalResponse (node:5450) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecated. In the future, promise rejections that are not handled will terminate the Node.js process with a non-zero exit code. |

This error is caused by the lack of time to instantiate chaincode. Simply increase the chaincode instantiation time from 60000 to 120000 in the app/instantiate-chaincode.js

        let results = await channel.sendInstantiateProposal(request, 120000);

Hit Ctrl + C to terminate the command and Re-run $ ./runApp.sh and $ ./testAPIs.sh -l node, there should be no error.

***Remark***: copy and paste hotkey are Ctrl + Shift + C and Ctrl + Shift + V respectively by default.

If there is error for installing npm modules, remove node modules by $ rm -rf node\_modules/ then run $ sudo apt-get install build-essential to install essential packages for Ubuntu system. Then run $ npm install to reinstall npm modules.

***Remark***: Pay special attention to the hyphen (-) in the command. Sometimes hyphens in this doc may become other character which can result in error.

## Monitoring

Run $ docker ps to show the status of each running docker container. Following is the result.

|  |
| --- |
| CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES 7642384cfc67 hyperledger/fabric-peer:x86\_64-1.1.0 "peer node start" 23 seconds ago Up 17 seconds 0.0.0.0:8056->7051/tcp, 0.0.0.0:8058->7053/tcp peer1.org2.example.com 809cd7bd4838 hyperledger/fabric-peer:x86\_64-1.1.0 "peer node start" 23 seconds ago Up 16 seconds 0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp peer0.org1.example.com a7f4d3557f0d hyperledger/fabric-peer:x86\_64-1.1.0 "peer node start" 23 seconds ago Up 15 seconds 0.0.0.0:7056->7051/tcp, 0.0.0.0:7058->7053/tcp peer1.org1.example.com b56b16b3d84b hyperledger/fabric-peer:x86\_64-1.1.0 "peer node start" 23 seconds ago Up 19 seconds 0.0.0.0:8051->7051/tcp, 0.0.0.0:8053->7053/tcp peer0.org2.example.com ea6dca85ffc5 hyperledger/fabric-ca:x86\_64-1.1.0 "sh -c 'fabric-ca-se…" 25 seconds ago Up 20 seconds 0.0.0.0:8054->7054/tcp ca\_peerOrg2 0e3a6e82d529 hyperledger/fabric-ca:x86\_64-1.1.0 "sh -c 'fabric-ca-se…" 25 seconds ago Up 21 seconds 0.0.0.0:7054->7054/tcp ca\_peerOrg1 5cb8ab609550 hyperledger/fabric-orderer:x86\_64-1.1.0 "orderer" 25 seconds ago Up 22 seconds 0.0.0.0:7050->7050/tcp orderer.example.com |

## Wrap-up

Save all changes in IDE and run $ git add –A to add all changes to the staging area and $ git commit –m "your-update-message" to commit the changes to the git repository.

# Add peers in both organizations

## Initial setup

Create two new git branch by $ git checkout -b test-kafka-orderer and $ git checkout -b test-add-peer

Stay in test-add-peer branch.

Make sure bin/cryptogen is located under root directory. If not, run $ curl -sSL https://goo.gl/6wtTN5 | bash -s 1.1.0 1.1.0 0.4.6 under root directory.

Copy the user token for both org to prepare for the next section.

|  |
| --- |
| POST request Enroll on Org1 ...  {"success":true,"secret":"","message":"Jim enrolled Successfully","token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1NjE0Njk2ODMsInVzZXJuYW1lIjoiSmltIiwib3JnTmFtZSI6Ik9yZzEiLCJpYXQiOjE1NjE0MzM2ODN9.DZshXD-4xZftVu19ZwHp\_Q4ey4jFZaUc59Zby9SVcy8"}  ORG1 token is eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1NjE0Njk2ODMsInVzZXJuYW1lIjoiSmltIiwib3JnTmFtZSI6Ik9yZzEiLCJpYXQiOjE1NjE0MzM2ODN9.DZshXD-4xZftVu19ZwHp\_Q4ey4jFZaUc59Zby9SVcy8  POST request Enroll on Org2 ...  {"success":true,"secret":"","message":"Barry enrolled Successfully","token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1NjE0Njk2ODQsInVzZXJuYW1lIjoiQmFycnkiLCJvcmdOYW1lIjoiT3JnMiIsImlhdCI6MTU2MTQzMzY4NH0.5EnIO85RNrP9AWkX2jlSZDXUoCGwXrOMRQZuvKb2nmQ"}  ORG2 token is eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1NjE0Njk2ODQsInVzZXJuYW1lIjoiQmFycnkiLCJvcmdOYW1lIjoiT3JnMiIsImlhdCI6MTU2MTQzMzY4NH0.5EnIO85RNrP9AWkX2jlSZDXUoCGwXrOMRQZuvKb2nmQ |

If you accidentally close the terminal and cannot get back the token, you can register another user.

|  |
| --- |
| $ curl -s -X POST http://localhost:4000/users -H "content-type: application/x-www-form-urlencoded" -d 'username=Tony&orgName=Org1' |

Unless otherwise specified, all terminal commands are run under hyperledger-fabric-tutorial/balance-transfer directory.

## Generate key materials

In artifacts/channel/cryptogen.yaml , change Count of Template of Org1 and Org2 under PeerOrgs from 2 to 3 so the total number of peers can increase from 2 to 3.

Run the following command

$ ../bin/cryptogen extend --input artifacts/channel/crypto-config --config artifacts/channel/cryptogen.yaml

Two new folders are created under

artifacts/channel/crypto-config/peerOrganizations/org1.example.com/peers/peer2.org1.example.com and

artifacts/channel/crypto-config/peerOrganizations/org2.example.com/peers/peer2.org2.example.com

***Remark***: Do not re-generate all key materials which can make the genesis block invalid, unless the new genesis block is re-generated together.

## Edit docker compose file

Add the following code in artifacts/docker-compose.yaml

peer2.org1.example.com:

container\_name: peer2.org1.example.com

extends:

file: base.yaml

service: peer-base

environment:

- CORE\_PEER\_ID=peer2.org1.example.com

- CORE\_PEER\_LOCALMSPID=Org1MSP

- CORE\_PEER\_ADDRESS=peer2.org1.example.com:7051

ports:

- 7061:7051

- 7063:7053

volumes:

- ./channel/crypto-config/peerOrganizations/org1.example.com/peers/peer2.org1.example.com/:/etc/hyperledger/crypto/peer

depends\_on:

- orderer.example.com

peer2.org2.example.com:

container\_name: peer2.org2.example.com

extends:

file: base.yaml

service: peer-base

environment:

- CORE\_PEER\_ID=peer2.org2.example.com

- CORE\_PEER\_LOCALMSPID=Org2MSP

- CORE\_PEER\_ADDRESS=peer2.org2.example.com:7051

ports:

- 8061:7051

- 8063:7053

volumes:

- ./channel/crypto-config/peerOrganizations/org2.example.com/peers/peer2.org2.example.com/:/etc/hyperledger/crypto/peer

depends\_on:

- orderer.example.com

## Edit network configuration file

Open artifacts/network-config.yaml

Add the following code under peers of mychannel of channels

peer2.org1.example.com:

endorsingPeer: false

chaincodeQuery: true

ledgerQuery: true

eventSource: false

peer2.org2.example.com:

endorsingPeer: false

chaincodeQuery: true

ledgerQuery: true

eventSource: false

Add the following code under peers of Org1 of organizations

- peer2.org1.example.com

Add the following code under peers of Org2 of organizations

- peer2.org2.example.com

Add the following code under peers

peer2.org1.example.com:

url: grpcs://localhost:7061

eventUrl: grpcs://localhost:7063

grpcOptions:

ssl-target-name-override: peer2.org1.example.com

tlsCACerts:

path: artifacts/channel/crypto-config/peerOrganizations/org1.example.com/peers/peer2.org1.example.com/tls/ca.crt

peer2.org2.example.com:

url: grpcs://localhost:8061

eventUrl: grpcs://localhost:8063

grpcOptions:

ssl-target-name-override: peer2.org2.example.com

tlsCACerts:

path: artifacts/channel/crypto-config/peerOrganizations/org2.example.com/peers/peer2.org2.example.com/tls/ca.crt

## Deploy the new docker containers

Run the following command in other terminal.

$ docker-compose -f artifacts/docker-compose.yaml up -d peer2.org1.example.com

$ docker-compose -f artifacts/docker-compose.yaml up -d peer2.org2.example.com

Run $ docker ps to check if the two new containers are running.

Run $ docker logs -f 375... where 375... is the corresponding docker container id to check the log output of the docker container to see if there is any error.

## New peers join the channel

Run the following command twice with corresponding peer name, org name and token.

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/channels/mychannel/peers \  -H "authorization: Bearer <put Org1 JSON Web Token here>" \  -H "content-type: application/json" \  -d '{  "peers": ["peer2.org1.example.com"]  }' |

The following error message may be returned. It is related to the node app which can be ignored.

|  |
| --- |
| [2019-06-26 06:40:15.314] [ERROR] Join-Channel - REQUEST\_TIMEOUT:localhost:8053 [2019-06-26 06:40:15.314] [ERROR] Join-Channel - Problem setting up the event hub :Error: EventHub has been shutdown |

## Install chaincode for new peers

Run the following command twice with corresponding peer name, org name and token.

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/chaincodes \  -H "authorization: Bearer <put Org1 JSON Web Token here>" \  -H "content-type: application/json" \  -d "{  \"peers\": [\"peer2.org1.example.com\"],  \"chaincodeName\":\"mycc\",  \"chaincodePath\":\"artifacts/src/github.com/example\_cc/node\",  \"chaincodeType\": \"node\",  \"chaincodeVersion\":\"v0\"  }" |

## Testing

Stop the node app by pressing Ctrl + C on terminal running $ ./runApp.sh and run $ node app to restart the app with latest network configurations.

Run the following command to invoke a transaction to see if it succeeds.

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/channels/mychannel/chaincodes/mycc \  -H "authorization: Bearer <put Org1 JSON Web Token here>" \  -H "content-type: application/json" \  -d '{  "fcn":"move",  "args":["a","b","1"]  }' |

Open 6 terminals to check the logs of all peers docker container to check whether their latest block matches with each other. The following is the successful log output in node app.

|  |
| --- |
| [2019-06-26 07:02:37.361] [INFO] invoke-chaincode - The chaincode invoke chaincode transaction has been committed on peer localhost:8051 [2019-06-26 07:02:37.361] [INFO] invoke-chaincode - Transaction 9dee5ba44f8f9d5de81198c3e6c85b56decff1cdca888aba7fedb9ea7c2def6c has status of VALID in blocl 6 [2019-06-26 07:02:37.361] [INFO] invoke-chaincode - The invoke chaincode transaction was valid. [2019-06-26 07:02:37.362] [DEBUG] invoke-chaincode - ------->>> R E S P O N S E : ["The invoke chaincode transaction was valid.",{"status":"SUCCESS","info":""}] [2019-06-26 07:02:37.362] [INFO] invoke-chaincode - Successfully sent transaction to the orderer. [2019-06-26 07:02:37.362] [DEBUG] invoke-chaincode - Event results for event hub :localhost:8051 [2019-06-26 07:02:37.362] [DEBUG] invoke-chaincode - The invoke chaincode transaction was valid. [2019-06-26 07:02:37.362] [INFO] invoke-chaincode - Successfully invoked the chaincode Org2 to the channel 'mychannel' for transaction ID: 9dee5ba44f8f9d5de81198c3e6c85b56decff1cdca888aba7fedb9ea7c2def6c |

The following is the successful log output in the peer docker container respectively

|  |
| --- |
| 2019-06-26 07:02:37.358 UTC [eventhub\_producer] CreateBlockEvents -> DEBU a7b Channel [mychannel]: Block event for block number [6] contains transaction id: 9dee5ba44f8f9d5de81198c3e6c85b56decff1cdca888aba7fedb9ea7c2def6c |

## Wrap-up

Save all changes in IDE and run $ git add –A to add all changes to the staging area and $ git commit –m "your-update-message" to commit the changes to the git repository.

Branch add-peer contains all the code changes.

# Deploy a kafka-based multi-orderer network in single machine

Kafka is the consensus protocol which allows multiple orderers to function in the network. More information can be found in <https://hyperledger-fabric.readthedocs.io/en/release-1.1/kafka.html>

Switch to test-kafka-orderer git branch by by $ git checkout test-kafka-orderer

Unless otherwise specified, all terminal commands are run under hyperledger-fabric-tutorial/balance-transfer directory.

## Generate key materials

In artifacts/channel/cryptogen.yaml , replace Specs under OrdererOrgs with following code.

Specs:

- Hostname: orderer0

- Hostname: orderer1

Run the following command

$ ../bin/cryptogen extend --input artifacts/channel/crypto-config --config artifacts/channel/cryptogen.yaml

Two new folders are created under

artifacts/channel/crypto-config/ordererOrganizations/example.com/orderers/orderer0.example.com and

artifacts/channel/crypto-config/ordererOrganizations/example.com/orderers/orderer1.example.com

artifacts/channel/crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com can be deleted.

## Edit docker compose file

In artifacts/docker-compose.yaml, replace all orderer.example.com with orderer0.example.com

Add one more docker container orderer1.example.com with port mapping as 7049:7050 according to orderer0.example.com code example.

Under environment of orderer0.example.com and orderer1.example.com, add

- ORDERER\_KAFKA\_VERSION=0.9.0.1

Under depends\_on of 6 peer docker services, add - orderer1.example.com

## Understand docker compose files for kafka and zookeeper

Check out artifacts/docker-compose-kafka.yaml and understand the meaning of minimum in-sync-replica set and default replication factor of the kafka.

## Edit network configuration file

In artifacts/network-config.yaml, replace all orderer.example.com with orderer0.example.com and add orderer1.example.com under orderers of mychannel of channels

Add one more section orderer1.example.com under orderers according to orderer0.example.com code example with different port number, i.e. 7049 under url.

## Edit channel configuration file

In artifacts/channel/configtx.yaml, under Addresses of Orderer section, remove

- orderer.example.com:7050 and add

- orderer0.example.com:7050 and - orderer1.example.com:7050

Replace solo with kafka under OrdererType of Orderer section.

Remove 127.0.0.1:9092 and add the following code under Brokers of Kafka of Orderer section.

- kafka0:9092

- kafka1:9092

- kafka2:9092

- kafka3:9092

## Generate orderer genesis block and channel configuration

Specify the directory containing the configtx.yaml

$ export FABRIC\_CFG\_PATH=artifacts/channel/

Run the following command to generate genesis block.

$ ../bin/configtxgen -profile TwoOrgsOrdererGenesis -outputBlock ./artifacts/channel/genesis.block

Output similar to the following is returned.

|  |
| --- |
| 2019-07-03 14:59:46.209 HKT [common/tools/configtxgen] main -> INFO 001 Loading configuration  2019-07-03 14:59:46.265 HKT [common/tools/configtxgen] doOutputBlock -> INFO 002 Generating genesis block  2019-07-03 14:59:46.266 HKT [common/tools/configtxgen] doOutputBlock -> INFO 003 Writing genesis block |

Run the following command to generate channel configuration.

$ ../bin/configtxgen -profile TwoOrgsChannel

-outputCreateChannelTx ./artifacts/channel/mychannel.tx -channelID mychannel

Output similar to the following is returned.

|  |
| --- |
| 2019-07-08 14:26:15.515 HKT [common/tools/configtxgen] main -> INFO 001 Loading configuration  2019-07-08 14:26:15.570 HKT [common/tools/configtxgen] doOutputChannelCreateTx -> INFO 002 Generating new channel configtx  2019-07-08 14:26:15.697 HKT [common/tools/configtxgen] doOutputChannelCreateTx -> INFO 003 Writing new channel tx |

## Deploy docker-based network components

Remove all running containers (It will also remove other non-hyperledger-fabric docker containers!)

$ docker rm -f `docker ps -aq`

Remove all unused volume

$ docker volume prune -f

Deploy zookeeper cluster.

$ docker-compose -f artifacts/docker-compose-kafka.yaml up -d zookeeper0 zookeeper1 zookeeper2

Deploy kafka cluster.

$ docker-compose -f artifacts/docker-compose-kafka.yaml up -d kafka0 kafka1 kafka2 kafka3

Deploy multiple orderers.

$ docker-compose -f artifacts/docker-compose.yaml up -d orderer0.example.com orderer1.example.com

Deploy peers and ca for Org1.

$ docker-compose -f artifacts/docker-compose.yaml up -d peer0.org1.example.com peer1.org1.example.com ca.org1.example.com

Deploy peers and ca for Org2.

$ docker-compose -f artifacts/docker-compose.yaml up -d peer0.org2.example.com peer1.org2.example.com ca.org2.example.com

## Initialize the network through API

Delete folders containing original fabric-client credentials

$ rm -r fabric-client-kv-org[1-2]

Open another terminal to start the app

$ PORT=4000 node app

The following APIs are packaged into testAPIs.sh which can be executed by one command easily. However, running each API one by one can understand more on the establishment of the fabric network.

Register and enroll new users in Org1 and Org2, then copy the JSON Web Token (jwt) for later use

$ curl -s -X POST http://localhost:4000/users -H "content-type: application/x-www-form-urlencoded" -d 'username=org1TestAcc&orgName=Org1'

$ curl -s -X POST http://localhost:4000/users -H "content-type: application/x-www-form-urlencoded" -d 'username=org2TestAcc&orgName=Org2'

The expiry time of the jwt can be changed in config.json.

Create Channel request

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/channels \  -H "authorization: Bearer <put Org1 or Org2 JSON Web Token here>" \  -H "content-type: application/json" \  -d '{  "channelName":"mychannel",  "channelConfigPath":"../artifacts/channel/mychannel.tx"  }' |

Join Channel request for Org1 peers

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/channels/mychannel/peers \  -H "authorization: Bearer <put Org1 JSON Web Token here>" \  -H "content-type: application/json" \  -d '{  "peers": ["peer0.org1.example.com","peer1.org1.example.com"]  }' |

Join Channel request for Org2 peers

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/channels/mychannel/peers \  -H "authorization: Bearer <put Org2 JSON Web Token here>" \  -H "content-type: application/json" \  -d '{  "peers": ["peer0.org2.example.com","peer1.org2.example.com"]  }' |

Install chaincode for Org1 peers

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/chaincodes \  -H "authorization: Bearer <put Org1 JSON Web Token here>" \  -H "content-type: application/json" \  -d "{  \"peers\": [\"peer0.org1.example.com\",\"peer1.org1.example.com\"],  \"chaincodeName\":\"mycc\",  \"chaincodePath\":\"artifacts/src/github.com/example\_cc/node\",  \"chaincodeType\": \"node\",  \"chaincodeVersion\":\"v0\"  }" |

Install chaincode for Org2 peers

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/chaincodes \  -H "authorization: Bearer <put Org2 JSON Web Token here>" \  -H "content-type: application/json" \  -d "{  \"peers\": [\"peer0.org2.example.com\",\"peer1.org2.example.com\"],  \"chaincodeName\":\"mycc\",  \"chaincodePath\":\"artifacts/src/github.com/example\_cc/node\",  \"chaincodeType\": \"node\",  \"chaincodeVersion\":\"v0\"  }" |

Instantiate chaincode

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/channels/mychannel/chaincodes \  -H "authorization: Bearer <put Org1 or Org2 JSON Web Token here>" \  -H "content-type: application/json" \  -d "{  \"chaincodeName\":\"mycc\",  \"chaincodeVersion\":\"v0\",  \"chaincodeType\": \"node\",  \"args\":[\"a\",\"100\",\"b\",\"200\"]  }" |

***Remark***: More APIs can be found in hyperledger-fabric-tutorial/balance-transfer/README.md "Sample REST APIs Requests" section. There is a transaction to assign anchor peers in version 1.4 which enables different peers to discover each other.

## Testing

Invoke transaction

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/channels/mychannel/chaincodes/mycc \  -H "authorization: Bearer <put Org1 or Org2 JSON Web Token here>" \  -H "content-type: application/json" \  -d '{  "fcn":"move",  "args":["a","b","10"]  }' |

Query transaction by TransactionID returned after invoking transaction

|  |
| --- |
| $ curl -s -X GET http://localhost:4000/channels/mychannel/transactions/<put transaction id here>?peer=peer0.org1.example.com \  -H "authorization: Bearer < put Org1 or Org2 JSON Web Token here>" \  -H "content-type: application/json" |

Query block by block number

|  |
| --- |
| $ curl -s -X GET \  -H "authorization: Bearer < put Org1 or Org2 JSON Web Token here>" \  -H "content-type: application/json" \  "http://localhost:4000/channels/mychannel/blocks/<put block number here, e.g. 0 or 1>?peer=peer0.org1.example.com" |

Every block contains a header like the following.

|  |
| --- |
| {"header":{"number":"1","previous\_hash":"9db29e48906e70b5a61b6bafa1f4c8aa34a02c4722ef8805341ca9b0c6c02e77","data\_hash":"da397ca39f7b03bff451953ee41a89bfc174a3bb41d1b8feabe26001c4b2db11"},......} |

Try to compare the previous hash in block 1 and data hash in block 0 and figure out the reason behind.

The complete list of all APIs can be found in README.md

## Wrap-up

Save all changes in IDE and run $ git add –A to add all changes to the staging area and $ git commit –m "your-update-message" to commit the changes to the git repository.

Branch kafka-orderer contains all the code changes.

# Network backup and recovery

## Initial setup

Create a new git branch by $ git checkout -b test-backup

Network backup is done by using Docker Volume which is a mechanism for persisting data of Docker containers. More information can be found in <https://docs.docker.com/storage/volumes/>

Invoke few requests by using API from section 5.9, copy the transaction id and mark down the execution time for testing purpose later on.

Create a list of folders for storing the backup data under the ~ directory.

|  |
| --- |
| $ cd ~  $ mkdir -p hyperledger\_backup/peer0\_org1 hyperledger\_backup/peer1\_org1 hyperledger\_backup/peer0\_org2 hyperledger\_backup/peer1\_org2 hyperledger\_backup/orderer0 hyperledger\_backup/orderer1 hyperledger\_backup/kafka0 hyperledger\_backup/kafka1 hyperledger\_backup/kafka2 hyperledger\_backup/kafka3 |

Unless otherwise specified, all terminal commands are run under hyperledger-fabric-tutorial/balance-transfer directory.

## Edit docker compose file

In artifacts/docker-compose.yaml, under volumes of orderer0.example.com, add the following code.

- ../../../hyperledger\_backup/orderer0/orderer/:/var/hyperledger/production/orderer

For volumes of orderer1.example.com, the change is similar.

For volumes of peer0.org1.example.com, add the following code.

- ../../../hyperledger\_backup/peer0\_org1/production/:/var/hyperledger/production/

For volumes of peer1.org1.example.com, peer0.org2.example.com and peer2.org1.example.com, the changes are similar.

In artifacts/docker-compose-kafka.yaml, under environment of kafka0, add the following code.

- KAFKA\_LOG.DIRS=/tmp/kafka-logs

Under volumes of kafka0, add the following code.

- ../../../hyperledger\_backup/kafka0/kafka-logs/:/tmp/kafka-logs

For environment and volumes of kafka1, kafka2 and kafka3, the changes are similar.

## Perform data backup

To recover the fabric network, a backup of data from all peers, all orderers and all kafka are required.

The required data for backing-up the peer is under /var/hyperledger/production/ directory inside the peer docker container.

The required data for backing-up the orderer is under /var/hyperledger/production/orderer/ directory inside the orderer docker container.

The required data for backing-up the kafka is under /tmp/kafka-logs/ directory inside the kafka docker container.

Run $ docker exec –it 0b5 bash (where 0b5 is the docker container id) to enter the docker container shell and check the directory listed above to see what files are located there.

The timing to make a data backup is very important. Kafka is a distributed message queue. Each message enters kafka brokers is assigned with an ID also known as offset. The backup can only occur after all kafka brokers remove the expired offsets. No more API request should be sent after kafka brokers remove the expired offset, otherwise it takes 10 min for all kafka brokers to remove another expired offsets.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Send out last API request | ⇒ | All kafka brokers remove expired offsets | ⇒ | Start the backup process |

More details can be found in <https://docs.google.com/document/d/19JihmW-8blTzN99lAubOfseLUZqdrB6sBR0HsRgCAnY/edit>

***WARNING***: There is not guarantee that the network can be recovered with this kafka data backup method. More research is needed for fabric data backup best practice.

Check docker logs for all 4 kafka brokers to see whether expired offsets are removed.

|  |
| --- |
| [2019-07-15 09:43:36,055] INFO [Group Metadata Manager on Broker 0]: Removed 0 expired offsets in 0 milliseconds. (kafka.coordinator.GroupMetadataManager) |

Please make sure the sending time of API requests we sent in the section 5.1 is before the expired offsets removal time.

Copy the required data from the docker containers to the host machine.

|  |
| --- |
| $ docker cp 37a:/var/hyperledger/production/ ../../hyperledger\_backup/peer0\_org1/  $ docker cp 348:/var/hyperledger/production/ ../../hyperledger\_backup/peer1\_org1/  $ docker cp 281:/var/hyperledger/production/ ../../hyperledger\_backup/peer0\_org2/  $ docker cp a1c:/var/hyperledger/production/ ../../hyperledger\_backup/peer1\_org2/  $ docker cp 0b5:/var/hyperledger/production/orderer/ ../../hyperledger\_backup/orderer0/  $ docker cp fe4:/var/hyperledger/production/orderer/ ../../hyperledger\_backup/orderer1/  $ docker cp 4d5:/tmp/kafka-logs ../../hyperledger\_backup/kafka0/  $ docker cp ac1:/tmp/kafka-logs ../../hyperledger\_backup/kafka1/  $ docker cp 320:/tmp/kafka-logs ../../hyperledger\_backup/kafka2/  $ docker cp a2d:/tmp/kafka-logs ../../hyperledger\_backup/kafka3/ |

where the 3-alphanumeric-character after docker cp is the id of the respective Docker container. Alternatively, Docker container name can also be used to specify a container.

## Network recovery

Run the commands in section 5.7. There is no need for peers to join channel or instantiate the chaincode because the whole network has been initialized already.

## Testing

Invoke transaction

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/channels/mychannel/chaincodes/mycc \  -H "authorization: Bearer <put Org1 or Org2 JSON Web Token here>" \  -H "content-type: application/json" \  -d '{  "fcn":"move",  "args":["a","b","10"]  }' |

Query the transactions sent in section 5.1 with the following command

|  |
| --- |
| $ curl -s -X GET http://localhost:4000/channels/mychannel/transactions/<transaction id>?peer=peer0.org1.example.com \  -H "authorization: Bearer <put Org1 or Org2 JSON Web Token here>" \  -H "content-type: application/json" |

If there is any problem for executing above APIs, check the docker logs for orderer to see if there is any error.

Backup at wrong timing can result in the following error for orderers.

|  |
| --- |
| 2019-07-17 03:12:38.201 UTC [orderer/consensus/kafka] try -> DEBU 1c7 [channel: mychannel] Need to retry because process failed = kafka server: The requested offset is outside the range of offsets maintained by the server for the given topic/partition. |

Follow the steps in the previous wrap-up section. Branch backup contains all code changes.

## Further study

Further challenge: Move the backup files to other machine and perform recovery process.

Instead of mounting the host folders as stated above, there is another way to start the network by mounting the named volume. More details can be found in <https://docs.docker.com/compose/compose-file/compose-file-v2/#volume-configuration-reference>

# Deploy CouchDB as state database with rich query functionality

Create a new git branch by $ git checkout -b test-couchdb

Unless otherwise specified, all terminal commands are run under hyperledger-fabric-tutorial/balance-transfer directory.

## Edit docker compose file

Copy the couchdb0 service from hyperledger-fabric-tutorial/first-network/docker-compose-couch.yaml and paste it to bottom of the artifacts/docker-compose.yaml . Admin user, password and networks are not needed at the moment so the environment and networks section can be ignored. Replicate it for 4 times since there are 4 peers. The host port cannot be the same. (The format of port mapping is host-port:container-port)

Copy the following and paste it under environment of peer0.org1.example.com of the artifacts/docker-compose.yaml

- CORE\_LEDGER\_STATE\_STATEDATABASE=CouchDB

- CORE\_LEDGER\_STATE\_COUCHDBCONFIG\_COUCHDBADDRESS=couchdb0:5984

Replicate it for 4 times to all 4 peers and 5984 is the container port which can be kept the same. Change the couchdb0 accordingly.

## Networking concept

To understand the port mapping concept, we need to know the purpose of port. In short, it is a mechanism to distinguish different network services within a machine. For example, port 80 is used for the HTTP service. You can access Google through [http://www.google.com:80](http://www.google.com/) which is the same as <http://www.google.com> .

Each network machine or computer has one public IP only but many internal IPs. For external communication, each port can only be used for one service as there is only one public IP address. For internal communication, each port can be used for multiple times as long as their internal IP addresses are different. The host port refers to the port for external communication while the container port is for internal communication. Try to use telnet to test the connection of a certain port number of an IP address.

Let's take peer0.org2.example.com as an example. Run $ docker inspect peer0.org2.example.com and scroll to the last few rows and the IP address listed there is an internal IP address. 8051 is the host port and 7051 is the respective container port. $ telnet <external IP> 8051 , $ telnet <internal IP> 8051 and $ telnet <internal IP> 7051 give you different results.

## Deploy the new containers

Deploy couchdb containers

$ docker-compose -f artifacts/docker-compose.yaml up -d couchdb0.org1 couchdb1.org1 couchdb0.org2 couchdb1.org2

Remove one peer container then deploy a new peer container. Repeat it for 4 times.

$ docker rm –f 01b

$ docker-compose -f artifacts/docker-compose.yaml up -d peer0.org1.example.com

## Testing

Invoke new request and query old transaction to see if there is any error.

Follow the steps in the previous wrap-up section. Branch couchdb contains all code changes.

## Further Study

Since the original chaincode only allows balance-transfer between A and B, it is not a good example to demonstrate the rich query functionality of CouchDB as the data structure is too simple. Let's deploy a new chaincode which store the information of different marbles with properties of name, color, owner and size.

Install marble chaincode for Org 1

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/chaincodes \  -H "authorization: Bearer <put Org1 JSON Web Token here>" \  -H "content-type: application/json" \  -d "{  \"peers\": [\"peer0.org1.example.com\",\"peer1.org1.example.com\"],  \"chaincodeName\":\"marble\",  \"chaincodePath\":\"../chaincode/marbles02/node\",  \"chaincodeType\": \"node\",  \"chaincodeVersion\":\"v1\"  }" |

Install marble chaincode for Org 2

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/chaincodes \  -H "authorization: Bearer <put Org2 JSON Web Token here>" \  -H "content-type: application/json" \  -d "{  \"peers\": [\"peer0.org2.example.com\",\"peer1.org2.example.com\"],  \"chaincodeName\":\"marble\",  \"chaincodePath\":\"../chaincode/marbles02/node\",  \"chaincodeType\": \"node\",  \"chaincodeVersion\":\"v1\"  }" |

Instantiate chaincode

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/channels/mychannel/chaincodes \  -H "authorization: Bearer <put Org1 or Org2 JSON Web Token here>" \  -H "content-type: application/json" \  -d "{  \"chaincodeName\":\"marble\",  \"chaincodeVersion\":\"v1\",  \"chaincodeType\": \"node\",  \"args\":[]  }" |

## Testing

Invoke 4 new transactions

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/channels/mychannel/chaincodes/marble \  -H "authorization: Bearer <put Org1 or Org2 JSON Web Token here>" \  -H "content-type: application/json" \  -d '{  "fcn":"initMarble",  "args":["marble1","blue","35","tom"]  }' |

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/channels/mychannel/chaincodes/marble \  -H "authorization: Bearer <put Org1 or Org2 JSON Web Token here>" \  -H "content-type: application/json" \  -d '{  "fcn":"initMarble",  "args":["marble2","blue","50","david"]  }' |

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/channels/mychannel/chaincodes/marble \  -H "authorization: Bearer <put Org1 or Org2 JSON Web Token here>" \  -H "content-type: application/json" \  -d '{  "fcn":"initMarble",  "args":["marble3","red","50","owen"]  }' |

|  |
| --- |
| $ curl -s -X POST \  http://localhost:4000/channels/mychannel/chaincodes/marble \  -H "authorization: Bearer <put Org1 or Org2 JSON Web Token here>" \  -H "content-type: application/json" \  -d '{  "fcn":"initMarble",  "args":["marble4","red","12","leo"]  }' |

Visit <http://localhost:4984/_utils/> to access the CouchDB portal. Click "mychannel\_marble" > "Run A Query with Mango"

Input the query with the following format. Key can be \_id, color, name and owner. For example, query all marble with red color.

|  |
| --- |
| {  "selector": {  "key": "value"  }  } |

See whether the results match with the transaction data just invoked.

You may not realize the benefits of CouchDB in small amount of data. Yet, imagine to query tens of thousands of records. Using database with rich query functionality can get the result very efficiently.

# Deploy MongoDB for storage of user credentials

## Initial setup

Create a new git branch by $ git checkout -b test-mongo-kvs

After user registration and enrollment, two new folders fabric-client-kv-org1 and fabric-client-kv-org2 are created under hyperledger-fabric-tutorial/balance-transfer directory for storage of user credentials. To set up a more structured storage, MongoDB with fabric-ca-kvs-mongo npm module can be implemented.

Unless otherwise specified, all terminal commands are run under hyperledger-fabric-tutorial/balance-transfer directory.

Delete folders containing original fabric-client credentials $ rm -r fabric-client-kv-org[1-2]

## Edit client configuration files

Follow the "Config" instructions in <https://www.npmjs.com/package/fabric-sdk-kvs-mongo> to replace the corresponding sections in artifacts/org1.yaml and artifacts/org2.yaml

## Edit application file

In app/helper.js, after let config = '-connection-profile-path' of async function getClientForOrg (userorg, username), add the following line.

    hfc.setConfigSetting('key-value-store', 'fabric-sdk-kvs-mongo');

Run $ npm install fabric-sdk-kvs-mongo --save

## Deploy MongoDB

Run $ docker-compose -f artifacts/docker-compose-mongo.yaml

mongo-express is the front-end of MongoDB.

## Testing

Reload the node app

Register and enroll new users in Org1 and Org2

$ curl -s -X POST http://localhost:4000/users -H "content-type: application/x-www-form-urlencoded" -d 'username=org1TestAcc&orgName=Org1'

$ curl -s -X POST http://localhost:4000/users -H "content-type: application/x-www-form-urlencoded" -d 'username=org2TestAcc&orgName=Org2'

List out the folders under hyperledger-fabric-tutorial/balance-transfer directory to see if fabric-client-kv-org1 and fabric-client-kv-org2 still exist or not.

Visit MongoDB front-end via <http://localhost:8081> to check the user credential data.

Follow the steps in the previous wrap-up section. Branch mongo-kvs contains all code changes.

# Deploy Hyperledger Explorer

## Initial Setup

Clone the Github repository from <https://github.com/hyperledger/blockchain-explorer> by $ git clone https://github.com/hyperledger/blockchain-explorer.git

Switch to branch release-3.5 by $ git checkout release-3.5 and refer to README.md. For section "Build Hyperledger Explorer", you *may* ignore errors after running $ npm run test and $ npm test -- -u --coverage

## Testing

Invoke new transactions for both organizations to see if the records are correctly shown in the Explorer.

## Remark

Explorer cannot read the private keys stored in the MongoDB currently. You may consider to convert those keys to traditional format and store it in a folder.

This are several bugs for this version. Latest version of explorer can't fit with old version of Fabric.

First, the explorer stop updating after the fabric network idles for some period of time.

Second, if the explorer goes offline and at the same time there is new update for the fabric network, the explorer will not sync back the missing blocks and transactions.

# Deploy a cross-machine kafka-based multi-orderer network with Docker Swarm

## Initial setup

Create two new git branches by $ git checkout -b test-cross-machine-hosts-file and $ git checkout -b test-cross-machine-swarm Stay on test-cross-machine-swarm branch. Unless otherwise specified, all terminal commands are run under hyperledger-fabric-tutorial/balance-transfer directory.

Docker Swarm is a cluster management tool to enable communication of different containers across different machines. An overlay network is a connection layer on top of the Docker Swarm. More information can be found in <https://docs.docker.com/engine/swarm/> and <https://docs.docker.com/network/overlay/>

## Network topology

The following example requires three machines or cloud instances. Let's name them as Test1, Test2 and Test3. The deployment details of network component is listed below.

|  |  |  |
| --- | --- | --- |
| Test1 | Test2 | Test3 |
| node app  orderer0.example.com  peer0.org1.example.com peer1.org1.example.com ca.org1.example.com couchdb0.org1  couchdb1.org1  mongo  mongo-express | zookeeper2  kafka2  kafka3  peer0.org2.example.com peer1.org2.example.com ca.org2.example.com couchdb0.org2  couchdb1.org2 | zookeeper0  zookeeper1  kafka0  kafka1  orderer1.example.com |

The ports below need to be exposed in the firewall settings for communication across machines.

|  |  |
| --- | --- |
| Ports | Purpose |
| TCP 2377 | Docker cluster management communications |
| TCP and UDP 7946 | communication among Docker nodes |
| UDP 4789 | Docker overlay network traffic |
| TCP 7051, 7053, 7056, 7058, 8051, 8053, 8056, 8058 | Peer |
| TCP 7049, 7050 | Orderer |
| TCP 7054, 8054 | CA |
| TCP 4000 | Node app |
| TCP 27017, 8081 | MongoDB, Mongo Express |

***WARNING:*** MongoDB and Mongo Express cannot be exposed to all external IP in production environment.

## Edit Docker compose file

In artifacts/base.yaml, add the following code above services

networks:

fabric\_net:

external: true

Add the following code after all docker services

networks:

- fabric\_net

Change the CORE\_VM\_DOCKER\_HOSTCONFIG\_NETWORKMODE from artifacts\_default to fabric\_net

In artifacts/docker-compose.yaml, add the following code above services to enable overlay network.

networks:

fabric\_net:

external: true

Add the following code after all docker services except peer

networks:

- fabric\_net

Remove the following code for all peers

depends\_on:

- orderer0.example.com

- orderer1.example.com

In artifacts/docker-compose-kafka.yaml, add the following code above services

networks:

fabric\_net:

external: true

Remove the following code for all kafka

depends\_on:

- zookeeper0

- zookeeper1

- zookeeper2

## Edit network configuration file

Since Docker containers are deployed in different machines, their URLs need to be updated. In artifacts/network-config.yaml, apart from the Docker containers in Test1, url and eventUrl of other Docker containers should change from localhost to respective IP address. Port number remains unchanged.

## Clone the repository to three machines

Follow the steps in the previous wrap-up section. Use your own Github account to create a new repo and run the following.

Remove original Github repo

$ git remote rm origin

Add your newly created repo

$ git remote add origin <your github repo link>

Push the code to the repo

$ git push -u origin test-cross-machine-swarm

Clone the code to three machines.

## Setup Docker Swarm and overlay network

In Test1, run $ docker swarm init to initialize the Swarm network as a manager. It returns something similar to the following result.

$ docker swarm join --token SWMTKN-1-4bg8u1sjnk6sjegcdba5f03ij5yespoqn1g3qhji7hn213qyw0-ee8h9oinejen29d1t9r6pw4pf 172.0.0.71:2377

Run the above command in Test2 and Test3 to join the Swarm network. Following is the success response. This node joined a swarm as a worker.

Run $ docker node ls in Test1 to list out Swarm network nodes. You should see three nodes.

In Test1, run $ docker network create --attachable --driver overlay fabric\_net to create overlay network named fabric\_net.

Run $ docker network ls in all three machines. Only Test1 shows fabric\_net because the overlay network can only be extended to other nodes by bring up a docker container joining that overlay network.

In Test2 and Test3, run $ docker run -itd --name mybusybox --network fabric\_net busybox to create a busybox docker container joining fabric\_net. In fact, whether it is busybox doesn't matter, bringing up any container joining fabric\_net can have same effect.

Run $ docker network ls again in Test2 and Test3 to see if the fabric\_net exists or not.

If there is any error, $ sudo systemctl restart docker in all three machines to restart Docker engine or run $ docker swarm leave --force in all three machines to leave the Swarm network and repeat the above process again.

## Network components deployment

For all machines,

Remove all existing containers. (Make sure there is no other important containers running)

$ docker rm -f `docker ps -aq`

Clear unused Docker volume

$ docker volume prune –f

Remove cached Docker chaincode images

$ docker rmi -f $(docker images | grep "dev\|none\|test-vp\|peer[0-9]-" | awk '{print $3}')

Remove original backup data

$ sudo rm -r ../../hyperledger\_backup

For Test3,

$ docker-compose -f artifacts/docker-compose-kafka.yaml up -d zookeeper0 zookeeper1

$ docker-compose -f artifacts/docker-compose-kafka.yaml up -d kafka0 kafka1

$ docker-compose -f artifacts/docker-compose.yaml up -d orderer1.example.com

For Test2,

$ docker-compose -f artifacts/docker-compose-kafka.yaml up -d zookeeper2

$ docker-compose -f artifacts/docker-compose-kafka.yaml up -d kafka2 kafka3

$ docker-compose -f artifacts/docker-compose.yaml up -d peer0.org2.example.com peer1.org2.example.com ca.org2.example.com couchdb0.org2 couchdb1.org2

For Test1,

$ docker-compose -f artifacts/docker-compose.yaml up -d orderer0.example.com

$ docker-compose -f artifacts/docker-compose.yaml up -d peer0.org1.example.com peer1.org1.example.com ca.org1.example.com couchdb0.org1 couchdb1.org1

$ docker-compose -f artifacts/docker-compose-mongo.yaml up -d

$ npm install

$ node app

For the execution sequence, zookeeper should be deployed first, then followed by kafka and the rest.

## Testing

Open a new terminal in Test1 and run $ ./testAPIs.sh -l node

Branch cross-machine-swarm contains all code changes.

# Deploy a cross-machine kafka-based multi-orderer network with hosts file

## Initial Setup

Switch to test-cross-machine-hosts-file git branch by by $ git checkout test-cross-machine-hosts-file

Hosts file is a system network configuration file to define IP for different domain names. Run

$ vi /etc/hosts to check the hosts file in your system.

Follow the network topology in section 9.2 and

Unless otherwise specified, all terminal commands are run under hyperledger-fabric-tutorial/balance-transfer directory.

## Network topology

Follow the section 10.2 for the first table.

The ports below need to be exposed in the firewall settings for communication across machines.

|  |  |
| --- | --- |
| Ports | Purpose |
| TCP 7051, 7053, 7056, 7058, 8051, 8053, 8056, 8058 | Peer |
| TCP 7049, 7050 | Orderer |
| TCP 7054, 8054 | CA |
| TCP 2181-2183, 2888-2890, 3888-3890 | Zookeeper |
| TCP 9092, 10092, 11092, 12092 | Kafka |
| TCP 4000 | Node app |
| TCP 27017, 8081 | MongoDB, Mongo Express |

***WARNING:*** MongoDB and Mongo Express cannot be exposed to all external IP in production environment.

## Create hosts files

Since each Docker container is a virtual machine, there is a hosts file for each of them. A tailor-made hosts file is needed for each container. Let's create 6 hosts files under artifacts with following filenames, i.e. hosts-kafka01, hosts-kafka23, hosts-orderer0, hosts-orderer1, hosts-org1 and hosts-org2. You can also run

$ touch artifacts/hosts-{kafka01,kafka23,orderer0,orderer1,org1,org2} to create files.

Peer needs to connect with other peers and orderers. Orderer needs to connect with kafka-cluster. Kafka needs to connect with zookeeper-cluster. Zookeeper needs to connect with other zookeeper so its hosts file is duplicated with kafka one under our network topology. There is no need to specify the domain name in hosts file if the container is located within the same machine. Follow this logic to edit all hosts file.

## Edit Docker compose file

In artifacts/base.yaml, remove entire ports section for kafka and zookeeper.

In artifacts/docker-compose.yaml, remove the following code for all peers

depends\_on:

- orderer0.example.com

- orderer1.example.com

Add ./hosts-orderer1:/etc/hosts under volumes for all orderers and peers where orderer1 is the name of the respective container.

In artifacts/docker-compose-kafka.yaml, add the following code for zookeeper0, zookeeper1 and zookeeper3 respectively.

ports:

- 2181:2181

- 2888:2888

- 3888:3888

volumes:

- ./hosts-kafka01:/etc/hosts

ports:

- 2182:2181

- 2889:2888

- 3889:3888

volumes:

- ./hosts-kafka01:/etc/hosts

ports:

- 2183:2181

- 2890:2888

- 3890:3888

volumes:

- ./hosts-kafka23:/etc/hosts

Change the zookeeper port number to the respective hosts port number for ZOO\_SERVERS under environment of all zookeepers.

For all kafka brokers, follow zookeeper examples to add the similar code in ports and volumes. Apart from that, add the following code under environment, change the port number and IP according to different kafka.

- KAFKA\_ADVERTISED\_PORT=9092

- KAFKA\_ADVERTISED\_HOST\_NAME=192.168.1.3

Change the zookeeper port number to the respective hosts port number for KAFKA\_ZOOKEEPER\_CONNECT under environment of all kafka brokers.

Remove the following code for all kafka brokers

depends\_on:

- zookeeper0

- zookeeper1

- zookeeper2

## Edit network configuration file

Follow section 10.4

## Network components deployment and testing

Follow sections 10.6 and 10.7.

Follow the steps in the previous wrap-up section. Branch cross-machine-hosts-file contains all code changes.

## Alternatives

Rather than editing the hosts file, adding the following code with respective IP address to each container can also work.

extra\_hosts:

- "peer0.org2.example.com:192.168.1.2"

- "peer1.org2.example.com:192.168.1.2"

## Potential Problems

Since the host port numbers of all 4 kafka brokers stated in the genesis block are 9092, if kafka0, the kafka broker that expose port 9092, goes down and orderer restarts, the network cannot work properly.

# Remarks on high-availability of the Fabric network

The default application is not designed for high-availability purpose. It leads to several single points of failure.

If peer0 of either organization goes down, invoking and query transaction results in failure. Specifying the peer in the request is a workaround solution but it makes no sense for the system to find out which peer is running before specifying the designated peer.

If only orderer0 goes down, invoking transaction results in failure.

For kafka's crash fault tolerance, it is not a very reliable mechanism. By default, there are total of 4 kafka brokers and minimum in-sync-replica (ISR) and default replication factor (DRF) are set to 2 and 3 respectively. It can only allow maximum of 1 kafka broker to be out of service. If there are only 2 running kafka brokers, the network may not receive new invoke request. Since the DRF is set to 3, each kafka broker only connects to 2 more brokers. There is a high chance that the 2 stopped brokers originally form the ISR with one of the remaining running broker which means broker cannot maintain min ISR and it cannot function properly anymore.

Can setting DRF to 4 solve the problem? No, as channel creation depends on number of running broker greater than equal to DRF. No more channel can be created if less than 4 brokers are running which means there is no more crash fault tolerance. However, setting ISR to 1 maybe a better solution but more research is required.

For zookeeper, all network operations can be performed as usual if minimum two zookeepers are working out of total of three. After kafka cluster connected with each other, most network operations except channel creation can proceed even the entire zookeeper cluster goes out of service. However, all zookeepers have to function properly to enable the formation of a new kafka cluster or on-board new kafka brokers into existing kafka cluster.

# Further study

* Endorsement Policy
* Fabric SDK
* Chaincode design
* Chaincode upgrade
* Network component migration