**BYFN 소스 분석**

1. **./**byfn**.**sh generate

**./**byfn**.**sh generate 명령어를 사용할 경우 3가지의 함수를 사용하게 됩니다.

elif [ "${MODE}" == "generate" ]; then ## Generate Artifacts

generateCerts

replacePrivateKey

generateChannelArtifacts

**generateCerts() 함수를 통해 MSP를 생성합니다.**

function generateCerts() {

which cryptogen

if [ "$?" -ne 0 ]; then

echo "cryptogen tool not found. exiting"

exit 1

fi

// Fabric이 설치 되었는지 확인 후 설치되어 있지 않으면 종료 시킵니다.

echo

echo "##########################################################"

echo "##### Generate certificates using cryptogen tool #########"

echo "##########################################################"

if [ -d "crypto-config" ]; then

rm -Rf crypto-config

fi

// “crypto-config” 폴더가 존재하면 삭제시킵니다.

// “crypto-config” 폴더는 “crytogen generate” 명령어 입력시 MSP가 생성되는 폴더입니다.

set -x

cryptogen generate --config=./crypto-config.yaml

// “crypto-config”명령어를 통해 MSP를 생성합니다.

// MSP설정시 참조하는 설정은 “cryto-config.yaml” 파일 입니다.

res=$?

set +x

if [ $res -ne 0 ]; then

echo "Failed to generate certificates..."

exit 1

fi

// MSP생성에 실패하면 종료 시킵니다.

echo

}

**Cryto-config.yaml 설정.**

OrdererOrgs:

// OrdererOrgs의 MSP 정보를 정보 합니다.

- Name: Orderer

Domain: example.com

Specs:

- Hostname: orderer

- Hostname: orderer2

- Hostname: orderer3

- Hostname: orderer4

- Hostname: orderer5

PeerOrgs:

// OrdererOrgs의 MSP 정보를 정의 합니다.

- Name: Org1

Domain: org1.example.com

EnableNodeOUs: true

Template:

Count: 2

Users:

Count: 1

- Name: Org2

Domain: org2.example.com

EnableNodeOUs: true

// OU의 사용여부를 나타냅니다.

// OU는 Organization Unit으로 인증서 내부적으로 조직을 나눌떄 사용합니다.

Template:

Count: 2 // 생성할 Peer의 숫자를 나타냅니다.

Users:

Count: 1 // 생성할 User의 숫자를 나타냅니다.

**replacePrivateKey() docker-Compose 파일의 Privatekey이름을 바꿔줍니다.**

function replacePrivateKey() {

ARCH=$(uname -s | grep Darwin)

if [ "$ARCH" == "Darwin" ]; then

OPTS="-it"

else

OPTS="-i"

fi

cp docker-compose-e2e-template.yaml docker-compose-e2e.yaml

//기본 docker-compose 파일을 복사합니다.

// ./byfn up명령으로 네트워크를 시작할 떄 docker-come-e2e.yaml 파일을 이용합니다.

CURRENT\_DIR=$PWD

cd crypto-config/peerOrganizations/org1.example.com/ca/

PRIV\_KEY=$(ls \*\_sk)

cd "$CURRENT\_DIR"

sed $OPTS "s/CA1\_PRIVATE\_KEY/${PRIV\_KEY}/g" docker-compose-e2e.yaml

cd crypto-config/peerOrganizations/org2.example.com/ca/

PRIV\_KEY=$(ls \*\_sk)

cd "$CURRENT\_DIR"

sed $OPTS "s/CA2\_PRIVATE\_KEY/${PRIV\_KEY}/g" docker-compose-e2e.yaml

//생성된 MSP의 Privatekey파일명을 가져와서 docker-compose-e2e.yaml의 PrivateKey명을 바꿔줍니다.

if [ "$ARCH" == "Darwin" ]; then

rm docker-compose-e2e.yamlt

fi

}

**generateChannelArtifacts() 명령어를 통해 ChannelArtifact를 생성합니다.**

function generateChannelArtifacts() {

which configtxgen

if [ "$?" -ne 0 ]; then

echo "configtxgen tool not found. exiting"

exit 1

fi

echo "##########################################################"

echo "######### Generating Orderer Genesis block ##############"

echo "##########################################################"

# Note: For some unknown reason (at least for now) the block file can't be

# named orderer.genesis.block or the orderer will fail to launch!

echo "CONSENSUS\_TYPE="$CONSENSUS\_TYPE

set -x

if [ "$CONSENSUS\_TYPE" == "solo" ]; then

configtxgen -profile TwoOrgsOrdererGenesis -channelID byfn-sys-channel -outputBlock ./channel-artifacts/genesis.block

elif [ "$CONSENSUS\_TYPE" == "kafka" ]; then

configtxgen -profile SampleDevModeKafka -channelID byfn-sys-channel -outputBlock ./channel-artifacts/genesis.block

elif [ "$CONSENSUS\_TYPE" == "etcdraft" ]; then

configtxgen -profile SampleMultiNodeEtcdRaft -channelID byfn-sys-channel -outputBlock ./channel-artifacts/genesis.block

else

set +x

echo "unrecognized CONSESUS\_TYPE='$CONSENSUS\_TYPE'. exiting"

exit 1

fi

res=$?

set +x

if [ $res -ne 0 ]; then

echo "Failed to generate orderer genesis block..."

exit 1

fi

echo

echo "#################################################################"

echo "### Generating channel configuration transaction 'channel.tx' ###"

echo "#################################################################"

set -x

configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID $CHANNEL\_NAME

//configtxgen 명령어를 사용하면 cryto-config.yaml 파일을 참고하여 ChannelAritafact를 생성합니다.

//profile에 있는 TwoOrgsChannel 설정을 사용하여 /channel-artifacts폴더에 channel.tx 을 만듭니다.

res=$?

set +x

if [ $res -ne 0 ]; then

echo "Failed to generate channel configuration transaction..."

exit 1

fi

echo

echo "#################################################################"

echo "####### Generating anchor peer update for Org1MSP ##########"

echo "#################################################################"

set -x

configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID $CHANNEL\_NAME -asOrg Org1MSP

//profile에 있는 TwoOrgsChannel 설정을 사용하여 /channel-artifacts폴더에 Org1MSPanchors.tx를 생성합니다.

res=$?

set +x

if [ $res -ne 0 ]; then

echo "Failed to generate anchor peer update for Org1MSP..."

exit 1

fi

echo

echo "#################################################################"

echo "####### Generating anchor peer update for Org2MSP ##########"

echo "#################################################################"

set -x

configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate \

./channel-artifacts/Org2MSPanchors.tx -channelID $CHANNEL\_NAME -asOrg Org2MSP

//profile에 있는 TwoOrgsChannel 설정을 사용하여 /channel-artifacts폴더에 Org2SPanchors.tx를 생성합니다.

res=$?

set +x

if [ $res -ne 0 ]; then

echo "Failed to generate anchor peer update for Org2MSP..."

exit 1

fi

echo

}

**Configtx.yaml**

TwoOrgsChannel:

Consortium: SampleConsortium

<<: \*ChannelDefaults

Application:

<<: \*ApplicationDefaults

Organizations:

- \*Org1

- \*Org2

Capabilities:

<<: \*ApplicationCapabilities

Channel: &ChannelDefaults

Policies:

Readers:

Type: ImplicitMeta

Rule: "ANY Readers"

Writers:

Type: ImplicitMeta

Rule: "ANY Writers"

Admins:

Type: ImplicitMeta

Rule: "MAJORITY Admins"

Capabilities:

<<: \*ChannelCapabilities

Channel: &ChannelCapabilities

V1\_3: true

Application: &ApplicationDefaults

Organizations:

Policies:

Readers:

Type: ImplicitMeta

Rule: "ANY Readers"

Writers:

Type: ImplicitMeta

Rule: "ANY Writers"

Admins:

Type: ImplicitMeta

Rule: "MAJORITY Admins"

LifecycleEndorsement:

Type: ImplicitMeta

Rule: "MAJORITY Endorsement"

Endorsement:

Type: ImplicitMeta

Rule: "MAJORITY Endorsement"

Capabilities:

<<: \*ApplicationCapabilities

**./byfn generate 결과화면**



**생성된 MSP**



**생성된 Channel-artifacts**



1. ./byfn.sh up

**./**byfn**.**sh generate 명령어를 사용할 경우 networkUp() 함수를 사용하게 됩니다.

if [ "${MODE}" == "up" ]; then

networkUp

**networkUp() 함수를 통해 MSP가 존재하지 않을 시 MSP를 생성하고 Fabric 컨테이너를 생성합니다.**

function networkUp() {

checkPrereqs

# generate artifacts if they don't exist

if [ ! -d "crypto-config" ]; then

generateCerts

replacePrivateKey

generateChannelArtifacts

fi

//MSP폴더인 cryto-config가 존재 하지 않는다면 ./byfn.sh generate 명령어를 사용 했을때와 같이 위의 3개의 함수를 실행합니다.

COMPOSE\_FILES="-f ${COMPOSE\_FILE}"

//491번째 줄의 설정에 따라 compose 파일에 대해 옵션을 설정하지 않으면 docker-compose-cli.yaml파일을 default로 사용합니다.

if [ "${CERTIFICATE\_AUTHORITIES}" == "true" ]; then

COMPOSE\_FILES="${COMPOSE\_FILES} -f ${COMPOSE\_FILE\_CA}"

//501번째 줄의 설정에 따라 COPOSE\_FILE\_CA는 docker-compose-ca.yaml파일을 을 사용합니다.

export BYFN\_CA1\_PRIVATE\_KEY=$(cd crypto-config/peerOrganizations/org1.example.com/ca && ls \*\_sk)

//org1의 Private\_key 이름을 저장합니다.

export BYFN\_CA2\_PRIVATE\_KEY=$(cd crypto-config/peerOrganizations/org2.example.com/ca && ls \*\_sk)

//org2의 Private\_key 이름을 저장합니다.

fi

if [ "${CONSENSUS\_TYPE}" == "kafka" ]; then

COMPOSE\_FILES="${COMPOSE\_FILES} -f ${COMPOSE\_FILE\_KAFKA}"

elif [ "${CONSENSUS\_TYPE}" == "etcdraft" ]; then

COMPOSE\_FILES="${COMPOSE\_FILES} -f ${COMPOSE\_FILE\_RAFT2}"

fi

if [ "${IF\_COUCHDB}" == "couchdb" ]; then

COMPOSE\_FILES="${COMPOSE\_FILES} -f ${COMPOSE\_FILE\_COUCH}"

fi

IMAGE\_TAG=$IMAGETAG docker-compose ${COMPOSE\_FILES} up -d 2>&1

//IMAGETAF는 506번째 줄의 설정에 따라 latest를 사용합니다.

//compose-cli.yaml , docker-compose-ca.yam 설정에 따른 컨테너를 생성합니다.

// -d 옵션에 따라 백그라운드로 실행시키며 2>&1 설정에 따라 표준에러를 표준 출력합니다.

docker ps –a

// 구동중인 컨테이너 정보를 표시합니다.

if [ $? -ne 0 ]; then

echo "ERROR !!!! Unable to start network"

exit 1

fi

// 구동중인 컨테이너가 없으면 종료 합니다.

if [ "$CONSENSUS\_TYPE" == "kafka" ]; then

sleep 1

echo "Sleeping 10s to allow $CONSENSUS\_TYPE cluster to complete booting"

sleep 9

fi

if [ "$CONSENSUS\_TYPE" == "etcdraft" ]; then

sleep 1

echo "Sleeping 15s to allow $CONSENSUS\_TYPE cluster to complete booting"

sleep 14

fi

# now run the end to end script

docker exec cli scripts/script.sh $CHANNEL\_NAME $CLI\_DELAY $LANGUAGE $CLI\_TIMEOUT $VERBOSE $NO\_CHAINCODE

if [ $? -ne 0 ]; then

echo "ERROR !!!! Test failed"

exit 1

fi

}

**docker-compose-cli.yaml**

version: '2'

// docker-compose의 버전을 의미합니다.

volumes:

orderer.example.com:

peer0.org1.example.com:

peer1.org1.example.com:

peer0.org2.example.com:

peer1.org2.example.com:

// volumes 설정을 통해 호스트와 컨테이너의 폴더를 공유합니다.

networks:

byfn:

// 컨테이너끼리 통신 할 수 있는 네트워크를 생성합니다.

// byfn 네트워크를 사용하는 컨테이너들은 서로 통신할 수 있습니다.

services:

orderer.example.com:

extends:

file: base/docker-compose-base.yaml

//base/docker-compose-base.yaml을 참조하여 컨테이너를 생성합니다.

service: orderer.example.com

//base/docker-compose-base.yaml에서 참조하는 service 입니다.

container\_name: orderer.example.com

networks:

- byfn

peer0.org1.example.com:

container\_name: peer0.org1.example.com

extends:

file: base/docker-compose-base.yaml

service: peer0.org1.example.com

networks:

- byfn

peer1.org1.example.com:

container\_name: peer1.org1.example.com

extends:

file: base/docker-compose-base.yaml

service: peer1.org1.example.com

networks:

- byfn

peer0.org2.example.com:

container\_name: peer0.org2.example.com

extends:

file: base/docker-compose-base.yaml

service: peer0.org2.example.com

networks:

- byfn

peer1.org2.example.com:

container\_name: peer1.org2.example.com

extends:

file: base/docker-compose-base.yaml

service: peer1.org2.example.com

networks:

- byfn

cli:

container\_name: cli

image: hyperledger/fabric-tools:$IMAGE\_TAG

// 컨테이너에서 사용할 이미지를 지정합니다.

//IMAGE\_TAG는 .env 파일에 설정되어있는 latest를 사용합니다.

tty: true

stdin\_open: true

environment:

- GOPATH=/opt/gopath

- CORE\_VM\_ENDPOINT=unix:///host/var/run/docker.sock

#- FABRIC\_LOGGING\_SPEC=DEBUG

- FABRIC\_LOGGING\_SPEC=INFO

//표시되는 로그 레벨을 설정합니다.

- CORE\_PEER\_ID=cli

- CORE\_PEER\_ADDRESS=peer0.org1.example.com:7051

- CORE\_PEER\_LOCALMSPID=Org1MSP

//LOCALMSP

- CORE\_PEER\_TLS\_ENABLED=true

//TLS사용 여부를 설정 합니다.

- CORE\_PEER\_TLS\_CERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt

//TLS인증서 파일의 위치를 설정 합니다.

- CORE\_PEER\_TLS\_KEY\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key

//TLS 개인키 파일의 위치를 설정 합니다.

- 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

//TLS 인증서를 발급한 CA의 인증서 파일 위치를 설정 합니다.

- CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

//peer의 MPS 위치를 설정 합니다.

working\_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer

// command가 실행되는 위치를 설정 합니다.

command: /bin/bash

// 컨테이너 생성시 실행되는 명령어 입니다.

volumes:

- /var/run/:/host/var/run/

- ./../chaincode/:/opt/gopath/src/github.com/hyperledger/fabric-samples/chaincode

- ./crypto-config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/

- ./scripts:/opt/gopath/src/github.com/hyperledger/fabric/peer/scripts/

- ./channel-artifacts:/opt/gopath/src/github.com/hyperledger/fabric/peer/channel-artifacts

// 호스트와 컨테이너의 폴더를 공유합니다

//ex):: 호스트 위치:컨테이너 위치

depends\_on:

- orderer.example.com

- peer0.org1.example.com

- peer1.org1.example.com

- peer0.org2.example.com

- peer1.org2.example.com

networks:

- byfn

**docker-compose-base.yaml**

version: '2'

services:

orderer.example.com:

container\_name: orderer.example.com

extends:

file: peer-base.yaml

service: orderer-base

volumes:

- ../channel-artifacts/genesis.block:/var/hyperledger/orderer/orderer.genesis.block

- ../crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/msp:/var/hyperledger/orderer/msp

- ../crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/:/var/hyperledger/orderer/tls

- orderer.example.com:/var/hyperledger/production/orderer

ports:

- 7050:7050

//호스트 포트와 컨테이너의 포트를 포워딩 합니다.

//ex):: 호스트 포트:컨테이너 포트

peer0.org1.example.com:

container\_name: peer0.org1.example.com

extends:

file: peer-base.yaml

service: peer-base

environment:

- CORE\_PEER\_ID=peer0.org1.example.com

- CORE\_PEER\_ADDRESS=peer0.org1.example.com:7051

- CORE\_PEER\_LISTENADDRESS=0.0.0.0:7051

- CORE\_PEER\_CHAINCODEADDRESS=peer0.org1.example.com:7052

- CORE\_PEER\_CHAINCODELISTENADDRESS=0.0.0.0:7052

- CORE\_PEER\_GOSSIP\_BOOTSTRAP=peer1.org1.example.com:8051

- CORE\_PEER\_GOSSIP\_EXTERNALENDPOINT=peer0.org1.example.com:7051

- CORE\_PEER\_LOCALMSPID=Org1MSP

//0.0.0.0은 컨테이너의 로컬 주소를 나타냅니다.

volumes:

- /var/run/:/host/var/run/

- ../crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp:/etc/hyperledger/fabric/msp

- ../crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls:/etc/hyperledger/fabric/tls

- peer0.org1.example.com:/var/hyperledger/production

ports:

- 7051:7051

peer1.org1.example.com:

container\_name: peer1.org1.example.com

extends:

file: peer-base.yaml

service: peer-base

environment:

- CORE\_PEER\_ID=peer1.org1.example.com

- CORE\_PEER\_ADDRESS=peer1.org1.example.com:8051

- CORE\_PEER\_LISTENADDRESS=0.0.0.0:8051

- CORE\_PEER\_CHAINCODEADDRESS=peer1.org1.example.com:8052

- CORE\_PEER\_CHAINCODELISTENADDRESS=0.0.0.0:8052

- CORE\_PEER\_GOSSIP\_EXTERNALENDPOINT=peer1.org1.example.com:8051

- CORE\_PEER\_GOSSIP\_BOOTSTRAP=peer0.org1.example.com:7051

- CORE\_PEER\_LOCALMSPID=Org1MSP

volumes:

- /var/run/:/host/var/run/

- ../crypto-config/peerOrganizations/org1.example.com/peers/peer1.org1.example.com/msp:/etc/hyperledger/fabric/msp

- ../crypto-config/peerOrganizations/org1.example.com/peers/peer1.org1.example.com/tls:/etc/hyperledger/fabric/tls

- peer1.org1.example.com:/var/hyperledger/production

ports:

- 8051:8051

peer0.org2.example.com:

container\_name: peer0.org2.example.com

extends:

file: peer-base.yaml

service: peer-base

environment:

- CORE\_PEER\_ID=peer0.org2.example.com

- CORE\_PEER\_ADDRESS=peer0.org2.example.com:9051

- CORE\_PEER\_LISTENADDRESS=0.0.0.0:9051

- CORE\_PEER\_CHAINCODEADDRESS=peer0.org2.example.com:9052

- CORE\_PEER\_CHAINCODELISTENADDRESS=0.0.0.0:9052

- CORE\_PEER\_GOSSIP\_EXTERNALENDPOINT=peer0.org2.example.com:9051

- CORE\_PEER\_GOSSIP\_BOOTSTRAP=peer1.org2.example.com:10051

- CORE\_PEER\_LOCALMSPID=Org2MSP

volumes:

- /var/run/:/host/var/run/

- ../crypto-config/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/msp:/etc/hyperledger/fabric/msp

- ../crypto-config/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls:/etc/hyperledger/fabric/tls

- peer0.org2.example.com:/var/hyperledger/production

ports:

- 9051:9051

peer1.org2.example.com:

container\_name: peer1.org2.example.com

extends:

file: peer-base.yaml

service: peer-base

environment:

- CORE\_PEER\_ID=peer1.org2.example.com

- CORE\_PEER\_ADDRESS=peer1.org2.example.com:10051

- CORE\_PEER\_LISTENADDRESS=0.0.0.0:10051

- CORE\_PEER\_CHAINCODEADDRESS=peer1.org2.example.com:10052

- CORE\_PEER\_CHAINCODELISTENADDRESS=0.0.0.0:10052

- CORE\_PEER\_GOSSIP\_EXTERNALENDPOINT=peer1.org2.example.com:10051

- CORE\_PEER\_GOSSIP\_BOOTSTRAP=peer0.org2.example.com:9051

- CORE\_PEER\_LOCALMSPID=Org2MSP

volumes:

- /var/run/:/host/var/run/

- ../crypto-config/peerOrganizations/org2.example.com/peers/peer1.org2.example.com/msp:/etc/hyperledger/fabric/msp

- ../crypto-config/peerOrganizations/org2.example.com/peers/peer1.org2.example.com/tls:/etc/hyperledger/fabric/tls

- peer1.org2.example.com:/var/hyperledger/production

ports:

- 10051:10051

**peer-base.yaml**

version: '2'

services:

peer-base:

image: hyperledger/fabric-peer:$IMAGE\_TAG

environment:

- CORE\_VM\_ENDPOINT=unix:///host/var/run/docker.sock

- CORE\_VM\_DOCKER\_HOSTCONFIG\_NETWORKMODE=${COMPOSE\_PROJECT\_NAME}\_byfn

// .env파일에 있는 net이라는 이름을 사용 합니다.

- FABRIC\_LOGGING\_SPEC=INFO

#- FABRIC\_LOGGING\_SPEC=DEBUG

- CORE\_PEER\_TLS\_ENABLED=true

- CORE\_PEER\_GOSSIP\_USELEADERELECTION=true

//Gossip 프로콜의 리더 선출을 수동으로 할지 자동으로 할지 설정 합니다.(true=자동, false=수동)

- CORE\_PEER\_GOSSIP\_ORGLEADER=false

//위의 설정에서 false를 사용하여 리더 선출을 수동으로 할 때

해당 peer를 리더로 할지 설정합니다.

- CORE\_PEER\_PROFILE\_ENABLED=true

- CORE\_PEER\_TLS\_CERT\_FILE=/etc/hyperledger/fabric/tls/server.crt

- CORE\_PEER\_TLS\_KEY\_FILE=/etc/hyperledger/fabric/tls/server.key

- CORE\_PEER\_TLS\_ROOTCERT\_FILE=/etc/hyperledger/fabric/tls/ca.crt

working\_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer

command: peer node start

// 컨테이너 생성시 피어 노드를 구동합니다.

orderer-base:

image: hyperledger/fabric-orderer:$IMAGE\_TAG

environment:

- FABRIC\_LOGGING\_SPEC=INFO

- ORDERER\_GENERAL\_LISTENADDRESS=0.0.0.0

- ORDERER\_GENERAL\_GENESISMETHOD=file

//genesis block을 file로 할지 provision로 할지 설정합니다.

-ORDERER\_GENERAL\_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block

//genesis block의 위치를 설정합니다.

- 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

- ORDERER\_GENERAL\_CLUSTER\_CLIENTCERTIFICATE=/var/hyperledger/orderer/tls/server.crt

- ORDERER\_GENERAL\_CLUSTER\_CLIENTPRIVATEKEY=/var/hyperledger/orderer/tls/server.key

- ORDERER\_GENERAL\_CLUSTER\_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]

working\_dir: /opt/gopath/src/github.com/hyperledger/fabric

command: orderer

// 컨테이너 생성시 오더러 노드를 구동합니다.

**script.sh**

echo

echo " \_\_\_\_ \_\_\_\_\_ \_ \_\_\_\_ \_\_\_\_\_ "

echo "/ \_\_\_| |\_ \_| / \ | \_ \ |\_ \_|"

echo "\\_\_\_ \ | | / \_ \ | |\_) | | | "

echo " \_\_\_) | | | / \_\_\_ \ | \_ < | | "

echo "|\_\_\_\_/ |\_| /\_/ \\_\ |\_| \\_\ |\_| "

echo

echo "Build your first network (BYFN) end-to-end test"

echo

CHANNEL\_NAME="$1"

DELAY="$2"

LANGUAGE="$3"

TIMEOUT="$4"

VERBOSE="$5"

NO\_CHAINCODE="$6"

: ${CHANNEL\_NAME:="mychannel"}

: ${DELAY:="3"}

: ${LANGUAGE:="golang"}

: ${TIMEOUT:="10"}

: ${VERBOSE:="false"}

: ${NO\_CHAINCODE:="false"}

LANGUAGE=`echo "$LANGUAGE" | tr [:upper:] [:lower:]`

COUNTER=1

MAX\_RETRY=20

PACKAGE\_ID=""

if [ "$LANGUAGE" = "node" ]; then

CC\_SRC\_PATH="/opt/gopath/src/github.com/hyperledger/fabric-samples/chaincode/abstore/node/"

elif [ "$LANGUAGE" = "java" ]; then

CC\_SRC\_PATH="/opt/gopath/src/github.com/hyperledger/fabric-samples/chaincode/abstore/java/"

else

CC\_SRC\_PATH="github.com/hyperledger/fabric-samples/chaincode/abstore/go/"

// 옵션 설정이 없을 시 go체인코드가 있는 폴더를 체인코드 폴더로 설정합니다.

fi

echo "Channel name : "$CHANNEL\_NAME

. scripts/utils.sh

createChannel() {

    setGlobals 0 1

    if [ -z "$CORE\_PEER\_TLS\_ENABLED" -o "$CORE\_PEER\_TLS\_ENABLED" = "false" ]; then

set -x

        peer channel create -o orderer.example.com:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/channel.tx >&log.txt

        res=$?

set +x

    else

                set -x

        peer channel create -o orderer.example.com:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/channel.tx --tls $CORE\_PEER\_TLS\_ENABLED --cafile $ORDERER\_CA >&log.txt

// tls를 사용하는 옵션으로 channel.tx 파일을 참조하여 채널을 생성합니다.

        res=$?

                set +x

    fi

    cat log.txt

    verifyResult $res "Channel creation failed"

    echo "===================== Channel '$CHANNEL\_NAME' created ===================== "

    echo

}

joinChannel () {

    for org in 1 2; do

     for peer in 0 1; do

        joinChannelWithRetry $peer $org

        echo "===================== peer${peer}.org${org} joined channel '$CHANNEL\_NAME' ===================== "

        sleep $DELAY

        echo

     done

    done

}

echo "Creating channel..."

createChannel

// 채널을 생성합니다.

echo "Having all peers join the channel..."

joinChannel

//모든 피어들을 생성한 채널에 가입시킵니다.

echo "Updating anchor peers for org1..."

updateAnchorPeers 0 1

//org1의 anchor피어를 업데이트 합니다.

echo "Updating anchor peers for org2..."

updateAnchorPeers 0 2

//org2의 anchor피어를 업데이트 합니다.

if [ "${NO\_CHAINCODE}" != "true" ]; then

    packageChaincode 1 0 1

    echo "Installing chaincode on peer0.org1..."

    installChaincode 0 1

    echo "Install chaincode on peer0.org2..."

    installChaincode 0 2

    queryInstalled 0 1

    approveForMyOrg 1 0 1

    simulateCommitChaincodeDefinition 1 0 1 "\"Org1MSP\": true" "\"Org2MSP\": false"

    simulateCommitChaincodeDefinition 1 0 2 "\"Org1MSP\": true" "\"Org2MSP\": false"

    approveForMyOrg 1 0 2

    simulateCommitChaincodeDefinition 1 0 1 "\"Org1MSP\": true" "\"Org2MSP\": true"

    simulateCommitChaincodeDefinition 1 0 2 "\"Org1MSP\": true" "\"Org2MSP\": true"

    commitChaincodeDefinition 1 0 1 0 2

    queryCommitted 1 0 1

    queryCommitted 1 0 2

    chaincodeInvoke 1 0 1 0 2

    echo "Querying chaincode on peer0.org1..."

    chaincodeQuery 0 1 100

    echo "Sending invoke transaction on peer0.org1 peer0.org2..."

    chaincodeInvoke 0 0 1 0 2

    echo "Querying chaincode on peer0.org1..."

    chaincodeQuery 0 1 90

    echo "Installing chaincode on peer1.org2..."

    installChaincode 1 2

    echo "Querying chaincode on peer1.org2..."

    chaincodeQuery 1 2 90

// 체인코드가 제대로 작동하는 테스트 합니다..

fi

echo

echo "========= All GOOD, BYFN execution completed =========== "

echo

echo

echo " \_\_\_\_\_ \_ \_ \_\_\_\_ "

echo "| \_\_\_\_| | \ | | | \_ \ "

echo "| \_| | \| | | | | | "

echo "| |\_\_\_ | |\ | | |\_| | "

echo "|\_\_\_\_\_| |\_| \\_| |\_\_\_\_/ "

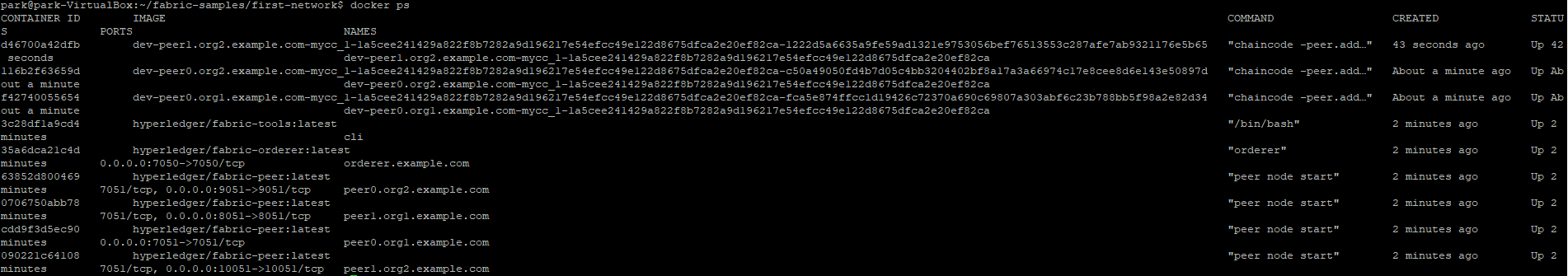
echo

exit 0

**./byfn up 결과화면**



**컨테이너 상태**



3./byfn.sh down

**./**byfn**.**sh down 명령어를 사용할 경우 networkDown() 함수를 사용하게 됩니다.

elif [ "${MODE}" == "down" ]; then ## Clear the network

networkDown

function networkDown() {

docker-compose -f $COMPOSE\_FILE -f $COMPOSE\_FILE\_COUCH -f $COMPOSE\_FILE\_KAFKA -f $COMPOSE\_FILE\_RAFT2 -f $COMPOSE\_FILE\_CA -f $COMPOSE\_FILE\_ORG3 down --volumes --remove-orphans

// 구동중인 fabric 컨테이너를 종료 합니다.

# Don't remove the generated artifacts -- note, the ledgers are always removed

if [ "$MODE" != "restart" ]; then

docker run -v $PWD:/tmp/first-network --rm hyperledger/fabric-tools:$IMAGETAG rm -Rf /tmp/first-network/ledgers-backup

//백업 파일을 삭제합니다.

clearContainers

//컨테이너를 삭제 합니다.

removeUnwantedImages

//도커 이미지를 삭제 합니다.

rm -rf channel-artifacts/\*.block channel-artifacts/\*.tx crypto-config ./org3-artifacts/crypto-config/ channel-artifacts/org3.json

//channel-artifact들을 삭제 합니다.

# remove the docker-compose yaml file that was customized to the example

rm -f docker-compose-e2e.yaml

//도커 컴포터 파일을 삭제 합니다.

fi

}

**./byfn down 결과화면**

