**Fabric2.3环境下单orderer双peer部署(2021年3月17日成功)**

**同步centos系统时间：**ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

$GOPATH/src/github.com/hyperledger/fabric/helloworld #生产路径

systemctl restart docker;

docker stop $(docker ps -a -q);

docker rm $(docker ps -a -q);

#Remove all unused local volumes；volume（数据卷）的设计目的就是数据的持久化，因为其生存周期独立于容器的生存周期，因此Docker不会在容器删除时删除其挂载的数据卷。

docker volume prune

#删除指定网络（自己生成的网络）

docker network rm config\_test

**※生成公私钥和证书：**

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

**注意：结果会生成ordererOrganizations和peerOrganizations及其内容，放置于crpyto-config文件夹，这是要把crpyto-config文件夹名改为organizations。**

**※为Org1和org2生成ccp文件：**

./organizations/ccp-generate.sh

**※创建通道创世区块channel1.block（使用configtx.yaml）**

export PATH=${PWD}/bin:$PATH;

export FABRIC\_CFG\_PATH=${PWD}/config;

./bin/configtxgen -profile TwoOrgsApplicationGenesis -outputBlock ./channel-artifacts/channel1.block -channelID channel1

**※docker中启动orderer、peer命令：**

docker-compose -f ./config/orderer-peer-cli.yaml up -d

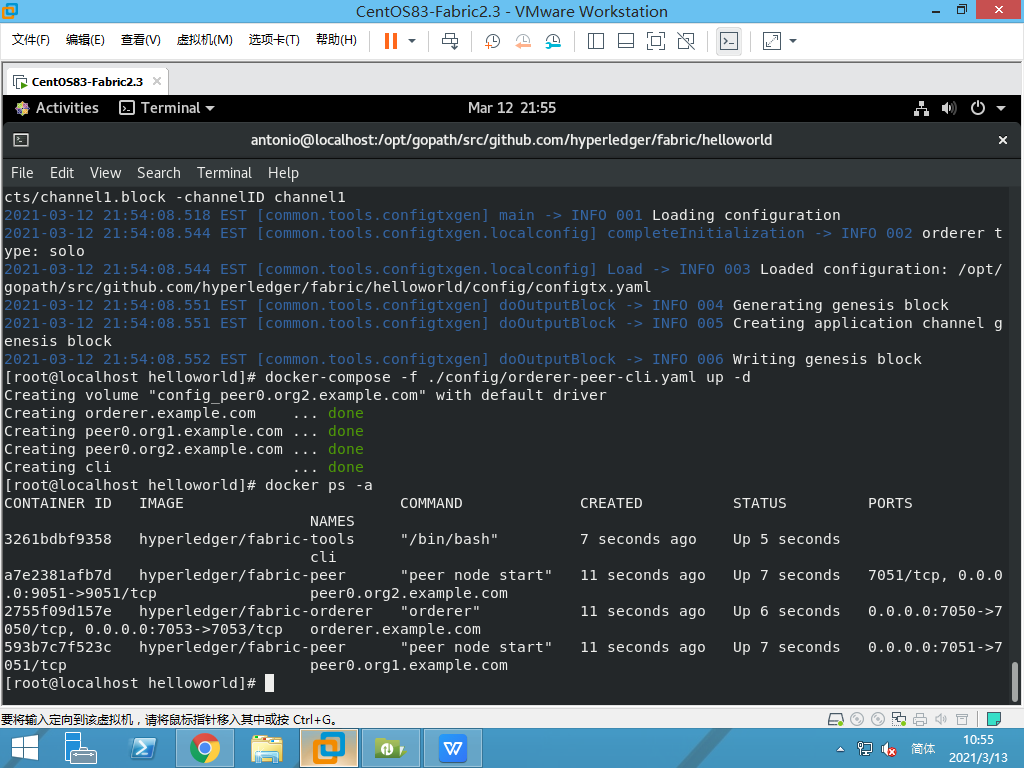
或docker-compose -f ${PWD}/config/orderer-peer-cli.yaml up -d

**解释1：其中，-f 可以使用多个，将多个配置文件合并为一个配置文件；-d，表示在后台运行的守护进程。**

**解释2：执行本命令，会生成一个network（如果已有同名网络，则不会生成）,这个网络的名称取决于两个方面，一是orderer-peer-cli.yaml中开头networks的名称test，二是命令本身orderer-peer-cli.yaml前面如果有文件夹的名称，如config，则network的名称就被命名为config\_test,如果orderer-peer-cli.yaml前面是./，则就会用执行本命令所在位置的文件夹，如helloworld，则network的就会被命名为helloworld\_test。所以，知道了这种命名规则，在使用orderer-peer-cli.yaml启动orderer、peer0.org1、peer0.org2的服务时，就需要事先在orderer-peer-cli.yaml中的CORE\_VM\_DOCKER\_HOSTCONFIG\_NETWORKMODE值写成相应的network的名称（如，config\_test）。**

**注意1：代码行中即便显示orderer.example.com、peer0.example.com、cli都显示done即启动成功了，也不代表真正启动成功了，还需要使用命令docker ps -a查看一下他们的status是否up，如果是exited,则要使用类似命令docker logs orderer.example.com查看一下日志中具体提示了什么样的错误。**

**注意2：orderer-peer-cli.yaml中volumes中所用msp和tls文件夹的路径，是相对orderer-peer-cli.yaml文件所在路径的路径，如ordererOrganizations文件夹在orderer-peer-cli.yaml上层的organizations文件夹里，则在volumes中这样标示：../organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp:/var/hyperledger/orderer/msp**



**※(无系统通道下使用osnadmin创建应用通道)创建通道channel1（让Orderer加入通道）：**

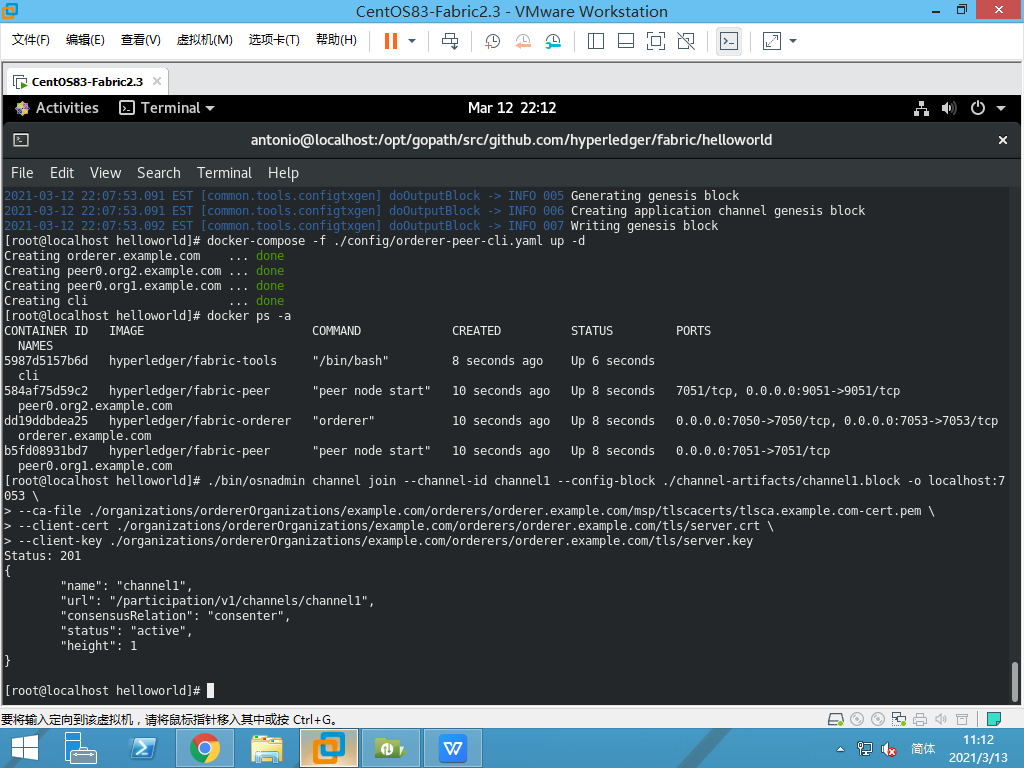
./bin/osnadmin channel join --channel-id channel1 --config-block ./channel-artifacts/channel1.block -o localhost:7053 \

--ca-file ./organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \

--client-cert ./organizations/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.crt \

--client-key ./organizations/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.key

**注意：如果configtx.yaml中OrdererType是solo而不是etcdraft，则会出现错误："error": "cannot join: failed cast: clusterConsenter is not a consensus.ClusterConsenter"**



**可以使用下面的osnadmin channel list命令查看orderer节点上的通道列表及其status和consensusRelation（共识关系）：**

./bin/osnadmin channel list --channel-id channel1 -o localhost:7053 \

--ca-file ./organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \

--client-cert ./organizations/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.crt \

--client-key ./organizations/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.key

**可以使用下面的osnadmin channel remove删除通道channel1,删除后显示status:204(如果该orderer节点是该通道唯一的orderer节点，则该通道就会无法在接受新的交易)：**

./bin/osnadmin channel remove --channel-id channel1 -o localhost:7053 \

--ca-file ./organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \

--client-cert ./organizations/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.crt \

--client-key ./organizations/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.key

**#将organization1加入通道channel1（**#使用路径下的core.yaml配置文件**）：**

export FABRIC\_CFG\_PATH=${PWD}/config;

export CORE\_PEER\_TLS\_ENABLED=true;

export CORE\_PEER\_LOCALMSPID="Org1MSP";

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt;

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp;

export CORE\_PEER\_ADDRESS=localhost:7051;

./bin/peer channel join -b ./channel-artifacts/channel1.block

**#将organization2加入通道channel1（**#使用路径下的core.yaml配置文件**）：**

export FABRIC\_CFG\_PATH=${PWD}/config;

export CORE\_PEER\_TLS\_ENABLED=true;

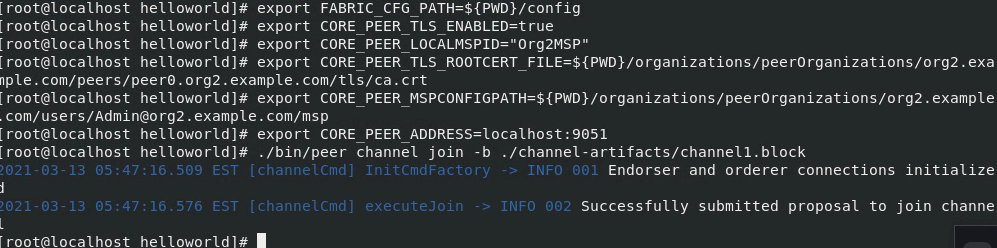
export CORE\_PEER\_LOCALMSPID="Org2MSP";

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt;

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp;

export CORE\_PEER\_ADDRESS=localhost:9051;

./bin/peer channel join -b ./channel-artifacts/channel1.block



**#在通道channel1上为Org1MSP设置锚节点（Anchor peer set for org 'Org1MSP' on channel 'channel1'）**

export CORE\_PEER\_TLS\_ENABLED=true;

export CORE\_PEER\_LOCALMSPID="Org1MSP";

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt;

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp;

export CORE\_PEER\_ADDRESS=localhost:7051;

./bin/peer channel fetch config config\_block.pb -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com -c channel1 --tls --cafile \

${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

./bin/configtxlator proto\_decode --input config\_block.pb --type common.Block | jq .data.data[0].payload.data.config >Org1MSPconfig.json;

jq '.channel\_group.groups.Application.groups.Org1MSP.values += {"AnchorPeers":{"mod\_policy": "Admins","value":{"anchor\_peers": [{"host": "peer0.org1.example.com","port": 7051}]},"version": "0"}}' Org1MSPconfig.json >Org1MSPmodified\_config.json; #修改配置文件：加入锚节点信息;jq是Linux下面把文本字符串格式化成json格式的工具

./bin/configtxlator proto\_encode --input Org1MSPconfig.json --type common.Config >original\_config.pb;

./bin/configtxlator proto\_encode --input Org1MSPmodified\_config.json --type common.Config >modified\_config.pb;

./bin/configtxlator compute\_update --channel\_id channel1 --original original\_config.pb --updated modified\_config.pb >config\_update.pb;

./bin/configtxlator proto\_decode --input config\_update.pb --type common.ConfigUpdate >config\_update.json;

echo '{"payload":{"header":{"channel\_header":{"channel\_id":"channel1", "type":2}},"data":{"config\_update":'$(cat config\_update.json)'}}}' | jq . >config\_update\_in\_envelope.json; #jq把文本字符串格式化成json;cat将几个文件合并为一个文件

./bin/configtxlator proto\_encode --input config\_update\_in\_envelope.json --type common.Envelope >Org1MSPanchors.tx

#向orderer发送锚节点配置文件Org1MSPanchors.tx

./bin/peer channel update -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com -c channel1 -f Org1MSPanchors.tx --tls --cafile \

${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

说明：因为无论配置交易文件 .tx和初始区块文件 .block都是二进制格式，用户无法直接编辑。 configtxlator工具将这些配置文件在二进制格式和方便阅读的json格式之间进行转换。所以在本地的应用中， 有一个专用的 configtxlator 进程给每一个使用者。

**#在通道channel1上为Org2MSP设置锚节点（Anchor peer set for org 'Org2MSP' on channel 'channel1'）**

export CORE\_PEER\_TLS\_ENABLED=true;

export CORE\_PEER\_LOCALMSPID="Org2MSP";

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt;

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp;

export CORE\_PEER\_ADDRESS=localhost:9051;

./bin/peer channel fetch config config\_block.pb -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com -c channel1 --tls --cafile \

${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

./bin/configtxlator proto\_decode --input config\_block.pb --type common.Block | jq .data.data[0].payload.data.config >Org2MSPconfig.json;

jq '.channel\_group.groups.Application.groups.Org2MSP.values += {"AnchorPeers":{"mod\_policy": "Admins","value":{"anchor\_peers": [{"host": "peer0.org2.example.com","port": 9051}]},"version": "0"}}' Org2MSPconfig.json >Org2MSPmodified\_config.json; #修改配置文件：加入锚节点信息

./bin/configtxlator proto\_encode --input Org2MSPconfig.json --type common.Config >original\_config.pb;

./bin/configtxlator proto\_encode --input Org2MSPmodified\_config.json --type common.Config >modified\_config.pb;

./bin/configtxlator compute\_update --channel\_id channel1 --original original\_config.pb --updated modified\_config.pb >config\_update.pb;

./bin/configtxlator proto\_decode --input config\_update.pb --type common.ConfigUpdate >config\_update.json;

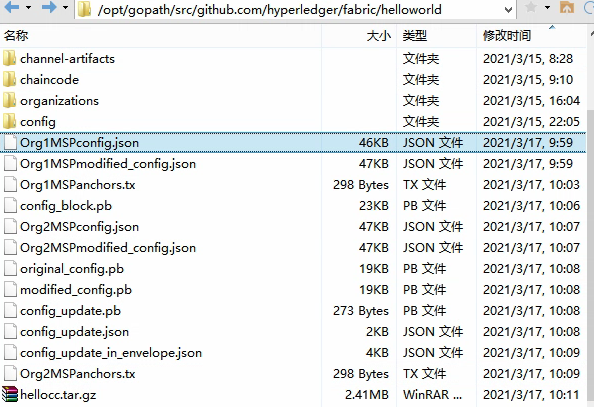
echo '{"payload":{"header":{"channel\_header":{"channel\_id":"channel1", "type":2}},"data":{"config\_update":'$(cat config\_update.json)'}}}' | jq . >config\_update\_in\_envelope.json; #jq把文本字符串格式化成json;cat将几个文件合并为一个文件

./bin/configtxlator proto\_encode --input config\_update\_in\_envelope.json --type common.Envelope >Org2MSPanchors.tx

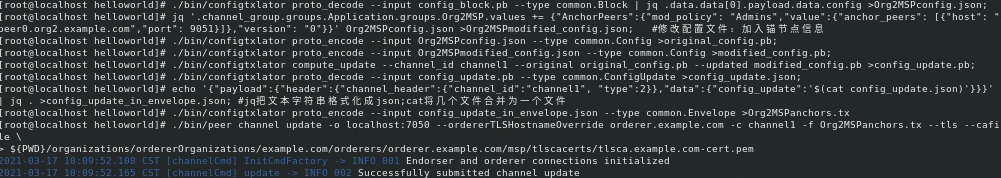
#向orderer发送锚节点配置文件Org2MSPanchors.tx

./bin/peer channel update -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com -c channel1 -f Org2MSPanchors.tx --tls --cafile \

${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem



执行设置锚节点命令结果



**#打包chaincode：**

./bin/peer lifecycle chaincode package hellocc.tar.gz -p ./chaincode/go/helloworld --label hello\_1

结果：在执行命令所在的文件夹helloworld文件夹中，就生成了hellocc.tar.gz。

**#分别在peer0.org1和peer0.org2上安装链码：**

**？从Fabric2.0开始，链码不用安装到每个peer上了，链码可以作为服务部署和执行于Fabric外，服务于所有peers？**

export CORE\_PEER\_TLS\_ENABLED=true;

export CORE\_PEER\_LOCALMSPID="Org1MSP";

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt;

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp;

export CORE\_PEER\_ADDRESS=localhost:7051;

./bin/peer lifecycle chaincode install hellocc.tar.gz

export CORE\_PEER\_TLS\_ENABLED=true;

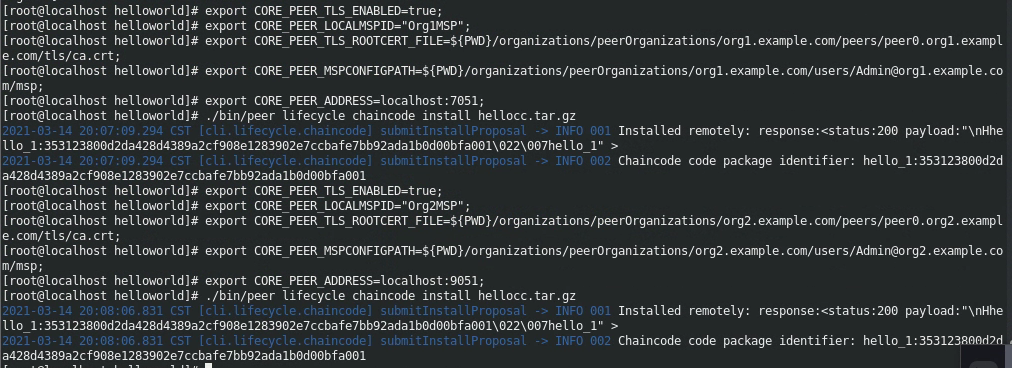
export CORE\_PEER\_LOCALMSPID="Org2MSP";

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt;

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp;

export CORE\_PEER\_ADDRESS=localhost:9051;

./bin/peer lifecycle chaincode install hellocc.tar.gz



查询package ID（用于查询peer）,使用命令：

./bin/peer lifecycle chaincode queryinstalled



**#approve 链码定义（defination），即批准链码定义。(每条完整的命令间用分号隔开，一下子把这些命令复制到termimal,可以从前往依次快速执行)**

export CORE\_PEER\_TLS\_ENABLED=true;

export CORE\_PEER\_LOCALMSPID="Org2MSP";

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt;

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp;

export CORE\_PEER\_ADDRESS=localhost:9051;

./bin/peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls \

--cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \

--channelID channel1 --name hello\_1 --version 1.0 --package-id hello\_1:5cd76591329d8c8fd9d23516484735adf574e88f13b81c0f09ff0330e71dc719 --sequence 1

结果显示，链码定义在peer0.org2上被approved，因为执行这段代码时，当前设置的时org2的环境变量。所以**也要在org1环境变量下执行一次批准链码定义**：

export CORE\_PEER\_TLS\_ENABLED=true;

export CORE\_PEER\_LOCALMSPID="Org1MSP";

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt;

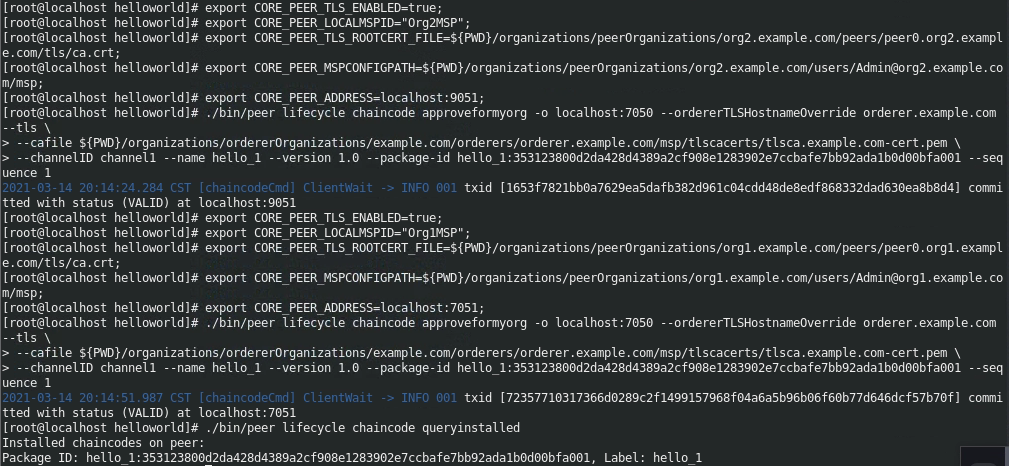
export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp;

export CORE\_PEER\_ADDRESS=localhost:7051;

./bin/peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls \

--cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \

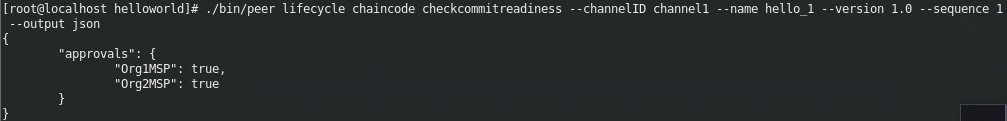
--channelID channel1 --name hello\_1 --version 1.0 --package-id hello\_1:5cd76591329d8c8fd9d23516484735adf574e88f13b81c0f09ff0330e71dc719 --sequence 1



注意：虽然显示成功了，但是把刚安装的链码给弄没了（查询发现竟然没有安装链码），查看日志，错误：Error reading from 172.20.0.1:52890: rpc error: code = Canceled desc = context canceled。网上资料都说是peer配置文件里CORE\_VM\_DOCKER\_HOSTCONFIG\_NETWORKMODE的设置问题（用的helloworld\_default）。**最后发现，是在安装链码的时候，没有分别在peer0.org1和peer0.org2上安装链码，而是直接执行了一条./bin/peer lifecycle chaincode install hellocc.tar.gz命令**。

**#检查链码定义（defination）的commit readiness,即检查链码是否可以向通道提交**

./bin/peer lifecycle chaincode checkcommitreadiness --channelID channel1 --name hello\_1 --version 1.0 --sequence 1 --output json



查询结果显示，链码定义的commit readiness在peer0.org1为approvals:true,在pee0.org2上为approvals:true。

**#向通道提交链码定义（defination）**

./bin/peer lifecycle chaincode commit -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls \

--cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \

--channelID channel1 --name hello\_1 \

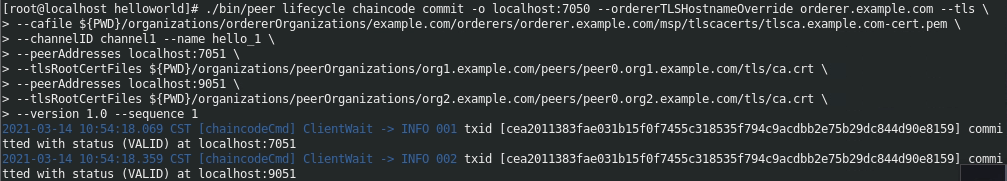
--peerAddresses localhost:7051 \

--tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt \

--peerAddresses localhost:9051 \

--tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt \

--version 1.0 --sequence 1



执行结果显示：在org1和org2上向通道提交链码定义的状态均为VALID。

注意：如果在任意org上没有approve链码定义（approveformyorg），则本步骤的命令将无法执行成功，会提示类似“chaincode definition not agreed to by this org (Org1MSP)”的错误提示。

可以在channel上的org1或channel上的org2上查询链码定义：

./bin/peer lifecycle chaincode querycommitted --channelID channel1 --name hello\_1



**#调用链码上的函数InitLedger（本案例链码是从fabric-samples中复制的test-network的链码）**

export PATH=${PWD}/bin:$PATH

export FABRIC\_CFG\_PATH=${PWD}/config/ #设置FABRIC\_CFG\_PATH 指向core.yaml文件

export CORE\_PEER\_TLS\_ENABLED=true #以下为设置环境变量为org1的，即在org1上调用链码

export CORE\_PEER\_LOCALMSPID="Org1MSP"

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

export CORE\_PEER\_ADDRESS=localhost:7051

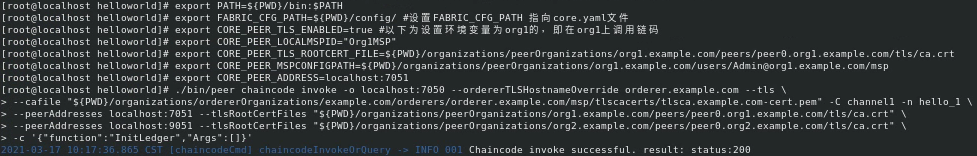
./bin/peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls \

--cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C channel1 -n hello\_1 \

--peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" \

--peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" \

-c '{"function":"InitLedger","Args":[]}'



**#查询channel ledger（账本）**

./bin/peer chaincode query **-**C channel1 **-**n hello\_1 **-**c '{"Args":["GetAllAssets"]}'



**#资产转移：将asset6从Michel名下过户给Christopher**

./bin/peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C channel1 -n hello\_1 --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"**TransferAsset**","Args":["asset6","Christopher"]}'

说明：因为背书策略要求交易需要org1和org2同时签名，所以链码调用命令需要使用参数--peerAddresse指向地址peer0.org1.example.com和peer0.org2.example.com；又由于网络启用了TLS，所以此命令也需要用参数--tlsRootCertFiles使用两个节点各自的TLS证书。



**#再次查询channel ledger**

./bin/peer chaincode query **-**C channel1 **-**n hello\_1 **-**c '{"Args":["GetAllAssets"]}'

结果显示：从org2的peer0上查询到的结果和org1的peer0查询的结果是一致的，特别突出的是，asset6的资产目前确实是属于Christopher的。

