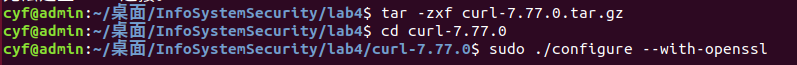
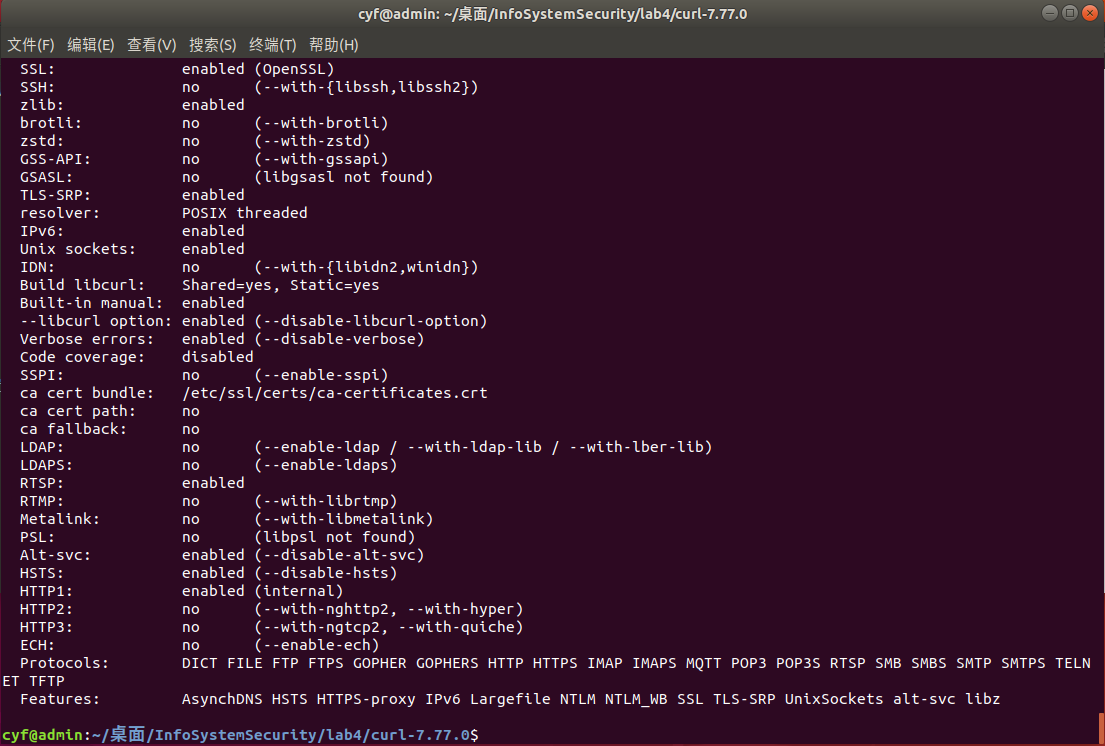
安装依赖



下载curl包并解压：

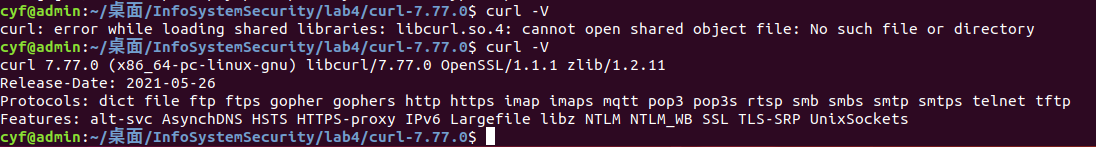
编译安装（需要make和gcc支持）：







安装结束后使用如下命令查看是否安装成功，同时注意要支持https协议：

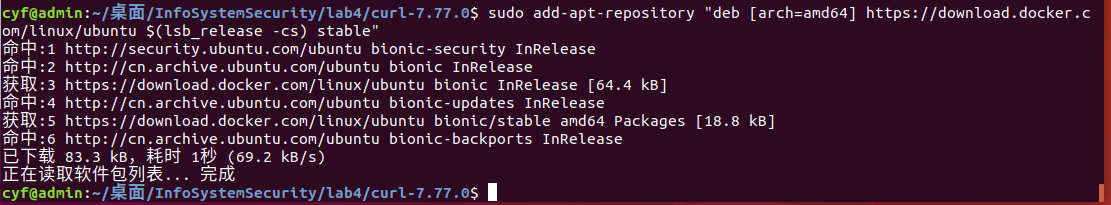


如果没有旧版本docker，直接使用如下命令安装。首先安装相关依赖：

接着添加docker官方GPG密钥，命令如下：

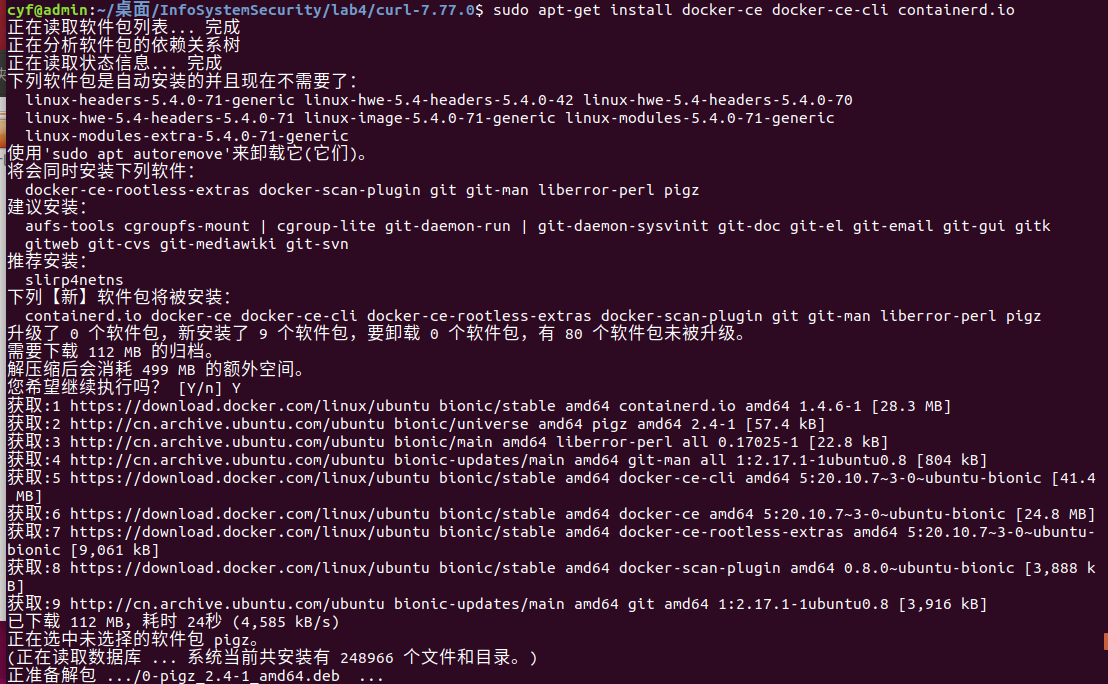


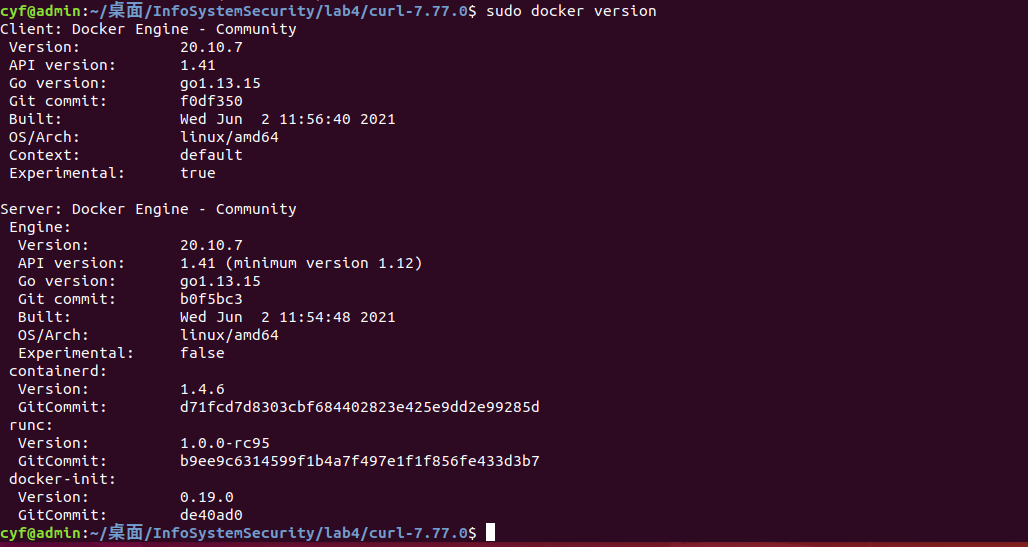
随后使用如下命令设立仓库：



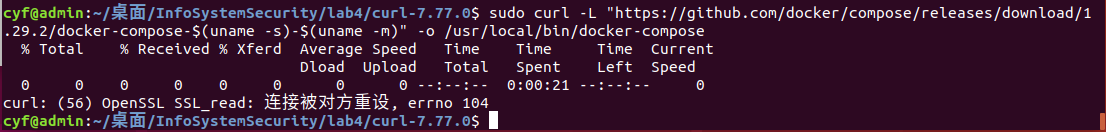
接着使用如下命令安装docker：

接着使用如下命令查看docker是否安装完成，请保证较高的docker版本（19.03.11以上），

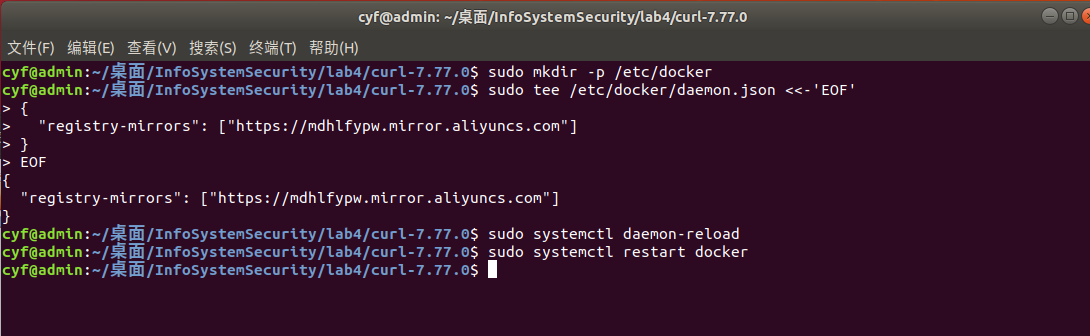
否则后续可能失败：

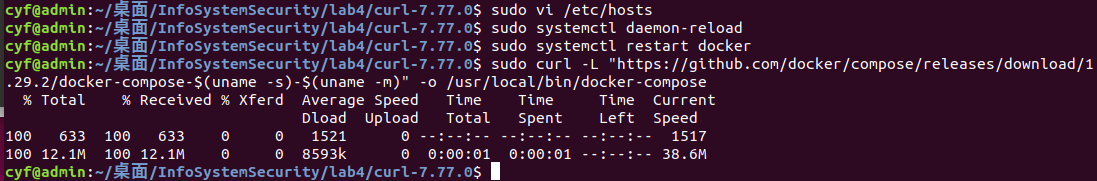


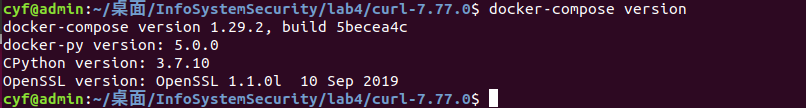
安装docker-compose

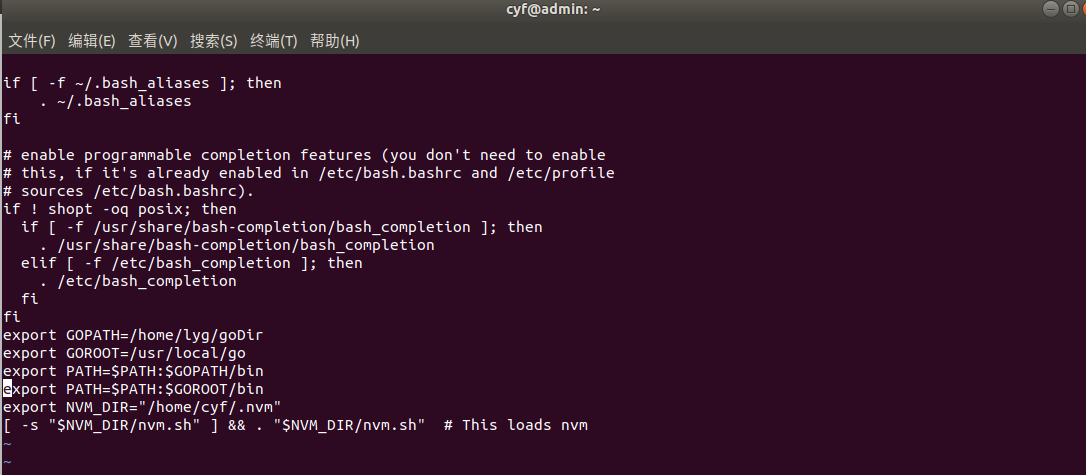


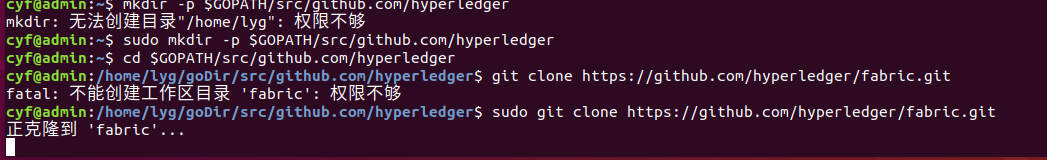
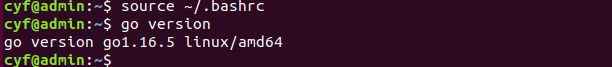


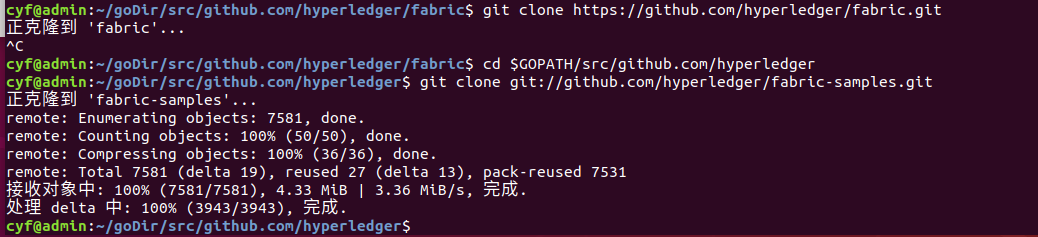
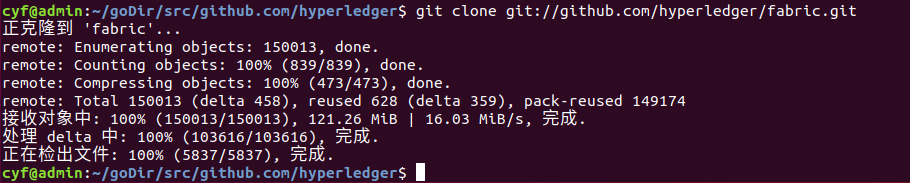








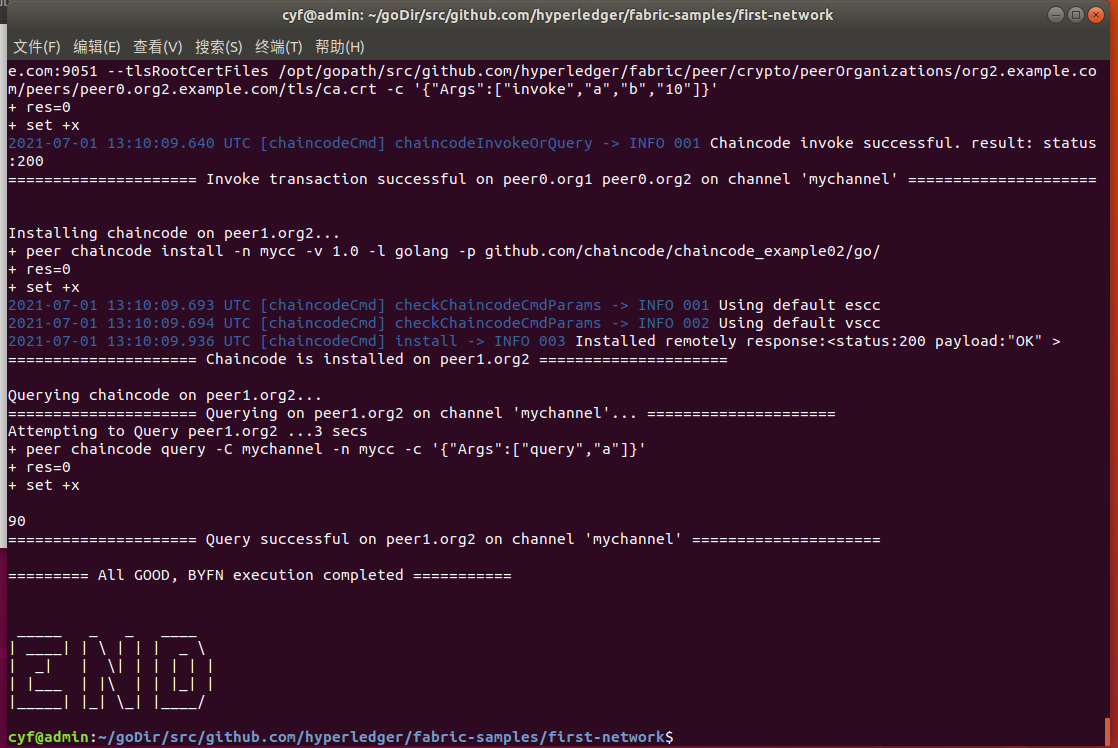
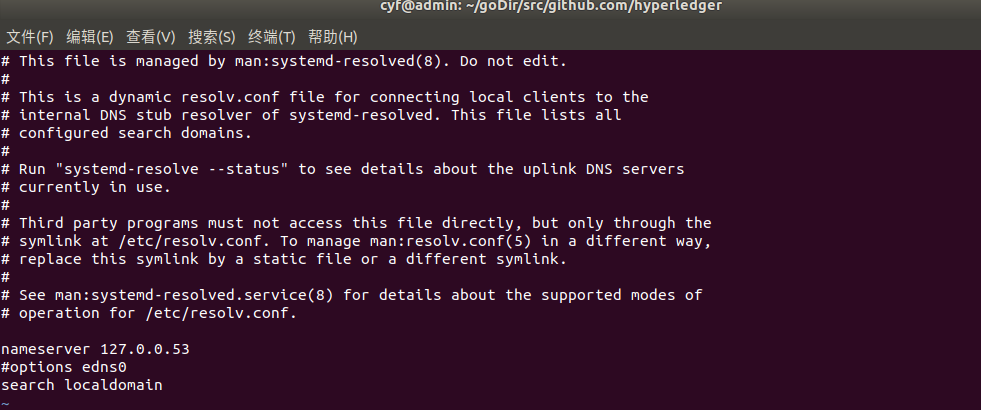


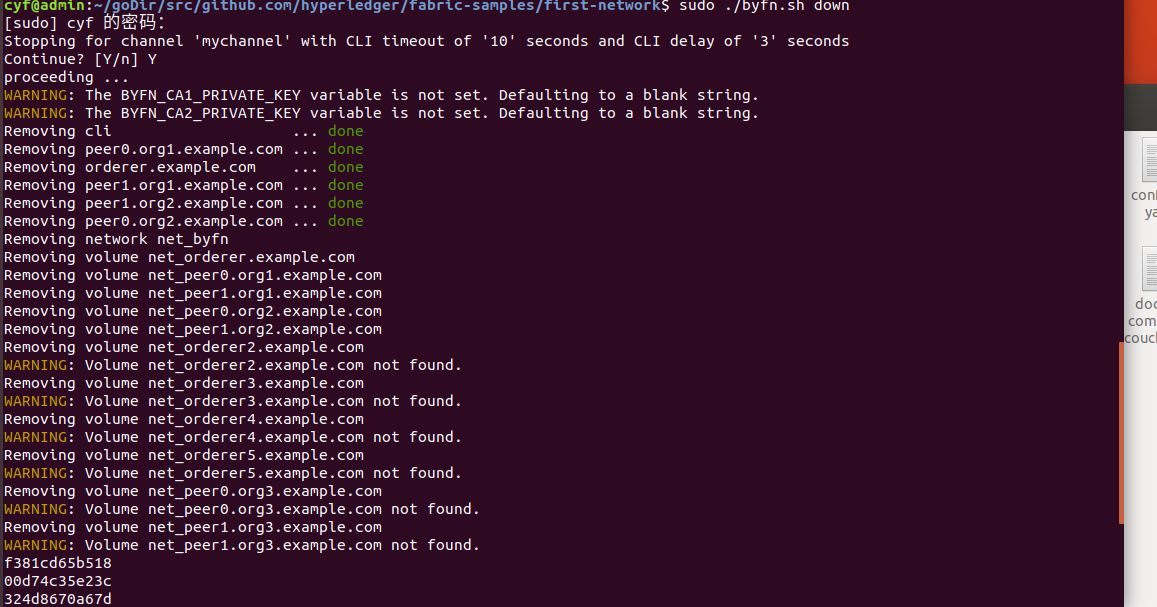












## 一、生成组织结构与身份证书

### 1.1、crypto-config.yaml配置文件

第一步加载crypto-config.yaml。该文件主要指定整个网络中的相关组织信息，我们稍加修改示例即可，最终结果如下

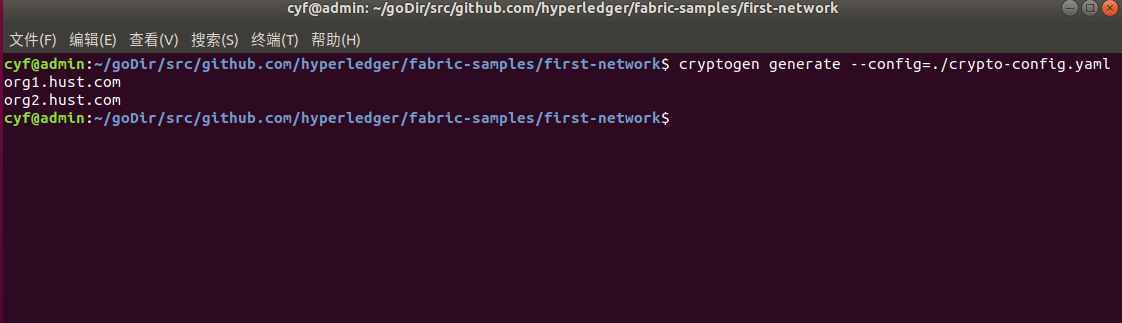
该配置创建了OrdererOrgs和PeerOrgs两个组织信息，其中PeerOrgs中有两个org，每个org中有两个peer节点和一个user。

**1.2、生成组织结构及身份证书**

我们使用以前编译好的cryptogen工具来生成，命令如下：

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

结果如下：



执行完该命令后就会在当前目录crypto-config文件夹，里边包括相关的结果及证书，我们可以用tree命令看详细情况，展开三级，

tree -L 3 crypto-config



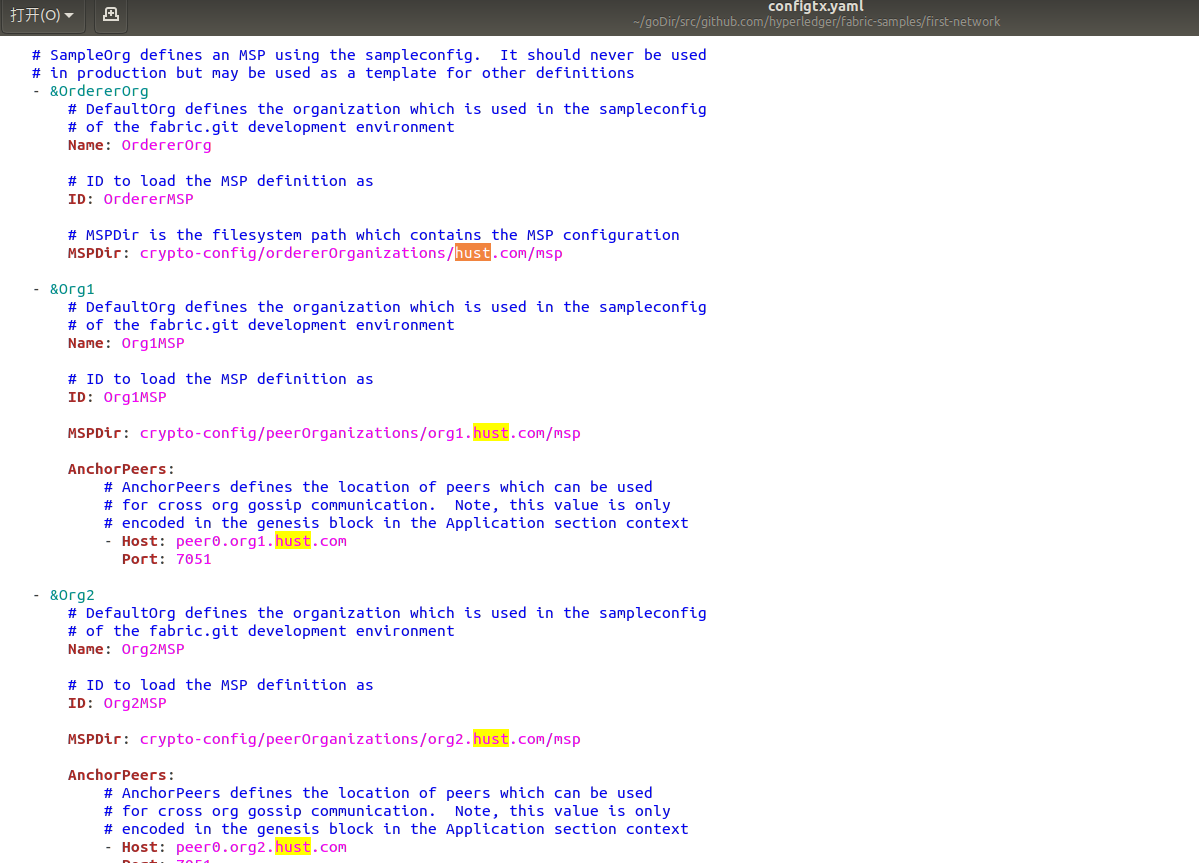
接下来要告诉configtxgen工具去找这个configtx.yaml文件，让他在当前的工作目录寻找：

export FABRIC\_CFG\_PATH=$PWD

## 二、生成其他配置文件

### 2.1、configtx.yaml文件

再生成组织结构等文件后，我们还需要创世区块和通道。该文件的生成依然依靠yaml文件，而这些相关文件主要在configtx.yaml文件,我们指定上一步生成文件的路径就可以了。

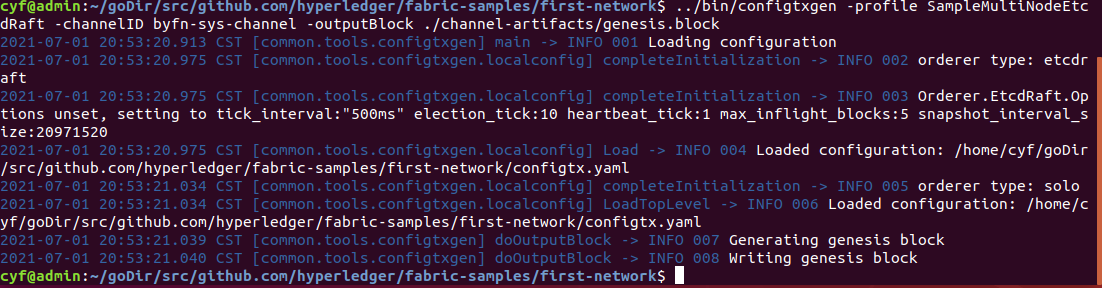


### 2.2、Orderer服务启动初始区块创建

### export FABRIC\_CFG\_PATH=$PWD

这一步使用到的工具为configtxgen，关于详细使用情况我们用help即可查看，生成配置文件的命令如下：

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

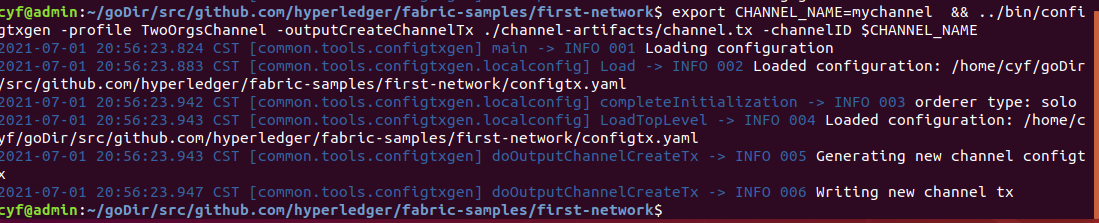


### 2.3、通道创始文件

相关命令：

export CHANNEL\_NAME=mychannel && ../bin/configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID $CHANNEL\_NAME

最终结果：

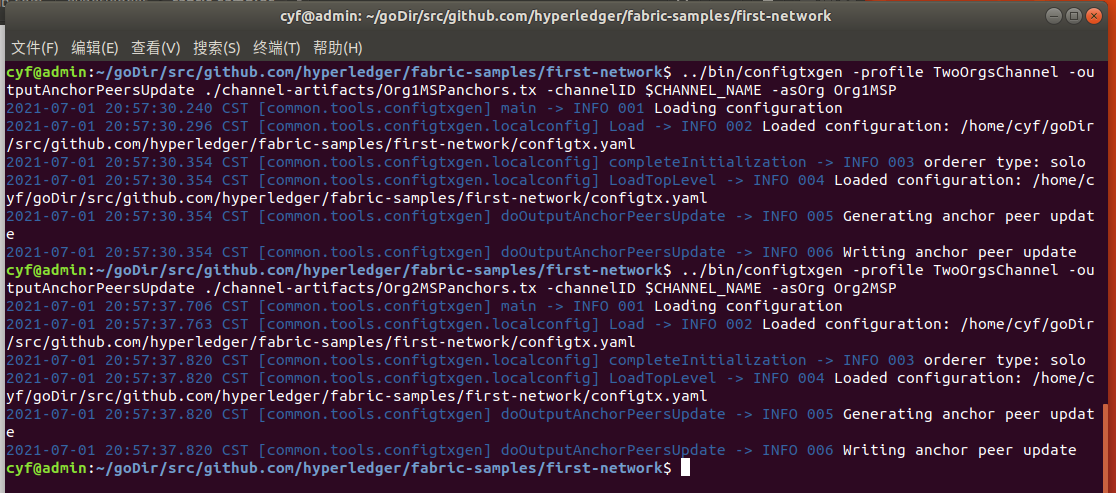


### 2.4、生成锚节点更新配置文件

锚节点更新配置文件在通道创建之后用来更新组织中的锚节点，同样基于configtx.yaml文件和configtxgen工具，相关命令如下：

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

../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID $CHANNEL\_NAME -asOrg Org2MSP



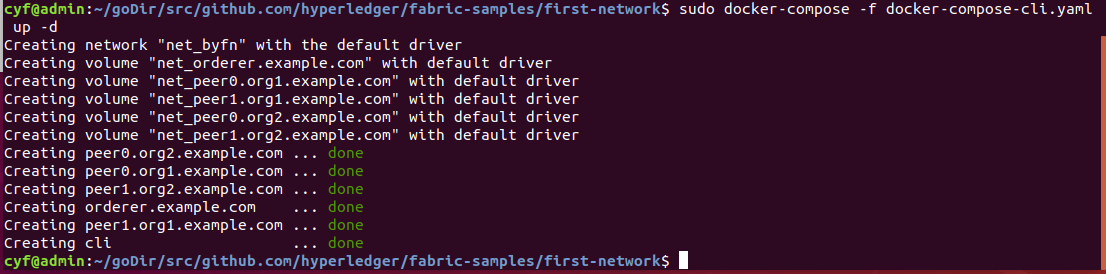
三、网络的启动

相关配置文件

截止上一步我们的网络相关配置文件已经生成，下一步就是启动网络，启动网络主要用到的文件有：docker-compose-cli.yaml 、base/docker-compose-base.yaml 、``base/peer-base.yaml`

用docker-compose来启动，相关命令：

sudo docker-compose -f docker-compose-cli.yaml up -d



## 四、通道创建

通道的概念类似子网，通道提供一种通信机制，能够将Peer和Orderer连接在一起，形成一个具有保密性质的通信链路。

### 4.1、创建通道

进入cli容器

Sudo docker exec -it cli bash

设置环境变量

# Environment variables for PEER0

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

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

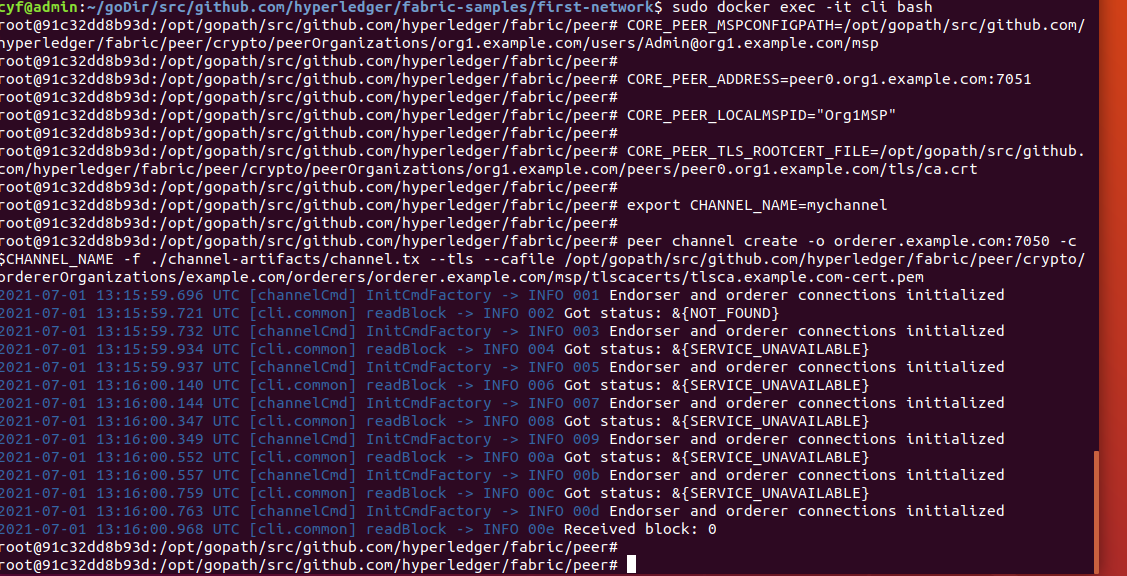
CORE\_PEER\_LOCALMSPID="Org1MSP"

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

创建通道

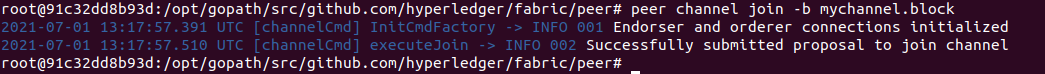
export CHANNEL\_NAME=mychannel

peer channel create -o orderer.example.com:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/channel.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem



应用通道所包含组织的成员节点可以加入通道中

peer channel join -b mychannel.block



加入peer0.org2.example.com

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp CORE\_PEER\_ADDRESS=peer0.org2.example.com:9051 CORE\_PEER\_LOCALMSPID="Org2MSP" CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt

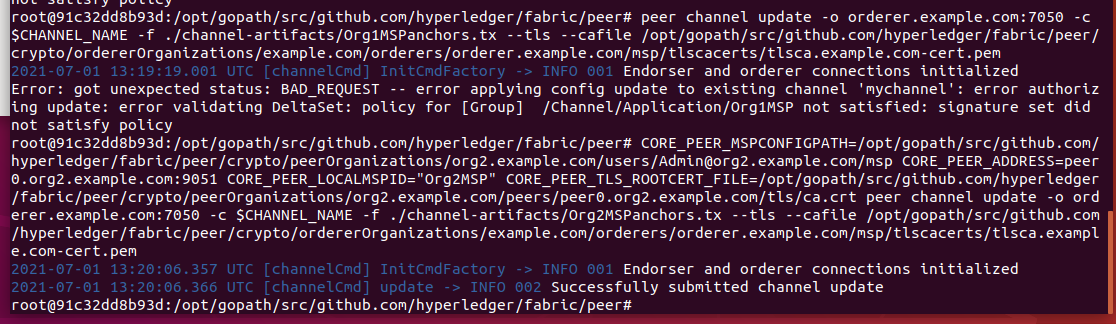
peer channel join -b mychannel.block



更新锚节点

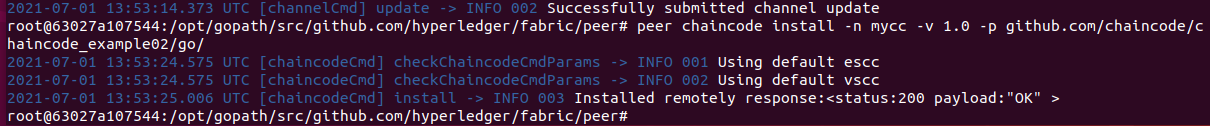
peer channel update -o orderer.example.com:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/Org1MSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp CORE\_PEER\_ADDRESS=peer0.org2.example.com:9051 CORE\_PEER\_LOCALMSPID="Org2MSP" CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt peer channel update -o orderer.example.com:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/Org2MSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

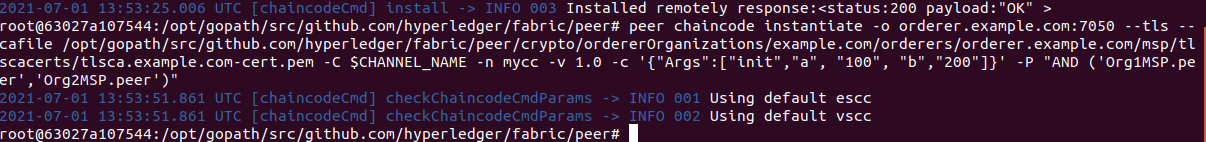


**安装实例化链码**

peer chaincode install -n mycc -v 1.0 -p github.com/chaincode/chaincode\_example02/go/



peer chaincode instantiate -o orderer.example.com:7050 --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n mycc -v 1.0 -c '{"Args":["init","a", "100", "b","200"]}' -P "AND ('Org1MSP.peer','Org2MSP.peer')"

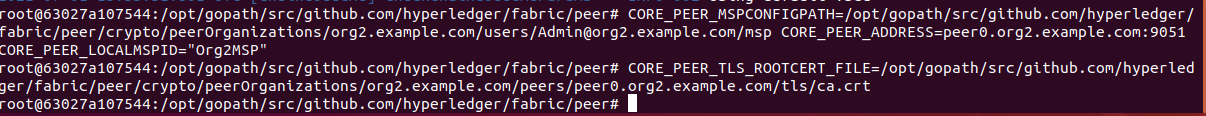


**安装链码在org2**

# 将Org2 加入通道

# 设置环境为Org2的

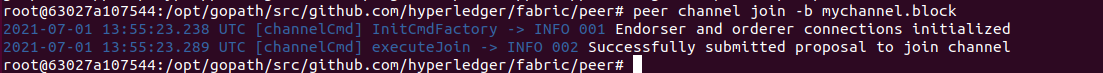
CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp CORE\_PEER\_ADDRESS=peer0.org2.example.com:9051 CORE\_PEER\_LOCALMSPID="Org2MSP" CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt



# 上面并不是将所有的节点都加入了通道只是简单地加入了peer0.org2.example.com，以便我们可以对定义在通道中的锚节点进行更新。

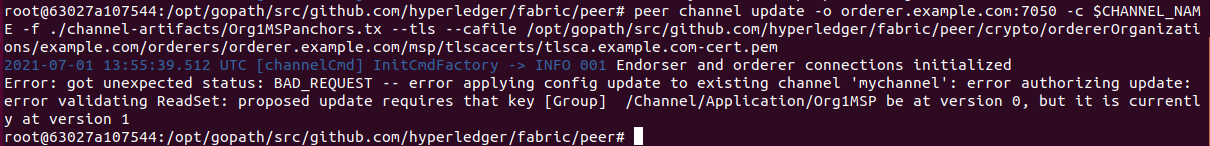
# 将peer0.org1.example.com加入通道中去。

peer channel join -b mychannel.block



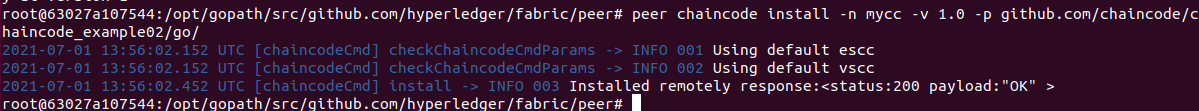
# 更新通道，将peer0.org1设为锚节点

peer channel update -o orderer.example.com:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/Org1MSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem



# 安装链码

peer chaincode install -n mycc -v 1.0 -p github.com/chaincode/chaincode\_example02/go/



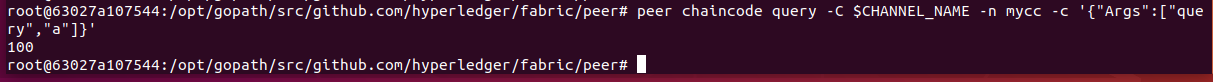
# 实例化链码

peer chaincode instantiate -o orderer.example.com:7050 --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n mycc -v 1.0 -c '{"Args":["init","a", "100", "b","200"]}' -P "AND ('Org1MSP.peer','Org2MSP.peer')"

##可以对链码进行查询、调用等操作了

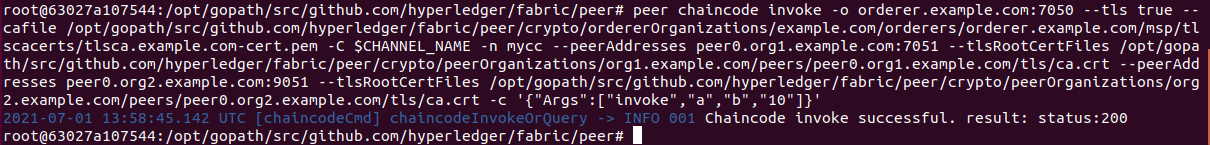
# 查询a

peer chaincode query -C $CHANNEL\_NAME -n mycc -c '{"Args":["query","a"]}'



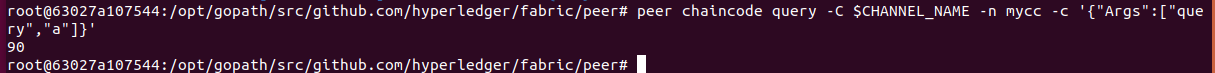
# 从a中取10到b（指定order服务点、tls模式、证书位置，通道名称，支出方节点地址、证书位置，收取方节点地址、证书位置， 指定Json格式的链码的构造消息函数{}）

peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n mycc --peerAddresses peer0.org1.example.com:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt --peerAddresses peer0.org2.example.com:9051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt -c '{"Args":["invoke","a","b","10"]}'



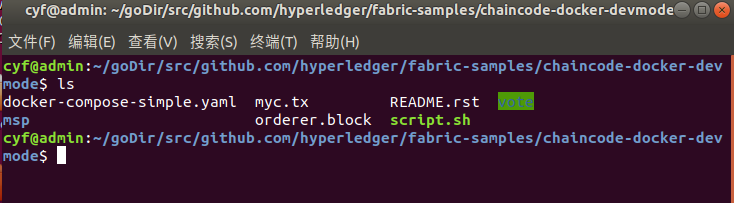
# 再次查询，看到金额发送变化

peer chaincode query -C $CHANNEL\_NAME -n mycc -c '{"Args":["query","a"]}'



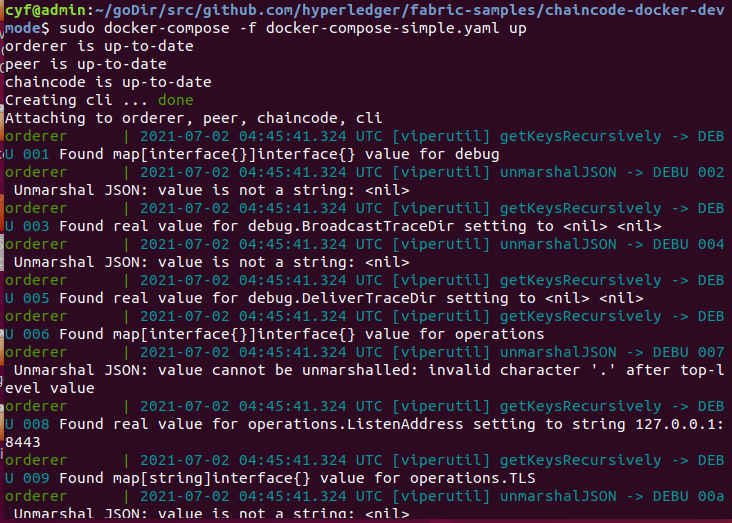
实现一个简单的投票链码(go编写)，包含两个功能，给用户投票，获取投票信息。

1. 编写智能合约代码vote.go
2. 将代码文件夹/vote拷贝到fabric-samples下面的chaincode文件夹



三、进入到fabric-samples/chaincode-docker-devmode，启动网络

docker-compose -f docker-compose-simple.yaml up



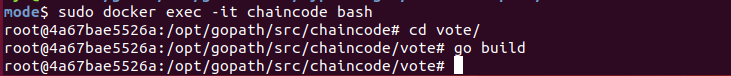
四、新开一个终端，编译链码，并启动

sudo docker exec -it chaincode bash

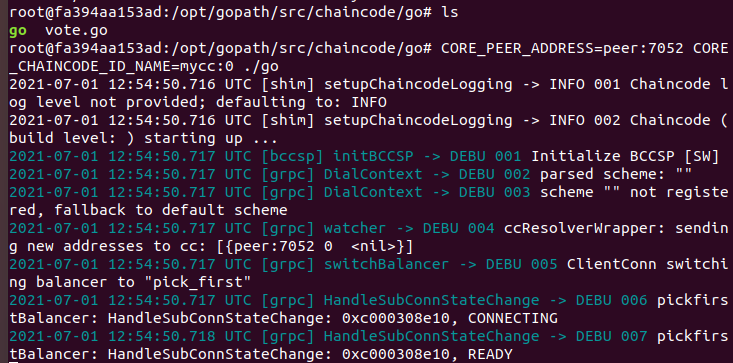


cd vote/

go build



CORE\_PEER\_ADDRESS=peer:7052 CORE\_CHAINCODE\_ID\_NAME=mycc:0 ./vote

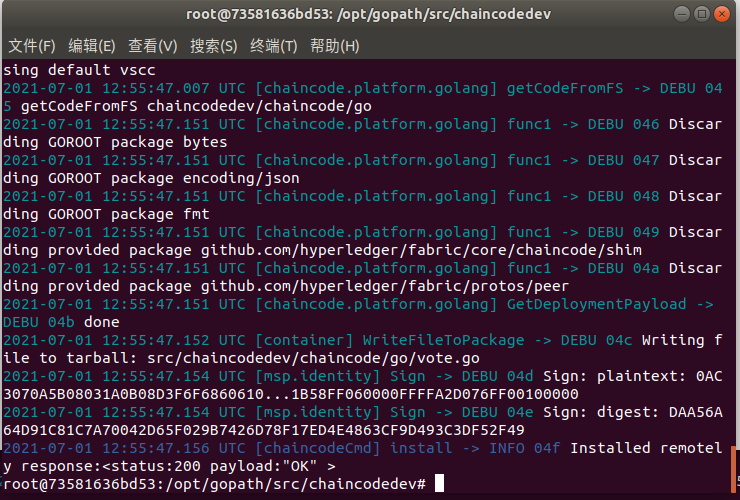


五、新开一个终端，进入docker容器，操作链码

docker exec -it cli bash

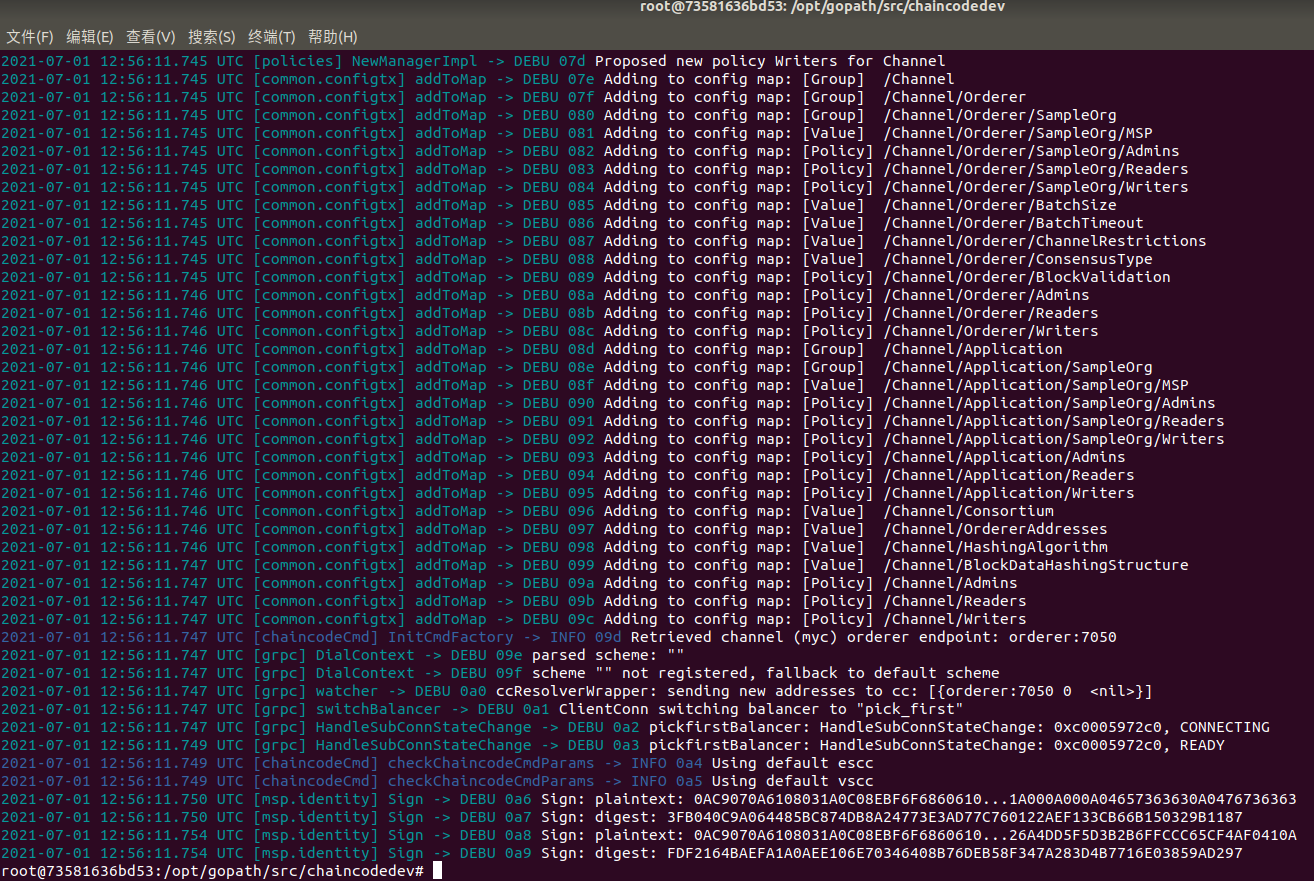
安装链码

peer chaincode install -p chaincodedev/chaincode/vote -n mycc -v 0



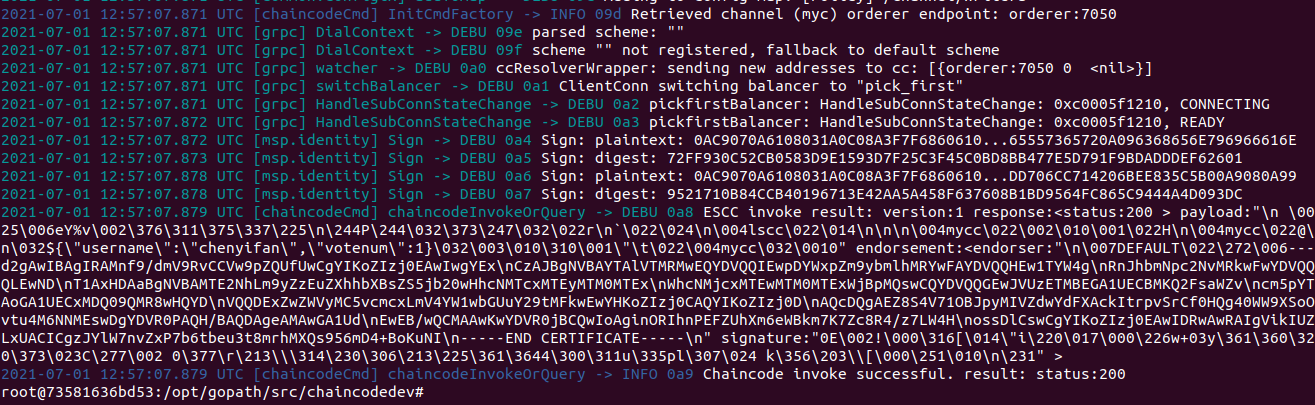
实例化链码

peer chaincode instantiate -n mycc -v 0 -c '{"Args":[]}' -C myc



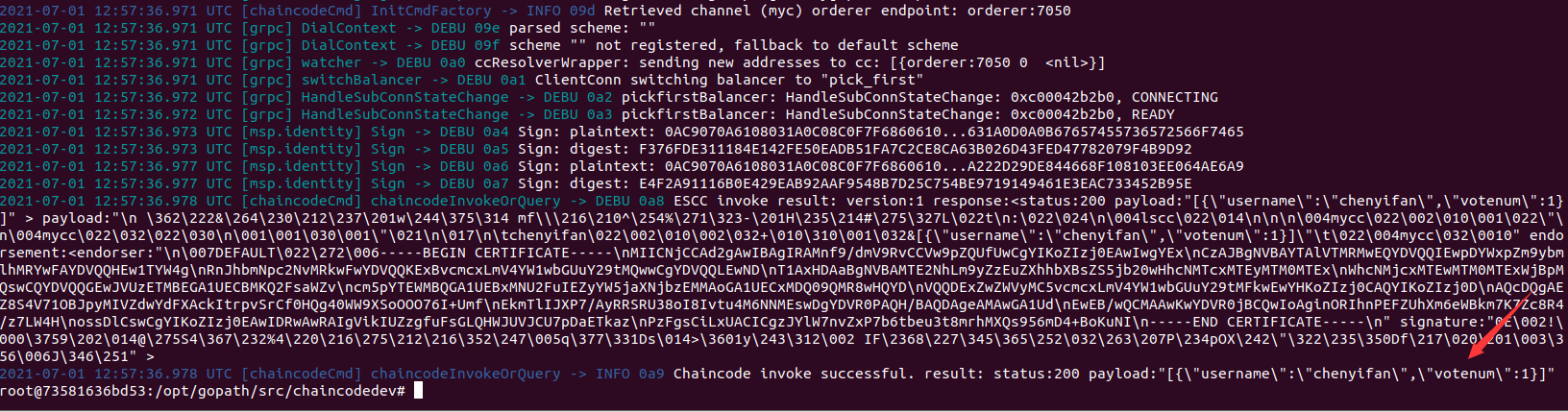
创建投票人信息，初始创建票数加一

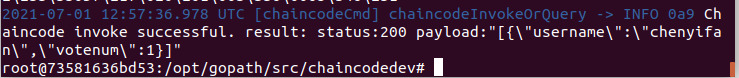
peer chaincode invoke -n mycc -c '{"Args":["voteUser", "chenyifan"]}' -C myc



查询投票人信息

peer chaincode invoke -n mycc -c '{"Args":["getUserVote"]}' -C myc





关闭网络。 docker-compose -f docker-compose-simple.yaml down