**1. 新环境初始化**

安装ssh   sudo apt-get install ssh

更改源   sudo vim /etc/apt/sources.list

:%s/us./cn./g

sudo apt-get update

安装python  sudo apt-get install python3.5

sudo apt-get install python-pip

sudo -H python -m pip install --upgrade pip

安装代理网络 sudo pip install shadowsocks

touch shadowsocks.json

{

"server" : "45.78.13.106",

"server\_port" : 44,

"localPort" : 1080,

"password" : "xxxxx",

"method" : "rc4-md5",

"remarks" : ""

}

后台运行sslocal

sudo mv shadowsocks.json /etc/

nohup sslocal -c /etc/shadowsocks.json &

sudo apt-get install polipo

#sudo vim /etc/polipo/config

    socksParentProxy = "127.0.0.1:1080"

    socksProxyType = socks5

    chunkHighMark = 50331648

    objectHighMark = 16384

    serverMaxSlots = 64

    serverSlots = 16

    serverSlots1 = 32

    proxyAddress = "0.0.0.0"

proxyPort = 8123

# sudo /etc/init.d/polipo restart

配置http代理

# export http\_proxy='http://127.0.0.1:8123'

# export https\_proxy='http://127.0.0.1:8123'

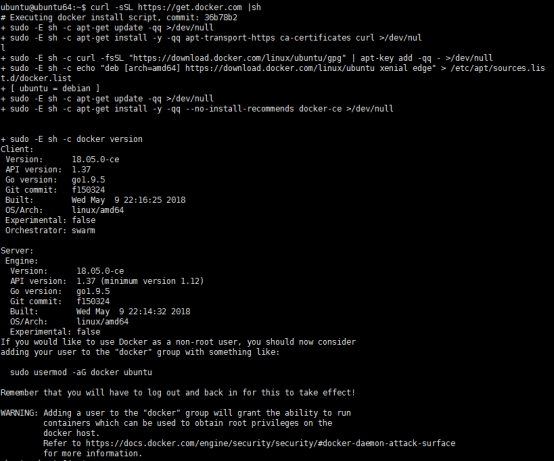
验证

# wget www.google.com

安装Docker

curl -sSL <https://get.docker.com> | sh

完成后如下图所示：版本为18.05



DAOCloud加速器

curl -sSL https://get.daocloud.io/daotools/set\_mirror.sh | sh -s http://85f9f519.m.daocloud.io





sudo vim /etc/default/docker

DOCKER\_OPTS="-s=aufs -r=true --api-cors-header='\*' -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock --registry-mirror=curl -sSL https://get.daocloud.io/daotools/set\_mirror.sh | sh -s http://85f9f519.m.daocloud.io"

加入用户到docker中

sudo usermod -aG docker fabric

启动docker服务

service docker restart

下载docker-compose

sudo curl -L https://github.com/docker/compose/releases/download/1.17.1/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

安装环境所需软件

go安装

[sudo apt-get install golang   |  卸载go  sudo apt-get purge golang-go]

#wget https://storage.googleapis.com/golang/go1.10.1.linux-amd64.tar.gz

#sudo tar -xzf go1.10.1.linux-amd64.tar.gz -C /usr/local

设置环境变量

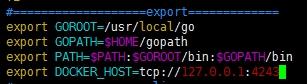
vim  ~/.bashrc

export GOROOT=/usr/local/go

export GOPATH=$HOME/gopath

export PATH=$PATH:$GOROOT/bin:$GOPATH/bin

export DOCKER\_HOST=tcp://127.0.0.1:4243



source .bashrc

**2.下载源码和镜像**

下载fabric代码

mkdir -p $GOPATH/src/github.com/hyperledger

cd $GOPATH/src/github.com/hyperledger

git clone https://github.com/hyperledger/fabric.git

下载fabric-ca代码

cd $GOPATH/src/github.com/hyperledger

git clone https://github.com/hyperledger/fabric-ca.git

下载案例快速搭建

cd  $GOPATH/src/github.com/hyperledger

git  clone https://github.com/hyperledger/fabric-samples.git

cd  fabric-samples/first-network

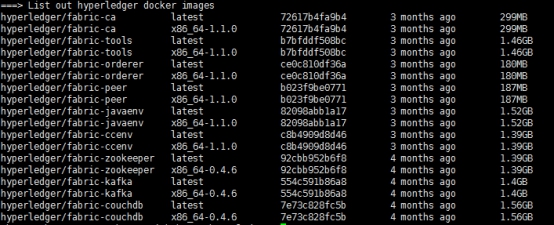
使用curl命令进行下载，之前确保外网可访问

sudo  curl **-**sSL https:**//**goo**.**gl**/**6wtTN5 **|** bash **-**s 1.1**.**0

如果上述操作有问题，可使用脚本直接下载镜像

|  |
| --- |
| #!/bin/bash  #  # Copyright IBM Corp. All Rights Reserved.  #  # SPDX-License-Identifier: Apache-2.0  #    # if version not passed in, default to latest released version  export VERSION=1.1.0  # if ca version not passed in, default to latest released version  export CA\_VERSION=$VERSION  # current version of thirdparty images (couchdb, kafka and zookeeper) released  export THIRDPARTY\_IMAGE\_VERSION=0.4.6  export ARCH=$(echo "$(uname -s|tr '[:upper:]' '[:lower:]'|sed 's/mingw64\_nt.\*/windows/')-$(uname -m | sed 's/x86\_64/amd64/g')" | awk '{print tolower($0)}')  #Set MARCH variable i.e ppc64le,s390x,x86\_64,i386  MARCH=`uname -m`    printHelp() {    echo "Usage: bootstrap.sh [<version>] [<ca\_version>] [-d -s -b]"    echo    echo "-d - bypass docker image download"    echo "-s - bypass fabric-samples repo clone"    echo "-b - bypass download of platform-specific binaries"    echo    echo "e.g. bootstrap.sh 1.1.1 -s"    echo "would download docker images and binaries for version 1.1.1"  }    dockerFabricPull() {    local FABRIC\_TAG=$1    for IMAGES in peer orderer ccenv javaenv tools; do        echo "==> FABRIC IMAGE: $IMAGES"        echo        docker pull hyperledger/fabric-$IMAGES:$FABRIC\_TAG        docker tag hyperledger/fabric-$IMAGES:$FABRIC\_TAG hyperledger/fabric-$IMAGES    done  }    dockerThirdPartyImagesPull() {    local THIRDPARTY\_TAG=$1    for IMAGES in couchdb kafka zookeeper; do        echo "==> THIRDPARTY DOCKER IMAGE: $IMAGES"        echo        docker pull hyperledger/fabric-$IMAGES:$THIRDPARTY\_TAG        docker tag hyperledger/fabric-$IMAGES:$THIRDPARTY\_TAG hyperledger/fabric-$IMAGES    done  }    dockerCaPull() {        local CA\_TAG=$1        echo "==> FABRIC CA IMAGE"        echo        docker pull hyperledger/fabric-ca:$CA\_TAG        docker tag hyperledger/fabric-ca:$CA\_TAG hyperledger/fabric-ca  }    : ${CA\_TAG:="$MARCH-$CA\_VERSION"}  : ${FABRIC\_TAG:="$MARCH-$VERSION"}  : ${THIRDPARTY\_TAG:="$MARCH-$THIRDPARTY\_IMAGE\_VERSION"}    samplesInstall() {    # clone (if needed) hyperledger/fabric-samples and checkout corresponding    # version to the binaries and docker images to be downloaded    if [ -d first-network ]; then      # if we are in the fabric-samples repo, checkout corresponding version      echo "===> Checking out v${VERSION} branch of hyperledger/fabric-samples"      git checkout v${VERSION}    elif [ -d fabric-samples ]; then      # if fabric-samples repo already cloned and in current directory,      # cd fabric-samples and checkout corresponding version      echo "===> Checking out v${VERSION} branch of hyperledger/fabric-samples"      cd fabric-samples && git checkout v${VERSION}    else      echo "===> Cloning hyperledger/fabric-samples repo and checkout v${VERSION}"      git clone -b master https://github.com/hyperledger/fabric-samples.git && cd fabric-samples && git checkout v${VERSION}    fi  }    binariesInstall() {    echo "===> Downloading version ${FABRIC\_TAG} platform specific fabric binaries"    curl https://nexus.hyperledger.org/content/repositories/releases/org/hyperledger/fabric/hyperledger-fabric/${ARCH}-${VERSION}/hyperledger-fabric-${ARCH}-${VERSION}.tar.gz | tar xz      echo "===> Downloading version ${CA\_TAG} platform specific fabric-ca-client binary"    curl https://nexus.hyperledger.org/content/repositories/releases/org/hyperledger/fabric-ca/hyperledger-fabric-ca/${ARCH}-${VERSION}/hyperledger-fabric-ca-${ARCH}-${VERSION}.tar.gz | tar xz    if [ $? != 0 ]; then       echo       echo "------> ${CA\_TAG} fabric-ca-client binary is not available to download  (Avaialble from 1.1.0-rc1) <----"       echo     fi  }    dockerInstall() {    which docker >& /dev/null    NODOCKER=$?    if [ "${NODOCKER}" == 0 ]; then        echo "===> Pulling fabric Images"        dockerFabricPull ${FABRIC\_TAG}        echo "===> Pulling fabric ca Image"        dockerCaPull ${CA\_TAG}        echo "===> Pulling thirdparty docker images"        dockerThirdPartyImagesPull ${THIRDPARTY\_TAG}        echo        echo "===> List out hyperledger docker images"        docker images | grep hyperledger\*    else      echo "========================================================="      echo "Docker not installed, bypassing download of Fabric images"      echo "========================================================="    fi  }    DOCKER=true  SAMPLES=true  BINARIES=true    # Parse commandline args pull out  # version and/or ca-version strings first  if echo $1 | grep -q '\d'; then    VERSION=$1;shift    if echo $1 | grep -q '\d'; then      CA\_VERSION=$1;shift    fi  fi    # then parse opts  while getopts "h?dsb" opt; do    case "$opt" in      h|\?)        printHelp        exit 0      ;;      d)  DOCKER=false      ;;      s)  SAMPLES=false      ;;      b)  BINARIES=false      ;;    esac  done    if [ "$SAMPLES" == "true" ]; then    echo    echo "Installing hyperledger/fabric-samples repo"    echo    samplesInstall  fi  if [ "$BINARIES" == "true" ]; then    echo    echo "Installing Hyperledger Fabric binaries"    echo    binariesInstall  fi  if [ "$DOCKER" == "true" ]; then    echo    echo "Installing Hyperledger Fabric docker images"    echo    dockerInstall  fi |

下载后如图所示



镜像下载完成后，会产生bin文件，下面有工具，将此工具的绝对路径添加到环境变量中

export PATH=下载位置的绝对路径/bin:$PATH

[export PATH=/home/ubuntu/gopath/src/github.com/hyperledger/fabric-samples/bin:$PATH]

**3.快速搭建过程**

cd ../first-network/

使用脚本byfn.sh生成初始环境信息

./byfn.sh -h  查看脚本信息

**3.1生成初始化信息**

执行脚本./byfn.sh  up会根据Docker Compose配置文件docker-compose-cli.yaml启动超级账本网络， 还会执行scripts/script.sh脚本安装和实例化链码， 并且执行简单链码调用和查询操作。

执行script.sh脚本时，会执行：创建通道，加入通道，更新通道信息，安装链码，实例化链码，链码调用，链码查询

创建通道

byfn.sh  generate  [根据crypto-config.yaml生成peer,order的MSP证书]

                  [根据configtx.yaml生成创世区块]

a. 使用cryptogen工具生成证书

命令：

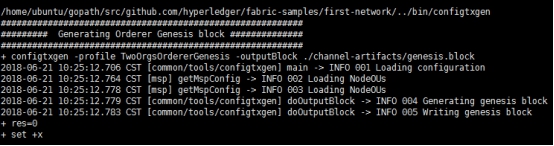
|  |
| --- |
| # cryptogen generate --config=./crypto-config.yaml org1.example.com org2.example.com |

b. 生成排序节点的初始区块

命令：

|  |
| --- |
| # configtxgen -profile TwoOrgsOrdererGenesis -outputBlock ./channel-artifacts/genesis.blcok  -profile 是根据configtx.yaml中的Profiles配置项描述如何生成组织：默认kafka  -outputBlock 指定创世区块的目录文件和名称 |

此处调用common/tools/configtxgen/main.go:main()->msp/configbuilder.go：getMspConfig()->common/tools/configtxgen/main.go:doOutputBlock()

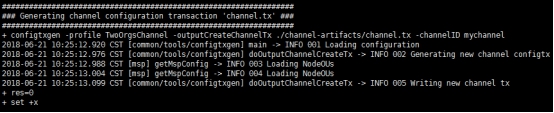


c. 生成通道交易配置文件

命令：

|  |
| --- |
| configtxgen  -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID mychannel |

此处调用：common/tools/configtxgen/main.go:main()->common/tools/configtxgen/main.go:doOutputChannelCreateTx()->msp/configbuilder.go:getMspConfig()->common/tools/configtxgen/main.go:doOutputChannelCreateTx()

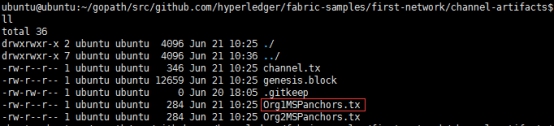


d. 生成Org1MSP/Org2MSP锚节点配置

命令：

|  |
| --- |
| configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID mychannel -asOrg Org1MSP  configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID mychannel -asOrg Org2MSP  -outputAnchorPeersUpdate:创建在configtx.yaml中指定的AnchorPeers |

此处调用：common/tools/configtxgen/main.go:main()->doOutputAnchorPeersUpdate()

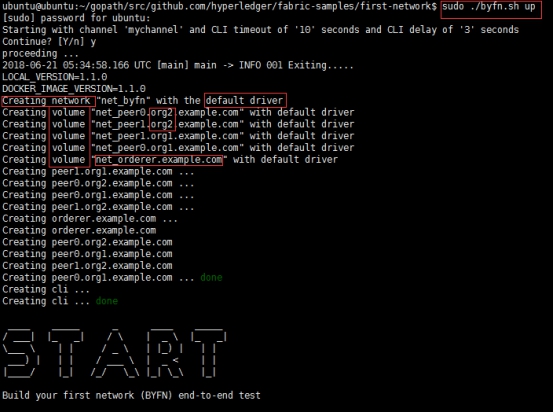


4.2 启动网络

启动网络时，由于docker权限的影响，所以此时运行时

sudo ./byfn.sh up.

开始执行脚本后，首先创建网络 net\_byfn



后续过程都基于fabric-1.1.0\examples\e2e\_cli\scripts\script.sh基本调用

**【安装过程中可以通过docker ps -a 查看容器中程序运行；安装chaincode后，通过docker image查看链码生成的镜像】**

启动完成后开始创建channel

条件：初始化时生成的根证书，证书，私钥，MSP等信息，及通道配置交易文件创建通道

命令：

|  |
| --- |
| peer channel create -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/channel.tx --tls true --cafile /opt/gopath.../tlsca.example.com-cert.pem |

此处主要调用peer/channel/channel.go的InitCmdFactory(),初始化Endorser和Orderer节点

channel创建完成后，Org1的peer1、peer2与Org2的peer1、peer2加入mychannel

条件：利用上面的根证书、证书、私钥、MSP等信息，加入channel

命令：

|  |
| --- |
| peer channel join -b mychannel.block |

此处调用的是peer/channel/join.go的executeJoin(cf \*ChannelCmdFactory)方法

更新Org1和Org2的anchor peer

命令：

|  |
| --- |
| peer channel update -o org1.example.com:7050 -c mychannel -f ./channel-artifacts/Org1MSPanchor.tx --tls true --cafile /opt/.../tlsca.example.com-cert.pem |

peer/main.go:channel.Cmd()->peer/channel/channel.go:updateCmd()->update()

给Org1、Org2的peer0安装链码

命令：

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

调用：peer/chaincode/chaincode.go:Cmd()->installCmd()->genChaincodeDeploymentSpec()->getChaincodeSpec()->checkChaincodeCmdParams()

emcc和vscc没有定义，使用默认配置进行安装

对Org2的peer0进行链码初始化

命令：

|  |
| --- |
| peer chaincode instantiate -o orderer.example.com:7050 --tls true --cafile /opt/.../tlsca.example.com-cert.pem -C mychannel -n mycc -l golang -v 1.0 -c '{"Args":["init","a","100","b","200"]}' -P 'OR ('\''Org1MSP.peer'\'','\''Org2MSP.peer'\'')' |

调用：peer/chaincode/chaincode.go:Cmd()->instantiateCmd()->genChaincodeDeploymentSpec()->getChaincodeSpec()->checkChaincodeCmdParams()

emcc和vscc没有定义，使用默认配置进行初始化

查询Org1的peer0信息

通过初始化Org2的peer0，查询Org1的peer0获取链码结果

命令：

|  |
| --- |
| peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}' |

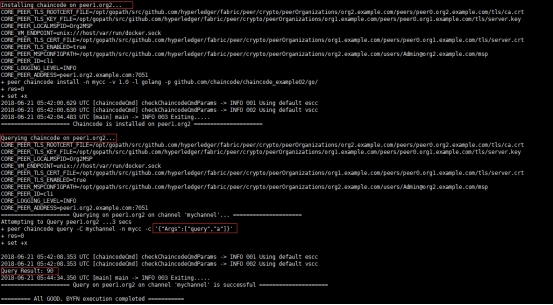
进行交易

命令：

|  |
| --- |
| peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/.../tlsca.example.com-cert.pem -C mychannel -n mycc -c '{"Args":["invoke","a","b","10"]}' |

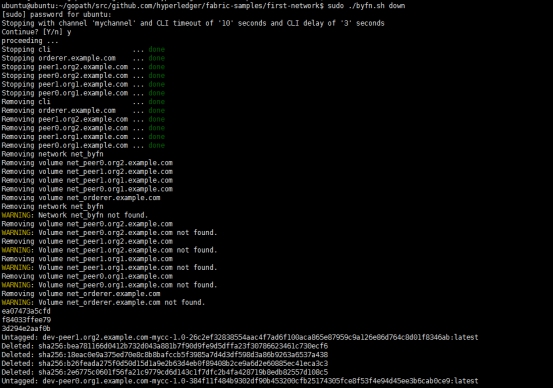
网络测试

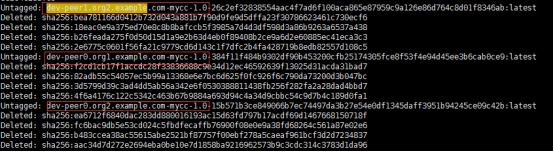
通过安装Org2.peer1链码，在Org2.peer1上查询a的值，得到正确的结果，说明安装链码时交易信息自动同步，网络正常



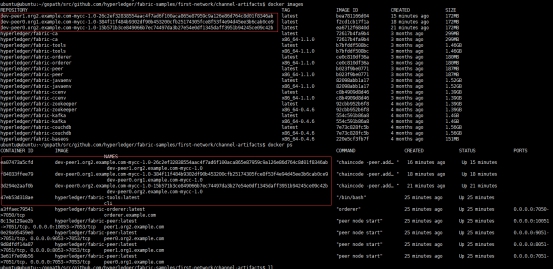
4.3关闭网络

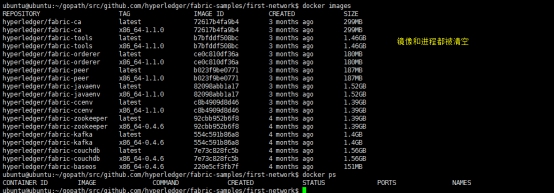
可以看到关闭peer节点，排序服务节点容器，链码容器，删除自动生成的镜像文件。





运行./byfn.sh up 后的docker显示信息与./byfn.sh down 后的docker显示信息





**3.2逐步搭建网络**

1. 使用cryptogen生成MSP证书

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

wpsFBC0.tmp.jpeg

2. 生成排序服务的创世区块

export "FABRIC\_CFG\_PATH=$PWD"

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

wpsFBC1.tmp.jpeg

3. 生成通道配置创世区块

export CHANNEL\_NAME=onechannel

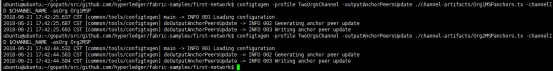
configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID $CHANNEL\_NAME

wpsFBC2.tmp.jpeg

4. 定义组织AnchorPeers

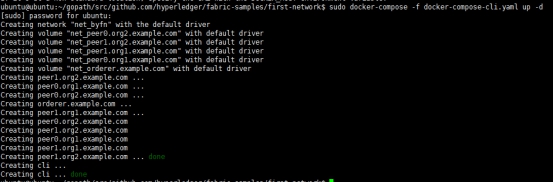
configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID $CHANNEL\_NAME -asOrg Org1MSP

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



启动网络

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



创建并加入网络通道

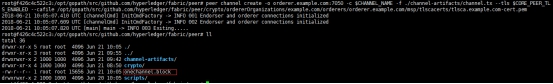
wpsFBD4.tmp.jpeg

wpsFBD5.tmp.jpeg

sudo docker exec -it cli bash

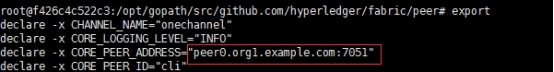
export “CHANNEL\_NAME=onechannel”

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



peer0.org1.example.com加入通道

由于默认为peer0.org1.example.com因此环境变量不需要更改



peer channel join -b onechannel.block

wpsFBD8.tmp.jpeg

peer channel list

wpsFBD9.tmp.jpeg

peer0.org2等加入通道

执行如下命令加入

|  |
| --- |
| export CORE\_PEER\_ADDRESS=peer0.org2.example.com:7051  export CORE\_PEER\_LOCALMSPID="Org2MSP"  export 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  export CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp  peer channel join -b onechannel.block    =================================  export CORE\_PEER\_ADDRESS=peer1.org1.example.com:7051  export CORE\_PEER\_LOCALMSPID="Org1MSP"  export 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  export CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp  peer channel join -b onechannel.block  ==========================  export CORE\_PEER\_ADDRESS=peer1.org2.example.com:7051  export CORE\_PEER\_LOCALMSPID="Org2MSP"  export 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  export CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp  peer channel join -b onechannel.block |



安装和实例化链码

Peer节点peer0.org1.example.com安装链码

|  |
| --- |
| export CORE\_PEER\_ADDRESS=peer0.org1.example.com:7051  // 设置MSP信息  export CORE\_PEER\_LOCALMSPID="Org1MSP"  export 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  export CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp  // 安装链码  peer chaincode install -n mycc -v 1.0 -p github.com/chaincode/chaincode\_example02/go |

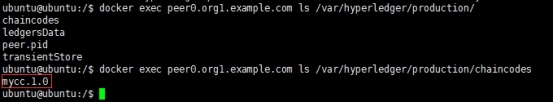
wpsFBDB.tmp.jpeg

安装的过程中如果遇到找不到文件的问题，可先搜索下example的文件位置，然后替换既可以了

wpsFBDC.tmp.jpeg

重启终端，进入peer0容器，查看链码是否安装

docker exec peer0.org1.example.com ls /var/hyperledger/production/chaincodes



其余Peer节点peer节点与peer0类似

链码的初始化

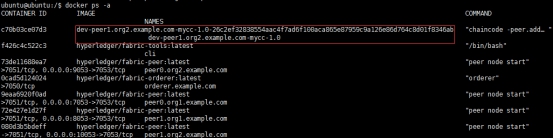
任意节点上都可

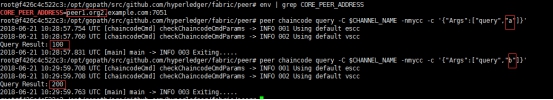
命令：

|  |
| --- |
| peer chaincode instantiate -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 onechannel -n mycc -l golang -v 1.0 -c '{"Args":["init","a","100","b","200"]}' -P 'OR ('\''Org1MSP.peer'\'','\''Org2MSP.peer'\'')' |

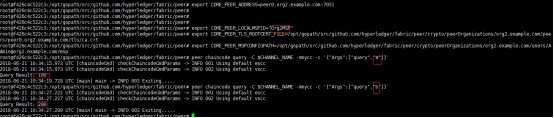
wpsFBDE.tmp.jpeg

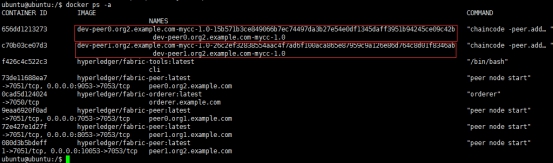
实例化后检查是否成功，使用docker ps -a ，dev-peer1.org2.example.com-mycc-1.0





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

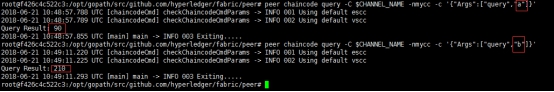




执行链码操作

|  |
| --- |
| peer chaincode invoke -oorderer.example.com:7050 --tls $CORE\_PEER\_TLS\_ENABLED --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 -c'{"Args":["invoke","a","b","10"]}' |
| 执行查询peer chaincode query -C $CHANNEL\_NAME -nmycc -c '{"Args":["query","a"]}' |

wpsFBF3.tmp.jpeg



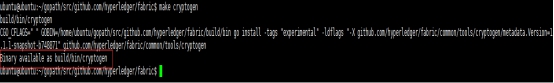
**4.FQA**

生成cryptogen等工具的方法

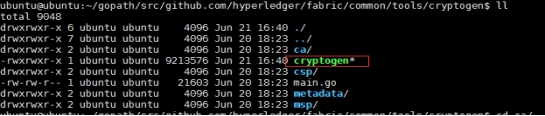
a.在fabric目录下 make cryptogen进行生成

如过遇到“vendor/github.com/miekg/pkcs11/pkcs11.go:26:18: fatal error: ltdl.h: No such file or directory”，则是系统需要安装

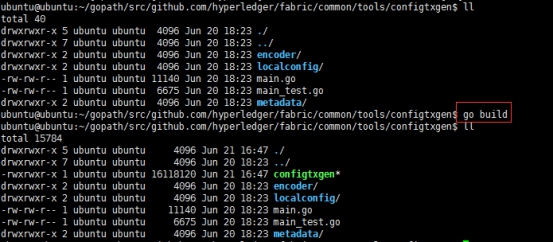
sudo apt-get install libtool libltdl7 libltdl-dev



b.直接在fabric/common/tools/cryptogen执行go build



生成configtxgen工具，同上



configtx.yaml文件中的一些说明

Profile结构中需要包括

Application:若无，创建通道会失败

Consortiums:若无，启动服务节点会失败

-inspectBlock :检查和输出创世区块的内容

-inspectChannelCreateTx :检查和输出创始区块的内容

-outputAnchorPeersUpdate:创建在configtx.yaml中指定的AnchorPeers

**5.KN**

tox概述

通用的虚拟环境管理和测试命令行工具。tox能够让我们在虚拟环境上定义出多套相互隔离的python环境（tox是openstack社区最基本的测试工具，比如python程序的兼容性，UT等）。它的目标是提供最先进的自动化打包，测试和发布功能。

优点：

1. 作为持续集成服务器的前端，减少测试工作所需时间

2. 检查软件包能否在不同python版本环境下正常安装

3. 在不同的环境中运行测试代码

travis和Jenkins

两者都是持续集成服务，区别是travis采用的是yaml脚本，简洁，明了，与github绑定，提供测试环境，可在代码变动后持续集成和打包。