基本环境配置：Go、Docker、Docker-Compose

### 1.在每个节点上下载镜像

cd profile

source download-dockerimages.sh -c x86\_64-1.0.0 -f x86\_64-1.0.0

### 2.启动orderer节点

cd profile

docker-compose -f docker-compose-orderer.yaml up -d

### 3.为自建社区启动couchdb

docker pull klaemo/couchdb

mkdir couchdb

docker run -p 5984:5984 -d --name my-couchdb -e COUCHDB\_USER=admin -e COUCHDB\_PASSWORD=password -v ~/couchdb:/opt/couchdb/data klaemo/couchdb

### 4.启动自建社区节点

（记得修改yaml文件的IP配置）

cd profile

docker-compose -f docker-compose-zijianshequ.yaml up –d

### 5.为科迅启动couchdb

docker pull klaemo/couchdb

mkdir couchdb

docker run -p 5984:5984 -d --name my-couchdb -e COUCHDB\_USER=admin -e COUCHDB\_PASSWORD=password -v ~/couchdb:/opt/couchdb/data klaemo/couchdb

### 6.启动科迅节点

（记得修改yaml文件的IP配置）

cd profile

docker-compose -f docker-compose-kexun.yaml up –d

### 7.创建channel

我们切换到peer0.org1.example.com服务器上，使用该服务器上的cli来运行创建Channel和运行ChainCode的操作。先用以下命令进入CLI内部Bash：

docker exec -it cli bash

创建Channel的命令是peer channel create，我们前面创建Channel的配置区块时，指定了Channel的名字是mychannel，那么这里我们必须创建同样名字的Channel。

ORDERER\_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

peer channel create -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/channel.tx --tls true --cafile $ORDERER\_CA

执行该命令后，系统会提示：

2017-10-30 18:30:35.210 UTC [channelCmd] readBlock -> DEBU 020 Received block:0

系统会在cli内部的当前目录创建一个mychannel.block文件，这个文件非常重要，接下来其他节点要加入这个Channel就必须使用这个文件。

### 8.各个peer加入channel

还是在peer0.org1的CLI上，我们要将这个Peer加入mychannel就很简单，只需要运行如下命令：

peer channel join -b mychannel.block

系统返回消息：

2017-10-30 18:40:21.405 UTC [channelCmd] executeJoin -> INFO 006 Peer joined the channel!

修改cli的环境变量，使其指向peer0.org2

[制代码](javascript:void(0);)

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

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:7051

peer channel join -b mychannel.block

[制代码](javascript:void(0);)

这样peer0.org1和peer0.org2就都已经加入channel了。

注：这一切都是我在peer0.org1的cli内完成的，这是由于加入channel依赖 [channel-ID].block文件，而且在peer0.org1服务器的cli内也留有了peer0.org2的证书文件，因此两个节点的加入都在一个cli内完成了。其实当一个节点创建channel以后可以把[channel-ID].block文件从容器内拷贝到主机，再分发给其他节点，这样其他节点就可以通过[channel-ID].block加入channel了。当cli容器被销毁时该容器下所有的文件都会被销毁，但是如果备份了[channel-ID].block，那么即使所有的peer和cli被销毁了，也可以再次通过[channel-ID].block加入channel、同步区块。

### 9.更新锚节点

关于AnchorPeer，其实我的每个组织只有一个peer节点，所以更新不更新锚节点也不重要。

对于Org1来说，peer0.org1是锚节点，我们需要切换到peer0.org1服务器上并更新锚节点：

[制代码](javascript:void(0);)

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

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

  peer channel update -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/Org1MSPanchors.tx --tls true --cafile $ORDERER\_CA

[制代码](javascript:void(0);)

另外对于Org2，peer0.org2是锚节点，切换到peer0.org2服务器上然后执行如下命令：

[制代码](javascript:void(0);)

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

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:7051

peer channel update -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/Org2MSPanchors.tx --tls true --cafile $ORDERER\_CA

[制代码](javascript:void(0);)

### 10.chaincode的安装与运行

以上，整个Fabric网络和Channel都准备完毕，接下来我们来安装和运行ChainCode。社区联盟的chaincode代码已经在上一章贴上了。

10.1安装chaincode

在cli上为每一个peer安装链上代码

，用peer chaincode install命令可以安装指定的ChainCode并对其命名：

[制代码](javascript:void(0);)

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

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

peer chaincode install -n mycc -v 1.0 -p github.com/hyperledger/fabric/examples/chaincode/go/community

[制代码](javascript:void(0);)

安装的过程其实就是对CLI中指定的代码进行编译打包，并把打包好的文件发送到Peer，等待接下来的实例化。

切换到peer0.org2服务器，给peer0.org2安装链上代码：

[制代码](javascript:void(0);)

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

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:7051

peer chaincode install -n mycc -v 1.0 -p github.com/hyperledger/fabric/examples/chaincode/go/community

[制代码](javascript:void(0);)

10.2实例化chaincode

实例化链上代码主要是在Peer所在的机器上对前面安装好的链上代码进行包装，生成对应Channel的Docker镜像和Docker容器。并且在实例化时我们可以指定背书策略。我们在peer0.org1服务器的cli内运行以下命令完成实例化：

[制代码](javascript:void(0);)

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

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

peer chaincode instantiate -o orderer.example.com:7050 --tls true --cafile $ORDERER\_CA -C mychannel -n mycc -v 1.0 -c '{"Args":["init"]}' -P "OR ('Org1MSP.member','Org2MSP.member')"

[制代码](javascript:void(0);)

使用docker ps可以看到有新的容器 dev-peer0.org1.example.com-mycc-1.0正在运行。

10.3在一个peer发起交易

我们可以调用initLedger功能，该功能会初始化两个帖子。对应的代码：

peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile $ORDERER\_CA -C mychannel -n mycc -c '{"Args":["initLedger"]}'

10.4在另一个节点上查询帖子

前面的操作都是在org1下面做的，那么处于同一个区块链（同一个Channel下）的org2，是否会看org1的更改呢？切换到peer0.org2服务器，由于mycc已经在前面org1的时候实例化了，也就是说对应的区块已经生成了，所以在org2不能再次初始化。我们直接运行查询命令：

[制代码](javascript:void(0);)

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

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:7051

peer chaincode query -C mychannel -n mycc -c '{"Args":["queryPost","POST1"]}'

[制代码](javascript:void(0);)

因为peer0.org2也需要生成Docker镜像，创建对应的容器，才能通过容器返回结果，所以等的时间会稍长一点。我们回到Ubuntu终端，执行docker ps，可以看到peer0.org2也多了一个容器：dev-peer0.org2.example.com-mycc-1.0