# 安装预置环境

先拉取Fabric源码

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

你还需要下载并安装Hyperledger Fabric Samples。你会注意到fabric-samples文件夹中包含了许多示例。我们将使用first-network这个例子。

使用git拉取文件

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

切换至1.0的版本。

git checkout -b release-1.0 origin/release-1.0

打开下载文件的目录

下载运行网络所需要的工具，cryptogen、configtxgen、configtxlator和peer。并将它们放置bin在当前工作目录的子目录中

curl -sSL https://goo.gl/kFFqh5 | bash -s 1.0.6

若次命令下载，报错如：curl: (7) Failed to connect to goo.gl port 443: Connection refused,请到此链接下载。

链接：https://pan.baidu.com/s/1fz0fFn-OL2NOStvP3cCuJQ 密码：adt6

现在让我们打开这个子目录。

cd first-network

# 快速运行脚本

我们提供一个完全注释的脚本byfn.sh，利用这些Docker镜像可以快速引导一个由4个代表2个不同组织的peer节点以及一个排序服务节点的Hyperledger fabric网络。它还将启动一个容器来运行一个将peer节点加入channel、部署实例化链码服务以及驱动已经部署的链码执行交易的脚本。

以下是该byfn.sh脚本的帮助文档：

./byfn.sh -h

Usage:

byfn.sh -m up|down|restart|generate [-c <channel name>] [-t <timeout>]

byfn.sh -h|--help (print this message)

-m <mode> - one of 'up', 'down', 'restart' or 'generate'

- 'up' - bring up the network with docker-compose up

- 'down' - bring up the network with docker-compose up

- 'restart' - bring up the network with docker-compose up

- 'generate' - generate required certificates and genesis block

-c <channel name> - config name to use (defaults to "mychannel")

-t <timeout> - CLI timeout duration **in** microseconds (defaults to 10000)

Typically, one would first generate the required certificates and

genesis block, **then** bring up the network. e.g.:

byfn.sh -m generate -c <channelname>

byfn.sh -m up -c <channelname>

如果你选择不提供channel名称，则脚本将使用默认名称mychannel。CLI超时参数（用-t标志指定）是一个可选值;如果你选择不设置它，那么CLI容器将会在脚本执行完之后退出。

# 脚本生成搭建网络前提

执行以下命令生成，输入y来执行描述的动作。第一步生成我们各种网络实体的所有证书和密钥，genesis block用于引导排序服务，以及配置Channel所需要的一组交易配置集合。

./byfn.sh -m generate

Generating certs and genesis block **for** with channel 'mychannel' and CLI timeout of '10000'

Continue (y/n)?y

# 脚本启动网络

接下来，你可以使用以下命令来启动整个网络。再试提示你是否继续。回答y：

./byfn.sh -m up

Starting with channel 'mychannel' and CLI timeout of '10000'

Continue (y/n)?y

# 脚本关闭网络

最后，让我们把它全部停下来，这样我们可以一步一步地探索网络设置。以下操作将关闭你的容器，移除加密材料和4个配置信息，并且从Docker仓库删除chaincode镜像。你将再一次被提示是否继续，回答y：

./byfn.sh -m down

Stopping with channel 'mychannel' and CLI timeout of '10000'

Continue (y/n)?y

此命令除了关闭网络，还会在本地将配置文件、密钥、证书等文件删除，也就是文件夹crypto-config和channel-artifacts内容删除。所以再起网络时，需要重新配置证书密钥文件。在后面将讲解关于底层工具和引导材料相关的更多信息，我们将浏览构建功能齐全的Hyperledger fabric网络的各种要求和步骤。

# 加密生成器

我们将使用cryptogen工具为我们生成各种网络实体的加密材料（x509证书）。这些证书是身份的代表，它们允许在我们的网络实体进行交流和交易时进行签名/验证身份验证。

Cryptogen消费一个包含网络拓扑的crypto-config.yaml(配置文件)，并允许我们为组织和属于这些组织的组件生成一组证书和密钥。每个组织都配置了唯一的根证书(ca-cert),它将特定组件（peers和orders）绑定到该组织。通过为每一个组织分配唯一的CA证书，我们正在模仿一个经典的网络，这个网络中的成员将使用自己的证书颁发机构。Hyperledger Fabric中的交易和通信是通过存储在keystore中的实体的私钥签名，然后通过公钥手段进行验证（signcerts）。

我们运行cryptogen工具，生成的证书和密钥将被保存到名为crypto-config的文件夹中。

# 配置交易生成器

configtxgen tool用于创建4个配置工作： order的genesis block, channel的channel configuration transaction, \* 以及两个anchor peer transactions一个对应一个Peer组织。

order block是一个ordering service的[创世区块](http://hyperledger-fabric.readthedocs.io/en/latest/glossary.html" \l "genesis-block)，channel transaction文件在Channel创建的时侯广播给order。anchor peer transactions，正如名称所示，指定了每个组织在此channel上的[Anchor peer](http://hyperledger-fabric.readthedocs.io/en/latest/glossary.html" \l "anchor-peer)。

Configtxgen使用一个包含示例网络的configtx.yaml文件。有3个成员-一个排序服务组织OrdererOrg以及两个节点组织（Org1&Org2）,每个组织管理和持有2个peer节点。该文件还指定了一个SampleConsortium的联盟，由上述2个节点组织构成。 请特别注意此文件顶部的”Profiles”部分。你会注意到我们有两个独特的标题。一个是orderer的创世区块-TwoOrgsOrdererGenesis-另一个是针对管道的TwoOrgsChannel。

这些标题很重要，因为在我们创建我们的工作的时侯她们将作为传递的参数。

注意

请注意我们的SampleConsortium在系统界别的配置文件中定义，然后由渠道级别配置文件引用。管道存在于联盟的范围内，所有的联盟必须定义在整个网络范围内。

此文件还包含两个值得注意的附加规格。首先，我们为每个组织指定了锚点节点（peer0.org1.example.com和peer0.org2.example.com）。其次，我们为每个成员指定MSP文件夹，用来存储每个组织在orderer genesis block中指定的根证书。这是一个关键的概念。现在任意和ordering service通信的网络实体都可以对其数字签名进行验证。

# 运行工具

首先，我们来运行cryptogen这个工具。我们的二进制文件在bin目录中，所以我们需要提供工具所在的相对路径。

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

接下来，我们需要告诉configtxgen工具需要提取的configtx.yaml所在的位置。我们会告诉它在我们当前所在工作目录：

首先，我们需要设置一个环境变量来告诉configtxgen哪里去寻找configtx.yaml。然后，我们将调用configtxgen工具去创建orderer genesis block：

export FABRIC\_CFG\_PATH=$PWD

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

接下来，我们需要创建channel transaction配置。请确保替换$CHANNEL\_NAME或者将CHANNEL\_NAME设置为整个说明中可以使用的环境变量：

export CHANNEL\_NAME=mychannel

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

接下来，我们将在正在构建的通道上定义Org1的anchor peer。请再次确认$CHANNEL\_NAME已被替换或者为以下命令设置了环境变量：

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

现在，我们将在同一个通道定义Org2的anchor peer：

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

# 启动网络

我们将利用docker-compose脚本来启动我们的区块链网络。docker-compose文件利用我们之前下载的镜像，并用以前生成的genesis.block来引导orderer。

在这里我们使用cli容器启动网络，启动的文件为docker-compose-cli.yaml。我们需要修改其中一个地方

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

# command: /bin/bash -c './scripts/script.sh ${CHANNEL\_NAME}; sleep $TIMEOUT' volumes

将command这一行注释。

如果没有注释，该脚本将在网络启动时执行所有命令，正如我们在[幕后发生的情况](http://hyperledger-fabric.readthedocs.io/en/latest/build_network.html" \l "behind-scenes)中所描述的那样。然而，我们想手动执行命令，以便公开每个调用的语法和功能。

适当地为TIMEOUT传递较高的值（以秒为单位）;默认情况下CLI容器将在60秒之后退出。

启动你的网络：

CHANNEL\_NAME=$CHANNEL\_NAME TIMEOUT=<pick\_a\_value> docker-compose -f docker-compose-cli.yaml up -d

如果要实时查看你的区块链网络的日志，请不要提供-d标志。如果你需要日志流，你需要打开第二个终端来执行CLI命令。

# 网络内的操作

## 设置环境变量

这里使用peer0.org1.example.com来对网络进行操作，我们需要使用下面给出四个环境变量来介绍我们的命令。为peer0.org1.example.com涉及的这些变量将被拷贝到CLI容器中，因此我们不需要复制它们。然而，如果你发送调用到其他的peer节点或者orderer，则需要相应地提供这些值。

命令如下：

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

切换到peer0.org2.example.com命令如下：

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

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

## 创建和加入channel

我们将使用docker exec命令进入CLI容器：

docker exec -it cli bash

如果成功，你将看到下列信息：

root@0d78bb69300d:/opt/gopath/src/github.com/hyperledger/fabric/peer#

回想以下，我们使用configtxgen工具生成信道配置-channel.tx。我们将这个配置作为请求的一部分传递给order。

注意-- cafile会作为命令的一部分。这是orderer的root cert的本地路径，允许我们去验证TLS握手。

我们使用-c标志指定channel的名字，-f标志指定配置交易。在这个例子中它是channel.tx，当然你也可以使用不同的名称，挂载你自己的交易配置。

创建channel的命令如下：

export CHANNEL\_NAME=mychannel

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

此命令返回一个创世区块-<channel-ID.block>-我们将使用它加入信道。它包含了channel.tx中的配置信息。

剩下的命令将会留在CLI容器内执行。你必须记住所有的命令必须在相应的环境变量下执行当目标节点是除了peer0.org1.example.com以外的节点。

现在让我们加入peer0.org1.example.com频道。

peer channel join -b mychannel.block

## 安装和实例化chaincode

chaincode是fabric网络中业务部分的核心，交易的具体实现，账本的修改查询都是通过chaincode来进行操作。因此，我们需要在每个会执行以及背书我们交易的peer节点安装chaincode，然后在信道上实例化chaincode。首先，在将示例代码安装到4个peer节点中的其中一个。这个命令将源代码放到peer节点的文件系统中。

安装命令如下：

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

接下来，在信道上实例化chaincode。这将初始化信道上的链码，设置链码的背书策略，为目标peer节点启动一个chaincode容器注意-P参数。这是我们需要指定的当这个chaincode的交易需要被验证的时侯的背书策略。

在下面的命令中，你会注意到我们指定-P "OR ('Org0MSP.member','Org1MSP.member')"作为背书策略。这意味着我们需要Org1或者Org2组织中的其中一个的节点的背书即可（即只有一个背书）。如果我们改变语法为AND那么我们就需要2个背书者。背书策略和chaincode在后面会详细介绍。

peer chaincode instantiate -o orderer.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 -v 1.0 -c '{"Args":["init","a", "100", "b","200"]}' -P "OR ('Org1MSP.member','Org2MSP.member')"

## 查询和调用chaincode

在chaincdoe实例化之后，便可以调用chaincdoe进行调用交易

查询操作：

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

调用操作：从a账户转10到b账户

peer chaincode invoke -o orderer.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"]}'

再次查询：

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

# 问题汇总

## 问题一

在执行./byfn.sh -m generate时出现以下错误

\* '' has invalid keys: capabilities

\* 'Profiles[TwoOrgsChannel].Application' has invalid keys: Capabilities

\* 'Profiles[TwoOrgsOrdererGenesis]' has invalid keys: Capabilities

\* 'Profiles[TwoOrgsOrdererGenesis].Orderer' has invalid keys: Capabilities

Failed to generate orderer genesis block...

是版本的问题，将fabric-samples的版本切换至1.0

## 问题二

在启动网络的时候，如通过脚本./byfn.sh -m up，或者通过docker-compose up 启动网络时，报以下错误，是因为找不到order节点。

UTC [grpc] Printf -> DEBU 003 grpc: addrConn.resetTransport failed to create client transport: connection error: desc = "transport: Error while dialing dial tcp 211.139.178.49:7050: i/o timeout"; Reconnecting to {orderer.example.com:7050 <nil>}

修改宿主机hosts， /etc/host,在实际环境中，建议通过配置 DNS 而不是修改 /etc/hosts 文件。

添加 127.0.0.1  orderer.example.com

修改 base/peer-base.yaml，添加 volumes：

volumes:

-/etc/hosts:/etc/hosts

## 问题三

在创建channel时，报以下的错误：

2017-05-22 17:36:52.235 UTC [logging] InitFromViper -> DEBU 001 Setting default logging level to DEBUG for command 'channel'

2017-05-22 17:36:52.235 UTC [msp] GetLocalMSP -> DEBU 002 Returning existing local MSP

2017-05-22 17:36:52.236 UTC [msp] GetDefaultSigningIdentity -> DEBU 003 Obtaining default signing identity

Error: Got unexpected status: BAD\_REQUEST

Usage:

peer channel create [flags]

可能是因为正在创建的channel名称已经存在。可在cli容器中通过peer channel list 命令查看节点中以存在的channel。若确认时因为channel名称冲突，解决方案有通过./byfn.sh -m down 删除容器和证书、密钥、交易文件。然后按照前面步骤再来一遍。还可以通过修改channel名称的方式创建一个不冲突的channel（修改名称的方式还未测试）

## 问题四

在peer channel join -b mychannel.block时报以下错误：

Error: Error getting endorser client channel: PER:404 - Error trying to connect to local peer

/opt/go/src/runtime/proc.go:192 runtime.main

/opt/go/src/runtime/asm\_amd64.s:2087 runtime.goexit

Caused by: context deadline exceeded

Usage:

peer channel join [flags]

是因为在进入cli容器时没有设置cli容器进入的节点信息，证书文件位置、组织名称、节点访问地址等。可在cli容器中输入以下内容解决问题。

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

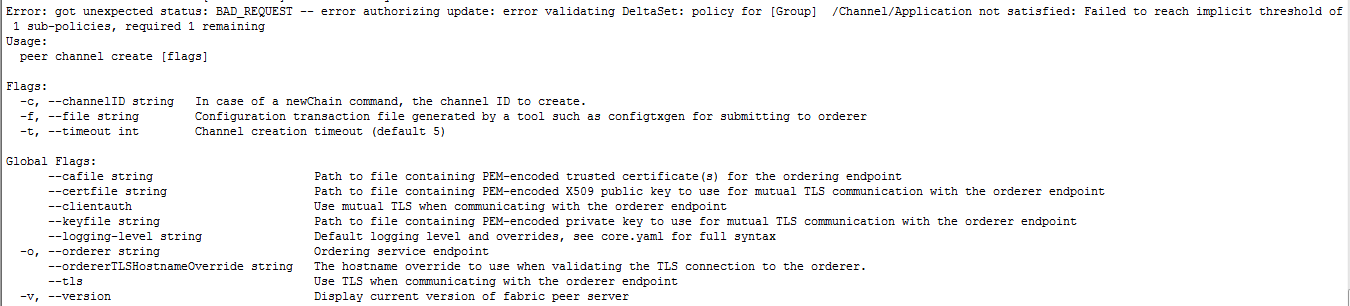
CORE\_PEER\_ADDRESS=peer0.org1.example.com:7051

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

## 问题五

在创建channel时，报以下的错误：



表明peer创建channel不符合交易配置文件的定义。例如在configtx.yaml文件中，定义mychannel的配置如下：

TwoOrgsChannel:

Consortium: SampleConsortium

Application:

<<: \*ApplicationDefaults

Organizations:

- \*Org1

- \*Org2

Capabilities:

<<: \*ApplicationCapabilities

配置文件中指定了TwoOrgsChannel对应的channel是由org1和org2进行管理的，但是在peer创建此channel时，peer和操作者的权限不属于org1或者org2，则会报上述错误。

可在cli容器中设置环境变量，然后peer节点和操作者权限更换至org1或者org2

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

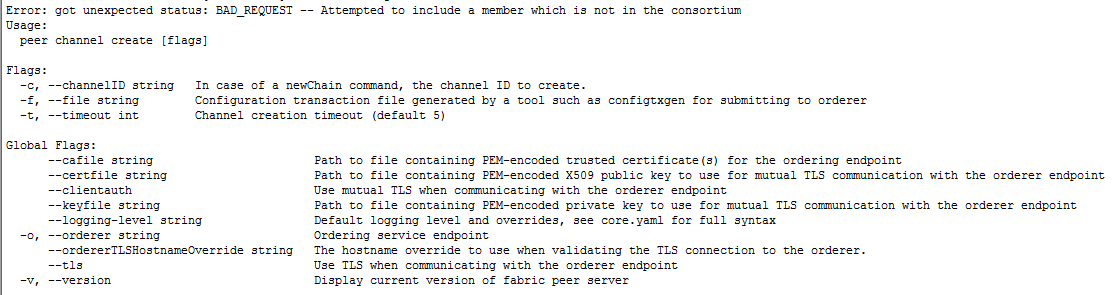
CORE\_PEER\_ADDRESS=peer0.org1.example.com:7051

CORE\_PEER\_LOCALMSPID="Org1MSP"

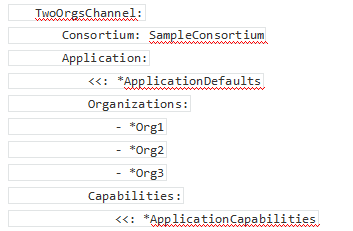
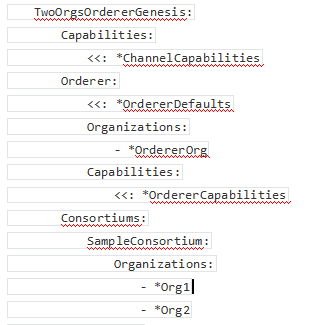
CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

## 问题六

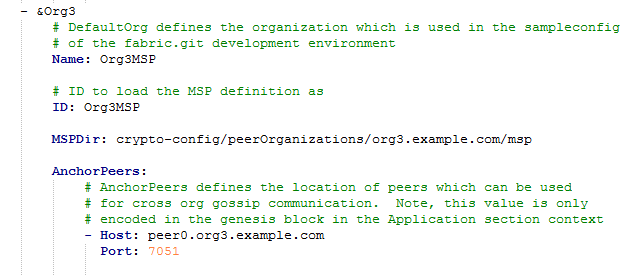
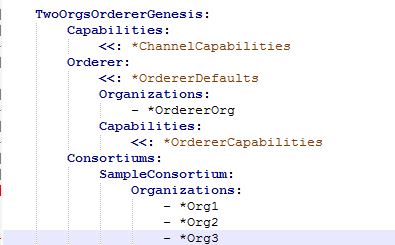
在创建channel时，报以下的错误：



表明在创建此channel使用的交易配置文件出现了问题，具体问题是交易配置文件configtx.yaml中使用了fabric网络中不存在的org（组织）如：



配置文件中指定了TwoOrgsChannel对应的channel是由org1、org2、org3共同进行管理的。但是org3并未TwoOrgsOrdererGenesis,也就是fabric的创世区块配置文件中定义，所以在peer创建此channel时使用到了并不存在于网络的org3组织，便有上述错误出现。解决办法是先修改crypto-config.yaml文件，添加org3组织，使用Cryptogen工具消费crypto-config.yaml生成org3对应的证书和密钥。再修改configtx.yaml，在创世区块添加org3的声明和在下面添加org3对应的证书密钥文件位置。然后再重新执行生成交易配置文件的命令。这样就能把org3添加至fabric网络中。修改的configtx.yaml文件部分如下：



## 总结

作者运行第一个fabrci网络时遇到了很多错误，大多是因为不细心，跳跃式看教程跳跃式操作导致的，所以建议，对照官网的教程，脚踏实地，一步一步按教程的操作来。

以下是在从进入文件夹first-network后，建立网络、创建密钥、执行链码等一系列操作的命令。