使用Fabric的测试网络

下载Hyperledger Fabric Docker镜像和示例后，您将可以使用以`fabric-samples`代码库中提供的脚本来部署测试网络。 您可以通过在本地计算机上运行节点来使用测试网络以了解Fabric。更有经验的开发人员可以使用 网络测试其智能合约和应用程序。该网络工具仅用作教育与测试目的。它不应该用作部署产品网络的模板。 该测试网络在Fabric v2.0中被引入作为`first-network`示例的长期替代。该示例网络使用Docker Compose部署了一个Fabric网络。 因为这些节点是隔离在Docker Compose网络中的，所以测试网络不配置为连接到其他正在运行的fabric节点。

注意: 这些指导已基于最新的稳定版Docker镜像和提供的tar文件中的预编译的安装软件进行验证。 如果您使用当前的master分支的镜像或工具运行这些命令，则可能会遇到错误。

开始之前

在运行测试网络之前，您需要克隆`fabric-samples`代码库并下载Fabric镜像。确保已安装 的 准备阶段 和 安装示例、二进制和 Docker 镜像.

启动测试网络

您可以在`fabric-samples`代码库的`test-network`目录中找到启动网络的脚本。 使用以下命令导航至测试网络目录：

```
cd fabric-samples/test-network
```

在此目录中，您可以找到带注释的脚本`network.sh`，该脚本在本地计算机上使用Docker镜像建立Fabric网络。 你可以运行`./network.sh -h`以打印脚本帮助文本：

```
Usage:
  network.sh <Mode> [Flags]
    Modes:
      up - bring up fabric orderer and peer nodes. No channel is created
      up createChannel - bring up fabric network with one channel
      createChannel - create and join a channel after the network is created
      deployCC - deploy the asset transfer basic chaincode on the channel or specify
      down - clear the network with docker-compose down
      restart - restart the network

    Flags:
    -ca <use CAs> -  create Certificate Authorities to generate the crypto material
    -c <channel name> - channel name to use (defaults to "mychannel")
    -s <dbtype> - the database backend to use: goleveldb (default) or couchdb
    -r <max retry> - CLI times out after certain number of attempts (defaults to 5)
    -d <delay> - delay duration in seconds (defaults to 3)
    -ccn <name> - the short name of the chaincode to deploy: basic (default),ledger, private, sec
    -ccl <language> - the programming language of the chaincode to deploy: go (default), java, ja
    -ccv <version>  - chaincode version. 1.0 (default)
    -ccs <sequence>  - chaincode definition sequence. Must be an integer, 1 (default), 2, 3, etc
    -ccp <path>  - Optional, chaincode path. Path to the chaincode. When provided the -ccn will b
    -cci <fcn name>  - Optional, chaincode init required function to invoke. When provided this f
    -i <imagetag> - the tag to be used to launch the network (defaults to "latest")
    -cai <ca_imagetag> - the image tag to be used for CA (defaults to "latest")
    -verbose - verbose mode
    -h - print this message
```

```
Possible Mode and flag combinations
  up -ca -c -r -d -s -i -verbose
  up createChannel -ca -c -r -d -s -i -verbose
  createChannel -c -r -d -verbose
  deployCC -ccn -ccl -ccv -ccs -ccp -cci -r -d -verbose

Taking all defaults:
  network.sh up

Examples:
  network.sh up createChannel -ca -c mychannel -s couchdb -i 2.0.0
  network.sh createChannel -c channelName
  network.sh deployCC -ccn basic -ccl javascript
```

在`test-network`目录中，运行以下命令删除先前运行的所有容器或工程：

```
./network.sh down
```

然后，您可以通过执行以下命令来启动网络。如果您尝试从另一个目录运行脚本，则会遇到问题：

```
./network.sh up
```

此命令创建一个由两个对等节点和一个排序节点组成的Fabric网络。 运行`./network.sh up`时没有创建任何channel， 虽然我们将在后面的步骤实现。 如果命令执行成功，您将看到已创建的节点的日志：

```
Creating network "net_test" with the default driver
Creating volume "net_orderer.example.com" with default driver
Creating volume "net_peer0.org1.example.com" with default driver
Creating volume "net_peer0.org2.example.com" with default driver
Creating orderer.example.com    ... done
Creating peer0.org2.example.com ... done
Creating peer0.org1.example.com ... done
CONTAINER ID        IMAGE                                  COMMAND             CREATED              S
8d0c74b9d6af        hyperledger/fabric-orderer:latest   "orderer"           4 seconds ago        U
ea1cf82b5b99        hyperledger/fabric-peer:latest      "peer node start"   4 seconds ago        U
cd8d9b23cb56        hyperledger/fabric-peer:latest      "peer node start"   4 seconds ago        U
```

如果未得到此结果，请跳至故障排除 寻求可能出现问题的帮助。 默认情况下，网络使用 cryptogen工具来建立网络。 但是，您也可以 通过证书颁发机构建立网络。

测试网络的组成部分

部署测试网络后，您可能需要一些时间来检查其网络组件。 运行以下命令以列出所有正在您的计算机上运行的Docker容器。 您应该看到由`network.sh`脚本创建的三个节点：

```
docker ps -a
```

与Fabric网络互动的每个节点和用户都必须属于一个网络成员的组织。 Fabric网络成员的所有组织通常称为联盟(consortium)。 测试网络有两个联盟成员，Org1和Org2。 该网络还包括一个维护网络排序服务的排序组织。

Peer 节点 是任何Fabric网络的基本组件。 对等节点存储区块链账本并在进行交易之前对其进行验证。 同行运行包含业务用于管理区块链账本的智能合约上的业务逻辑。

网络中的每个对等方都必须属于该联盟的成员。 在测试网络里，每个组织各自运营一个对等节点，`peer0.org1.example.com` 和 `peer0.org2.example.com`.

每个Fabric网络还包括一个排序服务。 虽然对等节点验证交易并将交易块添加到区块链账本，他们不决定交易顺序或包含他们进入新的区块。 在分布式网络上，对等点可能运行得很远彼此之间没有什么共同点，并且对何时创建事务没有共同的看法。 在交易顺序上达成共识是一个代价高昂的过程，为同伴增加开销。

排序服务允许对等节点专注于验证交易并将它们提交到账本。 排序节点从客户那里收到认可的交易后，他们就交易顺序达成共识，然后添加区块。 这些区块之后被分配给添加这些区块到账本的对等节点。 排序节点还可以操作定义Fabric网络的功能的系统通道，例如如何制作块以及节点可以使用的Fabric版本。 系统通道定义了哪个组织是该联盟的成员。

该示例网络使用一个单节点Raft排序服务，该服务由排序组织运行。 您可以看到在您机器上正在运行的排序节点 `orderer.example.com` 。 虽然测试网络仅使用单节点排序服务，一个真实的网络将有多个排序节点，由一个或多个多个排序者组织操作。 不同的排序节点将使用Raft共识算法达成跨交易顺序的共识网络。

创建一个通道

现在我们的机器上正在运行对等节点和排序节点， 我们可以使用脚本创建用于在Org1和Org2之间进行交易的Fabric通道。 通道是特定网络成员之间的专用通信层。通道只能由被邀请加入通道的组织使用，并且对网络的其他成员不可见。 每个通道都有一个单独的区块链账本。被邀请的组织“加入”他们的对等节点来存储其通道账本并验证交易。

您可以使用 `network.sh` 脚本在Org1和Org2之间创建通道并加入他们的对等节点。 运行以下命令以创建一个默认名称为“ mychannel”的通道：

```
./network.sh createChannel
```

如果命令成功执行，您将看到以下消息打印在您的日志：

```
========= Channel successfully joined ===========
```

您也可以使用channel标志创建具有自定义名称的通道。 作为一个例子，以下命令将创建一个名为 `channel1` 的通道：

```
./network.sh createChannel -c channel1
```

通道标志还允许您创建多个不同名称的多个通道。 创建 `mychannel` 或 `channel1` 之后，您可以使用下面的命令创建另一个名为 `channel2` 的通道：

```
./network.sh createChannel -c channel2
```

如果您想一步建立网络并创建频道，则可以使用 `up` 和 `createChannel` 模式一起：

```
./network.sh up createChannel
```

在通道启动一个链码

创建通道后，您可以开始使用智能合约与通道账本交互。 智能合约包含管理区块链账本上资产的业务逻辑。 由成员运行的应用程序网络可以在账本上调用智能合约创建，更改和转让这些资产。 应用程序还通过智能合约查询，以在分类帐上读取数据。

为确保交易有效，使用智能合约创建的交易通常需要由多个组织签名才能提交到通道账本。 多个签名是Fabric信任模型不可或缺的一部分。 一项交易需要多次背书，以防止一个通道上的单一组织使用通道不同意的业务逻辑篡改其对等节点的分类账本。 要签署交易，每个组织都需要调用并在其对等节点上执行智能合约，然后签署交易的输出。 如果输出是一致的并且已经有足够的组织签名，则可以将交易提交到账本。 该政策被称为背书政策，指定需要执行智能交易的通道上的已设置组织合同，针对每个链码设置为链码定义的一部分。

在Fabric中，智能合约作为链码以软件包的形式部署在网络上。 链码安装在组织的对等节点上，然后部署到某个通道，然后可以在该通道中用于认可交易和区块链账本交互。 在将链码部署到通道前，该频道的成员需要就链码定义达成共识，建立链码治理。 何时达到要求数量的组织同意后，链码定义可以提交给通道，并且可以使用链码了。

使用 `network.sh` 创建频道后，您可以使用以下命令在通道上启动链码：

```
./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go -ccl go
```

`deployCC` 子命令将在 `peer0.org1.example.com` 和 `peer0.org2.example.com` 上安装 **asset-transfer (basic)** 链码。 然后在使用通道标志（或 `mychannel` 如果未指定通道）的通道上部署指定的通道的链码。 如果您第一次部署一套链码，脚本将安装链码的依赖项。默认情况下，脚本安装Go版本的 asset-transfer (basic) 链码。 但是您可以使用语言便签 `-l`，用于安装 Java 或 javascript 版本的链码。 您可以在 `fabric-samples` 目录的 `asset-transfer-basic` 文件夹中找到 asset-transfer (basic) 链码。 此目录包含作为案例和用来突显 Fabric 特征的样本链码。

与网络交互

在您启用测试网络后，可以使用 `peer` CLI与您的网络进行交互。 `peer` CLI允许您调用已部署的智能合约，更新通道，或安装和部署新的智能合约。

确保您正在从 `test-network` 目录进行操作。 如果你按照说明安装示例，二进制文件和Docker映像， 您可以在 `fabric-samples` 代码库的 `bin` 文件夹中找到 `peer` 二进制文件。 使用以下命令将这些二进制文件添加到您的CLI路径：

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

您还需要将 `fabric-samples` 代码库中的 `FABRIC_CFG_PATH` 设置为指向其中的 `core.yaml` 文件：

```
export FABRIC_CFG_PATH=$PWD/../config/
```

现在，您可以设置环境变量，以允许您作为Org1操作 `peer` CLI：

```
# Environment variables for Org1

export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admi
export CORE_PEER_ADDRESS=localhost:7051
```

`CORE_PEER_TLS_ROOTCERT_FILE` 和 `CORE_PEER_MSPCONFIGPATH` 环境变量指向Org1的 `organizations` 文件夹中的的加密材料。 如果您使用 `./network.sh deployCC -ccl go` 安装和启动 asset-transfer (basic) 链码，您可以调用链码（Go）的 `InitLedger` 方法来赋予一些账本上的初始资产（如果使用 typescript 或者 javascript，例如 `./network.sh deployCC -l javascript`，你会调用相关链码的 `initLedger` 功能）。 运行以下命令用一些资产来初始化账本：

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --
```

如果命令成功，您将观察到类似以下的输出：

```
-> INFO 001 Chaincode invoke successful. result: status:200
```

现在你可以用你的 CLI 工具来查询账本。运行以下指令来获取添加到通道账本的资产列表：

```
peer chaincode query -C mychannel -n basic -c '{"Args":["GetAllAssets"]}'
```

如果成功，您将看到以下输出：

```
[
  {"ID": "asset1", "color": "blue", "size": 5, "owner": "Tomoko", "appraisedValue": 300},
  {"ID": "asset2", "color": "red", "size": 5, "owner": "Brad", "appraisedValue": 400},
  {"ID": "asset3", "color": "green", "size": 10, "owner": "Jin Soo", "appraisedValue": 500},
  {"ID": "asset4", "color": "yellow", "size": 10, "owner": "Max", "appraisedValue": 600},
  {"ID": "asset5", "color": "black", "size": 15, "owner": "Adriana", "appraisedValue": 700},
  {"ID": "asset6", "color": "white", "size": 15, "owner": "Michel", "appraisedValue": 800}
]
```

当一个网络成员希望在账本上转一些或者改变一些资产，链码会被调用。使用以下的指令来通过调用 asset-transfer (basic) 链码改变账本上的资产所有者：

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --
```

如果命令成功，您应该看到以下响应：

```
2019-12-04 17:38:21.048 EST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke su
```

因为 asset-transfer (basic) 链码的背书策略需要交易同时被 Org1 和 Org2 签名，链码调用指令需要使用 `--peerAddresses` 标签来指向 `peer0.org1.example.com` 和 `peer0.org2.example.com`。因为网络的 TLS 被开启，指令也需要用 `--tlsRootCertFiles` 标签指向每个 peer 节点的 TLS 证书。

调用链码之后，我们可以使用另一个查询来查看调用如何改变了区块链账本的资产。因为我们已经查询了 Org1 的 peer，我们可以把这个查询链码的机会通过 Org2 的 peer 来运行。设置以下的环境变量来操作 Org2：

```
# Environment variables for Org2

export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org2MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admi
export CORE_PEER_ADDRESS=localhost:9051
```

你可以查询运行在 `peer0.org2.example.com` asset-transfer (basic) 链码：

```
peer chaincode query -C mychannel -n basic -c '{"Args":["ReadAsset","asset6"]}'
```

结果显示 `"asset6"` 转给了 Christopher:

```
{"ID":"asset6","color":"white","size":15,"owner":"Christopher","appraisedValue":800}
```

关停网络

使用完测试网络后，您可以使用以下命令关闭网络：

```
./network.sh down
```

该命令将停止并删除节点和链码容器，删除组织加密材料，并从Docker Registry移除链码镜像。 该命令还删除之前运行的通道项目和docker卷。如果您遇到任何问题，还允许您再次运行 `./ network.sh up`。

下一步

既然您已经使用测试网络在您的本地计算机上部署了Hyperledger Fabric，您可以使用教程来开始开发自己的解决方案：

- 使用将智能合约部署到通道 教程了解如何来将自己的智能合约部署到测试网络。
- 访问编写您的第一个应用程序 教程了解如何从您的客户端程序使用Fabric SDK提供的API调用智能合约。
- 如果您准备将更复杂的智能合约部署到网络，请跟随商业票据教程 探索两个组织使用区块链网络进行商业票据交易的用例。

您可以在教程页上找到Fabric教程的完整列表。

使用认证机构建立网络

Hyperledger Fabric使用公钥基础设施(PKI)来验证所有网络参与者的行为。 每个节点，网络管理员和用户提交的交易需要具有公共证书和私钥以验证其身份。 这些身份必须具有有效的信任根源，该证书是由作为网络中的成员组织颁发的。 `network.sh` 脚本在创建对等和排序节点之前创建所有部署和操作网络所有需要的加密材料。

默认情况下，脚本使用cryptogen工具创建证书和密钥。 该工具用于开发和测试，并且可以快速为具有有效根信任的Fabric组织创建所需的加密材料。 当您运行 `./network.sh up` 时，您会看到cryptogen工具正在创建Org1，Org2和Orderer Org的证书和密钥。

```
creating Org1, Org2, and ordering service organization with crypto from 'cryptogen'

/Usr/fabric-samples/test-network/../bin/cryptogen

############################################################
##### Generate certificates using cryptogen tool #########
############################################################

############################################################
############ Create Org1 Identities ######################
############################################################
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org1.yaml --output=organiza
org1.example.com
+ res=0
+ set +x
############################################################
############ Create Org2 Identities ######################
############################################################
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org2.yaml --output=organiza
org2.example.com
+ res=0
+ set +x
############################################################
############ Create Orderer Org Identities ################
############################################################
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-orderer.yaml --output=organ
+ res=0
+ set +x
```

测试网络脚本还提供了使用证书颁发机构（CA）的网络的启动选项。 在产品网络中，每个组织操作一个CA（或多个中间CA）来创建属于他们的组织身份。 所有由该组织运行的CA创建的身份享有相同的组织信任根源。 虽然花费的时间比使用cryptogen多，使用CA建立测试网络，提供了在产品中部署网络的指导。 部署CA还可以让您注册Fabric SDK的客户端身份，并为您的应用程序创建证书和私钥。

如果您想使用Fabric CA建立网络，请首先运行以下命令关停所有正在运行的网络：

```
./network.sh down
```

然后，您可以使用CA标志启动网络：

```
./network.sh up -ca
```

执行命令后，您可以看到脚本启动了三个CA，每个网络中的组织一个。

```
###############################################################
##### Generate certificates using Fabric CA's ############
###############################################################
Creating network "net_default" with the default driver
Creating ca_org2    ... done
Creating ca_org1    ... done
Creating ca_orderer ... done
```

值得花一些时间检查`/ network.sh`脚本部署CA之后生成的日志。 测试网络使用Fabric CA客户端以每个组织的CA注册节点和用户身份。 之后这个脚本使用enroll命令为每个身份生成一个MSP文件夹。 MSP文件夹包含每个身份的证书和私钥，以及在运营CA的组织中建立身份的角色和成员身份。 您可以使用以下命令来检查Org1管理员用户的MSP文件夹：

```
tree organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/
```

该命令将显示MSP文件夹的结构和配置文件：

```
organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/
└── msp
    ├── IssuerPublicKey
    ├── IssuerRevocationPublicKey
    ├── cacerts
    │   └── localhost-7054-ca-org1.pem
    ├── config.yaml
    ├── keystore
    │   └── 58e81e6f1ee8930df46841bf88c22a08ae53c1332319854608539ee78ed2fd65_sk
    ├── signcerts
    │   └── cert.pem
    └── user
```

您可以在`signcerts`文件夹中找到管理员用户的证书，然后在`keystore`文件夹中找到私钥。 要了解有关MSP的更多信息，请参阅成员服务提供者概念主题。

cryptogen和Fabric CA都为每个组织在`organizations`文件夹中生成加密材料。 您可以在`organizations/fabric-ca`目录中的`registerEnroll.sh`脚本中找到用于设置网络的命令。 要了解更多有关如何使用Fabric CA部署Fabric网络的信息，请访问Fabric CA操作指南。 您可以通过访问identity和membership概念主题了解有关Fabric如何使用PKI的更多信息。

幕后发生了什么?

如果您有兴趣了解有关示例网络的更多信息，则可以调查`test-network`目录中的文件和脚本。 下面的步骤提供了有关在您发出`./network.sh up`命令时会发生什么情况的导览。

- `./ network.sh`为两个对等组织和排序组织创建证书和密钥。 默认情况下，脚本利用cryptogen工具使用位于`organizations/cryptogen`文件夹中的配置文件。 如果使用`-ca`标志创建证书颁发机构，则脚本使用Fabric CA服务器配置文件和位于`organizations/fabric-ca`文件夹的`registerEnroll.sh`脚本。 cryptogen和Fabric CA均会在`organisations`文件夹创建所有三个组织中的加密资料和MSP文件夹。
- 该脚本使用configtxgen工具创建系统通道生成块。 Configtxgen使用了`TwoOrgsOrdererGenesis`通道配置文件中的`configtx/configtx.yaml`文件创建创世区块。 区块被存储在`system-genesis-block`文件夹中。
- 一旦组织的加密资料和系统通道的创始块生成后，`network.sh`就可以启动网络的节点。 脚本使用`docker`文件夹中的`docker-compose-test-net.yaml`文件创建对等节点和排序节点。 `docker`文件夹还包含`docker-compose-e2e.yaml`文件启动网络节点三个Fabric CA。 该文件旨在用于Fabric SDK 运行端到端测试。 请参阅Node SDK代码库有关运行这些测试的详细信息。
- 如果您使用`createChannel`子命令，则`./ network.sh`使用提供的频道名称， 运行在`scripts`文件夹中的`createChannel.sh`脚本来创建通道。 该脚本使用`configtx.yaml`文件来创建通道创作事务，以及两个锚对等节点更新交易。 该脚本使用对等节点cli创建通道，加入`peer0.org1.example.com`和`peer0.org2.example.com`到频道， 以及使两个对等节点都成为锚对等节点。
- 如果执行`deployCC`命令， `./ network.sh`会运行`deployCC.sh`脚本在两个 peer 节点上安装**asset-transfer (basic)**链码， 然后定义通道上的链码。 一旦将链码定义提交给通道，对等节点cli使用`Init`初始化链码并调用链码将初始数据放入账本。

故障排除

如果您对本教程有任何疑问，请查看以下内容：

- 您应该始终重新启动网络。 您可以使用以下命令删除先前运行的工件，加密材料，容器，卷和链码镜像：

  ```
  ./network.sh down
  ```

  如果您不删除旧的容器，镜像和卷，将看到报错。
- 如果您看到Docker错误，请先检查您的Docker版本(Prerequisites)， 然后尝试重新启动Docker进程。 Docker的问题是经常无法立即识别的。 例如，您可能会看到您的节点无法访问挂载在容器内的加密材料导致的错误。

  如果问题仍然存在，则可以删除镜像并从头开始：

  ```
  docker rm -f $(docker ps -aq)
  docker rmi -f $(docker images -q)
  ```

- 如果您在创建，批准，提交，调用或查询命令时发现错误，确保您已正确更新通道名称和链码名称。 提供的示例命令中有占位符值。
- 如果您看到以下错误：

  ```
  Error: Error endorsing chaincode: rpc error: code = 2 desc = Error installing chaincode code m
  ```

您可能有先前运行中链码镜像（例如`dev-peer1.org2.example.com-asset-transfer-1.0`或`dev-peer0.org1.example.com-asset-transfer-1.0`）。 删除它们并再次尝试。

```
docker rmi -f $(docker images | grep dev-peer[0-9] | awk '{print $3}')
```

- 如果您看到以下错误：

```
[configtx/tool/localconfig] Load -> CRIT 002 Error reading configuration: Unsupported Config T
panic: Error reading configuration: Unsupported Config Type ""
```

那么您没有正确设置环境变量`FABRIC_CFG_PATH`。configtxgen工具需要此变量才能找到configtx.yaml。 返回执行`export FABRIC_CFG_PATH=$PWD/configtx/configtx.yaml`，然后重新创建您的通道工件。

- 如果看到错误消息指出您仍然具有“active endpoints”，请清理您的Docker网络。 这将清除您以前的网络，并以全新环境开始：

```
docker network prune
```

您将看到一下信息：

```
WARNING! This will remove all networks not used by at least one container.
Are you sure you want to continue? [y/N]
```

选`y`。

- 如果您看到类似下面的错误：

```
/bin/bash: ./scripts/createChannel.sh: /bin/bash^M: bad interpreter: No such file or directory
```

确保有问题的文件（在此示例中为**createChannel.sh**）为以Unix格式编码。 这很可能是由于未在Git配置中将`core.autocrlf`设置为`false`（查看Windows Extras）。 有几种解决方法。 如果你有例如vim编辑器，打开文件：

```
vim ./fabric-samples/test-network/scripts/createChannel.sh
```

然后通过执行以下vim命令来更改其格式：

```
:set ff=unix
```

- 如果您的排序者在创建时退出，或者您看到由于无法连接到排序服务创建通道命令失败， 请使用`docker logs`命令从排序节点读取日志。 你可能会看到以下消息：

```
PANI 007 [channel system-channel] config requires unsupported orderer capabilities: Orderer ca
```

当您尝试使用Fabric 1.4.x版本docker镜像运行网络时，会发生这种情况。 测试网络需要使用Fabric 2.x版本运行。

如果您仍然发现错误，请在**fabric-questions**上共享您的日志 Hyperledger Rocket chat或StackOverflow。