创建新通道

您可以使用本教程来学习如何使用configtxgen CLI工具创建新通道，然后使用peer channel命令让您的Peer节点加入该通道。尽管本教程将利用Fabric测试网络来创建新通道，但是本教程中的步骤生产环境中的网络运维人员也可以使用。

在创建通道的过程中，本教程将带您逐个了解以下步骤和概念：

- 配置configtxgen工具
- 使用configtx.yaml
- Orderer系统通道
- 创建应用通道
- Peer加入通道
- [设置锚节点]](#set-anchor-peers)

配置configtxgen工具

通过构建通道创建交易并将交易提交给排序服务来创建通道。通道创建交易指定通道的初始配置，并由排序服务用于写入通道创世块。尽管可以手动构建通道创建交易文件，但使用configtxgen工具会更容易。该工具通过读取定义通道配置的`configtx.yaml`文件，然后将相关信息写入通道创建交易中来工作。在下一节讨论`configtx.yaml`文件之前，我们可以先开始下载并配置好`configtxgen`工具。

您可以按照安装示例，二进制文件和Docker镜像的步骤下载`configtxgen`二进制文件。`configtxgen`和其他Fabric工具一起将被下载到本地fabric-samples的`bin`文件夹中。

对于本教程，我们将要在`fabric-samples`目录下的`test-network`目录中进行操作。使用以下命令进入到该目录：

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

在本教程的其余部分中，我们将在`test-network`目录进行操作。使用以下命令将configtxgen工具添加到您的CLI路径：

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

为了使用`configtxgen`，您需要将`FABRIC_CFG_PATH`环境变量设置为本地包含`configtx.yaml`文件的目录的路径。在本教程中，我们将在`configtx`文件夹中引用用于设置Fabric测试网络的`configtx.yaml`：

```
export FABRIC_CFG_PATH=${PWD}/configtx
```

您可以通过打印`configtxgen`帮助文本来检查是否可以使用该工具：

```
configtxgen --help
```

使用configtx.yaml

`configtx.yaml`文件指定新通道的**通道配置**。建立通道配置所需的信息在`configtx.yaml`文件中以读写友好的形式指定。`configtxgen`工具使用`configtx.yaml`文件中定义的通道配置文件来创建通道配置，并将其写入protobuf格式，然后由Fabric读取。

您可以在`test-network`目录下的`configtx`文件夹中找到`configtx.yaml`文件，该文件用于部署测试网络。该文件包含以下信息，我们将使用这些信息来创建新通道：

- **Organizations:** 可以成为您的通道成员的组织。每个组织都有对用于建立通道MSP的密钥信息的引用。
- **Ordering service:** 哪些排序节点将构成网络的排序服务，以及它们将用于同意一致交易顺序的共识方法。该文件还包含将成为排序服务管理员的组织。
- **Channel policies:** 文件的不同部分共同定义策略，这些策略将控制组织与通道的交互方式以及哪些组织需要批准通道更新。就本教程而言，我们将使用Fabric使用的默认策略。
- **Channel profiles:** 每个通道配置文件都引用`configtx.yaml`文件其他部分的信息来构建通道配置。使用预设文件来创建Orderer系统通道的创世块以及将被Peer组织使用的通道。为了将它们与系统通道区分开来，Peer组织使用的通道通常称为应用通道。

`configtxgen`工具使用`configtx.yaml`文件为系统通道创建完整的创世块。因此，系统通道配置文件需要指定完整的系统通道配置。用于创建通道创建交易的通道配置文件仅需要包含创建应用通道所需的其他配置信息。

您可以访问使用configtx.yaml创建通道创世块教程，以了解有关此文件的更多信息。现在，我们将返回创建通道的操作方面，尽管在后续的步骤中将引用此文件的某些部分。

启动网络

我们将使用正在运行的Fabric测试网络来创建新通道。出于本教程的考虑，我们希望从一个已知的初始状态进行操作。以下命令将停掉所有容器并删除任何之前生成的文件。确保您仍在本地`fabric-samples`的`test-network`目录中进行操作。

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

您可以使用以下命令来启动测试网络：

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

这个命令将使用在`configtx.yaml`文件中定义的两个Peer组织和单个Orderer组织创建一个Fabric网络。Peer组织将各自运营一个Peer节点，而排序服务管理员将运营单个Orderer节点。运行命令时，脚本将打印出正在创建的节点的日志：

```
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
```

我们测试网络实例的部署没有创建应用通道。但是，当您执行`./network.sh up`命令时，测试网络脚本会创建系统通道。在底层，脚本使用`configtxgen`工具和`configtx.yaml`文件构建系统通道的创世块。因为系统通道用于创建其他通道，所以在创建应用通道之前，我们需要花一些时间来了解Orderer系统通道。

Orderer系统通道

在Fabric网络中创建的第一个通道是系统通道。系统通道定义了形成排序服务的Orderer节点集合和充当排序服务管理员的组织集合。

系统通道还包括属于区块链联盟的组织。联盟是一组Peer组织，它们属于系统通道，但不是排序服务的管理员。联盟成员可以创建新通道，并包括其他联盟组织作为通道成员。

要部署新的排序服务，需要系统通道的创世块。当您执行`./network.sh up`命令时，测试网络脚本已经创建了系统通道创世块。创世块用于部署单个Orderer节点，该Orderer节点使用该块创建系统通道并形成网络的排序服务。如果检查`./ network.sh`脚本的输出，则可以在日志中找到创建创世块的命令：

```
configtxgen -profile TwoOrgsOrdererGenesis -channelID system-channel -outputBlock ./system-genesi
```

`configtxgen`工具使用来自`configtx.yaml`的`TwoOrgsOrdererGenesis`通道配置文件来写入创世块并将其存储在`system-genesis-block`文件夹中。 您可以在下面看到`TwoOrgsOrdererGenesis`配置文件：

```
TwoOrgsOrdererGenesis:
    <<: *ChannelDefaults
    Orderer:
        <<: *OrdererDefaults
        Organizations:
            - *OrdererOrg
        Capabilities:
            <<: *OrdererCapabilities
    Consortiums:
        SampleConsortium:
            Organizations:
                - *Org1
                - *Org2
```

配置文件的`Orderer:`部分创建测试网络使用的单节点Raft排序服务，并以`OrdererOrg`作为排序服务管理员。配置文件的`Consortiums`部分创建了一个名为`SampleConsortium:`的Peer组织的联盟。 这两个Peer组织Org1和Org2都是该联盟的成员。因此，我们可以将两个组织都包含在测试网络创建的新通道中。如果我们想添加另一个组织作为通道成员而又不将该组织添加到联盟中，则我们首先需要使用Org1和Org2创建通道，然后通过更新通道配置添加该组织。

创建应用通道

现在我们已经部署了网络的节点并使用`network.sh`脚本创建了Orderer系统通道，我们可以开始为Peer组织创建新通道。我们已经设置了使用`configtxgen`工具所需的环境变量。运行以下命令为`channel1`创建一个创建通道的交易：

```
configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel1.tx -chann
```

`-channelID`是将要创建的通道的名称。通道名称必须全部为小写字母，少于250个字符，并且与正则表达式`[a-z][a-z0-9.-]*`匹配。该命令使用`-profile`标志来引用`configtx.yaml`中的`TwoOrgsChannel:`配置文件，测试网络使用它来创建应用通道：

```
TwoOrgsChannel:
    Consortium: SampleConsortium
    <<: *ChannelDefaults
    Application:
        <<: *ApplicationDefaults
        Organizations:
            - *Org1
            - *Org2
        Capabilities:
            <<: *ApplicationCapabilities
```

该配置文件从系统通道引用`SampleConsortium`的名称，并且包括来自该联盟的两个Peer组织作为通道成员。因为系统通道用作创建应用通道的模板，所以系统通道中定义的排序节点成为新通道的默认共识者集合。排序服务的管理员成为该通道的Orderer管理员。可以使用通道更新在共识者者集合中添加或删除Orderer节点和Orderer组织。

如果命令执行成功，您将看到`configtxgen`的日志加载`configtx.yaml`文件并打印通道创建交易：

```
2020-03-11 16:37:12.695 EDT [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-03-11 16:37:12.738 EDT [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded config
2020-03-11 16:37:12.740 EDT [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 003 Genera
2020-03-11 16:37:12.789 EDT [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 004 Writin
```

我们可以使用`peer` CLI将通道创建交易提交给排序服务。要使用`peer` CLI，我们需要将`FABRIC_CFG_PATH`设置为`fabric-samples/config`目录中的`core.yaml`文件。通过运行以下命令来设置`FABRIC_CFG_PATH`环境变量：

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

在排序服务创建通道之前，排序服务将检查提交请求的身份的许可。默认情况下，只有属于系统通道的联盟组织的管理员身份才能创建新通道。发出以下命令，以Org1中的admin用户身份运行`peer` CLI：

```
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
```

现在，您可以使用以下命令创建通道：

```
peer channel create -o localhost:7050  --ordererTLSHostnameOverride orderer.example.com -c channe
```

上面的命令使用`-f`标志提供通道创建交易文件的路径，并使用`-c`标志指定通道名称。`-o`标志用于选择将用于创建通道的排序节点。`--cafile`是Orderer节点的TLS证书的路径。当您运行`peer channel create`命令时，`peer` CLI将生成以下响应：

```
2020-03-06 17:33:49.322 EST [channelCmd] InitCmdFactory -> INFO 00b Endorser and orderer connecti
2020-03-06 17:33:49.550 EST [cli.common] readBlock -> INFO 00c Received block: 0
```

由于我们使用的是Raft排序服务，因此您可能会收到一些状态不可用的消息，您可以放心地忽略它们。该命令会将新通道的创世块返回到`--outputBlock`标志指定的位置。

Peer加入通道

创建通道后，我们可以让Peer加入通道。属于该通道成员的组织可以使用peer channel fetch命令从排序服务中获取通道创世块。然后，组织可以使用创世块，通过peer channel join命令将Peer加入到该通道。一旦Peer加入通道，Peer将通过从排序服务中获取通道上的其他区块来构建区块链账本。

由于我们已经以Org1管理员的身份使用`peer` CLI，因此让我们将Org1的Peer加入到通道中。由于Org1提交了通道创建交易，因此我们的文件系统上已经有了通道创世块。使用以下命令将Org1的Peer加入通道。

```
peer channel join -b ./channel-artifacts/channel1.block
```

环境变量`CORE_PEER_ADDRESS`已设置为以`peer0.org1.example.com`为目标。命令执行成功后将生成`peer0.org1.example.com`加入通道的响应：

```
2020-03-06 17:49:09.903 EST [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connecti
2020-03-06 17:49:10.060 EST [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal
```

您可以使用peer channel getinfo命令验证Peer是否已加入通道：

```
peer channel getinfo -c channel1
```

该命令将列出通道的区块高度和最新区块的哈希。由于创世块是通道上的唯一区块，因此通道的高度将为1：

```
2020-03-13 10:50:06.978 EDT [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connecti
Blockchain info: {"height":1,"currentBlockHash":"kvtQYYEL2tz0kDCNttPFNC4e6HVUFOGMTIDxZ+DeNQM="}
```

现在，我们可以将Org2的Peer加入通道。设置以下环境变量，以Org2管理员的身份运行`peer` CLI。环境变量还将把Org2的Peer `peer0.org1.example.com`设置为目标Peer。

```
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
```

尽管我们的文件系统上仍然有通道创世块，但在更现实的场景下，Org2将从排序服务中获取块。例如，我们将使用`peer channel fetch`命令来获取Org2的创世块：

```
peer channel fetch 0 ./channel-artifacts/channel_org2.block -o localhost:7050 --ordererTLSHostnam
```

该命令使用`0`来指定它需要获取加入通道所需的创世块。如果命令成功执行，则应该看到以下输出：

```
2020-03-13 11:32:06.309 EDT [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connecti
2020-03-13 11:32:06.336 EDT [cli.common] readBlock -> INFO 002 Received block: 0
```

该命令返回通道生成块并将其命名为`channel_org2.block`，以将其与由Org1拉取的块区分开。现在，您可以使用该块将Org2的Peer加入该通道：

```
peer channel join -b ./channel-artifacts/channel_org2.block
```

配置锚节点

组织的Peer加入通道后，他们应至少选择一个Peer成为锚定节点。为了利用诸如私有数据和服务发现之类的功能，需要Peer锚节点。每个组织都应在一个通道上设置多个锚节点以实现冗余。有关Gossip和Peer锚节点的更多信息，请参见Gossip数据分发协议。

通道配置中包含每个组织的Peer锚节点的端点信息。每个通道成员可以通过更新通道来指定其Peer锚节点。我们将使用configtxlator工具更新通道配置，并为Org1和Org2选择锚节点。设置Peer锚节点的过程与进行其他通道更新所需的步骤相似，并介绍了如何使用`configtxlator`更新通道配置。您还需要在本地计算机上安装jq工具。

我们将从选择一个Peer锚节点作为Org1开始。第一步是使用`peer channel fetch`命令来获取最新的通道配置块。设置以下环境变量，以Org1 管理员身份运行`peer` CLI：

```
export FABRIC_CFG_PATH=$PWD/../config/
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${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
```

您可以使用以下命令来获取通道配置：

```
peer channel fetch config channel-artifacts/config_block.pb -o localhost:7050 --ordererTLSHostnam
```

由于最新的通道配置块是通道创世块，因此您将看到该通道的命令返回块0。

```
2020-04-15 20:41:56.595 EDT [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connecti
2020-04-15 20:41:56.603 EDT [cli.common] readBlock -> INFO 002 Received block: 0
2020-04-15 20:41:56.603 EDT [channelCmd] fetch -> INFO 003 Retrieving last config block: 0
2020-04-15 20:41:56.608 EDT [cli.common] readBlock -> INFO 004 Received block: 0
```

通道配置块存储在`channel-artifacts`文件夹中，以使更新过程与其他工件分开。进入到`channel-artifacts`文件夹以完成以下步骤：

```
cd channel-artifacts
```

现在，我们可以开始使用`configtxlator`工具开始通道配置相关工作。第一步是将来自protobuf的块解码为可以读写友好的JSON对象。我们还将去除不必要的块数据，仅保留通道配置。

```
configtxlator proto_decode --input config_block.pb --type common.Block --output config_block.json
jq .data.data[0].payload.data.config config_block.json > config.json
```

T这些命令将通道配置块转换为简化的JSON`config.json`，它将作为我们更新的基准。因为我们不想直接编辑此文件，所以我们将制作一个可以编辑的副本。我们将在以后的步骤中使用原始的通道配置。

```
cp config.json config_copy.json
```

您可以使用`jq`工具将Org1的Peer锚节点添加到通道配置中。

```
jq '.channel_group.groups.Application.groups.Org1MSP.values += {"AnchorPeers":{"mod_policy": "Adm
```

完成此步骤后，我们在`modified_config.json`文件中以JSON格式获取了通道配置的更新版本。现在，我们可以将原始和修改的通道配置都转换回protobuf格式，并计算它们之间的差异。

```
configtxlator proto_encode --input config.json --type common.Config --output config.pb
configtxlator proto_encode --input modified_config.json --type common.Config --output modified_co
configtxlator compute_update --channel_id channel1 --original config.pb --updated modified_config
```

名为`channel_update.pb`的新的protobuf包含我们需要应用于通道配置的Peer锚节点更新。我们可以将配置更新包装在交易Envelope中，以创建通道配置更新交易。

```
configtxlator proto_decode --input config_update.pb --type common.ConfigUpdate --output config_up
echo '{"payload":{"header":{"channel_header":{"channel_id":"channel1", "type":2}},"data":{"config
configtxlator proto_encode --input config_update_in_envelope.json --type common.Envelope --output
```

现在，我们可以使用最终的工件`config_update_in_envelope.pb`来更新通道。回到`test-network`目录：

```
cd ..
```

我们可以通过向`peer channel update`命令提供新的通道配置来添加Peer锚节点。因为我们正在更新仅影响Org1的部分通道配置，所以其他通道成员不需要批准通道更新。

```
peer channel update -f channel-artifacts/config_update_in_envelope.pb -c channel1 -o localhost:70
```

通道更新成功后，您应该看到以下响应：

```
2020-01-09 21:30:45.791 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
```

我们可以为Org2设置锚节点。因为我们是第二次进行该过程，所以我们将更快地完成这些步骤。 设置环境变量，以Org2管理员的身份运行`peer` CLI：

```
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
```

拉取最新的通道配置块，这是该通道上的第二个块：

```
peer channel fetch config channel-artifacts/config_block.pb -o localhost:7050 --ordererTLSHostnam
```

回到`channel-artifacts`目录：

```
cd channel-artifacts
```

然后，您可以解码并复制配置块。

```
configtxlator proto_decode --input config_block.pb --type common.Block --output config_block.json
jq .data.data[0].payload.data.config config_block.json > config.json
cp config.json config_copy.json
```

在通道配置中将加入通道的Org2的Peer添加为锚节点：

```
jq '.channel_group.groups.Application.groups.Org2MSP.values += {"AnchorPeers":{"mod_policy": "Adm
```

现在，我们可以将原始和更新的通道配置都转换回protobuf格式，并计算它们之间的差异。

```
configtxlator proto_encode --input config.json --type common.Config --output config.pb
configtxlator proto_encode --input modified_config.json --type common.Config --output modified_co
configtxlator compute_update --channel_id channel1 --original config.pb --updated modified_config
```

将配置更新包装在交易Envelope中以创建通道配置更新交易：

```
configtxlator proto_decode --input config_update.pb --type common.ConfigUpdate --output config_up
echo '{"payload":{"header":{"channel_header":{"channel_id":"channel1", "type":2}},"data":{"config
configtxlator proto_encode --input config_update_in_envelope.json --type common.Envelope --output
```

回到`test-network`目录.

```
cd ..
```

通过执行以下命令来更新通道并设置Org2的Peer锚节点：

```
peer channel update -f channel-artifacts/config_update_in_envelope.pb -c channel1 -o localhost:70
```

您可以通过运行`peer channel info`命令来确认通道已成功更新：

```
peer channel getinfo -c channel1
```

现在，已经通过在通道创世块中添加两个通道配置块来更新通道，通道的高度将增加到3：

```
Blockchain info: {"height":3,"currentBlockHash":"eBpwWKTNUgnXGpaY2ojF4xeP3bWdjlPHuxiPCTIMxTk=","p
```

在新通道上部署链码

通过将链码部署到通道，我们可以确认该通道已成功创建。我们可以使用`network.sh`脚本将Fabcar链码部署到任何测试网络通道。使用以下命令将链码部署到我们的新通道：

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

运行命令后，您应该在日志中看到链代码已部署到通道。调用链码将数据添加到通道账本中，然后查询。

```
[{"Key":"CAR0","Record":{"make":"Toyota","model":"Prius","colour":"blue","owner":"Tomoko"}},
{"Key":"CAR1","Record":{"make":"Ford","model":"Mustang","colour":"red","owner":"Brad"}},
{"Key":"CAR2","Record":{"make":"Hyundai","model":"Tucson","colour":"green","owner":"Jin Soo"}},
{"Key":"CAR3","Record":{"make":"Volkswagen","model":"Passat","colour":"yellow","owner":"Max"}},
{"Key":"CAR4","Record":{"make":"Tesla","model":"S","colour":"black","owner":"Adriana"}},
{"Key":"CAR5","Record":{"make":"Peugeot","model":"205","colour":"purple","owner":"Michel"}},
{"Key":"CAR6","Record":{"make":"Chery","model":"S22L","colour":"white","owner":"Aarav"}},
{"Key":"CAR7","Record":{"make":"Fiat","model":"Punto","colour":"violet","owner":"Pari"}},
{"Key":"CAR8","Record":{"make":"Tata","model":"Nano","colour":"indigo","owner":"Valeria"}},
{"Key":"CAR9","Record":{"make":"Holden","model":"Barina","colour":"brown","owner":"Shotaro"}}]
==================== Query successful on peer0.org1 on channel 'channel1' ====================
```