实验六：超级账本项目Fabric实验

超级账本 (Hyperledger) 是由 Linux 基金会于 2015 年启动的区块链开源项目，该项目一经启动，反响热烈，一举成为区块链生态圈的明星项目。目前，已经有超过 200 家的会员单位参与进超级账本项目。超级账本项目旨在让一个组织的组织成员都能享受到去中心化的区块链账本带来的应用级便利，都能够参与到区块链账本的维护和建设中，使区块链账本应用能够在各行各业上线、应用、落地，满足各种各样的安全性需求。

Fabric 是目前超级账本最为成熟和流行的一个项目。与比特币、以太坊等公链系统不同，Fabric 拥有一套完整的联盟链体系，一条区块链的规模相对较小，但其拥有严格的准入规则和身份管理规则，建立了模块化的共识机制、身份管理体系、智能合约、应用客户端、隐私保护模块和证书体系。Fabric系统具有强大的内部成员可信性，这样使得其共识机制可以集中处理交易的排序问题，多采用分布式系统的共识，如 Raft、Kafka 等，效率极高，因此交易速度和吞吐率都在数量级上超越了比特币、以太坊等经典的公链系统。

区块链系统根据面向体是否公开、准入规则、规模大小、去中心化程度，一般分为公有链、联盟链和私有链。其中，私有链的中心化程度较为严重，相比于公有链，联盟链加入了准入的规则，即节点或用户的身份需要经过认证。另外，联盟链实际上是介于公有链和私有链中间的一类区块链，只不过与私有链具有本质上的相似性，即拥有严格的准入规则，但其规模远超私有链，与无准入规则、公开可访问的公有链有本质上的区别。因此，也可以说，联盟链是一种特殊的私有链。

联盟链有以下特点:

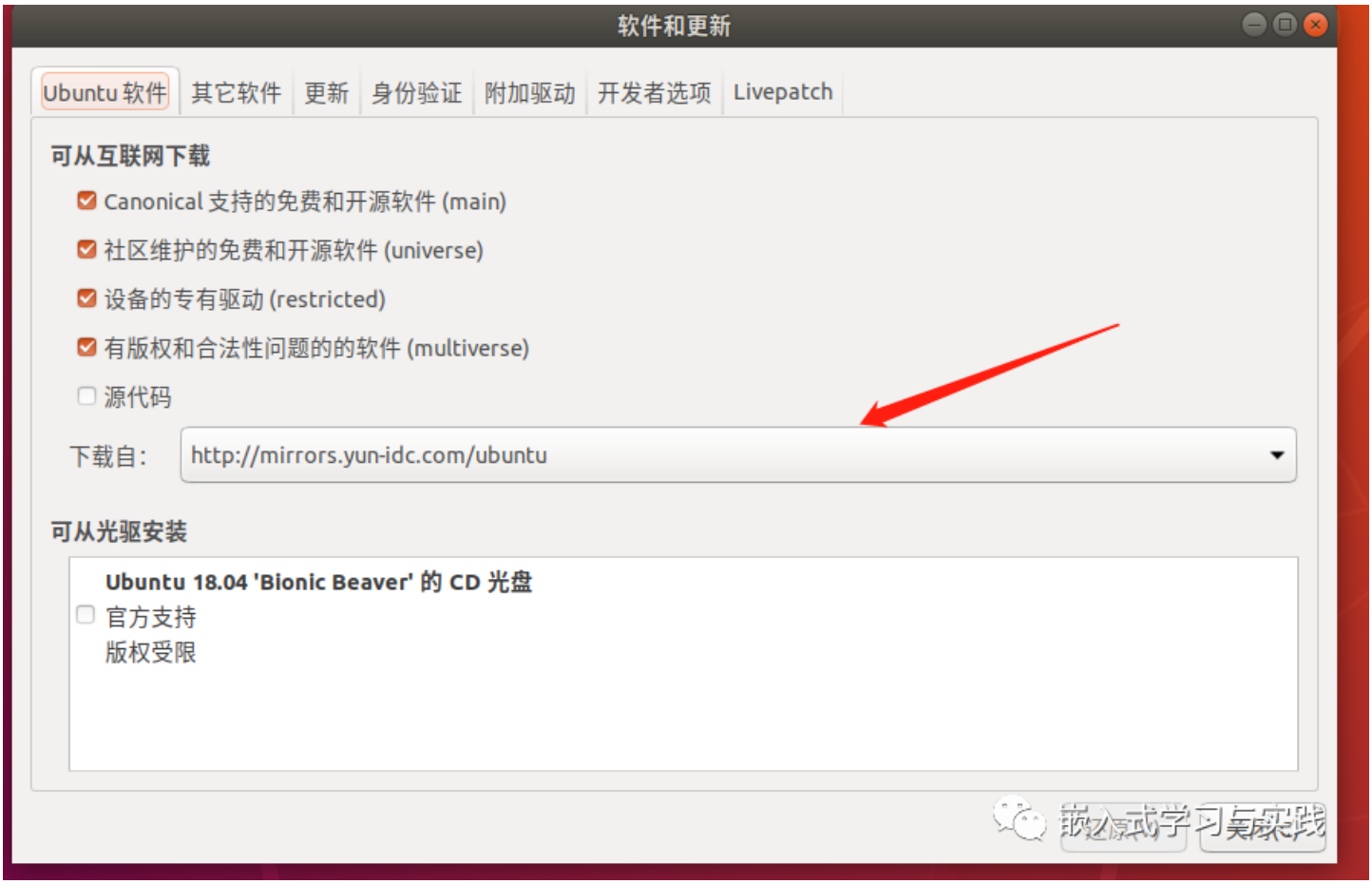
1. 弱中心化 (或称为多中心化、部分去中心化)。在某种意义上，联盟链不像公有链一样归属于任何大众，而是存在于规则严格的联盟组织内部。由于联盟链节点数量大多都是有限的，因此联盟组织内部达成共识相对较容易。这也是联盟链交易确认速度、吞吐率较高的重要原因。
2. 有较强的可控性。公有链的节点是海量的、无许可的，在公有链中，数据只要上链并确认，一般是不可篡改的。但在联盟链中，达成共识比较容易，因此有一定的可控性。
3. 数据默认不公开。联盟链仅供联盟组织内部成员调用其数据。

本章实验中的 Fabric 便是联盟链项目的典型例子。

一、Fabric环境的配置

1. **替换软件源**

方法一：在“软件与更新”中选择合适的软件源即可（推荐）



方法二：手动修改/etc/apt/source.list，替换完成后用"sudo apt update"和"sudo apt upgrade"命令对系统软件包进行更新

2. **更新Ubuntu系统环境**

```
1 sudo apt-get update
2 sudo apt-get upgrade
```

3. **安装Go语言环境**

- 下载go语言安装包

```
1 wget https://dl.google.com/go/go1.18.1.linux-amd64.tar.gz
2 sudo tar -xvf go1.18.1.linux-amd64.tar.gz
3 sudo mv go /usr/local
4 mkdir go
5 mkdir go/src
```

- 配置go语言环境变量

```
1 vim ~/.bashrc
```

```
# 添加如下内容
export GOROOT=/usr/local/go
export GOPATH=$HOME/go
export PATH=$GOPATH/bin:$GOROOT/bin:$PATH
export HUBPATH=/root/Cross-chain/CrossChainHub # 1.6步提前执行
# 保存更改
source ~/.bashrc
# 确认是否配置成功
go version
```

4. 安装docker与docker-compose

```
# 卸载旧版本
sudo apt-get remove docker docker-engine docker.io containerd runc
# 更新apt软件包索引
sudo apt-get update
# 安装软件包以允许apt通过HTTPS使用存储库
sudo apt-get install \
apt-transport-https \
ca-certificates \
curl \
gnupg-agent \
software-properties-common
# 添加Docker的官方GPG密钥
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
# 验证密钥
sudo apt-key fingerprint 0EBFCD88
# X86_64/amd64平台下设置存储库
sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"
# 安装docker
sudo apt-get install docker-ce docker-ce-cli containerd.io
# 安装docker-compose
# 下载运行文件
# https://github.com/docker/compose/releases/download/1.28.6/docker-compose-Linux-x86_64
sudo curl -L "https://github.com/docker/compose/releases/download/1.28.6/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
# 配置权限
sudo chmod +x /usr/local/bin/docker-compose
# 查看版本信息
docker version
```

```
docker-compose version
```

5. 配置代理

```
# 设置go代理
go env -w GO111MODULE="on"
go env -w GOPROXY=https://mirrors.aliyun.com/goproxy/
# 设置docker代理
sudo vim /etc/docker/daemon.json
# 添加如下内容
{
  "registry-mirrors": ["https://registry.docker-cn.com","https://6kx4zyno.mirror.aliyuncs.com","http://hub-mirror.c.163.com","https://docker.mirrors.ustc.edu.cn"]
}
# 重启生效
systemctl daemon-reload
systemctl restart docker
```

6. Fabric的安装

```
# 创建并进入工作目录
mkdir -p $GOPATH/src/github.com/hyperledger/
cd $GOPATH/src/github.com/hyperledger/
# 下载fabric 2.3.2并解压
sudo wget https://github.com/hyperledger/fabric/releases/download/v2.3.2/hyperledger-fabric-linux-amd64-2.3.2.tar.gz
sudo mkdir /usr/local/fabric
sudo tar -xzvf hyperledger-fabric-linux-amd64-2.3.2.tar.gz -C /usr/local/fabric
# 下载fabric-ca 1.5.0并解压
wget https://github.com/hyperledger/fabric-ca/releases/download/v1.5.0/hyperledger-fabric-ca-linux-amd64-1.5.0.tar.gz
sudo mkdir /usr/local/fabric-ca
sudo tar -xzvf hyperledger-fabric-ca-linux-amd64-1.5.0.tar.gz -C /usr/local/fabric-ca
#设置环境变量，在/etc/profile末尾添加
#Fabric
export FABRIC=/usr/local/fabric
export FABRICCA=/usr/local/fabric-ca
export PATH=$PATH:$FABRIC/bin:$FABRICCA/bin
#更新环境变量source /etc/profile
# 脚本完成后会发现当前目录有了fabric-samples目录，之后执行
```

```
19 docker images
```

二、test-network 的启动和 Channel 的配置

节点分类：Fabric 中的节点主要分为两大类: Peer 节点和 Orderer 节点。其中，Peer 节点的主要功能是背书，即先模拟执行客户端提交的交易提案，若执行通过，则对该交易进行背书并返还给客户端，还具备存储区块链链上的数据的功能。Orderer 节点也叫排序节点，主要负责对背书后的交易进行排序共识，并打包发送给 Peer 节点上链。每个Peer 节点可以担任多种角色，如 Anchor Peer (锚节点，允许被其他 Peer 节点发现)、Endorser Peer (背书节点，模拟执行客户端提交的交易提案对交易进行背书并返给客户端)、Leader Peer (主节点，具有和 Orderer 节点进行通信的功能)、Committer Peer (记账节点，广播、同步、存储区块链数据)。

通道(Channel)：通道是 Fabric 中非常重要的概念，对一般用户来说，通道是指应用通道。通道的主要作用是对不同的账本进行隔离，可以理解为，有几条通道就有几条独立运行的区块链。

1. 启动 test network

本实验要求跟随指导书的指令引导，搭建第一个 Fabric 网络。首先定位到以下目录:

```
1 cd ~
2 cd go/src/github.com/hyperledger/fabric/scripts/fabric-samples/test-network/
```

运行 `sudo ./network.sh up` 启动网络

```
1 Creating network "fabric_test" with the default driver
2 Creating volume "docker_orderer.example.com" with default driver
3 Creating volume "docker_peer0.org1.example.com" with default driver
4 Creating volume "docker_peer0.org2.example.com" with default driver
5 Creating peer0.org1.example.com ... done
6 Creating orderer.example.com    ... done
7 Creating peer0.org2.example.com ... done
8 Creating cli                    ... done
9 CONTAINER ID   IMAGE                               COMMAND             CREATED
                  STATUS
10 PORTS
11             NAMES
12 7738c1e84751   hyperledger/fabric-tools:latest     "/bin/bash"         Less
   than a second ago   Up Less than a second                   cli
13 1f24de2c6cd5   hyperledger/fabric-peer:latest      "peer node start"   2
   seconds ago            Up Less than a second   0.0.0.0:9051->9051/tcp, :::9051-
```

```
>9051/tcp, 0.0.0.0:19051->19051/tcp, :::19051->19051/tcp
peer0.org2.example.com
14 bfc48b20360c   hyperledger/fabric-orderer:latest   "orderer"           2
seconds ago            Up Less than a second   0.0.0.0:7050->7050/tcp, :::7050-
>7050/tcp, 0.0.0.0:7053->7053/tcp, :::7053->7053/tcp, 0.0.0.0:17050-
>17050/tcp, :::17050->17050/tcp   orderer.example.com
15 b9a61fdaf47a   hyperledger/fabric-peer:latest      "peer node start"   2
seconds ago            Up Less than a second   0.0.0.0:7051->7051/tcp, :::7051-
>7051/tcp, 0.0.0.0:17051->17051/tcp, :::17051->17051/tcp
peer0.org1.example.com
```

最终出现以上输出日志则表示网络启动成功，每个加入Fabric网络的Node和User都需要隶属于某个组织，以上网络中包含了两个平行组织———— `peer0.org1.example.com` 和 `peer0.org2.example.com` ，它还包括一个作为ordering service维护网络的 `orderer.example.com` 。

创建channel

上节已经在机器上运行了peer节点和orderer节点，现在可以使用network.sh为Org1和Org2之间创建channel。channel是特定网络成员之间的私有通道，只能被属于该通道的组织使用，并且对网络的其他成员是不可见的。每个channel都有一个单独的区块链账本，属于该通道的组织可以让其下peer加入该通道，以让peer能够存储channel上的帐本并验证账本上的交易。

使用以下命令创建自定义通道testchannel：

```
sudo ./network.sh createChannel -c testchannel
```

```
+ jq '.channel_group.groups.Application.groups.Org2MSP.values += {"AnchorPeers":{"mod_policy": "Admins","value":{"anchor
_peers": [{"host": "peer0.org2.example.com","port": 9051}]},"version": "0"}}' Org2MSPconfig.json
+ configtxlator proto_encode --input Org2MSPconfig.json --type common.Config
+ configtxlator proto_encode --input Org2MSPmodified_config.json --type common.Config
+ configtxlator compute_update --channel_id testchannel --original original_config.pb --updated modified_config.pb
+ configtxlator proto_decode --input config_update.pb --type common.ConfigUpdate
+ jq .
++ cat config_update.json
+ echo '{"payload":{"header":{"channel_header":{"channel_id":"testchannel", "type":2}},"data":{"config_update":{' '"chan
nel_id":' '"testchannel",' '"isolated_data":' '{},' '"read_set":' '{' '"groups":' '{' '"Application":' '{' '"groups":' '
{' '"Org2MSP":' '{' '"groups":' '{},' '"mod_policy":' '"",' '"policies":' '{' '"Admins":' '{' '"mod_policy":' '"",' '"po
licy":' null, '"version":' '"0"' '},' '"Endorsement":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '}
,' '"Readers":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Writers":' '{' '"mod_policy":' '""
,' '"policy":' null, '"version":' '"0"' '}' '},' '"values":' '{' '"MSP":' '{' '"mod_policy":' '"",' '"value":' null, '"v
ersion":' '"0"' '}' '},' '"version":' '"0"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"versi
on":' '"0"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"0"' '},' '"write_set":'
'{' '"groups":' '{' '"Application":' '{' '"groups":' '{' '"Org2MSP":' '{' '"groups":' '{},' '"mod_policy":' '"Admins",'
'"policies":' '{' '"Admins":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Endorsement":' '{' '
"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Readers":' '{' '"mod_policy":' '"",' '"policy":' null,
'"version":' '"0"' '},' '"Writers":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '}' '},' '"values":'
 '{' '"AnchorPeers":' '{' '"mod_policy":' '"Admins",' '"value":' '{' '"anchor_peers":' '[' '{' '"host":' '"peer0.org2.ex
ample.com",' '"port":' 9051 '}' ']' '},' '"version":' '"0"' '},' '"MSP":' '{' '"mod_policy":' '"",' '"value":' null, '"v
ersion":' '"0"' '}' '},' '"version":' '"1"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"versi
on":' '"0"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"0"' '}' '}}}}'
+ configtxlator proto_encode --input config_update_in_envelope.json --type common.Envelope
2021-08-15 03:21:02.233 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2021-08-15 03:21:02.246 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
Anchor peer set for org 'Org2MSP' on channel 'testchannel'
Channel 'testchannel' joined
```

部署chaincode

{{< notice warning >}}
建议部署操作全部在 `root` 账户下进行，否则可能发生未知错误，以下流程为笔者在非 `root` 用户下所遇问题，最终重建虚拟机全部指令在 `root` 账户下才完成部署。

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。可以通过 `./network.sh deployCC` 命令部署智能合约，但本过程可能会出现很多问题。
使用以下命令部署chaincode：

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

此命令执行后可能会出现错误： `scripts/deployCC.sh: line 114: log.txt: Permission denied` ，很明显这是权限不足所致，加上sudo试试：

```
1 sudo ./network.sh deployCC -c testchannel -ccn basic -ccp ../asset-transfer-
  basic/chaincode-go -ccl go
```

加上sudo后出现新的错误： `deployCC.sh: line 59: go: command not found` 。检查本用户 `go` 命令可用，检查 `root` 用户 `go` 命令可用，单单 `sudo` 后不能用。查阅资料后发现这是因为 `linux` 系统为了安全，限制在使用 `sudo` 时会清空自定义的环境变量，最简单的解决方法是在 `/etc/sudoers` 文件中直接将该限制注释([2])：

```
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
#Defaults       env_reset
#Defaults       mail_badpass
#Defaults       secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "#include" directives:

"/etc/sudoers" [readonly] 30L, 758C                                    11,1          Top
```

加上注释后重新执行上条命令，又出现了新的错误：

```
go: github.com/golang/protobuf@v1.3.2: Get
"https://proxy.golang.org/github.com/golang/protobuf/@v/v1.3.2.mod": dial tcp
172.217.160.81:443: i/o timeout
```

很明显这是因为本地网络无法访问proxy.golang.org所致，在命令行输入`go env -w GO111MODULE=on && go env -w GOPROXY=https://goproxy.cn,direct`命令配置国内代理([3])后再次执行。令人意外的是错误不变，设置的代理没有生效？手动使用`go get github.com/golang/protobuf`手动下载安装后再次运行错误还是不变，此时检查本地`GOPATH`目录下已有`github.com/golang/protobuf`包，为什么没有识别到？此时灵机一动，使用`sudo go env`查看`GOPATH`环境变量，发现与本地用户不一致，原来`sudo`命令会使用`root`的`go`环境变量，而之前设置的代理、下载的包都只能在本地用户下生效，因此这个问题最终的解决方案是直接切换到`root`用户下重新配置`go`代理并运行。成功运行后可看见如下结果：

```
2021-08-15 00:45:54.064 PDT [chaincodeCmd] ClientWait -> INFO 001 txid
[ebeb8df6904f45b81fb30714f7eecb30b4bbfd32f4acc809f34f7c660e396eb8] committed
with status (VALID) at localhost:7051
2021-08-15 00:45:54.144 PDT [chaincodeCmd] ClientWait -> INFO 002 txid
[ebeb8df6904f45b81fb30714f7eecb30b4bbfd32f4acc809f34f7c660e396eb8] committed
with status (VALID) at localhost:9051
Chaincode definition committed on channel 'testchannel'
Using organization 1
Querying chaincode definition on peer0.org1 on channel 'testchannel'...
Attempting to Query committed status on peer0.org1, Retry after 3 seconds.
peer lifecycle chaincode querycommitted --channelID testchannel --name basic
res=0
Committed chaincode definition for chaincode 'basic' on channel 'testchannel':
Version: 1.0, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc,
Approvals: [Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.org1 on channel 'testchannel'
Using organization 2
Querying chaincode definition on peer0.org2 on channel 'testchannel'...
Attempting to Query committed status on peer0.org2, Retry after 3 seconds.
peer lifecycle chaincode querycommitted --channelID testchannel --name basic
res=0
Committed chaincode definition for chaincode 'basic' on channel 'testchannel':
Version: 1.0, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc,
Approvals: [Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.org2 on channel 'testchannel'
Chaincode initialization is not required
```

合约交互

在安装fabric中我们已经设置了 `fabric` 可执行文件的环境变量，需保证可以成功在 `test-network` 目录下使用 `peer` 命令。

1. 设置FABRIC_CFG_PATH变量，其下需包含core.yaml文件

```
export FABRIC_CFG_PATH=$PWD/../config/
# export FABRIC_CFG_PATH=/usr/local/fabric/config/
```

2. 设置其它 `Org1` 组织的变量依赖

```
# Environment variables for Org1
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=localhost:7051
```

`CORE_PEER_TLS_ROOTCERT_FILE` 和 `CORE_PEER_MSPCONFIGPATH` 环境变量指向 `Org1` 的 `organizations` 文件夹中的身份证书。

3. 初始化chaincode

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C testchannel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt --peerAddresses localhost:9051 --tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt -c '{"function":"InitLedger","Args":[]}'
```

```
root@ubuntu:/home/test/fabric-samples/test-network# peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride
 orderer.example.com --tls --cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/m
sp/tlscacerts/tlsca.example.com-cert.pem -C testchannel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles ${PWD
}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt --peerAddresses localhost:905
1 --tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt -c
'{"function":"InitLedger","Args":[]}'
2021-08-15 01:33:54.515 PDT [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: statu
s:200
```

4. 查询账本资产列表

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

```
root@ubuntu:/home/test/fabric-samples/test-network# peer chaincode query -C testchannel -n basic -c '{"Args":["GetAllAss
ets"]}'
[{"ID":"asset1","color":"blue","size":5,"owner":"Tomoko","appraisedValue":300},{"ID":"asset2","color":"red","size":5,"ow
ner":"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}]
```

5. 修改账本资产

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride
orderer.example.com --tls --cafile
${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.
com/msp/tlscacerts/tlsca.example.com-cert.pem -C testchannel -n basic --
peerAddresses localhost:7051 --tlsRootCertFiles
${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.exampl
e.com/tls/ca.crt --peerAddresses localhost:9051 --tlsRootCertFiles
${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.exampl
e.com/tls/ca.crt -c '{"function":"TransferAsset","Args":
["asset6","Christopher"]}'
```

```
root@ubuntu:/home/test/fabric-samples/test-network# peer chaincode query -C testchannel -n basic -c '{"Args":["GetAllAss
ets"]}'
[{"ID":"asset1","color":"blue","size":5,"owner":"Tomoko","appraisedValue":300},{"ID":"asset2","color":"red","size":5,"ow
ner":"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}]
root@ubuntu:/home/test/fabric-samples/test-network# peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride
 orderer.example.com --tls --cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/m
sp/tlscacerts/tlsca.example.com-cert.pem -C testchannel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles ${PWD
}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt --peerAddresses localhost:905
1 --tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt -c
'{"function":"TransferAsset","Args":["asset6","Christopher"]}'
2021-08-15 01:45:14.604 PDT [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: statu
s:200
root@ubuntu:/home/test/fabric-samples/test-network# peer chaincode query -C testchannel -n basic -c '{"Args":["GetAllAss
ets"]}'
[{"ID":"asset1","color":"blue","size":5,"owner":"Tomoko","appraisedValue":300},{"ID":"asset2","color":"red","size":5,"ow
ner":"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":"Christopher","appraisedValue":800}]
root@ubuntu:/home/test/fabric-samples/test-network#
```

6. 关闭网络

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

该命令将停止并删除节点和链码容器、组织加密材料、删除之前运行的通道项目和docker卷，并从Docker Registry移除链码镜像。

{{< notice note >}}

因为 `asset-transfer (basic)` 链码的背书策略需要交易同时被 `Org1` 和 `Org2` 签名，所以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。