部署

食品溯源组织信息表:

机构名称	组织标识符	组织ID
奶牛场	dairy_org	OrgDairyMSP
加工厂	process_org	OrgProcessMSP
销售终端	sell_org	OrgSellMSP

编写生成组织、节点、用户证书的配置文件 - `crypto-config.yaml`

```
# crypto-config.yaml
# ---------------------------------------------------------------------------
# "OrdererOrgs" - Definition of organizations managing orderer nodes
# ---------------------------------------------------------------------------
OrdererOrgs:
  # ---------------------------------------------------------------------------
  # Orderer
  # ---------------------------------------------------------------------------
  - Name: Orderer
    Domain: trace.com
    Specs:
      - Hostname: orderer
# ---------------------------------------------------------------------------
# "PeerOrgs" - Definition of organizations managing peer nodes
# ---------------------------------------------------------------------------
PeerOrgs:
  # ---------------------------------------------------------------------------
  # Org1
  # ---------------------------------------------------------------------------
  - Name: OrgDairy
    Domain: dairy.trace.com
    EnableNodeOUs: true
    Template:
      Count: 2
    Users:
      Count: 1

  # ---------------------------------------------------------------------------
  # Org2: See "Org1" for full specification
  # ---------------------------------------------------------------------------
  - Name: OrgProcess
    Domain: process.trace.com
    EnableNodeOUs: true
    Template:
      Count: 2
    Users:
```

```
        Count: 1

    - Name: OrgSell
      Domain: sell.trace.com
      EnableNodeOUs: true
      Template:
        Count: 2
      Users:
        Count: 1

```

```
# 执行命令生成证书
$ cryptogen generate --config=./crypto-config.yaml
```

编写 configtx.yaml 配置文件, 生成系统创始块和channel创始块配置文件

```
# configtx.yaml

---
################################################################################
#
#   Section: Organizations
#
#   - This section defines the different organizational identities which will
#   be referenced later in the configuration.
#
################################################################################
Organizations:
    - &OrdererOrg
        Name: OrdererOrg
        ID: OrdererMSP
        MSPDir: crypto-config/ordererOrganizations/trace.com/msp

    - &org_dairy
        Name: OrgDairyMSP
        ID: OrgDairyMSP
        MSPDir: crypto-config/peerOrganizations/dairy.trace.com/msp
        AnchorPeers:
            - Host: peer0.dairy.trace.com
              Port: 7051

    - &org_process
        Name: OrgProcessMSP
        ID: OrgProcessMSP
        MSPDir: crypto-config/peerOrganizations/process.trace.com/msp
        AnchorPeers:
            - Host: peer0.process.trace.com
              Port: 7051
     - &org_sell
        Name: OrgSellMSP
        ID: OrgSellMSP
        MSPDir: crypto-config/peerOrganizations/sell.trace.com/msp
        AnchorPeers:
            - Host: peer0.sell.trace.com
```

```
39                Port: 7051
40
41  ################################################################################
42  #
43  #   SECTION: Capabilities
44  #
45  ################################################################################
46  Capabilities:
47      Global: &ChannelCapabilities
48          V1_1: true
49      Orderer: &OrdererCapabilities
50          V1_1: true
51      Application: &ApplicationCapabilities
52          V1_2: true
53
54  ################################################################################
55  #
56  #   SECTION: Application
57  #
58  ################################################################################
59  Application: &ApplicationDefaults
60      Organizations:
61
62  ################################################################################
63  #
64  #   SECTION: Orderer
65  #
66  ################################################################################
67  Orderer: &OrdererDefaults
68      # Available types are "solo" and "kafka"
69      OrdererType: solo
70      Addresses:
71          - orderer.trace.com:7050
72
73      BatchTimeout: 2s
74      BatchSize:
75          MaxMessageCount: 100
76          AbsoluteMaxBytes: 64 MB
77          PreferredMaxBytes: 512 KB
78      Kafka:
79          Brokers:
80              - 127.0.0.1:9092
81      Organizations:
82
83  ################################################################################
84  #
85  #   Profile
86  #
87  ################################################################################
88  Profiles:
89
90      OrgsOrdererGenesis:
91          Capabilities:
```

```
            <<: *ChannelCapabilities
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *OrdererOrg
            Capabilities:
                <<: *OrdererCapabilities
        Consortiums:
            SampleConsortium:
                Organizations:
                    - *org_dairy
                    - *org_process
                    - *org_sell
    OrgsChannel:
        Consortium: SampleConsortium
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *org_dairy
                - *org_process
                - *org_sell
            Capabilities:
                <<: *ApplicationCapabilities

```

创建创始区块文件 - genesis.block

```
# 在configtx.yaml所在的目录中创建子目录 channel-artifacts
# 1. 生成创始块文件
$ configtxgen -profile OrgsOrdererGenesis -outputBlock ./channel-artifacts/genesis.block
$ tree channel-artifacts/
channel-artifacts/
└── genesis.block
```

创建通道文件 - channel.tx

```
$ configtxgen -profile OrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID mychannel
# 查看当前节点加入的通道
$ peer channel list
$ tree channel-artifacts/
channel-artifacts/
├── channel.tx
└── genesis.block
```

创建组织的锚节点文件

```
# Dairy 组织锚节点文件
$ configtxgen -profile OrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/DairyMSPanchors.tx -channelID tracechannel -asOrg OrgDairyMSP
# Process 组织锚节点文件
$ configtxgen -profile OrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/ProcessMSPanchors.tx -channelID tracechannel -asOrg OrgProcessMSP
# Sell 组织锚节点文件
```

```
$ configtxgen -profile OrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/SellMSPanchors.tx -channelID tracechannel -asOrg OrgSellMSP
# 查看生成的文件
$ tree channel-artifacts/
channel-artifacts/
├── channel.tx
├── DairyMSPanchors.tx
├── genesis.block
├── ProcessMSPanchors.tx
└── SellMSPanchors.tx
```

部署orderer节点

编写 `docker-orderer.yaml` 启动文件

```
# docker-orderer.yaml
```

```
# 启动 orderer 排序服务节点
$ docker-compose -f docker-orderer.yaml up -d
```

部署 dairy 组织的 peer0 节点

编写 - `docker-peer0-dairy.yaml` 启动文件

```
# docker-peer0-dairy.yaml
# 该配置文件会启动两个容器，一个是peer0，另一个是cli

```

```
# 启动dairy组织的peer0节点和客户端cli
$ docker-compose -f docker-peer0-dairy.yaml up -d
# 查看启动的容器
$ docker ps
CONTAINER ID        IMAGE                                COMMAND             CREATED
    STATUS              PORTS                                              NAMES
c4abf6504a73        `hyperledger/fabric-tools:latest`     "/bin/bash"         44 seconds ago   Up 43 seconds                                                  `cli`
f781b0d2c918        `hyperledger/fabric-peer:latest`      "peer node start"   45 seconds ago   Up 44 seconds       0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp   `peer0.dairy.trace.com`
f983b08a1222        `hyperledger/fabric-orderer:latest`   "orderer"           56 seconds ago   Up 55 seconds       0.0.0.0:7050->7050/tcp                           `orderer.trace.com`
```

进入到已经启动的cli容器中

```
$ docker exec -it cli bash
# 进入到cli容器之后, 在cli默认的工作目录下执行以下命令
# cli容器中bash默认的工作目录为: /opt/gopath/src/github.com/hyperledger/fabric/peer#
# 1. 创建通道
$ peer channel create -o orderer.trace.com:7050 -c mychannel -f ./channel-artifacts/channel.tx --tls $CORE_PEER_TLS_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/trace.com/msp/tlscacerts/tlsca.trace.com-cert.pem
$ ls
channel-artifacts  crypto  `tracechannel.block` -> 创建通道成功, 得到的通道文件
# 2. 当前 peer0 节点加入到通道中
$ peer channel join -b tracechannel.block
# 3. 可选操作 - 更新组织-Dairy的锚节点
$ peer channel update -o orderer.trace.com:7050 -c mychannel -f ./channel-artifacts/DairyMSPanchors.tx --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/trace.com/msp/tlscacerts/tlsca.trace.com-cert.pem
```

部署 dairy 组织的 peer1节点

编写 - `docker-peer1-dairy.yaml` 启动文件

```
# docker-peer1-dairy.yaml

```

```
# 在配置文件docker-peer1-dairy.yaml的存储目录启动peer1容器
$ docker-compose -f docker-peer1-dairy.yaml up -d
```

> 操作peer节点必须通过客户端才能够完成, 由于我们的cli客户端只创建了一个, 所以操作peer1也可以使用cli来完成, 但是cli中操作peer的时候使用的环境变量是指向peer0的, 所以我们首先要做的是修改cli的环境变量
>
> - `CORE_PEER_ID`
> - `CORE_PEER_ADDRESS`
> - `CORE_PEER_GOSSIP_EXTERNALENDPOINT`
> - `CORE_PEER_GOSSIP_BOOTSTRAP`
> - `CORE_PEER_LOCALMSPID`
> - `CORE_PEER_MSPCONFIGPATH`
> - `CORE_PEER_TLS_ROOTCERT_FILE`
> - `CORE_PEER_TLS_CERT_FILE`
> - `CORE_PEER_TLS_KEY_FILE`

```
# 进入到cli客户端容器中
# 在cli容器中执行以下操作，将以下换行变量导入，这样做会覆盖原来环境变量中的值
export CORE_PEER_ID=peer1.dairy.trace.com
export CORE_PEER_ADDRESS=peer1.dairy.trace.com:7051
export CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1.dairy.trace.com:7051
export CORE_PEER_GOSSIP_BOOTSTRAP=peer1.dairy.trace.com:7051
export CORE_PEER_LOCALMSPID=OrgDairyMSP
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/dairy.trace.com/users/Admin@dairy.trace.com/msp
export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/dairy.trace.com/peers/peer1.dairy.trace.com/tls/ca.crt
export CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/dairy.trace.com/peers/peer1.dairy.trace.com/tls/server.crt
export CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/dairy.trace.com/peers/peer1.dairy.trace.com/tls/server.key
# 将peer1加入到通中
$ peer channel join -b tracechannel.block
```

部署process组织的peer0节点

部署process组织的peer1节点

部署sell组织的peer0节点

部署sell组织的peer1节点

chaincode编写

机构名称	chaincode名称
奶牛场 - dairy	dairycc
加工厂 - process	processcc
销售终端 - sell	sellcc

> 奶牛场:
>
> - 奶牛场1

- 奶牛场2
- 奶牛场3

加工厂:

- 大兴
- 顺义
- 怀柔

销售终端

- 超市
- 京东
- 天猫

每个组织做的事儿不同, 需要单独处理, 需要三份链代码

- dairy.go
- process.go
- sell.go

因此需要将这三份文件放到不同的目录中

如果要对牛奶进行溯源, 处理流程

每一代牛奶上都应有一个ID, 身份的唯一标识

- 通过标识查出牛奶是由谁卖出去 -> 天猫超市
- 根据天猫超市的标识查询 -> 天猫超市的供应商
- 根据供应商(加工厂)的ID查询 -> 牛奶来着那个奶牛场
- 通过奶牛场 -> 查到牛的情况

奶牛场组织的chaincode

加工厂组织的chaincode

销售终端组织的chaincode