

lluckydog / blockchainlab-2022 Public

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#)

main

...

[blockchainlab-2022](#) / [lab3](#) / 实验三.md

lluckydog Merge branch 'main' of github.com:lluckydog/blockchainlab-2022



2 contributors



363 lines (271 sloc) | 13.9 KB

...

## 实验三 Fabric搭建peer并加入通道

### 实验目的

- 了解fabric上的基本配置
- 在fabric网络中添加peer，并加入网络
- 了解fabric上的基本证书和网络

### 实验介绍

在Fabric中，根据提供的服务不同，可以把服务节点分为三类：CA、Orderer和Peer。

- CA：用于提供Fabric中组织成员的身份注册和证书颁发
- Orderer：排序节点，搜集交易并排序出块，广播给其他组织的主节点
- Peer：背书、验证和存储节点

实验使用的Fabric版本为release-2.2，所有概念、架构以及命令文档，都可以在官方文档中搜索翻阅 [一个企业级区块链平台 — hyperledger-fabricdocs master 文档](#)

本次实验的目标是使用Fabric搭建一个peer节点，并用这个peer节点加入已经创建的通道之中

### 实验内容

1. 使用CA服务器注册身份，并获得CA服务器颁发的身份证书
2. 在本地准备Peer节点启动所需要的文件，启动Peer节点
3. 将自己搭建的Peer节点加入通道mychannel

## 系统目录

实验开始，基本的组织结构为：

- tls-ca 对应tls的证书，TLS（Transport Layer Security，安全传输层），TLS是建立在传输层TCP协议之上的协议，服务于应用层，它的前身是SSL（Secure Socket Layer，安全套接字层），它实现了将应用层的报文进行加密后再交由TCP进行传输的功能。
- org1/admin 对应组织org1的admin相关身份信息的配置和证书
- org1/ca 对应组织org1的ca相关的证书
- mychannel.block 对应加入的通道mychannel的初始区块

## 注册身份，并且获取证书

首先你需要在 TLS 服务上注册 Peer 身份。

```
export FABRIC_CA_CLIENT_TLS_CERTFILES=${to_tls_certfiles}

export FABRIC_CA_CLIENT_HOME=${to_tls_ca_admin}

// 注册身份
fabric-ca-client register --id.name ${PEERNAME} --id.secret ${PEERSECRET} --i

//获得对应的tls_msp
//存储tls_msp的位置
export FABRIC_CA_CLIENT_MSPDIR=${to_hold_tlsmsp}
fabric-ca-client enroll -u https://${PEERNAME}:${PEERSECRET}@URL:PORT --enrol

//注册成功后，可以把keystore下的文件存储为证书。当然，你也可以手动将其重新命名。
mv tls-msp/keystore/*_sk tls-msp/keystore/key.pem
mv msp/keystore/*_sk msp/keystore/key.pem
```

## 变量含义

在 Linux 中，`${variable}` 代表一个变量。其中，`${}` 用于划分该变量名的边界。

在以上命令中，各变量所代表的含义：

- `${PEERNAME}`：在本实验中，你注册的 Peer 节点名称。在 Fabric 的基本配置中，对于 Peer 的用户名和密码的设置没有什么严格要求。注意，为了方便实验统计，希望大家用自己的学号作为用户名。

- `${PEERSECRET}` : 在本实验中, 你注册的 Peer 节点对应的密码。
- `${HOSTNAME}` : 由于我们使用的是 TLS CA 进行注册, 所以我们还需要额外的 `crs.hosts` 对应的域名。这里, 我们推荐将其设置为与 `${PEERNAME}` 保持一致。
- `${to_tls_certfiles}` : TLS-CA 下的可信根证书地址。
- `${to_tls_ca_admin}` : TLS-CA Admin 的路径地址。
- `${to_hold_tlsmsp}` : 本实验中, 你可以自行定义一个空文件夹来存放生成的 TLS MSP 文件。

URL和PORT对应为URL和端口, **具体信息请查看群公告**。其他有关配置信息的内容可以查看[材料](#)。

注意, 在每步操作中间推荐unset环境变量, 例如FABRIC\_CA\_CLIENT\_MSPDIR, FABRIC\_CA\_CLIENT\_TLS\_CERTFILES等

注册组织的peer身份

```
export FABRIC_CA_CLIENT_TLS_CERTFILES=${to_tls_certfiles}

export FABRIC_CA_CLIENT_HOME=${to_org_ca_admin}

fabric-ca-client register --id.name ${PEERNAME} --id.secret ${PEERSECRET} --i

//存储msp的位置
export FABRIC_CA_CLIENT_MSPDIR=${to_hold_msp}
//获得对应的msp
fabric-ca-client enroll -u https://${PEERNAME}:${PEERSECRET}@URL:PORT
```

## 变量含义

在 Linux 中, `${variable}` 代表一个变量。其中, `${}` 用于划分该变量名的边界。

在以上命令中, 各变量所代表的含义:

- `${PEERNAME}` : 在本实验中, 你注册的 Peer 节点名称。在 Fabric 的基本配置中, 对于 Peer 的用户名和密码的设置没有什么严格要求。注意, **为了方便实验统计, 希望大家用自己的学号作为用户名**。
- `${PEERSECRET}` : 在本实验中, 你注册的 Peer 节点对应的密码。
- `${HOSTNAME}` : 由于我们使用的是 ORG CA 进行注册, 所以我们还需要额外的 `crs.hosts` 对应的域名。这里, 我们推荐将其设置为与 `${PEERNAME}` 保持一致。
- `${to_tls_certfiles}` : ORG 下的可信根证书地址。
- `${to_org_ca_admin}` : ORG Admin 的路径地址。
- `${to_hold_tlsmsp}` : 本实验中, 你可以自行定义一个空文件夹来存放生成的 MSP 文件。

如果对于具体的证书信息的内容有疑问，我们可以通过指令查看X.509证书的具体内容：

查看pem证书

```
openssl x509 -in ./msp/signcerts/cert.pem -text
```

或者

```
keytool -printcert -file XXX.pem
```

## 启动Peer节点

主要工作为

1. 构造msp文件夹
2. 设置环境变量（如果有需要）
3. 启动peer和CLI

### msp文件夹配置

msp文件夹对应位置为

```
org/msp
org/admin/msp
org/peer/msp
```

其中org/msp对应的证书为：

```
cp ${HOME}/fabric/org1/admin/msp/signcerts/cert.pem org1/msp/admincerts/admin-
org1-cert.pem
cp ${HOME}/fabric/org1/ca/crypto/ca-cert.pem org1/msp/cacerts/org1-ca-cert.pem
cp ${HOME}/fabric/org1/peer2/assets/tls-ca/tls-ca-cert.pem
org1/msp/tlscacerts/tls-ca-cert.pem
```

对应需要创建每个组织的config.yaml文件

注意，Certificate指的是该角色证书对应的颁发者证书信息，需要根据config.yaml放的位置不一样改成不一样的值，具体可以参考admin/msp/config.yaml

```
//config.yaml template
//对应msp文件夹下的内容修改cacerts对应的证书
NodeOUs:
  Enable: true
  ClientOUIdentifier:
    Certificate: cacerts/org1-ca-cert.pem
    OrganizationalUnitIdentifier: client
  PeerOUIdentifier:
    Certificate: cacerts/org1-ca-cert.pem
```

```

OrganizationalUnitIdentifier: peer
AdminOUIdentifier:
Certificate: cacerts/org1-ca-cert.pem
OrganizationalUnitIdentifier: admin
OrdererOUIdentifier:
Certificate: cacerts/org1-ca-cert.pem
OrganizationalUnitIdentifier: orderer

```

此部分详细操作见附件 Peer节点配置与启动.pdf

## 启动peer和cli

为了方便同学们的实验，我们使用docker-compose来启动peer和cli，这样也可以比较方便地配置不同服务的环境变量。

在启动服务前，首先创建docker network来方便容器之间通信 `docker network create -d bridge fabric-ca`

### 启动peer服务

我们也提供了一个peer节点的docker-compose.yml

```

version: "3"
networks:
  fabric-ca:
    external:
      name: fabric-ca
services:
  PEERNAME:
    # PEERNAME 与注册的节点信息保持一致
    container_name: PEERNAME
    image: hyperledger/fabric-peer:amd64-2.2.0
    environment:
      - GOPROXY=https://goproxy.cn,direct
      - GO111MODULE=on
    # CORE_PEER_ID对应注册peer名字
      - CORE_PEER_ID=org-peer
      - CORE_PEER_ADDRESS=${HOSTNAME}:7051
      - CORE_PEER_LOCALMSPID=org1MSP
    # 对应组织的msp的位置
      - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/org1/peer2/msp
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      - CORE_VM_DOCKER_ATTACHSTDOUT=true
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=fabric-ca
      - CORE_PEER_TLS_ENABLED=true
    # 对应组织org1对应的tls签名证书位置
      - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/org1/peer2/tls-
msp/signcerts/cert.pem
    # 对应组织org1的key位置
      - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/org1/peer2/tls-

```

```

msp/keystore/key.pem
# 对应组织org1的tls根证书位置
- CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/org1/peer2/tls-
msp/tlscacerts/tls-0-0-0-0-7052.pem
# GOSSIP前是否先和orderer节点通信
- CORE_PEER_GOSSIP_USELEADERELECTION=false
- CORE_PEER_GOSSIP_ORGLEADER=true
# 组织外部GOSSIP通信的配置
- CORE_PEER_GOSSIP_EXTERNALENDPOINT=${HOSTNAME}:7051
- CORE_PEER_GOSSIP_SKIPHANDSHAKE=true
working_dir: /opt/gopath/src/github.com/hyperledger/fabric/org1/peer2
volumes:
- /var/run:/host/var/run
//对应挂载的位置
- ${HOME}/fabric/org1/peer2:/etc/hyperledger/org1/peer2
networks:
- fabric-ca
#对应服务端端口映射
ports:
- 7051:7051

```

docker-compose 的 yml 配置需要严格符合 yaml 语法。yaml 配置文件写好之后，可以 Google 一下相关的 yaml lint 在线网站，检查一下自己的语法是否符合要求。一个示例网站在 <http://www.yamllint.com/>

其中CORE\_PEER\_ADDRESS, CORE\_PEER\_MSPCONFIGPATH, CORE\_PEER\_TLS\_CERT\_FILE, CORE\_PEER\_TLS\_KEY\_FILE, CORE\_PEER\_TLS\_ROOTCERT\_FILE, CORE\_PEER\_GOSSIP\_EXTERNALENDPOINT需要根据实际情况进行配置，对应的位置为

```

${HOME}/fabric/org1/peer2/${relative_root}
=/etc/hyperledger/org1/peer2/${relative_root}

```

具体可以了解[docker-compose](#)的mount机制

我们可以通过命令 `docker-compose up -d` 来启动容器，-d代表后台运行。

在初次启动时，我们可以 `docker logs contain_id`，在观察到日志出现

```

serve -> INFO 020 Started peer with ID=[name:"peer2-org2" ], network ID=[dev],
address=[peer2-org2:7052]

```

说明peer服务启动成功

## 启动CLI

为了方便和peer服务的交互，我们启动了CLI容器，这个容器是为了和peer节点交互使用的。由于我们的peer和CLI服务在同一台机器上，我们需要保证他们的docker服务在同一个网络下。在本次试验中，我们开始创建了网络 fabric-ca 来保证这一点。

```
version: "3"
networks:
  fabric-ca:
    external:
      name: fabric-ca
services:
  cli-org1:
    container_name: cli-org1
    image: hyperledger/fabric-tools:amd64-2.2.0
    tty: true
    stdin_open: true
    environment:
      - GOPROXY=https://goproxy.cn,direct
      - GO111MODULE=on
      - GOPATH=/opt/gopath
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      - FABRIC_LOGGING_SPEC=DEBUG
      - CORE_PEER_ID=cli-org1
      # peer服务对应的位置和peer的端口
      - CORE_PEER_ADDRESS={HOSTNAME}:7051
      - CORE_PEER_LOCALMSPID=org1MSP
      - CORE_PEER_TLS_ENABLED=true
      # tls根证书
      - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/org1/peer2/tls-
msp/tlscacerts/tls-172-16-4-35-7052.pem
      # tls证书
      - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/org1/peer2/tls-
msp/signcerts/cert.pem
      # 存储的密钥
      - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/org1/peer2/tls-
msp/keystore/key.pem
      - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/org1/peer2/msp
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/org1
    command: /bin/bash
    volumes:
      - ${HOME}/fabric/org1/peer2:/etc/hyperledger/org1/peer2
      -
      ${HOME}/fabric/org1/peer2/assets/chaincode:/opt/gopath/src/github.com/hyperledger
samples/chaincode
      - ${HOME}/fabric/org1/admin:/etc/hyperledger/org1/admin
    networks:
      - fabric-ca
```

配置的参数和之前大志类似，这边就不过多赘述

## 添加通道

如果证书和配置全部正确，此时我们可以加入通道了。我们通过CLI容器来把peer节点加入通道

```
docker exec -it cli-org1 /bin/bash
```

我们通过指令 `peer channel list` 可以查看此时加入的通道信息。

之后，由于我们需要加入通道，此时我们需要使用admin的MSP来证明我们的身份

```
//设置身份
export CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/org1/admin/msp
//添加通道对应的区块
peer channel join -b mychannel.block
//查看加入的通道
peer channel list
```

此时我们查看加入的通道，可以看到

```
Channels peers has joined:
mychannel
```

同时查看peer容器的日志，可以看到：

```
2022-05-20 05:38:30.125 UTC [ledgermgmt] CreateLedger -> INFO 024 Created
ledger [mychannel] with genesis block
2022-05-20 05:38:30.128 UTC [gossip.gossip] JoinChan -> INFO 025 Joining
gossip network of channel mychannel with 1 organizations
```

注意，mychannel.block的位置位于本地路径

为/home/ubuntu/fabric/org1/mychannel.block，由于docker mount的路径中不包括这个路径，需要拷贝到docker mount的路径下，推荐放到\${HOME}/fabric/org1/peer下

加入通道也可以通过peer channel fetch来进行操作

```
peer channel fetch config config_block.pb -o orderer1-org0:7050 --
ordererTLSHostnameOverride orderer1-org0 -c $CHANNEL_NAME --tls --cafile
$ORDERER_CA
```

详见[fetch\\_update](#)

## 最终目录为



```
├── admin
│   ├── fabric-ca-client-config.yaml
│   └── msp
├── ca
│   ├── admin
│   └── crypto
├── cli
│   ├── docker-compose.yml
│   └── test.sh
├── msp
│   ├── admincerts
│   ├── cacerts
│   ├── config.yaml
│   ├── tlscacerts
│   └── user
├── peer2
│   ├── assets
│   ├── docker-compose
│   ├── fabric-ca-client-config.yaml
│   ├── msp
│   └── tls-msp
└── register.sh
```

## 参考文档

---

[fabric官方文档](#)

[docker官方文档](#)

[fabric ca使用手册](#)