**Fabric1.0多机部署**

# 环境准备

## 服务器准备

我们要部署的是4Peer+1Orderer的架构，也就是官方的e2c\_cli架构。为此我们需要准备5台机器。我们可以开5台虚拟机，也可以购买5台云服务器，不管怎么样，我们需要这5台机器网络能够互通，而且安装相同的系统，我们用的是Ubuntu 16.04版。为了方便，我建议先启用1台虚拟机，在其中把准备工作做完，然后基于这台虚拟机，再复制出4台即可。这里是我用到5台Server的主机名（角色）和IP：

|  |  |
| --- | --- |
| orderer.example.com | 192.168.15.129 |
| peer0.org1.example.com | 192.168.15.137 |
| peer1.org1.example.com | 192.168.15.138 |
| peer0.org2.example.com | 192.168.15.139 |
| peer1.org2.example.com | 192.168.15.140 |

## 服务器环境搭建

接下来我们需要准备软件环境，包括Go、Docker、Docker Compose。在此以官网的例子fabric-samples为判断。能启动fabric-samples中的first-network网络，并能执行官网教程的chaincode安装、实例化、查询。则视为环境搭建成功。（单机搭建环境见—《搭建第一个个fabric网络》）

## Fabric源码下载

git clone https://github.com/hyperledger/fabric.git

我们运行的是1.0.0的版本，因为Fabric一直在更新，我们将版本切换至1.0.0的版本。在下载的fabric目录下。

git branch -a

查看所有版本

git checkout -b release-1.0 origin/release-1.0

在本地创建并切换至1.0版本

## Fabric Docker镜像的下载

cd ~/go/src/github.com/hyperledger/fabric/examples/e2e\_cli/

source download-dockerimages.sh -c x86\_64-1.0.0 -f x86\_64-1.0.0

# Docker-compose文件配置

在Fabric的源码中，提供了单机部署4Peer+1Orderer的示例，在Example/e2e\_cli文件夹中。我们可以在其中一台机器上运行单机的Fabric实例，确认无误后，在该机器上，生成公私钥，修改该机器中的Docker-compose配置文件，然后把这些文件分发给另外4台机器。我们就以orderer.example.com这台机器为例

## 单机运行4+1 Fabric实例，确保脚本和镜像正常

./network\_setup.sh up

这个命令可以在本机启动4+1的Fabric网络并且进行测试，跑Example02这个ChainCode。我们可以看到每一步的操作，最后确认单机没有问题。确认我们的镜像和脚本都是正常的，我们就可以关闭Fabric网络，继续我们的多机Fabric网络设置工作。关闭Fabric命令：

./network\_setup.sh down

在这一步要注意以下几点

1. 提示操作权限不够提示permission denied时，给Fabric赋权限，chmod -R 755/go/src/github.com/hyperledger/fabric。
2. 确保Fabric和镜像的版本一致。
3. 若版本一致的情况下运行出错，删除所有其他无关的镜像，然后重新再执行下载镜像的操作。
4. 运行时产生的channel-artifacts文件夹下的文件全部删除，文件夹保留。产生crypto-config文件夹直接删除（确保之后生成的证书交易配置文件5个服务器相同）。
5. 修改/base中的文件peer-base.yaml 文件，- CORE\_VM\_DOCKER\_HOSTCONFIG\_NETWORKMODE=e2ecli\_default 改为 e2e\_cli\_default，否则报错。如果为fabric1.1.0的镜像则需要修改为e2e\_default

删除镜像按以下操作。

删除指定ID镜像

docker rmi +imagesID

要删除全部image的

docker rmi $(docker images -q)

若无法删除尝试先通过删除imageName，然后再删除imagesID

删除imageName

docker rmi +imagesName

若还无法删除，就先停止所有的container

docker stop $(docker ps -a -q)

再删除所有container，然后再删除镜像

docker rm $(docker ps -a -q)

## 生成公私钥、证书、创世区块等

公私钥和证书是用于Server和Server之间的安全通信，另外要创建Channel并让其他节点加入Channel就需要创世区块，这些必备文件都可以一个命令生成，官方已经给出了脚本：

./generateArtifacts.sh mychannel

运行这个命令后，系统会创建channel-artifacts文件夹，里面包含了mychannel这个通道相关的文件，另外还有一个crypto-config文件夹，里面包含了各个节点的公私钥和证书的信息。（要保证5台服务器这两个文件夹中的文件）

## 设置第一个peer节点的docker-compose文件

e2e\_cli中提供了多个yaml文件，我们可以基于docker-compose-cli.yaml文件创建：

cp docker-compose-cli.yaml docker-compose-peer.yaml

然后修改docker-compose-peer.yaml，去掉orderer的配置，只保留一个peer和cli，因为我们要多级部署，节点与节点之前又是通过主机名通讯，所以需要修改容器中的host文件，也就是extra\_hosts设置，修改后的peer配置如下：

peer0.org1.example.com:

  container\_name: peer0.org1.example.com

  extends:

    file:  base/docker-compose-base.yaml

    service: peer0.org1.example.com

  extra\_hosts:

   - orderer.example.com:192.168.15.129

同样，cli也需要能够和各个节点通讯，所以cli下面也需要添加extra\_hosts设置，去掉无效的依赖，并且去掉command这一行（去掉或注释！），因为我们是每个peer都会有个对应的客户端，也就是cli，所以我只需要去手动执行一次命令，而不是自动运行。修改后的cli要修改部分的配置如下：

depends\_on:

- peer0.org1.example.com

extra\_hosts:

- orderer.example.com:192.168.15.129

- peer0.org1.example.com:192.168.15.137

- peer1.org1.example.com:192.168.15.138

- peer0.org2.example.com:192.168.15.139

- peer1.org2.example.com:192.168.15.140

在单击模式下，4个peer会映射主机不同的端口，但是我们在多机部署的时候是不需要映射不同端口的，所以需要修改base/docker-compose-base.yaml文件，将所有peer的端口映射都改为相同的：

ports:

- 7051:7051

- 7052:7052

- 7053:7053

## 设置orderer节点的docker-compose文件

注：在进行这一步时，我选择的是克隆前面操作的虚拟机。克隆操作能达到所有的文件全部一样，只需修改少许文件。

与创建peer的配置文件类似，我们也复制一个yaml文件出来进行修改：

cp docker-compose-cli.yaml docker-compose-orderer.yaml

orderer服务器上我们只需要保留order设置，其他peer和cli设置都可以删除。orderer可以不设置extra\_hosts。

文件如下：

services:

orderer.example.com:

extends:

file: base/docker-compose-base.yaml

service: orderer.example.com

container\_name: orderer.example.com

## 设置其他peer节点的docker-compose文件

我们在前面配置的就是peer0.org1.example.com上的节点，所以复制过来后不需要做任何修改。因为服务器是通过克隆得到的，所以只需在docker-compose-peer.yaml基础上做修改就好。现在设置peer1.org1.example.com，因为peer1.org1.example.com与peer0.org1.example.com同属于一个组织，并且peer0.org1.example.com为锚节点，所以peer1.org1.example.com在extra\_hosts上需要添加peer0.org1.example.com依赖。其他修改的地方为，将服务和MSP、证书文件修改为peer1的路径（只需将peer0修改为peer1），最终修改后的文件如下。

services:

peer1.org1.example.com:

container\_name: peer1.org1.example.com

extends:

file: base/docker-compose-base.yaml

service: peer1.org1.example.com

extra\_hosts:

- orderer.example.com:192.168.15.129

- peer0.org1.example.com:192.168.15.137

cli:

container\_name: cli

image: hyperledger/fabric-tools

tty: true

environment:

- GOPATH=/opt/gopath

- CORE\_VM\_ENDPOINT=unix:///host/var/run/docker.sock

- CORE\_LOGGING\_LEVEL=DEBUG

- CORE\_PEER\_ID=cli

- CORE\_PEER\_ADDRESS=peer1.org1.example.com:7051

- CORE\_PEER\_LOCALMSPID=Org1MSP

- CORE\_PEER\_TLS\_ENABLED=true

- CORE\_PEER\_TLS\_CERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer1.org1.example.com/tls/server.crt

- CORE\_PEER\_TLS\_KEY\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer1.org1.example.com/tls/server.key

- CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer1.org1.example.com/tls/ca.crt

- CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

working\_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer

# command: /bin/bash -c './scripts/script.sh ${CHANNEL\_NAME}; sleep $TIMEOUT'

volumes:

- /var/run/:/host/var/run/

- ../chaincode/go/:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode/go

- ./crypto-config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/

- ./scripts:/opt/gopath/src/github.com/hyperledger/fabric/peer/scripts/

- ./channel-artifacts:/opt/gopath/src/github.com/hyperledger/fabric/peer/channel-artifacts

depends\_on:

- peer1.org1.example.com

extra\_hosts:

- orderer.example.com:192.168.15.129

- peer0.org1.example.com:192.168.15.137

- peer1.org1.example.com:192.168.15.138

- peer0.org2.example.com:192.168.15.139

- peer1.org2.example.com:192.168.15.140

之后peer0.org2.example.com根据peer0.org1.example.com，peer1.org2.example.com根据peer1.org1.example.com进行修改docker-compose-peer.yaml文件。

# 启动Fabric网络

现在所有文件都已经准备完毕，我们可以启动我们的Fabric网络了。

## 启动orderer

让我们首先来启动orderer节点，在orderer服务器上运行：

docker-compose -f docker-compose-orderer.yaml up –d

运行完毕后我们可以使用docker ps看到运行了一个名字为orderer.example.com的节点。

## 启动peer

然后我们切换到peer0.org1.example.com服务器，启动本服务器的peer节点和cli，命令为：

docker-compose -f docker-compose-peer.yaml up –d

运行完毕后我们使用docker ps应该可以看到2个正在运行的容器。

接下来依次在另外3台服务器运行启动peer节点容器的命令：

docker-compose -f docker-compose-peer.yaml up –d

现在我们整个Fabric4+1服务器网络已经成型，接下来是创建channel和运行ChainCode。

## 创建Channel测试ChainCode

我们切换到peer0.org1.example.com服务器上，使用该服务器上的cli来运行创建Channel和运行ChainCode的操作。首先进入cli容器：

docker exec -it cli bash

进入容器后我们可以看到命令提示变为：

[/opt/gopath/src/github.com/hyperledger/fabric/peer](mailto:root@b41e67d40583:/opt/gopath/src/github.com/hyperledger/fabric/peer)#

之后创建channel:

export CHANNEL\_NAME=mychannel

peer channel create -o orderer.example.com:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/channel.tx --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

之后的操作跟《搭建第一个个fabric网络》中的安装、实例化、操作链码一样。

1. 当前节点加入通道

peer channel join -b mychannel.block

1. 当前节点安装链码

peer chaincode install -n mycc -v 1.0 -p github.com/hyperledger/fabric/examples/chaincode/go/chaincode\_example02

1. 当前节点实例化链码

peer chaincode instantiate -o orderer.example.com:7050 --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n mycc -v 1.0 -c '{"Args":["init","a", "100", "b","200"]}' -P "OR ('Org1MSP.member','Org2MSP.member')"

1. 执行链码查询操作

peer chaincode query -C $CHANNEL\_NAME -n mycc -c '{"Args":["query","a"]}'

查询结果为100

# 多机联动测试

## 查询节点当前channel列表

切换至peer1.org1.example.com服务器，进入cli容器

docker exec -it cli bash

通过命令查看当前节点channel列表

peer channel list

发现当前节点无mychannel，执行命令ls，发现并无mychannel.block文件。所以需要获取mychannel.block文件。

## 获取mychannel.block文件

获取mychannel.block文件有两种方式。

一种是创建mychannel时会在当前目录下生成mychannel.block文件，可以通过复制粘贴的方式，复制到其他需要此文件的节点服务器中。因为创建时是处于docker容器中，所以需要docker容器和宿主机器之间文件的复制。

Docker容器文件复制到宿主机器：

docker cp de80ea6e859e:/opt/gopath/src/github.com/hyperledger/fabric/peer/mychannel.block /root

de80ea6e859e为docker容器的ID，后面为需要复制文件的路径，最后是将文件复制至宿主机器的具体位置

Docker宿主机器文件复制到容器：

docker cp /root/mychannel.block de80ea6e859e:/opt/gopath/src/github.com/hyperledger/fabric/peer/

还有一种方式则是通过访问orderer节点，获取channel的创世区块配置文件。

export ORDERER\_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

export CHANNEL\_NAME=mychannel

peer channel fetch 0 mychannel.block -o orderer.example.com:7050 -c $CHANNEL\_NAME --tls --cafile $ORDERER\_CA

## 节点加入channel

peer channel join -b mychannel.block

## 节点安装链码

peer chaincode install -n mycc -v 1.0 -p github.com/hyperledger/fabric/examples/chaincode/go/chaincode\_example02

## 执行链码查询操作

链码只需实例化一次，在peer0.org1.example.com节点中已经实例过，因此之后使用此链码的节点只需安装链码即可。

首先执行查询操作

peer chaincode query -C $CHANNEL\_NAME -n mycc -c '{"Args":["query","a"]}'

查询结果为100

## 执行链码修改操作

peer chaincode invoke -o orderer.example.com:7050 --tls $CORE\_PEER\_TLS\_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n mycc -c '{"Args":["invoke","a","b","10"]}'

执行成功后再执行查询操作

peer chaincode query -C $CHANNEL\_NAME -n mycc -c '{"Args":["query","a"]}'

查询结果为90

## 其他节点确认结果

切换到peer0.org1.example.com服务器，执行查询操作

peer chaincode query -C $CHANNEL\_NAME -n mycc -c '{"Args":["query","a"]}'

查询结果为80