**[Fabric 1.0的多机部署](http://www.cnblogs.com/studyzy/p/7237287.html)**

Fabric1.0已经正式发布一段时间了，官方给出的单机部署的脚本也很完备，基本上傻瓜式的一键部署，直接运行官方的network\_setup.sh up即可。但是在实际生产环境，我们不可能把所有的节点都放在一台机器中，所以必然会遇到多级部署的问题。下面我们就来讲讲怎么实现多机部署和测试官方的ChainCode。

**1.环境准备**

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 | 10.174.13.185 |
| peer0.org1.example.com | 10.51.120.220 |
| peer1.org1.example.com | 10.51.126.19 |
| peer0.org2.example.com | 10.51.116.133 |
| peer1.org2.example.com | 10.51.126.5 |

接下来我们需要准备软件环境，包括Go、Docker、Docker Compose，我在之前的[单机部署的博客](http://www.cnblogs.com/studyzy/p/6973334.html)中也讲到过具体的方法，这里再复述一下：

**1.1 Go的安装**

Ubuntu的apt-get虽然提供了Go的安装，但是版本比较旧，最好的方法还是参考官方网站，下载最新版的Go。具体涉及到的命令包括：

wget https://storage.googleapis.com/golang/go1.8.3.linux-amd64.tar.gz

sudo tar -C /usr/local -xzf go1.8.3.linux-amd64.tar.gz

接下来编辑当前用户的环境变量：

vi ~/.profile

添加以下内容：

export PATH=$PATH:/usr/local/go/bin

export GOROOT=/usr/local/go

export GOPATH=$HOME/go

export PATH=$PATH:/$HOME/go/bin

我们把go的目录GOPATH设置为当前用户的文件夹下，所以记得创建go文件夹

cd ~

mkdir go

**1.2 Docker安装**

我们可以使用阿里提供的镜像，安装也非常方便。通过以下命令来安装Docker

curl -sSL http://acs-public-mirror.oss-cn-hangzhou.aliyuncs.com/docker-engine/internet | sh -

安装完成后需要修改当前用户（我使用的用户叫fabric）权限：

sudo usermod -aG docker fabric

注销并重新登录，然后添加阿里云的Docker Hub镜像：

不同的版本添加方法是不一样的，官方的文档如下：

[**https://cr.console.aliyun.com/#/accelerator**](https://cr.console.aliyun.com/#/accelerator)

当然觉得阿里云镜像不好用，喜欢用DaoClound的也可以用DaoClound的镜像。

**1.3 Docker-Compose的安装**

Docker-compose是支持通过模板脚本批量创建Docker容器的一个组件。在安装Docker-Compose之前，需要安装Python-pip，运行脚本：

sudo apt-get install python-pip

安装完成后，接下来从DaoClound安装Docker-compose，运行脚本：

curl -L https://get.daocloud.io/docker/compose/releases/download/1.10.1/docker-compose-`uname -s`-`uname -m` > ~/docker-compose

sudo mv ~/docker-compose /usr/local/bin/docker-compose

chmod +x /usr/local/bin/docker-compose

**1.4 Fabric源码下载**

我们可以使用Git命令下载源码，也可以使用go get命令，偷懒一点，我们直接用go get命令获取最新的Fabric源码：

go get github.com/hyperledger/fabric

这个可能等的时间比较久，等完成后，我们可以在~/go/src/github.com/hyperledger/fabric中找到所有的最新的源代码。

由于Fabric一直在更新，所有我们并不需要最新最新的源码，需要切换到v1.0.0版本的源码即可：

cd ~/go/src/github.com/hyperledger/fabric

git checkout v1.0.0

**1.5 Fabric Docker镜像的下载**

这个其实很简单，因为我们已经设置了Docker Hub镜像地址，所以下载也会很快。官方文件也提供了批量下载的脚本。我们直接运行：

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

这样就可以下载所有需要的Fabric Docker镜像了。

**2.docker-compose 配置文件准备**

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

**2.1单机运行4+1 Fabric实例，确保脚本和镜像正常**

我们先进入这个文件夹，然后直接运行

./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

**2.2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。

**2.3设置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：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | 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:10.174.13.185" |

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，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配置如下：

[](javascript:void(0);)

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=peer0.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/peer0.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/peer0.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/peer0.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

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:

- peer0.org1.example.com

extra\_hosts:

- "orderer.example.com:10.174.13.185"

- "peer0.org1.example.com:10.51.120.220"

- "peer1.org1.example.com:10.51.126.19"

- "peer0.org2.example.com:10.51.116.133"

- "peer1.org2.example.com:10.51.126.5"

[](javascript:void(0);)

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

**2.4设置orderer节点的docker-compose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。

**2.5分发配置文件**

前面4步的操作，我们都是在orderer.example.com上完成的，接下来我们需要将这些文件分发到另外4台服务器上。Linux之间的文件传输，我们可以使用scp命令。

我先登录peer0.org1.example.com，将本地的e2e\_cli文件夹删除：

rm e2e\_cli –R

然后再登录到orderer服务器上，退回到examples文件夹，因为这样可以方便的把其下的e2e\_cli文件夹整个传到peer0服务器上。

scp -r e2e\_cli fabric@10.51.120.220:/home/fabric/go/src/github.com/hyperledger/fabric/examples/

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。

再次运行scp命令，复制到peer1.org1.example.com上，然后我们需要对docker-compose-peer.yaml做一个小小的修改，将启动的容器改为peer1.org1.example.com，并且添加peer0.org1.example.com的IP映射，对应的cli中也改成对peer1.org1.example.com的依赖。这是修改后的peer1.org1.example.com上的配置文件：

[](javascript:void(0);)

version: '2'

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:10.174.13.185"

- "peer0.org1.example.com:10.51.120.220"

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

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:10.174.13.185"

- "peer0.org1.example.com:10.51.120.220"

- "peer1.org1.example.com:10.51.126.19"

- "peer0.org2.example.com:10.51.116.133"

- "peer1.org2.example.com:10.51.126.5"

[](javascript:void(0);)

接下来继续使用scp命令将orderer上的文件夹传送给peer0.org2.example.com和peer1.org2.example.com，然后也是修改一下docker-compose-peer.yaml文件，使得其启动对应的peer节点。

**3.启动Fabric**

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。

**3.1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。

**3.2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。

**3.3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：

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

说明我们已经以root的身份进入到cli容器内部。官方已经提供了完整的创建Channel和测试ChainCode的脚本，并且已经映射到cli容器内部，所以我们只需要在cli内运行如下命令：

./scripts/script.sh mychannel

那么该脚本就可以一步一步的完成创建通道，将其他节点加入通道，更新锚节点，创建ChainCode，初始化账户，查询，转账，再次查询等链上代码的各个操作都可以自动化实现。直到最后，系统提示：

===================== All GOOD, End-2-End execution completed =====================

说明我们的4+1的Fabric多级部署成功了。我们现在是在peer0.org1.example.com的cli容器内，我们也可以切换到peer0.org2.example.com服务器，运行docker ps命令，可以看到本来是2个容器的，现在已经变成了3个容器，因为ChainCode会创建一个容器：

docker ps

CONTAINER ID        IMAGE                                 COMMAND                  CREATED             STATUS              PORTS                              NAMES

add457f79d57        dev-peer0.org2.example.com-mycc-1.0   "chaincode -peer.a..."   11 minutes ago      Up 11 minutes                                          dev-peer0.org2.example.com-mycc-1.0

0c06fb8e8f20        hyperledger/fabric-tools              "/bin/bash"              13 minutes ago      Up 13 minutes                                          cli

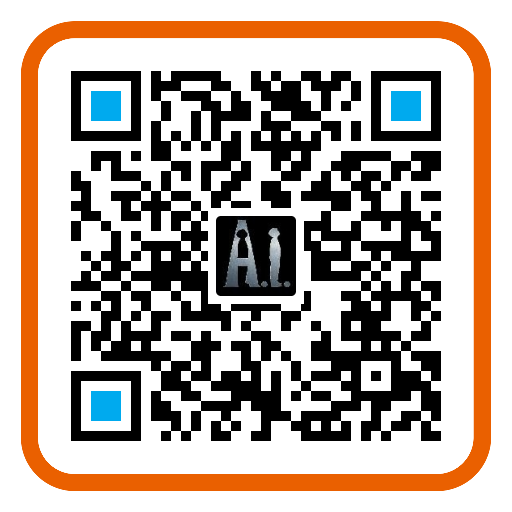
632c3e5d3a5e        hyperledger/fabric-peer               "peer node start"        13 minutes ago      Up 13 minutes       0.0.0.0:7051-7053->7051-7053/tcp   peer0.org2.example.com

**4.总结**

我在Fabric多机部署的过程中还是遇到了不少坑，前前后后花了2天的时间才趟坑完毕，实现了最终的4+1多机部署。其中与单机部署最大的不同的地方就是在单机部署的时候，我们是在同一个docker网络中，所以相互之间通过主机名通讯很容易，而在多机环境中，就需要额外设置DNS或者就是通过extra\_hosts参数，设置容器中的hosts文件。而且不能一股脑的就跟cli一样把5台机器的域名IP配置到peer中，那样会报错的，所以只需要设置需要的即可。

官方给的脚本已经替我们做了很多工作，同时也隐藏了很多细节，所以我们并没有真正了解其内部的实现过程，我以后会再写一篇博客详细介绍Fabric多机部署的详细过程。为了方便，我把设计到的几个docker-compose文件打包了一份放出来，如果大家想进行同样的部署，只需要修改一下IP即可复用。

[Docker-compose下载](http://files.cnblogs.com/files/studyzy/fabric4peer1orderer.zip)



支付宝打赏

[了解更多](https://github.com/greedying/tctip)