基于Hyperledger fabric 区块链与IPFS 分布式系统网络的部署步骤

**1.fabric开发环境搭建**

本文在开发过程中，采用Ubuntu22.04操作系统。在搭建系统开发环境之前，用户首先要安装Golang，并进行相关环境的配置工作。系统环境的配置大致分为下列四个步骤。

首先，用户需要选择一个版本超过7.0的Go语言安装包进行下载解压并安装。在本文中，采用 1.18.3 的 Go 语言版本来查询系统安装是否成功。如若出现下图所示指令，则证明

安装成功。



图1-1 Go 语言版本安装成功示意图

（1）创建go目录并配置环境：

mkdir $HOME/go

#用vi打开~./bashrc，配置环境变量

Vi ~/.bashrc

# 在最下方插入

export GOROOT=/usr/local/go

export GOPATH=$HOME/go

export PATH=$PATH:$GOROOT/bin:$GOPATH/bin

#使配置的环境变量生效

source ~/.bashrc

#检查是否配置正确

go version

# 配置goproxy环境变量加速国内下载

go env -w GOPROXY=https://goproxy.io

（2）在此系统中，开发者需要安装 Docker、Docker compose、cURL以及 Git四个辅助工具。其中，Docker为Fabric提供一个独立安全的运行环境，并便于对系统进行迁移。

安装git

sudo apt-get install git

安装curl

sudo apt-get install curl

安装docker

# 安装并检查版本

sudo apt-get -y install docker-compose

docker --version

docker-compose --version

# 提示，碰到docker有关的错误，先执行下列2.3.1步骤

#1.重启docker

sudo systemctl start docker

#设置系统启动时docker启动，可选

sudo systemctl enable docker

#2.将用户添加到docker组，确保在用户命令下可以执行

sudo gpasswd -a $USER docker

#3.更新用户组

newgrp docker

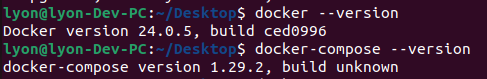
# docker信息

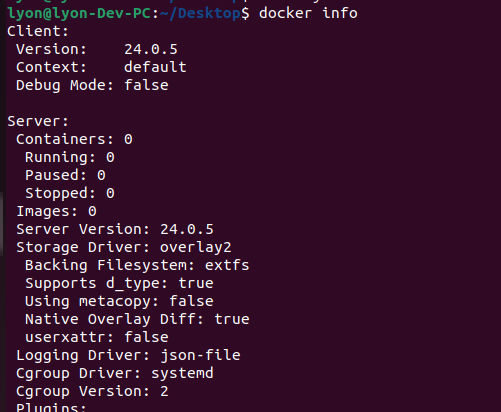
docker info

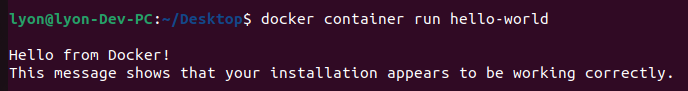
# 测试docker

docker container run hello-world

下图为Docker compose、Docker、Git安装以及docker测试示意图。







**安装示例、二进制和 Docker 镜像**

克隆 hyperledger/fabric-samples 仓库

mkdir -p $GOPATH/src/github.com/hyperledger

cd $GOPATH/src/github.com/hyperledger

# 获取fabric-samples源码

git clone <https://github.com/hyperledger/fabric-samples.git>

选择适当的版本标签，进入目录，切换分支

cd fabric-samples

# 可自行选择版本

git checkout release-2.2

#查看版本

git branch

将指定版本的 Hyperledger Fabric 平台特定二进制文件和配置文件安装到 fabric-samples 下的 /bin和 /config 目录中，下载指定版本的 Hyperledger Fabric docker 镜像。

先配置镜像源：

sudo vi /etc/docker/daemon.json

#把以下代码加进去

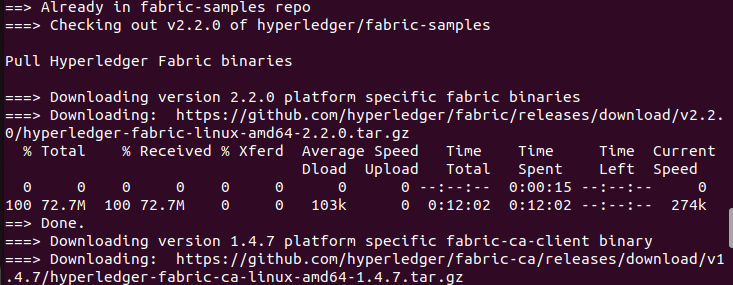
{

"registry-mirrors":["https://registry.docker-cn.com"]

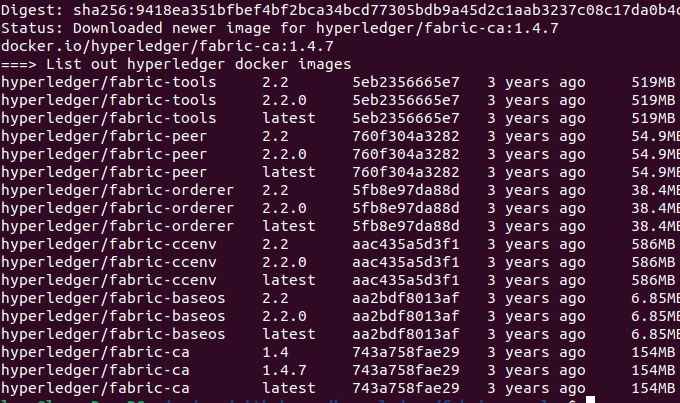
}

执行

curl -sSL https://raw.githubusercontent.com/hyperledger/fabric/master/scripts/bootstrap.sh | bash -s 2.2.0 1.4.7 0.4.18

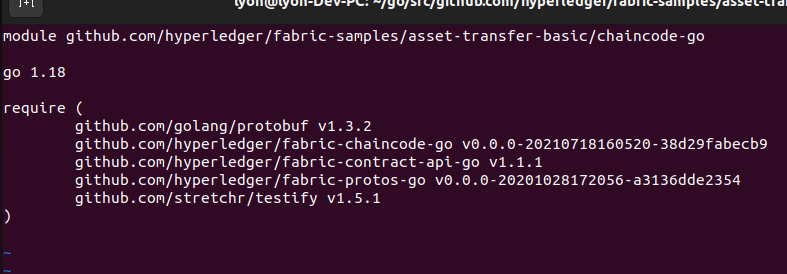


可以看到全部下载完毕了



cd asset-transfer-basic/chaincode-go

sudo vi go.mod *# 进入文件发现是1.14 改成1.18*



环境变量设置

vi ~/.bashrc

# 添加下面的变量

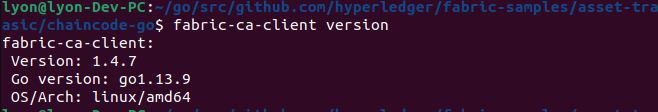
export PATH=$PATH:$GOPATH/src/github.com/hyperledger/fabric-samples/bin

# 使之生效

source ~/.bashrc

# 检验成功否

fabric-ca-client version



**2.使用Fabric测试网络**

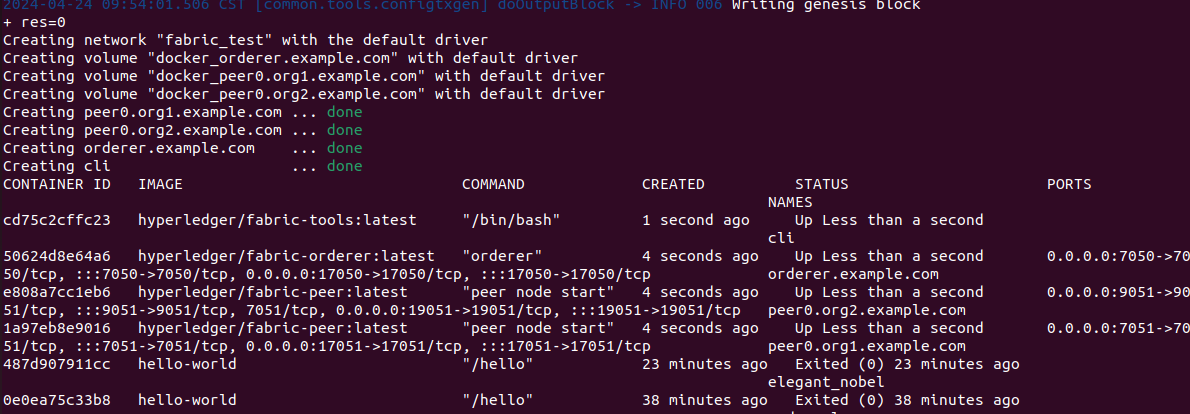
启动测试网络：

进入fabric-samples下的test-network

执行 sudo ./network.sh up

如果没出现错误，说明你的fabric已经搭建好了。

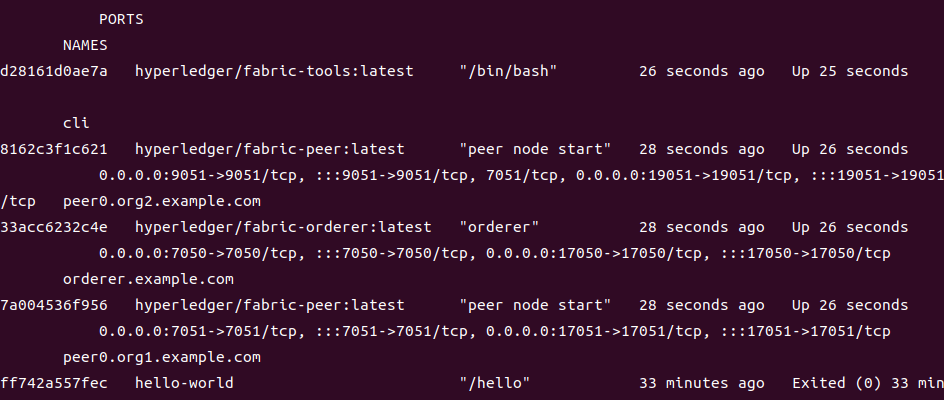
执行完毕后看到，我们创建了排序组织orderer、联盟成员org1的peer0节点和联盟成员org2的peer0节点以及相应的镜像image。



测试网络的组成部分

执行docker ps -a

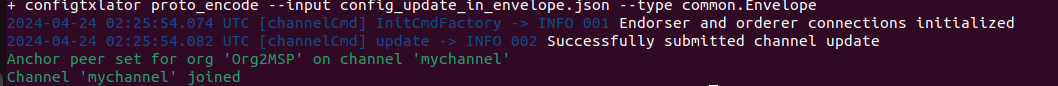
可以看到创建的三个节点和之前测试的hello-world



Fabric网络成员的所有组织称为联盟，此测试网络有两个联盟成员org1、2，一个维护网络排序服务的组织orderer，每个组织各运营一个对等节点，peer0.org1.example.com和peer0.org2.example.com

创建一个channel

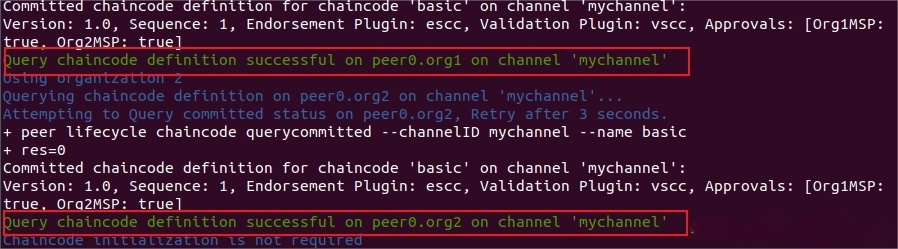
使用./network.sh createChannel 在org1和org2之间创建通道并加入他们的对等节点。



在通道启动一个链码

./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go -ccl go

如下图就是链码启动成功了



与网络交互

把fabric-samples的bin文件下的二进制文件添加到CLI路径：

export PATH=${PWD}/../bin:$PATH

还需要将fabric-samples代码库中的FABRIC\_CFG\_PATH设置为指向其中的core.yaml文件：

export FABRIC\_CFG\_PATH=$PWD/../config/

设置环境变量，允许用户作为org1操作peer的CLI：

# Environment variables for Org1

export CORE\_PEER\_TLS\_ENABLED=true

export CORE\_PEER\_LOCALMSPID="Org1MSP"

#CORE\_PEER\_TLS\_ROOTCERT\_FILE和CORE\_PEER\_MSPCONFIGPATH环境变量指向Org1的organizations文件夹中的的加密材料。

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

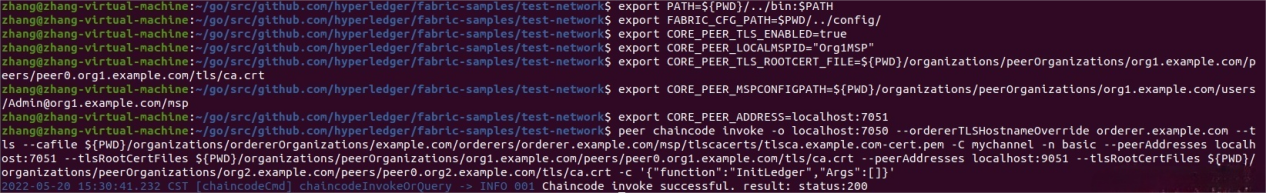
export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

export CORE\_PEER\_ADDRESS=localhost:7051

调用链码（Go）的 InitLedger 方法来赋予一些账本上的初始资产，运行如下命令，用一些资产来初始化账本：

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 mychannel -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":[]}'

如图，初始化成功！



至此测试完毕，关闭网络./network.sh down.

**3.ipfs环境搭建**

下载安装包

Ubuntu下安装，所以下载go-ipfs的地址为：

<https://dist.ipfs.io/go-ipfs/v0.4.15/go-ipfs_v0.4.15_linux-amd64.tar.gz>

安装

将压缩包上传到根目录，如：/home/simon/，然后解压。

$ tar xvfz go-ipfs\_v0.4.15\_linux-amd64.tar.gz

go-ipfs/build-log

go-ipfs/install.sh

go-ipfs/ipfs

go-ipfs/LICENSE

go-ipfs/README.md

$ cd go-ipfs

$ sudo mv ipfs /usr/local/bin/ipfs

$ ipfs version #查看版本

ipfs version 0.4.15

项目配置

为了运行项目，我们需要通过在本地建立一个IPFS节点。

$ ipfs init

$ cd /home/simon/

$ ll -a

drwxrwxr-x 5 simon simon 4096 4月 15 18:39 .ipfs/ #多出了这样一个文件夹

#......其它文件，略

ipfs在目录/home/simon/下创建一个.ipfs文件夹，通过ll -a可以查看到。

修改节点默认存储空间

假设我们现在需要将本地节点的可使用硬盘容量设置成20G

$ cd /home/simon/.ipfs

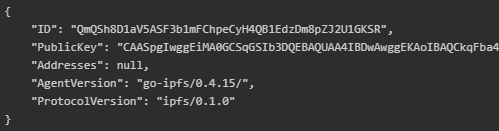
$ export EDITOR=/usr/bin/vim #定义默认编辑器，也可以修改成/usr/bin/vi

$ ipfs config edit #修改ipfs配置文件

搜索StorageMax关键词，定位到第7行，将默认的10GB修改成20GB，保存即可。

查看节点id

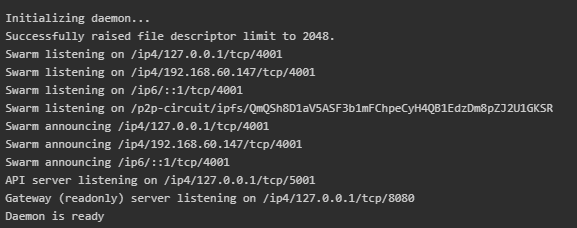
$ ipfs id



QmQSh8D1aV5ASF3b1mFChpeCyH4QB1EdzDm8pZJ2U1GKSR即我们的节点ID，与我们的身份证一样，表示节点在ipfs网络中的唯一标识。

启动节点服务器

$ ipfs daemon



此时我们已经启动了节点服务器，本地节点数据在同步到IPFS网络中。

##### 跨域资源共享CORS配置

$ ipfs config --json API.HTTPHeaders.Access-Control-Allow-Methods '["PUT", "GET", "POST", "OPTIONS"]'

$ ipfs config --json API.HTTPHeaders.Access-Control-Allow-Origin '["\*"]'

$ ipfs config --json API.HTTPHeaders.Access-Control-Allow-Credentials '["true"]'

$ ipfs config --json API.HTTPHeaders.Access-Control-Allow-Headers '["Authorization"]'

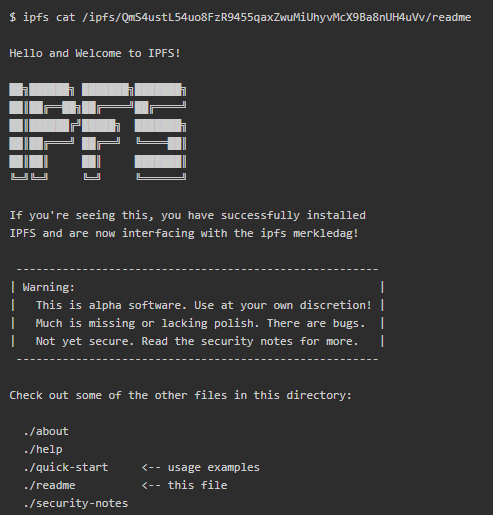
$ ipfs config --json API.HTTPHeaders.Access-Control-Expose-Headers '["Location"]'

##### 验证

再次启动节点服务器

$ ipfs daemon

新开一个终端窗口



在Ubuntu中，在浏览器中输入下面的网址http://localhost:5001/webui，可以看到我们本地的节点信息，以及全球的节点信息，也可以查询到时我们上面文件的哈希对应的文件内容。