**实验8 Fabric环境搭建与案例运行**

**【实验介绍】**

本次实验从环境安装以及部署开始，在之后会具体讲解Fabric环境部署的每一个步骤，建议读者按照这些步骤完成安装工作，以更进一步了解Fabric和它的依赖环境的细节，以及待加强学习的部分。环境的安装部署是一个较繁琐的过程，其中可能会出现各种的错误，比如，配置文件设置错误、路径设置错误、网络错误等，这些均会导致安装的失败。而这一过程也正是提高对系统熟悉程度以及系统操作熟练度的机会。另外，实验已预先提供可导入虚拟机的系统文件。值得说明的是，在Fabric的相关实验中，用到的宿主机环境是Ubuntu16.04。

最后，在完成环境搭建工作后，建议读者尝试运行官方给的End2End案例，一方面测试一下环境搭建是否成功；另一方面通过运行案例，可以对联盟链的拓扑关系以及逻辑架构有一个更加深入的理解。

在进行具体的实验之前，以下的预备知识被认为会对实验的推进起到帮助。

（1）Fabric依赖环境

本次Fabric实验中涉及的依赖环境主要包括：

1）Git：它是一个开源的版本控制软件，可以高效地处理各种规模的项目版本管理；

2）Docker：它是一个使用Golang开发的应用容器引擎，其将软件打包为可在任意环境中部署的容器。显然，该容器具有运行软件需要的库及系统工具等；

3）Docker-Compose：它是由Docker官方负责的开源项目，实现了对多个容器的快速编排。

关于Fabric的具体安装流程将在实验准备中详细描述。

（2）Fabric系统架构

HyperLedger Fabric是使用Golang实现的一个模块化的区块链平台。它具有部署灵活、可插拔及扩展便捷等特点。与之前介绍的大多区块链平台不同，fabric具有可信任的先决条件，这使得它更适用于联盟链的使用场景。接下来将对它的架构展开进一步的介绍。

Fabric包括如下三个模块：区块链服务、成员服务与链码服务。其中链码即智能合约。

1）区块链模块下主要包括如下四个组件：分布式账本、共识机制、P2P网络与账本存储。Fabric中使用P2P网络框架来维护分布式账本，同时可以根据需求及使用场景配置诸如PBFT、PoW、PoS等不同的共识算法；

2）成员模块下主要包括如下三个组件：注册、审计及身份管理。正如之前介绍的，fabric是联盟链的代表之一，所以其中的节点需要经过许可后才可加入到区块链中。Fabric使用公钥基础设施及共识技术做到了这一点；

3）链码模块下包括两个组件：安全注册及安全容器。Fabric中的链码（即智能合约）可采用Golang、Java等编写，之后将其运行在Docker容器中。

除了三个基本模块外，还有以下的概念也被认为是同等重要的。

1）账本：fabric中的账本包括区块链和状态两个部分。其中区块保存着历史交易，状态是保存着账本最新状态的KV数据库；

2）交易：fabric中的交易包括部署与调用两种。前者表示将链码部署到节点上，后者表示当客户端调用链码来执行交易时，从KV数据库中读取抑或写入数据。

至此，有关fabric的基本概念已介绍完毕。接下来将对其交易流程展开介绍：

1）首先由客户端提交一个交易提案，其包括要调用的链码中的函数、参数及签名等；2）endorser（背书节点）收到提案并确认客户端具有权限后，在数据库内执行交易，之

后将交易结果及endorser签名等信息返回给客户端；

3）客户端验证endorser签名并判断提案的执行与预先指定的规则是否一致之后，将收到的所有endorser的应答打包为一个交易发送给排序节点；

4）排序节点完成收到交易的排序，并将一定数量的交易打包进区块，将该区块发送至提交节点；

5）提交节点在接收到block并对其中的交易检验后，将其添加在区块链的末尾并修改KV数据库的对应数据。

**【实验要求】**

（1）完成Fabric基本环境部署

（2）运行End2End案例以理解Fabric的系统架构

**【实验准备】**

1. **安装Git**

更新系统软件

|  |
| --- |
| sudo apt-get update |

安装Git

|  |
| --- |
| sudo apt-get install git |

查看Git版本信息，正确显示版本号即说明安装成功(后续相关的安装均通过查看版本判断是否安装成功，不再赘述)

|  |
| --- |
| git version |

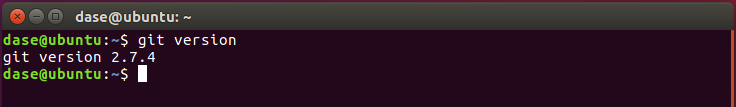


图8-1 git版本显示

1. **安装Docker**

自动安装

|  |
| --- |
| curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun |

查看docker版本

|  |
| --- |
| docker version |

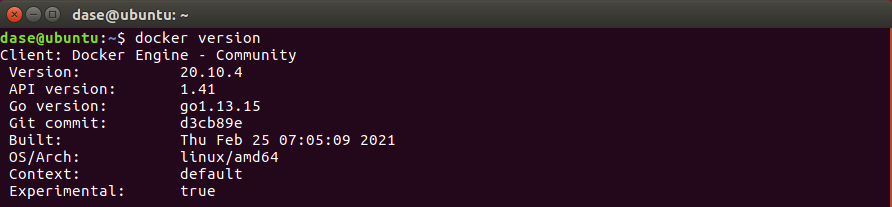


图8-2 docker版本显示

创建docker用户组，如果显示当前用户组已存在，忽略即可，继续下一个步骤

|  |
| --- |
| sudo groupadd docker |

添加当前用户至用户组（${USER}为自己当前的用户名）

|  |
| --- |
| sudo usermod -aG docker ${USER} |

重启docker

|  |
| --- |
| sudo systemctl restart docker |

最后退出当前账户再重新登入即可

1. **安装Docker-Compose**

下载docker-compose

|  |
| --- |
| sudo curl -L "https://github.com/docker/compose/releases/download/1.23.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose |

增加执行权限

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

查看版本

|  |
| --- |
| docker-compose --version |

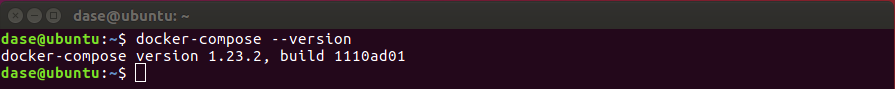


图8-3 docker-compose版本显示

1. **安装Go语言环境**

获取安装包

|  |
| --- |
| curl -O https://storage.googleapis.com/golang/go1.10.1.linux-amd64.tar.gz |

使用tar命令把安装包解压缩/usr/local文件夹下面，要求先删除其他版本go

|  |
| --- |
| rm -rf /usr/local/go  sudo tar -zxvf go1.10.1.linux-amd64.tar.gz -C /usr/local |

设置环境变量

|  |
| --- |
| sudo vim /etc/profile |

在文件末尾添加内容，其中go的安装目录是/usr/local/go，go的工作目录是/home/${USER}/go

|  |
| --- |
| export GOPATH=$HOME/go  export GOROOT=/usr/local/go  export PATH=$GOROOT/bin:$PATH |

使用source命令，使其配置信息生效

|  |
| --- |
| source /etc/profile |

查看go的版本信息

|  |
| --- |
| go version |

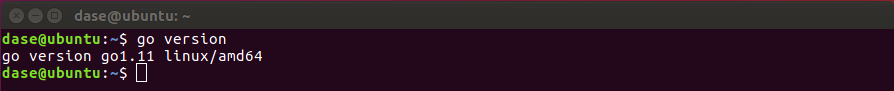


图8-4 Golang版本显示

在GOPATH目录下创建go目录

|  |
| --- |
| mkdir go |

1. **Fabric安装**

进入go项目路径

|  |
| --- |
| cd /home/${USER}/go |

在/home/ ${USER}/go下创建目录

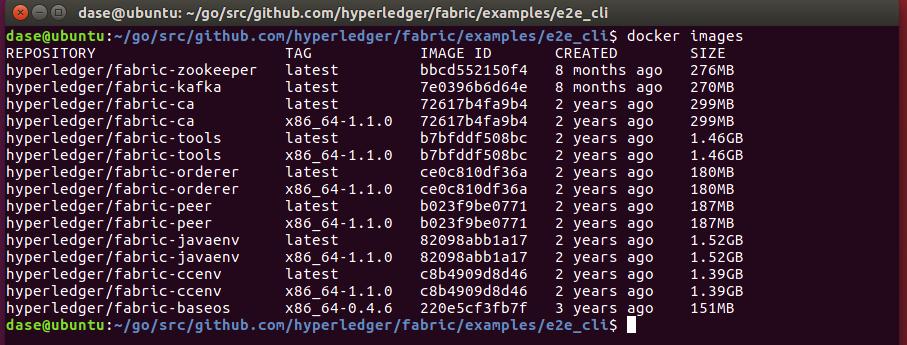
|  |
| --- |
| mkdir -p src/github.com/hyperledger  cd src/github.com/hyperledger  git clone https://github.com/hyperledger/fabric.git |

进入 fabric 目录查看版本分支并切换分支，此处选择对应版本或公开发行版，选择v1.1.0

|  |
| --- |
| cd fabric  git branch  git checkout v1.1.0 |

进入fabric/examples/e2e\_cli/目录，完成镜像下载，执行命令：

|  |
| --- |
| cd fabric/examples/e2e\_cli  source download-dockerimages.sh -c x86\_64-1.1.0 -f x86\_64-1.1.0  docker images |

图8-5 本地镜像的显示

如上图所示，通过docker images命令查看本地镜像，主要包括了CA，排序，节点，以及java & go链码等其他镜像，在实际开发中，就是使用这些镜像来搭建所需的架构。

因为时过境迁，有些镜像已经拉不到x86\_64-1.1.0版本了，比如zookeeper、kafka、couchdb、baseos。缺的镜像可以自己去https://hub.docker.com 里通过复制命令拉取latest版本，baseos使用x86\_64-0.4.6版本。命令如下：

|  |
| --- |
| docker pull hyperledger/fabric-zookeeper:latest  docker pull hyperledger/fabric-kafka:latest  docker pull hyperledger/fabric-couchdb:latest  docker pull hyperledger/fabric-baseos:x86\_64-0.4.6 |

获取最新镜像的方法如图8-6所示：

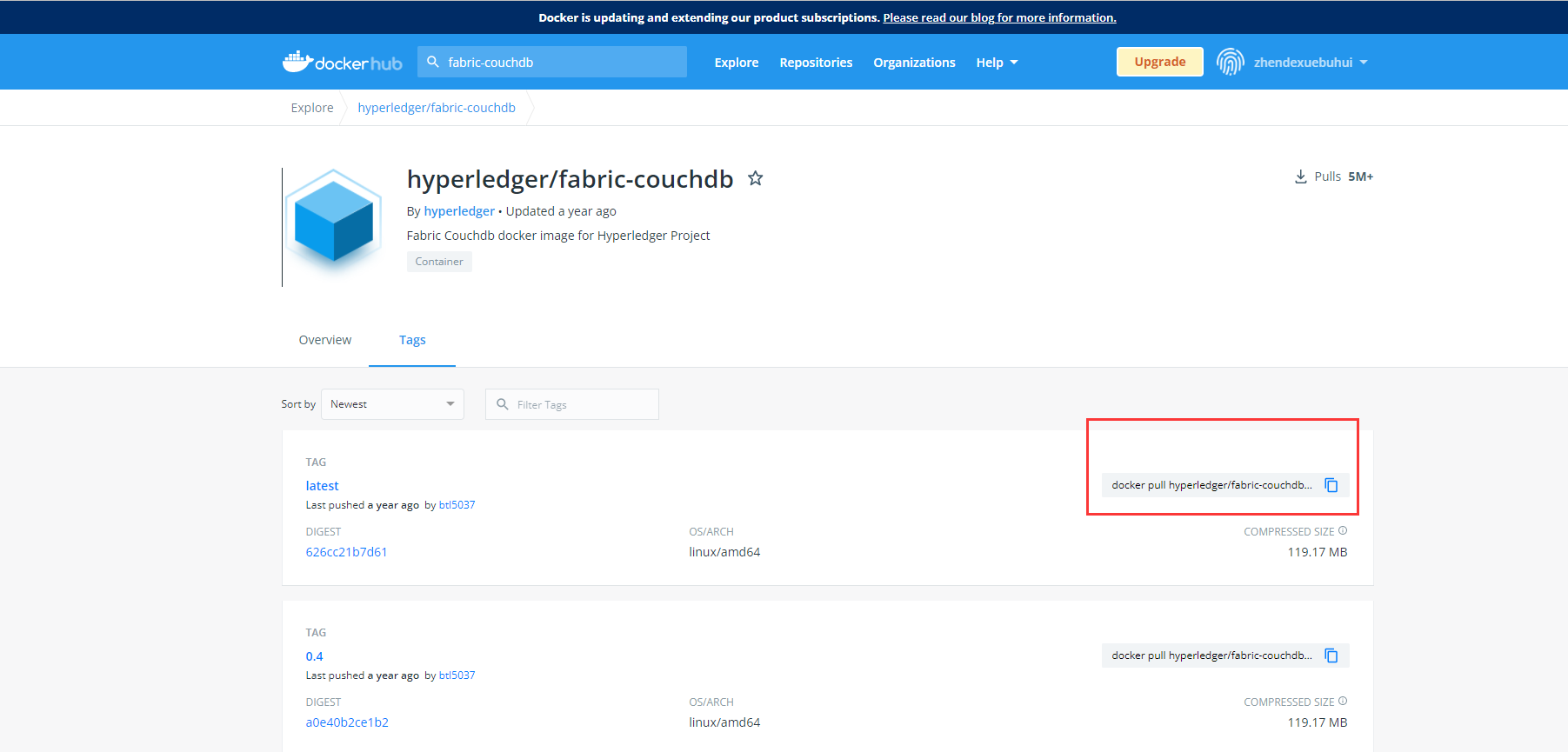


图8-6 获取最新镜像的方法

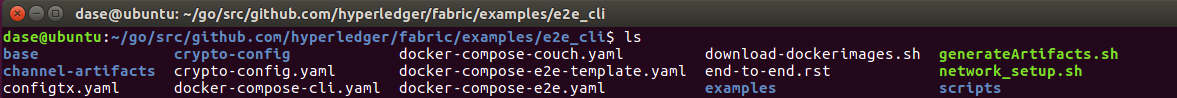
**【实验过程】**

上述步骤完成了Fabric所需环境的部署，接下来，我们通过案例的运行，一方面测试安装是否成功，另一方面通过该案例了解联盟链的拓扑关系。

还是在刚才的e2e\_cli文件夹下：

|  |
| --- |
| cd $GOPATH/src/github.com/hyperledger/fabric/examples/e2e\_cli |

看一下该案例中的相关文件：

图8-7 案例中的文件展示

可以看到包含不少配置文件，其中的docker-\*.yaml是不同网络拓扑的 docker-compose配置文件，crypto-config.yaml是用来给网络节点、CA、用户初始化PKI环境（生成的文件存储在crypto-config目录下），configtx.yaml是用来生成channel和order初始化信息（生成文件存储在channel-artifacts目录下），examples目录下存储的是智能合约代（chaincode)，其余的.sh文件就是这个例子的执行脚本。

修改目录下base/peer-base.yaml文件

|  |
| --- |
| CORE\_VM\_DOCKER\_HOSTCONFIG\_NETWORKMODE=e2e\_cli\_default |

$GOPATH/src/github.com/hyperledger/fabric/examples/e2e\_cli目录下

docker-compose-cli.yaml文件cli下environment中

base/peer-base.yaml文件中environment中

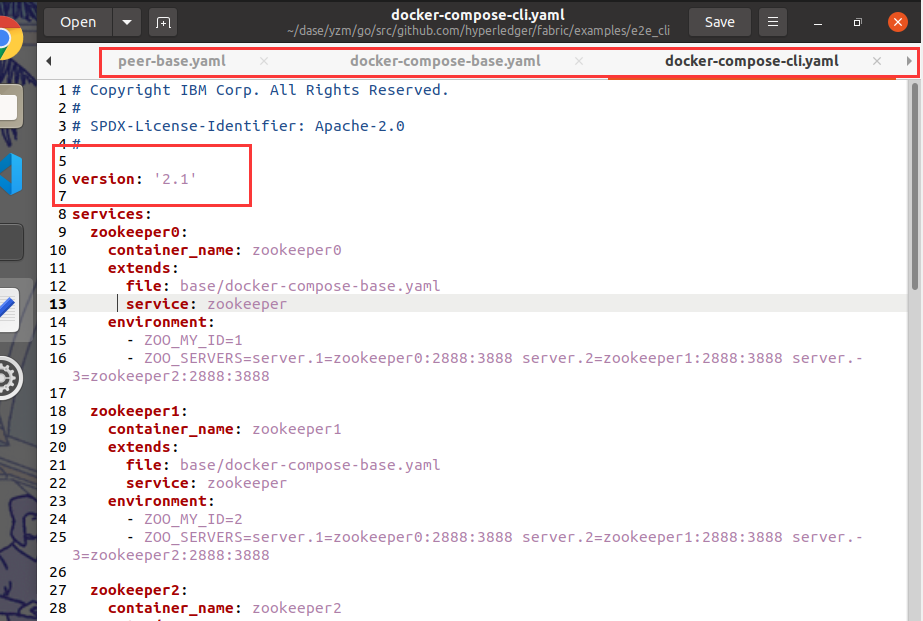
base/docker-compose-base.yaml文件中各odrer和peer的environment中

添加如下语句

|  |
| --- |
| - GODEBUG=netdns=go |

由于docker守护进程的更新，要求客户端至少是1.24版本，以便能够正确地进行通信和执行操作，因此需要对克隆下来的Yaml文件中的docker客户端版本配置进行修改，否则会报错：“client version 1.22 is too old. Minimum supported API version is 1.24, please upgrade your client to a newer version”。

将base/peer-base.yaml、base/docker-compose-base.yaml、docker-compose-cli.yaml中开头的“version: 2”改成“version: 2.1”。



每次运行之前先清理一下docker网络环境并停止已经运行的e2e案例

|  |
| --- |
| ./network\_setup.sh down  docker rm -f $(docker ps -qa)  docker network prune |

运行案例

|  |
| --- |
| ./network\_setup.sh up |

自动配置好相应的网络架构后，会看到如下输出：

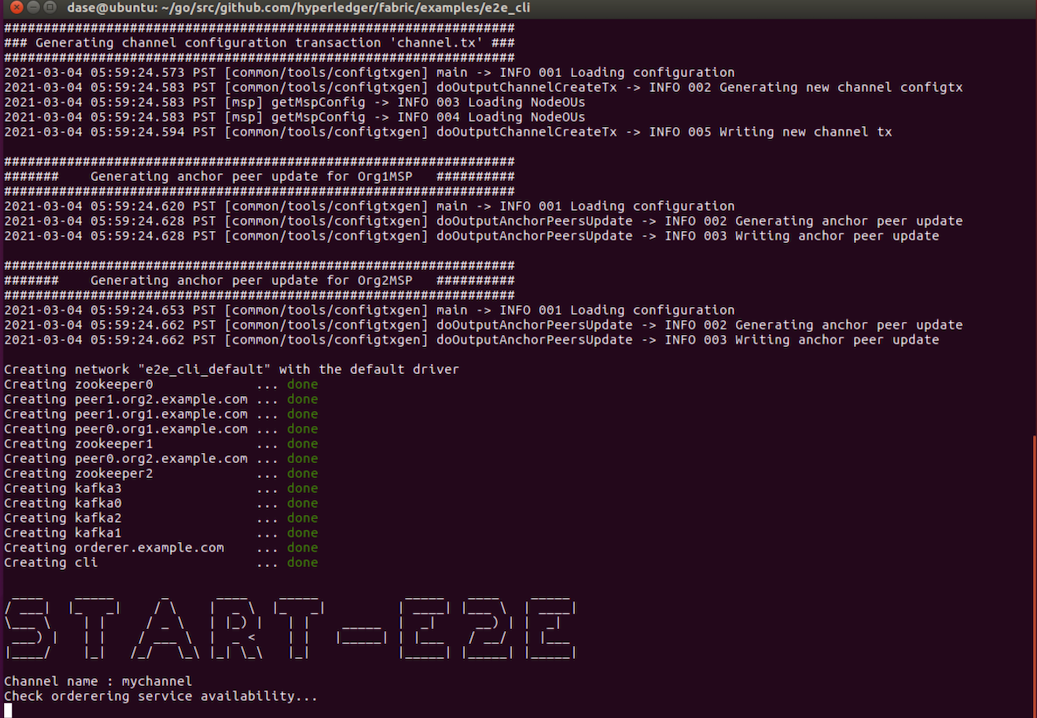
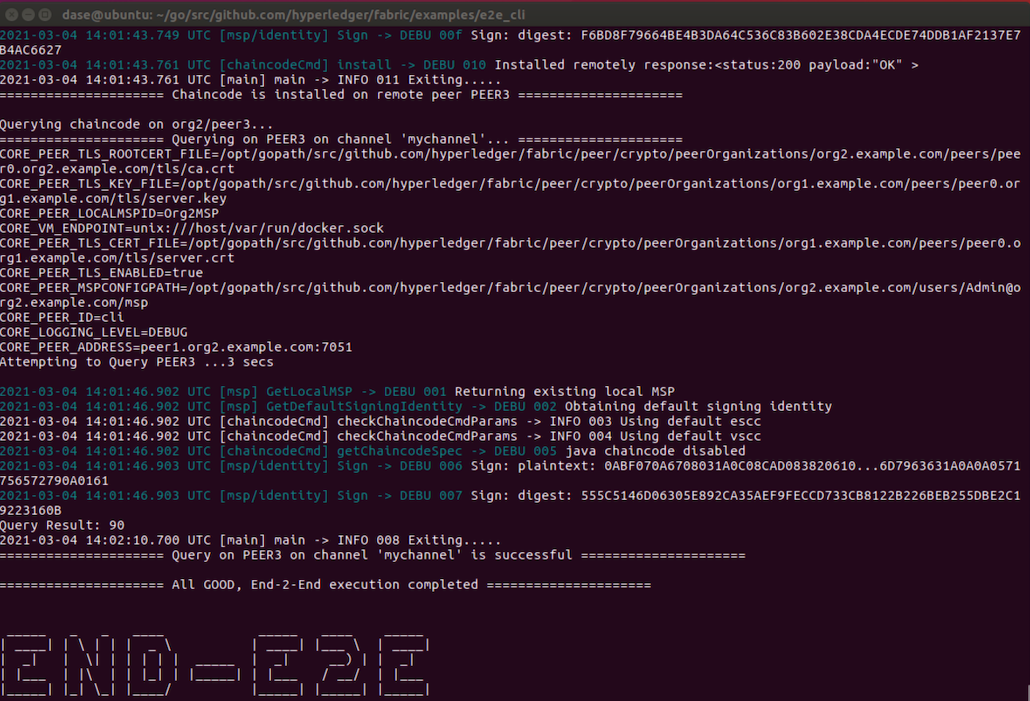


图8-8 完成网络配置后的输出

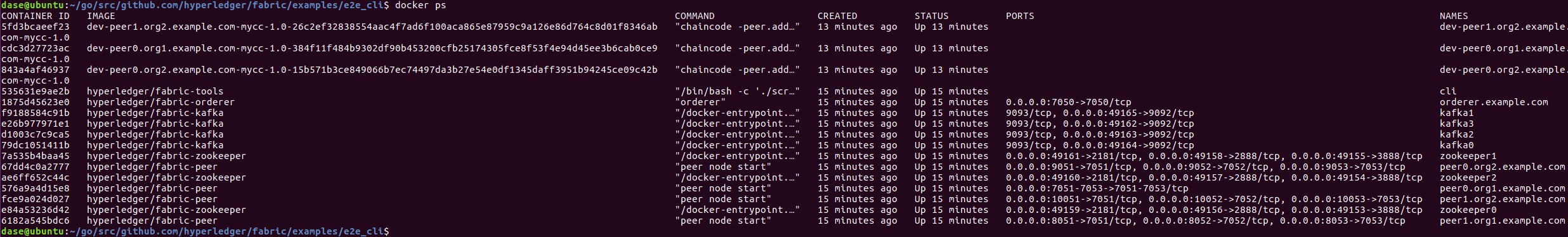
然后经过一段时间的信息打印，其实这个信息打印的过程，就是在配置好的fabric网络中进行了智能合约（chain code）的部署和调用，执行结束后，会看到如下信息：

图8-9 案例执行结束后的输出

看到上述内容，恭喜，说明你成功运行了官方给的案例，你的fabric环境安装与配置是完全成功的。但是你可能根本不知道这一大串的输出到底是怎么回事，这里进行一下讲解，借此了解联盟链的拓扑关系，以及从网络初始化到合约部署以及调用的大致流程。

该官方案例是把网络建立，合约初始化，查询操作和交易操作，一次性全部执行一遍，所以略显复杂。我们先查看一下当前的容器：

|  |
| --- |
| docker ps |

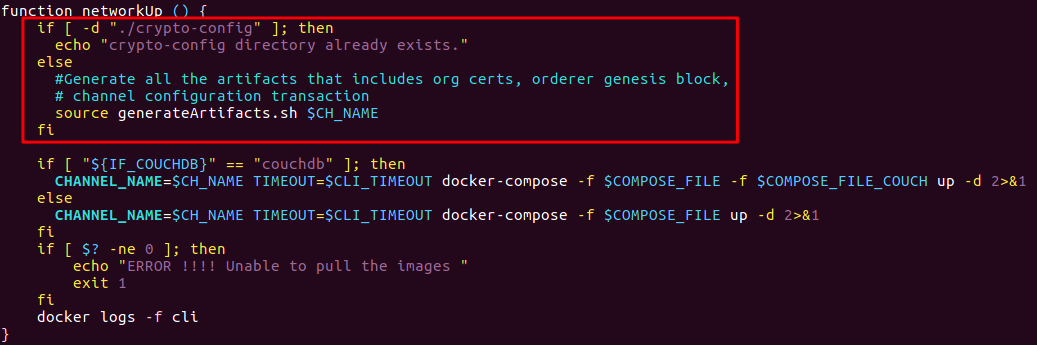
图8-10 当前的容器

如图所示，其实网络结构很简单，四个peer节点是用来提出交易（或者用来背书，或者用来commit），order节点用来接收交易并校验，kafka和zookeeper集群是用来做具体的排序共识，cli用来给开发者提供命令行操作入口。其中四个peer节点分别从属两个不同组织（org1，org2），这是一个比较简单的联盟链拓扑关系。

整个例子的执行分成以下三部分：

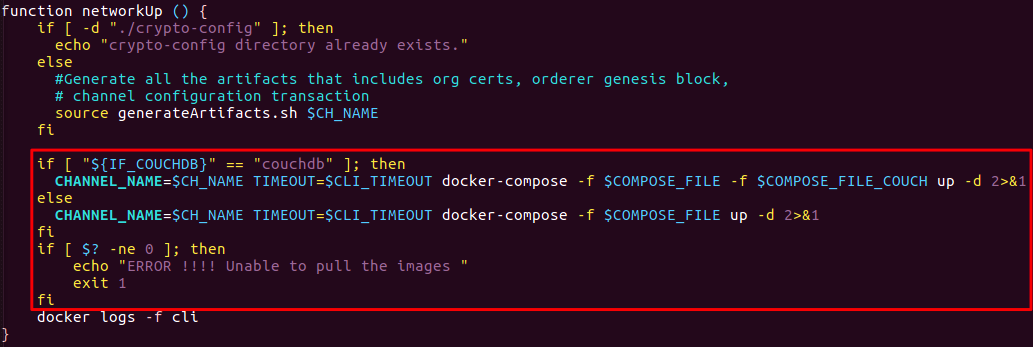
1. **fabric系统的初始化**

generateArtifacts.sh根据crypto-config.yaml生成不同组织下的节点（CA，peer）用户（admin）的公私钥、数字证书；根据configtx.yaml生成创世块、channel初始化配置、锚节点配置（可省略）

图8-11 案例执行（第一块）对应代码

1. **使用docker-compose文件创建网络**

根据docker-compose文件生成网络。例子默认是使用docker-compose-cli.yaml。

图8-12 案例执行（第二块）对应代码

1. **channel的创建、配置及fabric的运行**

docker-compose-cli.yaml中调用了scripts/script.sh，正如下面代码注释，创建channel、按配置文件把peer加入channel、安装和初始化chaincode、Query&Invode操作

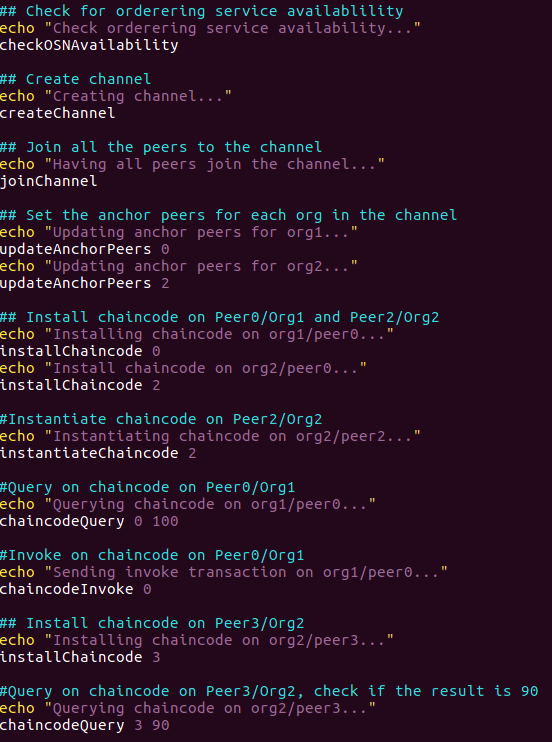


图8-13 案例执行（第三块）对应代码注释

**【实验小结】**

本次实验从依赖环境和系统架构两个方面对Fabric展开了介绍，并详细描述了部署Fabric及其环境的每一个步骤。之后通过运行End2End案例来测试Fabric部署是否成功；这也可使读者进一步了解联盟链的拓扑结构，及系统从初始化、合约部署乃至合约调用的整体流程。

Fabric部署灵活、可插拔及扩展便捷的特性使得其成为联盟链使用场景下最成功的案例之一。经过本次实验读者了解了Fabric的基础知识及系统架构，这也为后续的三个实验打下了基础。

**【习题】**

1. Fabric中的kafka和zookeeper分别实现了什么功能？

2. Fabric与以太坊同为分布式的智能合约平台，它们最显著的区别是什么？

**【参考文献】**

1. Androulaki E, Barger A, Bortnikov V, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains[C]//Proceedings of the thirteenth EuroSys conference. 2018: 1-15.
2. Cachin C. Architecture of the hyperledger blockchain fabric[C]//Workshop on distributed cryptocurrencies and consensus ledgers. 2016, 310(4).
3. 冯翔. 区块链开发实战: Hyperledger Fabric关键技术与案例分析[M]. 机械工业出版社, 2018.