Fabric 环境搭建

## Fabric 介绍

我们知道智能合约比较成功的就是[以太坊](https://learnblockchain.cn/2017/11/20/whatiseth/)了。以太坊主要是公有链，其实对企业应用来说并不是特别合适，而且本身并没有权限控制功能，面向企业的，主要还是 HyperLedger Fabric，当然还有 R3 的 Corda。这里我们主要是讲 Fabric。  
Fabric 是一个面向企业应用的区块链框架，基于 Fabric 的开发可以粗略分为几个层面：

**1.** 参与 Fabric 的底层开发，这主要是 fabric，fabric-ca 和 sdk 等核心组件。  
**2.** 参与 Fabric 周边生态的开发，如支持如支持 fabric 的工具 explorer, composer 等。  
**3.** 利用 fabric 平台开发应用，这就是利用 fabirc 提供的各种 sdk 来为应用服务（应用开发）

大部分企业会参与 2-3 的内容，以 3 为主来服务应用场景，以 2 为辅。因为现在除了区块链核心功能尚未完善外，对区块链的管理，运维，监控，测试，优化，调试等工具非常匮乏。企业将不得不面对自己开发一些工作。

### Fabric 环境依赖

Fabric 官方推荐的开发环境是基于 docker 搭建的，使用 docker 搭建需要一下前置条件：

* docker 一一 Docker version 17.06.2-ce 或以上版本
* Docker Compose 一一 1.14 或以上版本
* Go 一一 1.10 或以上版本， Node.js 一一 8.9.x 或以上版本
* Python 一一主要是 python-pip

### Fabric 环境搭建具体步骤

这里使用的是Ubuntu 16.04.6版本，内存最好在1G以上。

#### ****1. 安装 go 及环境变量配置****

(1) 下载最新版本的 go 二进制文件

|  |  |
| --- | --- |
| 1 | $ wget https://dl.google.com/go/go1.9.2.linux-amd64.tar.gz |

(2) 解压文件

|  |  |
| --- | --- |
| 1 | $ sudo tar -C /usr/local -xzf go1.9.2.linux-amd64.tar.gz |

(3) 配置环境变量

|  |  |
| --- | --- |
| 1 | vim ~/.profile |

添加以下内容：

|  |  |
| --- | --- |
| 1 2 3 4 | export PATH=$PATH:/usr/local/go/bin export GOROOT=/usr/local/go export GOPATH=$HOME/go export PATH=$PATH:$HOME/go/bin |

编辑保存并退出 vi 后，记得使这些环境变量生效

|  |  |
| --- | --- |
| 1 | source ~/.profile |

#### ****2. 安装 Docker****

Fabric 的 chaincode 是运行在 docker 里的。

**(1)** 由于 apt 官方库里的 docker 版本可能比较旧，所以先卸载可能存在的旧版本：

|  |  |
| --- | --- |
| 1 | sudo apt-get remove docker docker-engine docker-ce docker.io |

**(2)** 更新 apt 包索引：

|  |  |
| --- | --- |
| 1 | sudo apt-get update |

**(3)** 安装以下包以使 apt 可以通过 HTTPS 使用存储库（repository）：

|  |  |
| --- | --- |
| 1 | sudo apt-get install -y apt-transport-https ca-certificates curl software-properties-common |

**(4)** 添加 Docker 官方的 GPG 密钥：

|  |  |
| --- | --- |
| 1 2 3 4 | curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add - 备注：可验证秘钥指纹 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88 使用如下命令验证： sudo apt-key fingerprint 0EBFCD88 |

**(5)** 使用下面的命令来设置 stable 存储库：

|  |  |
| --- | --- |
| 1 | sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" |

**(6)** 再更新一下 apt 包索引：

|  |  |
| --- | --- |
| 1 | sudo apt-get update |

**(7)** 安装最新版本的 Docker CE：

|  |  |
| --- | --- |
| 1 2 3 4 5 | sudo apt-get install -y docker-ce 注意：在生产系统上，可能会需要应该安装一个特定版本的Docker CE，而不是总是使用最新版本： 列出可用的版本：apt-cache madison docker-ce 选择要安装的特定版本，第二列是版本字符串，第三列是存储库名称，它指示包来自哪个存储库，以及扩展它的稳定性级别。要安装一个特定的版本，将版本字符串附加到包名中，并通过等号(=)分隔它们： sudo apt-get install docker-ce=<VERSION> |

**(8)** 测试是否安装成功

|  |  |
| --- | --- |
| 1 | docker --version |

**(9)** 使用阿里提供的镜像，否则后面下载 Fabric 镜像会非常慢  
cd 到 /etc/docker 目录下，创建文件 daemon.json，输入下面的内容：

|  |  |
| --- | --- |
| 1 2 3 | {  "registry-mirrors": ["https://obou6wyb.mirror.aliyuncs.com"] } |

保存并退出，接着执行：

|  |  |
| --- | --- |
| 1 2 | sudo systemctl daemon-reload sudo systemctl restart docker |

**(10)** 查看 docker 服务是否启动：

|  |  |
| --- | --- |
| 1 | systemctl status docker |

**(11)** 若未启动，则启动 docker 服务：

|  |  |
| --- | --- |
| 1 | sudo service docker start或者sudo systemctl start docker |

#### ****3. 安装最新版本的 Docker-compose****

**(1)** 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：

|  |  |
| --- | --- |
| 1 | sudo apt-get install python-pip |

**(2)** 安装 Docker-compose：

|  |  |
| --- | --- |
| 1 | pip install docker-compose |

**(3)** 验证是否成功：

|  |  |
| --- | --- |
| 1 | sudo docker-compose --version |

安装 Docker 还可以参考[此篇文章](https://blog.csdn.net/so5418418/article/details/78355868)

#### ****4.Fabric 源码下载****

**(1)** 新建存放测试、部署代码的目录。

|  |  |
| --- | --- |
| 1 | mkdir -p ~/go/src/github.com/hyperledger/ |

**(2)** cd 到刚创建的目录

|  |  |
| --- | --- |
| 1 | cd ~/go/src/github.com/hyperledger |

**(3)** 下载 Fabric, 这里使用使用 git 命令下载源码：

|  |  |
| --- | --- |
| 1 | git clone https://github.com/hyperledger/fabric.git |

**(4)** 由于 Fabric 一直在更新，新版本的并不稳定，所有我们并不需要最新的源码，需要切换到 v1.0.0 版本的源码：

|  |  |
| --- | --- |
| 1 | cd ~/go/src/github.com/hyperledger/fabric  git checkout v1.0.0 |

#### ****5. 下载 Fabric Docker 镜像****

**(1)** 前面步骤 4 下载完成后，我们可以看到当前工作目录 (~/go/src/github.com/hyperledger/) 下多了一个 fabric 的文件夹，  
接下来我们 cd 到~/go/src/github.com/hyperledger/fabric/examples/e2e\_cli 目录下执行：

|  |  |
| --- | --- |
| 1 | source download-dockerimages.sh -c x86\_64-1.0.0 -f x86\_64-1.0.0 |

(注：一定要下载完所有镜像并且镜像版本要和 Fabric 版本一致如何没有下载问继续执行 source download-dockerimages.sh 命令直到在完如图所有镜像)，执行完所有会用到的 Fabric docker 镜像都会下载下来了。  
运行以下命令检查下载的镜像列表：

|  |  |
| --- | --- |
| 1 | docker images |

**注意**：如果下载时遇到权限问题，需要切换到 root 用户下：su root  
**(2)** 重启 Docker

|  |  |
| --- | --- |
| 1 | service docker restart |

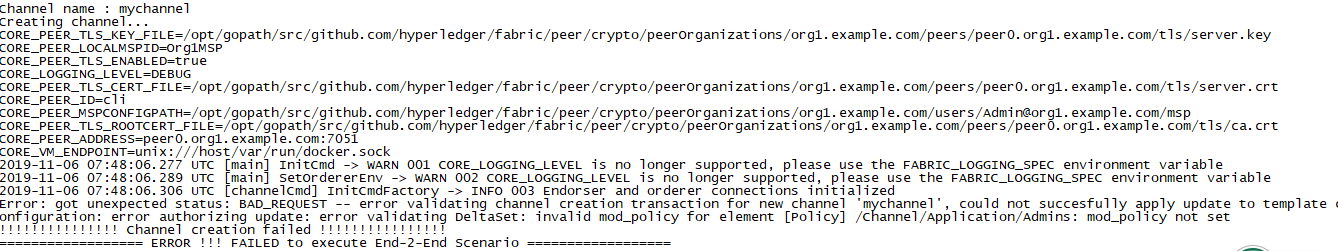
#### ****6. 测试 Fabric 环境是否成功****

有两个坑需要提前填一下：

1. Fabric默认安装路径在/opt/gopath中，但是fabric却不会自动创建这个路径，需要你手动创建一下，否则程序会包莫名其妙的错误哦。（如果这一步没有做，而程序报错，都可以尝试做一下这一步。）

mkdir –p /opt/gopath

否则可能报下面的错误：



1. 如果报错Error: Error endorsing chaincode: rpc error: code = Unknown desc = Error starting container: API error (404): {"message":"network e2ecli\_default not found"}  
     
   **原因是：**

e2e\_cli目录是固定的，启动后会创建一个docker network以此为名字，这里是e2e\_cli。如果修改该目录，要修改/opt/gopath/src/github.com/hyperledger/fabric/examples/**e2e\_cli**/base目录下的peer-base.yaml  
将网络名改成如下名称即可  
- CORE\_VM\_DOCKER\_HOSTCONFIG\_NETWORKMODE=**e2e\_cli**\_default

在~/go/src/github.com/hyperledger/fabric/examples/e2e\_cli 下执行如下命令启动测试

|  |  |
| --- | --- |
| 1 | ./network\_setup.sh up |

这个指令具体进行了如下操作：

1. 编译生成 Fabric 公私钥、证书的程序，程序在目录：fabric/release/linux-amd64/bin
2. 基于 configtx.yaml 生成创世区块和通道相关信息，并保存在 channel-artifacts 文件夹。基于 configtx.yaml 生成创世区块和通道相关信息，并保存在 channel-artifacts 文件夹。
3. 基于 crypto-config.yaml 生成公私钥和证书信息，并保存在 crypto-config 文件夹中。基于 crypto-config.yaml 生成公私钥和证书信息，并保存在 crypto-config 文件夹中。
4. 基于 docker-compose-cli.yaml 启动 1Orderer+4Peer+1CLI 的 Fabric 容器。基于 docker-compose-cli.yaml 启动 1Orderer+4Peer+1CLI 的 Fabric 容器。  
   在 CLI 启动的时候，会运行 scripts/script.sh 文件，这个脚本文件包含了创建 Channel，加入 Channel，安装 Example02，运行 Example02 等功能。

运行完如果出现下图所示，说明整个 Fabric 网络已经通了。  


### 测试 Fabric 网络

接下来我们手动测试下 Fabric 网络，Fabric 提供了 SDK 和 CLI 两种交互方式，这里我们使用的是 CLI。  
这里我们使用官方提供的小例子进行测试，在官方例子中，channel 名字是 mychannel，链码（智能合约）的名字是 mycc。  
首先要登录到 CLI 这个容器中，才能执行 Fabric 的 CLI 命令：

|  |  |
| --- | --- |
| 1 | docker exec -it cli bash |

这时用户名变为 root@caa22f87a5bf，当前目录变为 /opt/go/src/github.com/hyperledger/fabric/peer#，接着可执行 peer 命令，体验区块链的命令行使用方式。

**1. 查看 a 账户的余额**

|  |  |
| --- | --- |
| 1 | peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}' |

此时我们可以看到控制台输出有：

|  |  |
| --- | --- |
| 1 | Query Result: 90 |

这里 90 就是 a 账户的余额

**2. 调用链码，转账**

这里我们让 b 账户向 a 账户转账 10：

|  |  |
| --- | --- |
| 1 | peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/go/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C mychannel -n mycc -c '{"Args":["invoke","b","a","10"]}' |

转账成功后，我们可以看到有输出如下提示：

|  |  |
| --- | --- |
| 1 | DEBU 009 ESCC invoke result: version:1 response:<status:200 message:"OK" |

接下来我们使用前面的命令继续查看 a 账户的余额，输出结果如下：

|  |  |
| --- | --- |
| 1 | Query Result: 100 |

很明显我们已经转账成功了。

**退出 cli 容器：**

直接执行

|  |  |
| --- | --- |
| 1 | exit |

最后如果我们要关闭 Fabric 网络，cd 到 ~/go/src/github.com/hyperledger/fabric/examples/e2e\_cli 下（注意这里的路径按自己前面创建的，不一定要和我一样），执行：

|  |  |
| --- | --- |
| 1 | ./network\_setup.sh down |

参考文献：

<https://learnblockchain.cn/2018/11/21/fabric_introduction/>

<https://www.cnblogs.com/zhangmingcheng/p/9052003.html>

# [Hyperledger Fabric开发实战-04编写智能合约](https://www.jianshu.com/p/6b7049b4bdba)

Hyperledger Fabric的智能合约叫做Chaincode，是业务的成载体，负责具体的业务逻辑

## Chaincode代码

Fabric的Chaincode运行在容器中，可以使用Go，Java，Node.js语言开发，Golang是目前为主比较成熟稳定的。

### Chaincode组成

Chaincode主要由下面几个组成:

* 必须在main包下
* 引用必要的依赖
* 定义一个结构体，并为结构体绑定Init和Invoke方法
* shim.ChaincodeStubInterface和pb.Response
* main方法

下面以一个例子进行说明

// 1.在main包下

package main

// 2.引入必要的依赖

import(

"fmt"

"github.com/hyperledger/fabric/core/chaincode/shim"

pb "github.com/hyperledger/fabric/protos/peer"

)

// 3.定义一个结构体

type mychaincode struct {

}

// 4.为结构体绑定init和invoke方法

func (t \*mychaincode) Init(stub shim.ChaincodeStubInterface) pb.Response{

fmt.Println(" << ====[Init] success init it is view in docker ======")

return shim.Success([]byte("success init"))

}

func (t \*mychaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response{

fmt.Println(" << ====[Invoke] success init it is view in docker ======")

return shim.Success([]byte("success init"))

}

// 5.主方法

func main(){

err := shim.Start(new(mychaincode))

if err != nil{

fmt.Println("Error starting Simple chaincode : %s",err)

}

}

### Chaincode部署

Chaincode的运行分为：install，instantiate和invoke三步

编写以上的代码后，我们将其放入一个文件夹，例如: /home/ssj234/fabricwksp/05-chaincode/下，在部署的时候，会去$GOPATH的src下面寻找文件夹，因此，我们在如下路径编写Chaincode

cd /home/ssj234/fabricwksp/05-chaincode/

mkdir -p src/firstchaincode

cd src/firstchaincode

# 编写代码

vim firstchaincode.go

go build

**安装**

# 设置GOPATH，让其能够找到Chaincode代码

# 设置要链接的Peer节点和Peer的Admin证书

export set GOPATH=/home/ssj234/fabricwksp/05-chaincode

export set FABRIC\_CFG\_PATH=/home/ssj234/fabricwksp/02-helloworld/peer

export set CORE\_PEER\_LOCALMSPID=Org1MSP

export set CORE\_PEER\_ADDRESS=peer0.org1.cmbc.com:7051

export set CORE\_PEER\_MSPCONFIGPATH=/home/ssj234/fabricwksp/02-helloworld/fabricconfig/crypto-config/peerOrganizations/org1.cmbc.com/users/Admin@org1.cmbc.com/msp

# 部署chaincode 代码，-n后面是名字 -v 是Chaincode的版本 -p后面是在$GOPAT/src下的相对路径

peer chaincode install -n hellochaincode -v 1.1 -p hellochaincode

**实例化**  
实例化的时候会调用init方法

# 设置GOPATH

# 设置要连接的Peer节点和其Admin证书

export set GOPATH=/home/ssj234/fabricwksp/05-chaincode

export set FABRIC\_CFG\_PATH=/home/ssj234/fabricwksp/02-helloworld/peer

export set CORE\_PEER\_LOCALMSPID=Org1MSP

export set CORE\_PEER\_ADDRESS=peer0.org1.cmbc.com:7051

export set CORE\_PEER\_MSPCONFIGPATH=/home/ssj234/fabricwksp/02-helloworld/fabricconfig/crypto-config/peerOrganizations/org1.cmbc.com/users/Admin@org1.cmbc.com/msp

# 初始化chaincode 代码，需要指定channel的名字，-c是传入的参数

peer chaincode instantiate -o orderer.cmbc.com:7050 -C cmbcchannel666 -n firstchaincode -v 1.1 -c '{"Args":["init","a","100","b","200"]}' -P "OR ('Org1MSP.member','Org2MSP.member')"

instantiate实例化命令运行的时间较长，银行会启动一个docker容器来执行chaincode，我们可以通过docker ps命令观察,会看到下面的一个容器正在运行

STATUS PORTS NAMES

bee440516894 dev-peer0.org1.cmbc.com-firstchaincode-1.2-876a5b6466d4eeac14cf940b95e3538a8ace73b133fad7c70041ea17d7cf4b8e "chaincode -peer.a..." 2 minutes ago Up 2 minutes

使用docker logs bee440516894查看日志，会输出Init方法中打印的内容

<< ====[Init] success init it is view in docker ======

**调用**

使用peer chaincode invoke可以调用chaincode的invoke方法

# 设置$GOPATH

# 设置要连接的Peer和Peer的Admin证书

export set GOPATH=/home/ssj234/fabricwksp/05-chaincode

export set FABRIC\_CFG\_PATH=/home/ssj234/fabricwksp/02-helloworld/peer

export set CORE\_PEER\_LOCALMSPID=Org1MSP

export set CORE\_PEER\_ADDRESS=peer0.org1.cmbc.com:7051

export set CORE\_PEER\_MSPCONFIGPATH=/home/ssj234/fabricwksp/02-helloworld/fabricconfig/crypto-config/peerOrganizations/org1.cmbc.com/users/Admin@org1.cmbc.com/msp

# 调用chaincode 代码，需要制定orderer服务器，channel的名称和Chaincode的名称和版本号

peer chaincode invoke -o orderer.cmbc.com:7050 -C cmbcchannel666 -n firstchaincode -v 1.1 -c '{"Args":["invoke","a","100","b","200"]}'

执行完成，通过docker logs命令可以查看到Invoke方法的打印内容。

参考文献：https://www.jianshu.com/p/2e389823f556