**MATRX 超级账本Fabric部署方案  
说明书**

**版本: V1.0**

**作者：MATRIX 基金会北京研究院**

**日期：201X.XX.XX**

任何情况下，与本系统及其衍生产品，以及与之相关的全部文件（包括本文件及其任何附件中的全部信息）相关的全部知识产权（包括但不限于版权、商标和技术秘密）皆属于 MATRIX 基金会所有。

本文件中的信息是保密的，且仅供用户指定的接收人内部使用。未经 MATRIX 基金会事先书面同意，本文件的任何用户不得对本产品和本文件中的信息向任何第三方（包括但不限于除用户指定接收人以外的管理人员、员工和关联公司）进行披露、出借、许可、转让、出售、分发、传播或进行与本产品和本文件相关的任何其他处置，也不得使该等第三方以任何形式使用本产品和本文件中的信息。

未经 MATRIX 基金会事先书面允许，不得为任何目的、以任何形式或任何方式对本文件进行复制、修改或分发。本文件的任何用户不得更改、移除或损害本文件所使用的任何商标。

本文件按“原样”提供，就本文件的正确性、准确性、可靠性或其他方面，MATRIX 基金会并不保证本文件的使用或使用后果。本文件中的全部信息皆可能在没有任何通知的情形下被进一步修改，MATRIX 基金会对本文件中可能出现的任何错误或不准确之处不承担任何责任。

在任何情况下，MATRIX 基金会均不对任何因使用本软件产品和本文件中的信息而引起的任何直接损失、间接损失、附带损失、特别损失或惩罚性损害赔偿（包括但不限于获得替代商品或服务、丧失使用权、数据或利润；或商业中断），责任或侵权（包括过失或其他侵权）承担任何责任，即使 MATRIX 基金会事先获知上述损失可能发生。

MATRIX 基金会保留所有权利

# 目 录

[目 录 1](#_Toc26283)

[变更记录 3](#_Toc28941)

[1 引言 4](#_Toc29944)

[1.1 文档用途 4](#_Toc2355)

[1.2 术语 4](#_Toc4344)

[1.3 使用对象 5](#_Toc32508)

[2 模块描述 6](#_Toc27883)

[2.1 Fabric部署模块功能清单 6](#_Toc25433)

[2.2 Fabric环境准备及安装 7](#_Toc18127)

[2.2.1 环境准备 7](#_Toc29152)

[2.2.2 安装Ubuntu系统及相关工具 7](#_Toc13122)

[2.2.3 go环境安装 9](#_Toc7823)

[2.2.4 docker安装 11](#_Toc10428)

[2.2.5 docker-compose安装 12](#_Toc22185)

[2.2.6 Fabric源代码安装 13](#_Toc5498)

[2.2.7 Fabric docker镜像 14](#_Toc13678)

[2.3 Fabric环境部署 17](#_Toc24007)

[2.3.1 Fabric集群关键参数简介 17](#_Toc13258)

[2.3.2 配置文件crypto-config.yaml生成MSP证书 18](#_Toc19850)

[2.3.3 配置文件configtx.yaml生成创世区块 20](#_Toc21026)

[2.3.4 创建自己的docker-compose.yaml文件 22](#_Toc7999)

[2.3.5 初始化Fabric环境 27](#_Toc30188)

[2.3.6 链码安装与运行 32](#_Toc3015)

# 变更记录

| 修订日期 | 设计内容 | 作者 | 评审者 | 评审日期 |
| --- | --- | --- | --- | --- |
| 2018.7.5 | 初始版 | 黄志 |  |  |
|  |  |  |  |  |

# 引言

## 文档用途

编写本文挡的目的是为了描述超级账本Fabric部署的整个过程、可操控范围内的详细细节以及部署过程中的重点注意事项，为实施工程师、开发工程师及其他相关人员提供开发和测试依据

## 术语

| **术语** | **备注** |
| --- | --- |
| chaincode | 链码运行在区块链上的智能合约 |
| channel | 通道 |
| orderer | 排序节点 |
| peer | 记账节点（含主节点和背书节点） |
| [Hyperledger Fbric](https://blog.csdn.net/qq_27818541/article/details/77847483" \l "configuration-block-配置块" \t "_self) CA | 证书颁发机构组件(PKI体系) |
| Ledger | 账本 |
| MSP | 会员服务提供商 |
| Proposal | 交易提案（peer背书、链码执行） |
| Transaction | 交易 |
| Instantiate | 实例（链码实例化） |
| Genesis Block | 创世区块 |
| Fabric(fabric) | Hyperledger Fbric 超级账本的简称 |

## 使用对象

MATRIX 研究院

# 模块描述

Fabric开发部署方案分几大块：

* Fabric环境准备
* Fabric环境部署
* Fabric链码开发
* Fabric SDK 应用系统开发

## Fabric部署模块功能清单

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **模块ID** | **模块名称** | **关联描述** | **更新日期** |
| 1 | 001 | Fabric环境准备 | 环境 | 2018-07-06 |
| 2 | 002 | Fabric环境部署 | 环境 | 2018-07-09 |
| 3 | 003 | Fabric链码开发 | 开发 | 2018-07-10 |
| 4 | 004 | Fabric SDK 应用系统开发 | 开发 | 2018-07-11 |

## Fabric环境准备及安装

安装Hyperledger Fabric1.0版本产品，提供可以运行智能合约的生产环境部署方案。依次按照下列步骤安装。

### 环境准备

Hyperledger Fabric 1.0默认的共识是solo，即单节点共识机制（不适合生产需要），生产上需要集群化配置，kafka集群、zookeeper集群、orderer生产者消费者集群、peer节点集群是必须要选择的集群结构。

* zookeeper节点：3个
* kafka节点：4个
* orderer节点：3个
* peer节点：4个
* cli节点：随意数

因同一台虚拟机（或实机）可同时兼顾部署其他节点的功能相互叠（通过端口区分即可），实际的节点数需求区间为[4,14]。

### 安装Ubuntu系统及相关工具

环境准备：

虚拟机：VMware或VirtualBox

Linux OS 镜像：ubuntu-16.04.4-desktop-amd64.iso

相关工具：apt-get、git、curl、wget、ssh

步骤说明：

1）下载较新的VirtualBox或者VM ware版本。

2）下载Ubuntu 16.04 镜像并在虚拟机上安装linux操作系统（也可使用纯命令行版本Linux OS

ubuntu-16.04.4-server-amd64.iso）。

3）安装工具 apt-get、git、curl、wget、ssh、npm、nodejs

4）修改/etc/apt/sources.list（使用国内的源，替换国外的源），将[http://us.\*](http://us.*)的文件替换成[http://cn.\*](http://cn.*)的文件（vi命令:%s/us./cn./g，:wq退出保存即可）。

5）apt-get更新：

sudo apt-get update

6）安装ssh，安装后即可使用SecureCRT等客户端远程访问ubuntu系统。ssh安装：

sudo apt-get install ssh

启动ssh服务：

sudo /etc/init.d/ssh start

7）其他工具安装：

sudo apt-get install git

sudo apt-get install curl

sudo apt-get install wget

sudo apt-get install npm

sudo apt-get install nodejs

1. 每个待加入MSP组织的节点都需要按照上述过程安装。

### go环境安装

指定下载go版本，推荐使用go1.10版本 linux/amd64。

下载方法（建议采用SecureCRT等客户端远程访问工作）：

wget <https://storage.googleapis.com/golang/go1.10.linux-amd64.tar.gz>

解压go版本（放到$HOME下）：

cd ~

sudo tar -C /usr/local -xzf go1.10.linux-amd64.tar.gz

mkdir go #~/go作为GOPATH工作主目录，需创建

修改环境变量配置文件：

vi .bash\_profile

在.bash\_profile中增加go语言工作环境：

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

export GOROOT=/usr/local/go

export GOPATH=$HOME/go

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

set –o vi

修改生效：

保存退出.bash\_profile或. .bash\_profile，建议使用保存退出的方法，不建议采用. .bash\_profile方法。

安装go tools（可选，第三方库用于开发）：

Fabric使用GO开发，其中用到了一些官方或者第三方的工具，如若需要必须安装此模块。安装方法：

mkdir –p $GOPATH/src/golang\_org

cd $GOPATH/src/golang\_org/

git clone https://github.com/golang/tools.git

简要说明：安装完成后，go运行环境即可使用，包括自主编程go项目工程。go 环境开发安装完成后，可以通过运行go可以检测是否安装成功（也可以通过go version查看版本等基本信息进行检测）。

### docker安装

直接下载apt-get指定安装包（17ce系列版本）：

sudo apt-get install docker.io

或下载阿里云的安装版本：

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

修改用户名组：

sudo usermod -aG docker userid

sudo apt-get install docker-compose

检查docker：

docker version

启动docker(暂时不启动):

service docker start

### docker-compose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，需要安

装python运行环境（docker-compose 只能运行在python 2.X版本，推荐使用python

2.7系列版本）。这里我们选择采用pip安装python-pip。安装python-pip会安装

pip所依赖的python和各种库，python环境也会随带安装。

安装python-pip：

sudo apt-get install python-pip

安装docker-compose:

sudo apt-get install docker-compose

或

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

检查python和docker-compose版本：

docker-compose version #同时也检查了python 版本和compose组件依赖运

行的环境（比如openssl）

### Fabric源代码安装

使用Git命令下载源码，首先需要建立对应的目录，然后进入该目录，Git下载

源码：

创建源码存放父目录：

mkdir -p ~/go/src/github\_com/hyperledger

打开源码存放父目录：

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

使用git clone Fbric源码：

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

打开Fabric目录：

cd ~/go/src/github\_com/hyperledger/fbric

使用git获取指定的fabric版本:

git checkout v1.0.1

### Fabric docker镜像

镜像安装可以使用两种方式，一种是编译，编译Fabric的代码主要是通过make

命令来完成，利用fabric代码中提供的完整的Makefile，编译各类镜像；另一种方式是根据shell脚本download-dockerimages.sh下载镜像。我们这块使用脚本下载镜像的方式，暂不使用make 命令（若网络没有限制的情况下,一句make all即可）。

打开脚本download-dockerimages.sh所在目录：

cd $GOPATH/src/github\_com/hyperledger/fabric/examples/e2e\_cli

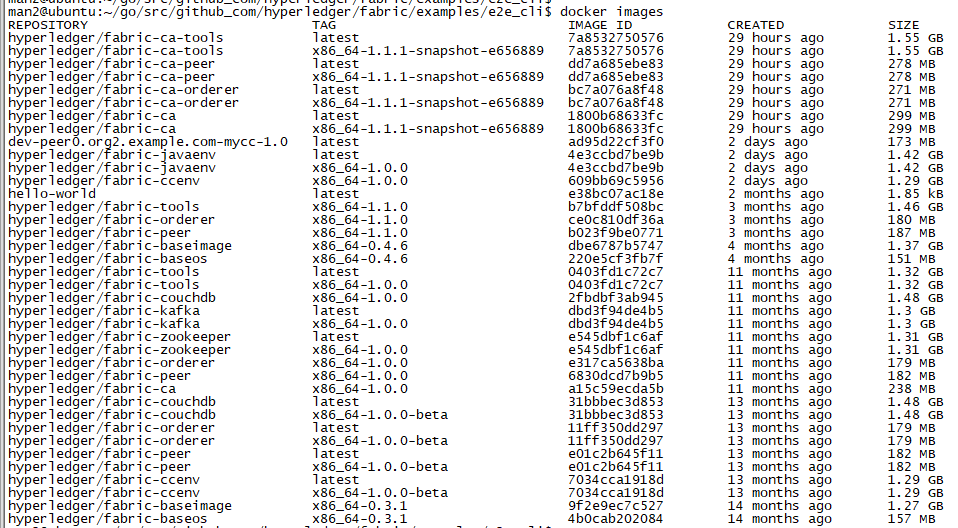
下载fabric docker镜像（64位X86系统的1.0.1匹配兼容版本）：

source download-dockerimages.sh -c x86\_64-1.0.1 -f x86\_64-1.0.1

查看下载镜像：

docker images

内容参考如下（images有其他的方式下载，镜像较多，只做参考，具体镜像文件依实际情况为准）



搭建Fabric网络时，较为重要的工具需要单独编译（configtxgen、cryptogen、configtxlator）。

打开fabric源代码目录：

cd $GOPATH/src/github\_com/hyperledger/fabric

逐个编译：

make configtxgen

make cryptogen

make configtxlator

将 $GOPATH/src/github\_com/hyperledger/fabric /build/bin 路径增加到PATH中：

docker镜像文件列表简介：

|  |  |  |
| --- | --- | --- |
| docker容器名称 | 功能 | 备注 |
| orderer | 排序服务 | Fabric核心服务（打包区块） |
| peer | 背书节点、记账节点、主节点、锚节点等功能 | Fabric核心服务，共识机制的关键部分（背书模拟、验证记账） |
| ca | CA认证体系 | PKI认证体系，MSP范畴 |
| kafka | 排序服务的共识算法机制 | 共识算法，保证交易顺序，带有消息机制的缓冲功能 |
| couchdb | 状态数据库服务 | 数据存储范畴 |
| zookeeper | 分布式应用提供一致性服务 | 排序服务支持kafka |
| ccenv | 链码服务 |  |
| tools | 工具类服务，支持fabric运行 |  |
| javaenv | 链码java版本服务 | 需要 chaintool的jar包 |

## Fabric环境部署

### Fabric集群关键参数简介

在搭建HyperLedger Fabric环境的过程中，会用到配置文件configtx.yaml，该配置文件主要用于构建创世区块（在构建创世区块之前需要先创建与之对应的所有节点的验证文件集合），其中在配置Orderer信息中有一个OrdererType参数(共识机制选项)。

该参数默认值为："solo" #单节点共识机制方案

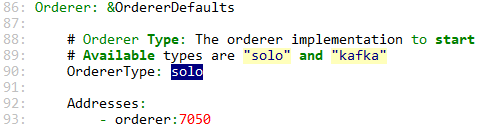
修改configtx.yaml配置文件：

vi $GOPATH/src/github\_com/hyperledger/fabric/examples/e2e\_cli/configtx.yaml

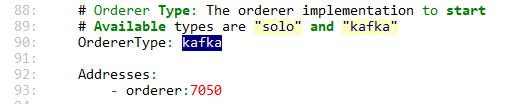
修改内容：

必须修改为:"kafka" #分布式队列共识机制方案

修改前：



修改后：



采用kafka共识机制，增加orderer排序服务的容错空间，当我们向peer节点提交交易transaction的时候，peer节点基于SDK会返回一个读写集结果，该结果会发送给orderer节点进行共识和排序，此时如果orderer节点突然down掉，致使请求服务失效而引发的数据丢失等问题，且目前的sdk对orderer发送交易transaction的回调会占用极长的时间，当大批量数据导入的时候该回调可认为不可用。

因此，在部署生产环境时，需要对orderer进行容错处理，而所谓的容错即搭建一个orderer节点集群，该集群会依赖于kafka和zookeeper。

### 配置文件crypto-config.yaml生成MSP证书

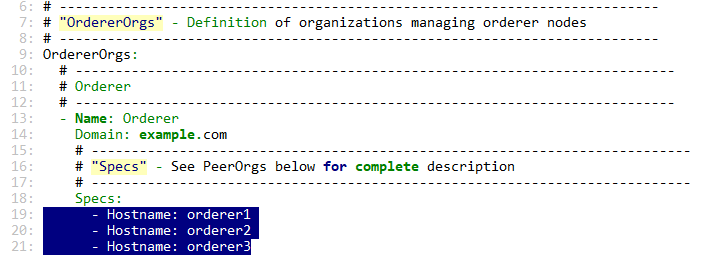
该文件被crytogen工具调用，文件中包括了网络拓扑，同时允许我们给organization以及component（隶属于organization的组件）生成一个证书与私钥的集合。每一个organization被分配一个唯一的根证书（绑定了隶属于organization的具体的component，包括所有的peer与所有的orderer）。Hyperledger Fabric的transaction与通信均被节点的私钥(keystore)进行签名，被公钥进行验证（signcerts）。

在配置文件crypto-comfig中有一个OrdererOrgs配置段，在该配置中我们可以设置当前Fabric生产中允许的最大orderer节点数量及名称，配置如下图所示：

修改crypto-config.yaml文件：

vi $GOPATH/src/github\_com/hyperledger/fabric/examples/e2e\_cli/crypto-config.yaml

修改内容：



简要说明：org1.example.com和org2.example.com两个组织根据需要再改，这块不改。

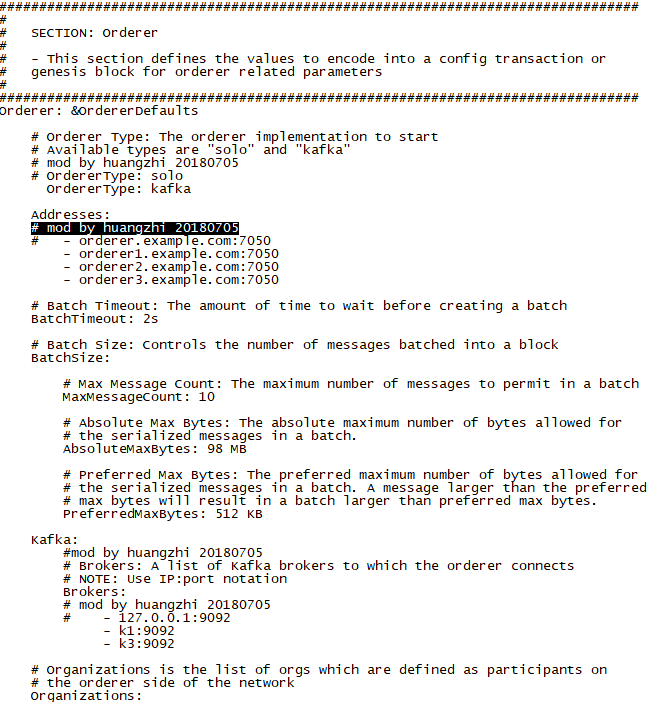
生成MSP证书：

cryptogen generate --config=./crypto-config.yaml

### 配置文件configtx.yaml生成创世区块

修改配置文件：

vi $GOPATH/src/github\_com/hyperledger/fabric/examples/e2e\_cli/configtx.yaml



配置环境变量到.bash\_profile（需要重启终端），配置内容如下：

#FABRIC\_CFG path cfg 配置路径

export FABRIC\_CFG\_PATH=$GOPATH/src/github\_com/hyperledger/fabric/examples/e2e\_cli

# CHANNEL\_NAME 设置通道名

export CHANNEL\_NAME=mychannel

生成排序服务创世区块：

configtxgen -profile TwoOrgsOrdererGenesis -outputBlock ./channel-artifacts/genesis.block

生成通道配置创世区块:

configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID $CHANNEL\_NAME

简要说明：CHANNEL\_NAME可以环境变量配置也可以直接即时脚本配置。

定义组织锚节点：

组织Org1MSP：

configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID $CHANNEL\_NAME -asOrg Org1MSP

组织Org2MSP：

configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID $CHANNEL\_NAME -asOrg Org2MSP

简要说明：模拟网络测试时，可以使用集中式的SOLO算法做排序服务。

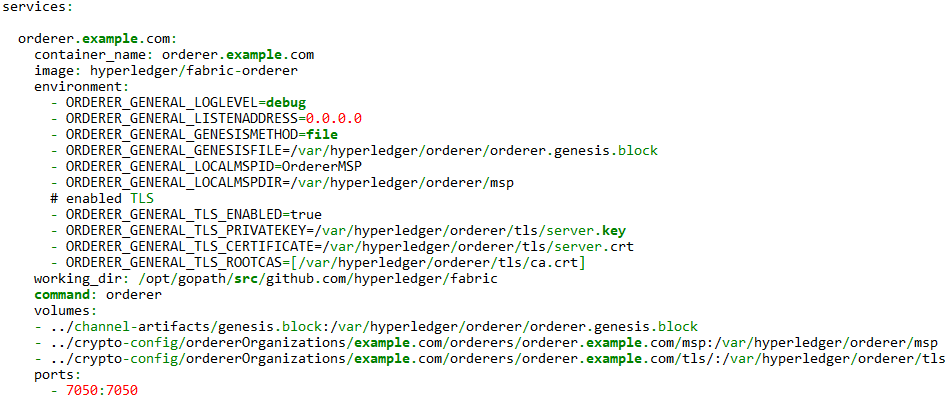
### 创建自己的docker-compose.yaml文件

yaml文件有三大类，分别是：orderer节点服务yaml文件、peer节点服务yaml文件、cli客户端节点yaml文件。

orderer的yaml配置文件：base/docker-compose-base.yaml(相对工作路径，可修改)，内容如下：



和orderer节点相关的配置：



重点说明：

ORDERER\_GENERAL\_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block

ORDERER\_GENERAL\_LOCALMSPDIR=/var/hyperledger/orderer/msp

ORDERER\_GENERAL\_GENESISFILE环境变量：创世区块host和docker之间的映射，对应于volumes的路径映射：

- ../channel-artifacts/genesis.block:/var/hyperledger/orderer/orderer.genesis.block

ORDERER\_GENERAL\_LOCALMSPDIR环境变量：MSP路径host和docker之间的映射，对应于volumes的路径映射：

- ../crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/msp:/var/hyperledger/orderer/msp

peer的yaml配置文件：base/docker-compose-base.yaml和peer-base.yaml，内容如下：

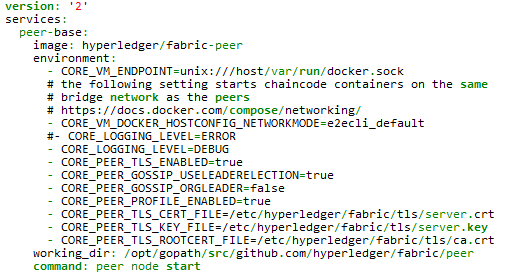
 

和peer节点相关的配置：

docker-compose-base.yaml配置的peer相关内容：



peer-base.yaml配置的相关内容：



重点说明：

- CORE\_PEER\_ADDRESS=peer0.org1.example.com:7051

- CORE\_PEER\_CHAINCODELISTENADDRESS=peer0.org1.example.com:7052

- CORE\_PEER\_GOSSIP\_EXTERNALENDPOINT=peer0.org1.example.com:7051

file: peer-base.yaml

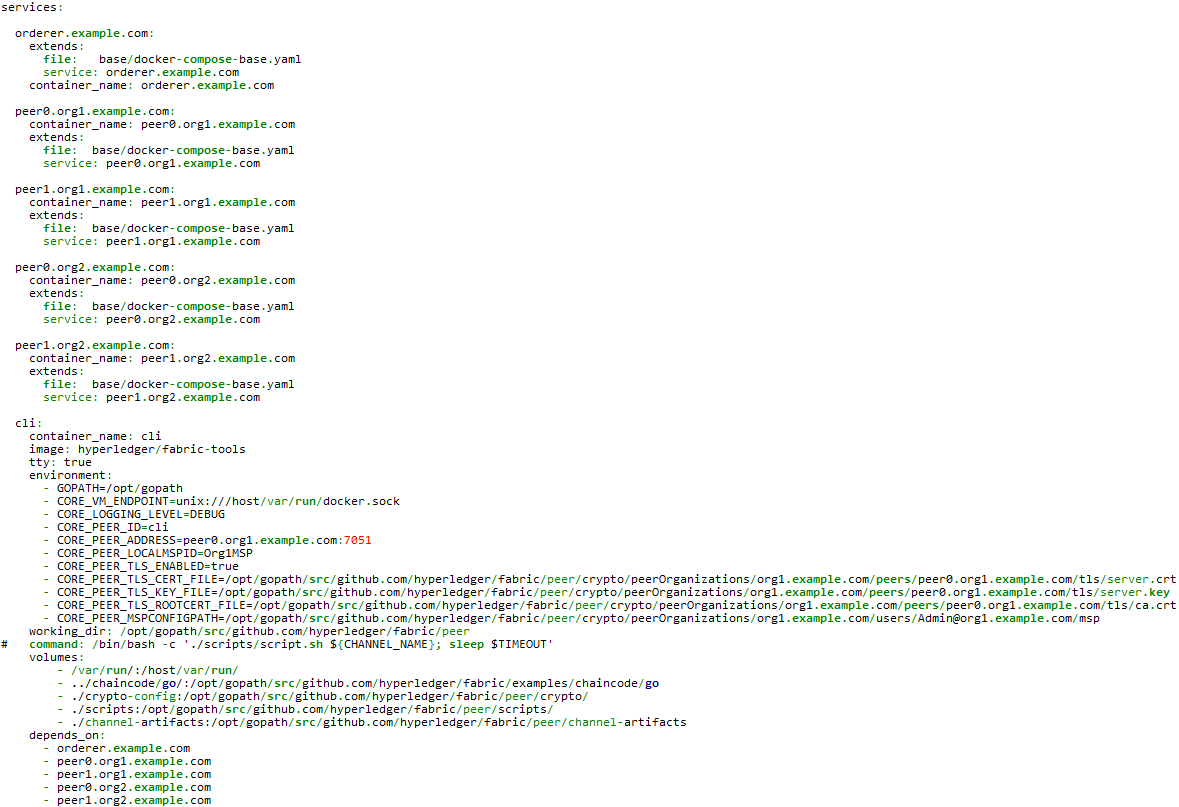
CORE\_PEER\_ADDRESS和CORE\_PEER\_GOSSIP\_EXTERNALENDPOINT外部请求时做为背书、记账、主节点、锚节点监听地址，CORE\_PEER\_CHAINCODELISTENADDRESS负责监听链码内部的调用监听，主要处理链码生存周期的管理调度及事件驱动机制的监听。

file配置的事peer节点基本参数所属的文件名。

cli客户端节点yaml文件：docker-compose-cli.yaml，内容如下：



和cli相关的配置：



重点说明：

- CORE\_PEER\_ADDRESS=peer0.org1.example.com:7051

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

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

cli启动之时默认连接是peer0.org1.example.com，默认[Admin@org1.example.com](mailto:Admin@org1.example.com)账户身份连接到peer的。

安装的链码chaincode都是在fabric/examples/chaincode/go目录下，当我们要开发自己的链码chaincode，只要放到该目录下即可，直接编译安装(go build chaincodede

go 代码文件）。

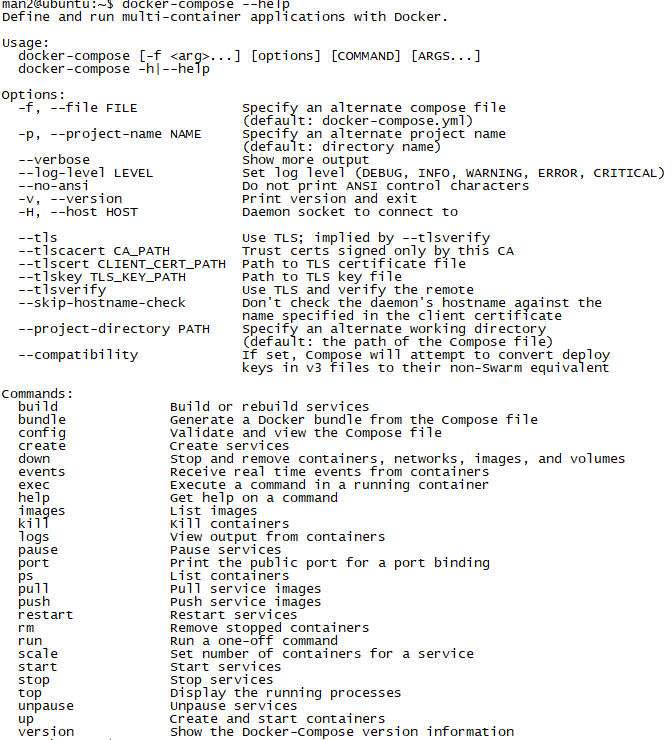
### 初始化Fabric环境

#### 启动Fabric环境容器

使用docker组件启动（运行的Fabric沙箱环境，相对安全可靠）区块链网络：

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

附docker-compose的使用方法：



简要说明：命令行中使用了-d参数，如果不加-d参数，则当前终端就会一直附加在docker-compose上，加上该参数，则docker容器就在后台运行。

#### 创建channel

进入cli容器（fabric运行环境只能在docker中）：

docker exec -it cli bash

创建channel： peer channel create，创建channel的配置区块时，指定channel的名字是mychannel这里我们必须创建同样名字的channel。

设置CA证书路径：

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

创建channel：

peer channel create -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/channel.tx --tls true --cafile $ORDERER\_CA

简要说明：创建channel成功时会提示“UTC [channelCmd] readBlock -> DEBU 020 Received block:0”，同时会生成一个mychannel.block文件。

#### 各peer节点加入channel

**在cli容器中，将默认连接的peer0.org1.example.com节点加入mychannel通道中：**

peer channel join -b mychannel.block

简要说明：加入通道成功后会提示“UTC [channelCmd] executeJoin -> INFO 006 Peer joined the channel!”，代表加入通道成功。

其他peer节点加入通道：

修改docker-compose-cli.yaml文件，修改peer节点对应内容：

**peer1.org1.example.com节点加入mychannel通道中：**

CORE\_PEER\_LOCALMSPID="Org1MSP"

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

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

执行peer节点加入通道命令，将peer1.org1.example.com节点加入通道中：

peer channel join -b mychannel.block

**peer0.org2.example.com节点加入mychannel通道中：**

CORE\_PEER\_LOCALMSPID="Org2MSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp

CORE\_PEER\_ADDRESS=peer0.org2.example.com:7051

执行peer节点加入通道命令，将peer0.org2.example.com节点加入通道中：

peer channel join -b mychannel.block

**peer0.org2.example.com节点加入mychannel通道中：**

CORE\_PEER\_LOCALMSPID="Org2MSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer1.org2.example.com/tls/ca.crt

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp

CORE\_PEER\_ADDRESS=peer1.org2.example.com:7051

执行peer节点加入通道命令，将peer1.org2.example.com节点加入通道中：

peer channel join -b mychannel.block

#### 锚节点更新（可选）

锚节点不是完全必须的，不设置锚节点的话，组织内会自动选择一个编号小的peer节点作为锚节点。

**对于组织Org1来说，peer0.org1作为锚节点更新：**

CORE\_PEER\_LOCALMSPID="Org1MSP"

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

CORE\_PEER\_ADDRESS=peer0.org1.example.com:7051

执行锚节点更新：

peer channel update -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/Org1MSPanchors.tx --tls true --cafile $ORDERER\_CA

**对于组织Org2来说，peer0.org2作为锚节点更新：**

CORE\_PEER\_LOCALMSPID="Org2MSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp

CORE\_PEER\_ADDRESS=peer0.org2.example.com:7051

peer channel update -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/Org2MSPanchors.tx --tls true --cafile $ORDERER\_CA

### 链码安装与运行

#### 链码编译打包

安装Fabric后，在路径hyperledger/fabric/examples/chaincode/下会有两类语言的链码实例。一类是基于go语言开发的的链码源码实例，另一类是基于java语言开发的链码源码实例。

链码本身是一个可执行程序，通过背书策略基于远程gRPC调用和peer背书节点进行数据交换，由系统链码LSCC来调度管理链码的生存周期。

链码的编译(go语言)，chaincode\_example01.go和chaincode\_example02.go为例：

编译命令：

go build chaincode\_example01.go

go build chaincode\_example02.go

生成可执行码：

chaincode\_example01

chaincode\_example02

生成的可执行码打包放置到相对路径下：

hyperledger/fabric/examples/chaincode/go/chaincode\_example01

hyperledger/fabric/examples/chaincode/go/chaincode\_example02

简要说明：该过程在这里明确了其细节说明，其实在fabric的链码安装步骤里隐含了调度编译打包的过程。

#### 链码安装

链码的安装需要在各相关的peer背书节点(背书策略定义决定)上进行，这里假设4个peer节点都想对链码chaincode\_example02进行操作，那么就需要安装4次。安装步骤如下：

若cli容器一直是打开状态则忽略下面两行的操作。

进入cli容器（fabric运行环境只能在docker中）：

docker exec -it cli bash

将配置切换到peer0.org1这个peer节点，修改docker-compose-cli.yaml如下：

CORE\_PEER\_LOCALMSPID="Org1MSP"

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

CORE\_PEER\_ADDRESS=peer0.org1.example.com:7051

使用peer chaincode install命令可以安装指定的chaincode\_example02链码并对其命名（-n指定）：

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

简要说明：-n选项指定了安装的链码，-v指定了链码的版本号（不同版本号可以多通道并存，需要注意），其他peer节点的链码安装需要重复上述过程。

#### 链码实例化

链码实例化主要是在peer所在的机器上对前面安装好的链上代码进行实例化,和实际存在的通道建立逻辑连接（同一个链码可以运行在不同的通道上，也就是在不同的通道上进行实例化，且版本可以不同），生成对应channel的docker镜像和docker容器。并且在实例化时我们必须指定背书策略，指定哪些peer节点为背书节点（底层对接系统链码ESCC）。我们运行以下命令完成实例化（init实例化方法）：

peer chaincode instantiate -o orderer.example.com:7050 --tls true --cafile $ORDERER\_CA -C mychannel -n mycc -v 1.0 -c '{"Args":["init","a","100","b","200"]}' -P "OR ('Org1MSP.member','Org2MSP.member')"

简要说明：-c指定的是链码初始化init函数的参数数据，-P指定的是背书策略，这里的选择是只要是Org1MSP.member或Org2MSP.member两个组织的任意一个组织的成员即可满足背书策略（用的逻辑指令字“OR”）。链码实例化后，就将链码装入了新的容器运行，等待cli或应用程序（通过SDK开发的）的提案请求（通过背书节点间接请求），并且将 –c指定的初始化数据通过共识机制写入区块链账本（也写入索引数据库、状态数据库）。链码的实例化在一个通道中只能实例化一次，在通道的一个peer节点上实例化后则不能再在其他同一个通道的其他节点上实例化，但对于通道内的其他peer节点来说，若想使用链码则必须进行链码安装。

链码的实例化，可以根据应用系统的需要自开发应用链码，形成完备的应用系统（完备的应用系统可以通过多通道+多链码的应用部署，属于基于应用系统开发的范畴）。至此，区块链的部署工作基本完毕。