超级账本 Fabric v1.0 多节点集群的部署

张海宁、陈家豪
VMware 中国研发中心

本文读者需要对 Docker，docker-compose 比较熟悉，同时对 Fabric 架构有一定了解。

一、概述

在千呼万唤之后， 犹抱琵琶的超级账本 1.0 GA 版即将揭开面纱， 翘首以待的社区用户将广泛使用这个版本。本文将介绍如何使用 Docker 容器技术来建立起一个多节点 Fabric 集群，并且描述在集群上如何进行基本的操作，如 chaincode 的生命周期维护等。文中采用 Fabric 1.0 beta 的端到端（e2e_cli）示例作为基础来说明原理 。本文提供是为手动配置的方法，今后将介绍利用容器平台（如 K8s 等）自动部署超级账本的方式。

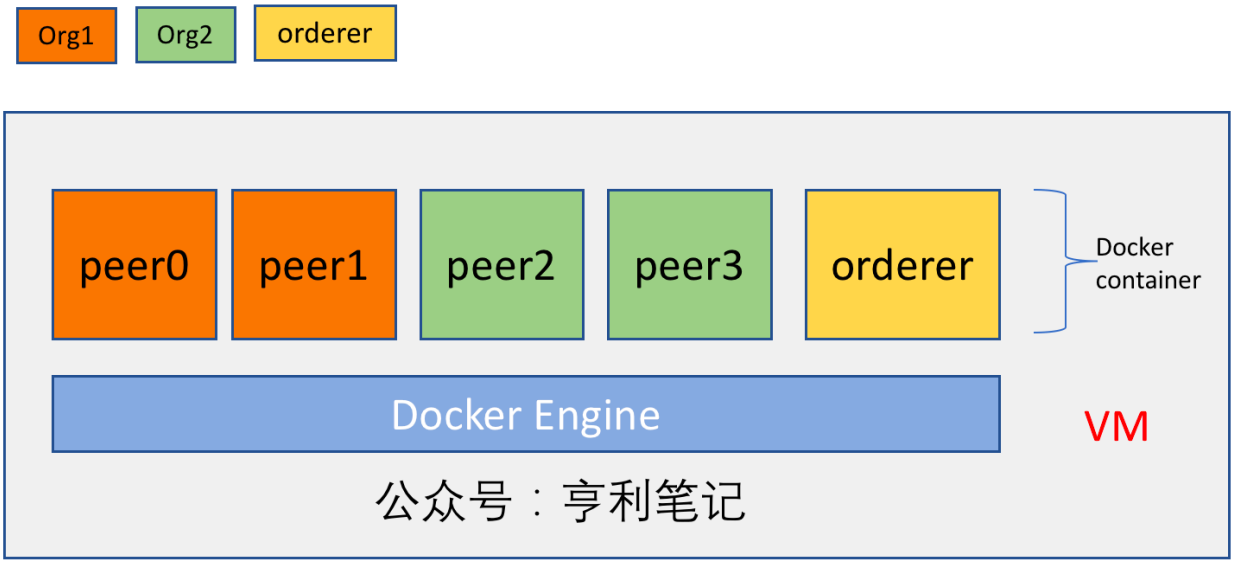


图 1.1 单节点下的 Fabric 网络结构图

Fabric 源码中包含一个简单的 e2e_cli 单机部署示例，方便用户理解、研究和开发应用。如图 1.1 所示， 在单个机器节点上通过 docker-compose 建立了 5 个节点 的 Fabric 网络，每个节点都是由单独的 Docker 容器来模拟。其中 peer0 和 peer1 是同属于 org1 的节点，peer2 和 peer3 是同属于 org2 的节点，它们都加入了相同的 channel 中，并在该 channel 中进行交易，而 orderer 则为该 channel 中的交易提供排序服务。

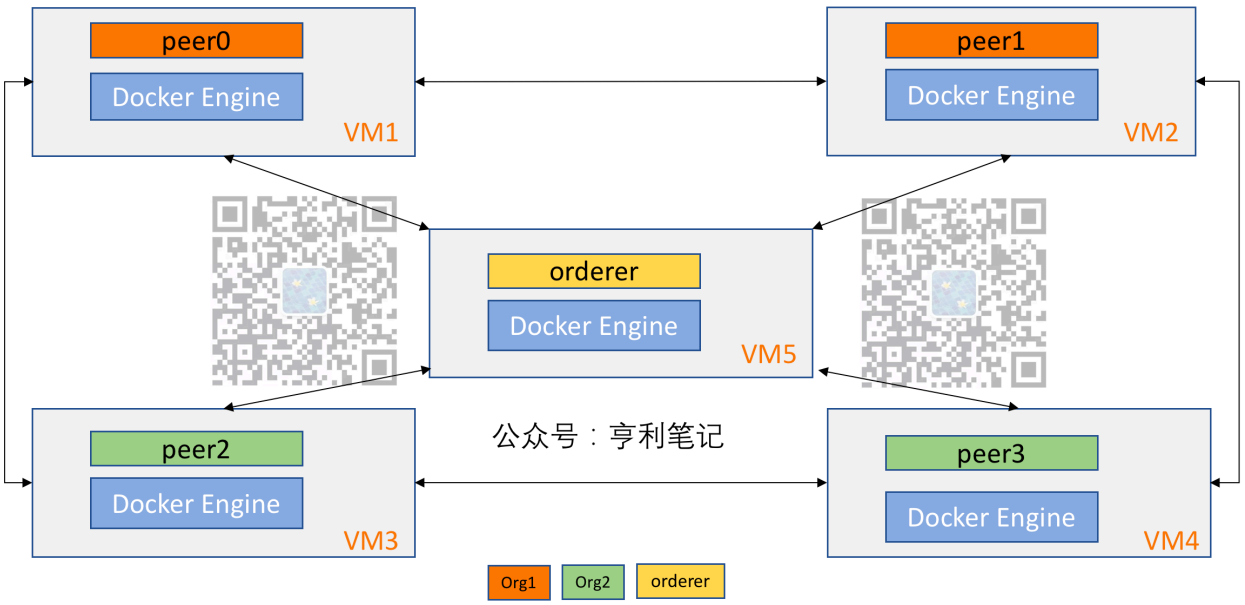


图 1.2 多节点下的 Fabric 网络结构图

e2e_cli 的示例虽然简单，但它把多个节点混合部署在一起，无法区分哪些配置对应哪个节点。另外，在实际场景中，Fabric 节点可能会由不同的组织分别拥有和维护，peers 和 orderer 必然会分布在不同的物理节点上，因此多节点的 Fabric 部署成为需要解决的问题，图 1.2 是多节点 Fabric 集群拓扑图。

下面是把单节点 e2e_cli 示例改为多节点的大致步骤：

1. 准备环境

   运行 Fabric 节点需要依赖以下工具：

   a. Docker ：用于管理 Fabric 镜像以及运行 peer 和 orderer 等组件
   b. Docker-compose： 用于配置 Fabric 容器
   c. Fabric 源码：源码提供了用于生成证书和配置 channel 的工具和测试代码
   d. Go 语言开发环境：源码的工具编译依赖于 Go 语言

2. 配置多节点 Fabric 集群

   在单节点 e2e_cli 示例中，所有节点部署在同一个 docker-compose 的内部网络中，通过容器的 7051 端口进行通信。但是在多节点的情况下，容器之间不能进行直接通讯，因此需要把容器的 7051 端口映射到宿主机上，通过各个宿主机的 7051 端口来实现节点间通信。 我们在每个节点中修改 docker-compose.yaml 中的 service 定义，在不同节点只启动需要的 service。例如，在节点 1 中只启动 peer0 的 service，在节点 5 中仅启动 orderer 等。

3. 启动多节点 Fabric 集群

在各个节点上配置好 Fabric 的启动环境后，需要依次登录到节点上通过 docker-compose up 的方式启动 Fabric 节点。由于启动环境有依赖关系，如 peer1 以 peer0 作为发现节点，因此需要先启动 peer0 再启动 peer1。

4. 配置 channel

在 Fabric 中，channel 代表了一个私有的广播通道，保证了消息的隔离性和私密性，它由 orderer 来管理。channel 中的成员共享该 channel 的账本，并且只有通过验证的用户才能在 channel 中进行交易，与一个 channel 相关的属性记录在该 channel 的初始区块中，可通过 reconfiguration 交易进行更改。 channel 的初始区块由 create channel 交易生成，peer 向 orderer 发送该交易时会带有的 config.tx 文件，该文件定义 channel 的相关属性。

5. 发布 chaincode

chaincode 是开发人员按照特定接口编写的智能合约，通过 SDK 或者 CLI 在 Fabric 的网络上安装并且初始化后，该应用就能访问网络中的共享账本。

chaincode 的生命周期如下:

a. install （安装）

chaincode 要在 Fabric 网络上运行，必须要先安装在网络中的 peer 上，安装同时注明版本号保证应用的版本控制。

b. instantiate （实例化）

在 peer 上安装 chaincode 后，还需要实例化才能真正激活该 chaincode。在实例化的过程中，chaincode 就会被编译并打包成容器镜像，然后启动运行。若 chaincode 在实例化的过程中更新了数据状态，如给某个变量赋予初始值，则该状态变化会被记录在共享账本中。每个应用只能被实例化一次，实例化可在任意一个已安装该 chaincode 的 peer 上进行。

c. invoke 和 query （调用和查询）

chaincode 在实例化后，用户就能与它进行交互，其中 query 查询与应用相关的状态（即只读），而 invoke 则可能会改变其状态。

d. upgrade（升级）

在 chaincode 添加新功能或出现 bug 需要升级时，可以通过 upgrade 交易来实现。这时需要把新的代码通过 install 交易安装到正在运行该 chaincode
的 peer 上，安装时需注明比先前版本更高的版本号，接下来只需要向任意一个安装了新代码的 peer 发送 upgrade 交易就能更新 chaincode，chaincode 在更新前的状态也会得到保留。

二、操作步骤

1、 环境构建与测试

在本文中用到的宿主机环境是 Ubuntu ,版本为 Ubuntu 14.04.5 LTS，通过 Docker 容器来运行 Fabric 的节点, 版本为 v1.0 beta。因此，启动 Fabric 网络中的节点需要先安装 Docker、Docker-compose 和 Go 语言环境，然后在网上拉取相关的 Docker 镜像，再通过配置 compose 文件来启动各个节点。

1.1、Docker 与 Docker-compose 安装

首先切换到 root 用户

```
su root
```

Docker 通过官方提供的脚本可以很方便的安装，先运行以下命令安装 curl：

```
apt-get update && apt-get install curl
```

然后用 wget 来下载脚本并且安装：

```
wget -qO- https://get.docker.com/ | sh
```

当安装完成后，可以通过 docker version 命令来查看 docker 的版本信息：

```
Client:
 Version:      17.05.0-ce
 API version:  1.29
 Go version:   go1.7.5
 Git commit:   89658be
 Built:        Thu May  4 22:10:54 2017
 OS/Arch:      linux/amd64

Server:
 Version:      17.05.0-ce
 API version:  1.29 (minimum version 1.12)
 Go version:   go1.7.5
 Git commit:   89658be
 Built:        Thu May  4 22:10:54 2017
 OS/Arch:      linux/amd64
 Experimental: false
```

接下来就要下载 docker-compose 了，它可以通过 curl 来下载：

```
curl -L https://github.com/docker/compose/releases/download/1.12.0/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose
```

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

执行完毕后可以通过 docker-compose version 来查看 docker-compose 信息：

```
docker-compose version 1.12.0, build b31ff33
docker-py version: 2.2.1
CPython version: 2.7.12
OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
```

docker 和 docker-compose 安装完毕。

1.2、Go 1.8.3 安装：

1). 通过以下命令下载 go1.8.3：

```
curl -O https://storage.googleapis.com/golang/go1.8.3.linux-amd64.tar.gz
```

2). 解压 go1.8.3.linux-amd64.tar.gz 到/usr/local：

```
tar -C /usr/local -xzf go1.8.3linux-adm64.tar.gz
```

3). 在~/.profile 中添加$GOPATH 环境变量，并把 Go 加到$PATH 环境变量，编辑~/.profile 在最后添加两行：

```
export PATH=$PATH:/usr/local/go/bin
export GOPATH=/opt/gopath
```

4). 在终端运行 source ~/.profile 后用 go version 可查看 Go 语言的版本，返回如下信息说明安装成功：

```
go1.8.3 linux/amd64
```

1.3、下载 Fabric 源码：

下载 Fabric 源码是因为要用到源码中提供的例子和工具，工具的编译需要用到 Go 环境，因此需要把源码目录放到$GOPATH 下。通过上面 Go 的安装配置，$GOPATH 设置为/opt/gopath，用以下命令创建并且进入到 hyperledger 目录中：

mkdir -p /opt/gopath/github.com/hyperledger/ && cd /opt/gopath/github.com/hyperledger

从 github 上下载 Fabric 源码：

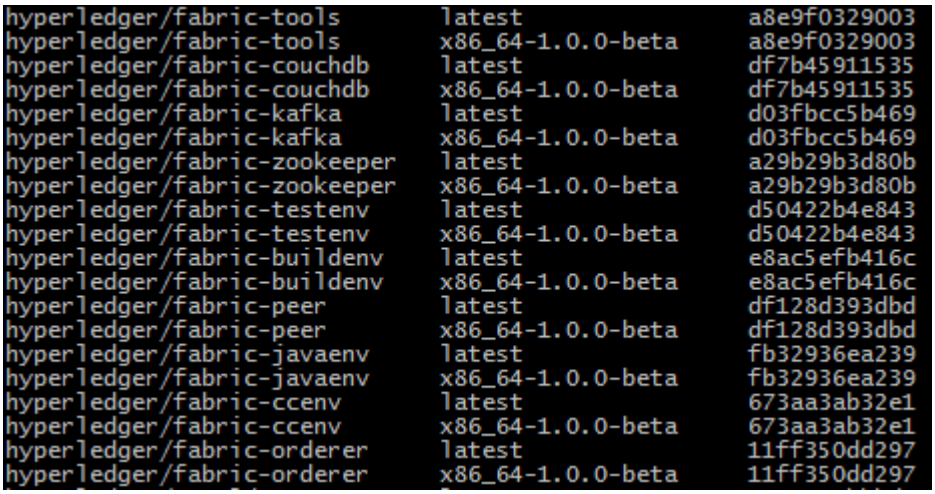
git clone https://github.com/hyperledger/fabric.git

因为镜像使用的是 beta 版本，因此需要把 Fabric 切换到 v1.0.0 beta 源码分支，以兼容 fabric/example/e2e 中的配置:

git checkout v1.0.0-beta

1.4、docker 镜像下载

构建 Fabric 网络所需要用到的 docker 镜像如下：



hyperledger/fabric-tools	latest	a8e9f0329003
hyperledger/fabric-tools	x86_64-1.0.0-beta	a8e9f0329003
hyperledger/fabric-couchdb	latest	df7b45911535
hyperledger/fabric-couchdb	x86_64-1.0.0-beta	df7b45911535
hyperledger/fabric-kafka	latest	d03fbcc5b469
hyperledger/fabric-kafka	x86_64-1.0.0-beta	d03fbcc5b469
hyperledger/fabric-zookeeper	latest	a29b29b3d80b
hyperledger/fabric-zookeeper	x86_64-1.0.0-beta	a29b29b3d80b
hyperledger/fabric-testenv	latest	d50422b4e843
hyperledger/fabric-testenv	x86_64-1.0.0-beta	d50422b4e843
hyperledger/fabric-buildenv	latest	e8ac5efb416c
hyperledger/fabric-buildenv	x86_64-1.0.0-beta	e8ac5efb416c
hyperledger/fabric-peer	latest	df128d393dbd
hyperledger/fabric-peer	x86_64-1.0.0-beta	df128d393dbd
hyperledger/fabric-javaenv	latest	fb32936ea239
hyperledger/fabric-javaenv	x86_64-1.0.0-beta	fb32936ea239
hyperledger/fabric-ccenv	latest	673aa3ab32e1
hyperledger/fabric-ccenv	x86_64-1.0.0-beta	673aa3ab32e1
hyperledger/fabric-orderer	latest	11ff350dd297
hyperledger/fabric-orderer	x86_64-1.0.0-beta	11ff350dd297

进入到 fabric/examples/e2e_cli 目录下，运行./download-dockerimages.sh 来下载必要镜像，镜像下载完成后，就可以通过 docker save 命令把镜像打包成压缩文件，传送到各个 VM。当 VM 接收到压缩文件后，可以通过 docker load 来解压和导入镜像。如果有私有的容器 registry，如 Harbor 等，也可以把镜像推送到私有 registry，再从各个机器中拉取。

通过以下命令来保存所有 tag 含有 beta 标识的镜像到名字为 images 的压缩文件中：

docker save $(docker images | grep beta | awk {‘print $1’}) -o images

生成 images 文件后，就可以通过 scp 把它拷贝到还没有镜像的其他节点中，例如，地址为 10.112.122.6 的节点需要安装以上镜像，可以通过以下命令把 images 远程拷贝到 10.112.122.6 的 home 目录下：

scp images root@10.112.122.6:~

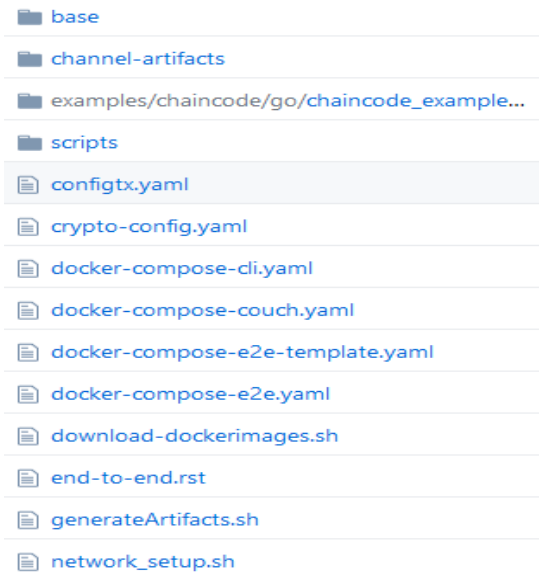
然后 在 10.112.112.6 这台主机的 home 目录上运行：

```
docker load -i images
```

等待一段时间后通过 docker images 命令就能查看到相关镜像的信息。

1.5、运行测试

进入到 fabric/example/e2e_cli 文件夹，文件结构如下：



network_setup.sh 是一键测试脚本，该脚本启动了 6 个 docker 容器，其中有 4 个容器运行 peer 节点和 1 个容器运行 orderer 节点，它们组成一个 Fabric 集群。另外，还有一个 cli 容器用于执行创建 channel、加入 channel、安装和执行 chaincode 等操作。测试用的 chaincode 定义了两个变量，在实例化的时候给这两个变量赋予了初值，通过 invoke 操作可以使两个变量的值发生变化。
通过以下命令执行测试：

```
bash network_setup.sh up
```

接下来会有许多的调试信息，具体可参考 e2e_cli 目录下的 script/script.sh 文件，当终端出现以下信息时说明测试通过，所有部件工作正常：

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

至此，环境配置工作完毕，通过 docker ps -a 命令可以查看各容器的状态。chaincode 会在独立的容器中运行，因此会出现 3 个以 dev 开头的容器，它们与各自的 peer 对应，记录了 peer 对 chaincode 的操作。

```
CONTAINER ID        IMAGE                                  COMMAND                  CREATED             STATUS
817b897d64ef        dev-peer1.org2.example.com-mycc-1.0    "chaincode -peer.a..."   3 minutes ago       Up 3 minutes
5665f47c9e24        dev-peer0.org1.example.com-mycc-1.0    "chaincode -peer.a..."   4 minutes ago       Up 4 minutes
28ce9a1dba00        dev-peer0.org2.example.com-mycc-1.0    "chaincode -peer.a..."   4 minutes ago       Up 4 minutes
18615e039f45        hyperledger/fabric-tools               "/bin/bash -c './s..."   8 minutes ago       Up 8 minutes
625df2554690        hyperledger/fabric-peer                "peer node start"        8 minutes ago       Up 8 minutes
bcbce1a18d60        hyperledger/fabric-peer                "peer node start"        8 minutes ago       Up 8 minutes
e0b5f88298a4        hyperledger/fabric-peer                "peer node start"        8 minutes ago       Up 8 minutes
505c8cfd03dc        hyperledger/fabric-peer                "peer node start"        8 minutes ago       Up 8 minutes
4c24182b0004        hyperledger/fabric-orderer             "orderer"                8 minutes ago       Up 8 minutes
```

2、 创建 Fabric 多节点集群

2.1 前期准备

我们将重现 Fabric 自带的 e2e_cli 例子中的集群，不同的是把容器分配到不同的虚拟机上，彼此之间通过网络来进行通信，网络构建完成后则进行相关的 channel 和 chaincode 操作。

先准备 5 台虚拟机（VM），所有虚拟机均按照上述环境构建与测试步骤配置，当然也可安装一个虚拟机模板，然后克隆出其他虚拟机。其中 4 台虚拟机运行 peer 节点，另外一台运行 orderer 节点，为其他的四个节点提供 order 服务。

在本文中，虚拟机的参数如下：

Name	IP	节点标识	节点 Hostname	Organization
VM1	10.112.122.144	peer0	peer0.org1.example.com	Org1
VM2	10.112.122.46	peer1	peer1.org1.example.com	Org1
VM3	10.112.122.12	peer2	peer0.org2.example.com	Org2
VM4	10.112.122.45	peer3	peer1.org2.example.com	Org2
VM5	10.112.122.69	orderer	orderer.example.com	Orderer

2.2 使用 generateArtifacts.sh 生成证书和 config.tx

在任意 VM 上运行 fabric/examples/e2e_cli 目录下的 generateArtifacts.sh 脚本，可生成两个目录，它们分别为 channel-artifacts 和 crypto-config，两个目录的结构分别如下：

```
-channel-artifacts
   -channel.tx
    -genesis.block
   -Org1MSPanchors.tx
   -Org2MSPanchors.tx
```

上述目录里的文件用于 orderer 创建 channel，它们根据 configtx.yaml 的配置生成。

```
-crypto-config
  -ordererOrganizations
  -peerOrganizations
```

上述目录里面有 orderer 和 peer 的证书、私钥和以及用于通信加密的 tls 证书等文件，它通过 configtx.yaml 配置文件生成。

2.3 配置

以下各 VM 的工作目录为：

$GOPATH/src/github.com/hyperledger/fabric/examples/e2e_cli

可在任意 VM 上运行以下命令生成构建 Fabric 网络所需的成员证书等必要材料：

```
bash generateArtifacts.sh
```

该命令只需在某个 VM 上运行一次，其他 VM 上就不需要运行。

在运行该命令的 VM 中会生成 channel-artifacts 和 crypto-config 目录，需要把它们拷贝到其他 VM 的 e2e_cli 目录下，如果在 VM 中已经存在该目录则先把目录删除。当每个 VM 中都有统一的 channel-artifacts 和 crypto-config 目录后接下来就开始配置 compose 文件。

I.VM1 配置：

1. 修改/etc/hosts 的映射关系

因为容器内部通过域名的方式访问 orderer,因此需要通过修改/etc/hosts 把 orderer 的域名和 ip 地址对应起来，在文件中添加:

```
10.112.122.69    orderer.example.com
```

2. 修改 docker-compose-cli.yaml

在默认的情况下，docker-compose-cli.yaml 会启动 6 个 service（容器），它们分别为 peer0.org1.example.com、peer1.org1.example.com、peer0.org2.example.com、peer1.org2.example.com、orderer.example.com 和 cli，因为每台机器只运行与之对应的一个节点，因此需要注释掉无需启动的 service。

(1) 除 peer0.org1.example.com 和 cli service 外，其他 service 全部注释。

(2) 在 cli 的 volumes 中加入映射关系：

```
- ./peer/:/opt/gopath/src/github.com/hyperledger/fabric/peer/
- /etc/hosts:/etc/hosts
```

(3) 注释cli中的depends_on和command:

```
   depends_on:
      #- orderer.example.com
      - peer0.org1.example.com
      #- peer1.org1.example.com
      #- peer0.org2.example.com
      #- peer1.org2.example.com

      #command: /bin/bash -c './scripts/script.sh ${CHANNEL_NAME}; sleep      $TIMEOUT'
```

之前我们把容器中的工作目录挂载到宿主机的 e2e_cli/peer 目录下，是因为在执行 create channel 的过程中，orderer 会返回一个 mychannel.block 作为 peer 加入 channel 的依据，其他的 peer 要加入到相同的 channel 中必须先获取 mychannel.block。因此，通过挂载目录从宿主机就能方便获得该 mychannel.block，并且把它传输到其他的 VM 上。
挂载/etc/hosts 的目的是把主机中 orderer.exmaple.com 与 IP 地址 10.112.122.69 的映射关系带入容器中,目的是让 cli 能通过域名访问 orderer。在实际环境中，建议通过配置 DNS 而不是修改 /etc/hosts 文件（下同）。

3. 修改 base/peer-base.yaml，添加 volumes

```
volumes:
  -/etc/hosts:/etc/hosts
```

这样 peer 容器能通过域名访问 orderer。

II.VM2 配置：

1. 修改/etc/hosts 的映射关系

peer1.org1.example.com 使用了 peer0.org1.example.com 作为它的初始化节点，因此需要在映射关系中还需要加入 VM1 的 ip 地址。

```
10.112.122.69   orderer.example.com
10.112.122.144  peer0.org1.example.com
```

2. 修改 docker-compose-cli.yaml

(1) 类似 VM1，除 peer1.org1.example.com 和 cli service 外，其他 service 全部注释

(2) 在 cli 的 volumes 中加入映射关系：

```
- ./peer/:/opt/gopath/src/github.com/hyperledger/fabric/peer/
- /etc/hosts:/etc/hosts
```

(3) 注释cli中的depends_on和command:

```
depends_on:
 #- orderer.example.com
 #- peer0.org1.example.com
 - peer1.org1.example.com
 #- peer0.org2.example.com
 #- peer1.org2.example.com

#command: /bin/bash -c './scripts/script.sh ${CHANNEL_NAME}; sleep $TIMEOUT'
```

(4) 修改 cli 中的环境变量

```
CORE_PEER_ADDRESS=peer1.org1.example.com:7051
```

3. 修改 base/peer-base.yaml，同 VM1 的修改。

III.VM3 配置：

1、 修改/etc/hosts 的映射关系

```
10.112.122.69    orderer.example.com
```

2、 修改 docker-compose-cli.yaml

（1）VM3 上运行 peer2 节点，因此除 peer0.org2.example.com 和 cli service 外,其他 service 全部注释。

（2）在 cli 的 volumes 中加入映射关系：

```
- ./peer/:/opt/gopath/src/github.com/hyperledger/fabric/peer/
- /etc/hosts:/etc/hosts
```

（3）注释cli中的depends_on和command:

```
depends_on:
 #- orderer.example.com
 #- peer0.org1.example.com
 #- peer1.org1.example.com
 - peer0.org2.example.com
 #- peer1.org2.example.com

#command: /bin/bash -c './scripts/script.sh ${CHANNEL_NAME}; sleep $TIMEOUT'
```

（4）修改 cli 中的环境变量

```
CORE_PEER_LOCALMSPID="Org2MSP"
CORE_PEER_ADDRESS=peer0.org2.example.com:7051
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
```

3、 修改 base/peer-base.yaml，同 VM1。

IV.VM4 配置：

1. 修改/etc/hosts 的映射关系

peer1.org2.example.com 使用了 peer0.org2.example.com 作为它的初始化节点，因此需要在映射关系中加入 VM3 的 ip 地址

```
10.112.122.69     orderer.example.com
10.112.122.12     peer0.org2.example.com
```

2. 修改 docker-compose-cli.yaml

(1) VM4 运行 peer3，因此除 peer1.org2.example.com 和 cli service 外,其他 service 全部注释

(2) 在 cli 的 volumes 中加入映射关系：

```
- ./peer/:/opt/gopath/src/github.com/hyperledger/fabric/peer/
- /etc/hosts:/etc/hosts
```

(3) 修改cli中的depends_on和command:

```
depends_on:
  - peer1.org2.example.com
 #command: /bin/bash -c './scripts/script.sh ${CHANNEL_NAME}; sleep $TIMEOUT'
```

(4) 修改 cli 中的环境变量

```
CORE_PEER_LOCALMSPID="Org2MSP"
CORE_PEER_ADDRESS=peer1.org2.example.com:7051
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/peerOrganizat
  ions/org2.example.com/users/Admin@org2.example.com/msp
```

4、 修改 base/peer-base.yaml，同 VM1。

V.VM5 配置如下：

1.修改 docker-compose-cli.yaml
除 orderer 外的其他 service 全部注释，即只启动 orderer。

2.4 启动多节点集群

1. 启动 orderer

进入到 VM5 的 fabric/examples/e2e_cli 目录下，运行

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

此时终端会出现大量记录，当出现 Beginning to service requests 时，orderer 启动完成。有了 orderer 之后，就可以通过它来管理 channel 。

2. 启动 org1 的第一个节点 peer0，即 peer0.org1.example.com

进入到 VM1 的 fabric/examples/e2e_cli 目录下，运行

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

此时通过 docker ps -a 命令可以看到成功启动了 peer0.org1.example.com 和 cli 两个容器。接下来实现创建 channel、加入 channel 和安装 chanicode。

首先进入到 cli 容器内部：

```
docker exec -it cli bash
```

cli 与 orderer 之间的通讯使用 tls 加密，设置环境变量 ORDERER_CA 以作建立握手的凭证：

```
$ORDERER_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/examp
le.com/orderers/orderer.example.com/msp/cacerts/ca.example.com-cert.pem
```

注：以下所有涉及到 ORDERER_CA 环境变量的命令都需预先给该变量赋值。

进入到 cli 容器后会自动跳转到/opt/gopath/src/github.com/hyperledger/fabric/peer 目录，即工作目录，通过 compose 文件的配置，该目录映射为宿主机的/e2e_cli/peer。

在工作目录下输入以下命令，创建名为 mychannel 的 channel：

```
peer channel create -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/channel.tx --tls --cafile $ORDERER_CA
```

channel 创建成功后，会在当前目录下生成 mychannel.block 文件。每个 peer 在向 orderer 发送 join channel 交易的时候，需要提供这个文件才能加入到 mychannel 中，因此运行在其他 VM 上的 peer 需要得到 mychannel.block 文件来加入到 mychannel 中。由于之前的文件映射关系，mychannel.block 文件可在宿主机的 e2e_cli/peer 目录下获取，这时可以通过宿主机把 mychannel.block 拷贝到 VM2, VM3, VM4 的 e2e_cli/peer 目录下。

把 peer0.org1.example.com 加入到 mychannel 中：

```
peer channel join -b mychannel.block
```

更新 mychannel 中 org1 的 anchor peer 的信息：

```
peer channel update -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/Org1MSPanchors.tx --tls --cafile $ORDERER_CA
```

安装 chaincode 示例 chaincode_example02 到 peer0.org1.example.com 中：

```
peer chaincode install -n mycc -v 1.0 -p \
github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02
```

这时候 chaincode 代码已经安装到了 peer0 节点上，但并未实例化运行。接下来先配置好其他节点。

3. 启动 org1 的第二个节点 peer1，即 peer1.org1.example.com

进入到 VM2 的 fabric/examples/e2e_cli 目录下，运行

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

进入到 cli 容器内部：

```
docker exec -it cli bash
```

由于前面已经把 mychannel.block 拷贝到了 VM2 的 e2e_cli/peer 目录下，因此 mychannel.block 可通过容器内的/opt/gopath/src/github.com/hyperledger/fabric/peer 目录访问。

把 peer1.org1.example.com 加入到 mychannel 中：

```
peer channel join -b mychannel.block
```

安装 chaincode_example02 到 peer1.org1.example.com 中：

```
peer chaincode install -n mycc -v 1.0 -p \
```

```
github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02
```

4. 启动 org2 的第一个节点 peer2，即 peer0.org2.example.com
进入到 VM3 的 fabric/examples/e2e_cli 目录下，运行

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

进入到 cli 容器内部：

```
docker exec -it cli bash
```

把 peer0.org2.example.com 加入到 mychannel 中：

```
peer channel join -b mychannel.block
```

更新 mychannel 中 org2 的 anchor peer 的信息：

```
peer channel update -o orderer.example.com:7050 -c mychannel -f ./channel-
artifacts/Org2MSPanchors.tx --tls --cafile $ORDERER_CA
```

安装 chaincode_example02 到 peer0.org2.example.com 中：

```
peer chaincode install -n mycc -v 1.0 -p \
github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02
```

5. 启动 org2 的第二个节点 peer3，即启动 peer1.org2.example.com

进入到 VM4 的 fabric/examples/e2e_cli 目录下，运行

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

首先进入到 cli 容器内部：

```
docker exec -it cli bash
```

把 peer1.org2.example.com 加入到 mychannel 中：

```
peer channel join -b mychannel.block
```

安装 chaincode_example02 到 peer1.org2.example.com 中：

```
peer chaincode install -n mycc -v 1.0 -p \
github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02
```

2.5 运行 chaincode

通过前面的步骤，整个多节点 Fabric 网络已经运行起来了，每个 peer 都加入到了标识为 mychannel 的 channel 中，并且都安装了一个简单的 chaincode(该 chaincode 在安装时被标识为 mycc）。下面步骤运行和维护 chaincode。

1. 实例化 chaincode

chaincode 的实例化可在任意 peer 上进行，并且 chaincode 只能被实例化一次，下面以在 peer0.org2.example.com 上实例化 chaincode 为例。

首先登录 VM3 并进入到 cli 容器内部运行：

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

这时候会构建一个新的容器来运行 chaincode，通过 docker ps -a 命令可以看到新容器：

```
dev-peer0.org2.example.com-mycc-1.0
```

上述实例化中，我们对两个变量‘a’和‘b’分别赋予初值 100 和 200，通过 channel 它们的值被同步到了其他 peer 的账本上，即使其他 peer 还没有构建运行 chaincode 的容器。

2. 执行 chaincode 的 query 交易

由于 chaincode 已经被 peer0.org2.example.com 实例化了，因此其他 peer 不需要再次实例化它了，但是 chaincode 的状态（world state）却是已经记录在各个 peer 的账本上的。

接下来我们在 peer0.org1.example.com 上查看 chaincode 的状态，登录到 VM1 上并进入 cli 容器内部执行：

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

上面的命令查看 mycc 中变量 a 的值，由于在 peer 跟 chaincode 发生互动之前还不存在运行 chaincode 的容器，因此第一次交互的时候需要先构建运行 chaincode 的容器，等待一段时间后返回结果： 100 。

此时通过 docker ps -a 命令能看到新容器：

dev-peer0.org1.example.com-mycc-1.0

该值与实例化时的赋值一致，说明 peer0.org1 和 peer0.org2 两个 peer 可以相互通信。

3. 执行 chaincode 的 invoke 交易

接下来，我们执行一个 invoke 交易，使得变量 a 向变量 b 转帐 20，得到最终值为 ["a":"80","b":"220"]。

登录到 VM2 并进入到 cli 容器中中通过以下命令查询 mycc 的状态：

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

稍作等待后返回结果为 100，下面执行 invoke 交易，改变 a 的值为 80：

```
peer chaincode invoke -o orderer.example.com:7050  --tls --cafile $ORDERER_CA -C mychannel -n
  mycc -c '{"Args":["invoke","a","b","20"]}'
```

4. 再次执行 chaincode 的 query 交易

在 peer1.org1.example.com 上重复以上查看 chaincode 的步骤，得到返回结果为 80，说明测试通过，至此，Fabric 网络构建完毕，各个部件工作正常。

2.6 更新 chaincode

通过 channel upgrade 命令可以使得 chaincode 更新到最新的版本，而低版本 chaincode 将不能再使用。

登录到 VM1 的 cli 容器中再次安装 chaincode_example02，但赋予它更高的版本号 2.0：

```
peer chaincode install -n mycc -v 2.0 -p \
github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02
```

在 VM1 的 cli 容器升级 chaincode,添加两个变量‘c’和‘d’：

```
peer chaincode upgrade -o  orderer.example.com:7050 --tls --cafile $ORDERER_CA \
-C mychannel -n mycc -v 2.0 -c '{"Args":["init","c", "10", "d","20"]}'
```

等待一段时间后，可以通过 docker ps -a 来查看新容器构建的容器,该容器的名称为：

dev-peer0.org1.example.com-mycc-2.0

通过以下命令查询 c 的变量：

```
peer chaincode -n mycc -C mychannel -v 2.0 -c '{"Args":["query","c"]}'
```

返回结果为 10 。

再次查询 a 的变量：

```
peer chaincode -n mycc -C mychannel -v 2.0 -c '{"Args":["query","a"]}'
```

返回结果为 80，说明更新 chaincode 成功。

这时候对账本的修改会通过 orderer 同步到其他 peer 上，但是在其他 peer 上将无法查看或更改 chaincode 的状态，因为它们还在使用旧版的 chaincode，所以其他 peer 要想正常访问还需再次安装 chaincode，并且设置相同的版本号(chaincode 代码没发生改变，只是安装时版本号更新为 2.0)，命令如下：

```
peer chaincode install -n mycc -v 2.0 -p \
github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02
```

结束语

本文介绍了如何构建多节点 Fabric 集群的基本方法。为说明原理，安装配置过程是全手动的，因此比较繁琐。今后我们将介绍如何使用自动化方式，如 Ansible 或容器平台 K8s 等部署 Fabric，敬请关注。