**部署方案说明**

# 操作系统

* 推荐使用Ubuntu 16.04 64bits，本文后续的说明都是基于此发行版

# 安装Docker

* 参考 <https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/>

$ sudo apt-get install -y \

apt-transport-https \

ca-certificates \

curl \

software-properties-common

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

$ sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable"

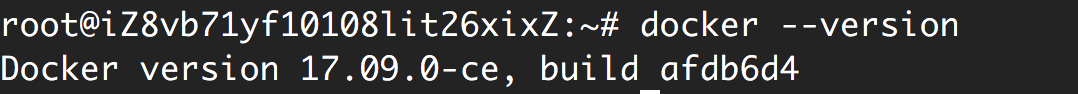
$ sudo apt-get update

$ sudo apt-get install –y docker-ce

$ sudo curl -L https://github.com/docker/compose/releases/download/1.18.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

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

* 检查Docker版本



# 部署准备

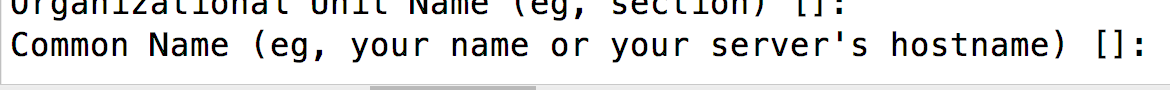
* 参见《01\_开发环境搭建》，部署好开发环境
* 进入abchain主项目的deployment目录，或者将此目录中的内容复制到部署工作将要进行的目录，后面所述部署步骤均将在此目录下进行
* 可以参考5.1节首先编译和准备base镜像

# 准备SSL证书

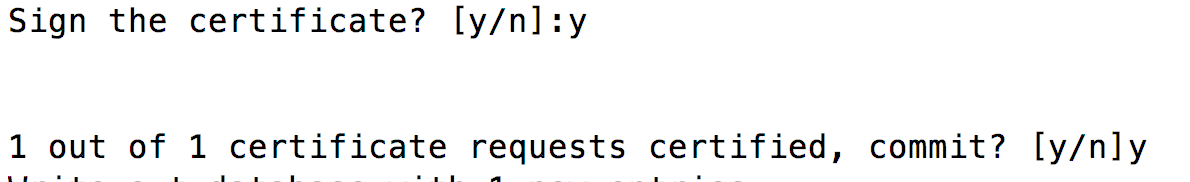
Fabric peer之间如果需要使用SSL加密通讯内容，需要为每个通讯节点提供一份所需的证书。通常的做法是使用一个根证书为每个节点都单独签发一份证书。certs目录下提供了生成证书所需的脚本

## 生成新证书

* 进入 certs目录
* 参考 build-all.sh 生成证书，相关配置可以在fabric.cnf文件中进行修改
  + 对每个节点专用的证书，Common Name 通常填写访问此节点使用的域名



* + 对下面两个提示均响应 y



## 证书的使用

build-all.sh生成一个根证书（ca.crt）和为两个fabric节点（gateway和membersrvc）分别签发的证书，以及对应的私钥（.key文件）。copy-certs.sh脚本将证书复制到证书应当安装的目标目录。

* 根证书（ca.crt/key）可用于为每个节点签发证书，在ca.key生成后，使用build-key.sh脚本可以继续签发单独的证书
* ca.crt应当安装到每个节点，用于节点识别来自其它节点的SSL通讯所用证书的合法性
* 为每个节点生成的证书和密钥（.crt和.key文件）的配置方式在后文详述

# 准备Docker镜像文件

fabric基于docker的部署要求一个基础的docker镜像，其中要求提供和开发环境类似的go编译环境，rocksdb和其它依赖库的支持。base目录提供生成此镜像需要的脚本和文件

## 编译base镜像

* 在base目录下执行build.sh，成功后将在本地生成abchain/fabric-baseimage镜像，此镜像也被标记为hyperledger/fabric-baseimage以提供向前兼容性

## 其它镜像和启动容器

运行fabric系统的各个主要角色（membersrvc，validator和peer）所需的镜像构建文件（Dockerfile）在部署目录的membersrvc，vp和nvp下。在部署目录下通过docker compose可以构建相应的镜像：

* 执行 docker-compose build
  + 这一命令使用部署目录下的docker-compose.yml脚本构建出membersrvc, vp和nvp对应的docker镜像
  + 这些镜像的构建要求提供SSL使用的证书和密钥，第4节中的脚本会将需要的内容复制到对应的目录中
* 执行 docker-compose up -d
  + 这一命令在本机启动一组完整的fabric系统实例，包括1个 CA 实例、4个VP 实例以及1个NVP 实例，采用 PBFT 共识机制

## 账户登陆和信息存储

fabric系统使用中心化的账户体系，由CA实例（membersrvc）进行管理。用户经过特定的peer登陆其账户，并从CA获取证书保存在peer的数据目录下。用户的登陆是一次性的，此后用户必须通过其第一次登陆的peer来提交事务。

* 用户的证书等数据存放在peer的数据目录下的crypto/client目录中和用户名相同的子目录下
* 可以进入已经运行的fabric peer容器中为此peer登陆特定的用户，例如，在5.2节中启动的示例系统内，进入nvp实例容器，执行下列命令

$ peer network login user1 -p EcEeDZw8Jf8J

成功后，此nvp实例可使用user1提交事务

# 系统部署

部署目录的membersrvc，vp和nvp下包含了单独运行fabric节点所需的Docker容器文件，独立的Docker compose脚本和配置文件模板，用于在生产环境中部署fabric系统。通常的部署流程如下：

## 节点环境要求

* 部署fabric节点的系统必须完成01号文档（开发环境搭建）中的golang运行环境配置和代码获取流行，以及本文档第2节的docker安装流程
* 基于安全性的要求，fabric系统各节点之间的通讯应当在内网中实现。另一方面，节点可以面向互联网开放部分服务，以便允许用户通过SDK接入。我们通常将节点可被发现和访问的信息称为标识（例如IP或域名），因此一个节点应当具有内部标识（在内网中被发现和访问）和外部标识（在互联网上被发现和访问）

## 部署准备

* 执行第4节的证书准备流程，为系统各角色的fabric节点提供SSL证书。在典型的部署方案中，仅需要在向外部提供服务（例如SDK接入服务）的通讯中使用SSL进行加密，因此签发证书的CN应当设置为节点的外部标识（外部IP或域名）
* 执行5.1和5.2节的流程构建docker镜像，并将相应的镜像分发到目标节点

## 部署示例

在下面的示例中，我们在一组系统实例下完成一个和5.2节中示例系统类似的部署。此部署总共要求6个系统实例（1CA + 4VP + 1NVP），架构如下图所示（注意sdk不属于系统的一部分）：

VP1

VP2

VP4

VP3

CA

NVP

sdk

sdk

* CA和NVP实例包含对外开放的服务，因此，仅需为NVP和CA实例签发用于SSL通讯的证书
* 将abchain/vp镜像分发到实例VP1~4（或在实例上创建），abchain/nvp镜像分发到实例NVP，abchain/membersrvc镜像分发到实例CA
* 为方便起见，将deployment目录中的内容分发到每个实例
* 在生产环境下，我们将数据存储到实例本地；这通过docker的目录映射实现。deployment目录下提供的docker composer脚本指定了映射的目录位置，必须在每个实例上创建对应的映射目录

## CA 实例的部署

* 在CA实例中，进入deployment目录的 membersrvc 子目录
* 修改 docker-compose.yml 中的配置
  + volumes 设置数据在实例的保存位置，默认使用 ~/abchain/membersrvc，需要创建此目录

$ mkdir –p ~/abchain/membersrvc

* + ports 设置host 到container的端口映射，必须为container的8002 端口提供对外访问入口（例如member.abchain.org:8002）
* 修改 membersrvc.yaml 中的配置
  + 将aca.address修改为CA节点的对外访问入口，并将server-name修改为入口的域名（实际是CA节点所用SSL证书的CN）
  + 配置文件中已经包含一组预设的用户数据，可要求技术支持按需求提供用户数据或参考fabric技术文档进行修改
* 启动 CA 实例

$ docker-compose up -d

## NVP实例的部署

* 在NVP实例中，进入deployment目录的 nvp子目录
* 修改 docker-compose.yml 中的配置
  + 类似地，volumes 设置数据在实例的保存位置，使用默认的配置需要创建~/abchain/nvp目录：

$ mkdir –p ~/abchain/nvp

* + 在ports中，为container的7051 端口提供**内部**访问入口，例如NVP实例的内网IP为192.168.0.1，则可设置访问入口为192.168.0.1:7051
  + 为container的8000端口提供接入SDK的Server服务
  + 可选地，为container的8080端口提供区块链状态查询的Web服务
  + CORE\_PEER\_ID 为全网唯一的节点 ID，必须符合nvp节点的命名规范（nvp[0-90-9]）
  + CORE\_SECURITY\_ENROLLID为标识节点的 NVP账号名，依据 membersrvc.yaml 进行设置
  + CORE\_SECURITY\_ENROLLSECRET为节点的 NVP账号密码，依据 membersrvc.yaml 进行设置
  + CORE\_PEER\_DISCOVERY\_ROOTNODE为节点发现参数，在我们的系统中，此NVP节点负责节点发现，因此不需提供
* 启动 NVP 实例

$ docker-compose up –d

## VP实例的部署

以实例VP1为例，其它实例的部署过程可类推

* 在VP1实例中，进入deployment目录的 vp子目录
* 修改 docker-compose.yml 中的配置
  + 和NVP节点类似，默认情况下创建~/abchain/vp目录用于保存数据：

$ mkdir –p ~/abchain/vp

* + 和NVP节点相同，在ports中，为container的7051 端口提供**内部**访问入口例如192.168.0.2:7051
  + 指定CORE\_PEER\_ADDRESS为上一步的内部访问入口，节点将此参数广播到其它节点以实现各节点之间的通讯
  + CORE\_PEER\_ID 为全网唯一的节点 ID，必须符合vp节点的命名规范（vp[0-90-9]）
  + 和NVP实例类似，通过CORE\_SECURITY\_ENROLLID和CORE\_SECURITY\_ENROLLSECRET指定标识此节点的 VP账号名和密码，依据 membersrvc.yaml 进行设置
  + 在我们的系统中，设置CORE\_PEER\_DISCOVERY\_ROOTNODE为NVP实例的内部访问入口，（例如6.4节中的示例192.168.0.1:7051）
* 启动单个 VP 实例

$ docker-compose up -d

至此，系统部署完成，可以支持应用SDK的接入，部署chaincode并开始运行联盟链