设置网络

本文介绍了如何在您的本地计算机上设置网络，以执行不同的开发和测试活动。除非您打算参与 Hyperledger Fabric 项目的开发，否则您可能会希望直接采用下面这种更为常用的方法 - 利用发布的 Docker 镜像来处理不同的 Hyperledger Fabric 组件。如果不是这种情况，请跳过该部分，直接进入下面的第二种方法。

利用发布的 Docker 镜像

本方法直接利用 Hyperledger Fabric 项目发布在 DockerHub 上的 Docker 镜像，包括您希望创建的网络的 Docker 命令或 Docker Compose 描述。

安装 Docker

注：在本地的 Mac 和 Windows 上运行 Docker 时，不提供 IP 转发支持。因此，我们建议您不要同时运行多个 Fabric Peer 镜像，因为您可能不希望在同一个端口上绑定多个应用。对于大多数应用和链码而言，通过单个 Fabric Peer 节点运行的开发/测试都不成问题，但如果您较为关注 Fabric 的性能和功能弹性，例如一致性，则另当别论。有关更高级的测试，我们强烈建议您使用 Fabric 的 Vagrant 开发环境。

在使用该方法时，可以选择多种方式来运行 Docker，例如使用 Docker Toolbox，或者对于 Mac OSX 或 Windows 系统，也可以使用某个新的本地 Docker 运行时环境。在 Mac 和 Windows 系统上运行 Docker 与在 Linux 上的虚拟化情境中运行 Docker 有一些细微的差别。在涉及到各种组件的具体运行时，我们将在适当的地方对这些差别进行解释。

从 DockerHub 中拉取镜像

安装并运行了 Docker(1.11 版或更高版本)后，在启动任何 Fabric 组件之前，首先需要从 DockerHub 中拉取镜像。

```
docker pull hyperledger/fabric-peer:latest

docker pull hyperledger/fabric-membersrvc:latest
```

构建镜像

注：对于大多数用户而言，我们并不推荐该方法。如果您已按照前面所述从 DockerHub 拉取了镜像，即可进行下一步操作。

第二种方法是利用开发环境设置（假定您已经安装了开发环境），以便通过 hyperledger/fabric GitHub 仓库中的拷贝构建并部署您自己的二进制文件和/或 Docker 镜像。该方法适合于希望直接参与 Hyperledger Fabric 项目的开发人员，或者希望从 Hyperledger 代码库进行部署的开发人员。

应在设置开发环境一节中描述的 Vagrant 环境中运行以下命令。

若要创建 hyperledger/fabric-peer Docker 镜像：

```
cd $GOPATH/src/github.com/hyperledger/fabric

make peer-image
```

若要创建 hyperledger/fabric-membersrvc Docker 镜像：

```
make membersrvc-image
```

启动多个验证 Peer 节点

使用 docker images 再次核对可用镜像。此时，您应能看到 hyperledger/fabric-peer 和 hyperledger/fabric-membersrvc 镜像。例如，

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hyperledger/fabric-membersrvc	latest	7d5f6e0bcfac	12 days ago	1.439 GB
hyperledger/fabric-peer	latest	82ef20d7507c	12 days ago	1.445 GB

如果看不到这些镜像，则返回到上一步。

通过已有的相关 Docker 镜像，我们可以开始运行 Peer 节点和 membersrvc 服务。

确定 CORE_VM_ENDPOINT 变量的值

接下来，我们需要针对 CORE_VM_ENDPOINT 确定 Docker daemon 程序的地址。在 Vagrant 开发环境或 Docker Toolbox 环境中，可以通过 ip add 命令确定该地址。例如，

```
$ ip add

<<< detail removed >>>

3：docker0:<NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default

    link/ether 02:42:ad:be:70:cb brd ff:ff:ff:ff:ff:ff

    inet 172.17.0.1/16 scope global docker0

        valid_lft forever preferred_lft forever
```

```
    inet6 fe80::42:adff:febe:70cb/64 scope link
       valid_lft forever preferred_lft forever
```

您的输出中可能包含类似 172.17.0.1/16 scope global docker0 的 inet。这表示 docker0 接口的 IP 地址为 172.17.0.1。对 CORE_VM_ENDPOINT 选项使用该 IP 地址。有关环境变量的更多信息，请参阅 fabric 存储库中的 core.yaml 配置文件。

如果您针对 Mac 或 Windows 系统使用本地 Docker，则 CORE_VM_ENDPOINT 的值应设置为 unix:///var/run/docker.sock。需要确认检查该设置。设置为 127.0.0.1:2375 应该也是可以的。

对 CORE_PEER_ID 进行赋值

每个验证 Peer 节点的 CORE_PEER_ID 的 ID 值必须具有唯一性，而且必须为小写字母字符串。对于验证 Peer 节点的命名，我们通常使用约定俗成的方法，即 vpN；其中 N 是一个整数，从 0 开始增加，0 表示根节点，每增加一个附加节点，该数值便增加 1，例如 vp0、vp1、vp2 等。

一致性

默认情况下，我们使用名为 NOOPS 的一致性插件，该插件并不会真正地执行一致性。在运行单个 Peer 节点时，如果运行 NOOPS 以外的其他插件，则意义不大。如果您希望在多个 Peer 节点情境中使用其他一致性插件，请参阅下文的使用一致性插件部分。

Docker Compose

我们将使用 Docker Compose 来启动各种 Fabric 组件容器，这是最简单的方法。首先，您应该通过初始设置步骤安装 Docker Compose。通过安装 Docker Toolbox 或任何本地 Docker 运行时均可附带安装 Compose。

启动单个验证 Peer 节点：

让我们启动第一个验证 Peer 节点（根节点）。将 CORE_PEER_ID 设置为 vp0，并按照上文所述设置 CORE_VM_ENDPOINT 的值。以下所示即为在 **Vagrant** 开发环境中启动单个容器所需的 docker-compose.yml。

```
vp0:
  image: hyperledger/fabric-peer
  environment:
    - CORE_PEER_ID=vp0
    - CORE_PEER_ADDRESSAUTODETECT=true
    - CORE_VM_ENDPOINT=http://172.17.0.1:2375
    - CORE_LOGGING_LEVEL=DEBUG
```

```
  command: peer node start
```

您可以从与 docker-compose.yml 文件相同的目录通过以下命令启动该 Compose。

```
$ docker-compose up
```

以下所示为相应的 Docker 命令：

```
$ docker run --rm -it -e CORE_VM_ENDPOINT=http://172.17.0.1:2375 -e
CORE_LOGGING_LEVEL=DEBUG -e CORE_PEER_ID=vp0 -e
CORE_PEER_ADDRESSAUTODETECT=true hyperledger/fabric-peer peer node start
```

在针对 Mac 或 Windows 系统运行 Docker 时，我们需要明确地映射端口，也需要将 CORE_VM_ENDPOINT 设置为一个与前述讨论不同的值。

以下所示为针对 Mac 或 Windows 系统上的 Docker 的 docker-compose.yml：

```
vp0:
  image: hyperledger/fabric-peer
  ports:
    - "7050:7050"
    - "7051:7051"
    - "7052:7052"
  environment:
    - CORE_PEER_ADDRESSAUTODETECT=true
    - CORE_VM_ENDPOINT=unix:///var/run/docker.sock
    - CORE_LOGGING_LEVEL=DEBUG
  command: peer node start
```

这种单个 Peer 节点配置（运行 NOOPS 一致性插件）应能够满足许多开发/测试情境。NOOPS 并不会真正地提供一致性，从根本上来说它属于一个模拟一致性的空操作。举例来说，在直接开发和测试链代码时，该插件足以满足需求，但在链代码利用身份、访问控制、机密性和隐私相关的成员服务时，则另当别论。

通过 CA 运行

如果您希望充分确保安全性（认证和授权）、隐私和机密性，则需要运行 Fabric 的证书授权（CA)。相关信息请参阅 CA 设置说明。

启动附加验证 Peer 节点：

按照我们在上文中构建的模式，使用 vp1 作为第二个验证 Peer 节点的 ID。如果使用了 Docker Compose ,则可以简单地连接两个 Peer 节点。以下所示为 Vagrant 环境中存在两个 Peer 节点 vp0 和

vp1 时的 docker-compose.yml。

```
vp0:
  image: hyperledger/fabric-peer
  environment:
    - CORE_PEER_ADDRESSAUTODETECT=true
    - CORE_VM_ENDPOINT=http://172.17.0.1:2375
    - CORE_LOGGING_LEVEL=DEBUG
  command: peer node start
vp1：
  extends:
    service: vp0
  environment:
    - CORE_PEER_ID=vp1
    - CORE_PEER_DISCOVERY_ROOTNODE=vp0:7051
  links:
    - vp0
```

如果想要使用 Docker 命令行启动另一节点，我们需要获取第一个验证 Peer 节点的 IP 地址，以此作为新 Peer 节点的根节点。其地址会出现在第一个 Peer 节点的终端窗口上（例如 172.17.0.2），且应通过 CORE_PEER_DISCOVERY_ROOTNODE 环境变量进行传递。

```
docker run --rm -it -e CORE_VM_ENDPOINT=http://172.17.0.1:2375
-e CORE_PEER_ID=vp1 -e CORE_PEER_ADDRESSAUTODETECT=true -e
CORE_PEER_DISCOVERY_ROOTNODE=172.17.0.2:7051 hyperledger/fabric-peer peer node start
```

使用一致性插件

若要使用一致性插件，您需要进行一些特定的配置。举例来说，若要使用作为 Fabric 的一部分提供的 Practical Byzantine Fault Tolerant (PBFT) 一致性插件，则可执行以下配置：

1. 在 core.yaml 中，将 peer.validator.consensus 的值设置为 pbft。
2. 在 core.yaml 中，确保 peer.id 按顺序设置为 vpN，其中 N 是一个整数，从 0 开始，最大值是 N-1。举例来说，在存在 4 个校验节点的情况下，可将 peer.id 设置为 vp0、vp1、vp2、vp3。
3. 在 consensus/pbft/config.yaml 中，将 general.mode 的值设置为 batch，并将 general.N 的值设置为网络中验证 Peer 节点的个数，并将 general.batchsize 设置为每次批处理中所含事务的数量。
4. 在 consensus/pbft/config.yaml 中，您可以设置批处理周期（general.timeout.batch）、请求与执

行之间的允许延迟（general.timeout.request）、视图切换时间（general.timeout.viewchange）等可选项的时间值。

有关更多详情，请参阅 core.yaml 和 consensus/pbft/config.yaml。

所有此类设置可通过 CORE_PEER_VALIDATOR_CONSENSUS_PLUGIN=pbft 或 CORE_PBFT_GENERAL_MODE=batch 等命令行环境变量重写。

日志控制

有关 peer 及已部署链代码日志输出控制的信息，请参阅日志控制。
