用 Kubernetes 部署超级账本 Fabric 的区块链即服务

张海宁、陈家豪
VMware 中国研发中心

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

1. 概述

盼望着，盼望着，超级账本 Fabric 1.0 正式来了，社区用户为之欢呼雀跃：终于等到一个企业级区块链应用平台了。然而在激动过后，回归平静之时，人们却往往发现，搭建 Fabric 平台是个相当披荆斩棘的历程。不仅要具备密码学、分布式计算、共识算法等区块链理论基础，而且要熟悉容器、Golang / Node.js 这些企业用户不常用的工程技术，这常常是很多人把区块链放弃在起跑线的原因。降低使用门槛，提高易用性，将是今后一段时间内推广企业区块链应用的重要工作。

之前我们的文章描述过安装部署多节点 Fabric 集群的步骤，旨在说明 Fabric 基本的运作原理，因此部署过程是手 (cao) 动 (gen) 的安装方式。在实际的开发测试中，需要自动化部署来提高效率，本文介绍如何利用容器平台 Kubernetes（K8s）来自动部署 Fabric，实现区块链即服务 (Blockchain as a Service, BaaS)的原型。

需要指出的是，BaaS 目前多用于开发测试，即在同一个 BaaS 平台，部署多个区块链节点，每个节点代表不同组织机构。这样显然是中心化的部署方式，所以只能用于开发测试用途。真实环境的部署需要分布在网络中多个 BaaS 协同工作才能完成，这是另外一个尚待完善的工作。

我们选择把 Fabric 部署在 K8s 上有几个原因。首先 Fabric 的组件都经过容器封装好的，很方便部署在 K8s 这类容器平台上，并借助平台实现高可用、监控管理、自动化运维等目的。其次，Fabric 是分布式系统，根据应用的具体需求，集群的各个组件数会有不同，需要灵活地配置和调整。而 K8s 是面向微服务架构的容器平台，扩展方便，能够很好地满足 Fabric 这方面的要求。还有就是 K8s 具备了多租户的能力，可在同一个平台中运行多个互相隔离的 Fabric 实例，方便开发测试，比如一个实例作为开发用，另一个实例作为测试用。

在超级账本中有个子项目叫 Cello，其目的是提供 Hyperledger 的 BaaS。据了解，目前已经支持把 Fabric 部署在 Docker 和 Swarm 上，有关 K8s 的支持还在开发中。由

于 Fabric 的设计中没有考虑到 K8s 等平台的特点，因此把 Fabric 部署在 K8s 上还需要一些变通的处理方法，后文相关部分会提到 。

2. 整体架构

2.1 基础设施

1) 网络

Kubernetes 集群由多个节点组成，为使得集群上的容器正常通信，需要创建一个 overlay 网络，并把集群上的容器都连接到这个网络上。

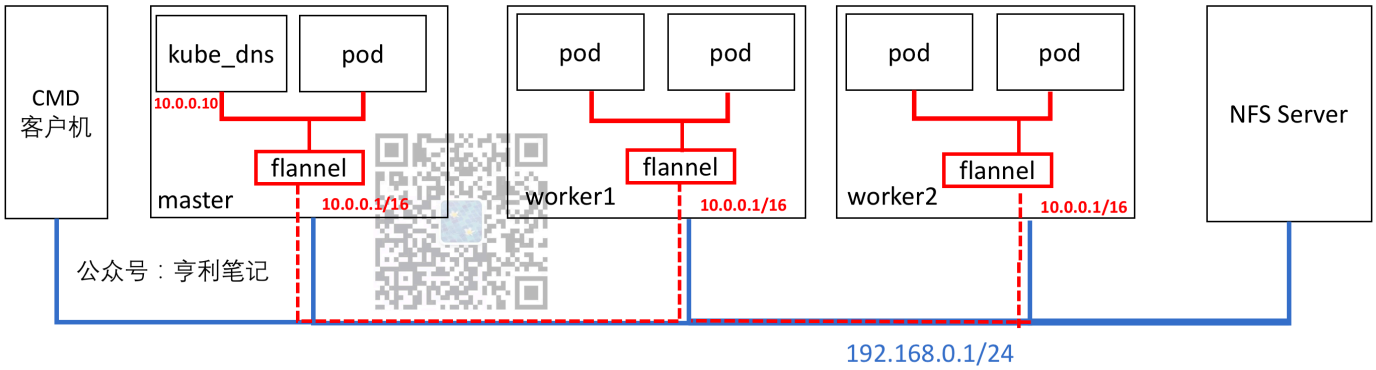


图 2- 1

如图 2-1 所示，宿主机网络由蓝色线标记，节点有 cmd 客户机, Kubernetes 的 master 和 worker，还有 NFS 服务器。其中，cmd 客户机作为操作 K8S 和 Fabric 集群的命令行客户机。NFS 服务器在各个节点间用于共享 Fabric 和 K8s 的各种配置文件，也可以用其他 K8s 支持的共享存储代替。

通过 Kubernetes 的 cluster add-ons 可以很方便地创建 overlay 网络，如 Flannel。如图 2-1 的红色线所示(为说明 Flannel 作用省去部分细节)，Kubernetes 把所有 pod 加入到 Flannel 网络中，因此 pod 中的容器可以相互通信。Flannel 网络的地址段可以在 add-on 的配置文件中指定，同时 kube_dns 的 IP 地址也可以通过配置修改，但该 IP 地址必须处于指定地址段中。如图 2-1 中 Flannel 的网络为 10.0.0.1/16，kube_dns 的 IP 地址为 10.0.0.10。

在 Fabric 设计中， chaincode 目前是以 Docker 容器的方式运行在 peer 容器所在的宿主机上，peer 容器需要调用 Docker 引擎的接口来构建和创建 chaincode 容器，调用接口是通过这个连接：

unix:///var/run/docker.sock

这种方式其实是有安全隐患的，这里不作过多讨论。正确的姿势应该是调用 chaincode 专用的运行环境，如新起一个 Docker Host，用 TCP 接口远程调用。

通过 docker.sock 创建的容器脱离在 Kubernetes 的体系之外，虽然它仍在 Flannel 网络上，但却无法获得 peer 节点的 IP 地址。这是因为创建该容器的 Docker 引擎使用宿主机默认的 DNS 解析来 peer 的域名，所以无法找到。

为了解决解析域名的问题， 需要在每个 worker 的 DOCKER_OPTS 中加入相关参数，以图 2-1 为例，kube_dns 的 IP 为 10.0.0.10，宿主机网络 DNS 的 IP 地址假设为 192.168.0.1，为使得 chaincode 的容器可以解析到 peer 节点，修改步骤如下：

编辑 /etc/default/docker，在 DOCKER_OPTS 中添加以下参数，设置 Kubernetes 使用的 DNS （很重要！）

1. ：

```
"--dns=10.0.0.10 --dns=192.168.0.1  --dns-search \
 default.svc.cluster.local --dns-search svc.cluster.local \
--dns-opt ndots:2 --dns-opt timeout:2 --dns-opt attempts:2 "
```

2. 运行以下命令重启 Docker (注: 不同的 Linux 环境中命令可能会有不同)：

```
systemctl daemon-reload
systemctl restart docker
systemctl restart docker.service
```

2) 共享存储

K8s 和 Fabric 集群需要较多的配置文件，为方便管理，可通过 NFS 服务器来统一储存这些文件，如图 2-1 所示。

cmd 客户机可通过 cryptogen 工具生成 crypto-config 目录，该用于储存 Fabric 集群中节点的配置文件，如 peer0.org1 所用到的 msp 存放在以下目录：

crypto-config/PeerOrganization/org1/peers/peer0/msp

cmd 客户机只需要把 NFS 的共享目录挂载到本地/opt/share/，再把 crypto-config 写到该目录，即可让各个节点访问到配置文件。

在 Kubernetes 中，通过 PV 和 PVC 来把 NFS 上的文件挂载到容器中，除了创建相应的 PV 和 PVC 外，还需在节点的配置文件中把正确的路径挂载进去。若 NFS 服务器的共享目录为/opt/share， 创建 PV 时可指定 peer.org1 的挂载点为：

/opt/share/crypto-config/PeerOrganization/org1/

然后创建与该 PV 相对应的 PVC，这样在节点的配置文件中就可以使用 PVC 来挂载这个目录。节点需要根据自己的 ID 在挂载点后面加上相应的路径来保证挂载的配置文件无误，如 peer0.org1 应在路径后加上 peers/peer0/msp，则其挂载目录的完整路径如下：

/opt/share/crypto-config/PeerOrganization/org1/peers/peer0/msp

2.2 组件划分

在 Kubernetes 中， Namespace 是一个很重要的概念，它用于划分不同的虚拟集群，而 Fabric 中 organization 基于证书签发机构划分，把 K8S 的 namespace 与 Fabric 的 organization 对应，既使得 organization 之间相互独立，又充分利用了 K8S 的 DNS 服务，各个 organization 可以通过域名区分。采用 namespace 分隔各个组织的组件，还可实现网络策略(network policy)来隔离不同组织，实现多租户的能力(本文未涉及)。

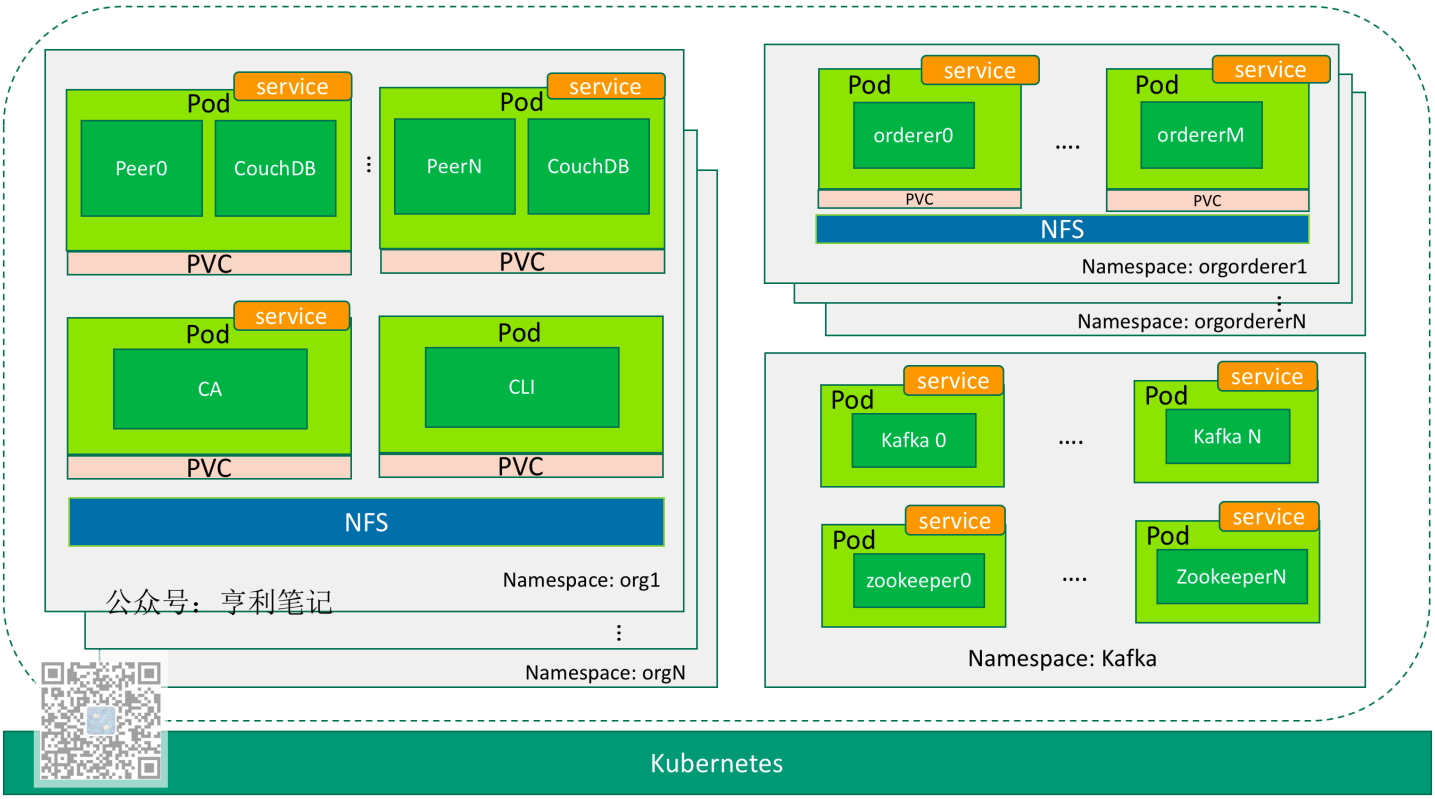


图 2- 2

如图 2-2 所示，假设 Fabric 网络中有多个 peer organization 和 orderer organization，下面阐述如何在 Kubernetes 进行划分和对应：

a. 若第 N 个 Fabric 的 peer organization 的域名为 orgN，则其在 Kubernetes 上对应的 namespace 设置为 orgN，在该 namespace 下有多个 pod，pod 的类型如下：
   1) Peer Pod：包括 Fabric peer，couchDB（可选），代表每个组织的 peer 节点。
   2) CA Server Pod： Fabric CA Server。
   3) CLI Pod：（可选）提供命令行工具的环境，用于操作本组织的节点、channel 或 chaincode。

b. 若第 N 个 Fabric 的 orderer organization 的域名为 orgordererN，则其在 Kubernetes 上对应的 namespace 为 orgordererN，该 namespace 下只有一种 Pod，它用于运行 orderer 节点。

c. Kafka namespace 与 Fabric 的 organization 并无关系，它只用来运行和管理 Zookeeper 和 Kafka 容器，实现共识算法。

2.3 Pod 之间通信

Kubernetes 中的每个 Pod 都有独立的 IP 地址，然而在各个 Pod 之间直接通过 IP:port 的方式来通信会带来很多麻烦，因此有必要给每一个 Pod 绑定一个的 service，以便用 service 名称来访问。

service 的命名方式应当遵循以下原则，彰显与其绑定的 Pod 信息：

1) service 与 pod 的 namespace 应当一致。
2) service 的 name 应与 Pod 中容器的 id 一致。

例如，Fabric 中属于 org1 的 peer0 节点，在 K8S 中用 namespace 为 org1、名字为 peer0 的 Pod 来运行，与该 Pod 绑定的 service 全称应为 peer0.org1。其中 peer0 为 service 的名称，org1 为 service 的 namespace。这样的映射关系已经和主机域名很接近了。

3. 源码的说明与使用

3.1 环境准备

假定 K8s 平台已经成功部署，并且在各个 worker 节点已经预先下载相应的 Fabric v1.0.0 镜像，如表 3.1。（预先下载镜像是由于国内网速较慢，在一台机器预先下载后，可用 Docker 命令导出并导入其他机器。）

IMAGE	TAG	ID
hyperledger/fabric-tools	x86_64-1.0.0	0403fd1c72c7
hyperledger/fabric-orderer	x86_64-1.0.0	e317ca5638ba
hyperledger/fabric-peer	x86_64-1.0.0	6830dcd7b9b5
hyperledger/fabric-ccenv	x86_64-1.0.0	7182c260a5ca
hyperledger/fabric-ca	x86_64-1.0.0	a15c59ecda5b
hyperledger/fabric-baseimage	x86_64-0.3.1	9f2e9ec7c527
hyperledger/fabric-baseos	x86_64-0.3.1	4b0cab202084

表 3- 1

下载本文涉及代码的目录结构及其作用如下：

- Fabric-on-k8s
  - README.md
  - setupCluster
    - generateALL.sh // 负责生成 K8S 部署文件
    - transform // 用于启动部署文件
    - templates // 存放模板
    - cluster-config.yaml // 用于配置 Fabric 集群
    - configtx.yaml // 用于配置 channel

3.2 配置文件说明

在规划 Fabric 集群部署时，要按实际需求，编辑以下两个 Fabric 集群的定义文件：

a. cluster-config.yaml

cryptogen 工具根据 cluster-config.yaml 来生成 Fabric 成员的证书，一个简单的例子如下:

```
OrdererOrgs:
 - Name: Orderer
   Domain: orgorderer1
   Template:
     Count: 1

PeerOrgs:
 - Name: Org1
   Domain: org1
   Template:
     Count: 2

 - Name: Org2
   Domain: org2
   Template:
     Count: 2
```

其中 OrdererOrgs 和 PeerOrgs 关键字区分 organization 的类型，两种组织的内部结构如下：

1) OrdererOrgs 中定义了一个名字为 Orderer，域名为 orgorderer1 的 org，并且它指定 template 中 count 的数值为 1，则在该 org 下只有一个 orderer，其 id 为 orderer0。

2) PeerOrgs 中定义了两个 org，分别为 Org1 和 Org2，对应的域名为 org1、org2，与 orderer 类似，每个 org 生成了两个 peer，虽然 org1 中 peer0 和 org2 中 peer0 的 ID 重复，但是他不属于同一个 org，通过域名很容易就能区分出它们。

需要注意的是，由于 K8S 中的 namespace 不支持 '.' 和大写字母，因此各个组织的域名不能包含这些字符。

更多关于 cluster-config.yaml 的配置方式，请读者自行参考 Fabric 源码中的关于 cryptogen 的描述(fabric/common/tools/cryptogen/main.go.)

以上定义的 cluster-config.yaml，cryptogen 工具会生成 crypto-config 目录，该目录的结构如下：

```
crypto-config
  --- ordererOrganizations
     --- orgorderer1
        --- msp
        ---  ca
         --- tlsca
        --- users
        --- orderers
           --- orderer0.orgorderer1
              --- msp
              --- tls

  --- peerOrganizations
     --- org1
        --- msp
        ---  ca
         --- tlsca
        --- users
        --- peers
            --- peer0.org1
               --- msp
               --- tls
            --- peer1.org1
               --- msp
               --- tls
     --- org2
        --- msp
        ---  ca
        --- tlsca
```

```
--- users
--- peers
    --- peer0.org2
        --- msp
        --- tls
    --- peer1.org2
        --- msp
        --- tls
```

可以看出，每个 org 都包含了 msp、ca、tlsca 和 users 目录，然后根据 org 类型的不同，还分别有 peers 和 orderers 目录，里面存放着 org 中每个成员的 msp 和 tls 文件。

b. configtx.yaml

configtxgen 工具根据该文件生成 Orderer 初始化的时候要使用的 genesis.block，获知 organization 的各种信息，因此，用户要根据 cluster-config.yaml 中关于 organization 的定义来修改 configtx.yaml 以生成合适的 genesis.block。例如，用户在 cluster-config.yaml 中增加了一个 Org3，并且要创建一个包含 Org1，Org2，Org3 的集群，则应该通过以下两步修改 configtx.yaml：

1. 在 profile 中增加 Org3，如图 3-1：

```
Profiles:

    TwoOrgsOrdererGenesis:
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *OrdererOrg
        Consortiums:
            SampleConsortium:
                Organizations:
                    - *Org1
                    - *Org2
                    - *Org3
    TwoOrgsChannel:
        Consortium: SampleConsortium
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *Org1
                - *Org2
                - *Org3
```

图 3- 1

2. 在 Organization 中增加 Org3 的 MSPDir，如图 3-2:

```
- &Org3
    # DefaultOrg defines the organization which is used in the sampleconfig
    # of the fabric.git development environment
    Name: Org3MSP

    # ID to load the MSP definition as
    ID: Org3MSP

    MSPDir: crypto-config/peerOrganizations/org3/msp

    AnchorPeers:
        # AnchorPeers defines the location of peers which can be used
        # for cross org gossip communication.  Note, this value is only
        # encoded in the genesis block in the Application section context
        - Host: peer0.org3
          Port: 7051
```


图 3- 2

注意的是每个 organization 中的 MSPDir 的值必须是这种形式：crypto-config/{OrgType}/{OrgName}/mps

3.3 模板文件

在 Kubernetes 中部署 Fabric 时，需要为每个节点编写相应的配置文件。由于节点数可能很多，这是既复杂又易错的重复劳动。为提高效率，可通过模板自动生成配置文件。本文使用了 5 个模板文件，可用脚本替换其中的变量，均在笔者给出示例代码中的 templates 目录中，这些模板的作用如下：

a. fabric_1_0_tmeplate_namespace.yaml

定义 Fabric 集群在 K8s 中的 namespace，它对应着 organization 的域名。为了在多节点共享证书等文件，使用了 NFS 服务器作为存储。在 K8s 中通过相应的 PV 和 PVC，namespace 下的 Pod 可以通过 PVC 来获取与之相应的文件。

b. fabric_1_0_template_cli.yaml

CLI pod 模板，每个 organization 中都配备了一个 CLI pod，目的是提供命令行界面，可统一管理组织内的所有 peer，其中包括 channel 的创建，chaincode 的安装等。CLI Pod 的 CORE_PEER_ADDRESS 环境变量默认值为 org 中的第一个 peer，可以通过修改该环境变量来连接不同的 peer。

yaml 文件中的 command 是为了防止 CLI Pod 自动退出，CLI 的默认工作目录为/opt/gopath/src/github.com/hyperledger/fabric/peer。由于该目录下的 channel-artifacts 挂载了 NFS 上/opt/share/channel-artifacts，因此把创建 channel 时返回的 xxx.block 文件放在该目录下供所有 CLI Pod 共享。

c. fabric_1_0_template_ca.yaml

Fabric 的 CA 服务的 Pod 定义模板，用于 organization 中的证书管理，其 yaml 文件除了定义 deployment 外，还定义了 service。service 通过 selector 与 deployment 绑定，其中 deployment 中的 label 是 selector 与其绑定的根据。

d. fabcric_1_0_template_orderer.yaml

Orderer 的 pod 定义模板，需要注意的是，cryptogen 并不会生成 genesis.block，然而缺少该文件时，orderer 会启动失败，因此在启动 orderer 之前需要预先生成 genesis.block，并将其放在相应的 org 目录下。

e. fabric_1_0_template_peer.yaml

每个 peer pod 的定义模板。在该 yaml 中分别定义了 peer 和 couchDB 两个 container。在实例化 chaincode (cc) 时，peer 需要连接 Docker 引擎来创建 cc 容器，因此要把 worker 宿主机的 var/run/docker.sock 映射到 peer 容器内部（参见图 2-1）。

3.4 源码使用

以下操作都在图 2-1 的 cmd 客户机上进行，NFS 的共享目录为/opt/share，该共享目录的拥有者:用户组设为 nobody:nogroup。

a．生成启动文件

步骤：

1. 把 NFS 的/opt/share 目录挂载到 host 的/opt/share 。

2.下载本文配套源码并进入 Fabric-on-K8S/ 目录，通过以下命令下载 Fabric 的 cryptogen 等工具：

：

$ curl https://nexus.hyperledger.org/content/repositories/releases/org/hyperledger/fabric/hyperledger-fabric/linux-amd64-1.0.0/hyperledger-fabric-linux-amd64-1.0.0.tar.gz | tar xz

下载完毕后会在当前目录生成一个 bin 目录，该目录包含 cryptogen 和 configtx 等文件。

3. 更改 templates/fabric_1_0_template_pod_cli 的 NFS 地址，如图 3-3 所示。

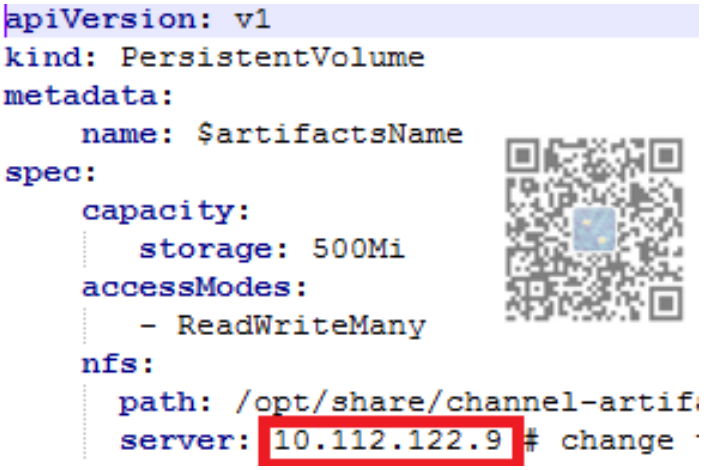


图 3- 3

4. 更改 templates/fabric_1_0_template_pod_namespace 的 NFS 地址，如图 3-4。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: $pvName
spec:
  capacity:
    storage: 500Mi
  accessModes:
    - ReadWriteMany
  nfs:
    path: $path
    server: 10.112.122.9  #change
```

图 3- 4

5. 依照 3.2 的说明配置 cluster-config.yaml 和 configtx.yaml 。

6. 通过以下命令生成启动所需要的文件:

$ sudo bash generateAll.sh

运行 generateAll.sh 脚本时，除了调用 cryptogen 生成 crypto-config 目录之外，还在目录中的各个 organization 子目录下插入相应的 K8S 配置文件。以 org1 为例，其目录下会有几个 yaml 文件用于启动：

```
crypto-config
 --- peerOrganizations
    --- org1
            --- org1-ca.yaml
            --- org1-cli.yaml
            --- org1-namespace.yaml
            --- msp
            ---  ca
            --- tlsca
            --- users
            --- peers
                --- peer0.org1
                    --- peer0.org1.yaml
                    --- msp
                    --- tls
                --- peer1.org1
                    --- peer1.org1.yaml
                    --- msp
                    --- tls
```

b. 运行启动脚本

通过以下命令启动 Fabric 集群 (需要安装 PyYAML-3.5):

```
$ python3.5 transform/run.py
```

对每个 Fabric 的 PeerOrganization，启动脚本的工作流程如下：

- 在 Kubernetes 中创建 org 的 namespace；
- 创建 org 的 ca pod；
- 创建 org 的 CLI pod；
- 遍历 orgM/peers 的子目录找出相应的 yaml 文件，并启动所有 peer。

c. 查看 cluster 状态

创建完成后，查看各个 pod 的状态，若都显示为 running 则说明所有部件工作正常，命令如下，结果如图 3-5：

```
$ kubectl get pods –all-namespaces
```

```
org1          ca-3347986348-9jvht                1/1       Running   0          1h        172.1.73.5     node1
org1          cli-1569835662-01c5z               1/1       Running   0          1h        172.1.12.5     node2
org1          peer0-org1-1343141255-h8kgk        2/2       Running   0          1h        172.1.86.6     node3
org1          peer1-org1-2603922830-35q6c        2/2       Running   0          1h        172.1.12.6     node2
org2          ca-2708682628-qpz64                1/1       Running   0          1h        172.1.73.3     node1
org2          cli-2586364563-vclmr               1/1       Running   0          1h        172.1.42.4     node4
org2          peer0-org2-3143546256-9prph        2/2       Running   0          1h        172.1.73.4     node1
org2          peer1-org2-110343575-06pvc         2/2       Running   0          1h        172.1.42.5     node4
org3          ca-349255610-628k1                 1/1       Running   0          1h        172.1.12.3     node2
org3          cli-3602893464-7f6g1               1/1       Running   0          1h        172.1.73.2     node1
org3          peer0-org3-649967001-0v813         2/2       Running   0          1h        172.1.12.4     node2
org3          peer1-org3-1910748576-1j1bv        2/2       Running   0          1h        172.1.86.5     node3
org3          peer2-org3-3171530151-6whd0        2/2       Running   0          1h        172.1.42.3     node4
orgorderer    orderer0-orgorderer-73543963-plc1z 1/1       Running   0          1h        172.1.86.4     node3
```

图 3- 5

4. 测试 Fabric 集群

假设已经成功启动 3.2.a 中定义的 Fabric 集群，下面通过运行测试 chaincode 来判断 Fabric 集群是否如预期般工作。

首先创建和加入 channel， 使用 configtx 工具来生成与 channel 相关的文件：

[1] 进入 CMD 客户机的 Fabric-on-K8S/setupCluster/目录:

$ cd Fabric-on-K8S/setupCluster/

[2] 创建 channel 的 channel.tx 文件，该 channel 的 ID 为 mychannel:

$ ../bin/configtxgen -profile TwoOrgsChannel -outputCreateChannelTx \
./channel-artifacts/channel.tx -channelID mychannel

[3] 创建 channel 的升级文件，该文件用于更新 mychannel 中 Org1 的 anchor peer:
$ ../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate \
./channel-artifacts/Org1MSPanchors.tx -channelID mychannel -asOrg Org1MSP

[4] 创建 channel 的升级文件，该文件用于更新 mychannel 中 Org2 的 anchor peer:
$ ../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate \
./channel-artifacts/Org2MSPanchors.tx -channelID mychannel -asOrg Org2MSP

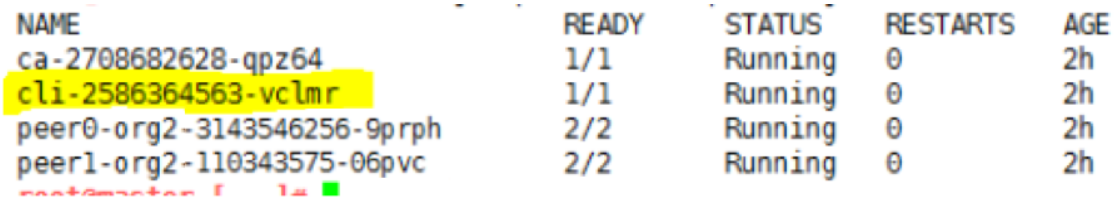
[5] 由于每个 Org 的 CLI Pod 需要用到以上步骤创建的文件，可以通过 NFS 来跟 CLI Pod 共享这些文件:

$ sudo cp -r ./channel-artifacts /opt/share/

完成以上工作后，就可以通过各组织的 CLI Pod 来测试集群是否正常运行。

通过以下操作进入任意 org 的 CLI Pod 内部，以 org1 为例：

1. 查看 namespace 为 org1 下的所有 Pod：
   $ kubectl get pods –namespaces org1



NAME	READY	STATUS	RESTARTS	AGE
ca-2708682628-qpz64	1/1	Running	0	2h
cli-2586364563-vclmr	1/1	Running	0	2h
peer0-org2-3143546256-9prph	2/2	Running	0	2h
peer1-org2-110343575-06pvc	2/2	Running	0	2h

图 4- 1

   如图 4-1 所示，org1 的 CLI Pod 为 cli-2586364563-vclmr。

2. 进入 cli-2586364563-vclmr Pod：

```
$ kubectl exec -it cli-2586364563-vclmr bash --namespace=org1
```

进入 CLI Pod 后， 可以执行以下命令以测试 Fabric 集群：

a. 创建 channel

```
$ peer channel create -o orderer0.orgorderer1:7050 \
-c mychannel -f ./channel-artifacts/channel.tx
```

b. 拷贝 mychannel.block 到 channel-artifacts 目录：

```
$ cp mychannel.block ./channel-artifacts
```

c. 加入 mychannel

```
$ peer channel join -b ./channel-artifacts/mychannel.block
```

d. 更新 anchor peer，每个 org 只需执行一次

```
$ peer channel update -o orderer0.orgorderer1:7050 \
-c mychannel -f ./channel-artifacts/Org1MSPanchors.tx
```

e. 安装 chaincode。

请读者下载 Fabric 的 chaincode_example02 目录并将其放置在 CMD 客户机的 /opt/share/channel-artifacts 目录下：

```
$ peer chaincode install -n mycc -v 1.0 -p github.com/hyperledger/fabric/peer/channel-artifacts/chaincode_example02
```

f. 实例化 chaincode

```
$ peer chaincode instantiate -o orderer0.orgorderer1:7050 \
-C mychannel -n mycc -v 1.0 -c '{"Args":["init","a","100","b","200"]}' \
-P "OR  ('Org1MSP.member','Org2MSP.member')"
```

通过以上命令实例化 mycc 后，读者可以自行切换到其他 org 的 CLI Pod 上通过加入 channel 等步骤，验证账本是否同步。

4.1 外部调用

在配置文件中 ca、peer 和 orderer 的 service 类型定义为 NodePort，这样做的目的是为了让用户在 K8S 外也能访问到 Fabric 中的各个成员，端口映射规则如下(以下出现 N 和 M 的范围分别为 N>=1,M>=0):

1. orgN 端口范围是 30000+(N-1)*100 ~ 30000+(N)*100-1，也就是说每一个 org 最多能分配到 100 个端口号，如 org1 的端口范围是 30000 到 30099。

2. CA 的 7054 的映射关系如下：
   ca.orgN:7054 -> worker:30000+(N-1)*100

3. 由于每个 peer 需要映射 7051 和 7052 两个端口，因此 org 中 peerM 的端口映射关系如下:

   peerM.orgN:7051 -> worker:30000+(N-1)*100 + 2 * M + 1
   peerM.orgN:7052 -> worker:30000+(N-1)*100 + 2 * M + 2

4. ordererN 的映射关系为：ordererN:7050 -> worker:23700+N

若 worker1 的 IP 地址为 192.168.0.7，它运行 peer0.org1，则 Kubernetes 外的用户需要通过 192.168.0.7:30001 地址才能访问 peer0.org1。

4.2 删除集群

当需要删除集群的时候，可以通过 transform 目录下的 delete.py 脚本来清理环境，该脚本会遍历 crypto-config 目录，找出所有的 yaml 文件，并通过 “kuberclt delete -f xxx.yaml” 的方式将资源逐个删除。

5. 小结

本文阐述了 K8S 与 Fabric 结合的重要性，并给出 Fabric 与 K8S 结合的思路与框架，然后结合给出的脚本工具来解析快捷部署的实现方式，最后是测试部署的集群是否正常工作。本文介绍的部署方法，是基于 Kubernetes 容器云平台实现 BaaS 的基础步骤。在此之上，可以增加更多的区块链层管理功能，图形化运维界面，使得开发人员投入更多的精力到应用的业务逻辑上。

本文涉及的代码可以在此下载:
https://github.com/hainingzhang/articles/tree/master/fabric_on_kubernetes

扫码关注公众号：**亨利笔记**，获取更多区块链和云计算等方面的科技文章。

https://github.com/hainingzhang/articles

VMware 公司招聘区块链实习生和外包开发工程师
VMware 公司为超级账本 Hyperledger 项目创始成员，中国研发中心现在招募区块链方向实习生和外包开发工程师，地点：北京知春里。

外包软件开发工程师：1-5 年软件开发经验，熟悉 Java 或 Go 开发语言，熟悉分布式系统、Docker，了解区块链技术优先。

实习生：要求在读研究生，计算机相关专业，懂 Java 或 Go 开发语言，能够实习 3 个月以上，熟悉区块链技术优先。欢迎自荐或推荐。

有兴趣者发简历到: BaaS@vmware.com