**一、预先准备环境**

**1. 准备服务器**

这里准备了三台ubuntu虚拟机，每台2cpu和8G内存，配置好root账户，并安装好了docker，后续的所有操作都是使用root账户。虚拟机具体信息如下表：

| **系统类型** | **IP地址** | **节点角色** | **Hostname** |
| --- | --- | --- | --- |
| ubuntu16.04 | 192.168.0.174  47.92.249.50 | master | i-master |
| ubuntu16.04 | 192.168.1.213  39.98.85.32 | Worker01 | i-worker1 |
| ubuntu16.04 | 192.168.1.214 | Worker02 | i-worker2 |

Tips: scp /root/.ssh/id\_rsa.pub 192.168.1.213:~/master\_rsa.pub

修改/etc/hosts 以及 ~/.ssh/authorized\_keys

**2. 安装docker（所有节点）**

一般情况使用下面的方法安装即可

**2.1 安装apt的https支持包并添加gpg秘钥**

$ apt-get update

$ apt-get install \

apt-transport-https \

ca-certificates \

curl \

software-properties-common

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

**2.2 更新apt-get源**

$ add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

$ apt-get update

**2.4 安装docker-ce**

* 安装最新的稳定版

$ apt-get install -y docker-ce

* 安装指定版本

#获取版本列表

$ apt-cache madison docker-ce

#指定版本安装(比如版本是17.09.1~ce-0~ubuntu)

$ apt-get install -y docker-ce=17.09.1~ce-0~ubuntu

* 接受所有ip的数据包转发

$ vi /lib/systemd/system/docker.service

#找到ExecStart=xxx，在这行上面加入一行，内容如下：(k8s的网络需要)

ExecStartPost=/sbin/iptables -I FORWARD -s 0.0.0.0/0 -j ACCEPT

* 启动服务

$ systemctl daemon-reload

$ systemctl restart docker

遇到问题可以参考：[官方教程](https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/)

**3. 系统设置（所有节点）**

**3.1 关闭、禁用防火墙(让所有机器之间都可以通过任意端口建立连接)**

$ ufw disable

#查看状态

$ ufw status

**3.2 设置系统参数 - 允许路由转发，不对bridge的数据进行处理**

#写入配置文件

$ cat <<EOF > /etc/sysctl.d/k8s.conf

net.ipv4.ip\_forward = 1

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

#生效配置文件

$ sysctl -p /etc/sysctl.d/k8s.conf

**3.3 配置host文件**

#配置host，使每个Node都可以通过名字解析到ip地址

$ vim /etc/hosts

#加入如下片段(ip地址和servername替换成自己的)

192.168.0.174 master

192.168.1.212 worker1

192.168.1.213 worker2

**4. 准备二进制文件（所有节点）**

kubernetes的安装有几种方式，不管是kube-admin还是社区贡献的部署方案都离不开这几种方式：

* **使用现成的二进制文件**

直接从官方或其他第三方下载，就是kubernetes各个组件的可执行文件。拿来就可以直接运行了。不管是centos，ubuntu还是其他的linux发行版本，只要gcc编译环境没有太大的区别就可以直接运行的。使用较新的系统一般不会有什么跨平台的问题。

* **使用源码编译安装**

编译结果也是各个组件的二进制文件，所以如果能直接下载到需要的二进制文件基本没有什么编译的必要性了。

* **使用镜像的方式运行**

同样一个功能使用二进制文件提供的服务，也可以选择使用镜像的方式。就像nginx，像mysql，我们可以使用安装版，搞一个可执行文件运行起来，也可以使用它们的镜像运行起来，提供同样的服务。kubernetes也是一样的道理，二进制文件提供的服务镜像也一样可以提供。

从上面的三种方式中其实使用镜像是比较优雅的方案，容器的好处自然不用多说。但从初学者的角度来说容器的方案会显得有些复杂，不那么纯粹，会有很多容器的配置文件以及关于类似二进制文件提供的服务如何在容器中提供的问题，容易跑偏。 所以我们这里使用二进制的方式来部署。二进制文件已经这里备好，大家可以打包下载，把下载好的文件放到每个节点上，放在哪个目录随你喜欢，**放好后最好设置一下环境变量$PATH**，方便后面可以直接使用命令。(科学上网的同学也可以自己去官网找找)  
####[下载地址（kubernetes 1.9.0版本）](https://pan.baidu.com/s/1bMnqWY)

**5. 准备配置文件（所有节点）**

上一步我们下载了kubernetes各个组件的二进制文件，这些可执行文件的运行也是需要添加很多参数的，包括有的还会依赖一些配置文件。现在我们就把运行它们需要的参数和配置文件都准备好。

**5.1 下载配置文件**

#到home目录下载项目

$ cd

$ git clone https://github.com/liuyi01/kubernetes-starter.git

#看看git内容

$ cd ~/kubernetes-starter && ls

**5.2 文件说明**

* **gen-config.sh**

shell脚本，用来根据每个同学自己的集群环境(ip，hostname等)，根据下面的模板，生成适合大家各自环境的配置文件。生成的文件会放到target文件夹下。

* **kubernetes-simple**

简易版kubernetes配置模板（剥离了认证授权）。 适合刚接触kubernetes的同学，首先会让大家在和kubernetes初次见面不会印象太差（太复杂啦~~），再有就是让大家更容易抓住kubernetes的核心部分，把注意力集中到核心组件及组件的联系，从整体上把握kubernetes的运行机制。

* **kubernetes-with-ca**

在simple基础上增加认证授权部分。大家可以自行对比生成的配置文件，看看跟simple版的差异，更容易理解认证授权的（认证授权也是kubernetes学习曲线较高的重要原因）

* **service-config**

这个先不用关注，它是我们曾经开发的那些微服务配置。 等我们熟悉了kubernetes后，实践用的，通过这些配置，把我们的微服务都运行到kubernetes集群中。

**5.3 生成配置**

这里会根据大家各自的环境生成kubernetes部署过程需要的配置文件。 在每个节点上都生成一遍，把所有配置都生成好，后面会根据节点类型去使用相关的配置。

#cd到之前下载的git代码目录

$ cd ~/kubernetes-starter

#编辑属性配置（根据文件注释中的说明填写好每个key-value）

$ vi config.properties

#生成配置文件，确保执行过程没有异常信息

$ ./gen-config.sh simple

#查看生成的配置文件，确保脚本执行成功

$ find target/ -type f

target/all-node/kube-calico.service

target/master-node/kube-controller-manager.service

target/master-node/kube-apiserver.service

target/master-node/etcd.service

target/master-node/kube-scheduler.service

target/worker-node/kube-proxy.kubeconfig

target/worker-node/kubelet.service

target/worker-node/10-calico.conf

target/worker-node/kubelet.kubeconfig

target/worker-node/kube-proxy.service

target/services/kube-dns.yaml

**执行gen-config.sh常见问题：**

1. gen-config.sh: 3: gen-config.sh: Syntax error: "(" unexpected

* bash版本过低，运行：bash -version查看版本，如果小于4需要升级
* 不要使用 sh gen-config.sh的方式运行（sh和bash可能不一样哦）

1. config.properties文件填写错误，需要重新生成 再执行一次./gen-config.sh simple即可，不需要手动删除target

# 二、基础集群部署 - kubernetes-simple

## 1. 部署ETCD（主节点）

#### 1.1 简介

  kubernetes需要存储很多东西，像它本身的节点信息，组件信息，还有通过kubernetes运行的pod，deployment，service等等。都需要持久化。etcd就是它的数据中心。生产环境中为了保证数据中心的高可用和数据的一致性，一般会部署最少三个节点。我们这里以学习为主就只在主节点部署一个实例。

如果你的环境已经有了etcd服务(不管是单点还是集群)，可以忽略这一步。前提是你在生成配置的时候填写了自己的etcd endpoint哦~

#### 1.2 部署

**etcd的二进制文件和服务的配置我们都已经准备好，现在的目的就是把它做成系统服务并启动。**

#把服务配置文件copy到系统服务目录

$ cp ~/kubernetes-starter/target/master-node/etcd.service /lib/systemd/system/

#enable服务

$ systemctl enable etcd.service

#创建工作目录(保存数据的地方)

$ mkdir -p /var/lib/etcd

# 启动服务

$ service etcd start

# 查看服务日志，看是否有错误信息，确保服务正常

$ journalctl -f -u etcd.service

## 2. 部署APIServer（主节点）

#### 2.1 简介

kube-apiserver是Kubernetes最重要的核心组件之一，主要提供以下的功能

* 提供集群管理的REST API接口，包括认证授权（我们现在没有用到）数据校验以及集群状态变更等
* 提供其他模块之间的数据交互和通信的枢纽（其他模块通过API Server查询或修改数据，只有API Server才直接操作etcd）

生产环境为了保证apiserver的高可用一般会部署2+个节点，在上层做一个lb做负载均衡，比如haproxy。由于单节点和多节点在apiserver这一层说来没什么区别，所以我们学习部署一个节点就足够啦

#### 2.2 部署

APIServer的部署方式也是通过系统服务。部署流程跟etcd完全一样，不再注释

$ cp target/master-node/kube-apiserver.service /lib/systemd/system/

$ systemctl enable kube-apiserver.service

$ service kube-apiserver start

$ journalctl -f -u kube-apiserver

#### 2.3 重点配置说明

[Unit]  
Description=Kubernetes API Server  
...  
[Service]  
#可执行文件的位置  
ExecStart=/home/michael/bin/kube-apiserver \  
#非安全端口(8080)绑定的监听地址 这里表示监听所有地址  
--insecure-bind-address=0.0.0.0 \  
#不使用https  
--kubelet-https=false \  
#kubernetes集群的虚拟ip的地址范围  
--service-cluster-ip-range=10.68.0.0/16 \  
#service的nodeport的端口范围限制  
--service-node-port-range=20000-40000 \  
#很多地方都需要和etcd打交道，也是唯一可以直接操作etcd的模块  
--etcd-servers=[http://192.168.1.102:2379](http://192.168.1.102:2379/) \  
...

## 3. 部署ControllerManager（主节点）

#### 3.1 简介

Controller Manager由kube-controller-manager和cloud-controller-manager组成，是Kubernetes的大脑，它通过apiserver监控整个集群的状态，并确保集群处于预期的工作状态。 kube-controller-manager由一系列的控制器组成，像Replication Controller控制副本，Node Controller节点控制，Deployment Controller管理deployment等等 cloud-controller-manager在Kubernetes启用Cloud Provider的时候才需要，用来配合云服务提供商的控制

controller-manager、scheduler和apiserver 三者的功能紧密相关，一般运行在同一个机器上，我们可以把它们当做一个整体来看，所以保证了apiserver的高可用即是保证了三个模块的高可用。也可以同时启动多个controller-manager进程，但只有一个会被选举为leader提供服务。

#### 3.2 部署

**通过系统服务方式部署**

$ cp target/master-node/kube-controller-manager.service /lib/systemd/system/

$ systemctl enable kube-controller-manager.service

$ service kube-controller-manager start

$ journalctl -f -u kube-controller-manager

#### 3.3 重点配置说明

[Unit]  
Description=Kubernetes Controller Manager  
...  
[Service]  
ExecStart=/home/michael/bin/kube-controller-manager \  
#对外服务的监听地址，这里表示只有本机的程序可以访问它  
--address=127.0.0.1 \  
#apiserver的url  
--master=[http://127.0.0.1:8080](http://127.0.0.1:8080/) \  
#服务虚拟ip范围，同apiserver的配置  
--service-cluster-ip-range=10.68.0.0/16 \  
#pod的ip地址范围  
--cluster-cidr=172.20.0.0/16 \  
#下面两个表示不使用证书，用空值覆盖默认值  
--cluster-signing-cert-file= \  
--cluster-signing-key-file= \  
...

## 4. 部署Scheduler（主节点）

#### 4.1 简介

kube-scheduler负责分配调度Pod到集群内的节点上，它监听kube-apiserver，查询还未分配Node的Pod，然后根据调度策略为这些Pod分配节点。我们前面讲到的kubernetes的各种调度策略就是它实现的。

#### 4.2 部署

**通过系统服务方式部署**

$ cp target/master-node/kube-scheduler.service /lib/systemd/system/

$ systemctl enable kube-scheduler.service

$ service kube-scheduler start

$ journalctl -f -u kube-scheduler

#### 4.3 重点配置说明

[Unit]  
Description=Kubernetes Scheduler  
...  
[Service]  
ExecStart=/home/michael/bin/kube-scheduler \  
#对外服务的监听地址，这里表示只有本机的程序可以访问它  
--address=127.0.0.1 \  
#apiserver的url  
--master=[http://127.0.0.1:8080](http://127.0.0.1:8080/) \  
...

## 5. 部署CalicoNode（所有节点）

#### 5.1 简介

Calico实现了CNI接口，是kubernetes网络方案的一种选择，它一个纯三层的数据中心网络方案（不需要Overlay），并且与OpenStack、Kubernetes、AWS、GCE等IaaS和容器平台都有良好的集成。 Calico在每一个计算节点利用Linux Kernel实现了一个高效的vRouter来负责数据转发，而每个vRouter通过BGP协议负责把自己上运行的workload的路由信息像整个Calico网络内传播——小规模部署可以直接互联，大规模下可通过指定的BGP route reflector来完成。 这样保证最终所有的workload之间的数据流量都是通过IP路由的方式完成互联的。

#### 5.2 部署

**calico是通过系统服务+docker方式完成的**

$ cp target/all-node/kube-calico.service /lib/systemd/system/

$ systemctl enable kube-calico.service

$ service kube-calico start

$ journalctl -f -u kube-calico

#### 5.3 calico可用性验证

**查看容器运行情况**

$ docker ps

CONTAINER ID IMAGE COMMAND CREATED ...

4d371b58928b calico/node:v2.6.2 "start\_runit" 3 hours ago...

**查看节点运行情况**

$ calicoctl node status

Calico process is running.

IPv4 BGP status

+---------------+-------------------+-------+----------+-------------+

| PEER ADDRESS | PEER TYPE | STATE | SINCE | INFO |

+---------------+-------------------+-------+----------+-------------+

| 192.168.1.103 | node-to-node mesh | up | 13:13:13 | Established |

+---------------+-------------------+-------+----------+-------------+

IPv6 BGP status

No IPv6 peers found.

**查看端口BGP 协议是通过TCP 连接来建立邻居的，因此可以用netstat 命令验证 BGP Peer**

$ netstat -natp|grep ESTABLISHED|grep 179

tcp 0 0 192.168.1.102:60959 192.168.1.103:179 ESTABLISHED 29680/bird

**查看集群ippool情况**

$ calicoctl get ipPool -o yaml

- apiVersion: v1

kind: ipPool

metadata:

cidr: 172.20.0.0/16

spec:

nat-outgoing: true

#### 5.4 重点配置说明

[Unit]  
Description=calico node  
...  
[Service]  
#以docker方式运行  
ExecStart=/usr/bin/docker run --net=host --privileged --name=calico-node \  
#指定etcd endpoints（这里主要负责网络元数据一致性，确保Calico网络状态的准确性）  
-e ETCD\_ENDPOINTS=[http://192.168.1.102:2379](http://192.168.1.102:2379/) \  
#网络地址范围（同上面ControllerManager）  
-e CALICO\_IPV4POOL\_CIDR=172.20.0.0/16 \  
#镜像名，为了加快大家的下载速度，镜像都放到了阿里云上  
registry.cn-hangzhou.aliyuncs.com/imooc/calico-node:v2.6.2

## 6. 配置kubectl命令（任意节点）

#### 6.1 简介

kubectl是Kubernetes的命令行工具，是Kubernetes用户和管理员必备的管理工具。 kubectl提供了大量的子命令，方便管理Kubernetes集群中的各种功能。

#### 6.2 初始化

使用kubectl的第一步是配置Kubernetes集群以及认证方式，包括：

* cluster信息：api-server地址
* 用户信息：用户名、密码或密钥
* Context：cluster、用户信息以及Namespace的组合

我们这没有安全相关的东西，只需要设置好api-server和上下文就好啦：

#指定apiserver地址（ip替换为你自己的api-server地址）

kubectl config set-cluster kubernetes --server=http://192.168.1.102:8080

#指定设置上下文，指定cluster

kubectl config set-context kubernetes --cluster=kubernetes

#选择默认的上下文

kubectl config use-context kubernetes

通过上面的设置最终目的是生成了一个配置文件：~/.kube/config，当然你也可以手写或复制一个文件放在那，就不需要上面的命令了。

## 7. 配置kubelet（工作节点）

#### 7.1 简介

每个工作节点上都运行一个kubelet服务进程，默认监听10250端口，接收并执行master发来的指令，管理Pod及Pod中的容器。每个kubelet进程会在API Server上注册节点自身信息，定期向master节点汇报节点的资源使用情况，并通过cAdvisor监控节点和容器的资源。

#### 7.2 部署

**通过系统服务方式部署，但步骤会多一些，具体如下：**

#确保相关目录存在

$ mkdir -p /var/lib/kubelet

$ mkdir -p /etc/kubernetes

$ mkdir -p /etc/cni/net.d

#复制kubelet服务配置文件

$ cp target/worker-node/kubelet.service /lib/systemd/system/

#复制kubelet依赖的配置文件

$ cp target/worker-node/kubelet.kubeconfig /etc/kubernetes/

#复制kubelet用到的cni插件配置文件

$ cp target/worker-node/10-calico.conf /etc/cni/net.d/

$ systemctl enable kubelet.service

$ service kubelet start

$ journalctl -f -u kubelet

#### 7.3 重点配置说明

**kubelet.service**

[Unit]  
Description=Kubernetes Kubelet  
[Service]  
#kubelet工作目录，存储当前节点容器，pod等信息  
WorkingDirectory=/var/lib/kubelet  
ExecStart=/home/michael/bin/kubelet \  
#对外服务的监听地址  
--address=192.168.1.103 \  
#指定基础容器的镜像，负责创建Pod 内部共享的网络、文件系统等，这个基础容器非常重要：K8S每一个运行的 POD里面必然包含这个基础容器，如果它没有运行起来那么你的POD 肯定创建不了  
--pod-infra-container-image=registry.cn-hangzhou.aliyuncs.com/imooc/pause-amd64:3.0 \  
#访问集群方式的配置，如api-server地址等  
--kubeconfig=/etc/kubernetes/kubelet.kubeconfig \  
#声明cni网络插件  
--network-plugin=cni \  
#cni网络配置目录，kubelet会读取该目录下得网络配置  
--cni-conf-dir=/etc/cni/net.d \  
#指定 kubedns 的 Service IP(可以先分配，后续创建 kubedns 服务时指定该 IP)，--cluster-domain 指定域名后缀，这两个参数同时指定后才会生效  
--cluster-dns=10.68.0.2 \  
...

**kubelet.kubeconfig**  
kubelet依赖的一个配置，格式看也是我们后面经常遇到的yaml格式，描述了kubelet访问apiserver的方式

apiVersion: v1  
clusters:  
- cluster:  
#跳过tls，即是kubernetes的认证  
insecure-skip-tls-verify: true  
#api-server地址  
server: [http://192.168.1.102:8080](http://192.168.1.102:8080/)  
...

**10-calico.conf**  
calico作为kubernets的CNI插件的配置

{

"name": "calico-k8s-network",

"cniVersion": "0.1.0",

"type": "calico",

<!--etcd的url-->

"ed\_endpoints": "http://192.168.1.102:2379",

"logevel": "info",

"ipam": {

"type": "calico-ipam"

},

"kubernetes": {

<!--api-server的url-->

"k8s\_api\_root": "http://192.168.1.102:8080"

}

}

## 8. 小试牛刀

到这里最基础的kubernetes集群就可以工作了。下面我们就来试试看怎么去操作，控制它。 我们从最简单的命令开始，尝试一下kubernetes官方的入门教学：playground的内容。了解如何创建pod，deployments，以及查看他们的信息，深入理解他们的关系。 具体内容请看慕课网的视频吧： [《Docker+k8s微服务容器化实践》](https://coding.imooc.com/class/198.html)

## 9. 为集群增加service功能 - kube-proxy（工作节点）

#### 9.1 简介

每台工作节点上都应该运行一个kube-proxy服务，它监听API server中service和endpoint的变化情况，并通过iptables等来为服务配置负载均衡，是让我们的服务在集群外可以被访问到的重要方式。

#### 9.2 部署

**通过系统服务方式部署：**

#确保工作目录存在

$ mkdir -p /var/lib/kube-proxy

#复制kube-proxy服务配置文件

$ cp target/worker-node/kube-proxy.service /lib/systemd/system/

#复制kube-proxy依赖的配置文件

$ cp target/worker-node/kube-proxy.kubeconfig /etc/kubernetes/

$ systemctl enable kube-proxy.service

$ service kube-proxy start

$ journalctl -f -u kube-proxy

#### 9.3 重点配置说明

**kube-proxy.service**

[Unit]  
Description=Kubernetes Kube-Proxy Server ...  
[Service]  
#工作目录  
WorkingDirectory=/var/lib/kube-proxy  
ExecStart=/home/michael/bin/kube-proxy \  
#监听地址  
--bind-address=192.168.1.103 \  
#依赖的配置文件，描述了kube-proxy如何访问api-server  
--kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig \  
...

**kube-proxy.kubeconfig** 配置了kube-proxy如何访问api-server，内容与kubelet雷同，不再赘述。

#### 9.4 操练service

刚才我们在基础集群上演示了pod，deployments。下面就在刚才的基础上增加点service元素。具体内容见[《Docker+k8s微服务容器化实践》](https://coding.imooc.com/class/198.html)。

## 10. 为集群增加dns功能 - kube-dns（app）

#### 10.1 简介

kube-dns为Kubernetes集群提供命名服务，主要用来解析集群服务名和Pod的hostname。目的是让pod可以通过名字访问到集群内服务。它通过添加A记录的方式实现名字和service的解析。普通的service会解析到service-ip。headless service会解析到pod列表。

#### 10.2 部署

**通过kubernetes应用的方式部署** kube-dns.yaml文件基本与官方一致（除了镜像名不同外）。 里面配置了多个组件，之间使用”---“分隔

#到kubernetes-starter目录执行命令

$ kubectl create -f target/services/kube-dns.yaml

#### 10.3 重点配置说明

请直接参考配置文件中的注释。

#### 10.4 通过dns访问服务

这了主要演示增加kube-dns后，通过名字访问服务的原理和具体过程。演示启动dns服务和未启动dns服务的通过名字访问情况差别。 具体内容请看[《Docker+k8s微服务容器化实践》](https://coding.imooc.com/class/198.html)吧~

# 三、完整集群部署 - kubernetes-with-ca

## 1. 理解认证授权

#### 1.1 为什么要认证

想理解认证，我们得从认证解决什么问题、防止什么问题的发生入手。  
防止什么问题呢？是防止有人入侵你的集群，root你的机器后让我们集群依然安全吗？不是吧，root都到手了，那就为所欲为，防不胜防了。  
其实网络安全本身就是为了解决在某些假设成立的条件下如何防范的问题。比如一个非常重要的假设就是两个节点或者ip之间的通讯网络是不可信任的，可能会被第三方窃取，也可能会被第三方篡改。就像我们上学时候给心仪的女孩传纸条，传送的过程可能会被别的同学偷看，甚至内容可能会从我喜欢你修改成我不喜欢你了。当然这种假设不是随便想出来的，而是从网络技术现状和实际发生的问题中发现、总结出来的。kubernetes的认证也是从这个问题出发来实现的。

#### 1.2 概念梳理

为了解决上面说的问题，kubernetes并不需要自己想办法，毕竟是网络安全层面的问题，是每个服务都会遇到的问题，业内也有成熟的方案来解决。这里我们一起了解一下业内方案和相关的概念。

* **对称加密/非对称加密** 这两个概念属于密码学的东西，对于没接触过的同学不太容易理解。可以参考知乎大神的生动讲解：[《如何用通俗易懂的话来解释非对称加密》](https://www.zhihu.com/question/33645891/answer/57721969)
* **SSL/TLS** 了解了对称加密和非对称加密后，我们就可以了解一下SSL/TLS了。同样，已经有大神总结了非常好的入门文章：[《SSL/TLS协议运行机制的概述》](http://www.ruanyifeng.com/blog/2014/02/ssl_tls.html)

#### 1.3 什么是授权

授权的概念就简单多了，就是什么人具有什么样的权限，一般通过角色作为纽带把他们组合在一起。也就是一个角色一边拥有多种权限，一边拥有多个人。这样就把人和权限建立了一个关系。

## 2. kubernetes的认证授权

Kubernetes集群的所有操作基本上都是通过kube-apiserver这个组件进行的，它提供HTTP RESTful形式的API供集群内外客户端调用。需要注意的是：认证授权过程只存在HTTPS形式的API中。也就是说，如果客户端使用HTTP连接到kube-apiserver，那么是不会进行认证授权的。所以说，可以这么设置，在集群内部组件间通信使用HTTP，集群外部就使用HTTPS，这样既增加了安全性，也不至于太复杂。  
对APIServer的访问要经过的三个步骤，前面两个是认证和授权，第三个是 Admission Control，它也能在一定程度上提高安全性，不过更多是资源管理方面的作用。

#### 2.1 kubernetes的认证

kubernetes提供了多种认证方式，比如客户端证书、静态token、静态密码文件、ServiceAccountTokens等等。你可以同时使用一种或多种认证方式。只要通过任何一个都被认作是认证通过。下面我们就认识几个常见的认证方式。

* **客户端证书认证** 客户端证书认证叫作TLS双向认证，也就是服务器客户端互相验证证书的正确性，在都正确的情况下协调通信加密方案。 为了使用这个方案，api-server需要用--client-ca-file选项来开启。
* **引导Token** 当我们有非常多的node节点时，手动为每个node节点配置TLS认证比较麻烦，这时就可以用到引导token的认证方式，前提是需要在api-server开启 experimental-bootstrap-token-auth 特性，客户端的token信息与预先定义的token匹配认证通过后，自动为node颁发证书。当然引导token是一种机制，可以用到各种场景中。
* **Service Account Tokens 认证** 有些情况下，我们希望在pod内部访问api-server，获取集群的信息，甚至对集群进行改动。针对这种情况，kubernetes提供了一种特殊的认证方式：Service Account。 Service Account 和 pod、service、deployment 一样是 kubernetes 集群中的一种资源，用户也可以创建自己的 Service Account。 ServiceAccount 主要包含了三个内容：namespace、Token 和 CA。namespace 指定了 pod 所在的 namespace，CA 用于验证 apiserver 的证书，token 用作身份验证。它们都通过 mount 的方式保存在 pod 的文件系统中。

#### 2.2 kubernetes的授权

在Kubernetes1.6版本中新增角色访问控制机制（Role-Based Access，RBAC）让集群管理员可以针对特定使用者或服务账号的角色，进行更精确的资源访问控制。在RBAC中，权限与角色相关联，用户通过成为适当角色的成员而得到这些角色的权限。这就极大地简化了权限的管理。在一个组织中，角色是为了完成各种工作而创造，用户则依据它的责任和资格来被指派相应的角色，用户可以很容易地从一个角色被指派到另一个角色。 目前 Kubernetes 中有一系列的鉴权机制，因为Kubernetes社区的投入和偏好，相对于其它鉴权机制而言，RBAC是更好的选择。具体RBAC是如何体现在kubernetes系统中的我们会在后面的部署中逐步的深入了解。

#### 2.3 kubernetes的AdmissionControl

AdmissionControl - 准入控制本质上为一段准入代码，在对kubernetes api的请求过程中，顺序为：先经过认证 & 授权，然后执行准入操作，最后对目标对象进行操作。这个准入代码在api-server中，而且必须被编译到二进制文件中才能被执行。 在对集群进行请求时，每个准入控制代码都按照一定顺序执行。如果有一个准入控制拒绝了此次请求，那么整个请求的结果将会立即返回，并提示用户相应的error信息。 常用组件（控制代码）如下：

* AlwaysAdmit：允许所有请求
* AlwaysDeny：禁止所有请求，多用于测试环境
* ServiceAccount：它将serviceAccounts实现了自动化，它会辅助serviceAccount做一些事情，比如如果pod没有serviceAccount属性，它会自动添加一个default，并确保pod的serviceAccount始终存在
* LimitRanger：他会观察所有的请求，确保没有违反已经定义好的约束条件，这些条件定义在namespace中LimitRange对象中。如果在kubernetes中使用LimitRange对象，则必须使用这个插件。
* NamespaceExists：它会观察所有的请求，如果请求尝试创建一个不存在的namespace，则这个请求被拒绝。

## 3. 环境准备

#### 3.1 停止原有kubernetes相关服务

开始之前我们要先把基础版本的集群停掉，包括service，deployments，pods以及运行的所有kubernetes组件

#删除services

$ kubectl delete services nginx-service

#删除deployments

$ kubectl delete deploy kubernetes-bootcamp

$ kubectl delete deploy nginx-deployment

#停掉worker节点的服务

$ service kubelet stop && rm -fr /var/lib/kubelet/\*

$ service kube-proxy stop && rm -fr /var/lib/kube-proxy/\*

$ service kube-calico stop

#停掉master节点的服务

$ service kube-calico stop

$ service kube-scheduler stop

$ service kube-controller-manager stop

$ service kube-apiserver stop

$ service etcd stop && rm -fr /var/lib/etcd/\*

#### 3.2 生成配置（所有节点）

跟基础环境搭建一样，我们需要生成kubernetes-with-ca的所有相关配置文件

$ cd ~/kubernetes-starter

#按照配置文件的提示编辑好配置

$ vi config.properties

#生成配置

$ ./gen-config.sh with-ca

#### 3.3 安装cfssl（所有节点）

cfssl是非常好用的CA工具，我们用它来生成证书和秘钥文件  
安装过程比较简单，如下：

#下载

$ wget -q --show-progress --https-only --timestamping \

https://pkg.cfssl.org/R1.2/cfssl\_linux-amd64 \

https://pkg.cfssl.org/R1.2/cfssljson\_linux-amd64

#修改为可执行权限

$ chmod +x cfssl\_linux-amd64 cfssljson\_linux-amd64

#移动到bin目录

$ mv cfssl\_linux-amd64 /usr/local/bin/cfssl

$ mv cfssljson\_linux-amd64 /usr/local/bin/cfssljson

#验证

$ cfssl version

#### 3.4 生成根证书（主节点）

根证书是证书信任链的根，各个组件通讯的前提是有一份大家都信任的证书（根证书），每个人使用的证书都是由这个根证书签发的。

#所有证书相关的东西都放在这

$ mkdir -p /etc/kubernetes/ca

#准备生成证书的配置文件

$ cp ~/kubernetes-starter/target/ca/ca-config.json /etc/kubernetes/ca

$ cp ~/kubernetes-starter/target/ca/ca-csr.json /etc/kubernetes/ca

#生成证书和秘钥

$ cd /etc/kubernetes/ca

$ cfssl gencert -initca ca-csr.json | cfssljson -bare ca

#生成完成后会有以下文件（我们最终想要的就是ca-key.pem和ca.pem，一个秘钥，一个证书）

$ ls

ca-config.json ca.csr ca-csr.json ca-key.pem ca.pem

## 4. 改造etcd

#### 4.1 准备证书

etcd节点需要提供给其他服务访问，就要验证其他服务的身份，所以需要一个标识自己监听服务的server证书，当有多个etcd节点的时候也需要client证书与etcd集群其他节点交互，当然也可以client和server使用同一个证书因为它们本质上没有区别。

#etcd证书放在这

$ mkdir -p /etc/kubernetes/ca/etcd

#准备etcd证书配置

$ cp ~/kubernetes-starter/target/ca/etcd/etcd-csr.json /etc/kubernetes/ca/etcd/

$ cd /etc/kubernetes/ca/etcd/

#使用根证书(ca.pem)签发etcd证书

$ cfssl gencert \

-ca=/etc/kubernetes/ca/ca.pem \

-ca-key=/etc/kubernetes/ca/ca-key.pem \

-config=/etc/kubernetes/ca/ca-config.json \

-profile=kubernetes etcd-csr.json | cfssljson -bare etcd

#跟之前类似生成三个文件etcd.csr是个中间证书请求文件，我们最终要的是etcd-key.pem和etcd.pem

$ ls

etcd.csr etcd-csr.json etcd-key.pem etcd.pem

#### 4.2 改造etcd服务

建议大家先比较一下增加认证的etcd配置与原有配置的区别，做到心中有数。 可以使用命令比较：

$ cd ~/kubernetes-starter/

$ vimdiff kubernetes-simple/master-node/etcd.service kubernetes-with-ca/master-node/etcd.service

**更新etcd服务：**

$ cp ~/kubernetes-starter/target/master-node/etcd.service /lib/systemd/system/

$ systemctl daemon-reload

$ service etcd start

#验证etcd服务（endpoints自行替换）

$ ETCDCTL\_API=3 etcdctl \

--endpoints=https://192.168.1.102:2379 \

--cacert=/etc/kubernetes/ca/ca.pem \

--cert=/etc/kubernetes/ca/etcd/etcd.pem \

--key=/etc/kubernetes/ca/etcd/etcd-key.pem \

endpoint health

## 5. 改造api-server

#### 5.1 准备证书

#api-server证书放在这，api-server是核心，文件夹叫kubernetes吧，如果想叫apiserver也可以，不过相关的地方都需要修改哦

$ mkdir -p /etc/kubernetes/ca/kubernetes

#准备apiserver证书配置

$ cp ~/kubernetes-starter/target/ca/kubernetes/kubernetes-csr.json /etc/kubernetes/ca/kubernetes/

$ cd /etc/kubernetes/ca/kubernetes/

#使用根证书(ca.pem)签发kubernetes证书

$ cfssl gencert \

-ca=/etc/kubernetes/ca/ca.pem \

-ca-key=/etc/kubernetes/ca/ca-key.pem \

-config=/etc/kubernetes/ca/ca-config.json \

-profile=kubernetes kubernetes-csr.json | cfssljson -bare kubernetes

#跟之前类似生成三个文件kubernetes.csr是个中间证书请求文件，我们最终要的是kubernetes-key.pem和kubernetes.pem

$ ls

kubernetes.csr kubernetes-csr.json kubernetes-key.pem kubernetes.pem

#### 5.2 改造api-server服务

**查看diff**

$ cd ~/kubernetes-starter

$ vimdiff kubernetes-simple/master-node/kube-apiserver.service kubernetes-with-ca/master-node/kube-apiserver.service

**生成token认证文件**

#生成随机token

$ head -c 16 /dev/urandom | od -An -t x | tr -d ' '

8afdf3c4eb7c74018452423c29433609

#按照固定格式写入token.csv，注意替换token内容

$ echo "8afdf3c4eb7c74018452423c29433609,kubelet-bootstrap,10001,\"system:kubelet-bootstrap\"" > /etc/kubernetes/ca/kubernetes/token.csv

**更新api-server服务**

$ cp ~/kubernetes-starter/target/master-node/kube-apiserver.service /lib/systemd/system/

$ systemctl daemon-reload

$ service kube-apiserver start

#检查日志

$ journalctl -f -u kube-apiserver

## 6. 改造controller-manager

controller-manager一般与api-server在同一台机器上，所以可以使用非安全端口与api-server通讯，不需要生成证书和私钥。

#### 6.1 改造controller-manager服务

**查看diff**

$ cd ~/kubernetes-starter/

$ vimdiff kubernetes-simple/master-node/kube-controller-manager.service kubernetes-with-ca/master-node/kube-controller-manager.service

**更新controller-manager服务**

$ cp ~/kubernetes-starter/target/master-node/kube-controller-manager.service /lib/systemd/system/

$ systemctl daemon-reload

$ service kube-controller-manager start

#检查日志

$ journalctl -f -u kube-controller-manager

## 7. 改造scheduler

scheduler一般与apiserver在同一台机器上，所以可以使用非安全端口与apiserver通讯。不需要生成证书和私钥。

#### 7.1 改造scheduler服务

**查看diff** 比较会发现两个文件并没有区别，不需要改造

$ cd ~/kubernetes-starter/

$ vimdiff kubernetes-simple/master-node/kube-scheduler.service kubernetes-with-ca/master-node/kube-scheduler.service

**启动服务**

$ service kube-scheduler start

#检查日志

$ journalctl -f -u kube-scheduler

## 8. 改造kubectl

#### 8.1 准备证书

#kubectl证书放在这，由于kubectl相当于系统管理员，我们使用admin命名

$ mkdir -p /etc/kubernetes/ca/admin

#准备admin证书配置 - kubectl只需客户端证书，因此证书请求中 hosts 字段可以为空

$ cp ~/kubernetes-starter/target/ca/admin/admin-csr.json /etc/kubernetes/ca/admin/

$ cd /etc/kubernetes/ca/admin/

#使用根证书(ca.pem)签发admin证书

$ cfssl gencert \

-ca=/etc/kubernetes/ca/ca.pem \

-ca-key=/etc/kubernetes/ca/ca-key.pem \

-config=/etc/kubernetes/ca/ca-config.json \

-profile=kubernetes admin-csr.json | cfssljson -bare admin

#我们最终要的是admin-key.pem和admin.pem

$ ls

admin.csr admin-csr.json admin-key.pem admin.pem

#### 8.2 配置kubectl

#指定apiserver的地址和证书位置（ip自行修改）

$ kubectl config set-cluster kubernetes \

--certificate-authority=/etc/kubernetes/ca/ca.pem \

--embed-certs=true \

--server=https://192.168.1.102:6443

#设置客户端认证参数，指定admin证书和秘钥

$ kubectl config set-credentials admin \

--client-certificate=/etc/kubernetes/ca/admin/admin.pem \

--embed-certs=true \

--client-key=/etc/kubernetes/ca/admin/admin-key.pem

#关联用户和集群

$ kubectl config set-context kubernetes \

--cluster=kubernetes --user=admin

#设置当前上下文

$ kubectl config use-context kubernetes

#设置结果就是一个配置文件，可以看看内容

$ cat ~/.kube/config

**验证master节点**

#可以使用刚配置好的kubectl查看一下组件状态

$ kubectl get componentstatus

NAME STATUS MESSAGE ERROR

scheduler Healthy ok

controller-manager Healthy ok

etcd-0 Healthy {"health": "true"}

## 9. 改造calico-node

#### 9.1 准备证书

后续可以看到calico证书用在四个地方：

* calico/node 这个docker 容器运行时访问 etcd 使用证书
* cni 配置文件中，cni 插件需要访问 etcd 使用证书
* calicoctl 操作集群网络时访问 etcd 使用证书
* calico/kube-controllers 同步集群网络策略时访问 etcd 使用证书

#calico证书放在这

$ mkdir -p /etc/kubernetes/ca/calico

#准备calico证书配置 - calico只需客户端证书，因此证书请求中 hosts 字段可以为空

$ cp ~/kubernetes-starter/target/ca/calico/calico-csr.json /etc/kubernetes/ca/calico/

$ cd /etc/kubernetes/ca/calico/

#使用根证书(ca.pem)签发calico证书

$ cfssl gencert \

-ca=/etc/kubernetes/ca/ca.pem \

-ca-key=/etc/kubernetes/ca/ca-key.pem \

-config=/etc/kubernetes/ca/ca-config.json \

-profile=kubernetes calico-csr.json | cfssljson -bare calico

#我们最终要的是calico-key.pem和calico.pem

$ ls

calico.csr calico-csr.json calico-key.pem calico.pem

#### 9.2 改造calico服务

**查看diff**

$ cd ~/kubernetes-starter

$ vimdiff kubernetes-simple/all-node/kube-calico.service kubernetes-with-ca/all-node/kube-calico.service

通过diff会发现，calico多了几个认证相关的文件：  
/etc/kubernetes/ca/ca.pem  
/etc/kubernetes/ca/calico/calico.pem  
/etc/kubernetes/ca/calico/calico-key.pem  
由于calico服务是所有节点都需要启动的，大家需要把这几个文件拷贝到每台服务器上

**更新calico服务**

$ cp ~/kubernetes-starter/target/all-node/kube-calico.service /lib/systemd/system/

$ systemctl daemon-reload

$ service kube-calico start

#验证calico（能看到其他节点的列表就对啦）

$ calicoctl node status

## 10. 改造kubelet

我们这里让kubelet使用引导token的方式认证，所以认证方式跟之前的组件不同，它的证书不是手动生成，而是由工作节点TLS BootStrap 向api-server请求，由主节点的controller-manager 自动签发。

#### 10.1 创建角色绑定（主节点）

引导token的方式要求客户端向api-server发起请求时告诉他你的用户名和token，并且这个用户是具有一个特定的角色：system:node-bootstrapper，所以需要先将 bootstrap token 文件中的 kubelet-bootstrap 用户赋予这个特定角色，然后 kubelet 才有权限发起创建认证请求。 **在主节点执行下面命令**

#可以通过下面命令查询clusterrole列表

$ kubectl -n kube-system get clusterrole

#可以回顾一下token文件的内容

$ cat /etc/kubernetes/ca/kubernetes/token.csv

8afdf3c4eb7c74018452423c29433609,kubelet-bootstrap,10001,"system:kubelet-bootstrap"

#创建角色绑定（将用户kubelet-bootstrap与角色system:node-bootstrapper绑定）

$ kubectl create clusterrolebinding kubelet-bootstrap \

--clusterrole=system:node-bootstrapper --user=kubelet-bootstrap

#### 10.2 创建bootstrap.kubeconfig（工作节点）

这个配置是用来完成bootstrap token认证的，保存了像用户，token等重要的认证信息，这个文件可以借助kubectl命令生成：（也可以自己写配置）

#设置集群参数(注意替换ip)

$ kubectl config set-cluster kubernetes \

--certificate-authority=/etc/kubernetes/ca/ca.pem \

--embed-certs=true \

--server=https://192.168.1.102:6443 \

--kubeconfig=bootstrap.kubeconfig

#设置客户端认证参数(注意替换token)

$ kubectl config set-credentials kubelet-bootstrap \

--token=8afdf3c4eb7c74018452423c29433609 \

--kubeconfig=bootstrap.kubeconfig

#设置上下文

$ kubectl config set-context default \

--cluster=kubernetes \

--user=kubelet-bootstrap \

--kubeconfig=bootstrap.kubeconfig

#选择上下文

$ kubectl config use-context default --kubeconfig=bootstrap.kubeconfig

#将刚生成的文件移动到合适的位置

$ mv bootstrap.kubeconfig /etc/kubernetes/

#### 10.3 准备cni配置

**查看diff**

$ cd ~/kubernetes-starter

$ vimdiff kubernetes-simple/worker-node/10-calico.conf kubernetes-with-ca/worker-node/10-calico.conf

**copy配置**

$ cp ~/kubernetes-starter/target/worker-node/10-calico.conf /etc/cni/net.d/

#### 10.4 改造kubelet服务

**查看diff**

$ cd ~/kubernetes-starter

$ vimdiff kubernetes-simple/worker-node/kubelet.service kubernetes-with-ca/worker-node/kubelet.service

**更新服务**

$ cp ~/kubernetes-starter/target/worker-node/kubelet.service /lib/systemd/system/

$ systemctl daemon-reload

$ service kubelet start

#启动kubelet之后到master节点允许worker加入(批准worker的tls证书请求)

#--------\*在主节点执行\*---------

$ kubectl get csr|grep 'Pending' | awk '{print $1}'| xargs kubectl certificate approve

#-----------------------------

#检查日志

$ journalctl -f -u kubelet

## 11. 改造kube-proxy

#### 11.1 准备证书

#proxy证书放在这

$ mkdir -p /etc/kubernetes/ca/kube-proxy

#准备proxy证书配置 - proxy只需客户端证书，因此证书请求中 hosts 字段可以为空。

#CN 指定该证书的 User 为 system:kube-proxy，预定义的 ClusterRoleBinding system:node-proxy 将User system:kube-proxy 与 Role system:node-proxier 绑定，授予了调用 kube-api-server proxy的相关 API 的权限

$ cp ~/kubernetes-starter/target/ca/kube-proxy/kube-proxy-csr.json /etc/kubernetes/ca/kube-proxy/

$ cd /etc/kubernetes/ca/kube-proxy/

#使用根证书(ca.pem)签发calico证书

$ cfssl gencert \

-ca=/etc/kubernetes/ca/ca.pem \

-ca-key=/etc/kubernetes/ca/ca-key.pem \

-config=/etc/kubernetes/ca/ca-config.json \

-profile=kubernetes kube-proxy-csr.json | cfssljson -bare kube-proxy

#我们最终要的是kube-proxy-key.pem和kube-proxy.pem

$ ls

kube-proxy.csr kube-proxy-csr.json kube-proxy-key.pem kube-proxy.pem

#### 11.2 生成kube-proxy.kubeconfig配置

#设置集群参数（注意替换ip）

$ kubectl config set-cluster kubernetes \

--certificate-authority=/etc/kubernetes/ca/ca.pem \

--embed-certs=true \

--server=https://192.168.1.102:6443 \

--kubeconfig=kube-proxy.kubeconfig

#置客户端认证参数

$ kubectl config set-credentials kube-proxy \

--client-certificate=/etc/kubernetes/ca/kube-proxy/kube-proxy.pem \

--client-key=/etc/kubernetes/ca/kube-proxy/kube-proxy-key.pem \

--embed-certs=true \

--kubeconfig=kube-proxy.kubeconfig

#设置上下文参数

$ kubectl config set-context default \

--cluster=kubernetes \

--user=kube-proxy \

--kubeconfig=kube-proxy.kubeconfig

#选择上下文

$ kubectl config use-context default --kubeconfig=kube-proxy.kubeconfig

#移动到合适位置

$ mv kube-proxy.kubeconfig /etc/kubernetes/kube-proxy.kubeconfig

#### 11.3 改造kube-proxy服务

**查看diff**

$ cd ~/kubernetes-starter

$ vimdiff kubernetes-simple/worker-node/kube-proxy.service kubernetes-with-ca/worker-node/kube-proxy.service

经过diff你应该发现kube-proxy.service没有变化

**启动服务**

#如果之前的配置没有了，可以重新复制一份过去

$ cp ~/kubernetes-starter/target/worker-node/kube-proxy.service /lib/systemd/system/

$ systemctl daemon-reload

#安装依赖软件

$ apt install conntrack

#启动服务

$ service kube-proxy start

#查看日志

$ journalctl -f -u kube-proxy

## 12. 改造kube-dns

kube-dns有些特别，因为它本身是运行在kubernetes集群中，以kubernetes应用的形式运行。所以它的认证授权方式跟之前的组件都不一样。它需要用到service account认证和RBAC授权。  
**service account认证：**  
每个service account都会自动生成自己的secret，用于包含一个ca，token和secret，用于跟api-server认证  
**RBAC授权：**  
权限、角色和角色绑定都是kubernetes自动创建好的。我们只需要创建一个叫做kube-dns的 ServiceAccount即可，官方现有的配置已经把它包含进去了。

#### 12.1 准备配置文件

我们在官方的基础上添加的变量，生成适合我们集群的配置。直接copy就可以啦

$ cd ~/kubernetes-starter

$ vimdiff kubernetes-simple/services/kube-dns.yaml kubernetes-with-ca/services/kube-dns.yaml

大家可以看到diff只有一处，新的配置没有设定api-server。不访问api-server，它是怎么知道每个服务的cluster ip和pod的endpoints的呢？这就是因为kubernetes在启动每个服务service的时候会以环境变量的方式把所有服务的ip，端口等信息注入进来。

#### 12.2 创建kube-dns

$ kubectl create -f ~/kubernetes-starter/target/services/kube-dns.yaml

#看看启动是否成功

$ kubectl -n kube-system get pods

## 13. 再试牛刀

终于，安全版的kubernetes集群我们部署完成了。  
下面我们使用新集群先温习一下之前学习过的命令，然后再认识一些新的命令，新的参数，新的功能。同样，具体内容请看[视频教程](https://coding.imooc.com/class/198.html)吧~

# 四、kubernetes集群部署微服务

## 1. 微服务部署方案 - 思路整理

##### 我们有如下微服务：

* 消息服务：message-service
* 课程dubbo服务：course-dubbo-service
* 课程web服务：course-edge-service
* 用户thrift服务：user-thrift-service
* 用户web服务：user-edge-service
* API网关：api-gateway

##### 把它们放到kubernetes集群运行我们要考虑什么问题？

* 哪些服务适合单独成为一个pod？哪些服务适合在一个pod中？
* 在一个pod里面的服务如何彼此访问？他们的服务如何对外提供服务？
* 单独的pod如何对外提供服务？
* 哪个服务作为整个服务的入口，入口服务如何对外提供服务？

## 2. 搞定配置

配置的模板已经为大家准备好了，但是还需要大家做一下处理才能使用哦，参考下面脚本：

$ cd ~/kubernetes-starter/service-config/

$ ls

api-gateway.yaml message-service.yaml

course-service.yaml user-service.yaml

#替换变量 - (hub.mooc.com:8080是我的环境的镜像仓库地址，大家修改为各自的仓库)

$ sed -i 's/{{HUB}}/hub.mooc.com:8080/g' \*

## 3. 部署服务

##### 部署前准备：

* **要过一遍我们现有的代码配置，看看是否有需要修改的，修改后需要新生成镜像**
* **要启动好微服务依赖的服务，像zookeeper，mysql，registry等**