Kubernetes

主要内容

Kubernetes 架构

Kubernetes 集群搭建

Kubernetes 基础概念

Kubernetes 部署容器化应用

DashBoard 工具

Ingress

Helm

Kubernetes 搭建高可用集群

Kubernetes 部署微服务项目

学习目标

知识点	要求
Kubernetes 架构	掌握
Kubernetes 集群搭建	掌握
Kubernetes 基础概念	掌握
Kubernetes 部署容器化应用	掌握
DashBoard 工具	掌握
Ingress	掌握
Helm	掌握

Kubernetes 搭建高可用集群	掌握
Kubernetes 部署微服务项目	掌握

一、 Kubernetes 介绍

1 Kubernetes 的产生

Kubernetes 是一个开源的容器编排管理工具。

容器编排工具：

最开始我们使用物理机部署项目，但物理机成本较高。

为了节约成本，且能实现应用之间的资源隔离，我们可以使用虚拟机在操作系统中模拟出多台子电脑，每台虚拟机运行单个应用。但是虚拟机启动慢，占用空间大，必须安装完整的操作系统，非常浪费资源。

于是，容器化技术应运而生，它不需要虚拟出整个操作系统，只需要虚拟一个小规模的环境即可。容器的启动速度很快，基本不消耗额外的系统资源。Docker 是应用最为广泛的容器技术。

但是随着微服务技术的广泛应用，部署项目容器的数量越来越多，由此衍生了管理容器的重大问题。Google 在 2014 年开源了容器编排引擎 Kubernetes，用于管理容器化应用程序的部署、规划、和扩展，使我们应用的部署和运维更加方便。

2 Kubernetes 简介

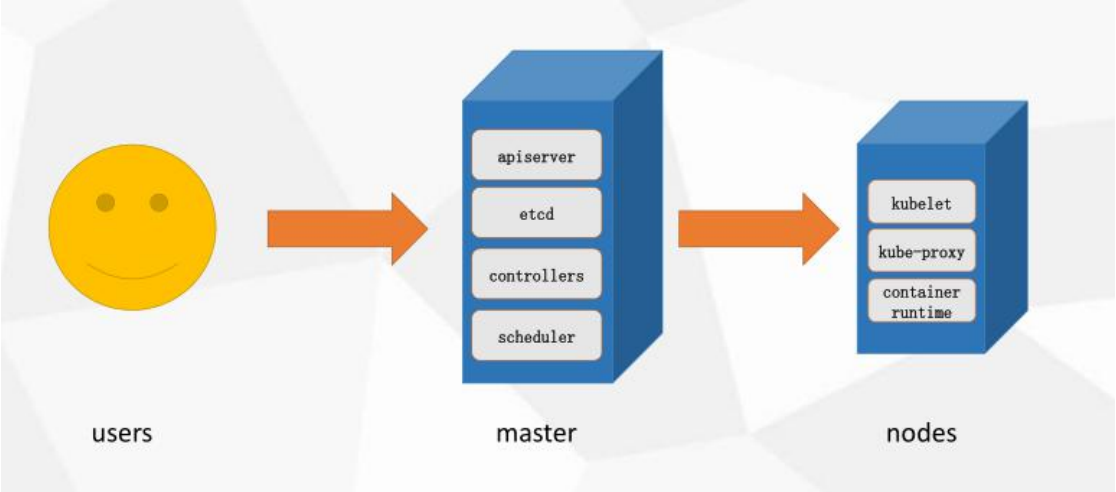
Kubernetes 来自于希腊语，含义是**舵手**或**领航员**，简称 k8s。

k8s 是采用 Go 语言开发的，Go 语言是谷歌 2009 发布的一款开源编程语言。

k8s 的前身是 Google 的 Borg 系统，2014 年 6 月由 Google 公司正式公布开源该系统，并命名为 Kubernetes，后来 Google 将 k8s 捐给了 CNCF（云原生计算基金会）。

其他容器编排管理工具：Docker swarm、Apache Mesos 等

3 Kubernetes 的架构



k8s 集群由 Master 节点和 Node 节点组成。

Master 节点

Master 节点是集群控制节点，管理和控制整个集群。基本上 k8s 的所有控制命令都发给它，它负责具体的执行过程。在 Master 上主要运行着：

apiserver：提供了集群管理的接口及模块之间的数据交互和通信的枢纽。

etcd：保存整个集群的状态。

controllers：自动化控制中心，负责维护管理集群状态，如：故障检测，自动扩展，滚动更新等。

scheduler： 负责资源调度，按照预定的调度策略将 Pod 调度到相应的机器上。

Node 节点

除了 master 以外的节点被称为 Node 节点，每个 Node 都会被 Master 分配一些工作负载（Docker 容器），当某个 Node 宕机时，该节点上的工作负载就会被 Master 自动转移到其它节点上。在 Node 上主要运行着：

kube-proxy：实现 service 的通信与负载均衡。

kubelet：用来处理 Master 节点下发到本节点的任务，管理 Pod 和其中的容器。

container runtime：运行容器所需要的一系列程序。

二、 Kubernetes 环境搭建

1 Kubernetes 环境搭建方式

搭建 Kubernetes 集群有多种方式：二进制包、kubeadm、第三方工具、云平台一键安装等方式都可以，课程中我们采用 kubeadm 搭建 Kubernetes 集群。

kubeadm 是官方社区推出的一个用于快速部署 kubernetes 集群的工具，这个工具能通过两条指令完成一个 kubernetes 集群的部署：

创建 Master 节点

```
kubeadm init
```

将 Node 节点加入到 Master 节点中

```
kubeadm join <Master 节点的 IP 和端口>
```

2 Kubernetes 部署环境要求

- 操作系统 CentOS7 以上
- 硬件配置：内存 2G 以上，CPU 2 核以上
- 集群内各个机器之间能相互通信，可以访问外网
- 禁止 swap 分区

  swap 分区：即硬盘交换区，当内存不够用时，可以临时使用硬盘中的这部分空间。

3 Kubernetes 集群搭建

3.1 创建三台 Linux 虚拟机

创建三台 Linux 虚拟机，一台作为 Master 节点，两台作为 Node 节点。

每台系统为 CentOS7，CPU 双核，内存 2G，硬盘 100G，网络选择桥接模式。

安装好后使用 SSH 工具（如 XShell）连接虚拟机，方便操作。

3.2 Kubernetes 部署环境准备

```
关闭防火墙
```

```
systemctl stop firewalld

systemctl disable firewalld
```

关闭 selinux

selinux：linux 的安全系统

```
sed -i 's/enforcing/disabled/' /etc/selinux/config
```

关闭 swap

```
sed -ri 's/.*swap.*/#&/' /etc/fstab
```

设置网桥参数

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

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

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

EOF
```

时间同步

```
yum install ntpdate -y

ntpdate time.windows.com
```

在 master 节点为各主机的 IP 命名

```
cat >> /etc/hosts << EOF
```

```
192.168.1.42 master

192.168.1.43 node1

192.168.1.44 node2

EOF
```

设置完成后重启虚拟机使修改生效

3.3 Kubernetes 安装

Kubernetes 的每个节点都需要安装 Docker、kubeadm、kubelet、kubectl

Docker：Kubernetes 默认容器运行环境是 Docker，因此首先需要安装 Docker

Kubectl：k8s 命令行工具，通过 kubectl 可以部署和管理应用，查看各种资源，创建、删除和更新组件

Kubeadm：用于构建 k8s 集群

Kubelet：负责启动 POD 和容器

3.3.1 安装 Docker

安装必要的一些系统工具

```
sudo yum install -y yum-utils device-mapper-persistent-data
```

配置清华大学的 yum 源镜像，加快下载速度

```
sudo yum-config-manager \

    --add-repo \
```

https://mirrors.tuna.tsinghua.edu.cn/docker-ce/linux/centos/docker-ce.repo

安装 Docker

```
yum -y install docker-ce
```

如果 yum 下载过程中断开，造成 yum 程序锁定，可运行以下命令关闭 yum 程序

```
rm -f /var/run/yum.pid
```

Docker 配置加速器加速镜像下载

```
mkdir -p /etc/docker
touch /etc/docker/daemon.json
cat >> /etc/docker/daemon.json << EOF
{
    "registry-mirrors": ["https://registry.docker-cn.com"]
}
EOF
```

自动运行 docker 服务

```
systemctl enable docker.service
```

3.3.2 安装 kubeadm，kubelet 和 kubectl

配置 yum 源镜像

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
```

```
[kubernetes]
name=Kubernetes
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64/
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg
EOF
```

安装软件

```
yum install kubelet-1.19.4 kubeadm-1.19.4 kubectl-1.19.4 -y
```

自动运行软件

```
systemctl enable kubelet.service
```

查看是否安装成功

```
yum list installed | grep kubelet
yum list installed | grep kubeadm
yum list installed | grep kubectl
```

安装软件后重启虚拟机

3.4 Kubernates 部署节点

3.4.1 部署 Master 节点

在 master 节点执行：

部署 master 节点

```
kubeadm init \
--apiserver-advertise-address=192.168.2.93 \
--image-repository registry.aliyuncs.com/google_containers \
--kubernetes-version v1.19.4 \
--service-cidr=10.96.0.0/12 \
--pod-network-cidr=10.244.0.0/16
```

service-cidr 的选取不能和 pod-cidr 及本机网络有重叠或者冲突，一般可以选择一个本机网络和 pod-cidr 都没有用到的私网地址段，比如 pod-cidr 使用 10.244.0.0/16，那么 service-cidr 可以选择 10.96.0.0/12，网络无重叠冲突即可；

执行命令

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

查看节点

```
kubectl  get  nodes
```

3.4.2 部署 Node 节点

在 Node 节点执行：

```
kubeadm join 192.168.1.42:6443 --token 3tfwx8.6jaaz2sijeo1rs4s \
        --discovery-token-ca-cert-hash
sha256:9c0fbb973d0b891fdb2b3e2bf21148f0f59a2edaf880784b7ad9ddffbcba390
4
```

3.4.3 安装网络插件

安装网络插件才能够让 Master 和 Node 节点可以通信，需要在 master 节点执行：

使用 rz 上传 kube-flannel.yml 文件

应用 kube-flannel.yml 文件得到运行时容器

```
kubectl apply -f kube-flannel.yml
```

等一会儿后查看节点状态

```
kubectl get nodes
```

3.4.4 修改节点名

在 node 节点执行

修改主机名

```
hostnamectl set-hostname node1
```

重置 node 节点

```
kubeadm reset
```

重新添加到 master 节点

```
kubeadm join 192.168.1.42:6443 --token 3tfwx8.6jaaz2sijeo1rs4s \
    --discovery-token-ca-cert-hash
sha256:9c0fbb973d0b891fdb2b3e2bf21148f0f59a2edaf880784b7ad9ddffbcba3904
```

在 master 节点执行

删除节点

```
kubectl delete node  节点名
```

至此我们的 k8s 环境就搭建好了

三、 Kubernetes 基础概念

1 Pod

在 Kubernetes 集群中，Pod 是 k8s 管理的最小单位级，它是一个或多个容器的组合。在 Pod 中，所有容器都被统一安排和调度。

Pod 中的容器有两个特点。

共享网络：Pod 中的所有容器共享同一个网络命名空间，包括 IP 地址和网络端口。

共享存储：Pod 中的所有容器能够访问共享存储卷，允许这些容器共享数据。

2 控制器

k8s 通过控制器管理和调度 Pod。k8s 有以下几种控制器：

- **ReplicationController/ReplicaSet**

RC 能够确保容器的副本数始终保持用户定义的副本数。即如果有容器异常退出，会自动创建新的 Pod 来替代，而由于异常多出来的容器也会自动回收。RS 跟 RC 没有本质不同，新版本的 k8s 建议使用 ReplicaSet 取代 ReplicationController。

虽然 RS 可以独立使用，但一般还是建议使用 Deployment 自动管理 RS。

- **Deployment**

Deployment 为 Pod 和 RS 提供了声明式定义方式管理应用 ,用来替代 RS 命令式定义方式管理应用。且 Deployment 提供了很多 RS 不支持的机制，比如滚动升级和回滚应用。

命令式：侧重于如何实现程序，就像我们刚接触编程的时候那样，我们需要把程序的实现过程按照逻辑结果一步步写下来。

声明式：侧重于定义想要什么，然后告诉计算机，让它帮你去实现。

- **DaemonSet**

DaemonSet 确保 Node 上只运行一个 Pod。当有 Node 加入集群时,会为集群新增一个 Pod。当有 Node 从集群移除时，这些 Pod 也会被回收。

- **Job**

Job 负责批处理任务，即仅执行一次的任务，它保证批处理任务的一个或多个 Pod

成功结束。比如运行一次 SQL 脚本。

- **Cron job**

  负责执行定时任务，即在给定时间点执行一次或周期性地在给定时间点执行任务。

3 Service

Service 可以管理多个 Pod 作为一个整体被外部访问。在 k8s 中有以下四种类型的 Service。

- **ClusterIP**

  默认类型，自动分配一个仅集群内部可以访问的虚拟 IP。

- **NodePort**

  在 ClusterIP 基础上为 Service 绑定一个端口，可以通过该端口访问服务。

- **LoadBalancer**

  在 NodePort 的基础上创建一个负载均衡器，并将请求转发到 NodePort。

- **ExternalName**

  把集群外部的服务引入到集群内部，在集群内部直接使用。

四、 Kubernates 部署容器化应用

容器化应用：把一个应用程序放在 docker 里部署，这个 docker 应用就是容器化应用，在 docker 中我们通过启动镜像部署容器化应用。

如何在 k8s 中部署容器化应用：

1. 获取镜像：编写 Dockerfile 制作镜像，或者从仓库拉取镜像
2. 控制器创建 pod：控制器启动镜像，创建容器并将容器放入 pod 中
3. 暴露应用，使外界可以访问应用

1 在 Kubernetes 集群中部署 Nginx

1. 从仓库拉取镜像，创建 pod

```
kubectl create deployment nginx --image=nginx
```

2. 创建 service 暴露 pod 端口

```
kubectl expose deployment nginx \
--port=80 \
--target-port=80 \
--type=NodePort
```

3. 访问应用，地址为：http://NodeIP:NodePort

4. 补充查看/删除命令：

查看节点

```
kubectl get node
```

查看 service

```
kubectl get service
```

查看控制器：

```
kubectl get deployment (deploy)
```

查看 pod

kubectl get pod

删除 service

kubectl delete service 服务名

删除控制器

kubectl delete deployment 控制器名

删除 pod

kubectl delete pod pod 名

2 在 Kubernetes 集群中部署 Tomcat

从仓库拉取镜像，创建 pod

kubectl create deployment tomcat --image=tomcat

创建 service 暴露 pod 端口

kubectl expose deployment tomcat \

--port=8080 \

--type=NodePort

访问 tomcat：http://NodeIP:NodePort

3 在 Kubernetes 集群中部署 SpringBoot 项目

3.1 构建 JDK 镜像

使用 rz 上传 JDK 压缩文件

制作 DockerFile

```
cat <<EOF > Dockerfile
# 基于 centos7，如果没有这个镜像那么它会下载这个镜像。
FROM centos:7
# 创建者
MAINTAINER hy
# 复制文件到指定目录并自动解压
ADD jdk-8u144-linux-x64.tar.gz /usr/local/jdk
# 设置环境变量
ENV JAVA_HOME=/usr/local/jdk/jdk1.8.0_144
ENV CLASSPATH=.:$JAVA_HOME/lib
ENV PATH=$JAVA_HOME/bin:$PATH:/usr/local/jdk/jdk1.8.0_144/bin
EOF
```

构建 JDK 镜像

```
docker build -t='jdk1.8' .
```

查看所有的镜像，此时就多了一个 jdk1.8

```
docker images
```

创建容器

```
docker run -di --name=jdk1.8 jdk1.8 /bin/bash
```

进入容器

```
docker exec -it jdk1.8 /bin/bash
```

3.2 构建项目镜像

使用 maven 将项目打成 jar 包，使用 rz 上传到虚拟机中

制作 DockerFile

```
cat <<EOF > Dockerfile
FROM jdk1.8
MAINTAINER hy
ADD springboot-k8s-1.0.0.jar /opt
RUN chmod +x /opt/springboot-k8s-1.0.0.jar
CMD java -jar /opt/springboot-k8s-1.0.0.jar
EOF
```

构建项目镜像：

```
docker build -t springboot-k8s-1.0.0.jar .
```

使用镜像启动容器：

```
docker run -d -p 8085:8080 springboot-k8s-1.0.0.jar
```

3.3 上传镜像

我们将项目上传镜像到 Docker Hub，方便项目部署

1. 注册登录 DockerHub 网站
2. 在 DockerHub 创建镜像仓库
3. 在 master 节点登录 DockerHub

```
docker login
```

4. 将本地镜像修改为规范的镜像名称：

```
docker tag docker 镜像名  注册用户名/仓库名
```

5. 上传镜像

```
docker push 注册用户名/仓库名
```

3.4 部署项目

命令行部署项目

```
kubectl create deployment springboot-k8s --image=注册用户名/仓库名
```

yml 文件部署项目

yml 文件是 k8s 的资源清单文件，我们可以通过 yml 文件精确修改构建参数。

1. 空运行测试，空运行测试没有真正运行项目，而是生成项目运行的 yml 配置文件。

```
kubectl create deployment springboot-k8s \
```

```
--image=461618768/springboot-k8s \
--dry-run \
--output yaml > deploy.yml
```

2. 修改 yml 文件，把镜像拉取策略改为 IfNotPresent，即如果本地有镜像就使用本地镜像，没有就拉取在线镜像。

```
containers:
        - image: springboot-k8s
          name: springboot-k8s-1-0-0-jar-8ntrx
          imagePullPolicy: IfNotPresent
```

3. 使用 yml 文件构建 deployment

```
kubectl apply -f deploy.yml
```

补充：yml 常用字段含义

参数名	字段类型	说明
apiversion	String	K8SAPI 的版本，目前基本上是 v1
kind	String	yml 文件定义的资源类型和角色,如：Deployment、Pod、Service
metadata	Object	元数据对象
metadata.name	String	元数据对象的名字，由我们编写，比如命名 Deployment
spec	Object	详细定义对象
spec.replicas	String	创建 pod 容器的数量

spec.selector	Object	标签选择，符合标签 Pod 会被这个 Service 管理
spec.strategy	Object	升级策略，默认为滚动升级
spec.containers[]	List	容器列表定义
spec.containers[].name	String	定义容器的名字
spec.containers[].image	String	定义要用到的镜像名称
spec.containers[].imagePullPolicy	String	定义镜像拉取策略。Always：默认，每次都尝试重新拉取镜像。Never：仅使用本地镜像。 IfNotPresent：如果本地有镜像就使用本地镜像，没有就拉取在线镜像。

3.5 暴露服务端口

```
kubectl expose deployment springboot-k8s --port=8080 --type=NodePort
```

3.6 访问项目

http://NodeIP:NodePort/资源路径

五、 DashBoard

1 概念

DashBoard 是 k8s 的可视化管理工具，可以基于 web 对 k8s 集群进行集中管理。

2 安装

DashBoard 的安装是将 DashBoard 作为一个容器部署在 k8s 中。

下载 yml 资源清单

wget

https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.4/aio/deploy/recommended.yaml

如果网络无法连接，课件中有该 yml 文件，下载后我们需要修改清单文件，暴露端口

```
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kubernetes-dashboard
spec:
  type: NodePort
  ports:
    - port: 443
      targetPort: 8443
      nodePort: 30001
  selector:
    k8s-app: kubernetes-dashboard
```

注释以下内容：

```
229       # nodeSelector:
230       #   "kubernetes.io/os": linux
```

```
296       # nodeSelector:
297       #   "kubernetes.io/os": linux
298       # Comment the following tolera
```

应用 yml 的资源清单，安装 DashBoard

kubectl apply -f recommended.yaml

查看安装是否成功 ,注意 DashBoard 安装到了 kubernetes-dashboard 命名空间下面

```
kubectl get pod -n kubernetes-dashboard
```

浏览器访问 https://ip:暴露端口

此时需要输入 token，token 的生成采用以下固定命令：

```
kubectl create serviceaccount dashboard-admin -n kube-system

kubectl create clusterrolebinding dashboard-admin
--clusterrole=cluster-admin --serviceaccount=kube-system:dashboard-admin

kubectl describe secrets -n kube-system $(kubectl -n kube-system get secret |
awk '/dashboard-admin/{print $1}')
```

补充：

namespace：命名空间，为了在多租户情况下，实现资源隔离，之前使用的是 default 命名空间。比如一套 k8s 集群可以针对开发环境和测试环境进行两套互不影响的部署，属于逻辑性隔离，可以针对 namespace 做资源配额。

六、 Ingress

1 NodePort 的三个端口

查看 service 的资源清单

```
kubectl get service 服务名 -o yaml
```

资源清单中有三个端口：

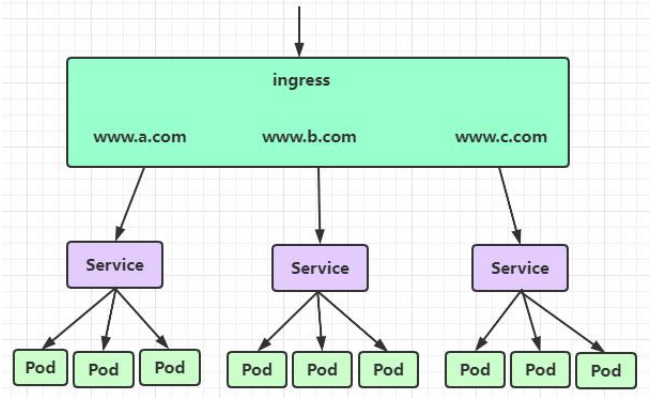
- nodePort：外部访问的端口
- port：集群内部通信的端口
- targetPort：容器中服务的端口

2 Ingress 的优点

NodePort 方式最大的缺点是每个 service 都要暴露端口，在部署微服务时会暴露大量端口加大管理难度，所以在生产环境中不推荐使用这种方式来直接发布服务。

使用 LoadBalancer 暴露服务可以解决端口过多的问题，但 LoadBlancer 需要向云平台申请负载均衡器，与云平台的耦合度太高，相当于购买了服务。

而 Ingress 相当于是服务网关，可以通过 URL 路径代理 service，只需要暴露一个端口就可以满足所有 service 对外服务的需求，生产环境建议使用这种方式。



3 安装 Ingress

Ingress 不是 k8s 的内置软件，需要单独安装，k8s 官方使用 Ingress Nginx 实现 Ingress

功能。

2.1 部署 Ingress Nginx

下载 Ingress Nginx 资源清单文件：

wget https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.41.2/deploy/static/provider/baremetal/deploy.yaml

如果网络无法连接，课件中有该 yml 文件，下载后我们需要修改清单文件

修改镜像地址为：

registry.cn-hangzhou.aliyuncs.com/google_containers/nginx-ingress-controller:0.33.0

为了能够正常访问 Ingress，还需要添加如下配置：

```
328     spec:
329       hostNetwork: true
330       dnsPolicy: ClusterFirst
331       containers:
332         - name: controller
333           image: registry.cn-hangzhou.aliyur
334           imagePullPolicy: IfNotPresent
335           lifecycle:
336             preStop:
```

运行资源清单，安装 Ingress Nginx

kubectl apply -f ingress-deploy.yaml

安装是否成功

kubectl get pod -n ingress-nginx

2.2 创建 Ingress 规则

删除相关配置：

```
kubectl delete -A ValidatingWebhookConfiguration ingress-nginx-admission
```

执行资源清单配置 Ingress 规则：

```
kubectl apply -f ingress-nginx-rule.yaml
```

查看 Ingress 规则：

```
kubectl get ingress
```

2.3 测试

由于 Ingress 暴露的是一个网址而不是 IP 地址，但服务并未发布到外网，所以测试计算机要将网址和 IP 地址做映射。

打开 C:\Windows\System32\drivers\etc\hosts，添加：

```
192.168.2.94 baizhan.abc
```

七、 Helm

1 概念

k8s 上的 deployment、service 等应用对象，都是由资源清单部署的。对于一个复杂的应用，会有很多类似的资源清单文件。例如微服务架构应用，组成应用的服务可能多达几十个。如果有更新或回滚应用的需求，可能要修改维护大量 yml 文件。

Helm 是一个 k8s 的应用管理工具，可以很方便的通过管理 yaml 文件来部署应用，更新应用版本。目前 Heml 的 V3 版本发布，极大简化了之前繁琐的使用方式。

Helm 中有 3 个重要概念：

- helm：命令行客户端工具，能够进行 chart 的创建，项目打包、发布和管理。
- chart：应用描述，一系列用于描述 k8s 资源相关文件的集合。
- release：基于 chart 部署的资源，chart 被 helm 运行后将会生成对应 release。

2 安装

使用 rz 将 helm 上传到 linux

tar zxvf helm-v3.0.0-linux-amd64.tar.gz

解压后移动到/usr/bin 目录下

mv linux-amd64/helm /usr/bin/

3 远程仓库中的 chart 安装 release

配置国内 chart 仓库

helm repo add stable http://mirror.azure.cn/kubernetes/charts

helm repo add aliyun https://kubernetes.oss-cn-hangzhou.aliyuncs.com/charts

helm repo update

查找 chart

helm search repo weave

查看 chart 具体信息

helm show chart stable/mysql

安装 release

helm install 应用名 仓库名

(helm install ui stable/weave-scope)

查看所有 release

helm list

查看发布状态

helm status 应用名

卸载 release

helm uninstall 应用名

4 自定义 chart 安装 release

构建 Chart

helm create mychart

进入构建好的 Chart

cd mychart/

进入 template 文件夹

```
cd template/
```

删除文件夹内所有自带文件

```
rm -rf *
```

生成 deployment 和 service 的 yml 文件

```
kubectl create deployment springboot-k8s
--image=461618768/springboot-k8s --dry-run --output yaml > deployment.yaml
kubectl expose deployment springboot-k8s --port=8080 --type=NodePort
--dry-run -o yaml > service.yaml
```

退出到上一级目录

```
cd ..
```

部署 release

```
helm install springboot-k8s .
```

应用升级

```
helm upgrade springboot-k8s .
```

5 使用 chart 模板安装 release

部署微服务项目时需要大量的 chart，但这些配置文件大部分参数一样，只有少部分不同，helm 提供了模板+自定义数据的方式，简化配置文件的编写和管理。

修改 values.yaml，添加需要配置的数据，如：

```
image: 461618768/springboot-k8s
serviceport: 8080
targetport: 8080
label: springboot-k8s
```

进入 template，修改 deployment.yaml 和 service.yaml，在需要配置数据的地方使用{{ .Values.参数名}}，如：

```
metadata:
  labels:
    app: {{ .Values.label }}
  name: {{ .Values.label }}
```

退出 template 目录

```
cd ..
```

查看模板渲染过的资源文件

```
helm install springboot-k8s --dry-run .
```

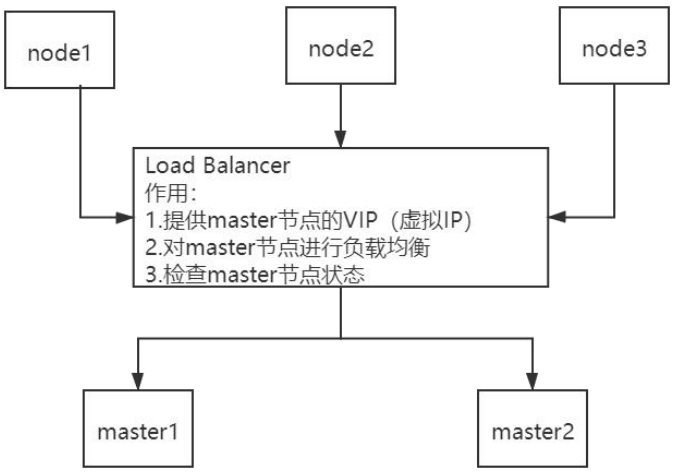
部署 release

helm install springboot-k8s .

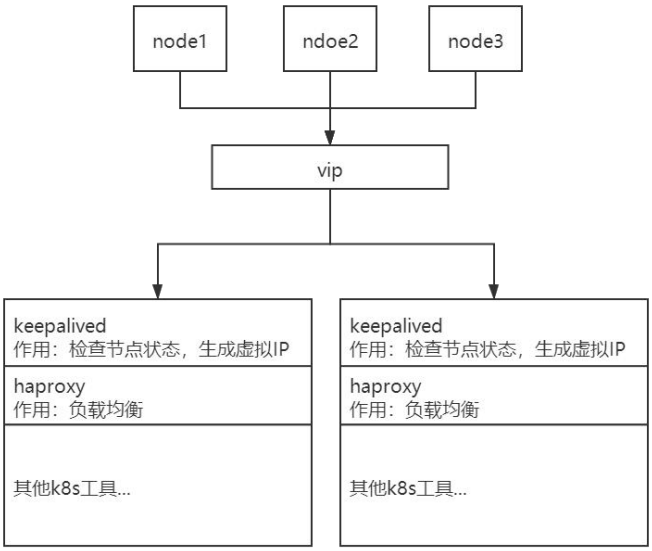
八、 搭建高可用集群

Master 节点扮演着总控中心的角色。如果 Master 节点故障，将无法进行任何集群管理。搭建高可用集群即在集群中搭建多个 master 节点，保证集群的安全性和稳定性。

高可用集群架构图：



高可用集群技术图：



1 安装一台 master 节点虚拟机

安装一台 k8s 环境的 master 节点，安装后设置该虚拟机的主机名

```
hostnamectl set-hostname master2
```

2 Master 节点安装 keepalived

在每台 master 节点做如下操作

安装相关包和 keepalived

```
yum install -y conntrack-tools libseccomp libtool-ltdl
yum install -y keepalived
```

配置 keepalived

```
cat > /etc/keepalived/keepalived.conf <<EOF
! Configuration File for keepalived
global_defs {
    router_id k8s
}
vrrp_script check_haproxy {
    script "killall -0 haproxy"
    interval 3
    weight -2
    fall 10
```

```
        rise 2
}
vrrp_instance VI_1 {
        state MASTER
        interface ens33
        virtual_router_id 51
        priority 250
        advert_int 1
        authentication {
                auth_type PASS
                auth_pass ceb1b3ec013d66163d6ab
        }
        virtual_ipaddress {
                192.168.1.50
        }
        track_script {
                check_haproxy
        }
}
EOF
```

注意 vip 和网卡名不要配错

设置自动启动 keepalive

```
systemctl start keepalived.service

systemctl enable keepalived.service

systemctl status keepalived.service
```

查看生成的 vip

```
ip a s ens33
```

3 Master 节点安装 haproxy

在每台 master 节点做如下操作：

安装 haproxy

```
yum install -y haproxy
```

配置 haproxy

```
cat > /etc/haproxy/haproxy.cfg << EOF
#---------------------------------------------------------------------
# Global settings
#---------------------------------------------------------------------
global
    log         127.0.0.1 local2
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
```

```
    maxconn        4000
    user           haproxy
    group          haproxy
    daemon
    stats socket /var/lib/haproxy/stats
defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option http-server-close
    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 3000
```

```
#---------------------------------------------------------------------

# kubernetes apiserver frontend which proxys to the backends

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

frontend kubernetes-apiserver

    mode                tcp

    bind                *:16443

    option              tcplog

    default_backend     kubernetes-apiserver

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

# round robin balancing between the various backends

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

backend kubernetes-apiserver

    mode        tcp

    balance     roundrobin

    server      master 192.168.1.42:6443 check

    server      master2 192.168.1.47:6443 check

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

# collection haproxy statistics message

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

listen stats

    bind                *:1080

    stats auth          admin:awesomePassword
```

```
        stats refresh        5s
        stats realm          HAProxy\ Statistics
        stats uri            /admin?stats
    EOF
```

配置中声明了代理的两个 master 节点服务器，指定了 haproxy 运行的端口为 16443，因此 16443 端口为集群的入口

设置自动启动

```
systemctl start haproxy
systemctl enable haproxy
```

查看安装状态

```
systemctl status haproxy
```

4 重新部署 Master1 节点

在 master1 节点操作

重置 master1 节点

```
kubeadm reset
```

使用配置文件创建 master 节点

```
cat <<EOF > kubeadm-config.yaml
apiServer:
```

```
  certSANs:
    - master
    - master2
    - k8s-vip
    - 192.168.1.42
    - 192.168.1.47
    - 192.168.1.50
    - 127.0.0.1
  extraArgs:
    authorization-mode: Node,RBAC
  timeoutForControlPlane: 4m0s
apiVersion: kubeadm.k8s.io/v1beta1
certificatesDir: /etc/kubernetes/pki
clusterName: kubernetes
controlPlaneEndpoint: "192.168.1.50:16443"
controllerManager: {}
dns:
  type: CoreDNS
etcd:
  local:
    dataDir: /var/lib/etcd
imageRepository: registry.aliyuncs.com/google_containers
```

```
kind: ClusterConfiguration
kubernetesVersion: v1.19.4
networking:
  dnsDomain: cluster.local
  podSubnet: 10.244.0.0/16
  serviceSubnet: 10.1.0.0/16
scheduler: {}
EOF
```

执行配置文件

```
kubeadm init --config kubeadm-config.yaml
```

按照提示配置环境变量

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

5 将 Master2 节点加入 k8s 集群中

将 master1 的密钥及相关文件复制到 master2

在 master1 执行：

```
ssh root@192.168.1.47 mkdir -p /etc/kubernetes/pki/etcd
scp /etc/kubernetes/admin.conf root@192.168.1.47:/etc/kubernetes
```

```
scp /etc/kubernetes/pki/{ca.*,sa.*,front-proxy-ca.*}
root@192.168.1.47:/etc/kubernetes/pki
scp /etc/kubernetes/pki/etcd/ca.* root@192.168.1.47:/etc/kubernetes/pki/etcd
```

master2 加入集群

```
kubeadm join master.k8s.io:16443 --token ckf7bs.30576l0okocepg8b
--discovery-token-ca-cert-hash
sha256:19afac8b11182f61073e254fb57b9f19ab4d798b70501036fc69ebef46094ab
a --control-plane
```

--control-plane 表示把 master 控制节点加入集群

配置环境变量

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

6 将 node 节点加入 k8s 集群

在 node 执行：

```
kubeadm join 192.168.1.50:16443 --token 9ov3pl.3mgb1d372ag2haq3 \
    --discovery-token-ca-cert-hash
sha256:9887fb98fc8d9c368926c558be0b706b3ff5b94162432bbfbeee05f128bc67c
```

0

如果安装失败，出现如下提示：

```
[ERROR FileContent--proc-sys-net-ipv4-ip_forward]: /proc/sys/net/ipv4/ip_forward contents are not set to 1
```

原因是 node 节点禁止 ip 转发，需要进行如下操作：

查看是否打开 IP 转发，0 代表禁止，1 代表转发

```
less /proc/sys/net/ipv4/ip_forward
```

如果为 0，需要更改文件的内容

```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

重启网络服务

```
service network restart
```

重启虚拟机，重新加入 k8s 集群

九、 k8s 部署微服务项目

1 准备微服务项目和 k8s 集群

我们准备一个微服务项目 demo，该项目只有一个功能：借阅图书。项目包括 eureka 注册中心，gateway 网关，用户服务，会员服务。借阅图书调用图书服务的方法，该方法内会使用 Feign 调用会员服务的方法验证是否存在该会员。

我们要将该项目部署到 k8s 集群中，该集群包含两个 master 节点，两个 node 节点。

2 部署 Eureka 注册中心

2.1 制作镜像

将注册中心打成 jar 包，使用 rz 上传到 master 节点中。

写 DockerFile

```
cat <<EOF > Dockerfile

FROM jdk1.8

MAINTAINER hy

ADD lib_registry-1.0.0.jar /opt

RUN chmod +x /opt/lib_registry-1.0.0.jar

CMD java -jar /opt/lib_registry-1.0.0.jar

EOF
```

制作 docker 镜像：

```
docker build -t lib_registry-1.0.0.jar .
```

2.2 上传镜像

登录 Docker Hub

```
docker login
```

创建镜像仓库，命名为 lib-register

将本地镜像修改为规范的镜像名称：

```
docker tag lib_registry-1.0.0.jar    461618768/lib-registry
```

上传镜像：

```
docker push 461618768/lib-registry
```

2.3 部署项目

```
kubectl create deployment lib-registry --image=461618768/lib-registry
```

2.4 暴露服务

给 Eureka 创建服务

```
cat <<EOF > eureka-server.yml
apiVersion: v1
kind: Service
metadata:
  name: lib-registry
  labels:
    app: lib-registry
spec:
  ports:
    - port: 9007
```

```
      targetPort: 8761
      nodePort: 31234
      name: lib-registry
  type: NodePort
  selector:
    app: lib-registry
EOF
```

执行资源清单

```
kubectl create -f eureka-server.yml
```

3 部署网关、book 服务、Member 服务

3.1 每个微服务的 application.yml 要做如下配置

```yaml
# 连接注册中心
eureka:
  instance:
    prefer-ip-address: true  #以IP注册进eureka,不以ID注册
  client:
    register-with-eureka: true #注册到eureka为true
    fetch-registry: true
    service-url:
      defaultZone: http://192.168.1.42:31234/eureka/ # 暴露的service
```

这样才能正常连接注册中心

3.2 制作镜像

将三个微服务打成 jar 包，使用 rz 上传到 master 节点中，之后分别构建镜像：

制作 gateway 镜像：

```
cat <<EOF > Dockerfile
FROM jdk1.8
MAINTAINER hy
ADD lib_gateway-1.0.0.jar /opt
RUN chmod +x /opt/lib_gateway-1.0.0.jar
CMD java -jar /opt/lib_gateway-1.0.0.jar
EOF

docker build -t lib_gateway-1.0.0.jar .
```

制作 member 镜像：

```
cat <<EOF > Dockerfile
FROM jdk1.8
MAINTAINER hy
ADD lib_member-1.0.0.jar /opt
RUN chmod +x /opt/lib_member-1.0.0.jar
CMD java -jar /opt/lib_member-1.0.0.jar
EOF

docker build -t lib_member-1.0.0.jar .
```

制作 book 镜像：

```
cat <<EOF > Dockerfile
FROM jdk1.8
MAINTAINER hy
ADD lib_book-1.0.0.jar /opt
RUN chmod +x /opt/lib_book-1.0.0.jar
CMD java -jar /opt/lib_book-1.0.0.jar
EOF

docker build -t lib_book-1.0.0.jar .
```

查看是否制作成功：

```
docker images
```

3.3 上传镜像

创建三个镜像仓库，命名为 lib-member、lib-book、lib-gateway、lib-registry

将本地镜像修改为规范的镜像名称：

```
docker tag lib_gateway-1.0.0.jar  461618768/lib-gateway
docker tag lib_member-1.0.0.jar  461618768/lib-member
docker tag lib_book-1.0.0.jar  461618768/lib-book
```

上传镜像：

```
docker push 461618768/lib-gateway

docker push 461618768/lib-member

docker push 461618768/lib-book
```

3.4 部署项目

```
kubectl create deployment lib-gateway --image=461618768/lib-gateway

kubectl create deployment lib-member --image=461618768/lib-member

kubectl create deployment lib-book --image=461618768/lib-book
```

3.5 Ingress 暴露网关

创建网关的 Service

```
kubectl expose deployment lib-gateway --port=80 --type=ClusterIP
```

修改 Ingress 规则文件

```
vim ingress-nginx-rule.yaml
```

执行资源清单配置 Ingress 规则：

```
kubectl apply -f ingress-nginx-rule.yaml
```

测试 Ingress

打开测试计算机的 C:\Windows\System32\drivers\etc\hosts，添加：

192.168.1.44 baizhan.lib

访问：http://baizhan.lib/book/takeBook?bookId=1&telPhone=13888888888

4 注意

k8s 主要是针对无状态应用设计的，所以一般情况下我们不在 K8S 中部署有状态应用，如 mysql、Eureka 等。

无状态应用：是指应用不会在会话中保存下次会话所需要的客户端数据。每一个会话都像首次执行一样，不会依赖之前的数据进行响应。如 WEB 服务器。

有状态应用：是指应用会在会话中保存客户端的数据，并在客户端下一次的请求中来使用那些数据。如数据库、Eureka 注册中心。

案例中我们部署了 Eureka，但实际运维过程中一般不会将其放入 k8s 集群。