厦门城市职业学院

|  |
| --- |
| 容器云平台构建与运维实践 |
| 实训大作业 |
| |  |  | | --- | --- | | **学号姓名：** | 刘承传 | | **专业班级：** | 云计算20B | | **指导老师：** | 蔡贵荣 | |

**1** **实训教学目的**

本课程实训教学以培养学生的动手能力为主导原则，让学生在理解基本理论的基础上，熟练掌握原生Kubernetes 云平台的部署、Kubernetes 容器云平台的基础使用、原生 Kubernetes 容器云平台基本测试使用、原生 Kubernetes 容器云平台应用部署、原生 Kubernetes 容器云平台运维等操作技能，掌握企业容器云的方案设计、构建和实施流程，为今后规划和部署容器云平台打下坚实基础；通过在教学过程中的规范要求，培养学生分析问题和解决实际问题的能力，强化学生的职业道德意识、职业素养意识和创新意识。

**2** **项目实训内容**

项目实训内容见表 1。

表 1 项目实训内容

|  |  |  |
| --- | --- | --- |
| 项目 | 实训内容 | 课时 |
| Kubernetes 容器云平台的部署与应用 | 1. Kubernetes 容器云平台部署 | 12 |
| 2. kubectl 工具的基本使用 | 6 |
| 3. Kubernetes 容器云平台应用 | 6 |
| 4. Kubernetes 容器云平台运维 | 6 |

**3** **实训操作环境**

(1) Windows 7 Professional（或更高版本）

(2) VMware® Workstation 15 Pro（或更高版本）

(3) CentOS 7.2（CentOS-7-x86\_64-DVD-1511.iso）镜像

1. CentOS 7.2 VM 主机若干台

**4** **主机配置要求**

根据实训指导书中的配置要求以及班级所属网段、学号、座号，对实训过程中涉及的主机名、IP 地址、文件名等加以修改。

(1) 主机配置要求见表 2。

表 2 VM 主机配置要求

|  |  |  |  |
| --- | --- | --- | --- |
| **节点** | **主机配置要求** | **IP 地址** | **主机名** |
| master 节点 | CPU 双核：开启虚拟化支持；内存 4~8G；  硬盘 100G：MBR 标准分区，boot 分区 500M、根分区 50G、swap 分区 8G；安装 CentOS 7.9；VMnet8 网卡：NAT 模式，子网 172.16.0.0/24，网关：172.16.0.2，或者默认 | 172.22.29.100 | 200250229-master、  master.40610932.com |
| node1 节点 | 172.16.0.101 | 40610932-node1、  node1.40610932.com |
| node2 节点 | 172.16.0.102 | 40610932-node2、  node2.40610932.com |

**注：本指导书中的学号、座号在实训时要改为自己的学号及学号末两位。**

(2) 班级所属网段与 IP 地址分配说明见表 3。

表 3 班级所属网段与 IP 地址分配

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **班级** | **网段与 IP 地址分配** | | **master 节点** | **node1 节点** | **node2 节点** |
| **A 班** | \*.**21**.\*.\* | IP 地址分配子 网 掩 码默 认 网 关 | 172.21.座号.100 255.255.255.0 172.21.座号.2 | 172.21.座号.101 255.255.255.0 172.21.座号.2 | 172.21.座号.102 255.255.255.0 172.21.座号.2 |
| **B 班** | \*.**22**.\*.\* | IP 地址分配子 网 掩 码默 认 网 关 | 172.22.座号.100 255.255.255.0 172.22.座号.2 | 172.22.座号.101 255.255.255.0 172.22.座号.2 | 172.22.座号.102 255.255.255.0 172.22.座号.2 |
| **C 班** | \*.**23**.\*.\* | IP 地址分配子 网 掩 码默 认 网 关 | 172.23.座号.100 255.255.255.0 172.23.座号.2 | 172.23.座号.101 255.255.255.0 172.23.座号.2 | 172.23.座号.102 255.255.255.0 172.23.座号.2 |
| **D 班** | \*.**24**.\*.\* | IP 地址分配子 网 掩 码默 认 网 关 | 172.24.座号.100 255.255.255.0 172.24.座号.2 | 172.24.座号.101 255.255.255.0 172.24.座号.2 | 172.24.座号.102 255.255.255.0 172.24.座号.2 |

**注：上述 IP 地址要结合各自所属班级网段及座号进行修改！**

**5** **实训任务实施**

1 使用 kubeadm 安装 Kubernetes 集群

【任务目标】

1. 了解 Kubernetes 容器云平台的架构

2. 了解 Kubernetes 容器云平台的基本原理

3. 掌握使用 Kubeadm 部署 Kubernetes 集群

【任务环境】

1. 虚拟机（CPU 双核以上）均已安装好 Docker，并关闭防火墙和 SELinux

2. 虚拟机能够连接外网

3. 各主机配置信息如表 1-1 所示

表 1-1 各主机配置信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 主机名 | IP 地址 | CPU/Mem | 操作系统 | 集群角色 |
| master40610932 | 172.16.0.100/24 | 2C/4G 以上 | CentOS 7.9 | 管理节点 |
| node40610932-1 | 172.16.0.101/24 | 2C/4G 以上 | CentOS 7.9 | 工作节点 |
| node40610932-2 | 172.16.0.102/24 | 2C/4G 以上 | CentOS 7.9 | 工作节点 |

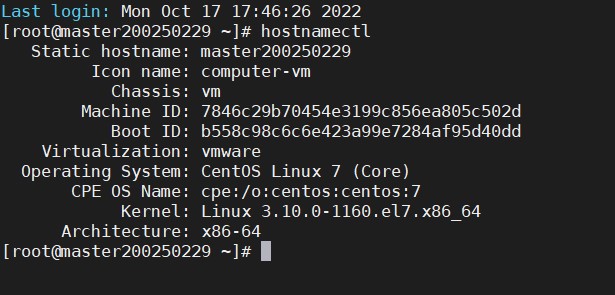
【任务实施】

**1． 系统环境配置**

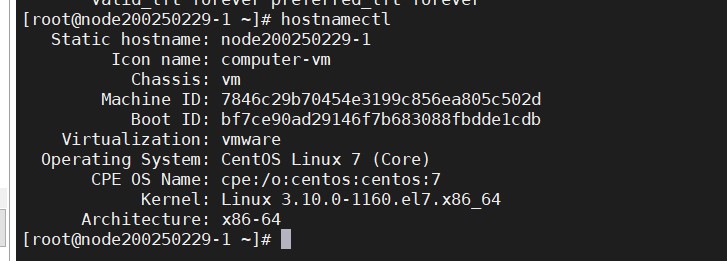
**在各主机节点上均需要配置。**

(1) 查看当前系统的内核版本，以及主机名是否符合配置要求

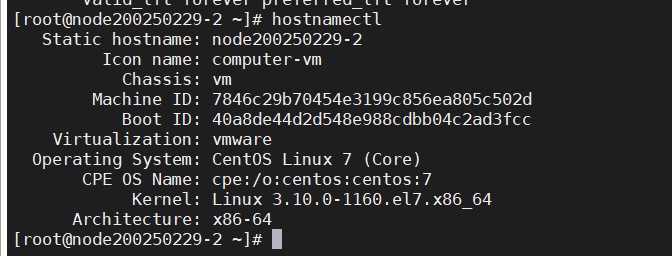
管理节点：



工作节点 1：



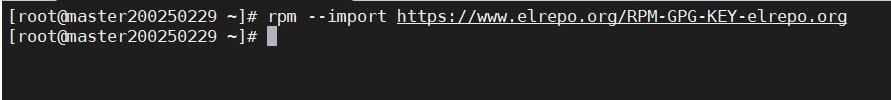
工作节点 2：



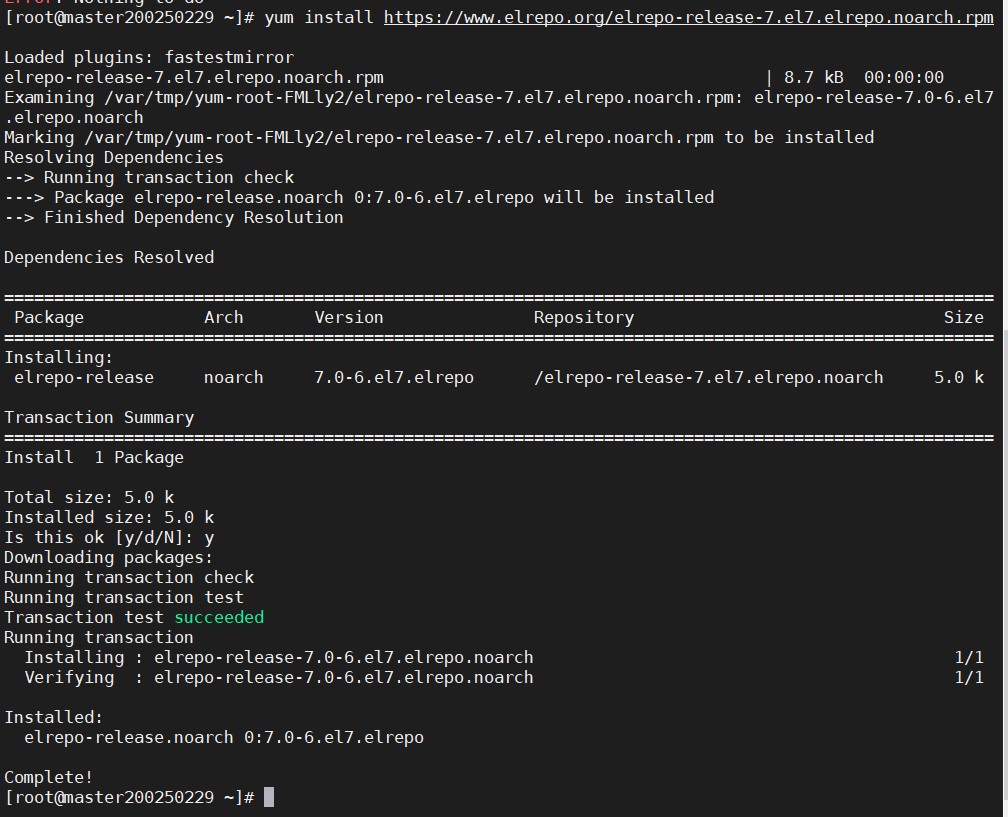
各主机当前系统的内核版本均为 Linux 3.10.0-1160.62.1.el7.x86\_64，而且主机名也符合配置要求。

(2) 升级当前系统的内核

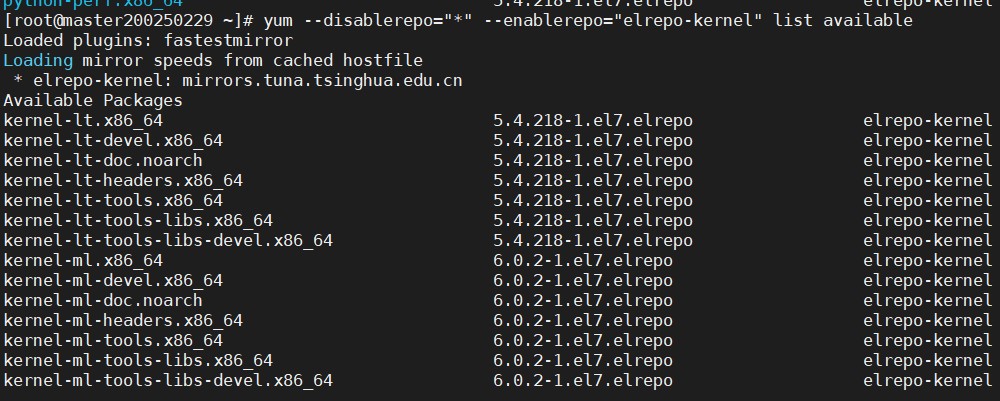
不升级内核不支持 ipvlan，有些网络会有问题！① 导入公钥 public key



② 安装 epel 源



③ 查看可用版本的内核



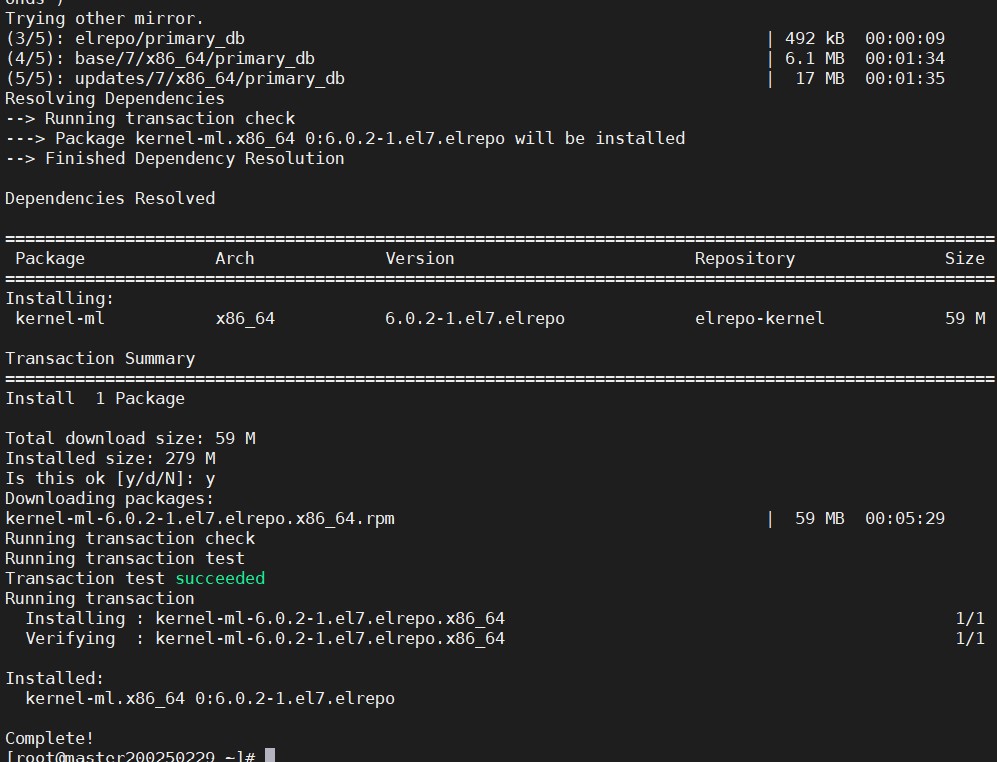
yum --disablerepo="\*" --enablerepo="elrepo-kernel" list available

--disablerepo="\*"：禁用所有 repo 配置文件

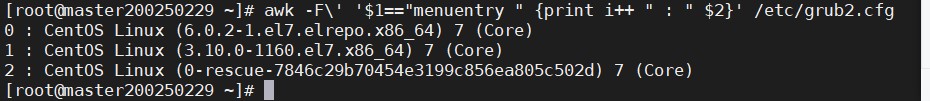
--enablerepo="elrepo-kernel"：仅使用仓库名为 elrepo-kernel 的配置文件

kernel-ml 中的 ml 是 main line stable 的缩写，elrepo-kernel 中罗列出来的是最新的稳定主线版本。kernel-lt 中的 lt 是 long term support 的缩写，elrepo-kernel 中罗列出来的长期支持版本。

④ 选择安装 ml 最新版本的内核

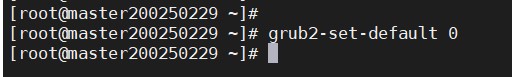


⑤ 查看系统上可以用的内核顺序



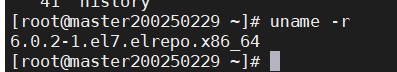
注意：前面的 0、1、2 等代表编号。

⑥ 设置默认启动内核

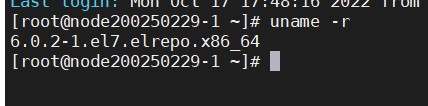


⑦ 重启并查看内核升级情况

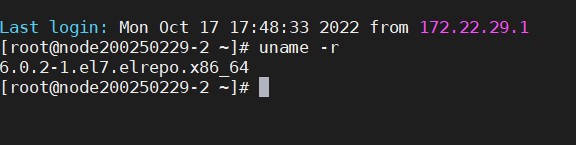
管理节点：



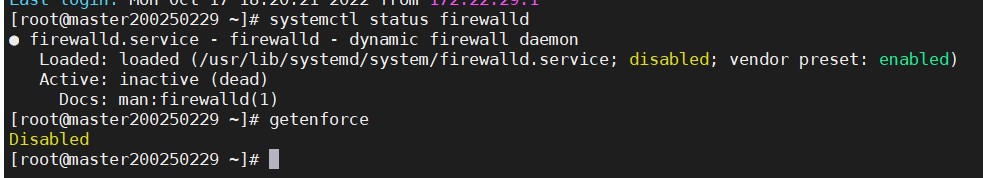
工作节点 1：

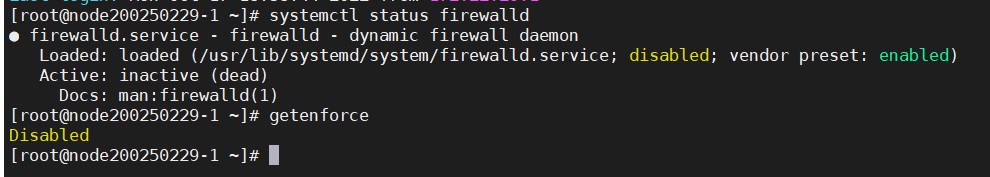


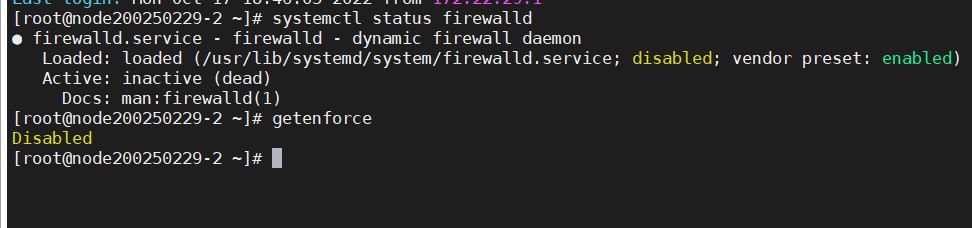
工作节点 2：



1. 检查是否已关闭防火墙及禁用 SELinux

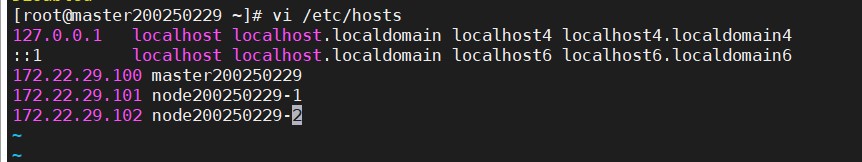




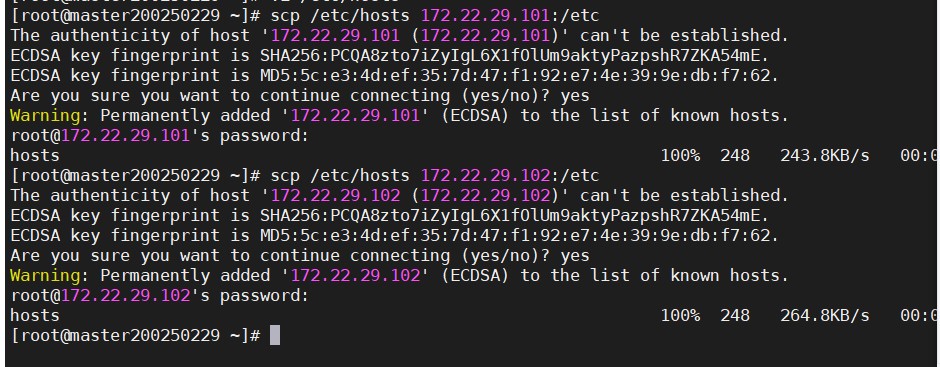


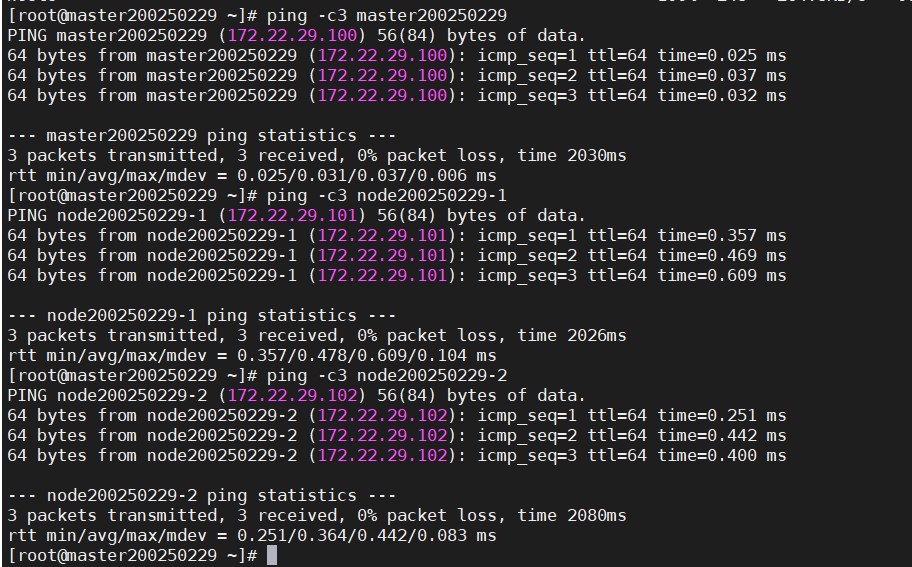
各主机系统均已关闭防火墙并禁用 SELinux。

(4) 配置主机映射



将/etc/hosts 复制到其他主机节点。

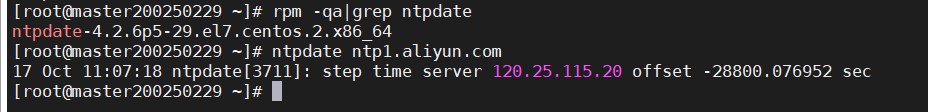




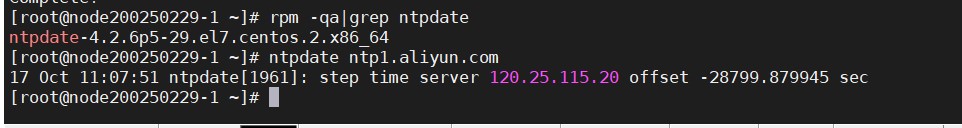
(5) 配置时间同步

在各主机上进行时钟同步，时钟同步服务器可自行配置，此处选择阿里云的时钟服务器。

管理节点：



工作节点 1：



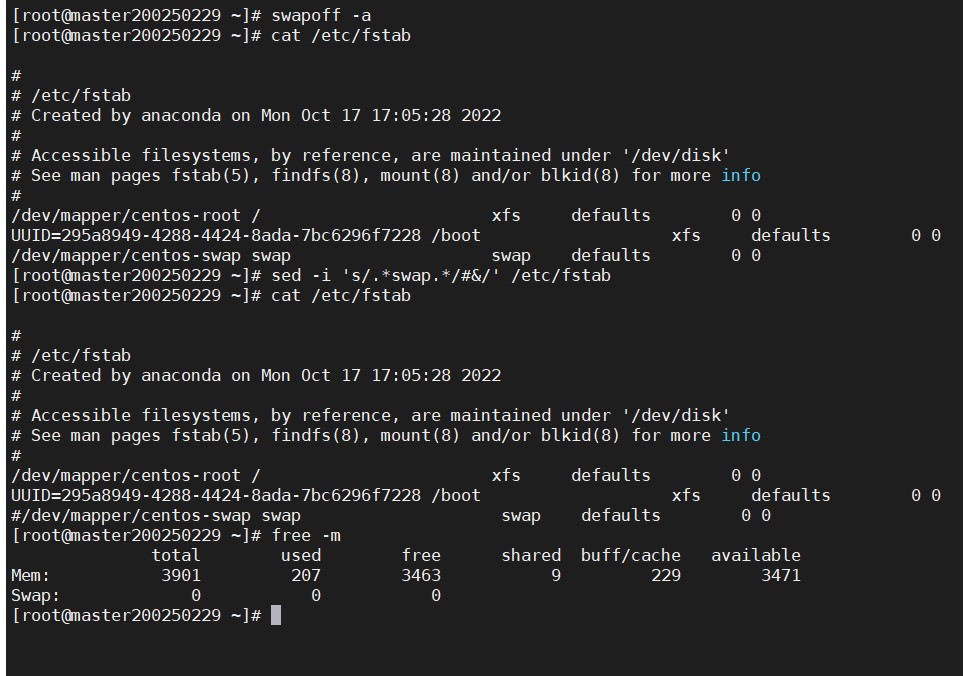
工作节点 2：



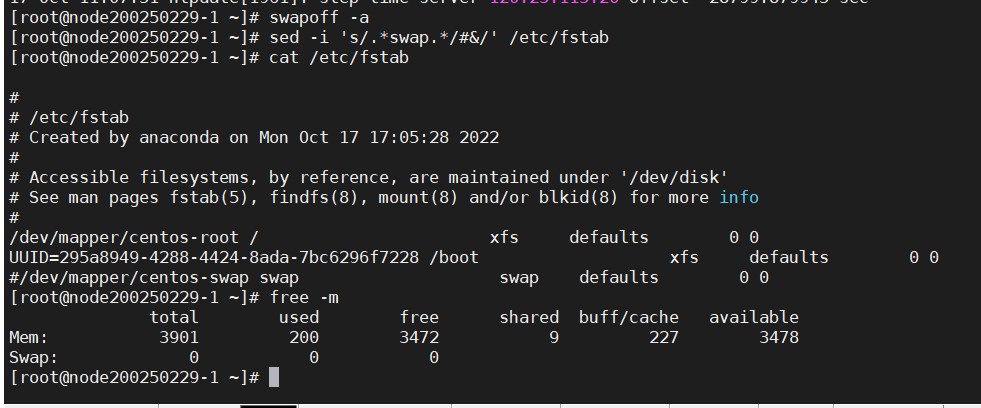
检查系统是否已安装 ntpdate 软件包，若未安装，则可以执行“yum -y install ntpdate”命令进行安装。

(6) 关闭 swap 分区

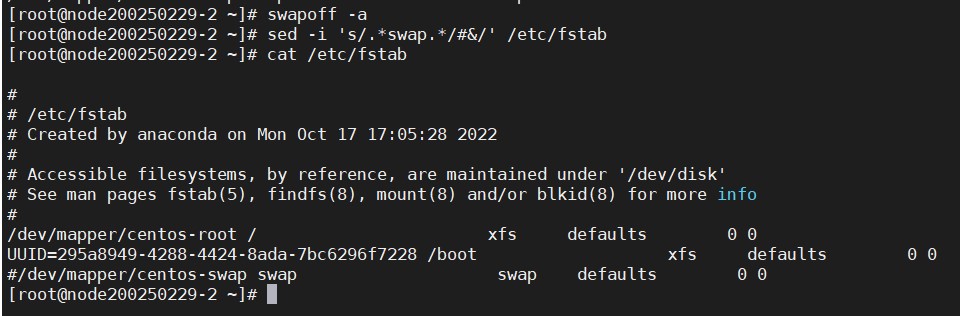
在所有节点上均要关闭 swap。管理节点：



工作节点 1：

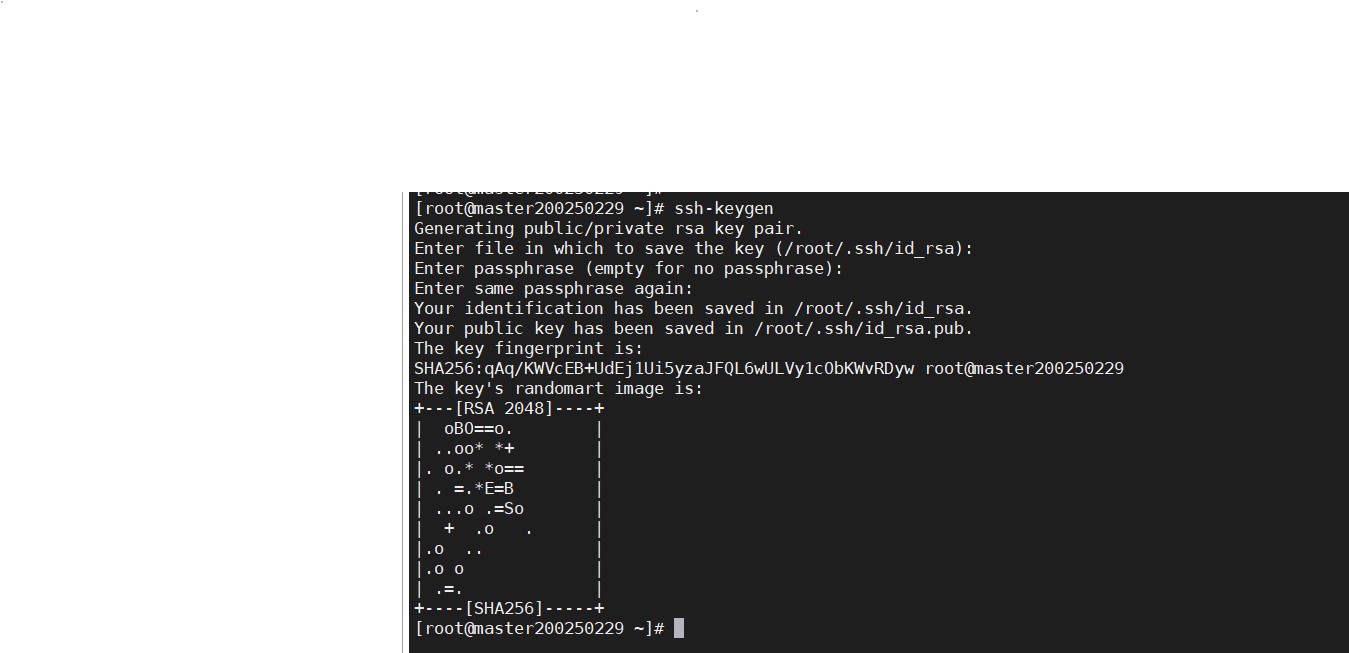


工作节点 2：



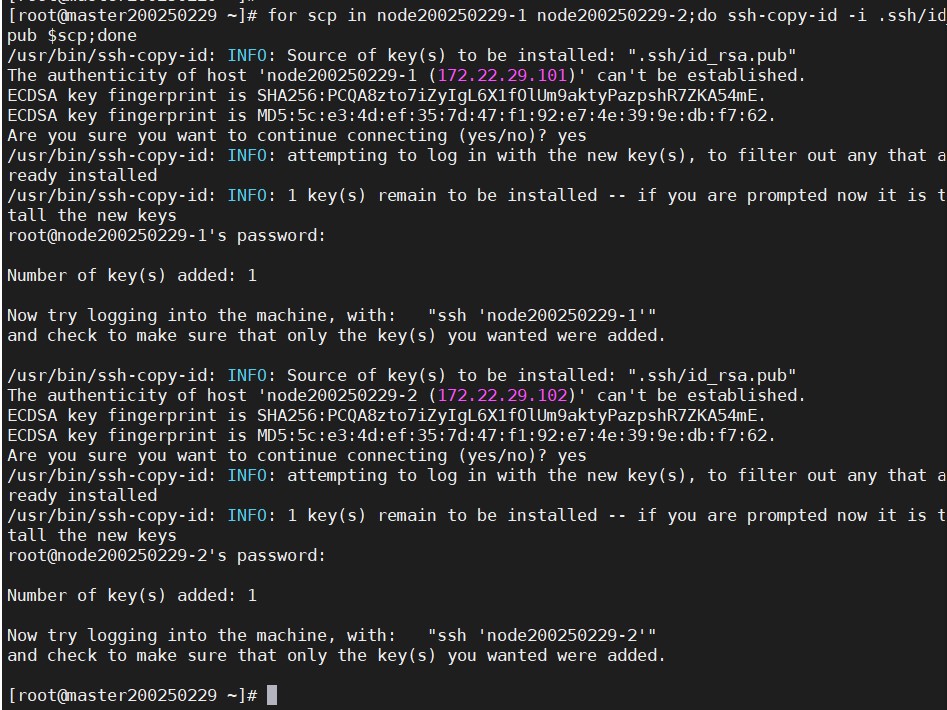
(7) 配置免密登录

在本地主机上使用 ssh-keygen 产生公钥私钥对。



执行 ssh-kengen 命令后会在~/.ssh/目录下生成两个文件：id\_rsa 和 id\_rsa.pub，第一个是私钥文件，第二个是公钥文件。

使用 ssh-copy-id 命令将公钥复制到远程主机上。



(8) 配置内核参数。

开启 IP 转发，并允许 iptables 对 bridge 流量进行处理。

管理节点：

2 Kubectl 常用命令与集群服务

【任务目标】

1. 掌握 Kubectl 工具的基本使用

2. 掌握 Kubectl 常用命令的使用

3. 使用 Kubectl 命令部署集群服务

【任务环境】

1. 虚拟机能够连接外网，并关闭防火墙和 SELinux

2. 虚拟机已安装好 Docker

3. 虚拟机已使用 Rancher 部署 Kubernetes 集群

【相关知识】

1. Kubectl 工具的使用

(1) kubectl get（列出资源）

K8S 把所有的东西都抽象成了资源，而 kubectl get 命令就是用来查看资源的，最常见的资源就是 Pod。

Pod 的概念其实和 Docker 中的容器非常相似。Pod 是 K8S 中的最小工作单位，可以把 Pod 理解成一个一个的小机器人，而 K8S 抽象出来的大资源池就是它们的工厂。

那么 Pod 和 Docker 容器是什么关系呢，简单来说，Pod 将一个或多个 Docker 容器封装成一个统一的整体进行管理并对外提供服务。不仅部署的服务要封装成 Pod，就连 K8S 平台自身也是运行在一堆 Pod 上。

查看一下 K8S 的 Pod 的命令：kubectl get pod -n kube-system

参数“-n”指定了要查看哪个命名空间下的 Pod。K8S 云平台自身所有的 Pod 都被放置在 kube-system命名空间下。K8S 可以用命名空间来做权限控制和资源隔离。如果不指定的话，Pod 将被放置在默认的命名空间 default 下。

执行了 kubectl get pod -n kube-system 命令后可以看到以下内容：

其中每一行就是一个资源，这里看到的资源是 Pod，这个列表里包含了 K8S 在所有节点上运行的 Pod。以下是查询结果中的参数说明：

⚫NAME：Pod 的名字，K8S 可以为 Pod 随机分配一个五位数的后缀。

⚫READY：Pod 中已经就绪的 Docker 容器的数量，Pod 封装了一个或多个 Docker 容器，此处“1/1”的

含义为“就绪 1 个容器/共计 1 个容器”。

⚫STATUS：Pod 的当前状态，常见的状态有 Running、Error、Pending 等。

⚫RESTART：Pod 一共重启了多少次，K8S 可以自动重启 Pod。

⚫AGE：Pod 启动的时间。

kubectl get 可以列出 K8S 中所有资源，这里只介绍了如何用 kubectl 获取 Pod 的列表，还可以获取其它

资源列表信息，如 get svc（查看服务）、get rs（查看副本控制器）、get deploy（查看部署）等。

如果想要查看更多信息，指定-o wide 参数即可，语法如下：# kubectl get <资源> -n <命名空间> -o wide

加上这个参数之后就可以看到资源的 IP 和所在节点。

(2) kubectl describe（查看详情）

kubectl describe 命令可以查看某一资源的具体信息，同样可以使用-n 参数指定资源所在的命名空间。例如，可以用如下命令来查看刚才 Pod 列表中的某个 Pod 的详细信息。

查询结果中可以看到很多的信息，首先是基本属性，可以在详细信息的开头找到。

其中比较常用的，例如 Node、Labels 和 Controlled By。通过 Node 可以快速定位到 Pod 所处的机器，通过 labels 可以检索到该 Pod 的大致用途及定位。而通过 Controlled By 可以知道该 Pod 是由哪种 K8S 资源创建的，然后就可以使用“kubectl get <资源名>”命令继续查找问题。

在中间部分可以找到 Containers 段落，详细描述了 Pod 中每个 Docker 容器的信息，比如 Image 字段，当 Pod 出现 ImagePullBackOff 错误的时候就可以查看该字段，确认拉取的是什么镜像。其他的字段名都很通俗，直接翻译即可。

在 describe 查看详情的时候，最常用的信息获取处就是 Event 段落，可以在介绍内容的末尾找到它。当 Pod 的状态不是 Running 时，这里一定会有或多或少的问题，然后就可以通过其中的信息分析 Pod 出现问题的详细原因了。

(3) kubectl logs（查看日志）

如果要查看一个 Pod 的具体日志，就可以通过“kubectl logs <pod 名>”命令来查看。kubectl logs 只能查看 Pod 的日志。通过添加-f 参数可以持续查看日志。例如，查看 kube-system 命名空间中某个 flannel Pod的日志。

(4) kubectl create（创建资源）

K8S 中的所有东西都可以通过 kubectl create 命令创建，无论是创建一个 Pod 还是一个大型的滚动升级服务 Deployment，create 命令都可以做到。使用 create 生成一个资源主要有两种常用方法，从 YAML 配置文件创建和简易创建。

如果想让 K8S 生成一个和预期一模一样的资源，就要充分而详细地描述这个资源，K8S 可以使用 YAML文件按照 K8S 指定好的结构定义一个对象，然后使用如下方法将该文件传递给 K8S。

# kubectl create -f <配置文件名.yaml>

例如，使用下面的配置文件就可以创建一个最简单的 Pod。[root@master40610932 ~]# vi kubia.yaml

apiVersion: v1 kind: Pod metadata:

name: kubia-manual

spec:

containers:

- image: luksa/kubia name: kubia ports:

- containerPort: 8080

protocol: TCP

然后使用“kubectl create -f kubia.yaml”命令即可创建。

[root@master40610932 ~]# kubectl create -f kubia.yaml

如果配置文件有问题，那么 K8S 就会报错，多数错误大都拼写导致的。使用 YAML 文件相对比较复杂，可以先来学习更简单的简易创建模式。

K8S 为一些常用的资源提供了简易创建的方法，比如 Service、Namespace、Deployment 等，这些方法可以使用“kubectl create <资源类型> <资源名>”方式创建。例如，创建一个名叫 hello-world 的命名空间，直接使用下面命令即可：

[root@master40610932 ~]# kubectl create namespace hello-world

(5) kubectl delete（删除资源）

kubectl delete 命令用于删除资源，语法如下：

# kubectl delete <资源类型> <资源名>

例如，删除一个名为 kubia-manual 的 Pod。[root@master40610932 ~]# kubectl delete pod kubia-manual如果要删除所有的 Pod，命令如下：

# kubectl delete pod --all

如果要删除一切，命令如下：

# kubectl delete all --all

(6) kubectl edit（修改配置）

K8S 每个资源都是通过一个 YAML 配置文件生成的，哪怕是简易创建的资源，也是从一个默认的配置文件创建而来的。可以使用“kubectl get <资源类型> <资源名> -o yaml”命令查看某个现有资源的配置项。例如，查看 kube-proxy-nwj4g 的配置项：

# kubectl get pod kube-proxy-nwj4g -n kube-system -o yaml

执行之后就可以看到一个很长的配置列表，使用 kubectl edit 命令就可以编辑刚才打开的配置列表。

# kubectl edit pod kube-proxy-nwj4g -n kube-system

注意：对于运行中的资源无法修改其名称或类型，可以修改其它属性，例如，将拉取镜像的标签指定

为 latest。

修改完成后输入“:wq”命令保存即可。

使用“kubectl edit <资源类型> <资源名>”命令可以编辑一个资源的具体配置项，edit 命令在实际使用中更偏向于人工修改某个配置项来解决问题。例如，修改镜像地址解决拉取不到镜像的问题。

(7) kubectl apply（应用配置）

kubectl edit 可以对某个资源进行简单快捷的修改，但是如果想对资源进行大范围的修改，就可以用到

kubectl apply 命令。其基本用法如下：

# kubectl apply -f <新配置文件名.yaml>

kubeclt apply 可以理解成 edit 命令的升级版，其最大的区别就是，apply 接受一个 YAML 配置文件，而不是打开一个编辑器去修改。K8S 在接受到这个配置文件后，会根据metadata 中的元数据来查找目标资源，如果没有则直接新建；如果有就依次比对配置文件之间有什么不同点，然后更新并应用不同的配置。

这么做的好处有很多，例如，通过 kubectl apply -f https://some-network-site/resourse.yaml 命令从一个网站上部署了用户的资源，这样当它的管理者更新了这个配置文件后，用户只需要再次执行这个命令，就可以应用更新后的内容了，而不用关心到底修改了哪些配置项。

2. 部署并发布简单应用

一旦运行了 K8S 集群，就可以在其上部署容器化应用程序。为此，需要创建 Deployment 配置，Deployment 指示 K8S 如何创建和更新应用程序。创建 Deployment 后，K8S 调度组件将应用程序实例提到集群中的各个节点上。

创建应用程序实例后，Kubernetes Deployment Controller 会持续监控这些实例。如果托管实例的节点关闭或被删除，则 Deployment 控制器会替换它，K8S 提供了一种自我修复机制来解决机器故障或维护问题，如图 2-1 所示。

图 2-1 K8S 自我修复机制

(1) 创建 Deployment

使用 kubectl create 命令创建一次部署，该部署用于管理 Pod。

kubectl create deployment first-nginx --image=nginx

命令中 first-nginx 为 Deployment 名称，--image 为指定使用的镜像，默认从 Docker Hub 拉取。

(2) 查看 Deployment

创建完成后查看 Deployment 列表。

kubectl get deployment

(3) 查看 Pod

查看 Pod 运行状态。

kubectl get pods

(4) 发布服务

K8S 暴露服务有 3 种方式，分别为 LoadBlancer Service、NodePort Service 和 Ingress。

Ingress 官方定义为管理对外服务到集群内服务之间规则的集合，可以理解为 Ingress 定义规则来允许进入集群的请求被转发到集群中对应服务上，从而实现服务暴露。Ingress 能把集群内 Service 配置成外网能够访问的 URL，流量负载均衡，终止 SSL，提供基于域名访问的虚拟主机等等。

LoadBlancer Service 是 K8S 结合云平台的组件，如国外 GCE、AWS、国内阿里云等等，使用它向使用的底层云平台申请创建负载均衡器来实现，但它有局限性，对于使用云平台的集群比较方便。

NodePort Service 是通过在节点上暴露端口，然后通过将端口映射到具体某个服务上来实现服务暴露，比较直观方便，但是对于集群来说，随着 Service 不断增加，需要的端口越来越多，很容易出现端口冲突，而且不容易管理。当然对于小规模的集群服务，还是比较不错的。

NodePort 服务是引导外部流量服务的最原始方式。NodePort 方式会在所有节点上开放一个特定端口，任何发送到该端口的流量都被转发到对应服务。此处案例采用 NodePort 的方式来暴露 Nginx 服务。

kubectl expose deployment first-nginx --port=80 --type=NodePort查看 Nginx 服务对外暴露的端口。

kubectl get svc

暴露服务后，即可通过“http://任意节点 IP:32629”来访问 Nginx 服务了，如图 2-2 和图 2-3 所示。

图 2-2 通过 Master 节点 IP 访问 Nginx 服务

图 2-3 通过 Node 节点 IP 访问 Nginx 服务

(5) Pod 动态伸缩

在实际生产系统中，经常会遇到某个服务需要扩容的场景，也可能会遇到由于资源紧张或者工作负载

降低而需要减少服务实例数量的场景。此时，可以利用 K8S 的弹性伸缩功能来完成这些任务。

弹性伸缩是指适应负载变化，以弹性可伸缩方式提供资源。反映到 K8S 中，根据负载的高低，动态地调整 Pod 的副本数。目前 K8S 提供了 API 接口实现 Pod 的弹性伸缩，Pod 的副本数本来通过 Replication Controller 进行控制，所以 Pod 的弹性伸缩就是修改 Replication Controller 的 Pod 副本数，可以通过 kubectl scale 命令来完成。

通过执行如下命令将 Nginx Deployment 控制的 Pod 副本数量从初始的 1 更新为 5。

kubectl scale deployment first-nginx --replicas=5

执行 kubectl get pods 命令来验证 Pod 的副本数量是否增加到 5。

kubectl get pods

将--replicas 设置为比当前 Pod 副本数量更小的数字，系统将会“杀掉”一些运行中的 Pod，即可实现应用集群缩容。

kubectl scale deployment first-nginx --replicas=2

kubectl get pods

还可以使用 autoscale 自动设置在 K8S 集群中运行的 pod 数量（水平自动伸缩）。

指定 Deployment、ReplicaSet 或 ReplicationController，并创建已经定义好资源的自动伸缩器。使用自动伸缩器可以根据需要自动增加或减少系统中部署的 Pod 数量。

kubectl autoscale deployment first-nginx --min=1 --max=5

设置后，Pod 的副本数就会根据负载在 1-5 之间自动伸缩。

【任务实施】

**1． 创建集群资源对象**

(1) 通过配置支持 JSON 和 YAML 格式的描述文件，创建一个集群资源对象

(2) 使用 create 命令创建 deployment 资源

(3) 查看 deployment

(4) 查看 pod

kubectl get pod -o wide

(5) 利用 Pod IP 登录 MySQL 服务

(6) 外部使用“Node IP+Port”的形式访问 MySQL 服务

首先，为上述 MySQL 服务创建一个 Service 配置文件。

其次，运行 MySQL Service 服务，并查看该服务的外部端口。

最后，外部使用“Node IP+Port”的形式访问 MySQL 服务。

**2． 使用命令管理资源对象**

(1) 创建 deloyment

(2) 查看 deployment 和 pod 信息

deployment 创建后会自动创建 pod，其名为“deployment 名称-hash 码”。

(3) 删除指定的 pod 和 deployment

从显示结果看到，原 nginx-demo-5dfc44fcdb-nlqcg 已经被删除，而新产生的 nginx-demo-5dfc44fcdb-9sk6r已经运行，这是确保 replicas 为 1 的动作。

(4) 删除 deployment 名为 nginx 的部署

(5) 再次查看 deployment 和 pod 信息

从显示结果可以看到相关的 deployment 和 pod 均被删除。

(6) 查看 k8s 下所有容器状态

(7) 查看 deployment 名为 nginx 的详细信息

**3． 部署 MySQL+Tomcat 服务**

任务：实现运行在 Tomcat 里的 WebApp，JSP 页面通过 JDBC 直接访问 MySQL 数据库并展示数据。

分析：需要部署 WebApp 容器和 MySQL 容器，并把 MySQL 容器的 IP 地址通过环境变量的方式注入

WebApp 容器里，同时，将 WebApp 容器的 8080 端口映射到宿主机的端口，以便在外部访问。

(1) 查看当前可用的 API 版本

(2) 为 MySQL 服务创建一个 Deployment 配置文件

MYSQL\_ROOT\_PASSWORD 密码设置为“123456”，以匹配后续下载的 tomcat-app。

(3) 将 mysql-deploy.yml 配置文件发布到 k8s 集群上运行

查看一下 mysql 服务的 pod。

从显示结果可知，该 mysql 服务运行在 40610932-node2 工作节点，容器 IP 地址为 10.244.2.7。

(4) 为 MySQL 服务创建一个 Service 配置文件

(5) 运行 MySQL 的 Service 服务，并查看该服务的 Cluster IP

从显示结果可知，该 MySQL 服务在集群内部通信 IP 为 10.109.34.117，而 Kubernetes 集群中其他 Pod就可以通过 Service ClusterIP+端口号（例如，登录 MySQL 服务 mysql -h 10.109.34.117 -P 3306 -u root -p）来连接和访问它。根据 Service 的唯一性，容器可以从环境变量中获取 Service 对应的 ClusterIP 地址和端口，从而发起 TCP/IP 连接请求。

验证一下，能否利用 Pod IP（10.244.2.7）登录 MySQL 服务？首先，Ping Pod IP，即 Docker 容器的 IP 地址。

其次，安装 MySQL 命令行工具。

MySQL 命令行工具安装后，就可以测试登录 MySQL Pod 容器了。

思考：是否可以利用 Cluster IP+端口（mysql -h 10.109.34.117 -P 3306 -u root -p）登录 MySQL 服务呢？

(6) 为 Tomcat 服务创建一个 Deployment 配置文件

这里 MYSQL\_SERVICE\_HOST 的值为 MySQL 服务的 Cluster IP。

(7) 将 myweb-deploy.yml 配置文件发布到 k8s 集群上运行

查看一下 myweb 的 pod 情况。

从显示结果可知，该 myweb 服务运行在 node40610932-1 和 node40610932-2 两个工作节点上。

(8) 为 Tomcat 服务创建一个 Service 配置文件

(9) 运行 Tomcat 的 Service 服务，并查看该服务的运行状态

从显示结果可知，myweb 服务对外的端口是 30008。

(10) 使用“http://Node IP:30008”网址访问 Tomcat 服务。例如，使用下列任一网址均可访问 myweb 服务

http://172.16.0.100:30008/demo/ http://172.16.0.101:30008/demo/ http://172.16.0.102:30008/demo/

添加数据前，访问 myweb 服务，如图 2-4 所示。

图 2-4 访问 myweb 服务

点击“Add…”链接，添加测试数据并提交，如图 2-5 所示。

图 2-5 添加测试数据并提交

图 2-6 数据已添加成功

出现“Success add your info”提示后，如图 2-6 所示，点击 return 按钮返回主页面，可以看到新数据

已更新到数据库，如图 2-7 所示。

图 2-7 数据表已更新

(11) 进入 MySQL 服务，确认刚才添加的数据

从显示结果可知，测试数据已成功添加到数据库。

3 Kubernetes 容器云应用部署

【任务目标】

1. 了解 YAML 文件的语法

2. kubectl 常用命令的使用

3. 掌握 Deployment、Pod、Service 的部署

4. 基于 Kubernetes 部署 Wordpress 服务

【任务环境】

1. 虚拟机已安装好 Docker，并关闭防火墙和 SELinux

2. 虚拟机已按要求部署 Kubernetes 集群

3. 虚拟机能够连接外网

【相关知识】

1. kubectl 常用命令

(1) 查看所有 pod 列表，-n 后跟 namespace，查看指定的命名空间

kubectl get pod

kubectl get pod -n namespace

kubectl get pod -o wide

(2) 查看 RC 和 svc（=service）列表，-o wide 查看详细信息

kubectl get rc,svc

kubectl get pod,svc -o wide

kubectl get pod <pod-name> -o yaml

(3) 显示 Node 的详细信息

kubectl describe node <IP>

(4) 显示 pod 的详细信息

kubectl describe pod <pod-name>

(5) 根据 yaml 格式文件创建资源，apply 可以重复执行，create 不可以重复执行

kubectl create -f <yaml 格式文件>

kubectl apply -f <yaml 格式文件>

(6) 删除基于 yaml 格式文件创建的 pod

kubectl delete -f <yaml 格式文件>

(7) 删除所有包含某个 label 的 pod 和 service

kubectl delete pod,svc -l name=<label-name>

(8) 删除所有 pod

kubectl delete pod --all

(9) 通过 bash 获得 pod 中某个容器的 TTY，相当于登录容器

kubectl exec -it <pod-name> -c <container-name> -- bash

(10) 查看容器的日志，-f 实时查看日志

kubectl logs <pod-name> kubectl logs -f <pod-name>

kubectl logs <pod-name> -c <container-name>

若 pod 只有一个容器，可以不加-c。

2. k8s 集群的三种 IP：Node IP、Pod IP、Cluster IP

(1) Node IP，Node 节点的 IP 地址

可以是物理机的 IP（也可能是虚拟机 IP）。每个 Service 都会在 Node 节点上开通一个端口，外部可以通过 Node IP:NodePort（例如 http://172.16.0.101:30080）即可访问 Service 里的 Pod，与访问服务器部署的项目一样（IP:端口/项目名）。

k8s 查询 Node IP，例如

上述显示出来的 INTERNAL-IP 就是 Node IP。

(2) Pod IP，docker 容器的 IP 地址

Pod IP 是 Pod 的 IP 地址，即 docker 容器的 IP 地址，其为虚拟 IP 地址。它是 Docker Engine 根据 docker网桥的 IP 地址段进行分配的，通常是一个虚拟的二层网络。相同 Service 下的 pod 间直接使用 Pod IP 相互通信，不同 Service 下的 pod 在集群之间通信要借助于 Cluster IP，pod 与集群外部通信，要借助于 Node IP。

k8s 查询 Pod IP，例如

上述显示出来的 IP 就是 Pod IP。

(3) Cluster IP，Service 的 IP 地址

Cluster IP 是 Service 的 IP 地址，其为虚拟 IP 地址，但更像是一个伪造的 IP 网络，外部网络无法访问，只许 k8s 集群内部访问使用。Service 地址和 Pod 地址在不同网段，Service 地址为虚拟地址，不配在 pod 上或主机上，外部访问时，先到 Node 节点网络，再转到 Service 网络，最后代理给 Pod 网络。

Cluster IP 由 Kubernetes 管理和分配 P 地址，仅仅作用于 Kubernetes Service 这个对象，无法被 ping 通，它没有一个“实体网络对象”来响应。Cluster IP 只能结合 Service Port 组成一个具体的通信端口，单独的Cluster IP 不具备通信的基础，并且它们属于 Kubernetes 集群这样一个封闭的空间。在不同 Service 下的 pod节点在集群间相互访问可以通过 Cluster IP。

k8s 查询 Cluster IP，例如

上述结果中的 CLUSTER-IP 就是 Cluster IP，-n default 或不指定命名空间，指在默认命名空间 default 下

的操作。

3. YAML 文件

YAML 是专门用来写配置文件的语言，非常简洁和强大，使用比 JSON 更方便。

YAML 语法规则：①大小写敏感。

②使用缩进表示层级关系。

③缩进时不允许使用 Tab 键，只允许使用空格。

④缩进的空格数目不重要，只要相同层级的元素左侧对齐即可。⑤“#”表示注释，从这个字符一直到行尾，都会被解析器忽略。在 Kubernetes 中，只需要知道两种结构类型即可：

①Lists

②Maps

使用 YAML 用于 K8S 的定义带来的好处包括：

①便捷性：不必添加大量的参数到命令行中执行命令。

②可维护性：YAML 文件可以通过源头控制，跟踪每次操作。③灵活性：YAML 文件可以创建比命令行更加复杂的结构。

(1) YAML Maps

Map 顾名思义指的是字典，即一个 Key:Value 的键值对信息。例如：---

apiVersion: v1

kind: Pod

注意：---为可选的分隔符，当需要在一个文件中定义多个结构的时候需要使用。上述内容表示有两个

键 apiVersion 和 kind，分别对应的值为 v1 和 Pod。

Maps 的 value 既能够对应字符串也能够对应一个 Maps。例如：---

apiVersion: v1 kind: Pod metadata:

name: kube100-site labels:

app: web

上述的 YAML 文件中，metadata 这个 KEY 对应的值为一个 Maps，而嵌套的 labels 这个 KEY 的值又是一个 Map。实际使用中可视情况进行多层嵌套。

YAML 处理器根据行缩进来知道内容之间的关联。上述例子中，使用两个空格作为缩进，但空格的数量并不重要，只是要求至少一个空格并且所有缩进保持一致。例如 name 和 labels 是相同缩进级别，因此 YAML处理器知道它们属于同一 map；app 是 lables 的值，因为 app 的缩进量比 labels 更大。

(2) YAML Lists

List 即列表，类似于数组，例如：args

- beijing

- shanghai

- shenzhen

- guangzhou

可以指定任何数量的项在列表中，每个项的定义以破折号“-”开头，并且与父元素之间存在缩进。在

JSON 格式中，表示如下：

{

"args": ["beijing", "shanghai", "shenzhen", "guangzhou"]

}

当然 Lists 的子项也可以是 Maps，Maps 的子项也可以是 List，例如：---

apiVersion: v1 kind: Pod metadata:

name: kube100-site

labels:

app: web

spec:

containers:

- name: front-end image: nginx ports:

- containerPort: 80

- name: flaskapp-demo image: jcdemo/flaskapp ports: 8080

如上述文件所示，定义一个 containers 的 List 对象，每个子项都由 name、image、ports 组成，每个 ports都有一个 KEY 为 containerPort 的 Map 组成

(3) 创建 Pod

创建 YAML 文件。例如：

vi test.yaml # api 版本

apiVersion: v1

# 指定创建资源对象kind: Pod

# 源数据、可以写 name，命名空间，对象标签

metadata:

# 服务名称

name: pod-test # 标签

labels:

# 标签名

os: centos

# 容器资源信息

spec:

# 容器管理

containers:

# 容器名称

- name: hello # 容器镜像

image: nginx:latest # 添加环境变量

env:

# 创建 key

- name: Test # 创建 value

value: "123456" # 启动容器后执行命令

command: ["bash","-c","while true;do date;sleep 1;done"]通过 YAML 文件创建管理 Pod，如下：

kubectl create -f test.yaml kubectl get pods

【任务实施】

**1． 在 Kubernetes 上部署 WordPress**

在 Kubernetes 上运行 WordPress 是一个可伸缩性服务运行于云原生集群的典型案例。首先是安装非常简单（在已有集群的情况下），其次是可靠性更高，第三是规模可以伸缩。当然可以在多个云之间更为容易地迁移也是非常重要的一点。

(1) 编写 YAML 文件

编写 YAML 文件 wordpress.yaml。

这里添加一个 label 标签，pod 向外暴露 pod 的 service NodePort 型端口需要用标签选择器。labels:

app: wordpress

这里不能是 localhost，数据库的 localhost 默认是去找 socket 文件，然而在不同的容器之间文件系统是隔离的，会导致报错。

env:

- name: WORDPRESS\_DB\_HOST

value: 127.0.0.1:3306

注意：这里针对 MySQL 这个容器做了一个数据卷的挂载，这是为了能够将 MySQL 的数据能够持久化

到节点上，这样下次 MySQL 容器重启过后数据不至于丢失。

(2) 创建 Pod

(3) 访问服务

查看 svc

可以看到 WordPress 服务产生了一个 30276 的端口，现在通过任意节点的 Node IP 加上 30276 端口，就可以访问 WordPress 应用了，如图 3-1、图 3-2 所示。

图 3-1 通过任意节点访问 WordPress 应用

图 3-2 WordPress 仪表盘

**2． 优化应用（选做）**

(1) 增加健康检测

livenessProbe 和 redinessProbe 是提高应用稳定性非常重要的方法。

livenessProbe:

tcpSocket: #通过端口探测的方式

port: 80 initialDelaySeconds: 3 periodSeconds: 3

readinessProbe:

tcpSocket:

#通过端口探测的方式

port: 80 initialDelaySeconds: 5 periodSeconds: 10

增加上面两个探针，每 10s 检测一次应用是否可读，每 3s 检测一次应用是否存活。

(2) 增加 Pod 资源限制

配置 HPA，让应用能够自动应对流量高峰期。resources:

limits:

cpu: 200m memory: 200Mi

requests:

cpu: 100m

memory: 100Mi

(3) 增加滚动更新策略

增加滚动更新策略，这样可以保证在更新应用的时候，服务不会被中断。

revisionHistoryLimit: 10 minReadySeconds: 5 strategy:

type: RollingUpdate rollingUpdate:

maxSurge: 1

maxUnavailable: 1

(4) 修改 MySQL 主机值

如果 MySQL 服务被重新创建了的话，它的 clusterIP 非常有可能就变化了，所以上面环境变量中的WORDPRESS\_DB\_HOST 的值就会有问题，会导致访问不了数据库服务，这个地方可以直接使用 Service 名称来代替 host，这样即使 clusterIP 变化了，也不会有任何影响。

env:

- name: WORDPRESS\_DB\_HOST

value: mysql:3306

(5) 添加 initcontainer

在部署 Wordpress 服务的时候，看看 MySQL 服务是否已启动起来了。如果没有启动起来，就无法连接数据库。此时在启动 Wordpress 应用之前，先检查一下 MySQL 服务，如果服务正常的话，就可以开始部署应用。

initContainers:

- name: init-db

image: busybox

command: ['sh', '-c', 'until nslookup mysql; do echo waiting for mysql service; sleep 2; done;']

直到 MySQL 服务创建完成后，initContainer 才结束，结束完成后开始下面的部署。

(6) 重新部署服务

把上述优化应用重新整合到 wordpress.yaml 文件中，重新部署服务。

kubectl create -f wordpress-all.yaml

(7) 配置 HPA

kubectl autoscale deployment wordpress-deploy --cpu-percent=10 --min=1 --max=10 -n blog

用 kubectl autoscale 命令为 wordpress-deploy 创建一个 HPA 对象，最小的 Pod 副本数为 1，最大为 10，

HPA 会根据设定的 CPU 使用率（10%）动态的增加或者减少 Pod 数量。

(8) 访问服务

在浏览器中输入 http://任意 Node\_IP:30276 来访问服务，如图 3-3 所示。

图 3-3 Wordpress 登录界面

至此，Wordpress 应用的优化就完成了。