

云计算导论实践----容器和 kubernetes 使用实践

1.前言

Kubernetes 单词起源于希腊语,是“舵手”或者“领航员”的意思,是“管理者”和“控制论”的根源。K8s 是把用 8 代替 8 个字符“ubernete”而成的缩写。首先,他是一个全新的基于容器技术的分布式架构领先方案。Kubernetes(k8s)是 Google 开源的容器集群管理系统(谷歌内部:Borg)。在 Docker 技术的基础上,为容器化的应用提供部署运行、资源调度、服务发现和动态伸缩等一系列完整功能,提高了大规模容器集群管理的便捷性。

Kubernetes 是一个完备的分布式系统支撑平台,具有完备的集群管理能力,多扩多层次的安全防护和准入机制、多租户应用支撑能力、透明的服务注册和发现机制、内建智能负载均衡器、强大的故障发现和自我修复能力、服务滚动升级和在线扩容能力、可扩展的资源自动调度机制以及多粒度的资源配额管理能力。

1.1 简介

Kubernetes 是一个全新的基于容器技术的分布式架构领先方案。是 Google 内部集群管理系统 Borg 的一个开源版本。直到 2015 年 4 月,随着论文发布,才被众人熟知。Kubernetes 是一个开放的开发平台。不局限于任何一种语言,没有限定任何编程接口。是一个完备的分布式系统支撑平台。它构建在 docker 之上,提供应用部署、维护、扩展机制等功能,利用 Kubernetes 能方便地管理跨机器运行容器化的应用。

1.2 Kubernetes 特征

- (1) 自主的管理容器,保证云平台中的容器按照用户的期望状态运行着。
- (2) 自动扩容,弹性伸缩。
- (3) 自动监控,删除出故障的应用。
- (4) 容器编排成组,并提供容器间的负载均衡。
- (5) 自我调度,自我管理。

1.3 核心组件

- (1) kubectl:客户端命令行工具,作为整个系统的操作入口。
- (2) kube-apiserver:以 RESTAPI 服务形式提供接口,作为整个系统的控制入口。
- (3) kube-controller-manager:执行整个系统的后台任务,包括节点状态状况、Pod 个数、Pods 和 Service 的关联等。
- (4) kube-scheduler:负责节点资源管理,接收来自 kube-apiserver 创建 Pods 任务,并分配到某个节点。
- (5) etcd:负责节点间的服务发现和配置共享。

(6) kube-proxy:运行在每个计算节点上，负责 Pod 网络代理。定时从 etcd 获取到 service 信息来做相应的策略。

(7) kubelet:运行在每个计算节点上，作为 agent，接收分配该节点的 Pods 任务及管理容器，周期性获取容器状态，反馈给 kube-apiserver。

1.4Kubernetes 解决的核心问题

容器是打包和运行应用程序的好方式。在生产环境中，你需要管理运行着应用程序的容器，并确保服务不会下线。例如，如果一个容器发生故障，则你需要启动另一个容器。如果此行为交由给系统处理，是不是会更容易一些？

这就是 Kubernetes 要来做的事情！Kubernetes 为你提供了一个可弹性运行分布式系统的框架。Kubernetes 会满足你的扩展要求、故障转移你的应用、提供部署模式等。例如，Kubernetes 可以轻松管理系统的 Canary(金丝雀)部署。

Kubernetes 为你提供：

(1) 服务发现和负载均衡

Kubernetes 可以使用 DNS 名称或自己的 IP 地址来暴露容器。如果进入容器的流量很大，Kubernetes 可以负载均衡并分配网络流量，从而使部署稳定。

(2) 存储编排

Kubernetes 允许你自动挂载你选择的存储系统，例如本地存储、公共云提供商等。

(3) 自动部署和回滚

你可以使用 Kubernetes 描述已部署容器的所需状态，它可以以受控的速率将实际状态更改为期望状态。例如，你可以自动化 Kubernetes 来为你的部署创建新容器，删除现有容器并将它们的所有资源用于新容器。

(4) 自动完成装箱计算

你为 Kubernetes 提供许多节点组成的集群，在这个集群上运行容器化的任务。你告诉 Kubernetes 每个容器需要多少 CPU 和内存(RAM)。Kubernetes 可以将这些容器按实际情况调度到你的节点上，以最佳方式利用你的资源。

(5) 自我修复

Kubernetes 将重新启动失败的容器、替换容器、杀死不响应用户定义的运行状况检查的容器，并且在准备好服务之前不将其通告给客户端。

(6) 密钥与配置管理

Kubernetes 允许你存储和管理敏感信息，例如密码、OAuth 令牌和 SSH 密钥。你可以在不重建容器镜像的情况下部署和更新密钥和应用程序配置，也无需在堆栈配置中暴露密钥。

1.5Kubernetes 不是什么

Kubernetes 不是传统的、包罗万象的 PaaS（平台即服务）系统。由于 Kubernetes 是

在容器级别运行，而非在硬件级别，它提供了 PaaS 产品共有的一些普遍适用的功能，例如部署、扩展、负载均衡，允许用户集成他们的日志记录、监控和警报方案。但是，Kubernetes 不是单体式（monolithic）系统，那些默认解决方案都是可选、可插拔的。Kubernetes 为构建开发人员平台提供了基础，但是在重要的地方保留了用户选择权，能有更高的灵活性。

Kubernetes:

(1) 不限制支持的应用程序类型。Kubernetes 旨在支持极其多种多样的工作负载，包括无状态、有状态和数据处理工作负载。如果应用程序可以在容器中运行，那么它应该可以在 Kubernetes 上很好地运行。

(2) 不部署源代码，也不构建你的应用程序。持续集成（CI）、交付和部署（CI/CD）工作流程取决于组织的文化和偏好以及技术要求。

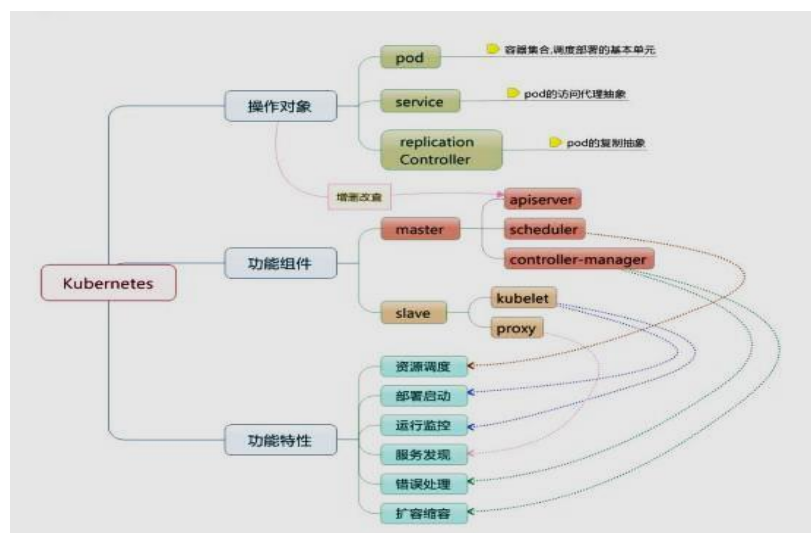
(3) 不提供应用程序级别的服务作为内置服务，例如中间件（例如消息中间件）、数据处理框架（例如 Spark）、数据库（例如 MySQL）、缓存、集群存储系统（例如 Ceph）。这样的组件可以在 Kubernetes 上运行，并且/或者可以由运行在 Kubernetes 上的应用程序通过可移植机制（例如开放服务代理）来访问。

(4) 不是日志记录、监视或警报的解决方案。它集成了一些功能作为概念证明，并提供了收集和导出指标的机制。

(5) 不提供也不要求配置用的语言、系统（例如 jsonnet），它提供了声明性 API，该声明性 API 可以由任意形式的声明性规范所构成。

(6) 此外，Kubernetes 不仅仅是一个编排系统，实际上它消除了编排的需要。编排的技术定义是执行已定义的工作流程：首先执行 A，然后执行 B，再执行 C。而 Kubernetes 包含了一组独立可组合的控制过程，可以持续地将当前状态驱动到所提供的预期状态。你不需要在乎如何从 A 移动到 C，也不需要集中控制，这使得系统更易于使用且功能更强大、系统更健壮，更为弹性和可扩展。

1.6 概览拓扑图



2.服务器购买

在华为云中购买两台按需计费的云服务器，相关配置如下：

用户名以及登陆密码均为:root:admin@1234

< k8s-master	< k8s-node																																				
<div>云服务器信息</div> <table><tr><td>ID</td><td>056e4452-b3a6-490a-ab49-f24de0866790</td></tr><tr><td>名称</td><td>k8s-master</td></tr><tr><td>描述</td><td>--</td></tr><tr><td>区域</td><td>华北-北京四</td></tr><tr><td>可用区</td><td>可用区1</td></tr><tr><td>规格</td><td>通用入门型 2vCPUs 4GiB t6.large.2</td></tr><tr><td>镜像</td><td>CentOS 7.9 64bit 公共镜像</td></tr><tr><td>虚拟私有云</td><td>vpc-default</td></tr><tr><td>全域弹性公网IP</td><td>-- 绑定</td></tr></table>	ID	056e4452-b3a6-490a-ab49-f24de0866790	名称	k8s-master	描述	--	区域	华北-北京四	可用区	可用区1	规格	通用入门型 2vCPUs 4GiB t6.large.2	镜像	CentOS 7.9 64bit 公共镜像	虚拟私有云	vpc-default	全域弹性公网IP	-- 绑定	<div>云服务器信息</div> <table><tr><td>ID</td><td>e86f7445-bf57-44dc-84a5-1250b95f80c1</td></tr><tr><td>名称</td><td>k8s-node</td></tr><tr><td>描述</td><td>--</td></tr><tr><td>区域</td><td>华北-北京四</td></tr><tr><td>可用区</td><td>可用区1</td></tr><tr><td>规格</td><td>通用入门型 2vCPUs 4GiB t6.large.2</td></tr><tr><td>镜像</td><td>CentOS 7.9 64bit 公共镜像</td></tr><tr><td>虚拟私有云</td><td>vpc-default</td></tr><tr><td>全域弹性公网IP</td><td>-- 绑定</td></tr></table>	ID	e86f7445-bf57-44dc-84a5-1250b95f80c1	名称	k8s-node	描述	--	区域	华北-北京四	可用区	可用区1	规格	通用入门型 2vCPUs 4GiB t6.large.2	镜像	CentOS 7.9 64bit 公共镜像	虚拟私有云	vpc-default	全域弹性公网IP	-- 绑定
ID	056e4452-b3a6-490a-ab49-f24de0866790																																				
名称	k8s-master																																				
描述	--																																				
区域	华北-北京四																																				
可用区	可用区1																																				
规格	通用入门型 2vCPUs 4GiB t6.large.2																																				
镜像	CentOS 7.9 64bit 公共镜像																																				
虚拟私有云	vpc-default																																				
全域弹性公网IP	-- 绑定																																				
ID	e86f7445-bf57-44dc-84a5-1250b95f80c1																																				
名称	k8s-node																																				
描述	--																																				
区域	华北-北京四																																				
可用区	可用区1																																				
规格	通用入门型 2vCPUs 4GiB t6.large.2																																				
镜像	CentOS 7.9 64bit 公共镜像																																				
虚拟私有云	vpc-default																																				
全域弹性公网IP	-- 绑定																																				
<div>计费模式</div> <table><tr><td>计费模式</td><td>按需计费</td></tr><tr><td>创建时间</td><td>2025/06/01 13:20:08 GMT+08:00</td></tr><tr><td>启动时间</td><td>2025/06/01 13:20:29 GMT+08:00</td></tr><tr><td>定时删除时间</td><td>-- 修改</td></tr></table>	计费模式	按需计费	创建时间	2025/06/01 13:20:08 GMT+08:00	启动时间	2025/06/01 13:20:29 GMT+08:00	定时删除时间	-- 修改	<div>计费模式</div> <table><tr><td>计费模式</td><td>按需计费</td></tr><tr><td>创建时间</td><td>2025/06/01 13:21:21 GMT+08:00</td></tr><tr><td>启动时间</td><td>2025/06/01 13:21:42 GMT+08:00</td></tr><tr><td>定时删除时间</td><td>-- 修改</td></tr></table>	计费模式	按需计费	创建时间	2025/06/01 13:21:21 GMT+08:00	启动时间	2025/06/01 13:21:42 GMT+08:00	定时删除时间	-- 修改																				
计费模式	按需计费																																				
创建时间	2025/06/01 13:20:08 GMT+08:00																																				
启动时间	2025/06/01 13:20:29 GMT+08:00																																				
定时删除时间	-- 修改																																				
计费模式	按需计费																																				
创建时间	2025/06/01 13:21:21 GMT+08:00																																				
启动时间	2025/06/01 13:21:42 GMT+08:00																																				
定时删除时间	-- 修改																																				

图1.k8s-master服务器配置图

图2.k8s-node服务器配置图

3.集群部署

3.1 安装 docker

分别在k8s-master和k8s-node两台主机进行如下操作。

3.1.1添加阿里源

使用如下命令安装yum-utils，其中包含yum-config-manager，后续需要利用该工具进行阿里源的添加。

```
yum -y install yum-utils
```

```
[root@k8s-master ~]# yum -y install yum-utils
Loaded plugins: fastestmirror
Determining fastest mirrors
base                                     | 3.6 kB  00:00:00
epel                                    | 4.3 kB  00:00:00
extras                                 | 2.9 kB  00:00:00
updates                                | 2.9 kB  00:00:00
(1/7): base/7/x86_64/group_gz         | 153 kB  00:00:00
(2/7): base/7/x86_64/primary_db       | 6.1 MB  00:00:00
(3/7): epel/x86_64/group              | 399 kB  00:00:00
(4/7): epel/x86_64/updateinfo         | 1.0 MB  00:00:00
(5/7): epel/x86_64/primary_db        | 8.7 MB  00:00:00
(6/7): extras/7/x86_64/primary_db     | 253 kB  00:00:00
(7/7): updates/7/x86_64/primary_db    | 27 MB  00:00:07
```

图3.yum-utils安装示意图

使用如下命令为云服务器添加阿里源，并生成缓存。

```
sudo yum-config-manager --add-repo
http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
```

```
[root@k8s-master ~]# sudo yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
Loaded plugins: fastestmirror
adding repo from: http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
grabbing file http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo to /etc/yum.repos.d/docker-ce.repo
repo saved to /etc/yum.repos.d/docker-ce.repo
```

图4.添加阿里源示意图

```
yum makecache fast
```

```
[root@k8s-master ~]# yum makecache fast
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
base | 3.6 kB | 00:00:00
docker-ce-stable | 3.5 kB | 00:00:00
epel | 4.3 kB | 00:00:00
extras | 2.9 kB | 00:00:00
updates | 2.9 kB | 00:00:00
(1/2): docker-ce-stable/7/x86_64/updateinfo | 55 B | 00:00:00
(2/2): docker-ce-stable/7/x86_64/primary_db | 152 kB | 00:00:00
Metadata Cache Created
```

图5.生成缓存示意图

```
yum -y install docker-ce-18.06.3.ce-3.el7
```

```
[root@k8s-master ~]# yum -y install docker-ce-18.06.3.ce-3.el7
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
Resolving Dependencies
--> Running transaction check
--> Package docker-ce.x86_64 0:18.06.3.ce-3.el7 will be installed
--> Processing Dependency: container-selinux >= 2.9 for package: docker-ce-18.06.3.ce-3.el7.x86_64
--> Processing Dependency: libcgroup for package: docker-ce-18.06.3.ce-3.el7.x86_64
--> Processing Dependency: libltdl.so.7()(64bit) for package: docker-ce-18.06.3.ce-3.el7.x86_64
--> Running transaction check
--> Package container-selinux.noarch 2:2.119.2-1.911c772.el7_8 will be installed
--> Processing Dependency: policycoreutils-python for package: 2:container-selinux-2.119.2-1.911c772.el7_8.noarch
--> Package libcgroup.x86_64 0:0.41-21.el7 will be installed
--> Package libtool-ltdl.x86_64 0:2.4.2-22.el7_3 will be installed
--> Running transaction check
```

图6.docker-ce安装示意图

通过执行如下指令可以检测docker是否成功安装，如果成功执行如下指令并返回docker的版本信息，则完成安装。

```
docker -v
```

```
[root@k8s-master ~]# docker -v
Docker version 18.06.3-ce, build d7080c1
```

图7.docker版本信息图

安装完成后的docker默认是没有启动的，因此我们需要启动docker，并且为了后续使用的方便我们设置docker为开机自启。

```
systemctl start docker && systemctl enable docker
```

```
[root@k8s-master ~]# systemctl start docker && systemctl enable docker
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to /usr/lib/systemd/system/docker.service.
```

图8.docker启动及设置自启示意图

3.2 安装集群组件

为后续方便主机识别，我们为两台主机的/etc/hosts文件添加如下内容，完成主机名到ip地址的对应。

```
vim /etc/hosts
```

使用vim打开文本文件后，添加如下信息：

```
vim /etc/hosts192.168.2.5 k8s-master
192.168.2.6 k8s-node
```

```

::1      localhost      localhost.localdomain  localhost6      localhost6.localdomain6
127.0.0.1    localhost      localhost.localdomain  localhost4      localhost4.localdomain4
127.0.0.1    k8s-master      k8s-master
192.168.2.5 k8s-master
192.168.2.6 k8s-node

```

图9./etc/hosts修改示意图

分别在不同主机上设置hostname。

```

hostnamectl --static set-hostname k8s-master
hostnamectl --static set-hostname k8s-node

```

关掉selinux。

```

setenforce 0
sed -i "s/^SELINUX=enforcing/SELINUX=disabled/g" /etc/sysconfig/selinux

```

```

[root@k8s-master ~]# setenforce 0
setenforce: SELinux is disabled
[root@k8s-master ~]# sed -i "s/^SELINUX=enforcing/SELINUX=disabled/g" /etc/sysconfig/selinux

```

图10.关闭selinux示意图

为了防止防火墙对后续实验造成影响，我们需要事先关闭防火墙。

```

systemctl stop firewalld
systemctl disable firewalld

```

```

[root@k8s-master ~]# systemctl stop firewalld
[root@k8s-master ~]# systemctl disable firewalld

```

图11.关闭防火墙示意图

k8s相关组件的使用需要关闭swap。

```

swapoff -a
sed -i 's/.*swap.*/#&/' /etc/fstab

```

```

[root@k8s-master ~]# swapoff -a
[root@k8s-master ~]# sed -i 's/.*swap.*/#&/' /etc/fstab

```

图12.关闭swap示意图

创建/etc/sysctl.d/k8s.conf文件，并向其写入相关内容，配置转发参数。

```

cat > /etc/sysctl.d/k8s.conf << EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF

```

```

sysctl --system

```

```

[root@k8s-master ~]# cat > /etc/sysctl.d/k8s.conf << EOF
> net.bridge.bridge-nf-call-ip6tables = 1
> net.bridge.bridge-nf-call-iptables = 1
> EOF

```

图13.修改k8s.conf文件示意图

```

[root@k8s-master ~]# sysctl --system
* Applying /usr/lib/sysctl.d/00-system.conf ...
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
* Applying /usr/lib/sysctl.d/10-default-yama-scope.conf ...
kernel.yama.ptrace_scope = 0
* Applying /usr/lib/sysctl.d/50-default.conf ...
kernel.sysrq = 16
kernel.core_uses_pid = 1

```

图14.写入内核参数示意图

为k8s配置国内yum源并生成缓存。

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

```
[root@k8s-master ~]# yum makecache fast
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
base | 3.6 kB 00:00:00
docker-ce-stable | 3.5 kB 00:00:00
epel | 4.3 kB 00:00:00
extras | 2.9 kB 00:00:00
kubernetes/signature | 454 B 00:00:00
Retrieving key from https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg
Importing GPG key 0x13EDEF05:
  Userid : "Rapture Automatic Signing Key (cloud-rapture-signing-key-2022-03-07-08_01_01.pub)"
  Fingerprint: a362 b822 f6de dc65 2817 ea46 b53d c80d 13ed ef05
  From : https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg
Is this ok [y/N]: y
kubernetes/signature | 1.4 kB 00:00:16 !!!
updates | 2.9 kB 00:00:00
kubernetes/primary | 137 kB 00:00:00
kubernetes | 1022/1022
Metadata Cache Created
```

图15.修改kubernetes.repo示意图

```
[root@k8s-master ~]# cat > /etc/yum.repos.d/kubernetes.repo << EOF
> [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
```

图16.生成缓存示意图

安装kubeadm,kubect1,kubelet,ntpd并设置开机自启，值得注意的是，这里所提及的kubeadm、kubect1以及kubelet和docker一样均有较为严格的版本限制。

```
yum install -y kubeadm-1.15.0 kubect1-1.15.0 kubelet-1.15.0 ntpdate
```

3.3 使用 kubeadm 初始化集群

在k8s-master上面执行一下命令，初始化集群信息。上文说到docker和k8s相关组件的安装均对版本由较为严格的限制，是由于高版本的docker以及k8s相关组件存在不匹配的问题，在这一步初始化的过程中会发生报错。并且k8s与

docker之间还有这严格的版本匹配关系。

```
Kubeadm init --image-repository registry.aliyuncs.com/google_containers
--pod-network-cidr=10.244.0.0/ 16 --service-cidr=10.96.0.0/ 12
```

```
To start using your cluster, you need to run the following as a regular user:

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

You should now deploy a pod network to the cluster.
Run 'kubectl apply -f [podnetwork].yaml' with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each on root:

kubeadm join 192.168.2.5:6443 --token 28bqru.ipbhyabx9pwfa1b2 \
--discovery-token-ca-cert-hash sha256:a89db5f916d3c0ad50d52e40fc5685201c2ae0286b11a05a2c80924a0c22b62c
[root@k8s-master ~]# mkdir -p $HOME/.kube
[root@k8s-master ~]# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
[root@k8s-master ~]# sudo chown $(id -u):$(id -g) $HOME/.kube/config
[root@k8s-master ~]#
```

图 17. k8s 初始化示意图

成功初始化以后，我们按照提示执行如下三步。

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

对于初始化信息中的如下部分需要妥善保存，这是k8s-node结点加入集群的凭证。

```
kubeadm join 192.168.2.5:6443 --token 28bqru.ipbhyabx9pwfa1b2 \
--discovery-token-ca-cert-hash
sha256:a89db5f916d3c0ad50d52e40fc5685201c2ae0286b11a05a2c80924a0c22b62c
```

如果希望子当以集群中的结点在集群中的ip地址，则需要加入flannel网络插件，拥有网络插件后，结点的ip地址所处网段将由k8s初始化命令中的--pod-network-cidr参数来决定。

```
kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

```
[root@k8s-master ~]# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
namespace/kube-flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
```

图18.加入flannel示意图

3.4 加入集群

在k8s-node结点中执行k8s-master初始化时生成的如下内容即可

```
kubeadm join 192.168.2.5:6443 --token 28bqru.ipbhyabx9pwfa1b2 \
--discovery-token-ca-cert-hash
sha256:a89db5f916d3c0ad50d52e40fc5685201c2ae0286b11a05a2c80924a0c22b62c
```



```
[root@k8s-node ~]# kubectl join 102.160.2.610443 --token 2nbgru.lpbhyas0pufa1b1 --discovery-token-cs-cert-hash sha355:a50xb5701d1cbad5ba31u46tc
6685201c2e80286t11a05a2e08924a0c72062c
[ecoflight] Running pre-flight checks
IMADNCH9 IaDocher3yateadCheck1: detected "ogroupfa" as the Docker cgroups driver. The recommended driver is 'systemd'. Please follow the guide
at https://kubernetes.io/docs/setup/eri/
[ecoflight] Beating configuration from the cluster...
[ecoflight] FYI: You can look at this config file with 'kubectl -n kube-system get configmap kubelet-config -o yaml'
[kubelet-start] Overwriting configuration for the kubelet from the "kubelet-config-1.15" ConfigMap in the kube-system namespace
[kubelet-start] Writing nodelet configuration to file "/var/lib/kubelet/config.json"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubelet-flags.env"
[kubelet-start] Activating the kubelet service
[kubelet-start] Waiting for the kubelet to perform the TLS bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiclient and a response was received.
* The kubelet has informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

[root@k8s-node ~]#
```

图19.加入集群示意图

4. 集群验证

4.1 验证集群节点

当我们完成集群加入工作时，集群不会马上变成ready状态而是处于unready，因为系统需要去下载docker镜像，稍等片刻后我们可以执行如下命令验证集群状态。

```
kubectl get node 或者 kubectl get node -o wide
```

如果集群节点长时间处于unready状态，说明出现了问题，可以使用如下命令来查看集群状态。

```
journalctl -xeu kubelet
```

如果状态反馈中存在如下内容，则可以参见参考资料2进行处理。出现该问题的主要原因是我们所安装的flannel插件没办法很好地进行工作，因此依据参考资料修改k8s的相关设置，使其不使用flannel进行网络配置即可，此时各节点在集群中的ip地址将处于docker网卡所处的网段，而非我们自定义的网段。

```
kubelet, k8s-master network is not ready: runtime network not ready:
```

```
NetworkReady=false
```

```
reason:NetworkPluginNotReady message:docker: network plugin is not ready:
```

```
cni config uninitialized
```

当一切准备完毕后，所有节点的状态将转变为Ready，此时表示集群搭建完成。

```
[root@k8s-master ~]# kubectl get node
NAME           STATUS    ROLES    AGE   VERSION
k8s-master     Ready     master   66m   v1.15.0
k8s-node       Ready     <none>   51m   v1.15.0
```

图20.集群状态示意图

4.2 验证 pod 信息

执行如下命令，如果所有pod的ready都变成1/1就表示集群搭建成功。

```
kubectl get pod -n kube-system
```

```
[root@k8s-master ~]# kubectl get pod -n kube-system
NAME                                READY   STATUS    RESTARTS   AGE
coredns-78d4cf999f-4r49q           1/1     Running   2           32m
coredns-78d4cf999f-q9h7s           1/1     Running   2           32m
etcd-k8s-master                     1/1     Running   2           32m
kube-apiserver-k8s-master           1/1     Running   2           32m
kube-controller-manager-k8s-master 1/1     Running   2           32m
kube-flannel-ds-amd64-rgpqq         1/1     Running   4           20m
kube-flannel-ds-amd64-vqnsf         1/1     Running   3           28m
kube-proxy-46dvw                    1/1     Running   1           20m
kube-proxy-dcxfl                    1/1     Running   2           32m
kube-scheduler-k8s-master           1/1     Running   2           31m
```

图21.集群状态示意图

5.服务部署

5.1Deployment

一次deployment为一次部署，如图Deployment调用ReplicaSet创建多个Pod副本，而ReplicaSet不需要我们去管理，所以我们只需要创建一个deployment即可，我们编写一个nginx-deployment.yaml。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

Yaml配置文件中的相关参数含义如下所示。

apiVersion:组名/版本号

Kind:类型

Metadata:元数据，下面name表示当前deployment名称为nginx-deployment

Spec:规格

Replicas:3表示副本集有3个

Spec.template.metadata.labels自定义标签，一般配合selector

spec.template.containers:可以有多个容器

-name表示容器[1]的名字

Image:使用镜像nginx:1.7.9

containerPort:容器端口80

使用如下命令来通过yaml配置文件部署服务。

```
kubectl apply -f nginx-deployment.yaml
```

```
[root@k8s-master ~]# kubectl apply -f nginx-deployment.yaml
deployment.extensions/nginx-deployment created
```

图 22. 部署服务示意图

我们可以通过如下命令来查看各个pod的状态，当所有pod都准备完毕后，我们的服务也就部署完毕了。

```
kubectl apply -f nginx-deployment.yaml
```

```
[root@k8s-master ~]# kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-6c66549b67-22smv  1/1     Running   0           74s
nginx-deployment-6c66549b67-jz9qb  1/1     Running   0           74s
nginx-deployment-6c66549b67-nlgm8  1/1     Running   0           74s
```

图23.pod状态示意图

此时我们使用-owide查看ip，此时的应用只有集群内部能访问，外部暂时不能访问，利用curl访问属于k8s-master的pod，可以发现回显nginx的首页内容。

```
[root@k8s-master ~]# kubectl get pod -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP            NODE           NOMINATED NODE   READINESS GATES
nginx-deployment-6c66549b67-22smv  1/1     Running   0           74s   10.1.1.1      k8s-node1     <none>            <none>
nginx-deployment-6c66549b67-jz9qb  1/1     Running   0           74s   10.1.1.2      k8s-node2     <none>            <none>
nginx-deployment-6c66549b67-nlgm8  1/1     Running   0           74s   10.1.1.3      k8s-node3     <none>            <none>

[root@k8s-master ~]# curl 10.1.1.1
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
body {
background-color: #fff2cc;
margin: 0 auto;
text-align: center;
}
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
a href="http://nginx.org/">a href="http://nginx.org/">nginx.org</p>
```

图24.访问网站示意图

5.2Service

Service定义了一个服务的访问入口地址，前端的应用通过这个入口地址访问其背后的一组由Pod副本组成的集群实例，来自外部的访问请求被负载均衡到后端的各个容器应用上。Service与其后端Pod副本集群之间则是通过Selector标签来实现对接的。

创建如下yaml配置文件，作为nginx服务的配置(nginx-svc.yaml)。

```
apiVersion : v1
kind: Service
metadata:
name: nginx-service
labels:
app: nginx
spec:
type: NodePort
ports:
- port: 80
targetPort: 80 selector:
app: nginx
```

成功创建yaml配置文件后，运行如下命令来通过yaml配置文件启动服务。

```
kubectl apply -f nginx-svc.yaml
```

```
[root@k8s-master ~]# kubectl apply -f nginx-svc.yaml
service/nginx-service created
```

图25.服务启动示意图

成功启动服务之后，我们可以通过如下命令来查看目前已经部署服务的状态，以及端口映射情况。

```
kubectl get svc
```

```
[root@k8s-master ~]# kubectl get svc
NAME                TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
kubernetes          ClusterIP   10.96.0.1     <none>         443/TCP        85m
nginx-service       NodePort    10.111.156.128 <none>         80:31342/TCP   45s
```

图26.服务状态示意图

因为我们的type指定的是NodePort所以我们可以直接在浏览器使用任意节点ip+30565访问，并且我们指定的pod有3个，他每次会根据负载策略去访问三个pod中的任意一个，不过这里我们只有一个pod属于k8s-master，有两个pod属于k8s-node，因此k8s-node访问时会根据负载均衡来访问两个pod中的一个，而k8s-master实际上就只有一个pod，负载均衡不起作用。

如下图所示，我们已经可以通过结点的ip地址来访问网站服务而不是集群节点的ip。

```
[root@k8s-master ~]# curl 192.168.2.5:31342
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 350px;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

图27.网站服务访问示意图

6. 镜像变更

6.1 查看当前镜像版本

使用如下指令可以查看pod的相关信息，可以比较直观的看出pod的name，状态，ip地址等常用信息。

```
kubectl get pod -o wide
```

```
[root@k8s-master ~]# kubectl get pod -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE             NOMINATED NODE   READINESS GATES
nginx-deployment-6c66540b67-22mrv   1/1     Running   0           19m   172.17.0.3      k8s-node         <none>            <none>
nginx-deployment-6c66540b67-j19qb   1/1     Running   0           19m   172.17.0.4      k8s-master       <none>            <none>
nginx-deployment-6c66540b67-nlge8   1/1     Running   0           19m   172.17.0.2      k8s-node         <none>            <none>
```

图28.pod状态示意图

如果希望查看某个pod的具体信息，可以使用如下指令，除去上述name，z状态等信息之外还能看到pod所用镜像，端口映射等更为详细的信息。

```
Kubectl describe pod podName
```

```
[root@k8s-master ~]# kubectl describe pod nginx-deployment-6c66540b67-j19qb
Name:                               nginx-deployment-6c66540b67-j19qb
Namespace:                           default
Priority:                             0
Node:                                k8s-master/192.168.2.5
Start Time:                          Sat, 10 Jun 2023 17:00:17 +0800
Labels:                               app=nginx
                                      pod-template-hash=6c66540b67
                                      track-stable
Annotations:                          <none>
Status:                               Running
IP:                                  172.17.0.4
Controlled By:                        ReplicaSet/nginx-deployment-6c66540b67
Containers:
  nginx:
    Container ID:   docker://926eb700905069f85a858d4268d4696d10812fcd80ad0d3610f729e918495
    Image:          nginx:1.7.0
    Image ID:       docker-pullable://nginx@sha256:a3456c651a15249c3a4ff5fcd261248286abac6c0d794aff548e9714646c451
    Port:           80/TCP
    Host Port:      80/TCP
    State:          Running
```

图29.pod详细状态示意图

6.2 更新镜像

当我们希望能够较为方便的更新镜像，比如版本、镜像类型等，可以通过如下指令来进行

```
kubectl set image deploy/nginx-deployment nginx=nginx:1.10 --record
```

更为通用的形式为

```
kubectl set image deploy/部署的服务名 原镜像名=新镜像名:版本 --record
```

```
[root@k8s-master ~]# kubectl set image deploy/nginx-deployment nginx=nginx:1.10 --record
deployment.extensions/nginx-deployment image updated
```

图30.pod更新镜像示意图

Kubectl get pod 可以看到之前的pod并不会被马上终止，而是一步步新运行一个，终止一个，达到灰度发布，不重启服务器更新的目的。稍等片刻后我们就可以看到之前的3个pod全部被替换。

```
[root@k8s-master ~]# kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-deployment-5c9c6bc7d8-n7hzn	0/1	ContainerCreating	0	26s
nginx-deployment-5c9c6bc7d8-qblks	0/1	ContainerCreating	0	26s
nginx-deployment-6c66549b67-22smv	1/1	Running	0	21m
nginx-deployment-6c66549b67-jz9qb	1/1	Running	0	21m

图31.镜像更新时pod状态示意图

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
nginx-deployment-5c9c6bc7d8-692vs	1/1	Running	0	62s	172.17.0.3	k8s-node	<none>	<none>
nginx-deployment-5c9c6bc7d8-n7hzn	1/1	Running	0	2m19s	172.17.0.2	k8s-node	<none>	<none>
nginx-deployment-5c9c6bc7d8-qblke	1/1	Running	0	2m19s	172.17.0.5	k8s-master	<none>	<none>

图32.更新镜像后pod状态示意图

```
[root@k8s-master ~]# kubectl describe pod nginx-deployment-5c9c6bc7d8-qblke
```

Name: nginx-deployment-5c9c6bc7d8-qblke
Namespace: default
Priority: 0
Node: k8s-master/192.168.2.5
StartTime: Sat, 18 Jun 2025 17:10:14 -0800
Labels: app=nginx, pod-template-hash=5c9c6bc7d8, ttack=atable
Annotations: <none>
Status: Running
IP: 172.17.0.5
Controlled By: ReplicaSet/nginx-deployment-5c9c6bc7d8
Containers:
 nginx:
 Container ID: docker://a05ca7a1d259b26c0ba5e9cb6b60640d592510b7189580fo99d1ff006763143
 Image: nginx:1.18
 Image ID: docker-pullable://nginx0cho256:6202beb06e061f44179002ca0050a0e11b961d12640101fca2110f0fd145d7575
 Port: 80/TCP
 Host Port: 8/TCP
 State: Running
 Started: Sat, 18 Jun 2025 17:11:45 -0800

图33.pod更新镜像状态示意图

7.操作回滚

7.1 查看历史版本

我们对于所部署的服务的变更操作是会被记录下来，一是方便用户查看操作记录，二是方便在用户操作错误时能够回滚恢复原状，使用如下命令可以查看服务的相关操

作历史。

```
kubectl rollout history deploy/nginx-deployment
```

可以看到以上有1和2两个版本，1表示之前的，2表示当前的，如果有多次操作，可能有3,4.....那么最新的就表示当前版本。

```
[root@k8s-master ~]# kubectl rollout history deploy/nginx-deployment
deployment.extensions/nginx-deployment
REVISION  CHANGE-CAUSE
1          <none>
2          kubectl set image deploy/nginx-deployment nginx=nginx:1.10 --record=true
```

图34.服务操作历史示意图

7.2 回滚版本

版本回滚只要有两个操作，分别是回滚到上一个版本以及回滚到指定版本，其对应的操作指令分别如下。

```
Kubectl rollout undo deploy/nginx-deployment
```

```
Kubectl rollout undo deploy/nginx-deployment --to-revision=2 (回滚到版本 2)
```

我们回滚到上一个版本试试，可以看到我们镜像版本已经回退到之前的1.7.9了，回滚过程也是逐个替换，滚动更新。

```
[root@k8s-master ~]# kubectl rollout undo deploy/nginx-deployment
deployment.extensions/nginx-deployment rolled back
```

图35.回滚到上一版本示意图

```
[root@k8s-master ~]# kubectl rollout undo deploy/nginx-deployment
deployment.extensions/nginx-deployment rolled back
[root@k8s-master ~]# kubectl get pod -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE           NOMINATED NODE   READINESS GATES
nginx-deployment-6c60d49b67-148aw   1/1     Running   0           18d   172.17.0.5      k8s-node       <none>            <none>
nginx-deployment-6c66540b67-tmr3q   1/1     Running   0           11a   172.17.0.4      k8s-node       <none>            <none>
nginx-deployment-6c66549b67-v22sh   1/1     Running   0           11a   172.17.0.4      k8s-master     <none>            <none>
```

图36.更新镜像后pod状态示意图

```
[root@k8s-master ~]# kubectl describe pod nginx-deployment-6c66549b67-v22sh
Name:          nginx-deployment-6c66549b67-v22sh
Namespace:     default
Priority:       0
Node:          k8s-master/192.168.2.5
Start Time:    Sat, 10 Jun 2023 17:39:39 +0800
Labels:        app=nginx
               pod-template-hash=6c66549b67
               track=stable
Annotations:   <none>
Status:        Running
IP:            172.17.0.4
Controlled By: ReplicaSet/nginx-deployment-6c66549b67
Containers:
  nginx:
    Container ID:  docker://51d142c68237889f268784dc8b19e91948793ce45949b61ee40i3c75b85883dd
    Image:          nginx:1.7.9
```

图37.回滚后pod状态示意图

8.Dashborad

下载yaml文件。

```
wget
```

```
https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.3/aio/deploy/recommended.yaml
```

然后修改service部分如下所示

```
kind: Service
apiVersion: v1 metadata:
labels:
k8s-app: kubernetes-dashboard name: kubernetes-dashboard
namespace: kubernetes-dashboard spec:
type: NodePort
ports:
- port: 443
targetPort: 8443
nodePort: 30443 selector:
k8s-app: kubernetes-dashboard
```

利用如下命令更新配置

```
kubectl apply -f recommended.yaml
```

由于我们设置的nodePort是30443，因此通过如下url可以访问dashboard。

```
https://<any_node_ip>:30443
```

Dashboard支持Kubeconfig和Token两种认证方式，我们这里选择Token认证方式登录，因此创建如下用户配置文件。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
name: admin-user
roleRef:
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: cluster-admin
subjects:
- kind: ServiceAccount
name: admin-user
namespace: kubernetes-dashboard > EOF
```

通过如下命令来创建登录用户。

```
kubectl apply -f dashboard-adminuser.yaml
```

上面创建了一个叫admin-user的服务账号，并放在kubernetes-dashboard命名空间下，并将cluster-admin角色绑定到admin-user账户，这样admin-user账户就有了管理员的权限。默认情况下，kubeadm创建集群时已经创建了cluster-admin角色，我们直接绑定即可。

通过如下命令查看用户token

```
kubectl -n kubernetes-dashboard describe secret $(kubectl -n kubernetes-dashboard get secret | grep admin-user | awk '{print $1}')
```

把获取到的Token复制到登录界面的Token输入框中，登陆成功。