K8S 概览

1.1 K8S 是什么?

K8S官网文档：**https://kubernetes.io/zh/docs/home/**

K8S 是Kubernetes的全称，源于希腊语，意为“舵手”或“飞行员”，官方称其是：用于自动部署、扩展和管理“容器化（containerized）应用程序”的开源系统。翻译成大白话就是：“**K8S 是负责自动化运维管理多个跨机器 Docker 程序的集群**”。

1.2 K8S核心特性

- 服务发现与负载均衡：无需修改你的应用程序即可使用陌生的服务发现机制。
- 存储编排：自动挂载所选存储系统，包括本地存储。
- Secret和配置管理：部署更新Secrets和应用程序的配置时不必重新构建容器镜像，且不必将软件堆栈配置中的秘密信息暴露出来。
- 批量执行：除了服务之外，Kubernetes还可以管理你的批处理和CI工作负载，在期望时替换掉失效的容器。
- 水平扩缩：使用一个简单的命令、一个UI或基于CPU使用情况自动对应用程序进行扩缩。
- 自动化上线和回滚：Kubernetes会分步骤地将针对应用或其配置的更改上线，同时监视应用程序运行状况以确保你不会同时终止所有实例。
- 自动装箱：根据资源需求和其他约束自动放置容器，同时避免影响可用性。
- 自我修复：重新启动失败的容器，在节点死亡时替换并重新调度容器，杀死不响应用户定义的健康检查的容器。

1.3 K8S集群安装

搭建K8S集群，准备三台2核4G的虚拟机(内存至少2G以上)，操作系统选择用centos 7以上版本，先在三台机器上装好docker(安装参考docker课程，docker版本最好跟docker课上用的版本一致，防止与K8S的兼容性问题)：

在三台机器上都执行如下命令操作：

```
1、关闭防火墙
systemctl stop firewalld
systemctl disable firewalld

2、关闭 selinux
sed -i 's/enforcing/disabled/' /etc/selinux/config # 永久关闭
setenforce 0 # 临时关闭

3、关闭 swap
 swapoff -a # 临时关闭
 vim /etc/fstab # 永久关闭
 #注释掉swap这行
 # /dev/mapper/centos-swap swap swap defaults 0 0

 systemctl reboot #重启生效
 free -m #查看下swap交换区是否都为0，如果都为0则swap关闭成功

```

```
4、给三台机器分别设置主机名
hostnamectl set-hostname <hostname>
第一台：k8s-master
第二台：k8s-node1
第三台：k8s-node2

5、在 k8s-master机器添加hosts，执行如下命令，ip需要修改成你自己机器的ip
cat >> /etc/hosts << EOF
192.168.65.160 k8s-master
192.168.65.203 k8s-node1
192.168.65.210 k8s-node2
EOF

6、将桥接的IPv4流量传递到iptables
cat > /etc/sysctl.d/k8s.conf << EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF

sysctl --system # 生效

7、设置时间同步
yum install ntpdate -y
ntpdate time.windows.com

8、添加k8s yum源
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=0
repo_gpgcheck=0
gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg
https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg
EOF

9、如果之前安装过k8s，先卸载旧版本
yum remove -y kubelet kubeadm kubectl

10、查看可以安装的版本
yum list kubelet --showduplicates | sort -r

11、安装kubelet、kubeadm、kubectl 指定版本，我们使用kubeadm方式安装k8s集群
yum install -y kubelet-1.18.0 kubeadm-1.18.0 kubectl-1.18.0

12、开机启动kubelet
systemctl enable kubelet
systemctl start kubelet
```

在k8s-master机器上执行初始化操作(里面的第一个ip地址就是k8s-master机器的ip，改成你自己机器的，后面两个ip网段不用动)

```
kubeadm init --apiserver-advertise-address=192.168.65.160 --image-repository registry.aliyuncs.com/google_containers --kubernetes-version v1.18.0 --service-cidr=10.96.0.0/12 --pod-network-cidr=10.244.0.0/16
```

执行完后结果如下图：

```
[bootstrap-token] Using token: hbovty.6x82bkdlsk6dfy32
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to get nodes
[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get lo
[bootstrap-token] configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a No
[bootstrap-token] configured RBAC rules to allow certificate rotation for all node client certificates in the cl
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate an
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!   代表初始化成功

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

  mkdir -p $HOME/.kube          这段命令需要在master执行下
  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 as root:

kubeadm join 192.168.65.160:6443 --token hbovty.6x82bkdlsk6dfy32 \        这段命令需要在所有node节点上执行
    --discovery-token-ca-cert-hash sha256:659511b431f276b2a5f47397677b1dff74838ae5eb18e24135e6dae1b8c45840
```

在k8s-master机器上执行如下命令：

```
#配置使用 kubectl 命令工具(类似docker这个命令)，执行上图第二个红框里的命令
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kubectl是否能正常使用
kubectl get nodes

#安装 Pod 网络插件
 kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
 # 如果上面这个calico网络插件安装不成功可以试下下面这个
 # kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kubeflannel.yml
```

在所有k8s node机器执行上图第三个红框里的命令

```
# 将node节点加入进master节点的集群里，复制上图第三个红框里的命令执行
kubeadm join 192.168.65.160:6443 --token hbovty.6x82bkdlsk6dfy32 \
 --discovery-token-ca-cert-hash sha256:659511b431f276b2a5f47397677b1dff74838ae5eb18e24135e6dae1b8c4584
0
```

在k8s-master机器执行查看节点命令

```
kubectl get nodes
```

```
[root@k8s-master local]# kubectl get nodes
NAME         STATUS   ROLES    AGE   VERSION
k8s-master   Ready    master   90d   v1.18.0
k8s-node1    Ready    <none>   90d   v1.18.0
k8s-node2    Ready    <none>   90d   v1.18.0
```

刚刚安装的三个k8s节点都已经准备就绪，**大功告成**！

补充：如果node节点添加进集群失败，可以删除节点重新添加

要删除 k8s-node1 这个节点，首先在 master 节点上依次执行以下两个命令

```
kubectl drain k8s-node1 --delete-local-data --force --ignore-daemonsets
kubectl delete node k8s-node1
```

执行后通过 kubectl get node 命令可以看到 k8s-node1 已被成功删除

接着在 k8s-node1 这个 Node 节点上执行如下命令，这样该节点即完全从 k8s 集群中脱离开来，之后就可以重新执行命令添加到集群

```
kubeadm reset
```

用K8S部署Nginx

在k8s-master机器上执行

```
# 创建一次deployment部署
kubectl create deployment nginx --image=nginx
kubectl expose deployment nginx --port=80 --type=NodePort
# 查看Nginx的pod和service信息
kubectl get pod,svc -o wide
```

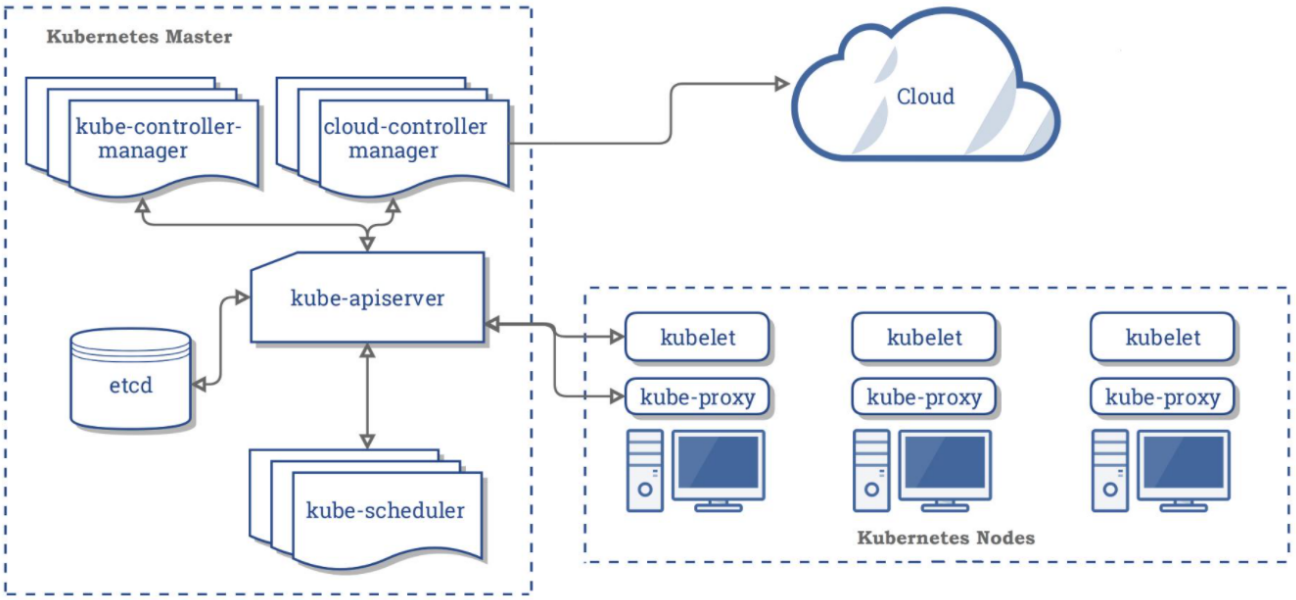
```
[root@k8s-master ~]# kubectl get pod,svc -o wide
NAME                         READY   STATUS    RESTARTS   AGE    IP               NODE        NOMINATED NODE   READINESS GATES
pod/nginx-f89759699-ngqjl    1/1     Running   0          106s   10.244.169.130   k8s-node2   <none>           <none>

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE   SELECTOR
service/kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        6d    <none>
service/nginx        NodePort    10.109.128.56   <none>        80:30433/TCP   22s   app=nginx
```

访问Nginx地址： http://任意节点的ip:图中Nginx的对外映射端口，http://192.168.65.203:30433

1.4 K8S 核心架构原理

我们已经知道了 K8S 的核心功能：自动化运维管理多个容器化程序。那么 K8S 怎么做到的呢？这里，我们从宏观架构上来学习 K8S 的设计思想。首先看下图：



K8S 是属于**主从设备模型（Master-Slave 架构）**，即有 Master 节点负责核心的调度、管理和运维，Slave 节点则执行用户的程序。但是在 K8S 中，主节点一般被称为**Master Node 或者 Head Node**，而从节点则被称为**Worker Node 或者 Node。**

注意：Master Node 和 Worker Node 是分别安装了 K8S 的 Master 和 Woker 组件的实体服务器，每个 Node 都对应了一台实体服务器（虽然 Master Node 可以和其中一个 Worker Node 安装在同一台服务器，但是建议 Master Node 单独部署），**所有 Master Node 和 Worker Node 组成了 K8S 集群**，同一个集群可能存在多个 Master Node 和 Worker Node。

首先来看**Master Node**都有哪些组件：

- **API Server。K8S 的请求入口服务。**API Server 负责接收 K8S 所有请求（来自 UI 界面或者 CLI 命令行工具），然后，API Server 根据用户的具体请求，去通知其他组件干活。
- **Scheduler。K8S 所有 Worker Node 的调度器。**当用户要部署服务时，Scheduler 会选择最合适的 Worker Node（服务器）来部署。
- **Controller Manager。K8S 所有 Worker Node 的监控器。**Controller Manager 有很多具体的 Controller， Node Controller、Service Controller、Volume Controller 等。Controller 负责监控和调整在 Worker Node 上部署的服务的状态，比如用户要求 A 服务部署 2 个副本，那么当其中一个服务挂了的时候，Controller 会马上调整，让 Scheduler 再选择一个 Worker Node 重新部署服务。
- **etcd。K8S 的存储服务。**etcd 存储了 K8S 的关键配置和用户配置，K8S 中仅 API Server 才具备读写权限，其他组件必须通过 API Server 的接口才能读写数据。

接着来看**Worker Node**的组件：

- **Kubelet。Worker Node 的监视器，以及与 Master Node 的通讯器。**Kubelet 是 Master Node 安插在 Worker Node 上的“眼线”，它会定期向 Master Node 汇报自己 Node 上运行的服务的状态，并接受来自 Master Node 的指示采取调整措施。负责控制所有容器的启动停止，保证节点工作正常。
- **Kube-Proxy。K8S 的网络代理。**Kube-Proxy 负责 Node 在 K8S 的网络通讯、以及对外部网络流量的负载均衡。
- **Container Runtime。Worker Node 的运行环境。**即安装了容器化所需的软件环境确保容器化程序能够跑起来，比如 Docker Engine运行环境。