搭建k8s集群预备步骤

0x00: 目标

基于Ubuntu 20.04系统使用kubeadm搭建一个`双节点`的k8s集群, 集群版本为`v1.20.4`, 并安装下列组件:

- CNI(flannel)
- metrics-server
- 集群控制面面板(kubernetes dashboard)
- 集群边缘网关(traefik)

0x01: 节点定义

在实验中会有以下节点定义:

- `宿主机`指的是运行multipass的机器
- `虚拟机`指的是`multipass`内运行的GuestOS
- `主节点`指的是运行集群控制面板的节点, 也是引导集群的节点
- `Worker节点`指的是加入到主节点的k8s节点

0x02: 创建虚拟机

由于MacOS和Windows都不是原生支持Docker的系统, 所以本实验统一使用multipass虚拟机进行, 按照官网的指引将multipass安装到宿主机上, 然后在宿主机上执行下列命令初始化主节点虚拟机

```
$ multipass launch -c 2 -m 2G -d 50G --name=k8s-master
```

使用下列命令初始化Worker节点虚拟机

```
$ multipass launch -c 2 -m 4G -d 50G --name=k8s-node-1
```

`-c`参数指定虚拟机允许使用的CPU核心数, `-m`参数指定虚拟机允许使用的运行内存, `-d`参数指定虚拟机允许使用的磁盘空间

> 虚拟机运行起来后并不会马上占用指定的资源, 这里的参数只是用作限制虚拟机使用资源的最高水位

等待虚拟机创建完毕后验证虚拟机状态

```
$ multipass ls
Name                    State             IPv4
Image
k8s-master              Running           192.168.64.8
Ubuntu 20.04 LTS
k8s-node-1              Running           192.168.64.12
Ubuntu 20.04 LTS
```

0x03: 初始化虚拟机

> 本步骤需要在主节点和Worker节点上都走一遍

使用`multipass shell $NAME`命令打开主节点的终端($NAME替换成主节点虚拟机名称和Worker节点虚拟机名称), 为了方便后续的实验, 为虚拟机设置一个root密码

```
$ sudo passwd
New password: 输入root密码
```

修改完root密码后使用`su`切换到root身份

```
$ su
Password: 输入root密码
root@k8s-master:/home/ubuntu#
```

接下来运行初始化节点脚本, 这个步骤主要是安装Docker和k8s的组件, 以及进行一些必要的系统参数调整

```
curl "https://bcy-next.oss-cn-hangzhou.aliyuncs.com/k8s-puzzle/00-init-node.sh" | sh -
```

脚本运行完毕后虚拟机会自动重启, 等待一会使用`multipass ls`命令确认虚拟机状态

0x04: 预拉取镜像

> 本步骤在主节点上进行, 如果想要加快Worker节点初始化过程也可以在Worker节点上运行, 拉取镜像会占用大量带宽, 不要在公司上班时间拉取

```
curl "https://bcy-next.oss-cn-hangzhou.aliyuncs.com/k8s-puzzle/01-pull-images.sh" | sh -
```

拉取的镜像包含:

- k8s组件镜像
- etcd镜像
- flannel
- traefik
- kubernetes-dashboard

0x05: 初始化集群

> 本步骤在主节点上进行

```
kubeadm init \
  --image-repository="registry.aliyuncs.com/google_containers" \
  --node-name="$(hostname -I | awk -F ' ' '{print $1}')" \
  --token-ttl=0 \
  --upload-certs \
  --kubernetes-version=v1.20.4 \
  --pod-network-cidr="10.244.0.0/16"
```

上述命令使用`kubeadm init`命令初始化一个集群, 此命令的可用参数列表可以参考官方文档, 下面解释一下用到的几个参数

- `--image-repository`指定初始化过程使用的镜像仓库, 由于众所周知的原因

- `--node-name`指定节点的名称, 默认是取`hostname`, 这里使用节点的IP作为节点名称
- `--pod-network-cidr`指定Pod的CIDR范围, CIDR与CNI强关联

初始化完毕后会输出以下内容

```
Your Kubernetes control-plane has initialized successfully!

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

// 执行此处的三条命令
  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
//
Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

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:

// 保存以下内容
kubeadm join 192.168.64.30:6443 --token 53hny9.t50ty38a9nx4uvrn \
    --discovery-token-ca-cert-hash sha256:b1ab7e7e93f927da681afb1808a3636f86a96d05b455192f4825df2d8b92e56c
```

按照上面的标注执行三条命令将kubeconfig移动到kubectl的默认路径, 然后将加入节点的命令妥善保存下面步骤需要用到

此时可以运行kubectl判断集群状态

```
kubectl get nodes
NAME            STATUS     ROLES                  AGE
VERSION
192.168.64.30   NotReady   control-plane,master   4m17s
v1.20.4
```

注意, kubeadm默认会给主节点打上一个NoSchedule的污点, 这会导致业务的Pod无法调度到主节点, 如需去掉这个污点自行参考官方文档

0x06: 初始化Worker节点

> 本步骤在Worker节点上进行

```
kubeadm join 192.168.64.30:6443 \
  --token 53hny9.t50ty38a9nx4uvrn \
  --discovery-token-ca-cert-hash
sha256:b1ab7e7e93f927da681afb1808a3636f86a96d05b455192f4825df
2d8b92e56c \
  --node-name="$(hostname -I | awk -F ' ' '{print $1}')"
```

注意, 需要执行的命令是上一步骤保存的命令, 另外需要额外加上`--node-name`参数

预期输出内容如下

```
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with
'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file
"/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags
to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS
Bootstrap...
```

```
8
9  This node has joined the cluster:
10 * Certificate signing request was sent to apiserver and a
   response was received.
11 * The Kubelet was informed of the new secure connection
   details.
12
13 Run 'kubectl get nodes' on the control-plane to see this
   node join the cluster.
14
```

> 如需要添加多个节点, 重复上面步骤即可

回到主节点查看Worker节点的状态

```
1  kubectl get nodes
2  NAME            STATUS     ROLES                  AGE
   VERSION
3  192.168.64.30   NotReady   control-plane,master   19m
   v1.20.4
4  192.168.64.31   NotReady   <none>                 38s
   v1.20.4
```

0x07: 安装flannel

> 本步骤在主节点上进行

由于集群缺少CNI组件, 所有节点都处于NotReady状态, 这里选用flannel作为集群CNI组件

```
1  kubectl apply -f https://bcy-next.oss-cn-
   hangzhou.aliyuncs.com/k8s-puzzle/02-kube-flannel.yml
```

稍等片刻使用下面命令判断flannel部署状态

```
kubectl get pod -n kube-system | grep flannel
kube-flannel-ds-h4crq                    1/1     Running   0
       78s
kube-flannel-ds-hj8h9                    1/1     Running   0
       78s
```

等到两个flannel的Pod都已经`Running`状态, 再次查看节点状态

```
kubectl get nodes
NAME            STATUS   ROLES                  AGE
VERSION
192.168.64.30   Ready    control-plane,master   29m
v1.20.4
192.168.64.31   Ready    <none>                 6m28s
v1.20.4
```

此时节点已经Ready!

0x08: 部署kubernetes-dashboard

> 本步骤在主节点上进行

kubernetes-dashboard是通用k8s集群控制面板, 方便以可视化方式操作集群和获取集群状态

```
kubectl apply -f https://bcy-next.oss-cn-
hangzhou.aliyuncs.com/k8s-puzzle/03-kubernetes-dashboard.yaml
```

稍等片刻使用下面命令判断kubernetes-dashboard部署状态

```
kubectl get pod -n kubernetes-dashboard
NAME                                         READY   STATUS
RESTARTS   AGE
dashboard-metrics-scraper-79c5968bdc-kjwvz   1/1     Running
  0          3m59s
kubernetes-dashboard-86bc9d74f9-h4mfz        1/1     Running
  0          3m59s
```

0x09: 访问kubernetes-dashboard

> 本步骤在主节点上进行

上面的`03-kubernetes-dashboard.yaml`配置文件使用的是ClusterIP类型的Service, 如需访问这个Service需要使用`port-forward`命令做端口转发, 然后就可以使用宿主机访问主节点虚拟机的IP访问

```
kubectl port-forward svc/kubernetes-dashboard 9090:9090 -n
kubernetes-dashboard --address 0.0.0.0
```

> kubectl port-forward运行过程中不能关闭shell窗口

切换到宿主机获取主节点的IP地址

```
multipass ls | grep k8s-master | awk -F ' ' '{print $3}'
192.168.64.30
```

打开宿主机的浏览器, 访问`http://192.168.64.30:9090`即可访问kubernetes-dashboard