原

sudo vim /etc/docker/daemon.json

{

"registry-mirrors": ["https://gt6j98xi.mirror.aliyuncs.com"],

"data-root": "/data/docker",

"exec-opts": ["native.cgroupdriver=systemd"],

"iptables": false,

"default-runtime": "nvidia",

"runtimes": {

"nvidia": {

"path": "nvidia-container-runtime",

"runtimeArgs": []

}

}

}

改

{

"registry-mirrors": ["https://registry.aliyuncs.com", "https://hub-mirror.c.163.com", "https://mirror.baidubce.com", "https://docker.mirrors.ustc.edu.cn/"],

"runtimes": {

"kata-fc": {

"path": "/opt/kata/bin/kata-fc"

},

"kata-qemu": {

"path": "/usr/bin/kata-runtime"

},

"nvidia": {

"path": "/usr/bin/nvidia-container-runtime",

"runtimeArgs": []

}

},

"default-runtime": "nvidia",

"data-root": "/data/docker",

"exec-opts": ["native.cgroupdriver=systemd"],

"iptables": false,

"insecure-registries": ["172.16.20.190:5000"]

}

1. docker

（1）docker安装

1）版本：18.09

2）在安装docker之前，使用以下命令安装一些依赖包:

sudo apt-get install apt-transport-https

sudo apt-get install ca-certificates curl software-properties-common

3）安装docker包：

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo apt-key fingerprint 0EBFCD88

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"

sudo apt-get update

sudo apt-cache madison docker-ce

sudo apt-get install -y docker-ce=5:18.09.9~3-0~ubuntu-bionic

sudo apt-get install docker-ce-cli=5:18.09.9~3-0~ubuntu-bionic

（2）相关配置

1）创建守护daemon.json文件

sudo touch /etc/docker/daemon.json

2）进行环境配置：

{

"registry-mirrors": ["https://hub-mirror.c.163.com", "https://mirror.baidubce.com", "https://docker.mirrors.ustc.edu.cn/"], # 配置镜像源

"exec-opts": ["native.cgroupdriver=systemd"], #配置cgroup driver为systemd.

"iptables": false, # 避免docker更改iptables，而由Kubernetes修改

"data-root": "/var/lib/docker", # docker容器的默认存储路径，可按照要求修改

"insecure-registries": ["172.16.20.190:5000"], # 配置私有镜像仓库存储本智算平台所使用的基础镜像，后文给出配置方法.

"default-runtime": "nvidia", # 设置docker的runtime来支持GPU环境

"runtimes": {

"nvidia": { # the detail configuration of the nvidia runtime

"path": "nvidia-container-runtime",

"runtimeArgs": []

}

}

}

3）更新并启用环境配置：

sudo systemctl daemon-reload

4）配置私有镜像仓库：

目前该平台私有镜像仓库配置于主机：102.168.1.152

命令如下：

sudo docker run -itd -v <私有镜像存储路径，可修改>:/var/lib/registry -p 5000:5000 --restart=always --name registry registry:latest

更新配置：

sudo systemctl daemon-reload

5) 重启docker完成配置：

sudo systemctl restart docker

sudo systemctl enable docker

2. nvidia-docker2

安装nvidia-docker2来支持容器使用GPU资源

（1）nvidia-docker2 安装

1）添加key：

curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | \

sudo apt-key add -

2）添加包的依赖源：

curl -s -L https://nvidia.github.io/nvidia-docker/ubuntu18.04/nvidia-docker.list | \

sudo tee /etc/apt/sources.list.d/nvidia-docker.list

3）进行依赖包安装：

sudo docker run --runtime=nvidia --rm nvidia/cuda:10.0-base nvidia-smi

3.containerd

1）设置 repository 和 GPG key并安装 nvidia-container-runtime

curl -s -L https://nvidia.github.io/nvidia-container-runtime/gpgkey | sudo apt-key add -

curl -s -L https://nvidia.github.io/nvidia-container-runtime/$(. /etc/os-release;echo $ID$VERSION\_ID)/nvidia-container-runtime.list | sudo tee /etc/apt/sources.list.d/nvidia-container-runtime.list

sudo apt install nvidia-container-runtime -y

2）配置 Containerd 使用 Nvidia container runtime

mkdir /etc/containerd

sudo containerd config default | sudo tee /etc/containerd/config.toml > /dev/null

### Kubernetes 使用设备插件（Device Plugins） 来允许 Pod 访问类似 GPU 这类特殊的硬件功能特性，但前提是默认的 OCI runtime 必须改成 nvidia-container-runtime

修改containerd.toml文件

cd /etc/containerd/

vim config.toml

以下是修改和配置好nvidia runtime，sandbox和plugin镜像源，数据挂载路径的文件内容

disabled\_plugins = []

imports = []

oom\_score = 0

plugin\_dir = ""

required\_plugins = []

root = "/data/containerd" #修改存储根目录

state = "/run/containerd"

temp = ""

version = 2

[cgroup]

path = ""

[debug]

address = ""

format = ""

gid = 0

level = ""

uid = 0

[grpc]

address = "/run/containerd/containerd.sock"

gid = 0

max\_recv\_message\_size = 16777216

max\_send\_message\_size = 16777216

tcp\_address = ""

tcp\_tls\_ca = ""

tcp\_tls\_cert = ""

tcp\_tls\_key = ""

uid = 0

[metrics]

address = ""

grpc\_histogram = false

[plugins]

[plugins."io.containerd.gc.v1.scheduler"]

deletion\_threshold = 0

mutation\_threshold = 100

pause\_threshold = 0.02

schedule\_delay = "0s"

startup\_delay = "100ms"

[plugins."io.containerd.grpc.v1.cri"]

cdi\_spec\_dirs = ["/etc/cdi", "/var/run/cdi"]

device\_ownership\_from\_security\_context = false

disable\_apparmor = false

disable\_cgroup = false

disable\_hugetlb\_controller = true

disable\_proc\_mount = false

disable\_tcp\_service = true

drain\_exec\_sync\_io\_timeout = "0s"

enable\_cdi = false

enable\_selinux = false

enable\_tls\_streaming = false

enable\_unprivileged\_icmp = false

enable\_unprivileged\_ports = false

ignore\_deprecation\_warnings = []

ignore\_image\_defined\_volumes = false

image\_pull\_progress\_timeout = "5m0s"

image\_pull\_with\_sync\_fs = false

max\_concurrent\_downloads = 3

max\_container\_log\_line\_size = 16384

netns\_mounts\_under\_state\_dir = false

restrict\_oom\_score\_adj = false

sandbox\_image = "registry.cn-hangzhou.aliyuncs.com/google\_containers/pause:3.6" #设置镜像加速

selinux\_category\_range = 1024

stats\_collect\_period = 10

stream\_idle\_timeout = "4h0m0s"

stream\_server\_address = "127.0.0.1"

stream\_server\_port = "0"

systemd\_cgroup = false #注意这里是false不能动

tolerate\_missing\_hugetlb\_controller = true

unset\_seccomp\_profile = ""

[plugins."io.containerd.grpc.v1.cri".cni]

bin\_dir = "/opt/cni/bin"

conf\_dir = "/etc/cni/net.d"

conf\_template = ""

ip\_pref = ""

max\_conf\_num = 1

setup\_serially = false

[plugins."io.containerd.grpc.v1.cri".containerd]

default\_runtime\_name = "nvidia" #指定nvidia runtime

disable\_snapshot\_annotations = true

discard\_unpacked\_layers = false

ignore\_blockio\_not\_enabled\_errors = false

ignore\_rdt\_not\_enabled\_errors = false

no\_pivot = false

snapshotter = "overlayfs"

[plugins."io.containerd.grpc.v1.cri".containerd.default\_runtime]

base\_runtime\_spec = ""

cni\_conf\_dir = ""

cni\_max\_conf\_num = 0

container\_annotations = []

pod\_annotations = []

privileged\_without\_host\_devices = false

privileged\_without\_host\_devices\_all\_devices\_allowed = false

runtime\_engine = ""

runtime\_path = ""

runtime\_root = ""

runtime\_type = ""

sandbox\_mode = ""

snapshotter = ""

[plugins."io.containerd.grpc.v1.cri".containerd.default\_runtime.options]

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes]

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]

base\_runtime\_spec = ""

cni\_conf\_dir = ""

cni\_max\_conf\_num = 0

container\_annotations = []

pod\_annotations = []

privileged\_without\_host\_devices = false

privileged\_without\_host\_devices\_all\_devices\_allowed = false

runtime\_engine = ""

runtime\_path = ""

runtime\_root = ""

runtime\_type = "io.containerd.runc.v2"

sandbox\_mode = "podsandbox"

snapshotter = ""

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]

BinaryName = ""

CriuImagePath = ""

CriuPath = ""

CriuWorkPath = ""

IoGid = 0

IoUid = 0

NoNewKeyring = false

NoPivotRoot = false

Root = ""

ShimCgroup = ""

SystemdCgroup = true #注意这里启用为true

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.nvidia]

privileged\_without\_host\_devices = false

runtime\_engine = ""

runtime\_root = ""

runtime\_type = "io.containerd.runc.v2"

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.nvidia.options]

BinaryName = "/usr/bin/nvidia-container-runtime"

SystemdCgroup = true # 添加这一行

[plugins."io.containerd.grpc.v1.cri".containerd.untrusted\_workload\_runtime]

base\_runtime\_spec = ""

cni\_conf\_dir = ""

cni\_max\_conf\_num = 0

container\_annotations = []

pod\_annotations = []

privileged\_without\_host\_devices = false

privileged\_without\_host\_devices\_all\_devices\_allowed = false

runtime\_engine = ""

runtime\_path = ""

runtime\_root = ""

runtime\_type = ""

sandbox\_mode = ""

snapshotter = ""

[plugins."io.containerd.grpc.v1.cri".containerd.untrusted\_workload\_runtime.options]

[plugins."io.containerd.grpc.v1.cri".image\_decryption]

key\_model = "node"

[plugins."io.containerd.grpc.v1.cri".registry]

config\_path = ""

[plugins."io.containerd.grpc.v1.cri".registry.auths]

[plugins."io.containerd.grpc.v1.cri".registry.configs]

[plugins."io.containerd.grpc.v1.cri".registry.headers]

[plugins."io.containerd.grpc.v1.cri".registry.mirrors]

[plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]

endpoint = ["https://kllbcvjd.mirror.aliyuncs.com"]

[plugins."io.containerd.grpc.v1.cri".registry.mirrors."gcr.io"]

endpoint = ["https://gcr.mirrors.ustc.edu.cn"]

[plugins."io.containerd.grpc.v1.cri".registry.mirrors."k8s.gcr.io"]

endpoint = ["https://gcr.mirrors.ustc.edu.cn/google-containers/"]

[plugins."io.containerd.grpc.v1.cri".registry.mirrors."quay.io"]

endpoint = ["https://quay.mirrors.ustc.edu.cn"]

[plugins."io.containerd.grpc.v1.cri".x509\_key\_pair\_streaming]

tls\_cert\_file = ""

tls\_key\_file = ""

[plugins."io.containerd.internal.v1.opt"]

path = "/opt/containerd"

[plugins."io.containerd.internal.v1.restart"]

interval = "10s"

[plugins."io.containerd.internal.v1.tracing"]

sampling\_ratio = 1.0

service\_name = "containerd"

[plugins."io.containerd.metadata.v1.bolt"]

content\_sharing\_policy = "shared"

[plugins."io.containerd.monitor.v1.cgroups"]

no\_prometheus = false

[plugins."io.containerd.nri.v1.nri"]

disable = true

disable\_connections = false

plugin\_config\_path = "/etc/nri/conf.d"

plugin\_path = "/opt/nri/plugins"

plugin\_registration\_timeout = "5s"

plugin\_request\_timeout = "2s"

socket\_path = "/var/run/nri/nri.sock"

[plugins."io.containerd.runtime.v1.linux"]

no\_shim = false

runtime = "nvidia"

runtime\_root = ""

shim = "containerd-shim"

shim\_debug = false

[plugins."io.containerd.runtime.v2.task"]

platforms = ["linux/amd64"]

sched\_core = false

[plugins."io.containerd.service.v1.diff-service"]

default = ["walking"]

[plugins."io.containerd.service.v1.tasks-service"]

blockio\_config\_file = ""

rdt\_config\_file = ""

[plugins."io.containerd.snapshotter.v1.aufs"]

root\_path = ""

[plugins."io.containerd.snapshotter.v1.blockfile"]

fs\_type = ""

mount\_options = []

root\_path = ""

scratch\_file = ""

[plugins."io.containerd.snapshotter.v1.btrfs"]

root\_path = ""

[plugins."io.containerd.snapshotter.v1.devmapper"]

async\_remove = false

base\_image\_size = ""

discard\_blocks = false

fs\_options = ""

fs\_type = ""

pool\_name = ""

root\_path = ""

[plugins."io.containerd.snapshotter.v1.native"]

root\_path = ""

[plugins."io.containerd.snapshotter.v1.overlayfs"]

mount\_options = []

root\_path = ""

sync\_remove = false

upperdir\_label = false

[plugins."io.containerd.snapshotter.v1.zfs"]

root\_path = ""

[plugins."io.containerd.tracing.processor.v1.otlp"]

endpoint = ""

insecure = false

protocol = ""

[plugins."io.containerd.transfer.v1.local"]

config\_path = ""

max\_concurrent\_downloads = 3

max\_concurrent\_uploaded\_layers = 3

[[plugins."io.containerd.transfer.v1.local".unpack\_config]]

differ = ""

platform = "linux/amd64"

snapshotter = "overlayfs"

[proxy\_plugins]

[stream\_processors]

[stream\_processors."io.containerd.ocicrypt.decoder.v1.tar"]

accepts = ["application/vnd.oci.image.layer.v1.tar+encrypted"]

args = ["--decryption-keys-path", "/etc/containerd/ocicrypt/keys"]

env = ["OCICRYPT\_KEYPROVIDER\_CONFIG=/etc/containerd/ocicrypt/ocicrypt\_keyprovider.conf"]

path = "ctd-decoder"

returns = "application/vnd.oci.image.layer.v1.tar"

[stream\_processors."io.containerd.ocicrypt.decoder.v1.tar.gzip"]

accepts = ["application/vnd.oci.image.layer.v1.tar+gzip+encrypted"]

args = ["--decryption-keys-path", "/etc/containerd/ocicrypt/keys"]

env = ["OCICRYPT\_KEYPROVIDER\_CONFIG=/etc/containerd/ocicrypt/ocicrypt\_keyprovider.conf"]

path = "ctd-decoder"

returns = "application/vnd.oci.image.layer.v1.tar+gzip"

[timeouts]

"io.containerd.timeout.bolt.open" = "0s"

"io.containerd.timeout.metrics.shimstats" = "2s"

"io.containerd.timeout.shim.cleanup" = "5s"

"io.containerd.timeout.shim.load" = "5s"

"io.containerd.timeout.shim.shutdown" = "3s"

"io.containerd.timeout.task.state" = "2s"

[ttrpc]

address = ""

gid = 0

uid = 0

4.Kubernetes 1.28.14部署注意事项

需要为每台主机运行步骤1)~5)来构建Kubernetes集群，然后需要在Kubernetes集群的master主机上运行步骤6)~8)，并为每台其他主机运行步骤9)来加入Kubernetes集群。最后，请按照步骤10)为构建集群中的每个主机配置kubelet。

1）关闭防火墙

sudo systemctl stop ufw

sudo systemctl disable ufw

2）禁用selinux：

sudo vim /etc/selinux/config

输入：SELINUX=permissive

3）#关闭swap

swapoff -a

#永久关闭swap

vim /etc/fstab

#/swapfile none swap sw 0 0

注释掉swap memory uuid对应的item，例如：

# UUID=1d3c29bb-d730-4ad0-a659-45b25f60c37d none swap sw 0 0

sudo nano /etc/initramfs-tools/conf.d/resume

注释掉swap memory uuid对应的item，例如：

# RESUME=UUID=1d3c29bb-d730-4ad0-a659-45b25f60c37d

4）系统网络环境配置：

a) 进入/etc/sysctl.conf

sudo nano /etc/sysctl.conf

输入:

net.ipv4.ip\_forward = 1

b) 启用配置：

sudo sysctl -p

c) 配置网桥：

sudo tee /etc/sysctl.d/k8s.conf <<-'EOF'

net.ipv4.ip\_forward = 1

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

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

EOF

d) 安装net.bridge包：

sudo apt-get install -y bridge-utils

sudo modprobe bridge

sudo modprobe br\_netfilter

在启动时默认启用：

sudo nano /etc/rc.local

输入:

modprobe bridge

modprobe br\_netfilter

e) 启用配置:

sudo sysctl --system

f) 将需要加入集群的主机IP信息写入/etc/hosts，例如：

192.168.1.151 node01

192.168.1.152 node02

192.168.1.153 node03

5）安装k8s1.28.14   
a）#更新包列表和安装apt-transport-https

sudo apt-get update

sudo apt-get install -y apt-transport-https

b）# 添加 Kubernetes 存储库密钥

curl -fsSL https://mirrors.aliyun.com/kubernetes-new/core/stable/v1.28/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

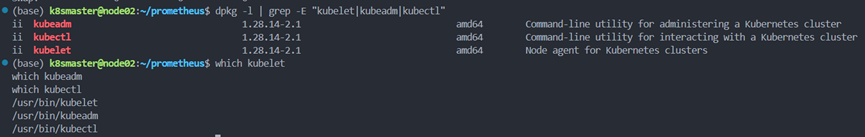
c）# 添加 Kubernetes 存储库

echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://mirrors.aliyun.com/kubernetes-new/core/stable/v1.28/deb/ /" | sudo tee /etc/apt/sources.list.d/kubernetes.list

d）#更新包列表并安装 Kubernetes 组件

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl



e) 固定版本：

sudo apt-mark hold kubelet kubeadm kubectl docker-ce docker-ce-cli

6）配置主节点并初始化

选择集群中的某台主机为平台Kubernetes集群的主节点，当前选择192.168.1.152

a) 生成初始化集群配置：

sudo kubeadm config print init-defaults > init-config.yaml

b) 在init-config.yaml文件中进行初始化配置，需要进行以下修改：

i) 设置 localAPIEndpoint.advertiseAddress: 0.0.0.0

ii) 设置 etcd.local.dataDir: <选定的etcd存储位置，默认为/var/lib/etcd>

iii) 设置集群组件默认开源镜像库为阿里源：

imageRepository: registry.aliyuncs.com/google\_containers

iv) 设置 kubernetesVersion: v1.17.4

v) 设置Pod IP范围：networking.podSubnet: 10.122.0.0/16

vi) 设置serviceIP范围 networking.serviceSubnet: 10.10.0.0/16

################初始化

# 创建目录

sudo mkdir -p /etc/kubernetes

vi /etc/Kubernetes/kubeadm-config.yaml

写入

apiVersion: kubeadm.k8s.io/v1beta3

bootstrapTokens:

- groups:

  - system:bootstrappers:kubeadm:default-node-token

  token: abcdef.0123456789abcdef # generate by the kubeadm

  ttl: 24h0m0s

  usages:

  - signing

  - authentication

kind: InitConfiguration

localAPIEndpoint:

  advertiseAddress: 0.0.0.0 # set it as 0.0.0.0 or the INTERNAL ip of the master host

  bindPort: 6443

nodeRegistration:

  criSocket: unix:///var/run/containerd/containerd.sock

  name: node # master node name

  taints:

  - effect: NoSchedule

    key: node-role.kubernetes.io/master

---

apiServer:

  timeoutForControlPlane: 4m0s

apiVersion: kubeadm.k8s.io/v1beta3

certificatesDir: /etc/kubernetes/pki

clusterName: kubernetes

controllerManager: {}

dns:

  type: CoreDNS

etcd:

  local:

    dataDir: /data/etcd # default: /var/lib/etcd, you can change the path to store the etcd data

imageRepository: registry.aliyuncs.com/google\_containers # default: gcr.k8s.io, if you can not connect to the aliyun, you can use the defult imageRepository.

kind: ClusterConfiguration

kubernetesVersion: v1.28.14 # version of the kubernetes

networking:

  dnsDomain: cluster.local

  podSubnet: 10.122.0.0/16 # set the ip address of the pod subnet of the Kubernetes cluster

  serviceSubnet: 10.10.0.0/16 # set the ip address of the service subenet of the Kubernetes cluster

scheduler: {}

c）为主节点初始化

sudo kubeadm config images pull --image-repository=registry.aliyuncs.com/google\_containers

拉取成功则显示类似以下的内容  
I1024 00:20:08.675118 3897722 version.go:256] remote version is much newer: v1.31.2; falling back to: stable-1.28

[config/images] Pulled registry.aliyuncs.com/google\_containers/kube-apiserver:v1.28.15

[config/images]Pulled registry.aliyuncs.com/google\_containers/kube-controller-manager:v1.28.15

[config/images] Pulled registry.aliyuncs.com/google\_containers/kube-scheduler:v1.28.15

[config/images] Pulled registry.aliyuncs.com/google\_containers/kube-proxy:v1.28.15

[config/images] Pulled registry.aliyuncs.com/google\_containers/pause:3.9

[config/images] Pulled registry.aliyuncs.com/google\_containers/etcd:3.5.15-0

[config/images] Pulled registry.aliyuncs.com/google\_containers/coredns:v1.10.1

然后执行init操作  
sudo kubeadm init --config=init-config.yaml

d) 为当前用户赋予控制集群权限：

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

e）shell 自动补全(Bash)

<https://kubernetes.io/docs/reference/kubectl/cheatsheet/#kubectl-autocomplete>

source <**(**kubectl completion bash**)**

echo "source <(kubectl completion bash)" >> ~/.bashrc

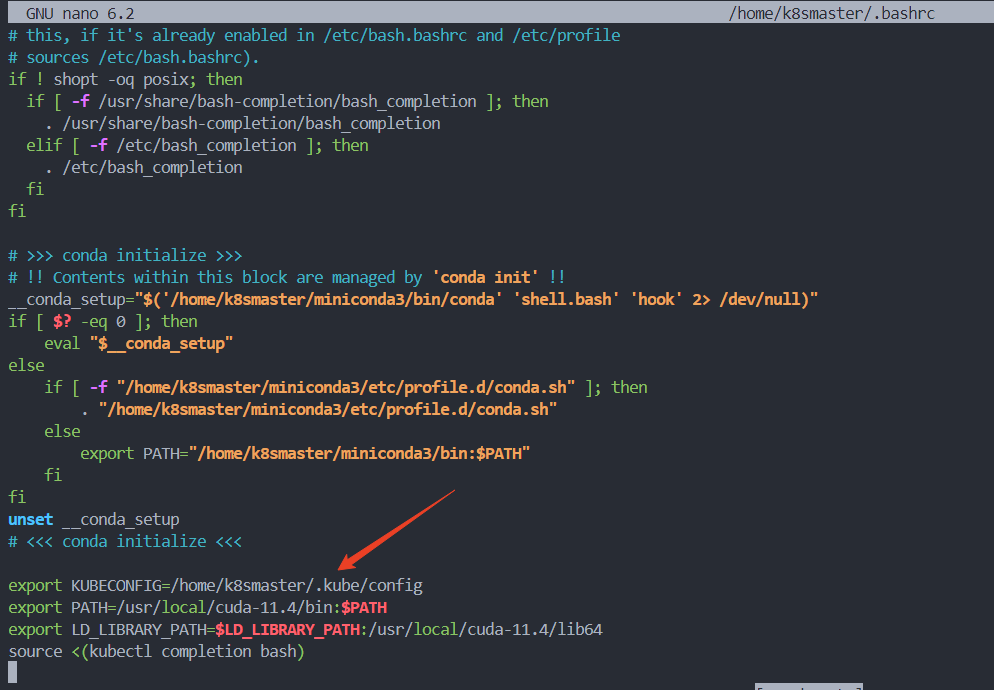
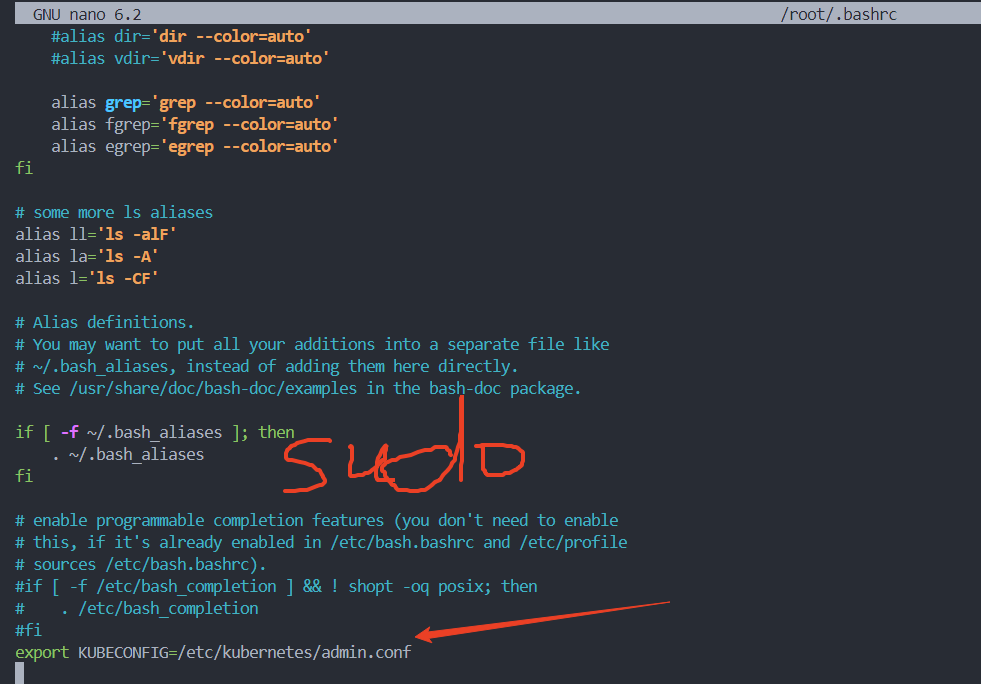
f）确保 kubectl 使用的是 ~/.kube/config 文件：

显式设置$KUBECONFIG：

export KUBECONFIG=/home/k8smaster/.kube/config

对于每个用户把上面的指令在.bashrc中修改后，source .bashrc

此外，要在root或者sudo中使用kubectl也必须在/root/.bashrc中，添加：export KUBECONFIG=/etc/kubernetes/admin.conf，然后source /root/.bashrc

7）开启主节点可调度容器：

kubectl taint node node01 node-role.kubernetes.io/master-

kubectl taint node node02 node-role.kubernetes.io/control-plane-

8）为主节点设置标识标签：

kubectl label node <主节点主机名> dash=true

例如kubectl label node voicec105 dash=true

9）配置kubelet

sudo nano /etc/systemd/system/kubelet.service.d/10-kubeadm.conf

# Note: This dropin only works with kubeadm and kubelet v1.11+

[Service]

Environment="KUBELET\_KUBECONFIG\_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf"

Environment="KUBELET\_CONFIG\_ARGS=--config=/var/lib/kubelet/config.yaml"

# This is a file that "kubeadm init" and "kubeadm join" generates at runtime, populating the KUBELET\_KUBEADM\_ARGS variable dynamically

EnvironmentFile=-/var/lib/kubelet/kubeadm-flags.env

# This is a file that the user can use for overrides of the kubelet args as a last resort. Preferably, the user should use

# the .NodeRegistration.KubeletExtraArgs object in the configuration files instead. KUBELET\_EXTRA\_ARGS should be sourced from this file.

EnvironmentFile=-/etc/default/kubelet

Environment="KUBELET\_LET\_LOAD\_ARGS=--enable\_load\_reader=true"

Environment="CADVISOR\_INTERVAL\_ARGS=--housekeeping\_interval=3s"

ExecStart=

ExecStart=/usr/bin/kubelet $KUBELET\_KUBECONFIG\_ARGS $KUBELET\_CONFIG\_ARGS $KUBELET\_KUBEADM\_ARGS $KUBELET\_EXTRA\_ARGS # $KUBELET\_LET\_LOAD\_ARGS $CADVISOR\_INTERVAL\_ARGS

sudo vi /usr/lib/systemd/system/kubelet.service  
原kubelet.service内容  
[Unit]

Description=kubelet: The Kubernetes Node Agent

Documentation=https://kubernetes.io/docs/

Wants=network-online.target

After=network-online.target

[Service]

ExecStart=/usr/bin/kubelet

Restart=always

StartLimitInterval=0

RestartSec=10

[Install]

WantedBy=multi-user.target

#修改/usr/lib/systemd/system/kubelet.service

[Unit]

Description=kubelet: The Kubernetes Node Agent

Documentation=https://kubernetes.io/docs/

After=containerd.service

Requires=containerd.service

[Service]

ExecStart=/usr/local/bin/kubelet \

--container-runtime=remote --runtime-request-timeout=15m --container-runtime-endpoint=unix:///run/containerd/containerd.sock \

--bootstrap-kubeconfig=/etc/kubernetes/bootstrap.kubeconfig \

--kubeconfig=/etc/kubernetes/kubelet.kubeconfig \

--config=/etc/kubernetes/kubelet.conf \

Restart=always

RestartSec=10s

[Install]

WantedBy=multi-user.target

启用配置：

sudo systemctl daemon-reload

sudo systemctl restart kubelet

sudo systemctl enable kubelet

10) 将其余主机添加至集群：

a) 在主节点上获取添加时需要的token：

sudo kubeadm token list

若不存在则生成：

sudo kubeadm token create

b) 获取CA certificate格式为SHA256编码的HASH值:

sudo openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | openssl dgst -sha256 -hex | sed 's/^.\* //'

c) 在其余主机中使用如下命令加入集群：

sudo kubeadm join <主节点IP>:6443 --token <token name> --discovery-token-ca-cert-hash sha256:<hash value of ca>

例如：

sudo kubeadm join 192.168.1.152:6443 --token m3vr25.g24757vwacsqf1uk --discovery-token-ca-cert-hash sha256:bd4f1d5c16a99da7be0b35438862b7d072b2dd79972d05ce13c925616e99e410

常见问题注意：

1. 在从节点上下载的kubernetes版本需要与主节点一致，若从节点有残留的不一致的k8s版本，需先卸载干净。
2. 从节点上kubelet（第9节中内容）与主节点保持一致，采用containerd运行时
3. 从节点join失败问题  
   ##在从节点中存在的join失败问题举例

sudo kubeadm join 192.168.1.152:6443 --token dji6cr.bhogea7liztsdn6h --discovery-token-ca-cert-hash sha256:e10410117f2f046f512d4b379cfc559363edc42213e15082d3d3e2f40d34af8a --ignore-preflight-errors=FileAvailable--etc-kubernetes-kubelet.conf,Port-10250,FileAvailable--etc-kubernetes-pki-ca.crt

[preflight] Running pre-flight checks

[WARNING FileAvailable--etc-kubernetes-kubelet.conf]: /etc/kubernetes/kubelet.conf already exists

[WARNING Port-10250]: Port 10250 is in use

[WARNING FileAvailable--etc-kubernetes-pki-ca.crt]: /etc/kubernetes/pki/ca.crt already exists

[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'

error execution phase kubelet-start: a Node with name "node01" and status "Ready" already exists in the cluster. You must delete the existing Node or change the name of this new joining Node

To see the stack trace of this error execute with --v=5 or higher

错误分析1

/etc/kubernetes/kubelet.conf 已经存在

端口 10250 已经被占用

/etc/kubernetes/pki/ca.crt 已经存在

解决步骤

1. 删除现有的 Kubernetes 配置文件

删除或备份现有的 Kubernetes 配置文件，以便重新加入集群：

sudo mv /etc/kubernetes/kubelet.conf /etc/kubernetes/kubelet.conf.backup

sudo mv /etc/kubernetes/pki/ca.crt /etc/kubernetes/pki/ca.crt.backup

2. 释放被占用的端口

检查并终止占用端口 10250 的进程：

sudo lsof -i :10250

找到占用该端口的进程 ID 并终止它：

sudo kill -9 <PID>

3. 重新运行 kubeadm join 命令

在解决了上述问题后，重新运行 kubeadm join 命令：

sudo kubeadm join 192.168.1.152:6443 --token m3vr25.g24757vwacsqf1uk --discovery-token-ca-cert-hash sha256:bd4f1d5c16a99da7be0b35438862b7d072b2dd79972d05ce13c925616e99e410忽略预检错误（可选）

如果你确定这些文件和端口的存在不会影响节点加入集群，可以使用 --ignore-preflight-errors 参数忽略这些错误：

sudo kubeadm join 192.168.1.152:6443 --token m3vr25.g24757vwacsqf1uk --discovery-token-ca-cert-hash sha256:bd4f1d5c16a99da7be0b35438862b7d072b2dd79972d05ce13c925616e99e410 --ignore-preflight-errors=FileAvailable--etc-kubernetes-kubelet.conf,Port-10250,FileAvailable--etc-kubernetes-pki-ca.crt

常见错误2：  
node重新加入error execution phase kubelet-start: error uploading crisocket: Unauthorized

则执行：  
sudo kubeadm reset

sudo rm /etc/cni/net.d/\* -f  
sudo systemctl daemon-reload  
sudo systemctl restart kubelet  
sudo kubeadm join 192.168.1.152:6443 --token m3vr25.g24757vwacsqf1uk --discovery-token-ca-cert-hash sha256:bd4f1d5c16a99da7be0b35438862b7d072b2dd79972d05ce13c925616e99e410

5. Flannel网络工具

在集群内安装此工具来实现不同主机间网络互联，只需要在主节点上操作即可：

配置文件如下  
apiVersion: v1

kind: Namespace

metadata:

  labels:

    k8s-app: flannel

    pod-security.kubernetes.io/enforce: privileged

  name: kube-flannel

---

apiVersion: v1

kind: ServiceAccount

metadata:

  labels:

    k8s-app: flannel

  name: flannel

  namespace: kube-flannel

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

  labels:

    k8s-app: flannel

  name: flannel

rules:

- apiGroups:

  - ""

  resources:

  - pods

  verbs:

  - get

- apiGroups:

  - ""

  resources:

  - nodes

  verbs:

  - get

  - list

  - watch

- apiGroups:

  - ""

  resources:

  - nodes/status

  verbs:

  - patch

- apiGroups:

  - networking.k8s.io

  resources:

  - clustercidrs

  verbs:

  - list

  - watch

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

  labels:

    k8s-app: flannel

  name: flannel

roleRef:

  apiGroup: rbac.authorization.k8s.io

  kind: ClusterRole

  name: flannel

subjects:

- kind: ServiceAccount

  name: flannel

  namespace: kube-flannel

---

apiVersion: v1

data:

  cni-conf.json: |

    {

      "name": "cbr0",

      "cniVersion": "0.3.1",

      "plugins": [

        {

          "type": "flannel",

          "delegate": {

            "hairpinMode": true,

            "isDefaultGateway": true

          }

        },

        {

          "type": "portmap",

          "capabilities": {

            "portMappings": true

          }

        }

      ]

    }

  net-conf.json: |

    {

      "Network": "10.122.0.0/16",

      "Backend": {

        "Type": "vxlan"

      }

    }

kind: ConfigMap

metadata:

  labels:

    app: flannel

    k8s-app: flannel

    tier: node

  name: kube-flannel-cfg

  namespace: kube-flannel

---

apiVersion: apps/v1

kind: DaemonSet

metadata:

  labels:

    app: flannel

    k8s-app: flannel

    tier: node

  name: kube-flannel-ds

  namespace: kube-flannel

spec:

  selector:

    matchLabels:

      app: flannel

      k8s-app: flannel

  template:

    metadata:

      labels:

        app: flannel

        k8s-app: flannel

        tier: node

    spec:

      affinity:

        nodeAffinity:

          requiredDuringSchedulingIgnoredDuringExecution:

            nodeSelectorTerms:

            - matchExpressions:

              - key: kubernetes.io/os

                operator: In

                values:

                - linux

      containers:

      - args:

        - --ip-masq

        - --kube-subnet-mgr

        - --iface=ens9f1np1

        command:

        - /opt/bin/flanneld

        env:

        - name: POD\_NAME

          valueFrom:

            fieldRef:

              fieldPath: metadata.name

        - name: POD\_NAMESPACE

          valueFrom:

            fieldRef:

              fieldPath: metadata.namespace

        - name: EVENT\_QUEUE\_DEPTH

          value: "5000"

        image: docker.m.daocloud.io/flannel/flannel-cni-plugin:v1.4.0-flannel1

        name: kube-flannel

        resources:

          requests:

            cpu: 100m

            memory: 50Mi

        securityContext:

          capabilities:

            add:

            - NET\_ADMIN

            - NET\_RAW

          privileged: false

        volumeMounts:

        - mountPath: /run/flannel

          name: run

        - mountPath: /etc/kube-flannel/

          name: flannel-cfg

        - mountPath: /run/xtables.lock

          name: xtables-lock

      hostNetwork: true

      initContainers:

      - args:

        - -f

        - /opt/bin/flanneld

        - /opt/cni/bin/flannel

        command:

        - cp

        image: docker.m.daocloud.io/flannel/flannel:v0.24.2        name: install-cni-plugin

        volumeMounts:

        - mountPath: /opt/cni/bin

          name: cni-plugin

      - args:

        - -f

        - /etc/kube-flannel/cni-conf.json

        - /etc/cni/net.d/10-flannel.conflist

        command:

        - cp

        image: docker.m.daocloud.io/flannel/flannel:v0.24.2

        name: install-cni

        volumeMounts:

        - mountPath: /etc/cni/net.d

          name: cni

        - mountPath: /etc/kube-flannel/

          name: flannel-cfg

      priorityClassName: system-node-critical

      serviceAccountName: flannel

      tolerations:

      - effect: NoSchedule

        operator: Exists

      volumes:

      - hostPath:

          path: /run/flannel

        name: run

      - hostPath:

          path: /opt/cni/bin

        name: cni-plugin

      - hostPath:

          path: /etc/cni/net.d

        name: cni

      - configMap:

          name: kube-flannel-cfg

        name: flannel-cfg

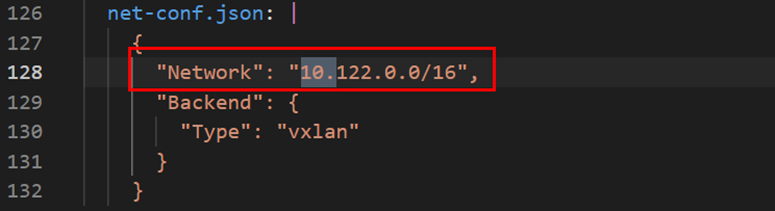
      - hostPath:

          path: /run/xtables.lock

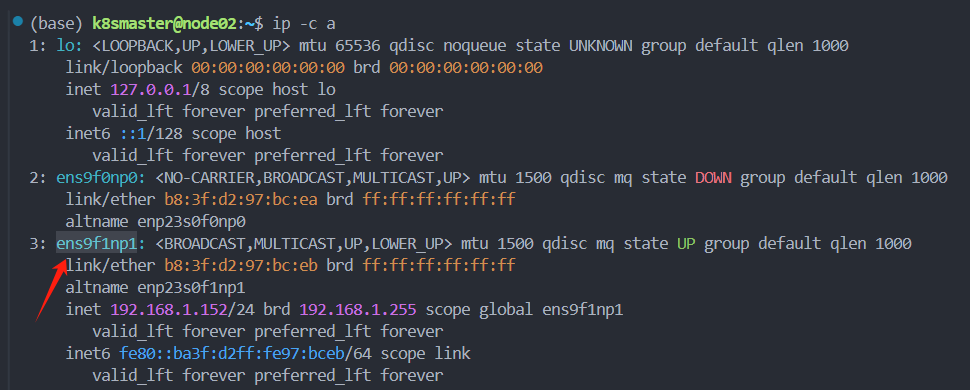
          type: FileOrCreate

        name: xtables-lock

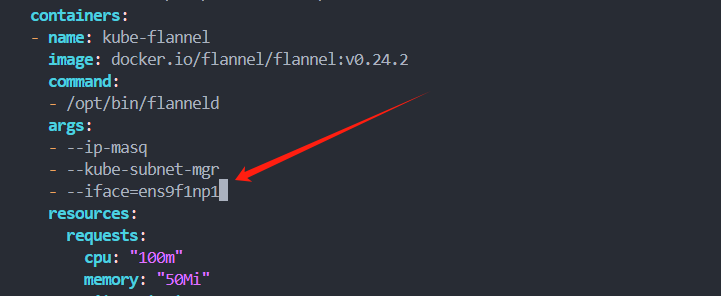
需要修改其中的网络IP范围与之前配置的Pod IP范围一致，具体如下：



根据本机的地址配置interface：



在flannel.yaml的containers中修改添加这一行：

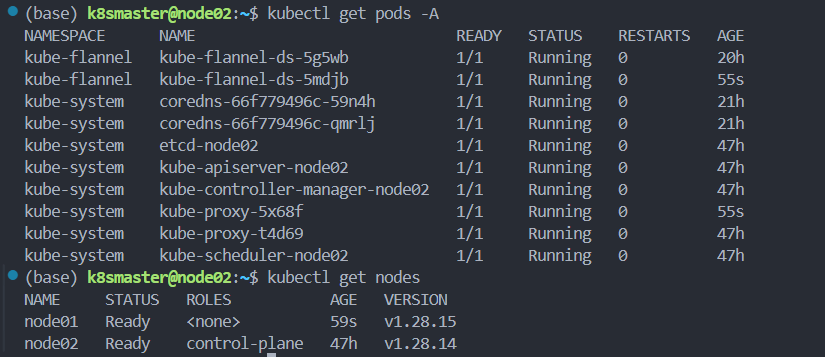


Ctrl+F找到image地址，换成  
docker.m.daocloud.io/flannel/flannel-cni-plugin:v1.4.0-flannel1  
docker.m.daocloud.io/flannel/flannel:v0.24.2

docker.m.daocloud.io/flannel/flannel:v0.24.2

在集群中容器化部署即可：

kubectl apply -f kube-flannel.yml



6. NFS文件系统

NFS在集群中有两个角色:一个NFS服务器和多个NFS客户机。网络文件系统(NFS)是一种分布式文件系统协议，它允许NFS 客户机的主机通过计算机网络像访问本地存储一样访问存储在NFS服务器主机中的文件。

（1）NFS服务器安装配置

1）在选定的NFS服务器上选择NFS文件系统根目录：

目前平台选定的NFS服务器为：192.168.1.152，NFS文件系统根目录为：/home/NFSshare

为该目录赋予读写权限：

sudo chmod -R 777 /home/NFSshare

sudo mkdir -p /home/NFSshare/k8s

sudo chown -R nobody:nogroup /home/NFSshare/k8s/

sudo chmod -R 777 /home/NFSshare/k8s/

2）安装nfs-kernel-server 和 portmap：

sudo apt install nfs-kernel-server

sudo apt install portmap

3）配置portmap：

进入文件：sudo nano /etc/default/portmap

输入配置：

-i 127.0.0.1

4）配置export:

配置内容包括：a）NFS 服务器允许挂载的主机(即NFS -client的ip地址范围)；b）修改NFS根路径内容时NFS 客户机的权限

进入文件：sudo vim /etc/exports

a) 配置如下：

<NFS根目录> <NFS客户机合法IP范围> (<NFS客户机权限>)

当前平台配置如下：

/home/NFSshare/k8s/ 192.168.1.0/24 (rw,sync,no\_root\_squash,no\_subtree\_check)

b) 启用配置：

sudo exportfs -a

重启后NFS服务生效：

sudo /etc/init.d/nfs-kernel-server restart

（2）NFS客户机

对平台集群中除了NFS服务器以外的每台主机都需要进行NFS客户机的配置来使用NFS根目录的文件。

1）安装nfs-common依赖包

sudo apt install -y nfs-common

2）在NFS客户机上新建与NFS根目录相同的路径。

mkdir -p <NFS文件系统根目录>

例如，在本平台指令如下：

mkdir -p /home/NFSshare

3）与NFS服务器的NFS文件系统根目录绑定：

sudo mount <NFS服务器 ip>:<NFS文件系统根目录路径名> <NFS客户机相同文件路径名>

例如在本平台绑定如下：

sudo mount 192.168.1.152:/home/NFSshare /home/NFSshare

7. 设置Kubernetes集群中默认的存储类型为NFS：

在平台中的配置文件可见：environment-configuration/nfs-storage-class/

具体包括三个文件：

1）1-nfs-client-sa.yaml

2）2-nfs-client-deploy.yaml

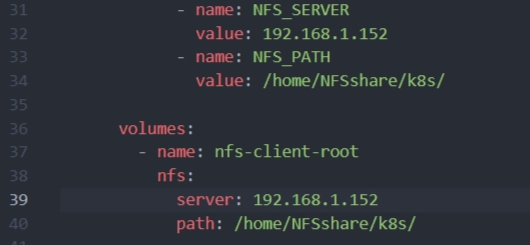
3）3-nfs-client-class.yaml

配置方式如下：

（1）为存储类型配置service account和对应的cluter role来配置权限：

kubectl apply -f /home/k8smaster/k8s\_create/nfs-storage-class/1-nfs-client-sa.yaml  
（2）配置相关控制器，部署为容器：

配置文件为2-nfs-client-deploy.yaml，注意要修改32行，39行为当前平台配置的NFS系统服务器IP：



修改配置后运行命令如下：

kubectl apply -f /home/k8smaster/k8s\_create/nfs-storage-class/2-nfs-client-deploy.yaml

（3）最后创建默认Kubernetes集群默认存储类型：

kubectl apply -f /home/k8smaster/k8s\_create/nfs-storage-class/3-nfs-client-class.yaml

常见错误  
Warning FailedScheduling 79s default-scheduler 0/2 nodes are available: 1 node(s) had untolerated taint {node-role.kubernetes.io/control-plane: }, 1 node(s) had untolerated taint {node.kubernetes.io/disk-pressure: }. preemption: 0/2 nodes are available: 2 Preemption is not helpful for scheduling.

这表明没有开启节点容忍污点，需要在deploy.yaml文件中添加

sudo vi /home/k8smaster/k8s\_create/nfs-storage-class/2-nfs-client-deploy.yaml  
在template下面添加  
 tolerations:

- key: node-role.kubernetes.io/control-plane

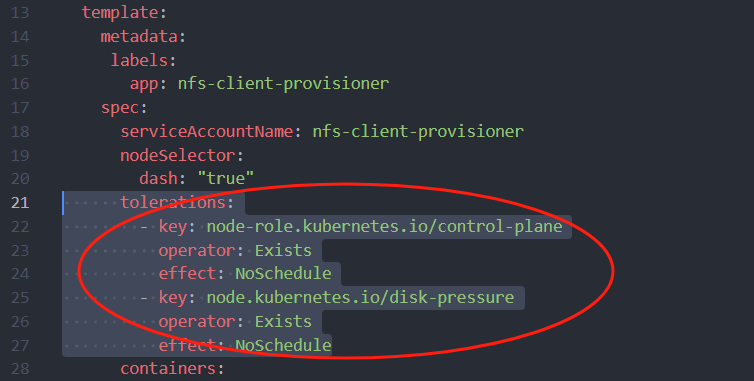
operator: Exists

effect: NoSchedule

- key: node.kubernetes.io/disk-pressure

operator: Exists

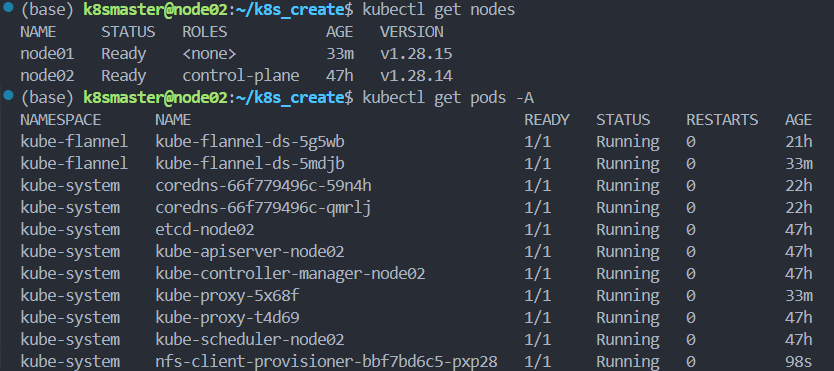
effect: NoSchedule



删除并重新部署

kubectl delete -f /home/k8smaster/k8s\_create/nfs-storage-class/2-nfs-client-deploy.yaml

kubectl apply -f /home/k8smaster/k8s\_create/nfs-storage-class/2-nfs-client-deploy.yaml



至此，基于containerd运行时的kubernetes1.28集群搭建成功

安装Prometheus、dcgm-exporter、node-exporter、grafana依次按照k8s-prometheus-grafana文件夹中的配置文件 kubectl apply -f