滴雨 K8S 容器云安装手册

环境准备

https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-init/

机器规划

Ip（例）	hostname	作用	注释
192.168.1.80	master	主节点	节点名字没有关系，不一定是Master,Worker什么的
192.168.1.81	node2	work 节点 1	
192.168.1.82	node1	work 节点 1	

内核升级

检查内核

1、Docker 要求 CentOS 系统的内核版本高于 3.10 ， 查看本页面的前提条件来验证你的 CentOS 版本是否支持 Docker 。

通过 uname -r 命令查看你当前的内核版本

$ uname -r

大版本升级

载入 elrepo 源，搜索内核更新资源，并进行更新操作。
具体实验步骤:

载入公钥
rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org

```
# 安装 ELRepo
rpm -Uvh http://www.elrepo.org/elrepo-release-7.0-3.el7.elrepo.noarch.rpm
# 载入 elrepo-kernel 元数据
yum --disablerepo=\* --enablerepo=elrepo-kernel repolist
# 查看可用的 rpm 包
yum --disablerepo=\* --enablerepo=elrepo-kernel list kernel*
# 安装最新版本的 kernel
yum --disablerepo=\* --enablerepo=elrepo-kernel install -y kernel-ml.x86_64
```

重启，选择新版本内核进入系统。

此时，操作系统使用的内核已升级为【4.15.0-1.el7.elrepo.x86_64】

内核工具包升级

```
# 删除旧版本工具包
yum remove kernel-tools-libs.x86_64 kernel-tools.x86_64
# 安装新版本工具包
yum --disablerepo=\* --enablerepo=elrepo-kernel install -y kernel-ml-tools.x86_64
```

原文链接：https://blog.csdn.net/breeze915/article/details/79243673

2、使用 root 权限登录 Centos。确保 yum 包更新到最新。

```
$ sudo yum update
```

3、卸载旧版本(如果安装过旧版本的话)

```
$ sudo yum remove docker docker-common docker-selinux docker-engine
```

4、安装需要的软件包，yum-util 提供yum-config-manager 功能，另外两个是 devicemapper 驱动依赖的

```
$ sudo yum install -y yum-utils device-mapper-persistent-data lvm2（安装 docker 时候重复）
```

修改开机时默认使用的内核

#使用 cat /boot/grub2/grub.cfg |grep menuentry 查看系统可用内核

注意：实体机可能是 /boot/efi/EFI/centos/grub.cfg

[root@bigapp-slave27 ~]# cat /boot/grub2/grub.cfg |grep menuentry

grub2-set-default 'CentOS Linux (3.10.0-327.el7.x86_64) 7 (Core)'

查看内核修改结果

[root@bigapp-slave27 ~]# grub2-editenv list
saved_entry=CentOS Linux (3.10.0-327.el7.x86_64) 7 (Core)

#查看系统安装了哪些内核包
[root@bigapp-slave27 ~]# rpm -qa |grep kernel
#使用 yum remove 或 rpm -e 删除无用内核
yum remove kernel-3.10.0-327.el7.x86_64

设置时区

set timezone
set timezone to Asia/Shanghai
timedatectl set-timezone Asia/Shanghai
write current time to hw
timedatectl set-local-rtc 0
restart time services
systemctl restart rsyslog
systemctl restart crond

如果网卡不起作用请参考
http://www.luyixian.cn/news_show_399493.aspx


r8168-8.048.03.tar.bz2

安装配置

配置 hosts

```
cat <<EOF >>/etc/hosts
192.168.1.80 master
192.168.1.81 node1
```

```
192.168.1.82 node2
EOF
```

关闭防火墙

```
systemctl disable firewalld.service && systemctl stop firewalld.service
# 检查
systemctl status firewalld
```

禁用 SELINXU

```
setenforce 0
sed -i 's/^SELINUX=enforcing$/SELINUX=disabled/' /etc/selinux/config
# 检查
cat /etc/selinux/config
```

禁用分区

```
swapoff -a && sysctl -w vm.swappiness=0
sed -ri '/^[^#]*swap/s@^@#@' /etc/fstab
# 检查
cat /etc/fstab
free -h
```

网桥 kernel 模块加载

```
# 检查
lsmod |grep ip_vs
lsmod |grep netfilter
# 加载
modprobe br_netfilter
modprobe ip_vs
# 启动加载
cat > /etc/modules-load.d/ipvs.conf << EOF
ip_vs
br_netfilter
EOF
systemctl enable --now systemd-modules-load.service
```

```
# 检查
systemctl status systemd-modules-load.service
```

网桥 kernel 配置

```
cat <<EOF > /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward=1
# 4.12 版本以上注释掉以下一条：
# net.ipv4.tcp_tw_recycle=0
vm.swappiness=0
vm.overcommit_memory=1
vm.panic_on_oom=0
# fs.inotify.max_user_watches=89100
fs.file-max=52706963
fs.nr_open=52706963
net.ipv6.conf.all.disable_ipv6=1
net.netfilter.nf_conntrack_max=2310720
EOF

sysctl -p /etc/sysctl.d/k8s.conf
```

安装 docker

修改 docker 源

```
yum install -y yum-utils device-mapper-persistent-data lvm2
# docker 源
yum-config-manager \
--add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

安装 docker（每台机器）

所有节点都需要安装 docker，可以安装最新 19.03.9
每个节点都需要使 docker 开机自启

```
yum list docker-ce --showduplicates
#  安装 k8s 支持的版本，在具体的 CHANGLOG-XX.md 中查找
# https://github.com/kubernetes/kubernetes/tree/master/CHANGELOG
# DOCKER_VERSION=18.09.9
DOCKER_VERSION=18.06.3.ce
yum install -y docker-ce-$DOCKER_VERSION \
docker-ce-cli-$DOCKER_VERSION \
containerd.io
```

安装最新版：

```
yum install docker-ce docker-ce-cli containerd.io
```

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

配置 docker

```
系统默认的资源分配是 CGGROUP, 需改成 systemd, 注意 JSON 格式，复制到网上检查格式是否正确，否则影响 DOCKER 启动。

mkdir /etc/docker
cat > /etc/docker/daemon.json <<EOF
{
   "registry-mirrors": ["https://registry.docker-cn.com"],
   "exec-opts": ["native.cgroupdriver=systemd"],
   "log-driver": "json-file",
   "log-opts": {
  "max-size": "100m"
   },
   "storage-driver": "overlay2",
   "storage-opts": [
  "overlay2.override_kernel_check=true"
   ]

}
EOF

systemctl restart docker
```

原文：https://kubernetes.io/docs/setup/production-environment/container-runtimes/
KUBENETES 会支持多种容器，docker, CRI 等，我们选择 docker.

```
# (Install Docker CE)
## Set up the repository
### Install required packages
yum install -y yum-utils device-mapper-persistent-data lvm2
```

```
## Add the Docker repository
yum-config-manager --add-repo \
  https://download.docker.com/linux/centos/docker-ce.repo
```

```
# Install Docker CE
yum update -y && yum install -y \
  containerd.io-1.2.13 \
  docker-ce-19.03.8 \
  docker-ce-cli-19.03.8
```

```
## Create /etc/docker
mkdir /etc/docker
```

```
# Set up the Docker daemon
cat > /etc/docker/daemon.json <<EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2",
  "storage-opts": [
    "overlay2.override_kernel_check=true"
  ]
}
EOF
```

```
mkdir -p /etc/systemd/system/docker.service.d
```

```
# Restart Docker
systemctl daemon-reload
systemctl restart docker
```

安装 k8s

kubernetes 源配置

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

安装 kubelet kubeadm kubectl（所有节点执行）

kubelet 运行在 Cluster 所有节点上，负责启动 Pod 和容器。

kubeadm 用于初始化 Cluster,。

install kube base images

kube base image download url https://download.csdn.net/download/joeyfu/12379340

用来检查，合适的版本

kubeadm config images list

W0411 16:44:06.032313 2168 configset.go:202] WARNING: kubeadm cannot validate component configs for API groups [kubelet.config.k8s.io kubeproxy.config.k8s.io]

k8s.gcr.io/kube-apiserver:v1.18.1

k8s.gcr.io/kube-controller-manager:v1.18.1

k8s.gcr.io/kube-scheduler:v1.18.1

k8s.gcr.io/kube-proxy:v1.18.1

k8s.gcr.io/pause:3.2

k8s.gcr.io/etcd:3.4.3-0

k8s.gcr.io/coredns:1.6.7

原文链接：https://blog.csdn.net/joeyfu/article/details/106042910

kubectl 是 Kubernetes 命令行工具。通过 kubectl 可以部署和管理应用，查看各种资源，创建、删除和更新各种组件。

```
yum list |grep kubelet   # 检查确认 docker 版本匹配
yum list kubelet --showduplicates -y

#K8S_VERSION=1.14.10-0 (旧版本不用)

K8S_VERSION=1.18.3-0
yum install -y kubelet-$K8S_VERSION kubeadm-$K8S_VERSION \
kubectl-$K8S_VERSION --disableexcludes=kubernetes


check:

yum list installed |grep kube
```

启动 kubelet
此时，还不能启动 kubelet，因为此时配置还不能，现在仅仅可以设置开机自启动

```
systemctl enable --now kubelet
```

Master 节点：

kubeadm config images pull # 拉取集群所需镜像，这个需要翻墙（新版本此大步不需要，直接从后面的 kubeadm init 开始）

```
# 集群所需镜像
kubeadm config images list      （#注 用新版本 18.1）
```

```
[root@diyu90 ~]# kubeadm config images list
W0529 09:46:32.050185     11047 configset.go:202] WARNING: kubeadm cannot validate component configs for API groups [kubelet.config.k8s.io kubeproxy.config.k8s.io]
k8s.gcr.io/kube-apiserver:v1.18.3
k8s.gcr.io/kube-controller-manager:v1.18.3
k8s.gcr.io/kube-scheduler:v1.18.3
k8s.gcr.io/kube-proxy:v1.18.3
k8s.gcr.io/pause:3.2
k8s.gcr.io/etcd:3.4.3-0
k8s.gcr.io/coredns:1.6.7
```

使用 kubeadm 创建集群

```
# 初始化 Master（Master 需要至少 2 核 2G 内存）此处会各种报错,异常...成功与否就在此，没有验证过少于 2 核
# --apiserver-advertise-address 指定与其它节点通信的接口
# --pod-network-cidr 指定 pod 网络子网，使用 fannel 网络必须使用这个 CIDR
192.168.1.80————>MASTER 地址，检查填上 kubernetes 版本

kubeadm init \
--apiserver-advertise-address=192.168.1.90 \
--image-repository registry.aliyuncs.com/google_containers \
--kubernetes-version v1.18.1 \
--service-cidr=10.1.0.0/16 \
--pod-network-cidr=10.254.0.10/16

#以上这条不用，我们用本地 192.168.1.10:5000 私有仓库 registry, 用 docker pull 192.168.1.10:5000,然后 改 TAG，见附录，如何拉取 IMAGE 和执行命令）
```

注意:
在使用 Kubeadm init 命令之后，要把提示中的 TOKEN 记下来，以备后用，如果没记，参见其它命令行的方式。

```
# 使用 kubeadm init 中最后的提示执行(之前 KUBEADMIN 执行时复制)
kubeadm join 192.168.1.80:6443 --token w2i0mh.5fxxz8vk5k8db0wq \
  --discovery-token-ca-cert-hash \      －－>在 Worker 工作节点中输主节点地址


kubectl 是管理 Kubernetes Cluster 的命令行工具，前面我们已经在所有的节点安装了 kubectl。Master 初始化完成后需要做一些配置工作，然后 kubectl 就能使用了。
```

为了使用更便捷，启用 kubectl 命令的自动补全功能。

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

下面四步在 Master 上执行：

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

kubectl get pods -n kube-system
```

kubectl get nodes

此时应显示：

```
kubectl get nodes
NAME        STATUS     ROLES      AGE     VERSION
diyu90       Ready      master        16h     v1.18.3
```

pod coredns 可能会出现会出现 pending 情况，可以先不考虑。
或者先安装 flannel 网络

node 端加入

确保 Kubelet 已启动

每一个 node 的 kubelet 都必须进去设置 cgroup-drive 和 swap 关闭的启动选项，跟主控节点一样（参见上民面安装 Kubernetes 的方法），然后需要重新启动，通过 systemctl daemon-reload/systemctl restart kubelet 来进行。

如果 node 上显示添加成功，但 Master 上显示不出来，在 node 机上使用 systemctl status kubelet 查看下服务的状态，检查里面的各项状态，单独处理。

```
# Node 端：
mkdir -p $HOME/.kube


# master 上远程复制到其它节点
scp $HOME/.kube/config 192.168.1.81:$HOME/.kube/config

scp $HOME/.kube/config 192.168.1.82:$HOME/.kube/config

# The connection to the server localhost:8080 was refused - did you specify the right host or port?

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



    # 使用 kubeadm init 中最后的提示执行(之前 KUBEADMIN 执行时复制)
kubeadm join 192.168.1.80:6443 --token w2i0mh.5fxxz8vk5k8db0wq \
  --discovery-token-ca-cert-hash \
 sha256:65e82e987f50908f3640df7e05c7a91f390a02726c9142808faa739d4dc24252
```

或是

如果没有复制之前的提示信息，通过以下命令获得：

从主节点复制命令产生的一条命令，如下例：

[root@diyu90 ~]# kubeadm token create $(kubeadm token generate) --print-join-command --ttl=24h

kubeadm join 192.168.1.80:6443 --token jskc16.tljbqy4rn96m7wce --discovery-token-ca-cert-hash sha256:c4a7ceccd7dca307a473a344006bc90d358a7e28160562971257l7f2aab3fb7d

以上命令复制到工作节点 node 上执行时，就可以让该节点加入集群。由下面标记角色。

kubectl label nodes k8s-node2 node-role.kubernetes.io/node=

一般情况，Master 节点，不参于调度 PODS，如果需要在 MASTER 起用户容器，用 drain 等命令。

初始化失败时解决方案

如果初始化失败，或发生其它节点未能加入集群，请使用如下代码清除后重新初始化(无需执行)
重置 kubeadm
如果初始化是出现问题，每个节点重置命令如下

```
kubeadm reset
rm -rf /var/lib/cni
rm -f $HOME/.kube/config
systemctl daemon-reload
systemctl restart kubelet

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

安装 pod 网络

要让 Kubernetes Cluster 能够工作，必须安装 Pod 网络，否则 Pod 之间无法通信。

Kubernetes 支持多种网络方案，这里我们先使用 flannel

补充 flannel 网络介绍
Flannel 是 CoreOS 团队针对 Kubernetes 设计的一个网络规划服务，简单来说，它的功能是让集群中的不同节点主机创建的 Docker 容器都具有全集群唯一的虚拟 IP 地址。
但在默认的 Docker 配置中，每个节点上的 Docker 服务会分别负责所在节点容器的 IP 分配。这样导致的一个问题是，不同节点上容器可能获得相同的内外 IP 地址。并使
这些容器之间能够之间通过 IP 地址相互找到，也就是相互 ping 通。

Flannel 的设计目的就是为集群中的所有节点重新规划 IP 地址的使用规则，从而使得不同节点上的容器能够获得"同属一个内网"且"不重复的"IP 地址，并让属于不同节点上的容器能够直接通过内网 IP 通信。

部署 flannel（每个机器都要装）

wget https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

kube-flannel.yml

由于不能访问 raw，直接访问 github.com，找到文件后，复制粘贴到 vi 中。
kubectl apply -f kube-flannel.yml （只在 MASTER 上运行）

每个节点重启 kubelet

systemctl restart kubelet

https://github.com/coreos/flannel/blob/master/Documentation/kubernetes.md

这个链接的一个重点是，检查 Flannel.yml 中的 CIDR 和 kubeadm 中输入的 POD network cidr 相一致。
The `network` in the `flannel` configuration should match the pod network CIDR. The choice of `backend` is also made here and defaults to VXLAN

master 端检查

```
kubectl get pods --all-namespaces
```

这里其实需要等一会，这个 node1 节点才会变成 Ready 状态，因为 node 节点需要下载四个镜像 flannel coredns kube-proxy pause

过了一会查看节点状态

```
[root@ken ~]# kubectl get nodes
NAME      STATUS    ROLES     AGE       VERSION
host1     Ready     <none>    4m15s     v1.13.2
host2     Ready     <none>    3m37s     v1.13.2
ken       Ready     master    39m       v1.13.2
```

K8S ingress

k8s 现在提供三种暴露服务的方式：LoadBlancer、NodePort 、Ingress。
所谓四层负载均衡，也就是主要通过报文中的目标地址和端口，再加上负载均衡设备设置的服务器选择方式，决定最终选择的内部服务器。service 是 k8s 暴露 http 服务的默认方式，其中 NodePort 类型可以将 http 服务暴露在宿主机的端口上，以便外部可以访问
模式缺点
一个 app 需要占用一个主机端口，NodePort 会在每个 node 上暴露对应的 port，不便管理
端口缺乏管理
L4 转发， 无法根据 http header 和 path 进行路由转发

所谓七层负载均衡，也称为“内容交换”，也就是主要通过报文中的真正有意义的应用层内容，再加上负载均衡设备设置的服务器选择方式，决定最终选择的内部服务器。在 service 之前加了一层 ingress，增加了 7 层的识别能力，可以根据 http header, path 进行路由转发。
Ingress 的实现分为两个部分 Ingress Controller 和 Ingress .
Ingress Controller 是流量的入口，是一个实体软件， 一般是 Nginx 和 Haproxy 。
Ingress 描述具体的路由规则。
Ingress Controller 会监听 api server 上的 /ingresses 资源 并实时生效。
Ingerss 描述了一个或者多个 域名的路由规则，以 ingress 资源的形式存在。
简单说： Ingress 描述路由规则， Ingress Controller 实时实现规则。

端口管理。减少不必要端口暴露，便于管理。
所有的请求，通过 Ingress 对应的 IP：PORT 进入，过滤/转发/负载均衡到相应的 service/pod
动态配置服务
好的解决方案就是让外界通过域名去访问 Service，而无需关心其 Node IP 及 Port。那为什么不直接使用 Nginx？这是因为在 K8S 集群中，如果每加入一个服务，我们都在 Nginx 中添加一个配置，其实是一个重复性的体力活

以 Ingress Nignx 为例，实现原理如下。

Ingress Controller 通过与 Kubernetes API 交互，能够动态的获取 cluster 中 Ingress rules 的变化，生成一段 Nginx 配置，再写到 Nginx-ingress-control 的 Pod 里，reload pod 使规则生效。从而实现注册的 service 及其对应域名/IP/Port 的动态添加和解析。

1. 配置 ingress

2. 配置 ingress controller 及 configmap（ingress nginx 通过读取 configmap 和

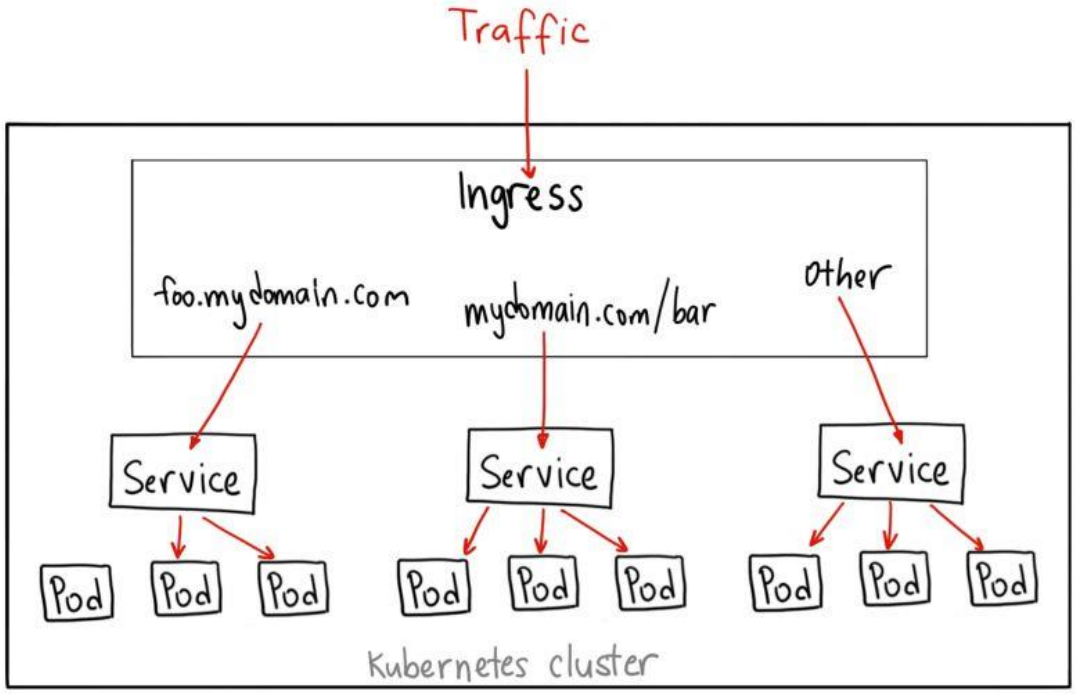
annotations 中的配置来配置 nginx

3. 配置相关权限，从而使 ingress resouce 可以访问 API Server，获取相关

cluster/service/pod 信息

4. 配置 ingress service

Ingress 事实上不是一种服务类型。相反，它处于多个服务的前端，扮演着“智能路由”或者集群入口的角色。
你可以用 Ingress 来做许多不同的事情，各种不同类型的 Ingress 控制器也有不同的能力。
GKE 上的默认 ingress 控制器是启动一个 HTTP(S) Load Balancer[3]。它允许你基于路径或者子域名来路由流量到后端服务。例如，你可以将任何发往域名 foo.yourdomain.com 的流量转到 foo 服务，将路径 yourdomain.com/bar/path 的流量转到 bar 服务。



Ingress 可能是暴露服务的最强大方式，但同时也是最复杂的。Ingress 控制器有各种类型，包括 Google Cloud Load Balancer， Nginx，Contour，Istio，等等。它还有各种插件，比如 cert-manager[5]，它可以为你的服务自动提供 SSL 证书。
如果你想要使用同一个 IP 暴露多个服务，这些服务都是使用相同的七层协议（典型如 HTTP），那么 Ingress 就是最有用的。如果你使用本地的 GCP 集成，你只需要为一个负载均衡器付费，且由于 Ingress 是“智能”的，你还可以获取各种开箱即用的特性（比如 SSL、认证、路由等等）

部署

下载部署

```
Ingress controller 可以只部署一个

git clone https://github.com/kubernetes/ingress-nginx.git
cd ingress-nginx
git checkout nginx-0.26.1
修改  ingress-nginx/deploy/static/mandatory.yaml
# 使用宿主机网络：
#    在 serviceAccountName: nginx-ingress-serviceaccount  这句后面增加 hostNetwork: true
# 此文档使用 DaemonSet 部署，使每一个 node 节点都会有一个 ingress-controller 容器
sed -i -e '/serviceAccountName/i\          hostNetwork: true' \
   -e '/image/i\                imagePullPolicy:  IfNotPresent' \
   -e 's/Deployment/DaemonSet/' \
   -e 's/replicas/#replicas/' \
  deploy/static/mandatory.yaml

注意这个要拷贝整个目录，不要只复制单独的 mandatory.yaml


kubectl apply -f deploy/static/mandatory.yaml

kubectl get all -n ingress-nginx
```

检查

```
[root@master ingress-nginx]# kubectl get all -n ingress-nginx
NAME                                READY    STATUS    RESTARTS    AGE
pod/nginx-ingress-controller-7xlwb    1/1      Running    0          21s
pod/nginx-ingress-controller-c5gfh    1/1      Running    0          21s

NAME                                    DESIRED    CURRENT    READY    UP-TO-
DATE    AVAILABLE    NODE SELECTOR              AGE
daemonset.apps/nginx-ingress-
controller    2          2          2        2             2             kubernetes.io/os=linux    22
s
[root@master ingress-nginx]
```

测试验证

```
cat <<EOF >nginx.yaml
apiVersion: v1
```

```
kind: Service
metadata:
name: nginx
labels:
  app: nginx
spec:
type: NodePort
ports:
 - port: 80
  targetPort: 80
  nodePort: 30100
  protocol: TCP
selector:
  app: nginx
---
kind: Deployment
apiVersion: apps/v1
metadata:
name: nginx
spec:
replicas: 1
selector:
  matchLabels:
    app: nginx
template:
  metadata:
    name: nginx
    labels:
      app: nginx
  spec:
    containers:
     - name: nginx
       image: nginx
       imagePullPolicy: IfNotPresent
       ports:
        - containerPort: 80
EOF
kubectl apply -f nginx.yaml
kubectl get all -o wide
# kubectl delete -f nginx.yaml

cat <<EOF >nginx-ing.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
```

```
name: nginx-ingress
spec:
rules:
 - host: test.nginx.ingress
   http:
     paths:
      - path: /
        backend:
          serviceName: nginx
          servicePort: 80
EOF
#  注意此处的 serviceName 和 servicePort 的值
# host 部分为自定义的域名
kubectl apply -f nginx-ing.yaml

测试
curl -v http://192.168.1.71  -H 'host: test.nginx.ingress'
或是在 hosts 中增加
192.168.1.71  test.nginx.ingress
然后浏览器访问 http://test.nginx.ingress      https://test.nginx.ingress
```

增加其他 TCP，UDP 端口暴露

```
# https://www.cnblogs.com/hongdada/p/11491974.html
# https://blog.csdn.net/weixin_30810583/article/details/98614907
# https://blog.csdn.net/hxpjava1/article/details/86756970
cat <<EOF > tcp-services-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
name: tcp-services
namespace: ingress-nginx
data:
 2181: "kafka/kafka-zookeeper:2181"
 9092: "kafka/kafka:9092"
EOF
kubectl create -f tcp-services-configmap.yaml

cat <<EOF > udp-services-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
name: udp-services
```

```
namespace: ingress-nginx
data:
 53: "kube-system/kube-dns:53"
kubectl create -f udp-services-configmap.yaml
netstat -ano |grep 2181
```

遗留问题

kubernetes-dashboard 443 端口转发

支持 HTTPS 访问，HTTP 自动跳转到 HTTPS

https://blog.csdn.net/qianghaohao/article/details/99354304

创建和安装加密访问凭证

```
# 创建 自签名证书
mkdir -p /k8s/kubernetes/ssl/dashboard
cd /k8s/kubernetes/ssl/dashboard
openssl req -x509 -nodes -days 3650 -newkey rsa:2048 -keyout kube-dashboard.key -out
kube-dashboard.crt -subj "/CN=dashboard.dew.com/O=dashboard.dew.com"

# 使用生成的证书创建 k8s Secret 资源，下一步创建的 Ingress 会引用这个 Secret
kubectl create secret tls kube-dasboard-ssl --key kube-dashboard.key --cert kube-
dashboard.crt -n kube-system
```

创建 Ingress 资源对象

```
cat <<EOF >/k8s/yaml/ing-kubernetes-dashboard.yaml

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

  name: ingress-kube-dashboard

  namespace: kube-system

  annotations:

    # use the shared ingress-nginx Inginx Ingress Controller 根据该注解自动发现 Ingress

    kubernetes.io/ingress.class: "nginx"
```

```
    nginx.ingress.kubernetes.io/secure-backends: "true"
    nginx.ingress.kubernetes.io/ssl-passthrough:  "true"
spec:
  tls:
  - hosts:
    - dashboard.dew.com
    # secretName https 证书 Secret
    secretName: kube-dasboard-ssl
  rules:
  # host 对外访问的域名
  - host: dashboard.dew.com
    http:
      paths:
      - path: /dashboard
        backend:
          # 集群对外暴露的 Service 名称
          serviceName: kubernetes-dashboard
          servicePort: 443
EOF
```

K8S 网管的安装

kubernetes-dashboard 安装

```
## master 节点上执行
```

暂时在 / 下安装
```bash
（以下是原包，用户现场不推荐）
git clone https://github.com/kubernetes/dashboard.git
cd dashboard/src/deploy/recommended
或是

mkdir /k8s/yaml && cd /k8s/yaml
wget https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta5/aio/deploy/recommended.yaml
```

kubernetes-dashboard

采用本地的或直接复制粘贴
本地已有改好的 YAML，避免下载外网，再手工编辑（192.168.1.10：/srv 下面找）

```
vim kubernetes-dashboard.yaml
# 1、修改 Dashboard Service 网络类型
# spec:
#   type: NodePort
#   ports:
#     - port: 443
#       targetPort: 8443
#       nodePort: 30000
#
# 创建 dashboard

kubectl apply -f /k8s/yaml/kubernetes-dashboard.yaml
```

创建 dashboard-serviceaccount 的账号

```bash
创建 dashboard 管理用户
kubectl create serviceaccount dashboard-serviceaccount -n kube-system
绑定用户为集群管理用户
kubectl create clusterrolebinding dashboard-cluster-account \
--clusterrole=cluster-admin \
--serviceaccount=kube-system:dashboard-serviceaccount

下面两句无需执行
# kubectl delete serviceaccount dashboard-serviceaccount -n kube-system
```

kubectl delete clusterrolebinding dashboard-cluster-account
```

```bash
kubectl get secret -n kube-system |grep dashboard-serviceaccount-token
kubectl get all -n kube-system -o wide
```

访问 kubernetes-dashboard

```bash
kubectl get pods -n kube-system -o wide
kubectl get svc -n kube-system -o wide
这种 node,如果 node 为 192.168.1.83 则 https://192.168.1.83:30000.
如果已经部署过 flanneld 则所有集群中 ip:30000 都可以访问
Web token 登录采用以下语句获取到的 token
kubectl describe secret -n kube-system $(kubectl get secret -n kube-system |grep dashboard-serviceaccount-token | awk '{print $1}')

```

问题解决

dashboard 启动出错

```bash
docker logs xxx
kubectl logs kubernetes-dashboard-865b64d96f-g5f9t --namespace=kube-system
错误信息:
        Error while initializing connection to Kubernetes apiserver. This most likely means that the cluster is misconfigured
(e.g., it has invalid apiserver certificates or service account''s configuration) or the --apiserver-host param points to a server that does not exist.
Reason: Get https://10.254.0.1:443/version: x509: certificate is valid for 10.0.0.1, 127.0.0.1, 10.2.8.44, 10.2.8.65, 10.2.8.34, not 10.254.0.1
解决方法:
    api-server 中证书缺少 10.254.0.1 网址的授权. server-csr.json
    Apiserver hosts 绑定 ip 应该是 10.254.0.1, 默认 pods 网端是 10.254.0.0/16, 其中 10.254.0.1 会用来 kubenetes 的 clusterip

# 同步证书并重启服务,主机上执行
# server-csr.json 中 ip 列表增加 ip。
```
```

cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes server-csr.json | cfssljson -bare server
scp server-key.pem server.pem k8s-node1:/k8s/kubernetes/ssl/
systemctl restart kube-apiserver
systemctl restart kube-controller-manager
systemctl restart kube-scheduler

```

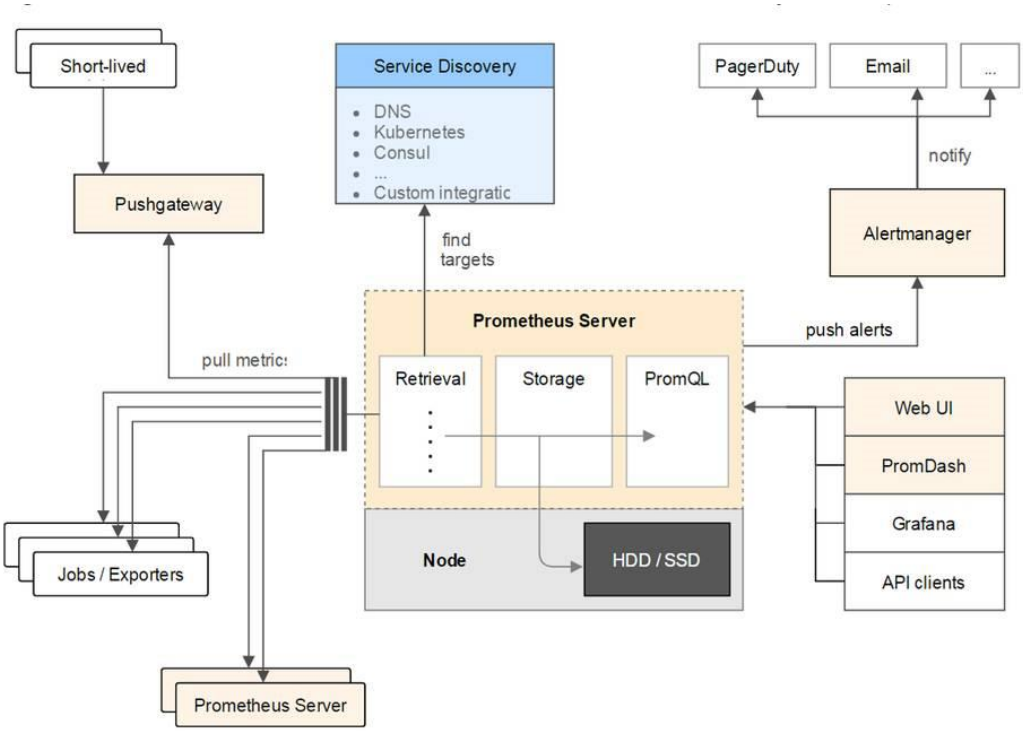
### chrome 无法登录
对主机做
```bash
mkdir -p /k8s/kubernetes/ssl/dashboard && cd /k8s/kubernetes/ssl/dashboard
#生成证书
openssl genrsa -out dashboard.key 2048
# 应该为可访问的 node 地址(网页服务器地址，MASTER 地址)
openssl req -new -out dashboard.csr -key dashboard.key -subj '/CN=192.168.1.80'
openssl x509 -req -in dashboard.csr -signkey dashboard.key -out dashboard.crt
# 检查
kubectl get secrets -n kube-system
# 删除原有的证书 secret
kubectl delete secret kubernetes-dashboard-certs -n kube-system
#创建新的证书 secret
kubectl create secret generic kubernetes-dashboard-certs --from-file=dashboard.key --from-file=dashboard.crt -n kube-system
#查看 pod
# kubectl get pod -n kube-system
kubectl get pod -n kube-system |grep kubernetes-dashboard
#重启 kubernetes-dashboard pod
kubectl delete pod \
  `kubectl get pod -n kubernetes-dashboard |grep kubernetes-dashboard |awk '{print $1}'` -n kubernetes-dashboard

至此可以在 chrome 上 htpps 登录，浏览器的 TOKEN 从上面命令取，暂时不用 KUBECONFIGURE
网页 token 登录采用以下语句获取到的 token
kubectl describe secret -n kube-system $(kubectl get secret -n kube-system |grep dashboard-serviceaccount-token | awk '{print $1}')
```
```

k8s 监控（proms,grafana)

https://www.cnblogs.com/yangxiaochu/p/10838570.html

配置原理



主机数据收集

主机数据的采集是集群监控的基础；外部模块收集各个主机采集到的数据分析就能对整个集群完成监控和告警等功能。一般主机数据采集和对外提供数据使用 cAdvisor 和 node-exporter 等工具。

node-exporter 运行在节点上采集节点主机本身的 cpu 和内存等使用信息，并对外提供获取主机性能开销的信息。

```
#安装 git，下载 yaml
git clone https://github.com/redhatxl/k8s-prometheus-grafana.git
```

```
#安装 node-exporter
kubectl create -f k8s-prometheus-grafana/node-exporter.yaml
```

node-exporter.yaml

完成对 kubernetes 的监控，监控收集数据一般有 PULL 和 PUSH 两种方式。PULL 方式是监控平台从集群中的主机上主动拉取采集到的主机信息，而 PUSH 方式是主机将采集到的信息推送到监控平台。常用的监控平台是 Prometheus,是采用 PULL 的方式采集主机信息

```
#安装 prometheus 组件
kubectl create -f k8s-prometheus-grafana/prometheus/rbac-setup.yaml
kubectl create -f k8s-prometheus-grafana/prometheus/configmap.yaml
kubectl create -f k8s-prometheus-grafana/prometheus/prometheus.deploy.yml
kubectl create -f k8s-prometheus-grafana/prometheus/prometheus.svc.yml
```

configmap.yaml

prometheus.deploy.yml

prometheus.svc.yml

rbac-setup.yaml

集群的整体信息已经收集汇总在 Prometheus 中，但 Prometheus 主要是对外提供数据获取接口，并不负责完成完善的图形展示，因此需要使用 DashBoard 工具对接 Prometheus 完成集群信息的图形化展示.

grafana 是一款采用 go 语言编写的开源应用，主要用于大规模指标数据的可视化展现，基于商业友好的 Apache License 2.0 开源协议。grafana 有热插拔控制面板和可扩展的数据源，目前已经支持绝大部分常用的时序数据库。

目前 grafana 支持的数据源

1. Graphite
2. Elasticsearch
3. CloudWatch
4. InfluxDB
5. OpenTSDB
6. Prometheus

```
#安装 grafana 组件
kubectl create -f    k8s-prometheus-grafana/grafana/grafana-deploy.yaml
kubectl create -f    k8s-prometheus-grafana/grafana/grafana-svc.yaml
kubectl create -f    k8s-prometheus-grafana/grafana/grafana-ing.yaml
```

```
[root@master2 ~]# kubectl get svc -n kube-system
NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
calico-typha           ClusterIP   10.108.235.193   <none>        5473/TCP         22h
grafana                NodePort    10.97.225.32     <none>        3000:31112/TCP   5h25m
kube-dns               ClusterIP   10.96.0.10       <none>        53/UDP,53/TCP    22h
kubernetes-dashboard   ClusterIP   10.103.205.123   <none>        443/TCP          22h
metrics-server         ClusterIP   10.98.146.172    <none>        443/TCP          22h
node-exporter          NodePort    10.97.174.113    <none>        9100:31672/TCP   5h31m
prometheus             NodePort    10.110.84.197    <none>        9090:30003/TCP   5h25m
traefik-web-ui         ClusterIP   10.102.184.161   <none>        80/TCP           22h
```



grafana-deploy.yaml


grafana-ing.yaml

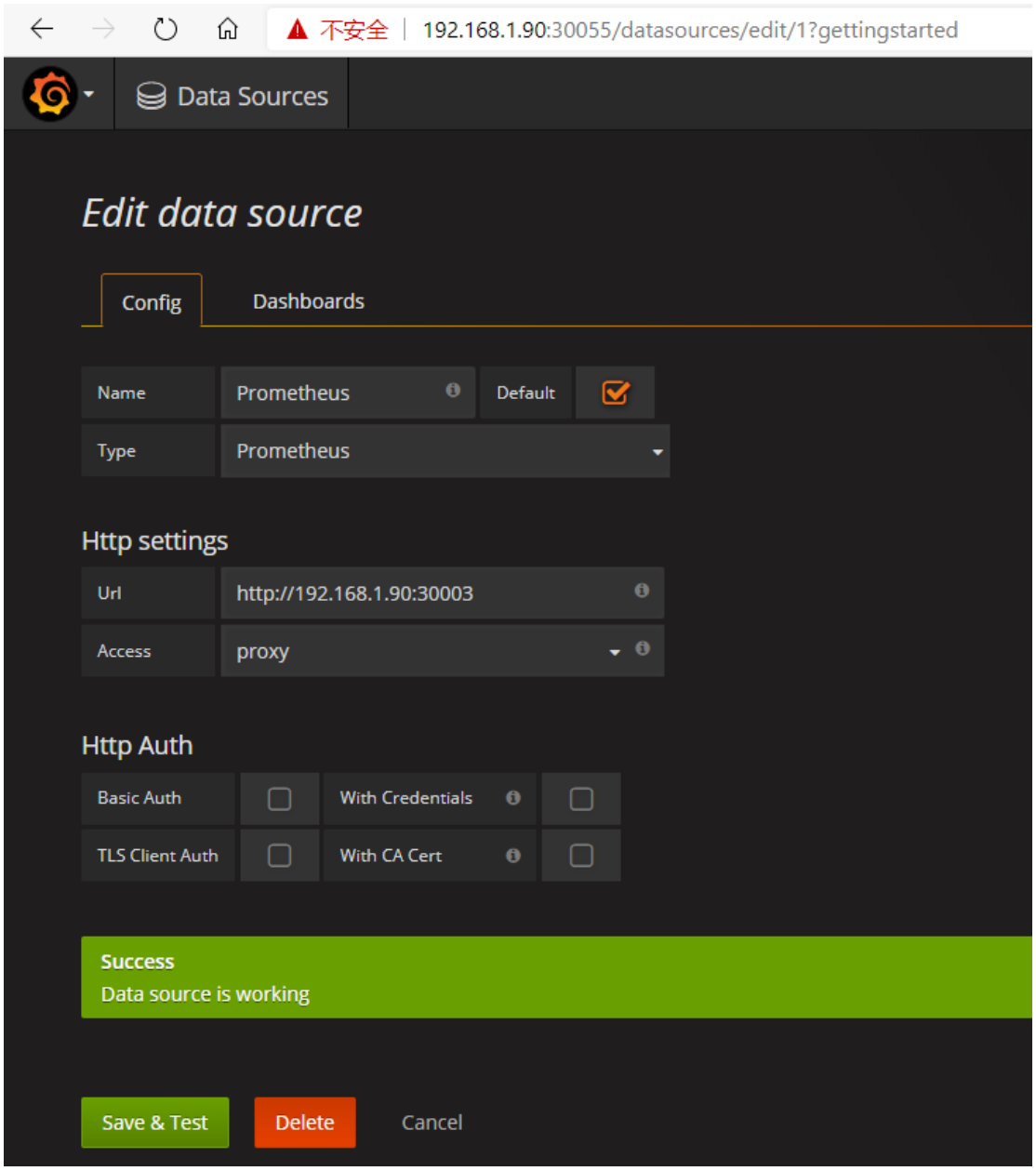

grafana-svc.yaml

kubectl get svc -n kube-system

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
grafana	NodePort	10.1.177.95	<none>	3000:30055/TCP	18m ————>取 30055
prometheus	NodePort	10.1.8.150	<none>	9090:30003/TCP	85s ----------> 30003

http://192.168.1.80:30055/?orgId=1 grafana

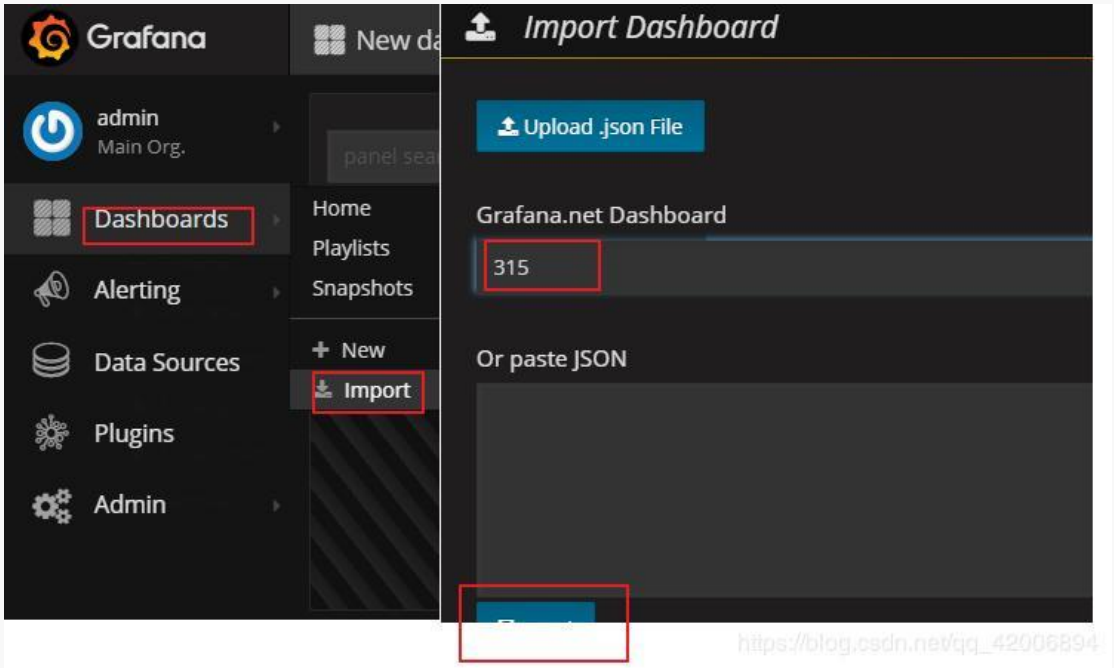
http://192.168.1.80:3003 prometheus

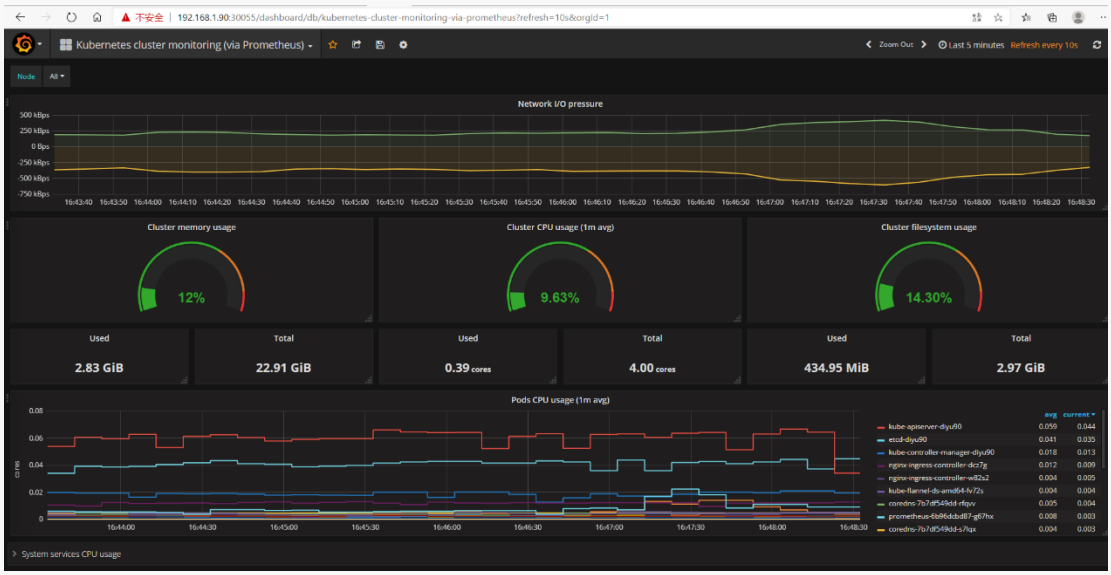


kubectl get svc -n kube-system
grafana admin/admin 登录
配置 Prometheus 数据源 30003 端口

https://grafana.com/grafana/dashboards/315

导入 315 的 dashboard






```
kubectl create secret generic cephfs-secret --type="kubernetes.io/cephfs" \
--from-literal=key=$(ceph  auth print-key  client.admin)  \
--namespace=kube-system
```

pv-pvc

```
kubectl create -f k8s-prometheus-grafana/ceph-fs-pv.yml  --namespace=kube-system
kubectl create -f k8s-prometheus-grafana/ceph-fs-pvc.yml  --namespace=kube-system
```

清除

```
kubectl delete -f    k8s-prometheus-grafana/grafana/grafana-ing.yaml
kubectl delete -f    k8s-prometheus-grafana/grafana/grafana-svc.yaml
kubectl delete -f    k8s-prometheus-grafana/grafana/grafana-deploy.yaml
#安装 grafana 组件
kubectl delete -f k8s-prometheus-grafana/prometheus/prometheus.svc.yml
kubectl delete -f k8s-prometheus-grafana/prometheus/prometheus.deploy.yml
kubectl delete -f k8s-prometheus-grafana/prometheus/configmap.yaml
#安装 prometheus 组件
kubectl delete -f k8s-prometheus-grafana/prometheus/rbac-setup.yaml
kubectl delete -f k8s-prometheus-grafana/node-exporter.yaml
```

k8s 集群使用 ceph rbd 块存储

ceph 支持对象存储，文件系统及块存储，是三合一存储类型。

kubernetes 的样例中有 cephfs 与 rbd 两种使用方式的介绍，cephfs 需要 node 节点安装 ceph 才能支持，rbd 需要 node 节点安装 ceph-common 才支持。 使用上的区别如下：

Volume Plugin	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
CephFS	✓	✓	✓
RBD	✓	✓	-

[Access Mode] (接入模式): **ReadWriteOnce**（RWO）：读写权限，但是只能被单个节点挂载 **ReadOnlyMany**（ROX）：只读权限，可以被多个节点挂载 **ReadWriteMany**（RWX）：读写权限，可以被多个节点挂载

[persistentVolumeReclaimPolicy]（回收策略: **Retain**（保留）- 保留数据，不会再分配给 pvc,需要管理员手工清理数据 **Recycle**（回收）- 清除 PV 中的数

据，保留 pv 资源,可以留供其他 pvc 使用 **Delete**（删除）- 删除整个 pv 资源及内部的数据。

安装 ceph-common

```
cat <<EOF > /etc/yum.repos.d/ceph.repo
[ceph]
name=ceph
baseurl=http://mirrors.aliyun.com/ceph/rpm-luminous/el7/x86_64/
gpgcheck=0
priority=1

[ceph-noarch]
name=cephnoarch
baseurl=http://mirrors.aliyun.com/ceph/rpm-luminous/el7/noarch/
gpgcheck=0
priority=1

[ceph-source]
name=Ceph source packages
baseurl=http://mirrors.aliyun.com/ceph/rpm-luminous/el7/SRPMS
enabled=0
gpgcheck=1
type=rpm-md
gpgkey=http://mirrors.aliyun.com/ceph/keys/release.asc
priority=1
EOF

yum install -y yum-utils \
&& yum-config-manager --add-repo https://dl.fedoraproject.org/pub/epel/7/x86_64/ \
&& yum install --nogpgcheck -y epel-release \
&& rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-7 \
&& rm -f /etc/yum.repos.d/dl.fedoraproject.org*

yum clean metadata && yum makecache

yum install -y ceph-common
```

StorageClass,PersistentVolume,PersistentVolumeClaim

- **StorageClass** – allows for dynamic provisioning of PersistentVolumes
- **PersistentVolume** – the low level representation of a storage volume
- **PersistentVolumeClaim** – the binding between a Pod and PersistentVolume
- **Pod** – a running container that will consume a PersistentVolume
- **Volume Driver** – the code used to communicate with the backend storage provider

Storage ceph RBD 使用

创建 rbd pool

```
# 确认 rbd pool 已创建
ceph osd pool ls
# 创建 rbd pool
ceph osd pool create rbd 32
# 确认 rbd pool 已创建
ceph df
## 检查用户
# ceph auth get-key client.kube
## 新建用户 kube
# ceph auth get-or-create client.kube mon 'allow r' \
#    osd 'allow class-read object_prefix rbd_children, allow rwx pool=kube' \
#    -o ceph.client.kube.keyring
# 同步到 k8s 的所有节点上
# scp /etc/ceph/ceph.client.kube.keyring node1:/etc/ceph/
```

这个是做 Storage pvc

创建 ceph secret

```
mkdir -p /k8s/yaml/ceph-rbd
# 创建 secret 保存 ceph 的 key。 此处使用 admin 的 key
cat <<EOF >/k8s/yaml/ceph-rbd/ceph-secret.yml
apiVersion: v1
kind: Secret
```

```
metadata:
name: ceph-secret
type: "kubernetes.io/rbd"
data:
 # Please note this value is base64 encoded. echo "keystring"|base64
key: `ceph auth get-key client.admin | base64`
EOF
kubectl create -f /k8s/yaml/ceph-rbd/ceph-secret.yml
# kubectl delete -f /k8s/yaml/ceph-rbd/pv-pvc/ceph-secret.yml
#  查看创建
kubectl get secret ceph-secret -o yaml
```

创建 StorageClass

```
mkdir  -p /k8s/yaml/ceph-rbd/storageclass
#  创建 classstorage
cat <<EOF >/k8s/yaml/ceph-rbd/storageclass/ceph-storageclass.yaml
apiVersion:  storage.k8s.io/v1
kind:  StorageClass
metadata:
   name: ceph-rbd-storage
provisioner:  kubernetes.io/rbd
#  采用第三方 rbd 方式
# provisioner:  ceph.com/rbd
parameters:
monitors:  192.168.1.89:6789,192.168.1.88:6789,192.168.1.87:6789
adminId:  admin
adminSecretName: ceph-secret
adminSecretNamespace: default
pool:  rbd
userId: admin
userSecretName: ceph-secret
fsType: xfs
imageFeatures: layering
imageFormat:  "2"
EOF
# fsType
# imageFeatures

kubectl create -f /k8s/yaml/ceph-rbd/storageclass/ceph-storageclass.yaml
```

```
# 检查 storageclass
kubectl get sc/storageclass
[root@master pv-pvc]# kubectl get sc
NAME                 PROVISIONER           AGE
ceph-rbd-storage     kubernetes.io/rbd     6m7s
[root@master pv-pvc]# kubectl get storageclass
NAME                 PROVISIONER           AGE
ceph-rbd-storage     kubernetes.io/rbd     6m12s
```

创建 PersistentVolume(Claim)

```
# 创建 pvc
cat <<EOF >/k8s/yaml/ceph-rbd/storageclass/ceph-pvc.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: ceph-rbd-claim
spec:
accessModes:
   - ReadWriteOnce
storageClassName: ceph-rbd-storage
resources:
  requests:
    storage: 1Gi
EOF
kubectl create -f /k8s/yaml/ceph-rbd/storageclass/ceph-pvc.yaml
# 检查
kubectl get pvc
kubectl get pv
[root@master pv-pvc]# kubectl get pvc
NAME                 STATUS    VOLUME                                     CAPACITY
ACCESS MODES     STORAGECLASS          AGE
ceph-rbd-claim     Bound     pvc-fc6c7f7f-2086-11ea-b6f6-
525400dd00c8     1Gi          RWO               ceph-rbd-storage     75s
[root@master pv-pvc]# rbd ls
kubernetes-dynamic-pvc-9fed5d3c-2084-11ea-8705-525400dd00c8
[root@master pv-pvc]#
```

创建 pod 测试

```
# 创建 nginx pod 挂载测试
cat <<EOF >/k8s/yaml/ceph-rbd/storageclass/nginx-storageclass-pod.yaml
apiVersion: v1
kind: Pod
metadata:
name: nginx-storageclass
labels:
  name: nginx-storageclass
spec:
containers:
 - name: nginx-pod1
  image: nginx:alpine
  ports:
   - name: web
     containerPort: 80
  volumeMounts:
   - name: ceph-rdb
     mountPath: /usr/share/nginx/html
volumes:
 - name: ceph-rdb
  persistentVolumeClaim:
    claimName: ceph-rbd-claim
EOF
kubectl apply -f /k8s/yaml/ceph-rbd/storageclass/nginx-storageclass-pod.yaml
# 检查
[root@master pv-pvc]# kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP             NODE
NOMINATED NODE   READINESS GATES
nginx-
pod1   1/1     Running   0          85s   10.254.18.2   192.168.1.81   <none>
<none>
```

修改内容测

```
[root@master pv-pvc]# curl http://10.254.18.2
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
```

```
<hr><center>nginx/1.17.6</center>
</body>
</html>
[root@master pv-pvc]#

# 修改文件内容
kubectl exec -ti nginx-pod1  -- /bin/sh -c 'echo Hello World from Ceph RBD!!! > /usr/share/nginx/html/index.html'

# 访问测试
POD_ID=$(kubectl get pods -o wide | grep nginx-pod1 | awk '{print $6}')
curl http://$POD_ID

[root@master pv-pvc]# curl http://10.254.18.2
Hello World from Ceph RBD!!!
[root@master pv-pvc]#
```

FQA:

```
kubectl exec -ti nginx-pod1  -- /bin/sh -c 'echo Hello World from Ceph RBD!!! > /usr/share/nginx/html/index.html'   这句执行时报下面的错
Error from server (Forbidden): Forbidden (user=system:anonymous, verb=get, resource=nodes, subresource=proxy)
解决办法
kubectl create clusterrolebinding system:anonymous     --clusterrole=cluster-admin     --user=system:anonymous
```

检查 rdb 块内容,映射查看映射磁盘

```
rbd ls rbd
rbd --image `rbd ls rbd` info --name client.admin
rbd map --image `rbd ls rbd` --name client.admin
rbd showmapped --name client.admin
mkdir /mnt/ceph-rdb
mount /dev/rbd0 /mnt/ceph-rdb
cat /mnt/ceph-rdb/index.html

# 重新修改
kubectl exec -ti nginx-pod1  -- /bin/sh -c 'echo Hello World from Ceph RBD!!! Again > /usr/share/nginx/html/index.html'
[root@master pv-pvc]# curl http://10.254.18.2
Hello World from Ceph RBD!!! Again
```

```
# 重做 mount 的话
[root@master pv-pvc]# dmesg |tail
[12828.371063] rbd: rbd1: capacity 1073741824 features 0x1
[12869.148364] XFS (rbd1): Filesystem has duplicate UUID 9cc129a5-b879-4489-bb0d-270f80fe6a36 - can't mount
# 删除后 /dev/rbd0 的文件后可以重新 mount 并且内容为
```

ceph RBD 使用

创建 rbd 影像

```
# Create RBD Image
rbd create k8s-ceph-rbd --size 1G
rbd ls rbd
rbd info k8s-ceph-rbd
# map
rbd map k8s-ceph-rbd
# 如果失败的话，内核不支持的特性，重新 map
rbd feature disable k8s-ceph-rbd exclusive-lock, object-map, fast-diff, deep-flatten
# 检查 map 状况
rbd showmapped
# ummap 镜像，否则 K8S 创建 Pod 时会无法挂载 RBD 镜像。
rbd unmap k8s-ceph-rbd
```

这个是做 pv 与 pvc 的

创建 ceph secret[已存在不需要做]

```
mkdir -p /k8s/yaml/ceph-rbd
# 检查
kubectl get secret
# 创建 secret 保存 ceph 的 key。 此处使用 admin 的 key
cat <<EOF >/k8s/yaml/ceph-rbd/ceph-secret.yml
apiVersion: v1
kind: Secret
```

```
metadata:
name: ceph-secret
type: "kubernetes.io/rbd"
data:
 # Please note this value is base64 encoded. echo "keystring"|base64
key: `ceph auth get-key client.admin | base64`
EOF
kubectl create -f /k8s/yaml/ceph-rbd/ceph-secret.yml
# kubectl delete -f /k8s/yaml/ceph-rbd/jenkis/ceph-secret.yml
#  查看创建
kubectl get secret ceph-secret -o yaml
```

创建 pv

```
mkdir  -p /k8s/yaml/ceph-rbd/pv-pvc
cat <<EOF >/k8s/yaml/ceph-rbd/pv-pvc/ceph-rbd-pv.yml
apiVersion: v1
kind:  PersistentVolume
metadata:
name: ceph-rbd-pv
spec:
capacity:
  storage: 1Gi
accessModes:
   - ReadWriteOnce
rbd:
  monitors:
     - '192.168.1.89:6789'
     - '192.168.1.88:6789'
     - '192.168.1.87:6789'
  pool:  rbd
  image:  k8s-ceph-rbd
  user: admin
  secretRef:
    name: ceph-secret
  fsType: xfs
  readOnly: false
persistentVolumeReclaimPolicy:  Recycle
EOF
kubectl create -f /k8s/yaml/ceph-rbd/pv-pvc/ceph-rbd-pv.yml
kubectl get pv
```

创建 pvc

```
cat <<EOF >/k8s/yaml/ceph-rbd/pv-pvc/ceph-rbd-pvc.yml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ceph-rbd-pv-claim
spec:
accessModes:
   - ReadWriteOnce
resources:
  requests:
    storage: 1Gi
EOF
kubectl create -f /k8s/yaml/ceph-rbd/pv-pvc/ceph-rbd-pvc.yml
kubectl get pvc
```

创建 pod

```
cat <<EOF >/k8s/yaml/ceph-rbd/pv-pvc/ceph-rbd-pod.yml
apiVersion: v1
kind: Pod
metadata:
name: ceph-rbd-pvc-busybox1
spec:
containers:
   - name: ceph-rbd-pvc-busybox1
     image: busybox
     command: ["sleep", "60000"]
     volumeMounts:
      - name: cephrbd-vol1
        mountPath: "/mnt/k8s-ceph-rbd-vol"
        readOnly: false
volumes:
   - name: cephrbd-vol1
     persistentVolumeClaim:
       claimName: ceph-rbd-pv-claim
EOF
kubectl apply -f /k8s/yaml/ceph-rbd/pv-pvc/ceph-rbd-pod.yml
kubectl get pods -o wide
```

确认卷

```
# k8s master
kubectl exec -it ceph-rbd-pvc-busybox1  /bin/sh
ls /mnt/k8s-ceph-rbd-vol
cat <<EOF >/mnt/k8s-ceph-rbd-vol/hello.c
#include<stdio.h>
int main(){
printf("hello  world\n");
return 0;
}
EOF
ls /mnt/k8s-ceph-rbd-vol
###
# rbd client
rbd ls
rbd map k8s-ceph-rbd
mkdir  -p /mnt/ceph-rbd
mount `rbd showmapped |grep k8s-ceph-rbd|awk '{print $5}' ` /mnt/ceph-rbd
ls /mnt/ceph-rbd #  此处会有刚刚创建的文件
cat /mnt/ceph-rbd/hello.c


#  验证重新加载
kubectl delete -f /k8s/yaml/ceph-rbd/ceph-rbd-pod.yml
cat <<EOF >/mnt/ceph-rbd/hello.c
#include<iostream>
int main(){
std::count<<"hello  c++"<<std::endl;
return 0;
}
EOF
umount  /mnt/ceph-rbd
rbd unmap  k8s-ceph-rbd
kubectl apply -f /k8s/yaml/ceph-rbd/ceph-rbd-pod.yml
kubectl exec -it ceph-rbd-pvc-busybox1  /bin/sh
cat /mnt/k8s-ceph-rbd-vol/hello.c
```

ceph RBD 使用--Define in Volume

创建 rbd 镜像

```
rbd create k8s-ceph-rbd2 --size 1G --image-feature layering
rbd info k8s-ceph-rbd2
rbd ls
```

创建 pod 带 image 直接映射

```
cat <<EOF > /k8s/yaml/ceph-rbd/ceph-rbd-volume-pod.yml
apiVersion: v1
kind: Pod
metadata:
name: ceph-rbd-pvc-busybox2
spec:
containers:
  - image: busybox
    name: ceph-rbd-pvc-busybox2-rw
    command: ["sleep", "60000"]
    volumeMounts:
     - name: rbdpd
       mountPath: /mnt/rbd
volumes:
  - name: rbdpd
    rbd:
      monitors:
       - '192.168.1.89:6789'
       - '192.168.1.88:6789'
       - '192.168.1.87:6789'
      pool: rbd
      image: k8s-ceph-rbd2
      fsType: xfs #ext4
      readOnly: false
      user: admin
      secretRef:
        name: ceph-secret
EOF
kubectl apply -f /k8s/yaml/ceph-rbd/ceph-rbd-volume-pod.yml
```

确认卷

```
kubectl exec -it ceph-rbd-pvc-busybox2  /bin/sh
cat <<EOF >/mnt/rbd/hello.py
#!/usr/bin/python
printf("hello  world\n");
EOF
rbd map k8s-ceph-rbd2
mount `rbd showmapped |grep k8s-ceph-rbd2|awk '{print $5}' ` /mnt/ceph-rbd
cat /mnt/ceph-rbd/hello.py
umount  /mnt/ceph-rbd
rbd unmap k8s-ceph-rbd2
```

k8s 使用 CephFS

创建 file system

```
su - ceph-admin
#  添加 MDS，这里把三个节点作为均元数据服务器 MDS。
ceph-deploy mds create ceph00 ceph01 ceph02
#  可能需要同步密码
# ceph-deploy --overwrite-conf admin ceph00 ceph01 ceph02
netstat -tnlp | grep mds
```

创建 ceph secret

```
kubectl create secret generic cephfs-secret --type="kubernetes.io/cephfs" \
--from-literal=key=$(ceph  auth print-key client.admin)  \
--namespace=default
```

创建 pv

```
mkdir  -p /k8s/yaml/ceph-fs
cat <<EOF >/k8s/yaml/ceph-fs/ceph-fs-pv.yml
apiVersion: v1
kind:  PersistentVolume
```

```
metadata:
name: cephfs-pv
labels:
  pv: cephfs-pv
spec:
capacity:
  storage: 1Gi
accessModes:
  - ReadWriteMany
cephfs:
  monitors:
    - '192.168.1.89:6789'
    - '192.168.1.88:6789'
    - '192.168.1.87:6789'
  user: admin
  secretRef:
    name: cephfs-secret
  readOnly: false
persistentVolumeReclaimPolicy:  Delete
EOF
kubectl create -f /k8s/yaml/ceph-fs/ceph-fs-pv.yml
kubectl get pv
```

创建 **pvc**

```
cat <<EOF > /k8s/yaml/ceph-fs/ceph-fs-pvc.yml
kind:  PersistentVolumeClaim
apiVersion: v1
metadata:
name: cephfs-pvc
spec:
accessModes:
  - ReadWriteMany
resources:
  requests:
    storage: 1Gi
selector:
  matchLabels:
    pv: cephfs-pv
EOF
kubectl create -f /k8s/yaml/ceph-fs/ceph-fs-pvc.yml
kubectl get pvc
```

创建 pod 测试并在指定机器上启动

```
kubectl get nodes --show-labels
# kubectl label node 192.168.1.81 hostname=node1
# kubectl label node 192.168.1.82 hostname=node2

cat <<EOF >/k8s/yaml/ceph-fs/ceph-fs-pod1.yaml
apiVersion: v1
kind: Pod
metadata:
name: nginx-pod-fs1
labels:
  name: nginx-pod-fs1
spec:
 # 指定在 node1 上启动
nodeSelector:
  hostname: node1
containers:
 - name: nginx-pod1
  image: nginx:alpine
  ports:
   - name: nginx-pod-fs1
     containerPort: 80
  volumeMounts:
   - name: ceph-fs1
     mountPath: /usr/share/nginx/html
      # 不在根目录直接挂在，挂在到对应指定的子目录下
      # subPath: nginx-fs
volumes:
 - name: ceph-fs1
  persistentVolumeClaim:
     claimName: cephfs-pvc
EOF
kubectl apply -f /k8s/yaml/ceph-fs/ceph-fs-pod1.yaml
kubectl get pod -o wide
```

确认

```
kubectl exec -ti nginx-pod-fs1  -- /bin/sh  -c 'echo Hello World from Ceph RBD!!! > /usr/share/nginx/html/index.html'
```

跨节点测试

在另外的 node 上运行,与上方 pod 共用一个 pvc

```
cat <<EOF >/k8s/yaml/ceph-fs/ceph-fs-pod2.yaml
apiVersion: v1
kind: Pod
metadata:
name: nginx-pod-fs2
labels:
  name: nginx-pod-fs2
spec:
 # 指定在 node1 上启动
nodeSelector:
  hostname: node2
containers:
 - name: nginx-pod2
  image: nginx:alpine
  ports:
   - name: nginx-pod-fs2
     containerPort: 80
  volumeMounts:
   - name: ceph-fs2
     mountPath: /usr/share/nginx/html
volumes:
 - name: ceph-fs2
  persistentVolumeClaim:
     claimName: cephfs-pvc
EOF
kubectl apply -f /k8s/yaml/ceph-fs/ceph-fs-pod2.yaml
kubectl get pod -o wide
```

检查 cephfs

```
mkdir -p /mnt/cephfs
mount -t ceph 192.168.1.89:6789,192.168.1.88:6789,192.168.1.87:6789:/ \
/mnt/cephfs -o name=admin,secret=`ceph auth get-key client.admin`
umount /mnt/cephfs
```

Storage cephfs 使用

创建 PersistentVolume(Claim)

创建 pod 测试
测试

跨集群测试

https://blog.csdn.net/aixiaoyang168/article/details/79056864

https://blog.csdn.net/weixin_33757911/article/details/91820211

附录：

关于 PV 和 PVC

1.当 pv 的容量大于 pvc 的需求时,pvc 可以成功自动绑定 pv; 2.当 pv 的容量小于 pvc 的需求时,pvc 无法绑定该 pv; 3.pv 和 pvc 的绑定关系是一一对应的. 4.pv/pvc 的创建顺序是:pv -> pvc -> pod 5.pv/pvc 的销毁顺序是:pod -> pvc -> pv,顺序一定不要错

一般删除步骤为：先删 pod 再删 pvc 最后删 pv

但是遇到 pv 始终处于“Terminating”状态，而且 delete 不掉。

直接删除 k8s 中的记录：

```
kubectl patch pv xxx -p '{"metadata":{"finalizers":null}}'
```

容器上装 ping 命令

```
apt-get update

apt-get install  iputils-ping
```

附件： 私有仓库的一般操作

每个 CLIENT 节点需配： 才能拉取私有仓库 IMAGES:
[root@diyu93 ~]# more /etc/docker/daemon.json
{ "insecure-registries":["192.168.1.10:5000"] }
[root@diyu91 ~]# systemctl restart docker

查看私服镜像所有仓库

curl http://localhost:5000/v2/_catalog
[root@diyu90 tmp]# curl http://192.168.1.10:5000/v2/_catalog
{"repositories":["busybox","coredns","helloworld"]}

查看仓库中镜像的所有标签列表

curl http://localhost:5000/v2/java/tags/list
[root@diyu90 tmp]# curl http://192.168.1.10:5000/v2/coredns/tags/list
{"name":"coredns","tags":["latest"]}

删除仓库中的镜像

curl --header "Accept: application/vnd.docker.distribution.manifest.v2+json" -I -X HEAD http://localhost:5000/v2/java/manifests/my
[root@diyu90 tmp]# curl --header "Accept: application/vnd.docker.distribution.manifest.v2+json" -I -X HEAD http://192.168.1.10:5000/v2/coredns/manifests/latest
HTTP/1.1 200 OK
Content-Length: 739
Content-Type: application/vnd.docker.distribution.manifest.v2+json
Docker-Content-Digest:
sha256:695a5e109604331f843d2c435f488bf3f239a88aec49112d452c1cbf87e88405
Docker-Distribution-Api-Version: registry/2.0
Etag: "sha256:695a5e109604331f843d2c435f488bf3f239a88aec49112d452c1cbf87e88405"
X-Content-Type-Options: nosniff
Date: Thu, 28 May 2020 08:19:37 GMT

[root@diyu90 tmp]# curl -X DELETE 192.168.1.10:5000/v2/coredns/manifests/sha256:695a5e109604331f843d2c435f488bf3f239a88aec49112d452c1cbf87e88405
{"errors":[{"code":"UNSUPPORTED","message":"The operation is unsupported."}]}

docker run -d -p 5000:5000 -e REGISTRY_STORAGE_DELETE_ENABLED=true --name registry --restart=always --privileged=true -v /docker/registry:/var/lib/registry registry:2

[root@diyu90 tmp]# curl -X DELETE

```
192.168.1.10:5000/v2/coredns/manifests/sha256:695a5e109604331f843d2c435f488bf3f239a
88aec49112d452c1cbf87e88405
[root@diyu90 tmp]# curl http://192.168.1.10:5000/v2/_catalog
{"repositories":["busybox","coredns","helloworld"]}
[root@diyu90 tmp]# curl --header "Accept: application/vnd.docker.distribution.manifest.v2+json" -I -X
HEAD  http://192.168.1.10:5000/v2/coredns/manifests/latest
HTTP/1.1 404 Not Found
Content-Type: application/json; charset=utf-8
Docker-Distribution-Api-Version: registry/2.0
X-Content-Type-Options: nosniff
Date: Thu, 28 May 2020 08:31:14 GMT
Content-Length: 96

[root@diyu90 tmp]# curl http://192.168.1.10:5000/v2/coredns/tags/list
{"name":"coredns","tags":null}
[root@diyu90 tmp]#

[root@diyu93 ~]# more /etc/docker/daemon.json
{ "insecure-registries":["192.168.1.10:5000"] }
```

附录：获取本地仓库 IMAGE

获取本地 image,每个节点均操作:

```
192.168.1.10:5000/kube-proxy                    v1.18.1          4e68534e24f6
7 weeks ago         117MB
k8s.gcr.io/kube-proxy                           v1.18.1          4e68534e24f6
7 weeks ago         117MB
192.168.1.10:5000/kube-apiserer                 v1.18.1          a595af0107f9
7 weeks ago         173MB
k8s.gcr.io/kube-apiserver                       v1.18.1          a595af0107f9
7 weeks ago         173MB
192.168.1.10:5000/kube-controller-manager       v1.18.1          d1ccdd18e6ed
7 weeks ago         162MB
k8s.gcr.io/kube-controller-manager              v1.18.1          d1ccdd18e6ed
7 weeks ago         162MB
192.168.1.10:5000/kube-scheduler                v1.18.1          6c9320041a7b
7 weeks ago         95.3MB
k8s.gcr.io/kube-scheduler                       v1.18.1          6c9320041a7b
7 weeks ago         95.3MB
192.168.1.10:5000/pause                         3.2              80d28bedfe5d
3 months ago        683kB
k8s.gcr.io/pause                                3.2              80d28bedfe5d
```

3 months ago 683kB
192.168.1.10:5000/coredns 1.6.7 67da37a9a360
4 months ago 43.8MB
k8s.gcr.io/coredns 1.6.7 67da37a9a360
4 months ago 43.8MB
192.168.1.10:5000/etcd 3.4.3-0 303ce5db0e90
7 months ago 288MB
k8s.gcr.io/etcd 3.4.3-0 303ce5db0e90
7 months ago 288MB

docker pull 192.168.1.10:5000/kube-proxy:v1.18.1
docker tag 192.168.1.10:5000/kube-proxy:v1.18.1 k8s.gcr.io/kube-proxy:v1.18.1

docker pull 192.168.1.10:5000/kube-apiserver:v1.18.1
docker tag 192.168.1.10:5000/kube-apiserver:v1.18.1 k8s.gcr.io/kube-apiserver:v1.18.1

docker pull 192.168.1.10:5000/kube-controller-manager:v1.18.1
docker tag 192.168.1.10:5000/kube-controller-manager:v1.18.1 k8s.gcr.io/kube-controller-manager:v1.18.1

docker pull 192.168.1.10:5000/kube-scheduler:v1.18.1
docker tag 192.168.1.10:5000/kube-scheduler:v1.18.1 k8s.gcr.io/kube-scheduler:v1.18.1

docker pull 192.168.1.10:5000/pause:3.2
docker tag 192.168.1.10:5000/pause:3.2 k8s.gcr.io/pause:3.2

docker pull 192.168.1.10:5000/coredns:1.6.7
docker tag 192.168.1.10:5000/coredns:1.6.7 k8s.gcr.io/coredns:1.6.7

docker pull 192.168.1.10:5000/etcd:3.4.3-0
docker tag 192.168.1.10:5000/etcd:3.4.3-0 k8s.gcr.io/etcd:3.4.3-0

kubeadm init \
--apiserver-advertise-address=192.168.1.90 \
--kubernetes-version v1.18.1 \
--service-cidr=10.1.0.0/16 \
--pod-network-cidr=10.254.0.10/16

附录：节点有问题时的解决办法

如从节点加入有问题，在从节点执行如下的命令（以下无需执行）

```
[root@ken2 ~]# kubeadm reset
```
再次加入即可。

补充：移除 NODE 节点的方法

第一步：先将节点设置为维护模式(host1 是节点名称)
[root@ken ~]# kubectl drain host1 --delete-local-data --force --ignore-daemonsets
node/host1 cordoned
WARNING: Ignoring DaemonSet-managed pods: kube-flannel-ds-amd64-ssqcl, kube-proxy-7cnsr
node/host1 drained

第二步：然后删除节点
[root@ken ~]# kubectl delete node host1
node "host1" deleted

第三步：查看节点
发现 host1 节点已经被删除了
[root@ken ~]# kubectl get nodes
NAME STATUS ROLES AGE VERSION
host2 Ready <none> 13m v1.13.2
ken Ready master 49m v1.13.2

如果这个时候再想添加进来这个 node，需要执行两步操作
第一步：停掉 kubelet(需要添加进来的节点操作)
[root@host1 ~]# systemctl stop kubelet

第二步：删除相关文件
[root@host1 ~]# rm -rf /etc/kubernetes/*

第三步：添加节点
[root@host1 ~]# kubeadm join 172.20.10.2:6443 --token rn816q.zj0crlasganmrzsr --discovery-token-ca-cert-hash
sha256:e339e4dbf6bd1323c13e794760fff3cbeb7a3f6f42b71d4cb3cffdde72179903

第四步：查看节点
[root@ken ~]# kubectl get nodes
NAME STATUS ROLES AGE VERSION
host1 Ready <none> 13s v1.13.2
host2 Ready <none> 17m v1.13.2
ken Ready master 53m v1.13.2

忘掉 token 再次添加进 k8s 集群

第一步：主节点执行命令
获取 token
[root@ken-master ~]# kubeadm token list
TOKEN TTL EXPIRES USAGES DESCRIPTION EXTRA GROUPS
ojxdod.fb7tqipat46yp8ti 10h 2019-05-06T04:55:42+08:00 authentication,signing The default bootstrap token generated by 'kubeadm init'. system:bootstrappers:kubeadm:default-node-token

第二步： 获取 ca 证书 sha256 编码 hash 值
[root@ken-master ~]# openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | openssl dgst -sha256 -hex | sed 's/^.* //'
2f8888cdb01191ff6dbca0edb02dbb21a14469028e4ff2598854a4544c5fa751

第三步：从节点执行如下的命令
[root@ken-node1 ~]# systemctl stop kubelet

第四步：删除相关文件
[root@ken-node1 ~]# rm -rf /etc/kubernetes/*

第五步：加入集群
指定主节点 IP，端口是 6443
在生成的证书前有 sha256:
[root@ken-node1 ~]# kubeadm join 192.168.64.10:6443 --token ojxdod.fb7tqipat46yp8ti --discovery-token-ca-cert-hash sha256:2f8888cdb01191ff6dbca0edb02dbb21a14469028e4ff2598854a4544c5fa751

检查 master 节点上运行的服务
1.api-server
2.schedule
3.控制器管理器
4.etcd
5.pod 网络–fannel

node 节点上运行的服务
1.kubelet 是 k8s 集群当中唯一一个不是以容器运行的客户端
2.kube-proxy
3.pod

运行 pod 有两种方式
1.通过 kubectl 命令行工具进行创建
2.通过 yml 文件

[root@zxw9 ~]# kubectl get cs

NAME STATUS MESSAGE ERROR
scheduler Healthy ok
controller-manager Healthy ok
etcd-0 Healthy {"health":"true"}

查看所有的 namespace
[root@ken1 ~]# kubectl get ns
NAME STATUS AGE
default Active 64m
kube-node-lease Active 64m
kube-public Active 64m
kube-system Active 64m

查看某个 namespace 运行了那些 pod
[root@ken1 ~]# kubectl get po -n kube-system
NAME READY STATUS RESTARTS AGE
coredns-bccdc95cf-ggx7z 1/1 Running 0 63m
coredns-bccdc95cf-xxgjp 1/1 Running 0 63m
etcd-ken1 1/1 Running 0 62m
kube-apiserver-ken1 1/1 Running 0 62m
kube-controller-manager-ken1 1/1 Running 0 62m
kube-flannel-ds-amd64-6qmv4 1/1 Running 0 16m
kube-flannel-ds-amd64-p8rkl 1/1 Running 0 38m
kube-flannel-ds-amd64-vnt8c 1/1 Running 0 35m
kube-proxy-4gqq7 1/1 Running 0 35m
kube-proxy-5n2f4 1/1 Running 0 63m
kube-proxy-fr262 1/1 Running 0 16m
kube-scheduler-ken1 1/1 Running 0 62m

查看更加完整信息
[root@ken1 ~]# kubectl get po -n kube-system -o wide