基于 Kubernetes v1.25.0 和 Docker 部署高可用集群

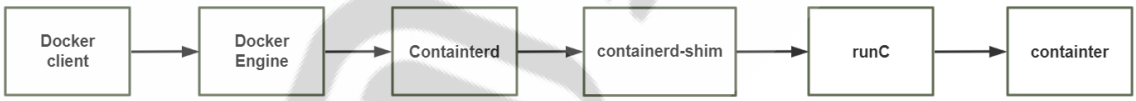
主要内容

- Kubernetes 集群架构组成
- 容器运行时 CRI
- Kubernetes v1.25 新特性
- Kubernetes v1.24 之后不再支持 Docker 的解决方案
- Kubernetes v1.25 高可用集群架构
- 基于 Kubernetes v1.25.0 和 Docker 部署高可用集群实战案例

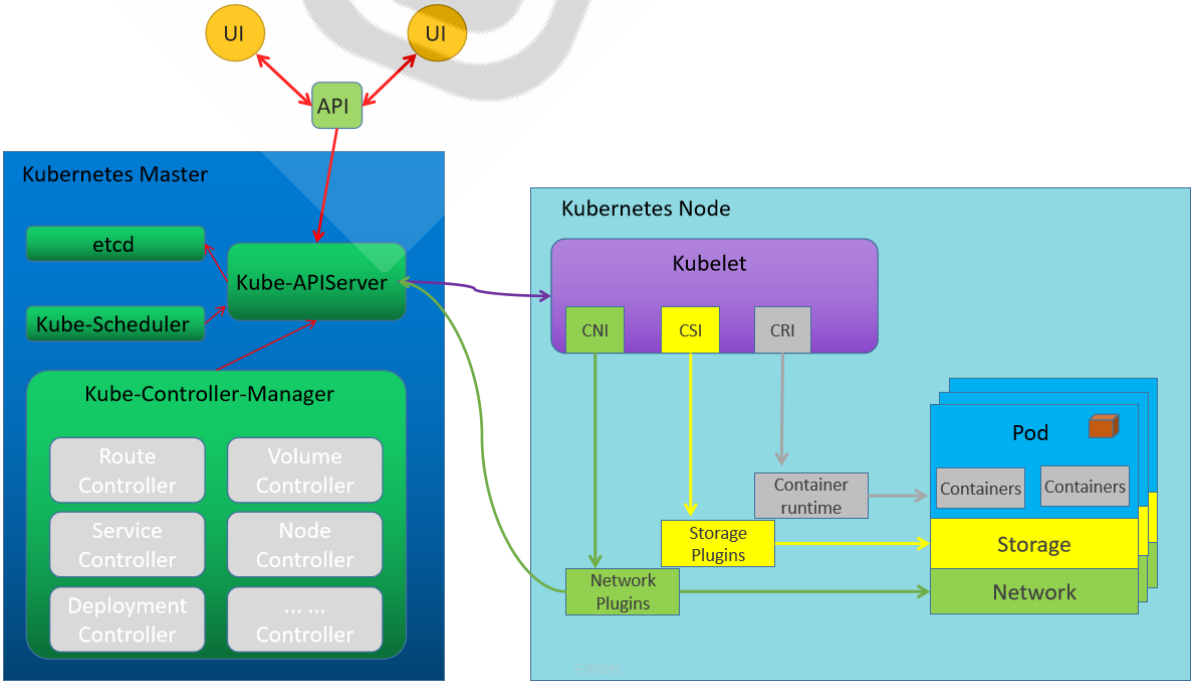
1 Kubernetes v1.25 新特性

1.1 Kubernetes 组件

Docker 的运行机制



Kubernetes的设计初衷是支持可插拔架构，从而利于扩展kubernetes的功能。



Kubernetes提供了三个特定功能的接口,kubernetes通过调用这几个接口，来完成相应的功能。

- 容器运行时接口CRI: Container Runtime Interface

  kubernetes 对于容器的解决方案，只是预留了容器接口，只要符合CRI标准的解决方案都可以使用
- 容器网络接口CNI: Container Network Interface

  kubernetes 对于网络的解决方案，只是预留了网络接口，只要符合CNI标准的解决方案都可以使用
- 容器存储接口CSI: Container Storage Interface

  kubernetes 对于存储的解决方案，只是预留了存储接口，只要符合CSI标准的解决方案都可以使用

  此接口非必须

容器运行时接口（CRI）

CRI是kubernetes定义的一组gRPC服务。Kubelet作为客户端，基于gRPC协议通过Socket和容器运行时通信。

CRI 是一个插件接口，它使 kubelet 能够使用各种容器运行时，无需重新编译集群组件。

Kubernetes 集群中需要在每个节点上都有一个可以正常工作的容器运行时， 这样 kubelet 能启动 Pod 及其容器。

容器运行时接口（CRI）是 kubelet 和容器运行时之间通信的主要协议。

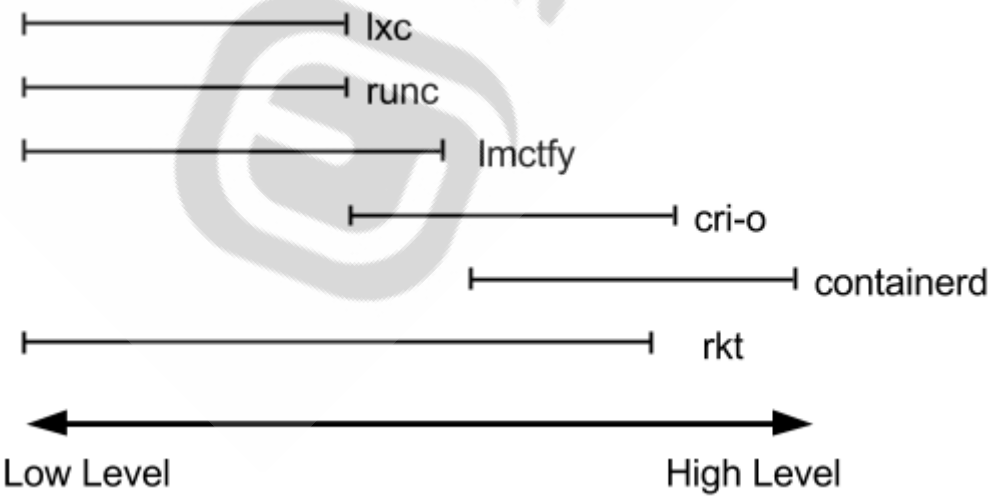
CRI 括两类服务：镜像服务（Image Service）和运行时服务（Runtime Service）。

镜像服务提供下载、检查和删除镜像的远程程序调用。

运行时服务包含用于管理容器生命周期，以及与容器交互的调用的远程程序调用。

OCI（Open Container Initiative，开放容器计划）定义了创建容器的格式和运行时的开源行业标准，包括镜像规范（Image Specification）和运行时规范(Runtime Specification)。

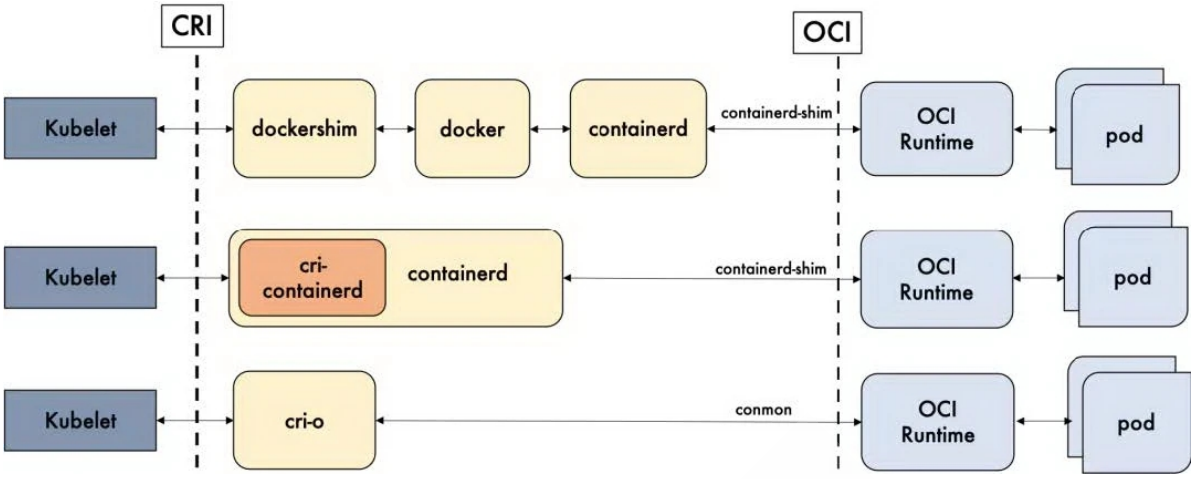
对于容器运行时主要有两个级别：Low Level(使用接近内核层) 和 High Level(使用接近用户层)目前，市面上常用的容器引擎有很多，主要有下图的那几种。



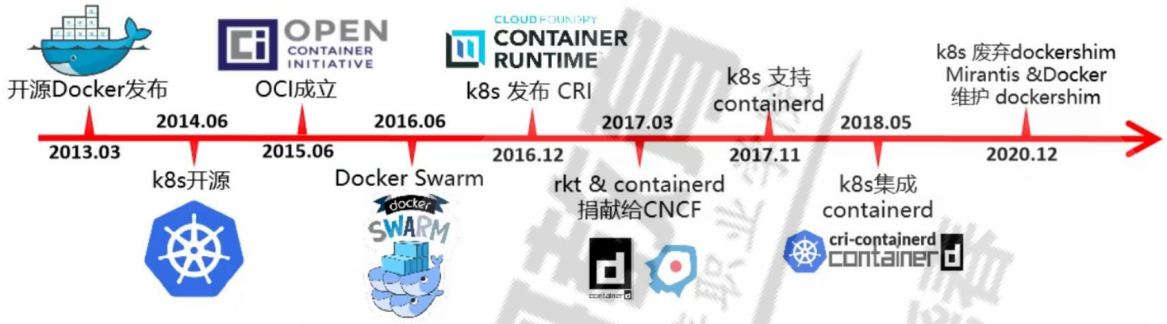
dockershim, containerd 和cri-o都是遵循CRI的容器运行时，我们称他们为高层级运行时（High-level Runtime）

其他的容器运营厂商最底层的runc仍然是Docker在维护的。

Google,CoreOS,RedHat都推出自己的运行时:lmctfy,rkt,cri-o,但到目前Docker仍然是最主流的容器引擎技术。



1.2 Kubernetes v1.24 之后不再支持 Docker？



2014年 Docker & Kubernetes 蜜月期

2015~2016年 Kubernetes & RKT vs Docker, 最终Docker 胜出

2016年 Kubernetes逐渐赢得任务编排的胜利

2017年 rkt 和 containerd 捐献给 CNCF

2020年 kubernetes宣布废弃dockershim，但 Mirantis 和 Docker 宣布维护 dockershim

2022年5月3日，Kubernetes v1.24正式发布，此版本提供了很多重要功能。该版本涉及46项增强功能：其中14项已升级为稳定版，15项进入beta阶段，13项则刚刚进入alpha阶段。此外，另有2项功能被弃用、2项功能被删除。v1.24 之前的 Kubernetes 版本包括与 Docker Engine 的直接集成，使用名为 **dockershim** 的组件。 值得注意的是v1.24 的 Kubernetes 正式移除对Dockershim的支持，即默认不再支持 docker

2022年8月24日,Kubernetes v1.25 正式发布

官方说明

```
https://kubernetes.io/zh-cn/docs/setup/production-environment/container-runtimes/
```

移除 Dockershim 的说明

https://kubernetes.io/zh-cn/blog/2022/02/17/dockershim-faq/

Dockershim的历史背景

https://mp.weixin.qq.com/s/elkfBVzN8-zC30111zFpMw

Kubernetes CNCF 2022-05-03 10:40 发表于香港
作者：Kat Cosgrove

从 Kubernetes v1.24 开始，Dockershim 会给移除，这对于项目来说是个积极的举措。然而，无论是在社会上，还是在软件开发中，上下文对于完全理解某些东西都是很重要的，这值得更深入的研究。在 Kubernetes v1.24 中移除 dockershim 的同时，我们在社区中看到了一些困惑（有时达到恐慌的程度），和对这一决定的不满，很大程度上是由于缺乏关于这移除的上下文。弃用并最终将 dockershim 从 Kubernetes 移除的决定，并不是迅速或轻率做出的。尽管如此，它已经操作了很长时间，以至于今天的许多用户都比这个决定更新，当然也比导致 dockershim 首先成为必要的选择更新。

那么，dockershim 是什么，为什么它会消失？

在 Kubernetes 的早期，我们只支持一个容器运行时。那个运行时是 Docker Engine。当时，没有太多其他选择，Docker 是处理容器的主要工具，所以这不是个有争议的选择。最终，我们开始添加更多的容器运行时，比如 rkt 和 hypernetes，很明显 Kubernetes 用户希望选择最适合他们的运行时。因此 Kubernetes 需要种方法，来允许集群操作者灵活地使用他们选择的任何运行时。

发布CRI[1]（Container Runtime Interface，容器运行时接口）就是为了提供这种灵活性。CRI 的引入对项目和用户来说都很棒，但它也引入了一个问题：Docker Engine 作为容器运行时的使用早于 CRI，Docker Engine 与 CRI 不兼容。为了解决这个问题，引入了一个小软件垫片（"shim"，dockershim）作为 kubelet 组件的一部分，专门用于填补 Docker Engine 和 CRI 之间的空白，允许集群运营商继续使用 Docker Engine 作为他们的容器运行时，基本上不会给中断。

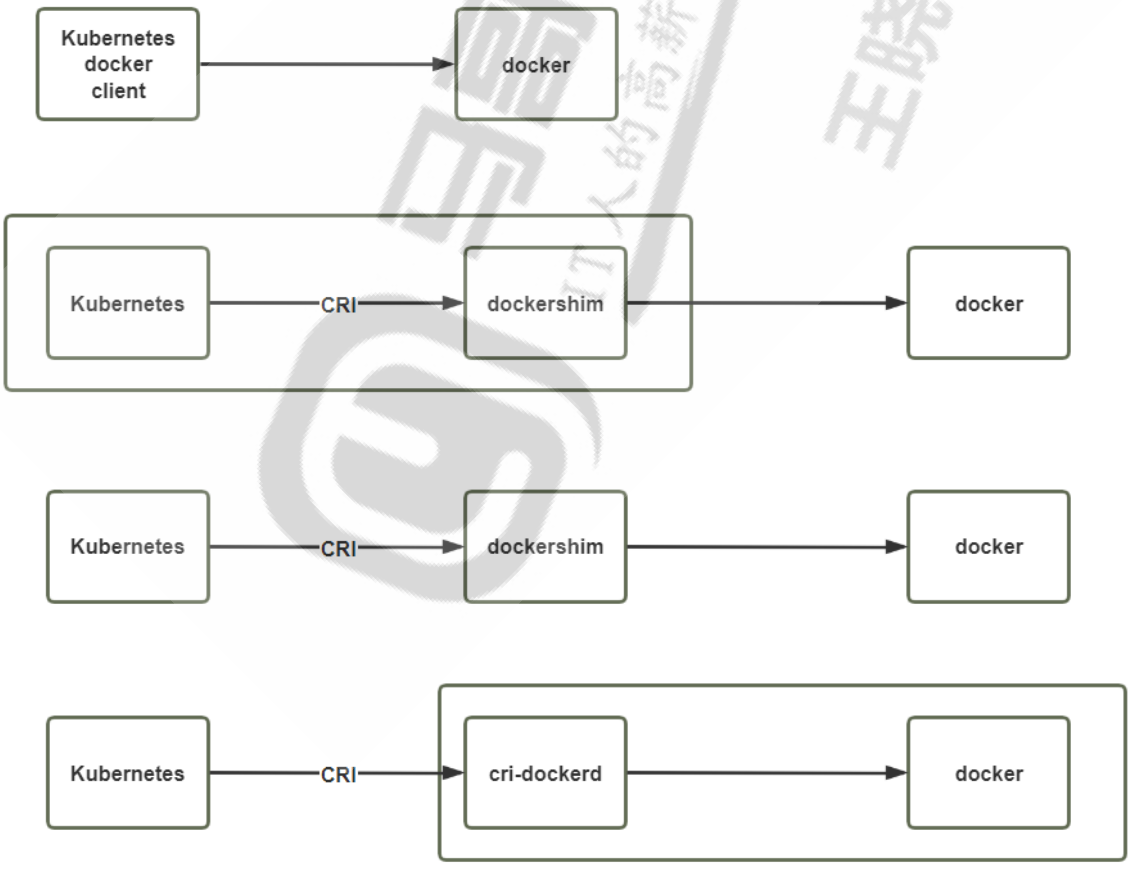
然而，这个小小的软件垫片从来就不是永久的解决方案。多年来，它的存在给 kubelet 本身带来了许多不必要的复杂性。由于这个垫片，Docker 的一些集成实现不一致，导致维护人员的负担增加，并且维护特定于供应商的代码不符合我们的开源理念。为了减少这种维护负担，并向一个支持开放标准的更具协作性的社区发展，KEP-2221 获引入[2]，它建议去掉 dockershim。随着 Kubernetes v1.20 的发布，这一弃用成为正式。

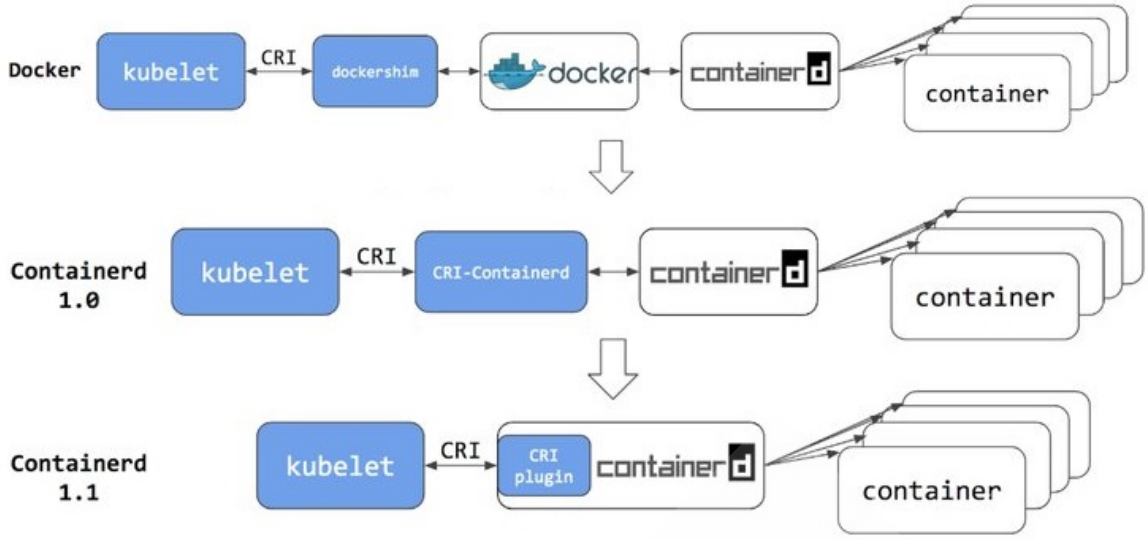
我们没有很好地传达这一点，不幸的是，弃用声明导致了社区内的一些恐慌。对于 Docker 作为一家公司来说这意味着什么，由 Docker 构建的容器镜像能否运行，以及 Docker Engine 实际是什么导致了社交媒体上的一场大火。这是我们的过失；我们应该更清楚地沟通当时发生了什么以及原因。为了解决这个问题，我们发布了一个博客[3]和相关的常见问题[4]，以减轻社区的恐惧，并纠正一些关于 Docker 是什么，以及容器如何在 Kubernetes 中工作的误解。由于社区的关注，Docker 和 Mirantis 共同同意以cri-docker[5]的形式继续支持 dockershim 代码，允许你在需要时继续使用 Docker Engine 作为你的容器运行时。为了让那些想尝试其他运行时（如 containerd 或 cri-o）的用户感兴趣，编写了迁移文档[6]。

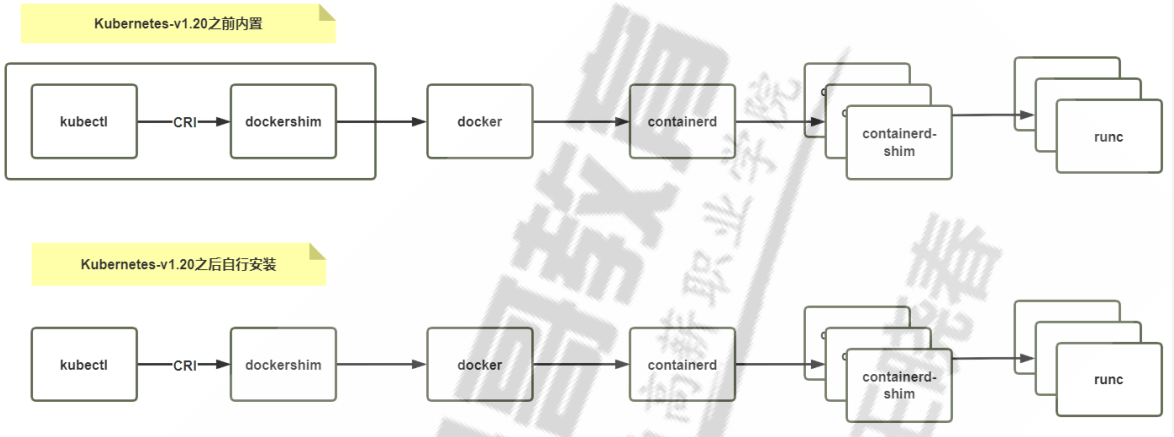
我们后来对社区进行了调查[7]，发现仍然有许多用户有问题和顾虑[8]。作为回应，Kubernetes 维护者和 CNCF 致力于通过扩展文档和其他程序来解决这些问题。事实上，这篇博文就是这个计划的一部分。随着如此多的最终用户成功地迁移到其他运行时，以及文档的改进，我们相信现在每个人都有了迁移的道路。

无论是作为工具还是作为公司，Docker 都不会消失。它是云原生社区和 Kubernetes 项目历史的重要组成部分。没有他们我们不会有今天。也就是说，从 kubelet 中移除 dockershim 最终对社区、生态系统、项目和整个开源都有好处。这是我们所有人一起支持开放标准的机会，我们很高兴在 Docker 和社区的帮助下这样做。

Kubernetes 调用 runtime 的变化







1.3 Kubernetes v1.25 新变化

2022年8月24日,Kubernetes v1.25 正式发布

Kubernetes 1.25主题是Combiner，即组合器。

Kubernetes 1.25中包含多达40项增强功能

```
https://mp.weixin.qq.com/s/PKoNkhPU6OhjuuPzELP-Rg
```

PodSecurityPolicy被移除；Pod Security Admission毕业为稳定版

PodSecurityPolicy是在1.21版本中被决定弃用的，到1.25版本则将被正式删除。之所以删除此项功能，是因为想要进一步提升其可用性，就必须引入重大变更。为了保持项目整体稳定，只得加以弃用。取而代之的正是在1.25版本中毕业至稳定版的Pod Security Admission。如果你当前仍依赖PodSecurityPolicy，请按照Pod Security Admission迁移说明[1]进行操作。

临时容器迎来稳定版

临时容器是指在Pod中仅存在有限时长的容器。当我们需要检查另一容器，但又不能使用kubectl exec时（例如在执行故障排查时），往往可以用临时容器替代已经崩溃、或者镜像缺少调试工具的容器。临时容器在Kubernetes 1.23版本中已经升级至Beta版，这一次则进一步升级为稳定版。

对cgroups v2的稳定支持

自Linux内核cgroups v2 API公布稳定版至今，已经过去两年多时间。如今，已经有不少发行版默认使用此API，Kubernetes自然需要支持该内核才能顺利对接这些发行版。Cgroups v2对cgroups v1做出了多项改进，更多细节请参见cgroups v2说明文档[2]。虽然cgroups v1将继续受到支持，但我们后续将逐步弃用v1并全面替换为v2。

更好的Windows系统支持

- 性能仪表板添加了对Windows系统的支持
- 单元测试增加了对Windows系统的支持
- 一致性测试增加了对Windows系统的支持
- 为Windows Operational Readiness创建了新的GitHub仓库

将容器注册服务从k8s.gcr.io移动至registry.k8s.io

1.25版本已经合并将容器注册服务从k8s.gcr.io移动至registry.k8s.io的变更。关于更多细节信息，请参阅相应wiki页面[3]，我们也通过Kubernetes开发邮件清单发出了全面通报。

SeccompDefault升级为Beta版

网络策略中的endPort已升级为稳定版

网络策略中的endPort已经迎来GA通用版。支持endPort字段的网络策略提供程序，现可使用该字段来指定端口范围以应用网络策略。在之前的版本中，每个网络策略只能指向单一端口。

请注意，endPort的起效前提是必须得到网络策略提供程序的支持。如果提供程序不支持endPort，而您又在网络策略中指定了此字段，则会创建出仅覆盖端口字段（单端口）的网络策略。

本地临时存储容量隔离迎来稳定版

本地临时存储容量隔离功能已经迎来GA通用版。这项功能最早于1.8版本中公布了alpha版，在1.10中升级至beta，如今终于成为稳定功能。它通解为各Pod之间的本地临时存储提供容量隔离支持，例如EmptyDir。因此如果Pod对本地临时存储容量的消耗超过了该上限，则会驱逐该Pod以限制其对共享资源的占用。

核心CSI迁移迎来稳定版

CSI迁移是SIG Storage在之前多个版本中做出的持续努力，目标是将树内存储卷插件移动到树外CSI驱动程序，并最终移除树内存储卷插件。此次核心CSI迁移已迎来GA通用版，GCE PD和AWS EBS的CSI迁移功能也同步达到GA阶段。vSphere的CSI迁移仍处于beta阶段（但也已经默认启用），Portworx的CSI迁移功能同样处于beta阶段（默认关闭）。

CSI临时存储卷提升至稳定版

CSI临时存储卷功能，允许用户在临时用例的pod规范中直接指定CSI存储卷。如此一来，即可使用已安装的存储卷直接在pod内注入任意状态，例如配置、机密、身份、变量或其他类似信息。这项功能最初于1.15版本中推出alpha版，现已升级为GA通用版。某些CSI驱动程序会使用此功能，例如负责存储秘密信息的CSI驱动程序。

CRD验证表达式语言升级至Beta版

CRD验证表达式语言现已升级为beta版，因此声明能够使用通用表达式语言（CEL）验证自定义资源。

服务器端未知字段验证升级为Beta版

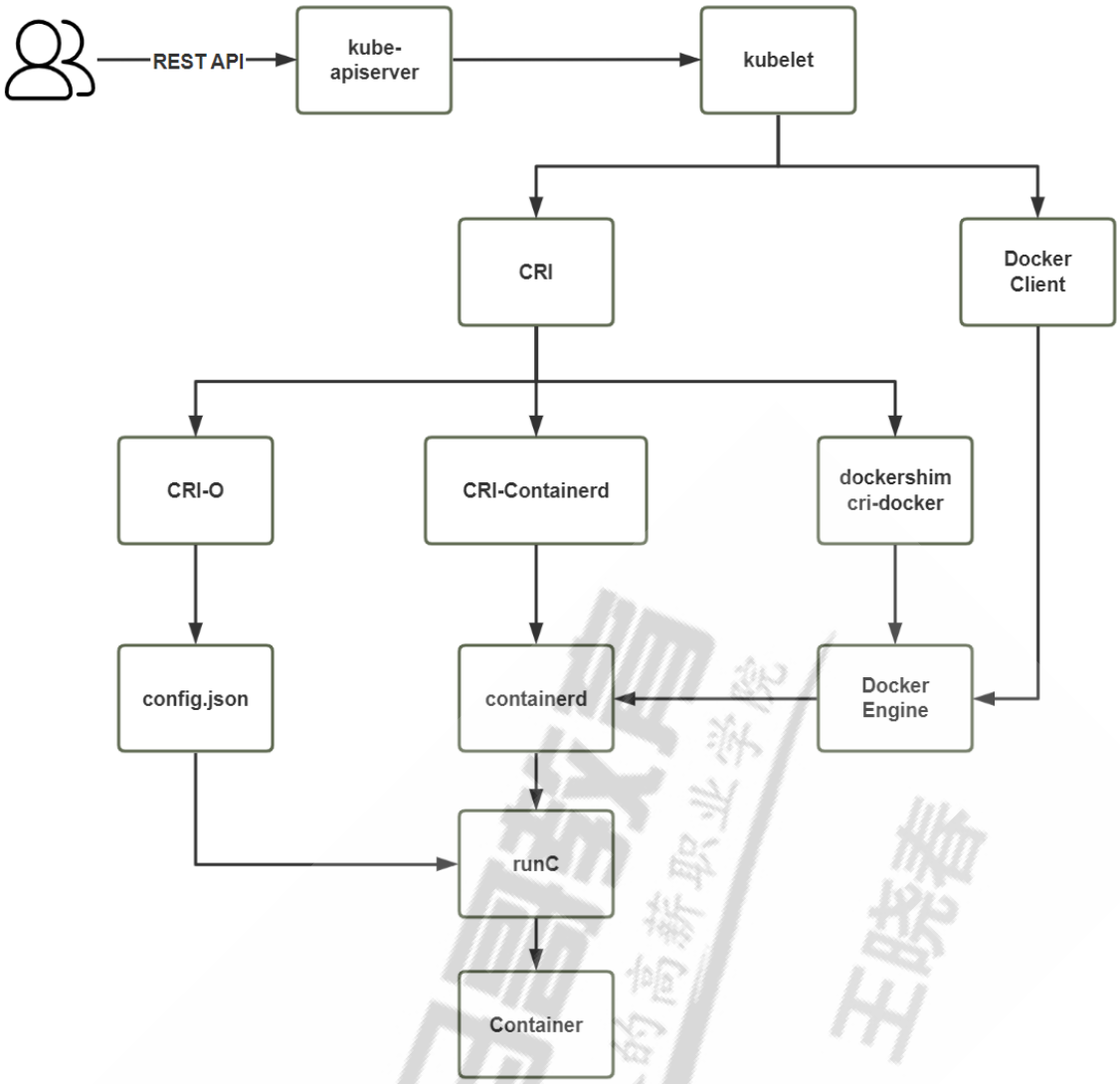
ServerSideFieldValidation功能门现已升级为Beta版（默认启用），允许用户在检测到未知字段时，有选择地触发API服务器上的模式验证机制。如此一来，即可从kubectl中删除客户端验证，同时继续保持对包含未知/无效字段的请求报错。

引入KMS v2 API

引入KMS v2 alpha1 API以提升性能，实现轮替与可观察性改进。此API使用AES-GCM替代了AES-CBC，通过DEK实现静态数据（即Kubernetes Secrets）加密。过程中无需额外用户操作，而且仍然支持通过AES-GCM和AES-CBC进行读取。

1.4 Kubernetes v1.25 集群创建方案

Kubernetes v1.25 集群创建



- 方式1: Containerd

  默认情况下,Kubernetes在创建集群的时候,使用的就是Containerd 方式。

- 方式2: Docker

  Docker使用的普及率较高,虽然Kubernetes-v1.24 默认情况下废弃了kubelet对于Docker的支持,但是我们还可以借助于Mirantis维护的cri-dockerd插件方式来实现Kubernetes集群的创建。

  Docker Engine 没有实现 CRI， 而这是容器运行时在 Kubernetes 中工作所需要的。 为此，必须安装一个额外的服务 cri-dockerd。 cri-dockerd 是一个基于传统的内置 Docker 引擎支持的项目，它在 1.24 版本从 kubelet 中移除

  项目站点: https://github.com/Mirantis/cri-dockerd

- 方式3: CRI-O

  CRI-O的方式是Kubernetes创建容器最直接的一种方式,在创建集群的时候,需要借助于cri-o插件的方式来实现Kubernetes集群的创建。

2 Kubernetes 高可用集群部署架构

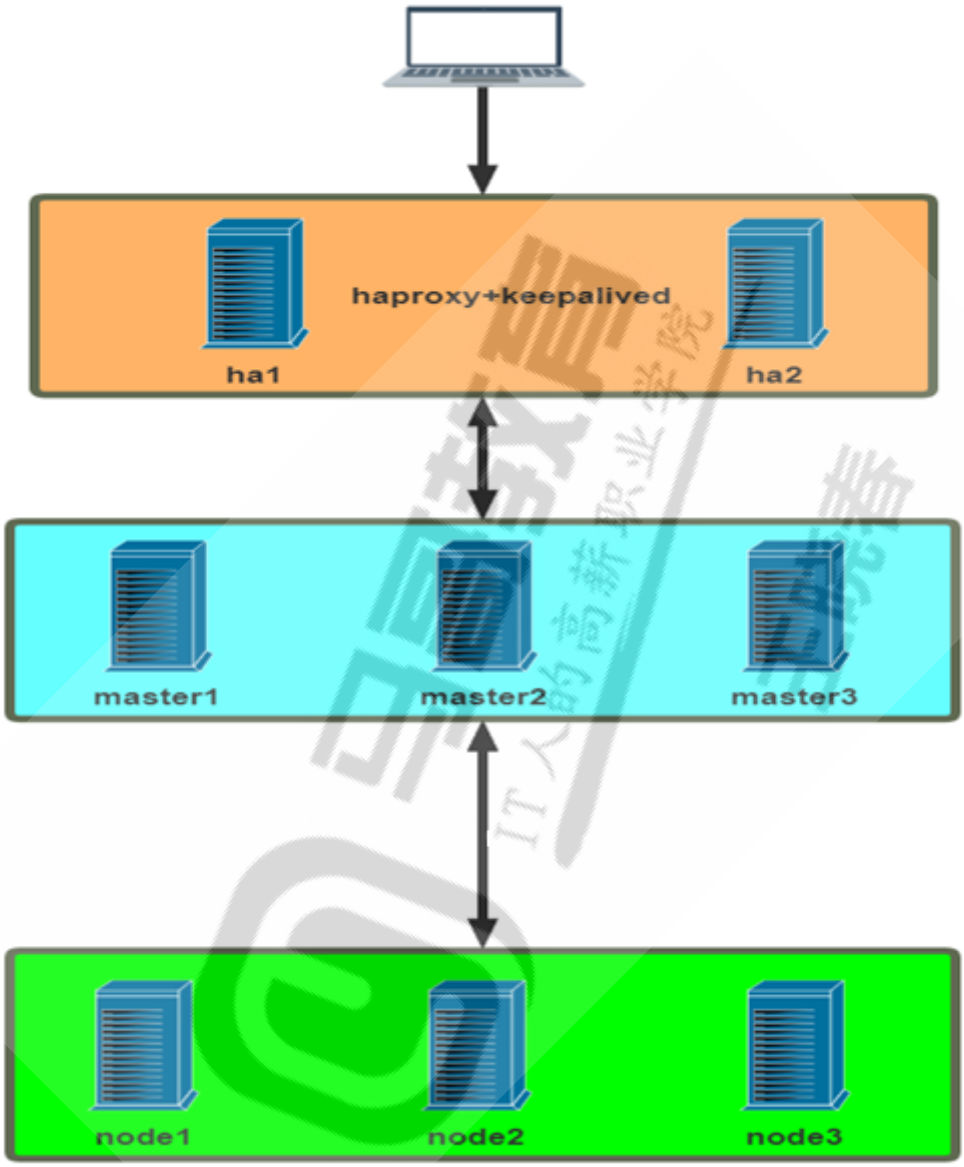
本示例中的Kubernetes集群部署将基于以下环境进行。

- 每个主机2G内存以上,2核CPU以上
- OS: Ubuntu 20.04.4
- Kubernetes：v1.25,0

- Container Runtime：Docker CE 20.10.17
- CRI：cri-dockerd v0.2.5

网络环境：

```
节点网络：10.0.0.0.0/24
Pod网络：10.244.0.0/16
Service网络：10.96.0.0/12
```



IP	主机名	角色
10.0.0.101	master1.wang.org	K8s 集群主节点 1，Master和etcd
10.0.0.102	master2.wang.org	K8s 集群主节点 2，Master和etcd
10.0.0.103	master3.wang.org	K8s 集群主节点 3，Master和etcd
10.0.0.104	node1.wang.org	K8s 集群工作节点 1
10.0.0.105	node2.wang.org	K8s 集群工作节点 2
10.0.0.106	node3.wang.org	K8s 集群工作节点 3
10.0.0.107	ha1.wang.org	K8s 主节点访问入口 1,提供高可用及负载均衡
10.0.0.108	ha2.wang.org	K8s 主节点访问入口 2,提供高可用及负载均衡
10.0.0.109	harbor.wang.org	容器镜像仓库
10.0.0.100	k8s.wang.org	VIP，在ha1和ha2主机实现

3 基于Kubeadm 实现 Kubernetes v1.25.0 集群部署流程说明

官方说明

```
https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/
https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/
https://kubernetes.io/zh-cn/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/
```

使用 `kubeadm`，能创建一个符合最佳实践的最小化 Kubernetes 集群。 事实上，你可以使用 `kubeadm` 配置一个通过 Kubernetes 一致性测试的集群。 `kubeadm` 还支持其他集群生命周期功能， 例如启动引导令牌和集群升级。

- Kubernetes集群API访问入口的高可用
- 每个节点主机的初始环境准备
- 在所有Master和Node节点都安装容器运行时,实际Kubernetes只使用其中的Containerd
- 在所有Master和Node节点安装kubeadm 、kubelet、kubectl
- 在所有节点安装和配置 cri-dockerd
- 在第一个 master 节点运行 kubeadm init 初始化命令 ,并验证 master 节点状态
- 在第一个 master 节点安装配置网络插件
- 在其它master节点运行kubeadm join 命令加入到控制平面集群中
- 在所有 node 节点使用 kubeadm join 命令加入集群
- 创建 pod 并启动容器测试访问， 并测试网络通信

4 基于Kubeadm 部署 Kubernetes v1.25.0 高可用集群案例

4.1 部署 Kubernetes 集群 API 访问入口的高可用

在10.0.0.107和10.0.0.108上实现如下操作

4.1.1 安装 HAProxy

利用 HAProxy 实现 Kubeapi 服务的负载均衡

```
#修改内核参数
[root@ha1 ~]#cat >> /etc/sysctl.conf <<EOF
net.ipv4.ip_nonlocal_bind = 1
EOF
[root@ha1 ~]#sysctl -p

#安装配置haproxy
[root@ha1 ~]#apt update
[root@ha1 ~]#apt -y install haproxy

##添加下面行
[root@ha1 ~]#cat >> /etc/haproxy/haproxy.cfg  <<EOF
listen stats
    mode http
    bind 0.0.0.0:8888
    stats enable
    log global
    stats uri /status
    stats auth admin:123456

listen  kubernetes-api-6443
    bind 10.0.0.100:6443
    mode tcp
    server master1 10.0.0.101:6443 check inter 3s fall 3 rise 3
    server master2 10.0.0.102:6443 check inter 3s fall 3 rise 3
    server master3 10.0.0.103:6443 check inter 3s fall 3 rise 3
EOF

[root@ha1 ~]#systemctl restart  haproxy
```

4.1.2 安装 Keepalived

安装 keepalived 实现 HAProxy的高可用

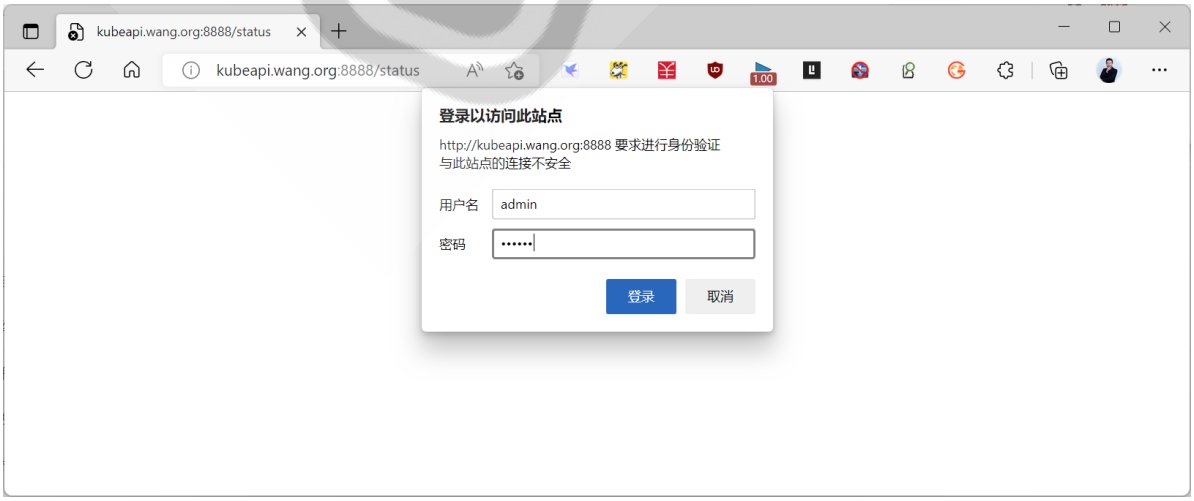
```
[root@ha1 ~]#apt update
[root@ha1 ~]#apt -y install keepalived
[root@ha1 ~]#vim /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
   router_id ha1.wang.org  #指定router_id,#在ha2上为ha2.wang.org
}

vrrp_script check_haproxy {             #定义脚本
    script "/etc/keepalived/check_haproxy.sh"
    interval 1
    weight -30
    fall 3
    rise 2
    timeout 2
```

```
}

vrrp_instance VI_1 {
    state MASTER            #在ha2上为BACKUP
    interface eth0
    garp_master_delay 10
    smtp_alert
    virtual_router_id 66    #指定虚拟路由器ID,ha1和ha2此值必须相同
    priority 100            #在ha2上为80
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 123456    #指定验证密码,ha1和ha2此值必须相同
    }
    virtual_ipaddress {
        10.0.0.100/24 dev eth0  label eth0:1  #指定VIP,ha1和ha2此值必须相同
    }
    track_script {
        check_haproxy                       #调用上面定义的脚本
    }
}


[root@ha1 ~]# cat > /etc/keepalived/check_haproxy.sh <<EOF
#!/bin/bash
/usr/bin/killall -0 haproxy  || systemctl restart haproxy
EOF

[root@ha1 ~]# chmod a+x /etc/keepalived/check_haproxy.sh

[root@ha1 ~]# systemctl restart keepalived
```

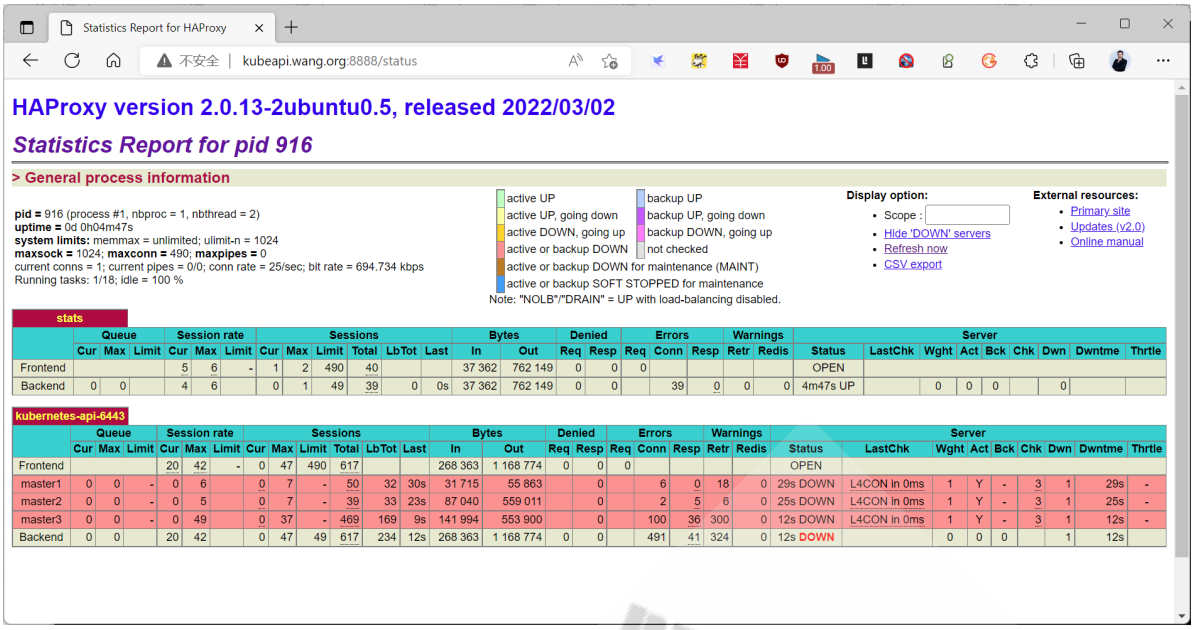
4.1.3 测试访问

浏览器访问验证,用户名密码: admin:123456

```
http://kubeapi.wang.org:8888/status
```



可以看如下显示



4.2 所有主机初始化

4.2.1 配置 ssh key 验证

配置 ssh key 验证,方便后续同步文件

```
ssh-keygen
ssh-copy-id -i ~/.ssh/id_rsa.pub root@10.0.0.102
.....
```

4.2.2 设置主机名和解析

```
hostnamectl set-hostname master1.wang.org

cat > /etc/hosts <<EOF
10.0.0.100 kubeapi.wang.org kubeapi
10.0.0.101 master1.wang.org master1
10.0.0.102 master2.wang.org master2
10.0.0.103 master2.wang.org master3
10.0.0.104 node1.wang.org node1
10.0.0.105 node2.wang.org node2
10.0.0.106 node3.wang.org node3
10.0.0.107 ha1.wang.org ha1
10.0.0.108 ha2.wang.org ha2
EOF

for i in {102..108};do scp /etc/hosts 10.0.0.$i:/etc/ ;done
```

4.2.3 禁用 swap

```
swapoff -a
sed -i '/swap/s/^/#/' /etc/fstab
#或者
systemctl disable --now swap.img.swap
systemctl mask swap.target
```

4.2.4 时间同步

```
#借助于chronyd服务（程序包名称chrony）设定各节点时间精确同步
apt -y install chrony
chronyc sources -v
```

4.2.5 禁用防火墙

```
#禁用默认配置的iptables防火墙服务
ufw disable
ufw status
```

4.2.6 内核参数调整（可选）

允许 iptables 检查桥接流量,若要显式加载此模块，需运行 sudo modprobe br_netfilter，通过运行 lsmod | grep br_netfilter 来验证 br_netfilter 模块是否已加载，

```
sudo modprobe br_netfilter
lsmod | grep br_netfilter
```

为了让 Linux 节点的 iptables 能够正确查看桥接流量，请确认 sysctl 配置中的 net.bridge.bridge-nf-call-iptables 设置为 1。

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter

# 设置所需的 sysctl 参数，参数在重新启动后保持不变
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

# 应用 sysctl 参数而不重新启动
sudo sysctl --system
```

4.3 所有主机安装 Docker 并修改配置

配置 cgroup 驱动程序,容器运行时和 kubelet 都具有名字为 "cgroup driver" 的属性，该属性对于在 Linux 机器上管理 CGroups 而言非常重要。
警告：你需要确保容器运行时和 kubelet 所使用的是相同的 cgroup 驱动，否则 kubelet 进程会失败。

范例:

```
#Ubuntu20.04可以利用内置仓库安装docker
apt update
apt -y install docker.io
```

```
#自Kubernetes v1.22版本开始，未明确设置kubelet的cgroup driver时，则默认即会将其设置为systemd。所有主机修改加速和cgroupdriver
cat > /etc/docker/daemon.json <<EOF
{
"registry-mirrors": [
"https://docker.mirrors.ustc.edu.cn",
"https://hub-mirror.c.163.com",
"https://reg-mirror.qiniu.com",
"https://registry.docker-cn.com"
],
 "exec-opts": ["native.cgroupdriver=systemd"]
}
EOF

for i in {102..106};do scp /etc/docker/daemon.json 10.0.0.$i:/etc/docker/ ;done

systemctl restart docker.service

#验证修改是否成功
docker info  |grep Cgroup
 Cgroup Driver: systemd
 Cgroup Version: 1
```

4.4 所有主机安装 kubeadm、kubelet 和 kubectl

通过国内镜像站点阿里云安装的参考链接：

```
https://developer.aliyun.com/mirror/kubernetes
```

范例: Ubuntu 安装

```
apt-get update && apt-get install -y apt-transport-https
curl https://mirrors.aliyun.com/kubernetes/apt/doc/apt-key.gpg | apt-key add -
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb https://mirrors.aliyun.com/kubernetes/apt/ kubernetes-xenial main
EOF

for i in {102..106};do scp /etc/apt/sources.list.d/kubernetes.list
10.0.0.$i:/etc/apt/sources.list.d/;done

apt-get update

#查看版本
apt-cache madison kubeadm|head
   kubeadm |  1.24.3-00 | https://mirrors.aliyun.com/kubernetes/apt kubernetes-xenial/main amd64 Packages
   kubeadm |  1.24.2-00 | https://mirrors.aliyun.com/kubernetes/apt kubernetes-xenial/main amd64 Packages
   kubeadm |  1.24.1-00 | https://mirrors.aliyun.com/kubernetes/apt kubernetes-xenial/main amd64 Packages
   kubeadm |  1.24.0-00 | https://mirrors.aliyun.com/kubernetes/apt kubernetes-xenial/main amd64 Packages

#安装指定版本
apt install -y  kubeadm=1.24.3-00 kubelet=1.24.3-00 kubectl=1.24.3-00
```

```
#安装最新版本
apt-get install -y kubelet kubeadm kubectl
```

范例: RHEL系统安装

```
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
setenforce 0
yum install -y kubelet kubeadm kubectl
systemctl enable kubelet && systemctl start kubelet
```

4.5 所有主机安装 cri-dockerd

Kubernetes自v1.24移除了对docker-shim的支持，而Docker Engine默认又不支持CRI规范，因而二者将无法直接完成整合。为此，Mirantis和Docker联合创建了cri-dockerd项目，用于为Docker Engine提供一个能够支持到CRI规范的垫片，从而能够让Kubernetes基于CRI控制Docker 。

项目地址：https://github.com/Mirantis/cri-dockerd

cri-dockerd项目提供了预制的二制格式的程序包，用户按需下载相应的系统和对应平台的版本即可完成安装，这里以Ubuntu 20.04 64bits系统环境，以及cri-dockerd目前最新的程序版本v0.2.5为例。

```
curl -LO https://github.com/Mirantis/cri-dockerd/releases/download/v0.2.5/cri-dockerd_0.2.5.3-0.ubuntu-focal_amd64.deb

dpkg -i cri-dockerd_0.2.5.3-0.ubuntu-focal_amd64.deb


[root@master1 ~]#for i in {102..106};do scp cri-dockerd_0.2.5.3-0.ubuntu-focal_amd64.deb 10.0.0.$i: ; ssh 10.0.0.$i "dpkg -i cri-dockerd_0.2.5.3-0.ubuntu-focal_amd64.deb";done

#完成安装后，相应的服务cri-dockerd.service便会自动启动。
```

4.6 所有主机配置 cri-dockerd

众所周知的原因,从国内 cri-dockerd 服务无法下载 k8s.gcr.io上面相关镜像,导致无法启动,所以需要修改 cri-dockerd 使用国内镜像源

```
vim /lib/systemd/system/cri-docker.service
#修改ExecStart行如下
ExecStart=/usr/bin/cri-dockerd --container-runtime-endpoint fd:// --pod-infra-container-image registry.aliyuncs.com/google_containers/pause:3.7

systemctl daemon-reload && systemctl restart cri-docker.service

#同步至所有节点
[root@master1 ~]#for i in {102..106};do scp /lib/systemd/system/cri-docker.service 10.0.0.$i:/lib/systemd/system/cri-docker.service; ssh 10.0.0.$i "systemctl daemon-reload && systemctl restart cri-docker.service";done
```

如果不配置,会出现下面日志提示

```
Aug 21 01:35:17 ubuntu2004 kubelet[6791]: E0821 01:35:17.999712    6791 remote_runtime.go:212] "RunPodSandbox from runtime service failed" err="rpc error: code = Unknown desc = failed pulling image \"k8s.gcr.io/pause:3.6\": Error response from daemon: Get \"https://k8s.gcr.io/v2/\": net/http: request canceled while waiting for connection (Client.Timeout exceeded while awaiting headers)"
```

4.7 提前准备 Kubernetes 初始化所需镜像(可选)

```
##Kubernetes-v1.24.X查看需要下载的镜像,发现k8s.gcr.io无法从国内直接访问
[root@master1 ~]#kubeadm  config images list
k8s.gcr.io/kube-apiserver:v1.24.3
k8s.gcr.io/kube-controller-manager:v1.24.3
k8s.gcr.io/kube-scheduler:v1.24.3
k8s.gcr.io/kube-proxy:v1.24.3
k8s.gcr.io/pause:3.7
k8s.gcr.io/etcd:3.5.3-0
k8s.gcr.io/coredns/coredns:v1.8.6

#Kubernetes-v1.25.0下载镜像地址调整为 registry.k8s.io,但仍然无法从国内直接访问
[root@master1 ~]#kubeadm  config images list
registry.k8s.io/kube-apiserver:v1.25.0
registry.k8s.io/kube-controller-manager:v1.25.0
registry.k8s.io/kube-scheduler:v1.25.0
registry.k8s.io/kube-proxy:v1.25.0
registry.k8s.io/pause:3.8
registry.k8s.io/etcd:3.5.4-0
registry.k8s.io/coredns/coredns:v1.9.3


#查看国内镜像
[root@master1 ~]#kubeadm  config images list  --image-repository registry.aliyuncs.com/google_containers
registry.aliyuncs.com/google_containers/kube-apiserver:v1.24.4
registry.aliyuncs.com/google_containers/kube-controller-manager:v1.24.4
registry.aliyuncs.com/google_containers/kube-scheduler:v1.24.4
registry.aliyuncs.com/google_containers/kube-proxy:v1.24.4
registry.aliyuncs.com/google_containers/pause:3.7
registry.aliyuncs.com/google_containers/etcd:3.5.3-0
registry.aliyuncs.com/google_containers/coredns:v1.8.6
```

```
#从国内镜像站拉取镜像,1.24以上还需要指定--cri-socket路径
[root@master1 ~]#kubeadm  config images pull --kubernetes-version=v1.24.3 --
image-repository registry.aliyuncs.com/google_containers   --cri-socket
unix:///run/cri-dockerd.sockA

#查看拉取的镜像
[root@master1 ~]#docker images
REPOSITORY                                                        TAG        IMAGE
ID       CREATED          SIZE
registry.aliyuncs.com/google_containers/kube-apiserver            v1.24.3
d521dd763e2e   4 weeks ago      130MB
registry.aliyuncs.com/google_containers/kube-scheduler            v1.24.3
3a5aa3a515f5   4 weeks ago      51MB
registry.aliyuncs.com/google_containers/kube-proxy                v1.24.3
2ae1ba6417cb   4 weeks ago      110MB
registry.aliyuncs.com/google_containers/kube-controller-manager   v1.24.3
586c112956df   4 weeks ago      119MB
registry.aliyuncs.com/google_containers/etcd                      3.5.3-0
aebe758cef4c   4 months ago     299MB
registry.aliyuncs.com/google_containers/pause                     3.7
221177c6082a   5 months ago     711kB
registry.aliyuncs.com/google_containers/coredns                   v1.8.6
a4ca41631cc7   10 months ago    46.8MB

#导出镜像
[root@master1 ~]#docker image save `docker image ls --format "{{.Repository}}:
{{.Tag}}"` -o k8s-images-v1.24.3.tar
[root@master1 ~]#gzip k8s-images-v1.24.3.tar
```

4.8 在第一个 master 节点初始化 Kubernetes 集群

kubeadm init 命令参考说明

```
--kubernetes-version：#kubernetes程序组件的版本号，它必须要与安装的kubelet程序包的版本号
相同
--control-plane-endpoint：#多主节点必选项,用于指定控制平面的固定访问地址，可是IP地址或DNS名
称，会被用于集群管理员及集群组件的kubeconfig配置文件的API Server的访问地址,如果是单主节点的控
制平面部署时不使用该选项,注意:kubeadm 不支持将没有 --control-plane-endpoint 参数的单个控制
平面集群转换为高可用性集群。
--pod-network-cidr：#Pod网络的地址范围，其值为CIDR格式的网络地址，通常情况下Flannel网络插
件的默认为10.244.0.0/16，Calico网络插件的默认值为192.168.0.0/16
--service-cidr：#Service的网络地址范围，其值为CIDR格式的网络地址，默认为10.96.0.0/12；通
常，仅Flannel一类的网络插件需要手动指定该地址
--service-dns-domain string #指定k8s集群域名，默认为cluster.local，会自动通过相应的DNS
服务实现解析
--apiserver-advertise-address：#API 服务器所公布的其正在监听的 IP 地址。如果未设置，则使
用默认网络接口。apiserver通告给其他组件的IP地址，一般应该为Master节点的用于集群内部通信的IP地
址，0.0.0.0表示此节点上所有可用地址,非必选项
--image-repository string #设置镜像仓库地址，默认为 k8s.gcr.io,此地址国内可能无法访问,可
以指向国内的镜像地址
--token-ttl #共享令牌（token）的过期时长，默认为24小时，0表示永不过期；为防止不安全存储等原因
导致的令牌泄露危及集群安全，建议为其设定过期时长。未设定该选项时，在token过期后，若期望再向集群
中加入其它节点，可以使用如下命令重新创建token，并生成节点加入命令。kubeadm token create --
print-join-command
--ignore-preflight-errors=Swap” #若各节点未禁用Swap设备，还需附加选项“从而让kubeadm忽略
该错误
--upload-certs #将控制平面证书上传到 kubeadm-certs Secret
```

```
--cri-socket   #v1.24版之后指定连接cri的socket文件路径,注意;不同的CRI连接文件不同
#如果是cRI是containerd，则使用--cri-socket unix:///run/containerd/containerd.sock
#如果是cRI是docker，则使用--cri-socket unix:///var/run/cri-dockerd.sock
#如果是CRI是CRI-o，则使用--cri-socket unix:///var/run/crio/crio.sock
#注意:CRI-o与containerd的容器管理机制不一样，所以镜像文件不能通用。
```

范例: 初始化集群

```
[root@master1 ~]#kubeadm init --control-plane-endpoint="kubeapi.wang.org" --
kubernetes-version=v1.25.0  --pod-network-cidr=10.244.0.0/16 --service-
cidr=10.96.0.0/12 --token-ttl=0   --cri-socket unix:///run/cri-dockerd.sock --
image-repository registry.aliyuncs.com/google_containers  --upload-certs

[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

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

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

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/

You can now join any number of the control-plane node running the following
command on each as root:

  kubeadm join kubeapi.wang.org:6443 --token ihbe5g.6jxdfwym49epsirr \
    --discovery-token-ca-cert-hash
sha256:b7a7abccfc394fe431b8733e05d0934106c0e81abeb0a2bab4d1b7cfd82104c0 \
    --control-plane --certificate-key
ae34c01f75e9971253a39543a289cdc651f23222c0e074a6b7ecb2dba667c059

Please note that the certificate-key gives access to cluster sensitive data, keep
it secret!
As a safeguard, uploaded-certs will be deleted in two hours; If necessary, you
can use
"kubeadm init phase upload-certs --upload-certs" to reload certs afterward.

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

kubeadm join kubeapi.wang.org:6443 --token ihbe5g.6jxdfwym49epsirr \
    --discovery-token-ca-cert-hash
sha256:b7a7abccfc394fe431b8733e05d0934106c0e81abeb0a2bab4d1b7cfd82104c0
```

如果想重新初始化,可以执行下面

```
#如果有工作节点,先在工作节点执行,再在control节点执行下面操作
kubeadm reset -f --cri-socket unix:///run/cri-dockerd.sock
rm -rf  /etc/cni/net.d/  $HOME/.kube/config
reboot
```

4.9 在第一个 master 节点生成 kubectl 命令的授权文件

kubectl是kube-apiserver的命令行客户端程序，实现了除系统部署之外的几乎全部的管理操作，是kubernetes管理员使用最多的命令之一。kubectl需经由API server认证及授权后方能执行相应的管理操作，kubeadm部署的集群为其生成了一个具有管理员权限的认证配置文件/etc/kubernetes/admin.conf，它可由kubectl通过默认的“$HOME/.kube/config”的路径进行加载。当然，用户也可在kubectl命令上使用--kubeconfig选项指定一个别的位置。

下面复制认证为Kubernetes系统管理员的配置文件至目标用户（例如当前用户root）的家目录下：

```
#可复制4.8的结果执行下面命令
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

4.10 实现 kubectl 命令补全

kubectl 命令功能丰富,默认不支持命令补会,可以用下面方式实现

```
kubectl completion bash > /etc/profile.d/kubectl_completion.sh
. /etc/profile.d/kubectl_completion.sh
exit
```

4.11 在第一个 master 节点配置网络组件

Kubernetes系统上Pod网络的实现依赖于第三方插件进行，这类插件有近数十种之多，较为著名的有flannel、calico、canal和kube-router等，简单易用的实现是为CoreOS提供的flannel项目。下面的命令用于在线部署flannel至Kubernetes系统之上：

首先，下载适配系统及硬件平台环境的flanneld至每个节点，并放置于/opt/bin/目录下。我们这里选用flanneld-amd64，目前最新的版本为v0.19.1，因而，我们需要在集群的每个节点上执行如下命令：

提示：下载flanneld的地址为 https://github.com/flannel-io/flannel/releases

随后，在初始化的第一个master节点k8s-master01上运行如下命令，向Kubernetes部署kube-flannel。

```
#默认没有网络插件,所以显示如下状态
[root@master1 ~]#kubectl get nodes
NAME               STATUS     ROLES           AGE   VERSION
master1.wang.org   NotReady   control-plane   17m   v1.24.3

[root@master1 ~]#kubectl apply -f https://raw.githubusercontent.com/flannel-
io/flannel/master/Documentation/kube-flannel.yml

#稍等一会儿,可以看到下面状态
[root@master1 ~]#kubectl get nodes
NAME               STATUS   ROLES           AGE   VERSION
master1.wang.org   Ready    control-plane   23m   v1.24.3
```

4.12 将所有 worker 节点加入 Kubernetes 集群

在所有worker节点执行下面操作,加上集群

```
#复制上面第4.8步的执行结果,额外添加--cri-socket选项修改为下面执行
[root@node1 ~]#kubeadm join kubeapi.wang.org:6443 --token
ihbe5g.6jxdfwym49epsirr \
    --discovery-token-ca-cert-hash
sha256:b7a7abccfc394fe431b8733e05d0934106c0e81abeb0a2bab4d1b7cfd82104c0  --cri-
socket unix:///run/cri-dockerd.sock

[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...

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

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


[root@node1 ~]#docker images
REPOSITORY                                          TAG       IMAGE ID
CREATED         SIZE
rancher/mirrored-flannelcni-flannel                 v0.19.1   252b2c3ee6c8   12
days ago     62.3MB
registry.aliyuncs.com/google_containers/kube-proxy  v1.24.3   2ae1ba6417cb   5
weeks ago     110MB
rancher/mirrored-flannelcni-flannel-cni-plugin      v1.1.0    fcecffc7ad4a   2
months ago    8.09MB
registry.aliyuncs.com/google_containers/pause       3.7       221177c6082a   5
months ago    711kB
registry.aliyuncs.com/google_containers/coredns     v1.8.6    a4ca41631cc7   10
months ago   46.8MB

#可以将镜像导出到其它worker节点实现加速
[root@node1 ~]#docker image save `docker image ls --format "{{.Repository}}:
{{.Tag}}"` -o k8s-images-v1.24.3.tar
[root@node1 ~]#gzip  k8s-images-v1.24.3.tar
[root@node1 ~]#scp k8s-images-v1.24.3.tar.gz node2:
[root@node1 ~]#scp k8s-images-v1.24.3.tar.gz node3:
[root@node2 ~]#docker load -i k8s-images-v1.24.3.tar.gz

[root@master1 ~]#kubectl get nodes
NAME               STATUS   ROLES           AGE   VERSION
master1.wang.org   Ready    control-plane   58m   v1.24.3
node1.wang.org     Ready    <none>          44m   v1.24.3
node2.wang.org     Ready    <none>          18m   v1.24.3
node3.wang.org     Ready    <none>          65s   v1.24.3
```

4.13 测试应用编排及服务访问

至此一个master附带有三个worker的kubernetes集群基础设施已经部署完成，用户随后即可测试其核心功能。

demoapp是一个web应用,可将demoapp以Pod的形式编排运行于集群之上，并通过在集群外部进行访问：

```
[root@master1 ~]#kubectl create deployment demoapp --
image=ikubernetes/demoapp:v1.0 --replicas=3

[root@master1 ~]#kubectl get pod -o wide
NAME                       READY   STATUS    RESTARTS   AGE     IP           NODE
            NOMINATED NODE   READINESS GATES
demoapp-78b49597cf-7pdww   1/1     Running   0          2m39s   10.244.2.2
node2.wang.org   <none>           <none>
demoapp-78b49597cf-wcjkp   1/1     Running   0          2m39s   10.244.2.3
node2.wang.org   <none>           <none>
demoapp-78b49597cf-zmlmv   1/1     Running   0          2m39s   10.244.1.4
node3.wang.org   <none>           <none>

[root@master1 ~]#curl 10.244.2.2
iKubernetes demoapp v1.0 !! ClientIP: 10.244.0.0, ServerName: demoapp-
78b49597cf-7pdww, ServerIP: 10.244.2.2!
[root@master1 ~]#curl 10.244.2.3
iKubernetes demoapp v1.0 !! ClientIP: 10.244.0.0, ServerName: demoapp-
78b49597cf-wcjkp, ServerIP: 10.244.2.3!
[root@master1 ~]#curl 10.244.1.4
iKubernetes demoapp v1.0 !! ClientIP: 10.244.0.0, ServerName: demoapp-
78b49597cf-zmlmv, ServerIP: 10.244.1.4!


#使用如下命令了解Service对象demoapp使用的NodePort，格式:<集群端口>:<POd端口>,以便于在集群外部进行访问
[root@master1 ~]#kubectl create service nodeport demoapp --tcp=80:80

[root@master1 ~]#kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
demoapp      NodePort    10.110.101.190   <none>        80:30037/TCP   102s
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP        67m

[root@master1 ~]#curl 10.110.101.190
iKubernetes demoapp v1.0 !! ClientIP: 10.244.0.0, ServerName: demoapp-
78b49597cf-wcjkp, ServerIP: 10.244.2.3!
[root@master1 ~]#curl 10.110.101.190
iKubernetes demoapp v1.0 !! ClientIP: 10.244.0.0, ServerName: demoapp-
78b49597cf-zmlmv, ServerIP: 10.244.1.4!
[root@master1 ~]#curl 10.110.101.190
iKubernetes demoapp v1.0 !! ClientIP: 10.244.0.0, ServerName: demoapp-
78b49597cf-7pdww, ServerIP: 10.244.2.2!

#用户可以于集群外部通过“http://NodeIP:30037”这个URL访问demoapp上的应用，例如于集群外通过浏览器访问“http://<kubernetes-node>:30037”。
[root@rocky8 ~]#curl 10.0.0.100:30037
iKubernetes demoapp v1.0 !! ClientIP: 10.244.0.0, ServerName: demoapp-
78b49597cf-wcjkp, ServerIP: 10.244.2.3!
[root@rocky8 ~]#curl 10.0.0.101:30037
```

```
iKubernetes demoapp v1.0 !! ClientIP: 10.244.1.0, ServerName: demoapp-
78b49597cf-7pdww, ServerIP: 10.244.2.2!
[root@rocky8 ~]#curl 10.0.0.102:30037
iKubernetes demoapp v1.0 !! ClientIP: 10.244.2.0, ServerName: demoapp-
78b49597cf-zmlmv, ServerIP: 10.244.1.4!

#扩容
[root@master1 ~]#kubectl scale deployment demoapp --replicas 5
deployment.apps/demoapp scaled
[root@master1 ~]#kubectl get pod
NAME                       READY   STATUS    RESTARTS   AGE
demoapp-78b49597cf-44hqj   1/1     Running   0          41m
demoapp-78b49597cf-45jd8   1/1     Running   0          9s
demoapp-78b49597cf-49js5   1/1     Running   0          41m
demoapp-78b49597cf-9lw2z   1/1     Running   0          9s
demoapp-78b49597cf-jtwkt   1/1     Running   0          41m

#缩容
[root@master1 ~]#kubectl scale deployment demoapp --replicas 2
deployment.apps/demoapp scaled

#可以看到销毁pod的过程
[root@master1 ~]#kubectl get pod
NAME                       READY   STATUS        RESTARTS   AGE
demoapp-78b49597cf-44hqj   1/1     Terminating   0          41m
demoapp-78b49597cf-45jd8   1/1     Terminating   0          53s
demoapp-78b49597cf-49js5   1/1     Running       0          41m
demoapp-78b49597cf-9lw2z   1/1     Terminating   0          53s
demoapp-78b49597cf-jtwkt   1/1     Running       0          41m

#再次查看,最终缩容成功
[root@master1 ~]#kubectl get pod
NAME                       READY   STATUS    RESTARTS   AGE
demoapp-78b49597cf-49js5   1/1     Running   0          42m
demoapp-78b49597cf-jtwkt   1/1     Running   0          42m
```

4.14 扩展 Kubernetes 集群为多主模式

```
https://kubernetes.io/zh-cn/docs/setup/production-environment/tools/kubeadm/high-availability/#manual-certs
```

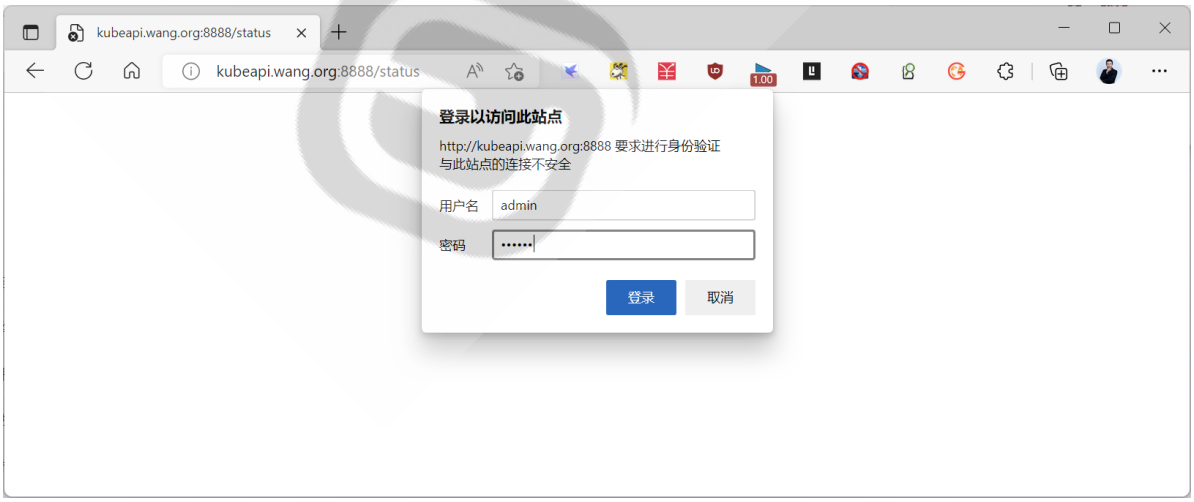
在 master2 和 master3 重复上面的 4.2-4.7 步后,再执行下面操作加入集群

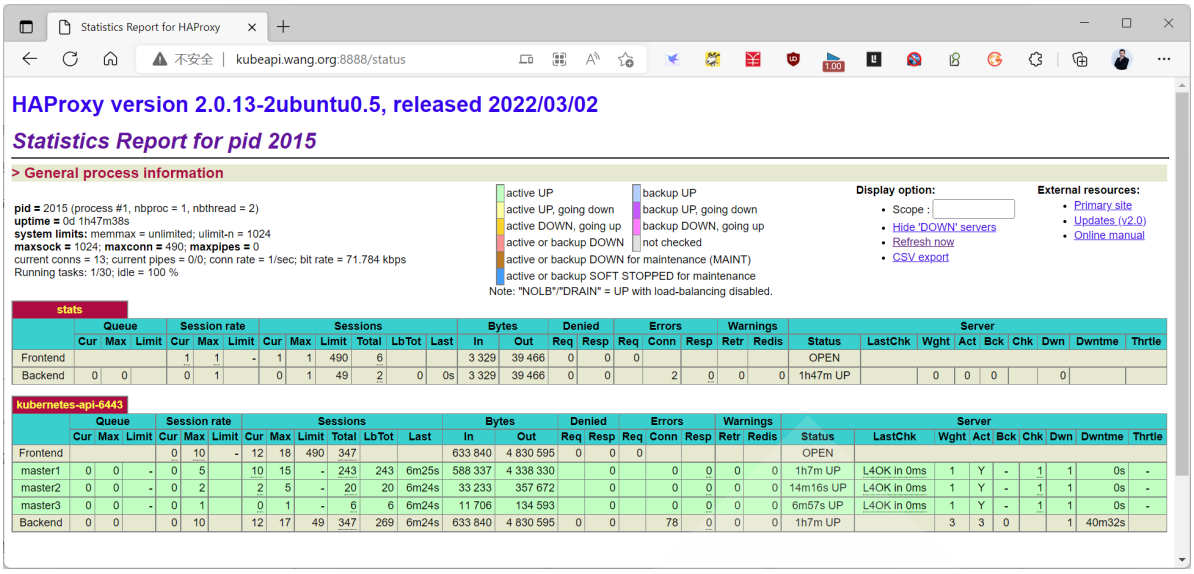
```
[root@master2 ~]#kubeadm join kubeapi.wang.org:6443 --token
ihbe5g.6jxdfwym49epsirr \
--discovery-token-ca-cert-hash
sha256:b7a7abccfc394fe431b8733e05d0934106c0e81abeb0a2bab4d1b7cfd82104c0 --
control-plane \
--certificate-key
ae34c01f75e9971253a39543a289cdc651f23222c0e074a6b7ecb2dba667c059 \
--cri-socket unix:///run/cri-dockerd.sock


This node has joined the cluster and a new control plane instance was created:

* Certificate signing request was sent to apiserver and approval was received.
```

```
* The Kubelet was informed of the new secure connection details.
* Control plane label and taint were applied to the new node.
* The Kubernetes control plane instances scaled up.
* A new etcd member was added to the local/stacked etcd cluster.

To start administering your cluster from this node, you need to run the
following as a regular user:

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

Run 'kubectl get nodes' to see this node join the cluster.


[root@master2 ~]#mkdir -p $HOME/.kube
[root@master2 ~]#sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
[root@master2 ~]#sudo chown $(id -u):$(id -g) $HOME/.kube/config


[root@master1 ~]#kubectl get nodes
NAME               STATUS     ROLES           AGE     VERSION
master1.wang.org   Ready      control-plane   29m     v1.24.3
master2.wang.org   Ready      control-plane   26m     v1.24.3
master3.wang.org   Ready      control-plane   3m48s   v1.24.3
node1.wang.org     Ready      <none>          19m     v1.24.3
node2.wang.org     Ready      <none>          18m     v1.24.3
node3.wang.org     Ready      <none>          18m     v1.24.3
```

浏览器访问验证,用户名密码: admin:123456

```
http://kubeapi.wang.org:8888/status
```





HAProxy version 2.0.13-2ubuntu0.5, released 2022/03/02

Statistics Report for pid 2015

> General process information

pid = 2015 (process #1, nbproc = 1, nbthread = 2)
uptime = 0d 1h47m38s
system limits: memmax = unlimited; ulimit-n = 1024
maxsock = 1024; **maxconn =** 490; **maxpipes =** 0
current conns = 13; current pipes = 0/0; conn rate = 1/sec; bit rate = 71.784 kbps
Running tasks: 1/30; idle = 100 %

- active UP
- active UP, going down
- active DOWN, going up
- active or backup DOWN
- active or backup DOWN for maintenance (MAINT)
- active or backup SOFT STOPPED for maintenance
- backup UP
- backup UP, going down
- backup DOWN, going up
- not checked

Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:
- Scope :
- Hide 'DOWN' servers
- Refresh now
- CSV export

External resources:
- Primary site
- Updates (v2.0)
- Online manual

stats

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				1	1	-	1	1	490	6			3 329	39 466	0	0	0					OPEN								
Backend	0	0		0	1		0	1	49	2	0	0s	3 329	39 466	0	0		2	0	0	0	1h47m UP		0	0	0		0		

kubernetes-api-6443

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	10	-	12	18	490	347			633 840	4 830 595	0	0	0					OPEN								
master1	0	0	-	0	5		10	15	-	243	243	6m25s	588 337	4 338 330		0		0	0	0	0	1h7m UP	L4OK in 0ms	1	Y	-	1	1	0s	-
master2	0	0	-	0	2		2	5	-	20	20	6m24s	33 233	357 672		0		0	0	0	0	14m16s UP	L4OK in 0ms	1	Y	-	1	1	0s	-
master3	0	0	-	0	1		0	1	-	6	6	6m24s	11 706	134 593		0		0	0	0	0	6m57s UP	L4OK in 0ms	1	Y	-	1	1	0s	-
Backend	0	0		0	10		12	17	49	347	269	6m24s	633 840	4 830 595	0	0		78	0	0	0	1h7m UP		3	3	0		1	40m32s	