kubeadm 构建单 master 多 node 的 K8S 集群

（Kubernetes 版本：1.28.1　containerd 运行时）

张岩峰老师微信，加我微信，邀请你加入 VIP 交流答疑群：

微信号：ZhangYanFeng0429

二维码：

k8s github 地址：

https://github.com/kubernetes/kubernetes/releases

一、环境规划

实验环境规划：

podSubnet（pod 网段）：10.244.0.0/16

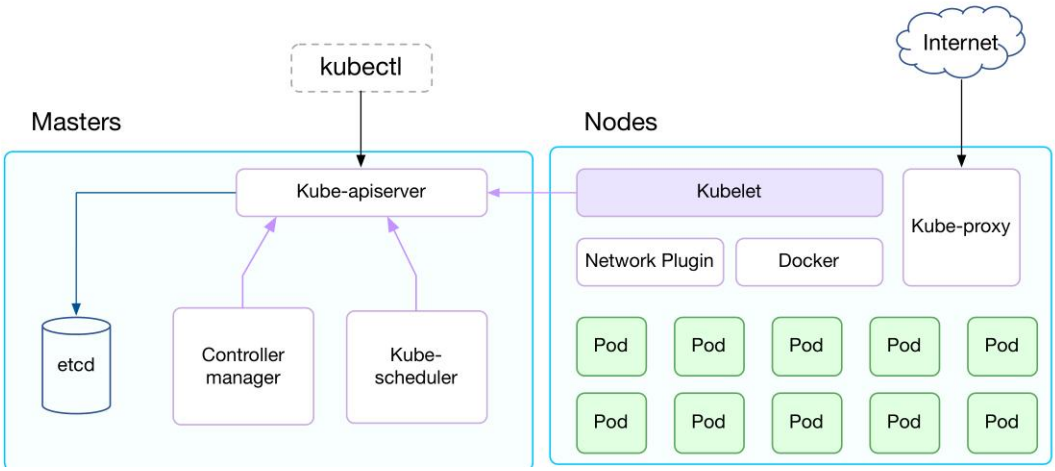
serviceSubnet（service 网段）：10.10.0.0/16

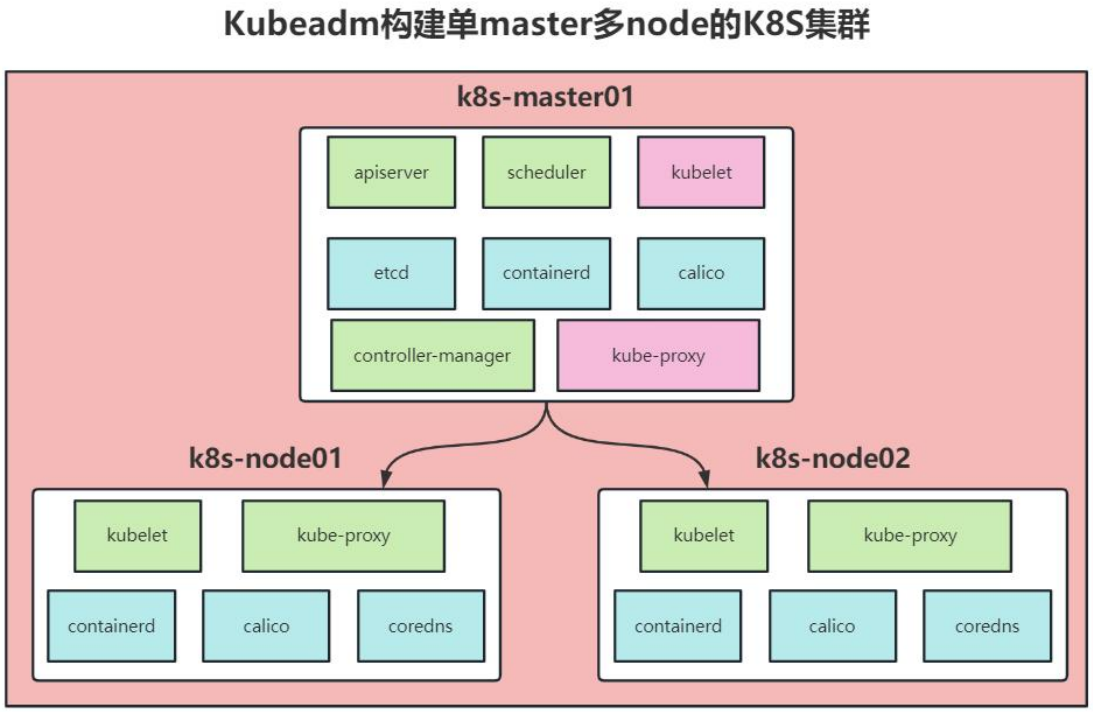
系统版本：Rocky 8.8 操作系统版本

角色	IP	主机名	组件	硬件
控制节点	192.168.128.11	k8s-master01	apiserver controller-manager scheduler etcd containerd	CPU：4vCPU 硬盘：100G 内存：4GB 开启虚拟化
工作	192.168.128.21	k8s-node01	kubelet	CPU：6vCPU

节点			kube-proxy containerd calico coredns	硬盘：100G 内存：6GB 开启虚拟化
工作节点	192.168.128.22	k8s-node02	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：6GB 开启虚拟化

拓扑图：





版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

二、初始化系统环境

1、配置机器主机名

128.11 节点执行：

```
[root@192 ~]# hostnamectl set-hostname k8s-master01 && bash
```

128.21 节点执行：

```
[root@192 ~]# hostnamectl set-hostname k8s-node01 && bash
```

128.22 节点执行：

```
[root@192 ~]# hostnamectl set-hostname k8s-node02 && bash
```

2、配置 hosts 解析

128.11、128.21、128.22 节点执行如下：

```
[root@k8s-master01 ~]# vi /etc/hosts
192.168.128.11 k8s-master01
192.168.128.21 k8s-node01
192.168.128.22 k8s-node02
```

3、配置主机之间无密码登录（生产环境不建议执行此操作）

128.11、128.21、128.22 节点执行如下：

```
[root@k8s-master01 ~]# ssh-keygen
[root@k8s-master01 ~]# ssh-copy-id k8s-master01
[root@k8s-master01 ~]# ssh-copy-id k8s-node01
[root@k8s-master01 ~]# ssh-copy-id k8s-node02
```

4、关闭交换分区 swap，提升性能

128.11、128.21、128.22 节点执行如下：

```
[root@k8s-master01 ~]# swapoff -a

永久关闭：注释 swap 挂载，给 swap 这行开头加一下注释
[root@k8s-master01 ~]# vi /etc/fstab
#/dev/mapper/centos-swap swap        swap    defaults        0 0
```

为什么要关闭 swap 交换分区？

swap 是交换分区，如果机器内存不够，会使用 swap 分区，但是 swap 分区的性能较低，k8s 设计的时候为了能提升性能，默认是不允许使用 swap 分区的。kubeadm 初始化的时候会检测 swap 是否关闭，如果没关闭，那就初始化失败。如果不想要关闭交换分区，安装 k8s 的时候可以指定 --ignore-preflight-errors=swap 来解决。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

5、修改机器内核参数

128.11、128.21、128.22 节点执行如下：

```
[root@k8s-master01 ~]# modprobe br_netfilter
[root@k8s-master01 ~]# echo "modprobe br_netfilter" >> /etc/profile
[root@k8s-master01 ~]# cat > /etc/sysctl.d/k8s.conf <<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF
[root@k8s-master01 ~]# sysctl -p /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1

一键执行：
modprobe br_netfilter
echo "modprobe br_netfilter" >> /etc/profile
cat > /etc/sysctl.d/k8s.conf <<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF
sysctl -p /etc/sysctl.d/k8s.conf
```

注意：这三条内核参数配置必须设置，否则在初始化 k8s 集群的时候，会报错。

6、关闭 firewalld 防火墙

128.11、128.21、128.22 节点执行如下：

```
[root@k8s-master01 ~]# systemctl stop firewalld;systemctl disable firewalld
```

7、关闭 selinux

128.11、128.21、128.22 节点执行如下：

```
[root@k8s-master01 ~]# sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
[root@k8s-master01 ~]# setenforce 0

一键执行：
sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
setenforce 0
```

8、配置阿里云 repo 源

128.11、128.21、128.22 节点执行如下：

```
yum -y install wget
cd /etc/yum.repos.d/
wget http://mirrors.aliyun.com/repo/Centos-8.repo
yum clean all
yum makecache
yum -y install lrzsz net-tools
```

9、配置时间同步

128.11、128.21、128.22 节点执行如下：

```
# 设置时区
[root@k8s-master01 ~]# timedatectl set-timezone Asia/Shanghai

[root@k8s-master01 ~]# yum -y install chrony
[root@k8s-master01 ~]# vi /etc/chrony.conf
server time1.aliyun.com iburst
server time2.aliyun.com iburst
[root@k8s-master01 ~]# systemctl restart chronyd
[root@k8s-master01 ~]# systemctl enable chronyd
```

10、开启 ipvs

128.11、128.21、128.22 节点执行如下：

```
[root@k8s-master01 ~]# vi /etc/sysconfig/modules/ipvs.modules
#!/bin/bash
ipvs_modules="ip_vs   ip_vs_lc   ip_vs_wlc   ip_vs_rr   ip_vs_wrr
ip_vs_lblc ip_vs_lblcr ip_vs_dh ip_vs_sh ip_vs_nq ip_vs_sed ip_vs_ftp
nf_conntrack"
for kernel_module in ${ipvs_modules}; do
 /sbin/modinfo -F filename ${kernel_module} > /dev/null 2>&1
 if [ 0 -eq 0 ]; then
 /sbin/modprobe ${kernel_module}
 fi
done
[root@k8s-master01           ~]#          chmod          755
/etc/sysconfig/modules/ipvs.modules             &&            bash
/etc/sysconfig/modules/ipvs.modules && lsmod | grep ip_vs
```

问题：

- ipvs 是什么？

ipvs（IP Virtual Server）实现了传输层负载均衡，也就是我们常说的 4 层 LAN 交换，作为 Linux 内核的一部分。ipvs 运行在主机上，在真实服务器集

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

群前充当负载均衡器。ipvs 可以将基于 TCP 和 UDP 的服务请求转发到真实服务器上，并使真实服务器的服务在单个 IP 地址上显示为虚拟服务。

- ipvs 和 iptable 对比分析

kube-proxy 支持 iptables 和 ipvs 两种模式，在 kubernetes v1.8 中引入了 ipvs 模式，在 v1.9 中处于 beta 阶段，在 v1.11 中已经正式可用了。iptables 模式在 v1.1 中就添加支持了，从 v1.2 版本开始 iptables 就是 kube-proxy 默认的操作模式，ipvs 和 iptables 都是基于 netfilter 的，但是 ipvs 采用的是 hash 表，因此当 service 数量达到一定规模时，hash 查表的速度优势就会显现出来，从而提高 service 的服务性能。那么 ipvs 模式和 iptables 模式之间有哪些差异呢？

1、ipvs 为大型集群提供了更好的可扩展性和性能

2、ipvs 支持比 iptables 更复杂的复制均衡算法（最小负载、最少连接、加权等等）

3、ipvs 支持服务器健康检查和连接重试等功能

11、安装基础软件包

128.11、128.21、128.22 节点执行如下：

```
[root@k8s-master01 ~]# yum install -y yum-utils device-mapper-persistent-data lvm2 wget net-tools nfs-utils lrzsz gcc gcc-c++ make cmake libxml2-devel openssl-devel curl curl-devel unzip sudo libaio-devel wget vim ncurses-devel autoconf automake zlib-devel epel-release openssh-server socat ipvsadm conntrack telnet ipvsadm
```

三、构建 K8S 集群

1、安装 k8s repo 源

128.11、128.21、128.22 节点执行如下：

```
# 下面是阿里云的 yum 源
[root@k8s-master01 ~]# vi /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64/
enabled=1
gpgcheck=0
[root@k8s-master01 ~]# yum makecache
[root@k8s-master01 ~]# yum clean all
```

2、部署 containerd 容器

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

在 Kubernetes 集群中，containerd 是容器运行时，它的主要作用是负责管理节点上的容器，实现容器的创建、销毁、运行、暂停、恢复等操作。而 Pod 是 Kubernetes 中最基本的调度单元，一个 Pod 包含一个或多个紧密关联的容器，在 Kubernetes 集群中，当一个 Pod 被调度到一个节点上时，Kubernetes 就会基于 containerd 在 pod 里运行容器。

128.11、128.21、128.22 节点执行如下：

（1）安装 docker-ce 源：

```
[root@k8s-master01 ~]# yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
```

（2）安装、配置、启动 containerd 容器：

```
# 安装
[root@k8s-master01 ~]# yum -y install containerd

# 导出默认的 containerd 配置
[root@k8s-master01 ~]# containerd config default > /etc/containerd/config.toml

# 修改 containerd 配置
[root@k8s-master01 ~]# vi /etc/containerd/config.toml
# 修改 cgroup Driver 为 systemd
SystemdCgroup = true
```

```
[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
```

```
# 镜像加速，endpoint 位置添加阿里云的镜像源
        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
          endpoint = ["https://yz9elmr9.mirror.aliyuncs.com"]
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
    [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://yz9elmr9.mirror.aliyuncs.com"]

    [plugins."io.containerd.grpc.v1.cri".x509_key_pair_streaming]
      tls_cert_file = ""
      tls_key_file = ""
```

```
    # 更改 sandbox_image
    "registry.aliyuncs.com/google_containers/pause:3.6"
    restrict_oom_score_adj = false
    sandbox_image = "registry.aliyuncs.com/google_containers/pause:3.6"
    selinux_category_range = 1024
    stats_collect_period = 10
```

```
    # 启动
    [root@k8s-node01 ~]# systemctl restart containerd && systemctl enable containerd && systemctl status containerd
```

上述修改的内容解释说明：

SystemdCgroup = true 表示把 containerd 驱动变成 systemd，跟 kubelet 驱动保持一致。

pause 容器：当 Kubernetes 启动一个 Pod 时，会为其创建一个 Pause 容器。Pause 容器是一个极小的 Linux 容器，它不做任何事情，只是为 Pod 中的其他容器创建一个 Linux 命名空间和一个网络命名空间，并且共享了一个 IPC 命名空间，以便其他容器可以与之通信。

3、安装初始化 k8s 需要的软件包

128.11、128.21、128.22 节点执行如下：

```
    [root@k8s-master01 ~]# yum install -y kubelet-1.28.1 kubeadm-1.28.1 kubectl-1.28.1
```

提示：每个软件包的作用

- kubelet

kubelet：kubelet 是 Kubernetes 集群中的一个核心组件，是每个节点上的代理服务，负责与主控制节点通信，管理节点上的 Pod 和容器。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

kubelet 的主要职责包括：

监控 pod 的状态并按需启动或停止容器、检查容器是否正常运行、与主控制节点通信，将节点状态和 Pod 状态上报给主控制节点、管理容器的生命周期，包括启动、停止、重启等、拉取镜像。

- kubeadm

kubeadm：用于初始化、升级 k8s 集群的命令行工具。

- kubectl

kubectl：用于和集群通信的命令行，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。

4、kubeadm 初始化 k8s 集群

在 master 节点使用 kubeadm 初始化 k8s 集群：

1、生成初始化文件

```
（1）获取默认的初始化参数文件
   [root@k8s-master01 ~]# kubeadm config print init-defaults > init.default.yaml

（2）修改初始化文件
   [root@k8s-master01 ~]# vi init.default.yaml
   apiVersion: kubeadm.k8s.io/v1beta3
   bootstrapTokens:
   - groups:
     - system:bootstrappers:kubeadm:default-node-token
     token: abcdef.0123456789abcdef
     ttl: 24h0m0s
     usages:
     - signing
     - authentication
   kind: InitConfiguration
   localAPIEndpoint:
     advertiseAddress: 192.168.128.11
     bindPort: 6443
   nodeRegistration:
     criSocket: unix:///run/containerd/containerd.sock
     imagePullPolicy: IfNotPresent
     name: k8s-master01
     taints: null
   ---
```

```
  apiServer:
    timeoutForControlPlane: 4m0s
  apiVersion: kubeadm.k8s.io/v1beta3
  certificatesDir: /etc/kubernetes/pki
  clusterName: kubernetes
  controllerManager: {}
  dns: {}
  etcd:
    local:
      dataDir: /var/lib/etcd
  imageRepository: registry.aliyuncs.com/google_containers
  kind: ClusterConfiguration
  kubernetesVersion: 1.28.1
  networking:
    dnsDomain: cluster.local
    serviceSubnet: 10.10.0.0/16
    podSubnet: 10.244.0.0/16
  scheduler: {}
  ---
  apiVersion: kubeproxy.config.k8s.io/v1alpha1
  kind: KubeProxyConfiguration
  mode: ipvs
  ---
  apiVersion: kubelet.config.k8s.io/v1beta1
  kind: KubeletConfiguration
  cgroupDriver: systemd
```

初始化文件说明：

```
  [root@k8s-master01 ~]# vi init.default.yaml
  apiVersion: kubeadm.k8s.io/v1beta3
  bootstrapTokens:
  - groups:
    - system:bootstrappers:kubeadm:default-node-token
    token: abcdef.0123456789abcdef
    ttl: 24h0m0s
    usages:
    - signing
    - authentication
  kind: InitConfiguration
  localAPIEndpoint:
    advertiseAddress: 192.168.128.11  #master 节点 IP 地址
    bindPort: 6443   #kube-apiserver 组件监听的地址
```

```
    nodeRegistration:
      criSocket: unix:///run/containerd/containerd.sock   #containerd容器运行时的路径
      imagePullPolicy: IfNotPresent  #镜像拉取策略
      name: k8s-master01    #加入到集群中，显示的名称
      taints: null    #在使用 kubeadm 初始化 Kubernetes 集群时，若不指定污点，将默认为 Taints 值为 null，这意味着新生成的所有 Node 节点都不带任何污点，可以接受所有 Pod 的调度请求，不会对 Pod 的调度造成任何限制。
    ---
    apiServer:
      timeoutForControlPlane: 4m0s
    apiVersion: kubeadm.k8s.io/v1beta3
    certificatesDir: /etc/kubernetes/pki  #证书生成的位置
    clusterName: kubernetes  #集群名称
    controllerManager: {}
    dns: {}
    etcd:
      local:
        dataDir: /var/lib/etcd    #etcd 的数据目录
    imageRepository: registry.aliyuncs.com/google_containers  #镜像加速地址
    kind: ClusterConfiguration
    kubernetesVersion: 1.28.1  #集群版本号
    networking:
      dnsDomain: cluster.local
      serviceSubnet: 10.10.0.0/16  #service 网段范围
      podSubnet: 10.244.0.0/16    #pod 网段范围
    scheduler: {}
    # 下面则表示启用 ipvs
    ---
    apiVersion: kubeproxy.config.k8s.io/v1alpha1
    kind: KubeProxyConfiguration
    mode: ipvs
    ---
    apiVersion: kubelet.config.k8s.io/v1beta1
    kind: KubeletConfiguration
    cgroupDriver: systemd
```

2、初始化集群

```
    # 检查初始化要拉取的镜像有哪些
    [root@k8s-master01 ~]# kubeadm config images list
```

```
[root@k8s-master01 ~]# kubeadm config images list
I0522 08:29:20.870609    2409 version.go:256] remote version is much newer: v1.27.2; falling back to: stable-1.26
registry.k8s.io/kube-apiserver:v1.26.5
registry.k8s.io/kube-controller-manager:v1.26.5
registry.k8s.io/kube-scheduler:v1.26.5
registry.k8s.io/kube-proxy:v1.26.5
registry.k8s.io/pause:3.9
registry.k8s.io/etcd:3.5.6-0
registry.k8s.io/coredns/coredns:v1.9.3
[root@k8s-master01 ~]#
```

```
# 拉取初始化时需要的镜像（可不执行）
[root@k8s-master01 ~]# kubeadm config images pull

# 直接进行初始化，镜像不存在会去拉取
[root@k8s-master01 ~]# kubeadm init --config=init.default.yaml
```

显示如下，表示安装完成：

```
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/

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

kubeadm join 192.168.128.11:6443 --token abcdef.0123456789abcdef \
        --discovery-token-ca-cert-hash sha256:cfc8252e72a9c2eb60a30f2f0e73e3832e4edd9b77b3181ed14ddab45f2cfd79
[root@k8s-master01 ~]#
```

创建与kube-apiserver交互的认证文件，必须拥有此文件，kube-apiserver才能鉴权通过，才能够允许你访问kubernetes集群资源。

将node节点加入集群的命令，需要保存下来，每个人的都不一样，这个token一定时间后会过期。

扩展：kubeadm init 初始化流程分析

kubeadm 在执行安装之前进行了相当细致的环境检测，下面看一看：

（1）检查执行 init 命令的用户是否为 root，如果不是 root，直接快速失败（fail fast）。

（2）检查待安装的 k8s 版本是否被当前版本的 kubeadm 支持（kubeadm 版本>=待安装 k8s 版本）。

（3）检查防火墙，如果防火墙未关闭，提示开放端口 10250。

（4）检查端口是否已被占用，6443（或你指定的监听端口）、10257、10259。

（5）检查文件是否已经存在，/etc/kubernetes/manifests/*.yaml。

（6）检查是否存在代理，连接本机网络、服务网络、Pod 网络，都会检查，目前不允许代理。

（7）检查容器运行时，使用 CRI 还是 Docker，如果是 Docker，进一步检查 Docker 服务是否已启动，是否设置了开机自启动。

（8）对于 Linux 系统，会额外检查以下内容：

（8.1）检查以下命令是否存在：crictl、ip、iptables、mount、nsenter、ebtables、ethtool、socat、tc、touch。

（8.2）检查 /proc/sys/net/bridge/bridge-nf-call-iptables 、/proc/sys/net/ipv4/ip-forward 内容是否为 1。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

（8.3）检查 swap 是否是关闭状态。

（9）检查内核是否被支持，Docker 版本及后端存储 GraphDriver 是否被支持。 对于 Linux 系统，还需检查 OS 版本和 cgroup 支持程度（支持哪些资源的隔离）。

（10）检查主机名访问可达性。

（11）检查 kubelet 版本，要高于 kubeadm 需要的最低版本，同时不高于待安装的 k8s 版本。

（12）检查 kubelet 服务是否开机自启动。

（13）检查 10250 端口是否被占用。

（14）如果开启 IPVS 功能，检查系统内核是否加载了 ipvs 模块。

（15）对于 etcd，如果使用 Local etcd，则检查 2379 端口是否被占用，/var/lib/etcd/是否为空目录。如果使用 External etcd，则检查证书文件是否存在（CA、key、cert），验证 etcd 服务版本是否符合要求。

（16）如果使用 IPv6，检查/proc/sys/net/bridge/bridge-nf-call-iptables、/proc/sys/net/ipv6/conf/default/forwarding 内容是否为 1。

以上就是 kubeadm init 需要检查的所有项目了！

3、创建 kubectl 授权文件

配置 kubectl 的配置文件 config，相当于对 kubectl 进行授权，这样 kubectl 命令可以使用这个证书对 k8s 集群进行管理。

```
[root@k8s-master01 ~]# mkdir -p $HOME/.kube
[root@k8s-master01 ~]# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
[root@k8s-master01 ~]# sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

4、查看集群节点

```
[root@k8s-master01 ~]# kubectl get nodes
NAME           STATUS     ROLES           AGE    VERSION
k8s-master01   NotReady   control-plane   6m7s   v1.28.1
# 此时集群状态还是 NotReady 状态，因为没有安装网络插件。
```

5、扩容 k8s 集群-添加第一个工作节点

（1）查看加入节点的命令（k8s-master01 节点执行）

```
[root@k8s-master01 ~]# kubeadm token create --print-join-command
kubeadm join 192.168.128.11:6443 --token 9k2fp6.ypfff5f5rj31f4n3 --discovery-token-ca-cert-hash sha256:cfc8252e72a9c2eb60a30f2f0e73e3832e4edd9b77b3181ed14ddab45f2cfd79
```

（2）将 k8s-node01 加入到 k8s 集群中（k8s-node01 节点执行）

```
[root@k8s-node01 ~]# kubeadm join 192.168.128.11:6443 --token
```

```
9k2fp6.ypfff5f5rj31f4n3                  --discovery-token-ca-cert-hash
sha256:cfc8252e72a9c2eb60a30f2f0e73e3832e4edd9b77b3181ed14ddab45f2cfd
79
```

看到如下信息，表示节点已经加入到集群当中了。

```
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.
```

（3）在 k8s-master01 上查看集群节点状况：

```
[root@k8s-master01 ~]# kubectl get nodes
NAME           STATUS     ROLES           AGE    VERSION
k8s-master01   NotReady   control-plane   9m7s   v1.28.1
k8s-node01     NotReady   <none>          12s    v1.28.1
```

6、扩容 k8s 集群-添加第二个工作节点

（1）查看加入节点的命令（k8s-master01 节点执行）

```
[root@k8s-master01 ~]# kubeadm token create --print-join-command
kubeadm join 192.168.128.11:6443 --token 9k2fp6.ypfff5f5rj31f4n3
--discovery-token-ca-cert-hash
sha256:cfc8252e72a9c2eb60a30f2f0e73e3832e4edd9b77b3181ed14ddab45f2cfd
79
```

（2）将 k8s-node02 加入到 k8s 集群中（k8s-node02 节点执行）

```
[root@k8s-node02 ~]# kubeadm join 192.168.128.11:6443 --token
9k2fp6.ypfff5f5rj31f4n3                  --discovery-token-ca-cert-hash
sha256:cfc8252e72a9c2eb60a30f2f0e73e3832e4edd9b77b3181ed14ddab45f2cfd
79
```

看到如下信息，表示节点已经加入到集群当中了。

```
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.
```

（3）在 k8s-master01 上查看集群节点状况：

```
[root@k8s-master01 ~]# kubectl get nodes
NAME           STATUS     ROLES           AGE    VERSION
k8s-master01   NotReady   control-plane   9m48s  v1.28.1
k8s-node01     NotReady   <none>          53s    v1.28.1
k8s-node02     NotReady   <none>          4s     v1.28.1
```

4、对节点设置角色标签

可以看到 k8s-node01、k8s-node02 的 ROLES 角色为空，可以把 k8s-node01 和 k8s-node02 的 ROLES 变成 work，按照如下方法：

语法格式：

```
语法：
    设置 role：
    kubectl label node [NAME] node-role.kubernetes.io/[ROLES]=

    取消 role：
    kubectl label node [NAME] node-role.kubernetes.io/[ROLES]-

    查询 role：
    kubectl get nodes
```

设置标签：

```
    [root@k8s-master01 ~]# kubectl label node k8s-node01 node-role.kubernetes.io/worker=worker
    [root@k8s-master01 ~]# kubectl label node k8s-node02 node-role.kubernetes.io/worker=worker

    [root@k8s-master01 ~]# kubectl get nodes
    NAME           STATUS     ROLES           AGE     VERSION
    k8s-master01   NotReady   control-plane   12m     v1.28.1
    k8s-node01     NotReady   worker          3m51s   v1.28.1
    k8s-node02     NotReady   worker          3m2s    v1.28.1
```

7、安装 kubernetes 网络组件-Calico

（1）安装 helm

K8s 版本支持的各个 helm 版本对照表：

官方网址：https://helm.sh/zh/docs/topics/version_skew/

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

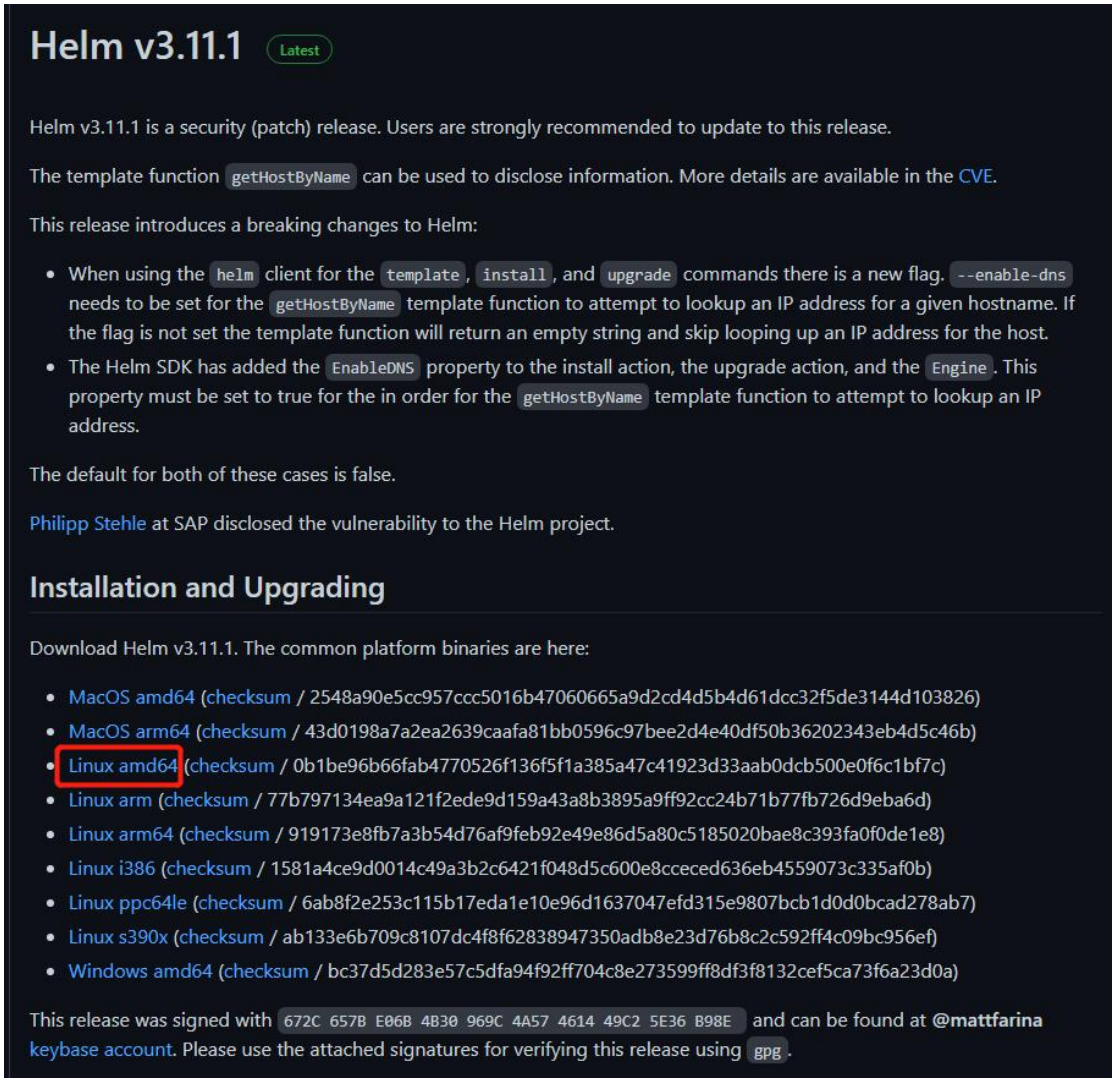
如果您选择了一个Kubernetes版本不支持的Helm，届时会风险。

请参考下表来确定哪个版本的Helm与您的集群兼容。

Helm 版本	支持的 Kubernetes 版本
3.12.x	1.27.x - 1.24.x
3.11.x	1.26.x - 1.23.x
3.10.x	1.25.x - 1.22.x
3.9.x	1.24.x - 1.21.x
3.8.x	1.23.x - 1.20.x
3.7.x	1.22.x - 1.19.x
3.6.x	1.21.x - 1.18.x
3.5.x	1.20.x - 1.17.x

下载 helm 软件包：
下载地址：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。



Helm v3.11.1 Latest

Helm v3.11.1 is a security (patch) release. Users are strongly recommended to update to this release.

The template function `getHostByName` can be used to disclose information. More details are available in the CVE.

This release introduces a breaking changes to Helm:

- When using the `helm` client for the `template`, `install`, and `upgrade` commands there is a new flag. `--enable-dns` needs to be set for the `getHostByName` template function to attempt to lookup an IP address for a given hostname. If the flag is not set the template function will return an empty string and skip looping up an IP address for the host.
- The Helm SDK has added the `EnableDNS` property to the install action, the upgrade action, and the `Engine`. This property must be set to true for the in order for the `getHostByName` template function to attempt to lookup an IP address.

The default for both of these cases is false.

Philipp Stehle at SAP disclosed the vulnerability to the Helm project.

Installation and Upgrading

Download Helm v3.11.1. The common platform binaries are here:

- MacOS amd64 (checksum / 2548a90e5cc957ccc5016b47060665a9d2cd4d5b4d61dcc32f5de3144d103826)
- MacOS arm64 (checksum / 43d0198a7a2ea2639caafa81bb0596c97bee2d4e40df50b36202343eb4d5c46b)
- Linux amd64 (checksum / 0b1be96b66fab4770526f136f5f1a385a47c41923d33aab0dcb500e0f6c1bf7c)
- Linux arm (checksum / 77b797134ea9a121f2ede9d159a43a8b3895a9ff92cc24b71b77fb726d9eba6d)
- Linux arm64 (checksum / 919173e8fb7a3b54d76af9feb92e49e86d5a80c5185020bae8c393fa0f0de1e8)
- Linux i386 (checksum / 1581a4ce9d0014c49a3b2c6421f048d5c600e8cceced636eb4559073c335af0b)
- Linux ppc64le (checksum / 6ab8f2e253c115b17eda1e10e96d1637047efd315e9807bcb1d0d0bcad278ab7)
- Linux s390x (checksum / ab133e6b709c8107dc4f8f62838947350adb8e23d76b8c2c592ff4c09bc956ef)
- Windows amd64 (checksum / bc37d5d283e57c5dfa94f92ff704c8e273599ff8df3f8132cef5ca73f6a23d0a)

This release was signed with `672C 657B E06B 4B30 969C 4A57 4614 49C2 5E36 B98E` and can be found at @mattfarina keybase account. Please use the attached signatures for verifying this release using `gpg`.

上传软件包到 k8s-master01 节点：

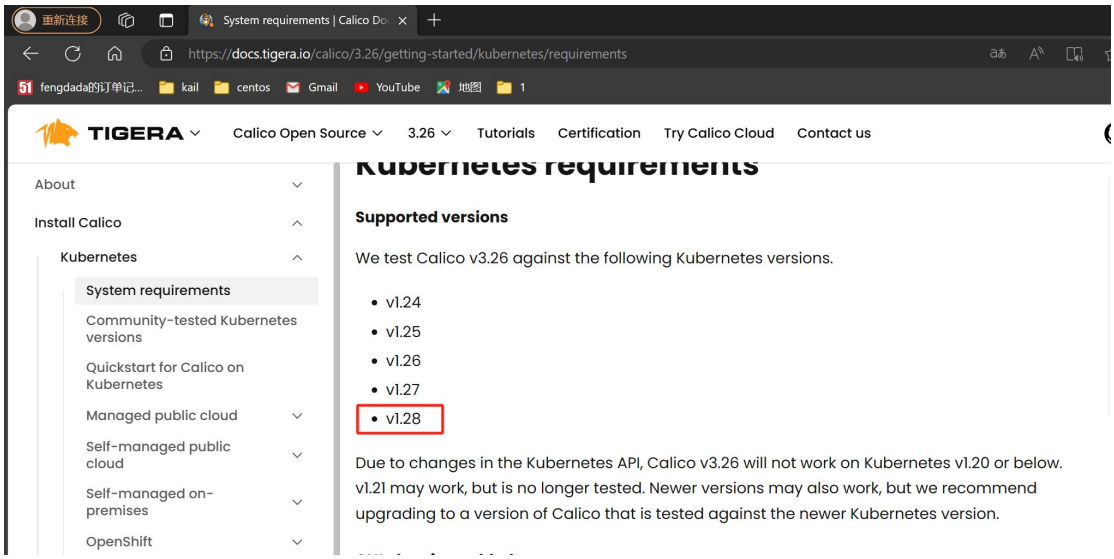
```
[root@k8s-master01 ~]# ll helm-v3.13.3-linux-amd64.tar.gz
[root@k8s-master01 ~]# ll helm-v3.13.3-linux-amd64.tar.gz
-rw-r--r--. 1 root root 16188560 Jan  6  2024 helm-v3.13.3-linux-amd64.tar.gz
[root@k8s-master01 ~]#
```

解压、安装：

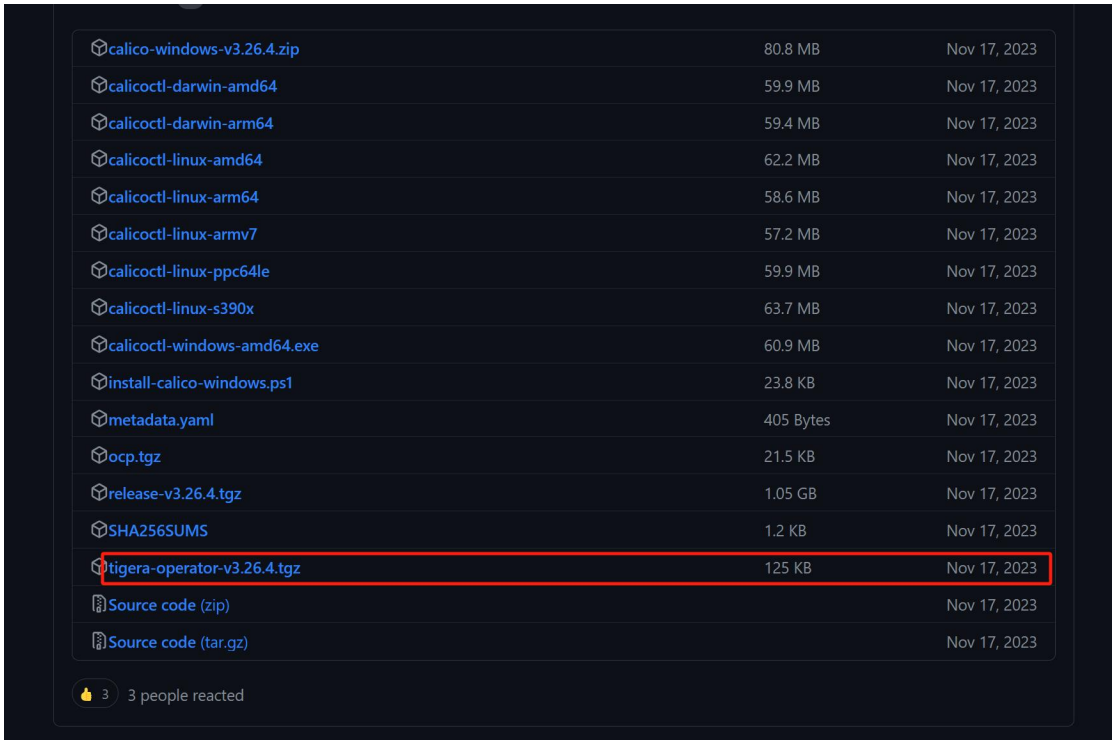
```
[root@k8s-master01 ~]# tar xf helm-v3.13.3-linux-amd64.tar.gz
[root@k8s-master01 ~]# mv linux-amd64/helm /usr/local/bin/
[root@k8s-master01 ~]# helm version
version.BuildInfo{Version:"v3.13.3",
GitCommit:"c8b948945e52abba22ff885446a1486cb5fd3474",
GitTreeState:"clean", GoVersion:"go1.20.11"}
```

（2）查看 calico 组件版本对 kubernetes 集群版本的要求

Calico 官网“版本对应关系”：
https://docs.tigera.io/calico/3.25/getting-started/kubernetes/requirements



（3）安装 calico
Calico github 下载地址：




```
[root@k8s-master01 ~]# wget https://github.com/projectcalico/calico/releases/download/v3.26.4/tigera-operator-v3.26.4.tgz
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
    # 安装 calico
    [root@k8s-master01 ~]# helm install calico tigera-operator-v3.26.4.tgz -n kube-system --create-namespace
    NAME: calico
    LAST DEPLOYED: Sat Jan  6 18:11:34 2024
    NAMESPACE: kube-system
    STATUS: deployed
    REVISION: 1
    TEST SUITE: None

    # 检查
    [root@k8s-master01 ~]# kubectl get pods -A
```

```
[root@k8s-master01 ~]# kubectl get pods -A
NAMESPACE          NAME                                       READY   STATUS    RESTARTS   AGE
calico-apiserver   calico-apiserver-d45f45fbf-jjj5b           1/1     Running   0          6m13s
calico-apiserver   calico-apiserver-d45f45fbf-x5n8g           1/1     Running   0          6m13s
calico-system      calico-kube-controllers-689465784b-4glqt   1/1     Running   0          17m
calico-system      calico-node-46g2v                          1/1     Running   0          17m
calico-system      calico-node-d7r5n                          1/1     Running   0          17m
calico-system      calico-node-z272l                          1/1     Running   0          17m
calico-system      calico-typha-594d6f88d-6d2pn               1/1     Running   0          17m
calico-system      calico-typha-594d6f88d-lpwsd               1/1     Running   0          16m
calico-system      csi-node-driver-74pnp                      2/2     Running   0          17m
calico-system      csi-node-driver-pf7xf                      2/2     Running   0          17m
calico-system      csi-node-driver-zfsj2                      2/2     Running   0          17m
kube-system        coredns-5bbd96d687-8jjf8                   1/1     Running   0          50m
kube-system        coredns-5bbd96d687-fspwl                   1/1     Running   0          50m
kube-system        etcd-k8s-master01                          1/1     Running   0          50m
kube-system        kube-apiserver-k8s-master01                1/1     Running   0          50m
kube-system        kube-controller-manager-k8s-master01       1/1     Running   0          50m
kube-system        kube-proxy-6stgq                           1/1     Running   0          50m
kube-system        kube-proxy-lrxwt                           1/1     Running   0          26m
kube-system        kube-proxy-rqh8r                           1/1     Running   0          24m
kube-system        kube-scheduler-k8s-master01                1/1     Running   0          50m
kube-system        tigera-operator-5d6845b496-74t82           1/1     Running   0          17m
[root@k8s-master01 ~]#
```

```
    [root@k8s-master01 ~]# kubectl get nodes
    NAME           STATUS   ROLES           AGE   VERSION
    k8s-master01   Ready    control-plane   70m   v1.28.1
    k8s-node01     Ready    worker          61m   v1.28.1
    k8s-node02     Ready    worker          60m   v1.28.1
```

8、测试在 k8s 创建 pod 是否可以正常访问网络

在 master 执行启动 pod：

```
    [root@k8s-master01 ~]# kubectl run busybox --image busybox:latest --restart=Never --rm -it busybox -- sh
    If you don't see a command prompt, try pressing enter.

    / # ping www.baidu.com
    PING www.baidu.com (39.156.66.18): 56 data bytes
```

```
    64 bytes from 39.156.66.18: seq=0 ttl=49 time=37.353 ms
    # 通过上面可以看到能访问网络，说明 calico 网络插件已经被正常安装了
```

9、测试 k8s 集群中部署 tomcat 服务

在 k8s-master01 节点执行：

（1）创建 pod：

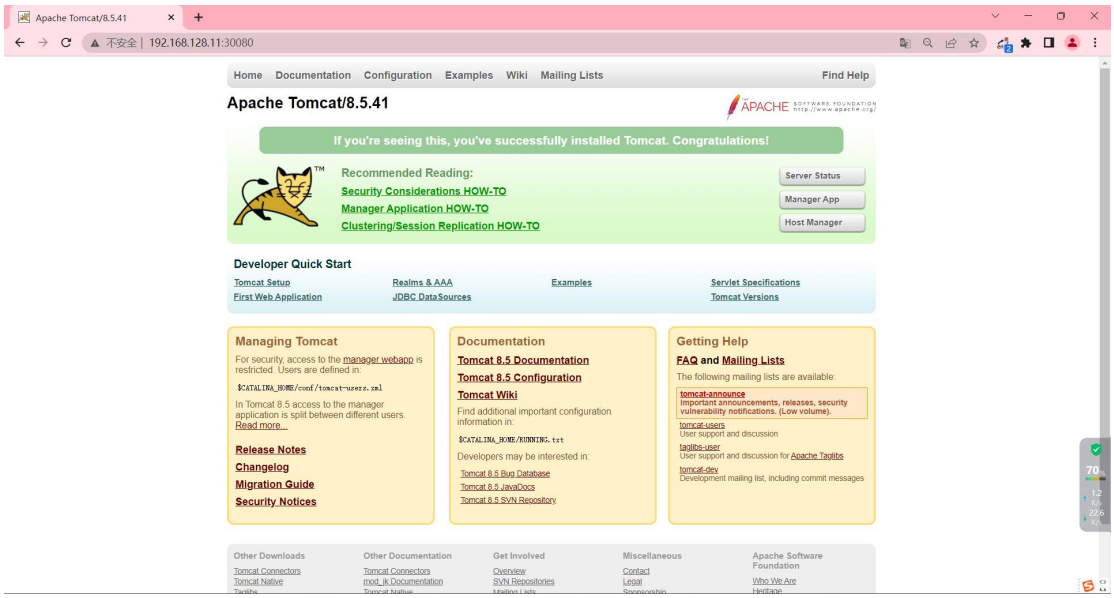
```
    [root@k8s-master01 ~]# vi tomcat.yaml
    apiVersion: v1
    kind: Pod
    metadata:
      name: demo-pod
      namespace: default
      labels:
        app: myapp
        env: dev
    spec:
      containers:
      - name:  tomcat-pod-java
        ports:
        - containerPort: 8080
        image: tomcat:8.5-jre8-alpine
        imagePullPolicy: IfNotPresent

    解释如下：
    apiVersion: v1  #pod 属于 k8s 核心组 v1
    kind: Pod  #创建的是一个 Pod 资源
    metadata:  #元数据
      name: demo-pod  #pod 名字
      namespace: default  #pod 所属的名称空间
      labels:
        app: myapp  #pod 具有的标签
        env: dev      #pod 具有的标签
    spec:
      containers:   #定义一个容器，容器是对象列表，下面可以有多个 name
      - name:  tomcat-pod-java  #容器的名字
        ports:
        - containerPort: 8080
        image: tomcat:8.5-jre8-alpine   #容器使用的镜像
        imagePullPolicy: IfNotPresent

    [root@k8s-master01 ~]# kubectl apply -f tomcat.yaml
```

```
[root@k8s-master01 ~]# kubectl get pods
NAME       READY   STATUS    RESTARTS   AGE
demo-pod   1/1     Running   0          2m5s
```

（2）创建 svc：

```
[root@k8s-master01 ~]# vi tomcat-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: tomcat
spec:
  type: NodePort
  ports:
    - port: 8080
      nodePort: 30080
  selector:
    app: myapp
    env: dev
[root@k8s-master01 ~]# kubectl apply -f tomcat-svc.yaml
[root@k8s-master01 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.10.0.1        <none>        443/TCP          9h
tomcat       NodePort    10.103.122.178   <none>        8080:30080/TCP   5s
```

在浏览器访问 k8s 任意节点的 ip+30080 端口即可请求到服务。



扩展：客户端是怎么访问到 tomcat pod 的？

当客户端浏览器访问 Kubernetes 的 Pod 时，整个流量通常经过以下组件：

1、客户端浏览器向 Kubernetes 集群中的服务发起请求。服务通常是由 Service 定义的，Service 实现了代理和负载均衡功能。

2、经过 Service 代理后，请求被转发到后端 Pod。

3、客户端流量进入 Kubernetes 集群，进入节点的 kube-proxy 组件，kube-proxy 负责为服务维护一个负载均衡的 IP 地址，并为此 IP 地址提供代理。

4、kube-proxy 根据 Service 中定义的负载均衡算法，将客户端流量转发给目标 Pod。在 Kubernetes 集群中，一种常用的负载均衡算法是 round-robin，即每个 Pod 轮流接收客户端请求。

5、Pod 中的应用程序处理客户端请求并返回响应，响应流量沿着相反的路径通过 kube-proxy 和 Service 组件返回给客户端浏览器。

总的来说，整个流量的路径可以被视为：

浏览器 -> Service（kube-proxy） -> Pod -> Service（kube-proxy） -> 浏览器

其中，kube-proxy 和 Service 组件是 Kubernetes 集群中的核心组件，用于负责路由和负载均衡流量。

10、测试 coredns 是否正常

```
[root@k8s-master01 ~]# kubectl run dig --rm -it --image=docker.io/azukiapp/dig /bin/sh
/ # nslookup kubernetes.default.svc.cluster.local
Server:    10.10.0.10
Address:  10.10.0.10#53

Name:  kubernetes.default.svc.cluster.local
Address: 10.10.0.1

/ # nslookup www.baidu.com
Server:    10.10.0.10
Address:  10.10.0.10#53

Non-authoritative answer:
www.baidu.com canonical name = www.a.shifen.com.
Name:  www.a.shifen.com
Address: 220.181.38.149
Name:  www.a.shifen.com
Address: 220.181.38.150
```

10.10.0.10 就是我们 coreDNS 的 clusterIP，说明 coreDNS 配置好了。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

解析内部 Service 的名称，是通过 coreDNS 去解析的。

注意：busybox 自带的 nslookup 实现的不是很完全，会导致测试 DNS 失败。所以这里使用的是带 nslookup 的 alphine。

11、部署 Docker 服务（非必须）

Kubernetes 1.24 之后的版本已经不支持 docker 了，但是还要把 docker 安装在 k8s 节点上，主要是为了用 docker build 基于 dockerfile 做镜像，docker 和 containerd 不冲突，不会互相影响。

安装、配置、启动 docker 容器：

128.11、128.21、128.22 节点执行如下：

```
[root@k8s-master01 ~]# yum -y install docker-ce
[root@k8s-master01 ~]# cd /etc/docker/
[root@k8s-master01 docker]# vi /etc/docker/daemon.json
{
  "registry-mirrors": ["https://q2gr04ke.mirror.aliyuncs.com"],
  "exec-opts": ["native.cgroupdriver=systemd"]
}
[root@k8s-master01 ~]# systemctl restart docker && systemctl enable docker && systemctl status docker
```