# kubernetes 前置基础知识

## 一、docker-ce

`docker-ce` 是 Docker Community Edition（社区版 Docker）的简称，它是一个开源的容器化平台，用于开发、部署和运行容器化应用。安装 `docker-ce` 后，你可以在你的系统中使用 Docker 提供的核心功能。

```yaml
# 安装 docker-ce
yum -y install docker-ce
```

`/etc/docker/daemon.json` 是 Docker 的守护进程（`dockerd`）的配置文件，用于管理 Docker 守护进程的行为。通过这个文件，你可以以 JSON 格式指定各种 Docker 的全局配置项，而无需在每次启动 `dockerd` 时手动添加命令行参数。

每次修改 `/etc/docker/daemon.json` 后需要：

```yaml
systemctl daemon-reload && systemctl restart docker
```

将 docker 加入开机自启：

```yaml
systemctl enable docker
```

## 二、cri-docker

**CRI-Dockerd** 是一种兼容 Kubernetes 的容器运行时接口（CRI）插件，用于将 Docker 作为 Kubernetes 的容器运行时继续使用。

**背景**

1. **Kubernetes 和 Docker 的分离：**

- 在 Kubernetes 1.20 版本中，Kubernetes 社区宣布将逐步淘汰对 Docker 的直接支持（即 Docker Shim）。
- 从 Kubernetes 1.24 开始，Docker Shim 被正式移除，这意味着 Kubernetes 不再直接支持 Docker 作为容器运行时。

2. **容器运行时接口 (CRI):**
   - Kubernetes 通过 **CRI 与不同的容器运行时（如 containerd、CRI-O）通信，而 Docker 本身不支持 CRI**。
   - Docker Shim 是一个桥接层，用于将 Docker 转换为 CRI 兼容的运行时。
3. **为什么需要 CRI-Dockerd:**
   - Kubernetes 用户依然希望继续使用 Docker，因为它具有丰富的工具链和生态支持。
   - CRI-Dockerd 作为一个独立的插件，填补了 **Docker 与 Kubernetes 之间**的空白，**使 Docker 能通过 CRI 与 Kubernetes 集成**。

---

## CRI-Dockerd 的作用

CRI-Dockerd 是一个将 Docker 引擎封装为 CRI 兼容运行时的适配器，允许 Kubernetes 使用 Docker 作为其容器运行时。

它主要负责以下任务：

1. **实现 CRI 规范：**
   - **将 Kubernetes 的 CRI 调用（如创建容器、拉取镜像）转换为 Docker API 调用**。
2. **兼容 Kubernetes:**
   - 保留 Kubernetes 和 Docker 的无缝集成体验，使现有用户无需完全迁移到其他容器运行时（如 containerd）。
3. **支持容器管理：**
   - 支持容器的启动、停止、重启、删除等基本操作。
   - 管理容器镜像的拉取和缓存。

---

## CRI-Dockerd 的主要组成

- **CRI 服务：**
  - 一个守护进程，负责监听 Kubernetes 发出的 CRI 请求。
- **Docker API 调用：**
  - 调用 Docker 的 REST API 来执行容器操作。

## 1. 安装 cri-docker

YAML

```yaml
# 安装 cri-docker
wget https://github.com/Mirantis/cri-dockerd/releases/download/v0.3.9/cri-dockerd-0.3.9.amd64.tgz
tar -xf cri-dockerd-0.3.9.amd64.tgz
cp cri-dockerd/cri-dockerd /usr/bin/
chmod +x /usr/bin/cri-dockerd
```

## 2. 配置 cri-docker 服务

YAML

```yaml
# 配置 cri-docker 服务
cat <<"EOF" > /usr/lib/systemd/system/cri-docker.service
[Unit]
Description=CRI Interface for Docker Application Container Engine
Documentation=https://docs.mirantis.com
After=network-online.target firewalld.service docker.service
Wants=network-online.target
Requires=cri-docker.socket
[Service]
Type=notify
ExecStart=/usr/bin/cri-dockerd --network-plugin=cni --pod-infra-container-image=registry.aliyuncs.com/google_containers/pause:3.8
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always
StartLimitBurst=3
StartLimitInterval=60s
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity
TasksMax=infinity
Delegate=yes
KillMode=process
[Install]
WantedBy=multi-user.target
EOF
```

`/usr/lib/systemd/system/cri-docker.service` 是 **Systemd 服务单元文件**，用于管理和控制 **CRI-Dockerd 服务**（即 CRI-Docker Daemon）。

**`/usr/lib/systemd/system/cri-docker.service` 的功能与作用：**

1. **定义服务启动和管理方式：**
   - 包含 CRI-Dockerd 的启动命令、依赖项、服务运行条件等信息。
   - Systemd 使用该文件启动、停止或重新加载 CRI-Dockerd 服务。
2. **服务运行配置：**
   - 配置服务运行所需的环境、工作目录、用户权限等。
3. **与其他服务的依赖：**
   - 指定 CRI-Dockerd 与 Docker、网络等其他服务的启动顺序或依赖关系。

### 3. 添加 cri-docker 套接字

YAML

```
# 添加 cri-docker 套接字
cat <<"EOF" > /usr/lib/systemd/system/cri-docker.socket
[Unit]
Description=CRI Docker Socket for the API
PartOf=cri-docker.service
[Socket]
ListenStream=%t/cri-dockerd.sock
SocketMode=0660
SocketUser=root
SocketGroup=docker
[Install]
WantedBy=sockets.target
EOF
```

`/usr/lib/systemd/system/cri-docker.socket` 是 **Systemd Socket 单元文件**，用于定义和管理 **CRI-Dockerd 的套接字监听服务**。此文件的作用是提供一个通信接口，通常是一个 Unix 套接字，用于 Kubernetes 和 CRI-Dockerd 的交互。

1. **定义通信套接字：**
   - 指定 CRI-Dockerd 使用的监听地址（如 Unix 套接字或 TCP 端口）。
   - 套接字是 Kubernetes 通过 CRI（容器运行时接口）与 Docker 交互的桥梁。
2. **Socket 激活：**
   - **当有客户端（如 Kubernetes）尝试连接套接字时，Systemd 会自动启动关联的服务单元（`cri-docker.service`）。**

3. **与服务的解耦：**
    - 将**套接字的监听逻辑从服务中分离**，可以优化资源使用，使服务仅在需要时启动。

### 4. 启动 cri-docker 对应服务

YAML

```yaml
# 启动 cri-docker 对应服务
systemctl daemon-reload
systemctl enable cri-docker
systemctl start cri-docker
systemctl is-active cri-docker
```

# 三、kubeadm

`kubeadm` 是 Kubernetes 官方提供的一个命令行工具，用于快速、简洁地**初始化 Kubernetes 集群**。它旨在简化 Kubernetes 集群的部署和管理过程，特别是控制平面组件的安装和配置。

## 主要功能

1. **集群初始化：**
    - 使用 `kubeadm init` 命令初始化 Kubernetes 集群的控制平面，包括：
        - 安装和配置控制平面组件（如 `kube-apiserver`、`kube-controller-manager`、`kube-scheduler`）。
        - 生成必要的认证文件和证书。
        - 配置 `etcd` 数据存储。
        - 启动网络插件所需的基本配置。
2. **节点加入：**
    - 使用 `kubeadm join` 命令，将工作节点加入到已经初始化的集群中。
    - 通过 `--token` 和控制平面地址进行认证和连接。
3. **证书管理：**
    - 提供证书检查和自动续订功能，确保集群中所需的证书不过期。
4. **集群配置文件：**
    - 支持通过 YAML 配置文件定义集群初始化和管理参数，提供更高的可定制性。
5. **升级管理：**
    - 使用 `kubeadm upgrade` 命令轻松升级 Kubernetes 集群到指定版本。

6. **重置集群：**
    - 使用 `kubeadm reset` 命令清理已初始化的节点，恢复到未初始化状态。

---

## 工作原理

`kubeadm` 本身不是一个长期运行的服务，它只是一个一次性执行的工具，用来部署和配置 Kubernetes 的组件。`kubeadm` 的主要职责是：

- 安装和配置 Kubernetes 控制平面组件。
- 创建和分发证书及密钥。
- 配置和启动必要的资源（如网络插件）。

**集群的运行由 Kubernetes 的核心组件（如 `kubelet` 和控制平面服务）完成，`kubeadm` 不参与后续的日常操作。**

YAML

```yaml
# 添加 kubeadm yum 源
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.29/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.29/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF

# 安装 kubeadm 1.29 版本
yum install -y kubelet-1.29.0 kubectl-1.29.0 kubeadm-1.29.0
systemctl enable kubelet.service
# 初始化主节点
kubeadm init --apiserver-advertise-address=192.168.66.11 --image-repository registry.aliyuncs.com/google_containers --kubernetes-version 1.29.2 --service-cidr=10.10.0.0/12 --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all --cri-socket unix:///var/run/cri-dockerd.sock
# work token 过期后，重新申请
kubeadm token create --print-join-command
# worker 加入
kubeadm join 192.168.10.11:6443 --token a6xh07.yg9wh2vru2grluwb
    --discovery-token-ca-cert-hash sha256:7cd8499abae48c8403800152cc0f655ac704ea00ae30a549acd9bbac7b26dca4 --cri-socket unix:///var/run/cri-dockerd.sock
```

# 四、calico

**Calico** 是一个高性能、可扩展的网络插件，用于 **Kubernetes 集群中的容器网络管理**。它提供了网络连接（Networking）和网络安全（Network Policy）的功能，支持多种网络模式，是 Kubernetes 中常用的 CNI（Container Network Interface）插件之一，支持大规模 Kubernetes 集群的容器网络部署。

## 核心功能

1. **容器网络连接（Networking）：**
   - Calico 为 Kubernetes 集群中的 **Pod 提供 L3 网络通信**。Kubernetes 集群中的 Pod 需要能够相互通信。Calico 提供了高效的网络连接，确保**所有运行的容器（Pod）能够通**

**过分配的 IP 地址直接通信**。实现了容器网络的互联互通，使 Kubernetes 集群中的服务和应用可以正常运行

- 支持多种网络模式（纯 L3 模式、BGP、VXLAN、IPIP 等），可在不同的基础设施上灵活部署。
- 提供高效的网络路由，能够直接使用底层物理网络，无需复杂的封装。

2. **网络策略管理（Network Policy）：**
   - 支持 Kubernetes 的 `NetworkPolicy` 功能，并扩展了更多高级网络策略。
   - 可以**限制 Pod 间的通信，或者限制 Pod 与外部网络的连接，提供细粒度的安全控制**。例如可以限制某些 Pod 只能访问特定的服务，或阻止不必要的入站流量。
3. **跨集群通信：**
   - Calico 支持通过 BGP 实现集群间的网络互联，方便跨集群通信和容器化应用的高可用部署。
4. **灵活的 IP 地址管理（IPAM）：**
   - 提供高效的 IP 地址管理功能，支持动态分配和静态分配，优化 Pod IP 的使用效率。
5. **支持多种环境：**
   - 兼容 Kubernetes、OpenShift、Docker Swarm 等容器编排平台。
   - 支持多云和混合云部署，以及裸机环境。

---

### 工作原理

1. **纯 L3 网络：**
   - Calico 通过使用 BGP 协议在各节点之间动态分发路由信息，每个 Pod 的 IP 地址可以直接路由到其他节点，而无需封装或隧道。
2. **IPIP 或 VXLAN：**
   - 如果底层网络不支持直接路由，Calico 会使用 IPIP 或 VXLAN 隧道封装以实现节点间的 Pod 通信。
3. **网络策略：**
   - Calico 在每个节点上运行一个 `calico-node` 守护进程，通过内核中的 `iptables` 或 eBPF 管理网络策略规则。

## 五、iKuai

iKuai 是一款基于 Linux 操作系统的企业级路由器 / 防火墙操作系统，广泛用于网络管理、路由控制、VPN 构建等场景。它的主要作用和功能如下：

### 1. 路由器和防火墙功能

- **作用**：iKuai 可以作为一个企业级的路由器来管理和转发网络流量，支持多种路由协议（如静态路由、动态路由等）。它还具有强大的防火墙功能，能够控制进出网络的流量，保护内网免受外部攻击。
- **应用场景**：用于中小型企业的网络边界，管理流量、设置安全策略。

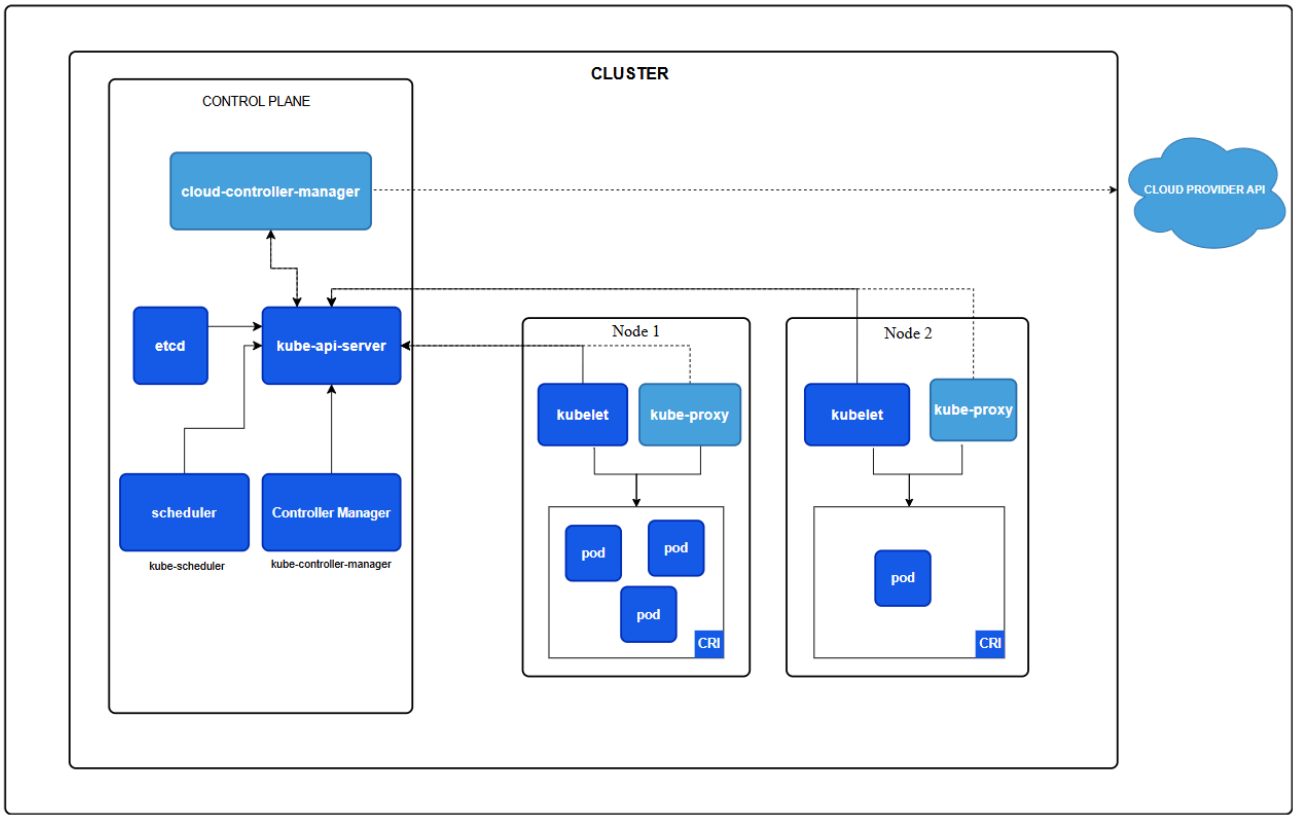
### 2. 网络地址转换 (NAT)

- **作用**：iKuai 支持 NAT（Network Address Translation），可以实现内部私有网络地址与公共网络地址之间的转换，通常用于家庭或企业中将多个设备通过一个公共 IP 地址连接到互联网。
- **应用场景**：例如家里或公司中所有的设备都通过一个公共 IP 地址访问互联网，而每个设备都有私有的局域网 IP 地址。

### 3. VPN 支持

- **作用**：iKuai 支持多种 VPN 技术（如 PPTP、L2TP、OpenVPN 等），允许远程用户通过加密的隧道连接到公司内部网络。这个功能非常适合远程办公或分支机构之间的连接。
- **应用场景**：企业需要为远程员工提供安全的远程访问时，iKuai 可以作为 VPN 服务器使用。

## 六、kubernetes 架构




🔗 Kubernetes 架构

**Kubelet** 是 Kubernetes 中负责节点管理和容器运行的核心组件之一。它运行在每个集群节点上（包括主节点和工作节点），负责管理节点上的容器生命周期、执行任务、报告节点状态、运行应用程序以及与 Kubernetes 控制平面的其他组件（如 API server）进行通信。

## Kubelet 的主要作用：

1. **确保容器的运行和健康**：
   - Kubelet 负责确保与 Pod 相关的容器运行在节点上，并且处于正确的状态。它会根据 PodSpec 来拉取镜像、启动容器、执行健康检查（如就绪探针、存活探针等）。
   - 它还会监控容器的运行状态，并在容器崩溃或失败时自动重新启动。
2. **与 Kubernetes API Server 通信**：
   - Kubelet 会定期与 API Server 通信，报告节点和容器的状态，包括容器是否正常运行，Pod 是否需要调度到该节点等。
   - 它通过 API Server 获取待执行的任务（PodSpec），然后根据任务要求启动或停止容器。
3. **容器生命周期管理**：
   - Kubelet 管理节点上的容器生命周期。它会根据 PodSpec 配置文件启动、停止和删除容器。
   - 在容器启动时，Kubelet 会确保其按需求启动，如果配置了健康检查（例如：就绪探针和存活探针），Kubelet 会根据探针的反馈调整容器的运行状态。

4. **健康检查**：
    - Kubelet 会定期执行容器的健康检查（如就绪探针和存活探针），并根据检查结果决定是否重启容器或将容器标记为不可用。
    - 如果容器或 Pod 未通过健康检查，Kubelet 会通知 API Server 更新状态，并执行必要的操作，如重新调度或重启容器。
5. **处理和报告资源需求**：
    - Kubelet 管理节点上的资源分配（如 CPU 和内存），确保容器按照资源请求进行调度。
    - 它会监控节点上的资源使用情况，并通过 API Server 将节点的资源可用性报告给集群调度器，以便调度器能够做出适当的调度决策。
6. **日志和监控**：
    - Kubelet 会收集容器和节点的日志，并将其发送到监控系统或日志系统中，以便运维人员可以查看和分析容器的运行状态和性能。
7. **与 cgroup 和 namespace 协同工作**：
    - Kubelet 使用 Linux cgroups 来限制和管理容器资源的使用（如 CPU 和内存）。
    - 它也使用 Linux namespaces 来确保容器的隔离性，并确保每个容器在自己的独立环境中运行。
8. **Pod 和容器的调度**：
    - Kubelet 与 Kubernetes 调度器（Scheduler）紧密合作，根据 Pod 的资源需求和节点资源的可用性，确保 Pod 被调度到合适的节点，并且容器能够正常启动和运行。
9. **Volume 管理**：
    - Kubelet 会挂载和管理数据卷（Volumes），根据 Pod 的配置自动将持久化存储挂载到容器中。

**总结**

Kubelet 是 Kubernetes 集群中非常重要的一个组件，负责管理和维护集群节点上的容器及 Pod 的生命周期。它确保容器按照定义的配置和需求启动、运行、停止，并定期报告容器和节点的健康状态。它还管理容器的资源分配、日志、健康检查等，使得 Kubernetes 集群能够有效地运行并确保容器的高可用性。