Steps for init kubernetes

At 78 (Where we have internet)

---

pull and push img to registry commands

```
docker pull registry.k8s.io/kube-apiserver:v1.29.0
docker pull registry.k8s.io/kube-controller-manager:v1.29.0
docker pull registry.k8s.io/kube-scheduler:v1.29.0
docker pull registry.k8s.io/kube-proxy:v1.29.0
docker pull registry.k8s.io/etcd:3.5.10-0
docker pull registry.k8s.io/coredns/coredns:v1.11.1
docker pull registry.k8s.io/pause:3.9

docker tag registry.k8s.io/kube-apiserver:v1.29.0 192.168.253.78:5000/k8s/kube-apiserve
docker tag registry.k8s.io/kube-controller-manager:v1.29.0 192.168.253.78:5000/k8s/kube
docker tag registry.k8s.io/kube-scheduler:v1.29.0 192.168.253.78:5000/k8s/kube-schedule
docker tag registry.k8s.io/kube-proxy:v1.29.0 192.168.253.78:5000/k8s/kube-proxy:v1.29.
docker tag registry.k8s.io/etcd:3.5.10-0 192.168.253.78:5000/k8s/etcd:3.5.10-0
docker tag registry.k8s.io/coredns/coredns:v1.11.1 192.168.253.78:5000/k8s/coredns/core
docker tag registry.k8s.io/pause:3.9 192.168.253.78:5000/k8s/pause:3.9

docker push 192.168.253.78:5000/k8s/kube-apiserver:v1.29.0
docker push 192.168.253.78:5000/k8s/kube-controller-manager:v1.29.0
docker push 192.168.253.78:5000/k8s/kube-scheduler:v1.29.0
docker push 192.168.253.78:5000/k8s/kube-proxy:v1.29.0
docker push 192.168.253.78:5000/k8s/etcd:3.5.10-0
docker push 192.168.253.78:5000/k8s/coredns/coredns:v1.11.1
docker push 192.168.253.78:5000/k8s/pause:3.9
```

---

at where you want to setup that kubernetes

---

checking certificates

`curl -Iv https://download.docker.com` `wget https://download.docker.com` <- just for checking only curl has problem or not for updating certificates (when you don't have any restriction this will work smoothly)

```
sudo apt install --reinstall ca-certificates
sudo update-ca-certificates
```

checking certificates permission `ls -l /etc/ssl/certs/ca-certificates.crt`

get certificates by openssl <- for validation

```
openssl s_client -connect download.docker.com:443 -showcerts
```

Now this is the how I updates certificates

```
echo | openssl s_client -connect download.docker.com:443 -showcerts 2>/dev/null | awk '
sudo echo | openssl s_client -connect download.docker.com:443 -showcerts 2>/dev/null |
echo | openssl s_client -connect download.docker.com:443 -showcerts 2>/dev/null | sudo
sudo cp fortinet-ca.crt /usr/local/share/ca-certificates/fortinet.crt
sudo update-ca-certificates
sudo apt update
```

for run kubernetes we have to turn off swap memory `sudo swapoff -a`

we also need overlay and br_netfilter

```
cat <
```

Full Expiation:

Here’s a detailed breakdown and explanation of the following command:

🧩 Step-by-Step Breakdown

1. `cat <<EOF`

- Starts a **here-document** (heredoc).
- Everything typed until the line with `EOF` is treated as input to `cat`.

2. Content of the heredoc:

```
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
```

These are **kernel parameter settings** that will be written to the file.

3. `| sudo tee /etc/sysctl.d/k8s.conf`

- `tee` writes the heredoc content to `/etc/sysctl.d/k8s.conf`.
- `sudo` ensures root permission is used for writing to a system directory.
- This avoids permission errors that would occur if you tried `>` `/etc/sysctl.d/k8s.conf` without `sudo`.

4. Final `EOF`

- Marks the end of heredoc input.

📄 What the File Will Contain

```
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
```

🔍 Purpose of Each sysctl Setting

1. `net.bridge.bridge-nf-call-iptables = 1`
   - Ensures bridged traffic (like Docker/k8s pods) is **processed by iptables rules**.
   - Required for Kubernetes network policies and traffic control.
2. `net.ipv4.ip_forward = 1`
   - Enables **IPv4 forwarding**, allowing the system to forward packets from one interface to another.
   - Essential for pod-to-pod communication and cluster routing.
3. `net.bridge.bridge-nf-call-ip6tables = 1`
   - Same as the first one but applies to **IPv6** traffic.
   - Ensures ip6tables (IPv6 firewall) handles bridged traffic.

✅ Why This is Done

- These settings are required for **Kubernetes networking to function properly**.
- Writing them to `/etc/sysctl.d/k8s.conf` ensures they are **persisted across reboots**.
- These will be loaded on boot by `systemd-sysctl`.

🔄 To Apply the Settings Immediately

After creating this file, run:

```
sudo sysctl --system
```

This applies all settings from files in `/etc/sysctl.d/`, `/run/sysctl.d/`, and `/usr/lib/sysctl.d/`.

Great question! Let's explain both:

🛠️ What is `sysctl`?

`sysctl` is a Linux command-line tool used to **view and modify kernel parameters** at runtime.

- These parameters control **core behaviors** of the Linux kernel, like networking, memory, and security.
- The values are stored under the virtual filesystem path: `/proc/sys/`.

For example:

- `net.ipv4.ip_forward` is stored at `/proc/sys/net/ipv4/ip_forward`
- You can read it with: `cat /proc/sys/net/ipv4/ip_forward`
- Or set it temporarily with: `sudo sysctl net.ipv4.ip_forward=1`

📘 What is the `--system` Flag?

```
sudo sysctl --system
```

This command **reloads all system-wide kernel parameters** from the following configuration files:

1. `/etc/sysctl.d/*.conf` – Custom user-defined configs.
2. `/run/sysctl.d/*.conf` – Temporary configs applied at runtime.
3. `/usr/lib/sysctl.d/*.conf` – Defaults shipped by packages or distributions.
4. `/etc/sysctl.conf` – Traditional single-file config (older method).

✅ Why use `--system`?

It ensures that **all configuration files are applied** in the correct order after boot, or after creating/updating `.conf` files.

🧪 Example Use Case

You added:

```
net.ipv4.ip_forward = 1
```

to `/etc/sysctl.d/k8s.conf`.

To apply it immediately **without rebooting**, you run:

```
sudo sysctl --system
```

This reads all the `.conf` files and applies their settings to the running system.

---

end of explaining

---

than `sudo mkdir -p /etc/apt/keyrings`

This is the part of the official Kubernetes installation process for Debian/Ubuntu:

- Adds the trusted source for downloading Kubernetes packages.
- Ensures package integrity via GPG key verification.
- Lets you install components like kubelet, kubeadm, and kubectl using apt.

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key |   gpg --dearmor -o
sudo curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key |   gpg --dearm
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key |   sudo gpg --dearm
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://pkgs.k8s
```

so after that I got this error

```
:~$ sudo apt-get update
Err:1 https://download.docker.com/linux/ubuntu noble InRelease
  403  Forbidden [IP: 13.225.5.103 443]
Hit:2 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:4 http://in.archive.ubuntu.com/ubuntu noble InRelease
Get:3 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.2
Get:5 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.2
Hit:6 http://in.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:7 http://in.archive.ubuntu.com/ubuntu noble-backports InRelease
Reading package lists... Done
E: Failed to fetch https://download.docker.com/linux/ubuntu/dists/noble/InRelease  403
E: The repository 'https://download.docker.com/linux/ubuntu noble InRelease' is no long
N: Updating from such a repository can't be done securely, and is therefore disabled by
N: See apt-secure(8) manpage for repository creation and user configuration details.
```

So we have two option:

1. Use jammy (Ubuntu 22.04) repo temporarily <= this is best solution so `jammy` I choose you!!! ~~use thunderbolt attack~~
2. Temporarily disable the Docker repo

Solution ~~high level~~

You're seeing this error because **Docker does not yet officially support Ubuntu 24.04 "Noble Numbat"** in its APT repository.

Use `jammy` (Ubuntu 22.04) repo temporarily

You can trick APT into using the **Ubuntu 22.04 (Jammy)** version of Docker packages, which often works fine on 24.04:

1. Edit the Docker APT source:

   ```
   sudo nano /etc/apt/sources.list.d/docker.list
   ```

2. Change the word `noble` to `jammy`:

   From:

```
deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.co
```

To:

```
deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.co
```

3. Save and exit (`Ctrl+S` , `Enter` , `Ctrl+X`).
4. Then:

   ```
   sudo apt-get update
   ```

Than installing our main characters....

I hope someone read this it's took lots of effort and research by vinit yap me so you thinking why 1.29 we have 1.32 also and it's latest stable right??? Sometimes you have to find answer yourself go and find I do many trials so IK why !!!

```
sudo apt-get install -y kubelet=1.29.2-1.1 kubeadm=1.29.2-1.1 kubectl=1.29.2-1.1
sudo apt-mark hold kubelet kubeadm kubectl
```

Now comes to main Part

Install cri-dockerd

just download bullseye version of cri for linux and put in drive mount that I hope until you have some hope left Than install it with dpkg

`sudo dpkg -i cri-dockerd_0.3.16.3-0.debian-bullseye_amd64.deb`

Than restart this services

```
sudo systemctl daemon-reexec
sudo systemctl daemon-reload
```

enable services

```
sudo systemctl enable cri-docker.service
sudo systemctl enable --now cri-docker.socket
sudo systemctl status cri-docker.service
```

Now we pull images that we pushed at our local registry

```
docker pull registry.k8s.io/kube-apiserver:v1.29.0
docker pull registry.k8s.io/kube-controller-manager:v1.29.0
docker pull registry.k8s.io/kube-scheduler:v1.29.0
docker pull registry.k8s.io/kube-proxy:v1.29.0
```

```
docker pull registry.k8s.io/etcd:3.5.10-0
docker pull registry.k8s.io/coredns/coredns:v1.11.1
docker pull registry.k8s.io/pause:3.9
```

✅ Configure containerd to Trust the Internal Registry

Edit the containerd config:

```
sudo mkdir -p /etc/containerd
containerd config default > /etc/containerd/config.toml
```

Then open it: `sudo nano /etc/containerd/config.toml` 🔍 Modify/Add This Section:

`Look for the [plugins."io.containerd.grpc.v1.cri".registry] block.`

You want something like this:

```
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
    endpoint = ["http://192.168.253.78:5000"]

[plugins."io.containerd.grpc.v1.cri".registry.mirrors."registry.k8s.io"]
    endpoint = ["http://192.168.253.78:5000"]
```

So if you wondered how toml file look like???

```
:~$ sudo nano /etc/containerd/config.toml
[sudo] password for indadmin:
:~$ sudo cat /etc/containerd/config.toml
disabled_plugins = []
imports = []
oom_score = 0
plugin_dir = ""
required_plugins = []
root = "/var/lib/containerd"
state = "/run/containerd"
temp = ""
version = 2

[cgroup]
  path = ""

[debug]
  address = ""
  format = ""
  gid = 0
  level = ""
  uid = 0

[grpc]
  address = "/run/containerd/containerd.sock"
  gid = 0
  max_recv_message_size = 16777216
  max_send_message_size = 16777216
  tcp_address = ""
  tcp_tls_ca = ""
  tcp_tls_cert = ""
  tcp_tls_key = ""
  uid = 0

[metrics]
```

```
  address = ""
  grpc_histogram = false

[plugins]

  [plugins."io.containerd.gc.v1.scheduler"]
    deletion_threshold = 0
    mutation_threshold = 100
    pause_threshold = 0.02
    schedule_delay = "0s"
    startup_delay = "100ms"

  [plugins."io.containerd.grpc.v1.cri"]
    cdi_spec_dirs = ["/etc/cdi", "/var/run/cdi"]
    device_ownership_from_security_context = false
    disable_apparmor = false
    disable_cgroup = false
    disable_hugetlb_controller = true
    disable_proc_mount = false
    disable_tcp_service = true
    drain_exec_sync_io_timeout = "0s"
    enable_cdi = false
    enable_selinux = false
    enable_tls_streaming = false
    enable_unprivileged_icmp = false
    enable_unprivileged_ports = false
    ignore_deprecation_warnings = []
    ignore_image_defined_volumes = false
    image_pull_progress_timeout = "5m0s"
    image_pull_with_sync_fs = false
    max_concurrent_downloads = 3
    max_container_log_line_size = 16384
    netns_mounts_under_state_dir = false
    restrict_oom_score_adj = false
    sandbox_image = "registry.k8s.io/pause:3.8"
    selinux_category_range = 1024
    stats_collect_period = 10
    stream_idle_timeout = "4h0m0s"
    stream_server_address = "127.0.0.1"
    stream_server_port = "0"
    systemd_cgroup = false
    tolerate_missing_hugetlb_controller = true
    unset_seccomp_profile = ""

    [plugins."io.containerd.grpc.v1.cri".cni]
      bin_dir = "/opt/cni/bin"
      conf_dir = "/etc/cni/net.d"
      conf_template = ""
      ip_pref = ""
      max_conf_num = 1
      setup_serially = false

    [plugins."io.containerd.grpc.v1.cri".containerd]
      default_runtime_name = "runc"
      disable_snapshot_annotations = true
      discard_unpacked_layers = false
      ignore_blockio_not_enabled_errors = false
      ignore_rdt_not_enabled_errors = false
      no_pivot = false
      snapshotter = "overlayfs"

      [plugins."io.containerd.grpc.v1.cri".containerd.default_runtime]
        base_runtime_spec = ""
        cni_conf_dir = ""
        cni_max_conf_num = 0
        container_annotations = []
        pod_annotations = []
        privileged_without_host_devices = false
        privileged_without_host_devices_all_devices_allowed = false
        runtime_engine = ""
        runtime_path = ""
        runtime_root = ""
        runtime_type = ""
        sandbox_mode = ""
        snapshotter = ""

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

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

        [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
          base_runtime_spec = ""
          cni_conf_dir = ""
          cni_max_conf_num = 0
          container_annotations = []
          pod_annotations = []
          privileged_without_host_devices = false
          privileged_without_host_devices_all_devices_allowed = false
          runtime_engine = ""
          runtime_path = ""
          runtime_root = ""
          runtime_type = "io.containerd.runc.v2"
          sandbox_mode = "podsandbox"
          snapshotter = ""

          [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
            BinaryName = ""
            CriuImagePath = ""
            CriuPath = ""
            CriuWorkPath = ""
            IoGid = 0
            IoUid = 0
            NoNewKeyring = false
            NoPivotRoot = false
            Root = ""
            ShimCgroup = ""
            SystemdCgroup = false

      [plugins."io.containerd.grpc.v1.cri".containerd.untrusted_workload_runtime]
        base_runtime_spec = ""
        cni_conf_dir = ""
        cni_max_conf_num = 0
        container_annotations = []
        pod_annotations = []
        privileged_without_host_devices = false
        privileged_without_host_devices_all_devices_allowed = false
        runtime_engine = ""
        runtime_path = ""
        runtime_root = ""
        runtime_type = ""
        sandbox_mode = ""
        snapshotter = ""

        [plugins."io.containerd.grpc.v1.cri".containerd.untrusted_workload_runtime.opti

    [plugins."io.containerd.grpc.v1.cri".image_decryption]
      key_model = "node"

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

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

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

      [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
          endpoint = ["http://192.168.253.78:5000"]

        [plugins."io.containerd.grpc.v1.cri".registry.mirrors."registry.k8s.io"]
          endpoint = ["http://192.168.253.78:5000"]

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

  [plugins."io.containerd.internal.v1.opt"]
    path = "/opt/containerd"

  [plugins."io.containerd.internal.v1.restart"]
    interval = "10s"

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

  [plugins."io.containerd.metadata.v1.bolt"]
    content_sharing_policy = "shared"

  [plugins."io.containerd.monitor.v1.cgroups"]
    no_prometheus = false

  [plugins."io.containerd.nri.v1.nri"]
    disable = true
    disable_connections = false
    plugin_config_path = "/etc/nri/conf.d"
    plugin_path = "/opt/nri/plugins"
    plugin_registration_timeout = "5s"
    plugin_request_timeout = "2s"
    socket_path = "/var/run/nri/nri.sock"

  [plugins."io.containerd.runtime.v1.linux"]
    no_shim = false
    runtime = "runc"
    runtime_root = ""
    shim = "containerd-shim"
    shim_debug = false

  [plugins."io.containerd.runtime.v2.task"]
    platforms = ["linux/amd64"]
    sched_core = false

  [plugins."io.containerd.service.v1.diff-service"]
    default = ["walking"]

  [plugins."io.containerd.service.v1.tasks-service"]
    blockio_config_file = ""
    rdt_config_file = ""

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

  [plugins."io.containerd.snapshotter.v1.blockfile"]
    fs_type = ""
    mount_options = []
    root_path = ""
    scratch_file = ""
  [plugins."io.containerd.snapshotter.v1.btrfs"]
    root_path = ""

  [plugins."io.containerd.snapshotter.v1.devmapper"]
    async_remove = false
    base_image_size = ""
    discard_blocks = false
    fs_options = ""
    fs_type = ""
    pool_name = ""
    root_path = ""

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

  [plugins."io.containerd.snapshotter.v1.overlayfs"]
    mount_options = []
    root_path = ""
    sync_remove = false
    upperdir_label = false

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

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

  [plugins."io.containerd.transfer.v1.local"]
    config_path = ""
    max_concurrent_downloads = 3
    max_concurrent_uploaded_layers = 3

    [[plugins."io.containerd.transfer.v1.local".unpack_config]]
      differ = ""
      platform = "linux/amd64"
      snapshotter = "overlayfs"

[proxy_plugins]

[stream_processors]

  [stream_processors."io.containerd.ocicrypt.decoder.v1.tar"]
    accepts = ["application/vnd.oci.image.layer.v1.tar+encrypted"]
    args = ["--decryption-keys-path", "/etc/containerd/ocicrypt/keys"]
    env = ["OCICRYPT_KEYPROVIDER_CONFIG=/etc/containerd/ocicrypt/ocicrypt_keyprovider.c
    path = "ctd-decoder"
    returns = "application/vnd.oci.image.layer.v1.tar"

  [stream_processors."io.containerd.ocicrypt.decoder.v1.tar.gzip"]
    accepts = ["application/vnd.oci.image.layer.v1.tar+gzip+encrypted"]
    args = ["--decryption-keys-path", "/etc/containerd/ocicrypt/keys"]
    env = ["OCICRYPT_KEYPROVIDER_CONFIG=/etc/containerd/ocicrypt/ocicrypt_keyprovider.c
    path = "ctd-decoder"
    returns = "application/vnd.oci.image.layer.v1.tar+gzip"

[timeouts]
  "io.containerd.timeout.bolt.open" = "0s"
  "io.containerd.timeout.metrics.shimstats" = "2s"
  "io.containerd.timeout.shim.cleanup" = "5s"
  "io.containerd.timeout.shim.load" = "5s"
  "io.containerd.timeout.shim.shutdown" = "3s"
  "io.containerd.timeout.task.state" = "2s"

[ttrpc]
  address = ""
  gid = 0
  uid = 0
```

Than restart it

```
sudo systemctl restart containerd
sudo systemctl status containerd
```

Now pull it

```
docker pull 192.168.253.78:5000/k8s/kube-apiserver:v1.29.0
docker pull 192.168.253.78:5000/k8s/kube-controller-manager:v1.29.0
docker pull 192.168.253.78:5000/k8s/kube-scheduler:v1.29.0
docker pull 192.168.253.78:5000/k8s/kube-proxy:v1.29.0
docker pull 192.168.253.78:5000/k8s/etcd:3.5.10-0
docker pull 192.168.253.78:5000/k8s/coredns/coredns:v1.11.1
docker pull 192.168.253.78:5000/k8s/pause:3.9

# tag it
docker tag 192.168.253.78:5000/k8s/kube-apiserver:v1.29.0 registry.k8s.io/kube-apiserve
docker tag 192.168.253.78:5000/k8s/kube-controller-manager:v1.29.0 registry.k8s.io/kube
docker tag 192.168.253.78:5000/k8s/kube-scheduler:v1.29.0 registry.k8s.io/kube-schedule
docker tag 192.168.253.78:5000/k8s/kube-proxy:v1.29.0 registry.k8s.io/kube-proxy:v1.29.
docker tag 192.168.253.78:5000/k8s/etcd:3.5.10-0 registry.k8s.io/etcd:3.5.10-0
docker tag 192.168.253.78:5000/k8s/pause:3.9 registry.k8s.io/pause:3.9
docker tag 192.168.253.78:5000/k8s/coredns/coredns:v1.11.1 registry.k8s.io/coredns/core
```

now init it !!!

`sudo kubeadm init --pod-network-cidr=192.168.0.0/16 --cri-socket=unix:///var/run/cri-dockerd.sock --image-repository=registry.k8s.io --kubernetes-version=v1.29.0`

Did you have to specify version??? Yes, my friend you have to specify all things while init, so just do it. Or you can make config file and pass it if you are too free, but I'm not that's why.

Than comes mendatory step that also given by kubernetes itself

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

init is done !!!

Apply CNI plugin

Let me clear WHY? first !!!

Calico is one of the most popular **network plugins** used in Kubernetes clusters. It plays a critical role in **networking and security** between your pods, especially in production-grade clusters.

Let’s walk through **what Calico is**, **why it’s needed**, and **how it fits into Kubernetes networking**.

🧭 What Is Calico?

Calico is a **Container Network Interface (CNI)** plugin that provides:

1. **Networking** – Makes sure all pods across all nodes can communicate.
2. **Network Policies** – Enforces security rules between pods (who can talk to whom).

Think of it as:

> *A networking and security engine that connects pods across machines, and can enforce rules like a firewall.*

⚙️ Why Do You Need Calico in Kubernetes?

Kubernetes **does not provide networking on its own**. It only defines **what the network should look like**, not how to implement it.

It expects a CNI plugin to:

- Assign IP addresses to pods
- Route traffic between them
- Enforce network policies (if defined)

That’s where Calico comes in.

🧱 Key Responsibilities of Calico

| Feature | Description | |--------|-------------| | **Pod-to-Pod Networking** | Calico makes sure that pods across different nodes can communicate using unique IPs. | | **IP Address Management (IPAM)** | Assigns IPs to pods without overlaps. | | **Routing** | Uses either **Linux routing tables** or **BGP** to route traffic. | | **Network Policies** | Allows you to define firewall-style rules — e.g., allow traffic only from certain namespaces or labels. | | **High Performance** | No overlay by default — uses native Linux networking. Fast and simple. |

🔄 What Happens Without Calico?

If you don't install **any** CNI plugin:

- Pods will be created, but won't get IP addresses.
- Kubernetes networking won't work — pods won’t communicate.
- `kubeadm` will show this:

```
[WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver.
...
[WARNING] Unable to determine the pod CIDR. Network might not work properly.
```

You **must** install a CNI plugin like **Calico**, **Flannel**, or **Cilium**.

🧩 How Calico Works in a Cluster

```
+--------------------------+
|        Node A            |
| +----------------------+ |
| |  Pod A (10.0.0.1)    | |
| +----------------------+ |
|          |               |
| +----------------------+ |
| |     Calico CNI       | |  <-- Assigns IP, sets routes
| +----------------------+ |
|          |               |
|     Linux Kernel Routing |
+--------------------------+

            ⇅ Pod-to-Pod Traffic

+--------------------------+
|        Node B            |
| +----------------------+ |
| |  Pod B (10.0.0.2)    | |
| +----------------------+ |
|          |               |
| +----------------------+ |
| |     Calico CNI       | |
| +----------------------+ |
|          |               |
|     Linux Kernel Routing |
+--------------------------+
```

Calico sets up **native IP routing** so that Pod A can talk directly to Pod B — even if they’re on different machines.

📜 Sample Network Policy with Calico

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-external
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: backend
```

This policy:

- Applies to pods labeled `role=db`
- Denies all inbound traffic **except** from pods labeled `role=backend`

Without Calico (or another network-policy-aware CNI), this policy won’t be enforced at all.

✅ Summary

Feature	Calico Provides
Pod networking	✅ Yes
IP address management	✅ Yes
Routing	✅ Yes (native or BGP)
Network policy enforcement	✅ Yes
Overlay network	❌ No (by default — that's good for performance!)

🧰 When to Use Calico

✅ Production-ready clusters
✅ When you need **high performance and simplicity**
✅ When you want **fine-grained network security policies**
✅ Works well in **cloud**, **on-prem**, and **bare metal**

IK Here's a detailed **comparison between the most popular Kubernetes CNI (Container Network Interface) plugins**:

🔰 Calico vs Flannel vs Cilium

These are three widely used **CNI plugins** that solve **pod networking** in Kubernetes, but each takes a different approach depending on your **performance, security, and operational complexity needs**.

Feature / Aspect	**Calico**	**Flannel**	**Cilium**
Primary Focus	Networking + Network Policies (Security)	Simple Networking	Advanced Security + Observability + eBPF
Network Backend	Native Linux routing or BGP	VXLAN, host-gw	eBPF/XDP-based (no iptables by default)
Overlay Support	Optional (VXLAN, IPIP, WireGuard)	Yes (VXLAN/IPIP)	Optional, mostly direct routing with eBPF
Network Policies	✅ Full Kubernetes network policy support	❌ Not supported	✅ Advanced policies (L7-aware)
Performance	🚀 Very fast (native routing)	⚠️ Slower (overlay adds latency)	🚀 Extremely fast (no iptables, uses eBPF)
Security	✅ L3/L4 policies (CIDR, labels)	❌ None	✅ L3-L7 policies, DNS-aware, TLS-aware
Complexity	Medium (BGP optional)	Very Simple	High (eBPF, complex features)
Observability	Basic via Prometheus, IPtables stats	Minimal	Advanced Hubble UI + Metrics + Tracing
Use Case Fit	High-perf clusters, on-prem, cloud-hybrid	Small, simple dev/test clusters	Secure, modern, multi-cloud & service-mesh ready
Kube Proxy Replacement	❌ No	❌ No	✅ Yes (`cilium-proxy`)
Windows Support	✅ Yes	✅ Yes	🚫 Not yet

🧪 What Happens Under the Hood?

🔷 Calico:

- Routes packets **natively** using Linux kernel routing tables or **BGP**.
- By default, uses **direct routing** = low overhead and better performance.
- Can optionally encrypt traffic (with WireGuard).

🔶 Flannel:

- Creates an **overlay network** using **VXLAN** or **host-gw** mode.
- Simpler setup — but uses **encapsulation**, so has **higher latency**.
- No support for Kubernetes network policies.

🟡 Cilium:

- Uses **eBPF** (extended Berkeley Packet Filter), a modern kernel tech.
- Doesn’t rely on `iptables`, which makes it **faster and more efficient**.
- Enables L7-aware policies (e.g., "allow only GET /api calls").
- Optional replacement for `kube-proxy` via `cilium-proxy`.

🧰 When Should You Use Each?

| Use Case / Requirement | Recommended CNI | |--|------------------| | 🔰 **Beginner, local dev, or test cluster** | **Flannel** – simple, easy | | 🚀 **High performance, scalable production cluster**| **Calico** – fast, simple routing | | 🔐 **Security-critical, zero-trust, or service mesh**| **Cilium** – advanced L3–L7 security | | 📊 **Deep observability & metrics** | **Cilium** (with Hubble) | | 🌐 **Multi-cloud / bare-metal mix** | **Calico** (works with BGP) | | 🧩 **Need Windows pod networking** | **Calico** or **Flannel** (Cilium doesn’t support Windows yet) |

🧠 TL;DR

| Plugin | Pros | Cons | |---------|------|------| | **Calico** | Fast, scalable, supports policies, no overlay by default | Slightly more config (BGP optional) | | **Flannel** | Simple setup, great for dev/test | No policies, slower due to VXLAN | | **Cilium** | Ultra-fast, modern eBPF, deep observability, L7 policies | Steeper learning curve, no Windows |

🔧 Commands to Install (Quick Look)

Calico (kubeadm setup)

```
kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.27.0/manifes
```

Flannel

```
kubectl apply -f https://raw.githubusercontent.com/flannel-io/flannel/master/Documentat
```

Cilium

```
curl -sL https://install.cilium.io | bash
```

Let me know:

- If you want to see benchmark stats

- How to switch CNIs on a running cluster
- Or how to test network policies using any of them!

Download it and put it in your machine `https://raw.githubusercontent.com/projectcalico/calico/v3.27.0/manifests/calico.yaml` here is the linke save as calico.yaml.

At 78 (Where we have internet)

---

pull and push img to registry commands

```
docker pull calico/cni:v3.27.0
docker pull calico/node:v3.27.0
docker pull calico/kube-controllers:v3.27.0

docker tag calico/cni:v3.27.0 192.168.253.78:5000/calico/cni:v3.27.0
docker tag calico/node:v3.27.0 192.168.253.78:5000/calico/node:v3.27.0
docker tag calico/kube-controllers:v3.27.0 192.168.253.78:5000/calico/kube-controllers:

docker push 192.168.253.78:5000/calico/cni:v3.27.0
docker push 192.168.253.78:5000/calico/node:v3.27.0
docker push 192.168.253.78:5000/calico/kube-controllers:v3.27.0
```

Oh nooo, it didn’t work? Actually, wait—just use `sudo` .

Apply the YAML `kubectl apply -f calico.yaml`

✅ What To Do Now

Watch Calico Pods Start `sudo kubectl get pods -n kube-system -w`

After all of this If you got versing issue I'm showing you how to debug

```
~$ kubectl get nodes
NAME              STATUS     ROLES           AGE    VERSION
oriondevsrv-amd   NotReady   control-plane   133m   v1.29.2
```

- here eveything is good (not like your life) but the problem is it's not ready like you not read for this document

1. Check Node Description `kubectl describe node oriondevsrv-amd`
2. Is Kubelet Running? `systemctl status kubelet` Or restart it just to be sure: `sudo systemctl restart kubelet`
3. Check Container Runtime `systemctl status containerd`
4. CNI (Calico) Not Fully Running Check Pod Statuses in kube-system Run: `kubectl get pods -n kube-system` We're especially looking for:

- calico-node-xxxx
- calico-kube-controllers-xxxx
- kube-proxy Check their statuses: they must be `Running` or `Completed`.

Check calico-node DaemonSet Logs `kubectl logs -n kube-system -l k8s-app=calico-node`

Check Calico Node Pod Directly `kubectl describe pod -n kube-system <name-of-calico-node-pod>` Look at:

- Events at the bottom (look for errors)
- Any ImagePullBackOff, CrashLoopBackOff, or failed init containers

🛠️ If You Need to Force a Restart (Optional) `kubectl delete pod -n kube-system -l k8s-app=calico-node`

If you got problem is calico-node image is missing or incorrectly referenced

Check the image reference in your calico.yaml `grep image calico.yaml` Ensure image exists locally or in local registry `sudo docker images | grep calico`

So I got versioning issue `sed -i 's/v3.25.0/v3.27.0/g' calico.yaml` < replace in linux > The command:

```
sed -i 's/v3.25.0/v3.27.0/g' calico.yaml
```

Breakdown of the Command

| Part | Explanation | |------|-------------| | `sed` | The command-line stream editor tool used for parsing and transforming text. | | `-i` | Edit files **in-place** (i.e., directly modifies the file without needing to save as a new file). | | `'s/v3.25.0/v3.27.0/g'` | This is the **substitution expression**:
- `s` stands for **substitute**.
- `v3.25.0` is the **pattern to match**.
- `v3.27.0` is the **replacement string**.
- `g` means **global** — replace **all** occurrences on each line, not just the first. | | `calico.yaml` | The **target file** to be edited. |

Most imp

checking log in kubectl `kubectl logs calico-node-zc7d4 -n kube-system -c install-cni`

5. The calico-node service account lacks the required RBAC permissions to generate a token for calico-cni-plugin, which is needed to build the in-cluster kubeconfig for CNI. Inspect and Confirm RBAC is Included Open `calico.yaml` and check that it includes:

`ClusterRole` and `ClusterRoleBinding` for calico-node

ServiceAccount definitions for:

- calico-node
- calico-kube-controllers

Delete and restart pods after applying new yaml

```
kubectl delete pod -l k8s-app=calico-node -n kube-system
kubectl delete pod -l k8s-app=calico-kube-controllers -n kube-system
```

Re-check pod status

`kubectl get pods -n kube-system -o wide`

Verify RBAC

```
kubectl get clusterrole calico-node -o yaml
kubectl get clusterrolebinding calico-node -o yaml
```

if you got

```
The ClusterRoleBinding "calico-cni-plugin" is invalid: roleRef: Invalid value: rbac.Rol
```

Why This Happened

- The calico.yaml file you applied contains a ClusterRoleBinding named calico-cni-plugin, and the one already existing in your cluster has a different roleRef. Kubernetes refuses to overwrite it.

Solution:

Delete the Existing ClusterRoleBinding (Recommended)

```
kubectl delete clusterrolebinding calico-cni-plugin
kubectl apply -f calico.yaml
```

> *This removes the conflicting binding and allows the new manifest to re-create it with the correct roleRef.*

Than Restart Calico Pods `kubectl delete pod -l k8s-app=calico-node -n kube-system`

If you got `clusterrolebinding.rbac.authorization.k8s.io/calico-cni-plugin created` than your problem is solved my friend.

Now

Delete and Restart Calico Pods

```
kubectl delete pod -l k8s-app=calico-node -n kube-system
kubectl delete pod -l k8s-app=calico-kube-controllers -n kube-system
```

Watch Pod Status `kubectl get pods -n kube-system -o wide` ot `watch kubectl get pods -n kube-system`

For checking cluser health

```
kubectl get pods -n kube-system -o wide
kubectl get nodes -o wide
```

Testing Time

---

get testbox images by run that into 78 server (Where we got internet)

```
docker pull busybox:1.28
docker save busybox:1.28 -o busybox.tar
```

now load that in your machine `docker load -i busybox.tar`

delete old node if exist

`kubectl delete pod testbox --ignore-not-found`

run textbox in kubernetes `kubectl run testbox --image=busybox:1.28 --restart=Never -- sleep 3600`

after sometime run

```
kubectl exec -it testbox -- sh
nslookup kubernetes.default
```

check it's running or not

```
kubectl get pods
kubectl describe pod testbox
```

in describe -> event you got error like

```
Events:
  Type     Reason            Age   From               Message
  ----     ------            ----  ----               -------
  Warning  FailedScheduling  55s   default-scheduler  0/1 nodes are available: 1 node(s
```

Solution:

Thank you — that message explains **exactly why your test pod won’t start**:

```
0/1 nodes are available: 1 node(s) had untolerated taint {node-role.kubernetes.io/contr
```

❗ Root Cause

Your **only node** is a **control-plane node**, and by default Kubernetes applies a **taint** to prevent normal workloads from being scheduled on it:

```
key: node-role.kubernetes.io/control-plane
effect: NoSchedule
```

This is meant to protect your control plane — but in **single-node clusters**, you'll want to **allow scheduling test workloads (like BusyBox)** on the control-plane.

Kubernetes adds this **taint** to control-plane nodes:

```
node-role.kubernetes.io/control-plane:NoSchedule
```

Which means:

- **No pods** (unless they "tolerate" the taint) can be scheduled on these nodes.
- This is **default kubeadm behavior** for all control-plane nodes.

So yes — **by default**, every time you create a new control-plane node (or reinitialize a cluster), this taint is applied.

✅ Option 1: Patch kubeadm config to skip tainting (Before Cluster Init)

If you're using `kubeadm init`, you can **disable the taint** entirely during cluster creation:

```yaml
# kubeadm-config.yaml
apiVersion: kubeadm.k8s.io/v1beta3
kind: InitConfiguration
nodeRegistration:
  taints: []
```

Then run:

```
kubeadm init --config kubeadm-config.yaml
```

This way, the taint is **never added**, and workloads can run by default on the control-plane.

✅ Option 2: Automatically remove taint after init

You can create a post-`kubeadm init` script or automation that runs:

```
kubectl taint nodes $(hostname) node-role.kubernetes.io/control-plane:NoSchedule-
```

This is great for scripted/automated installs.

✅ Option 3: Use Tolerations in Deployment Templates

If you always want specific workloads to run on control-plane nodes *without removing the taint*, modify your pod or deployment YAML like this:

```yaml
spec:
  tolerations:
  - key: "node-role.kubernetes.io/control-plane"
    operator: "Exists"
    effect: "NoSchedule"
```

But that needs to be added **for each pod or deployment**.

🛡️ Best Practice:

For development, testing, or edge clusters that will *only ever have a single node*, the **simplest and safest option** is:

> ***Remove the taint once after install*** *using:*

```
kubectl taint nodes $(hostname) node-role.kubernetes.io/control-plane:NoSchedule-
```

Once removed, you don't need to worry again unless:

- You re-init the node
- You add new nodes with taints

Now undertand it

Yes — exactly. You only need to run the command **once per new control-plane node** (typically just once after `kubeadm init`) unless:

- You reinitialize the cluster from scratch
- You add another control-plane node
- Or the taint is re-applied manually or during upgrades

✅ Breakdown of the Command

```
kubectl taint nodes $(hostname) node-role.kubernetes.io/control-plane:NoSchedule-
```

🔍 Here's what each part means:

| Part | Meaning | |------|--------| | `kubectl taint nodes` | Taints (or untaints) a node | | `$(hostname)` | Dynamically inserts the name of the current node (e.g., `oriondevsrv-amd`) | | `node-role.kubernetes.io/control-plane:NoSchedule-` | This is the taint you're **removing** | | `-` at the end | Indicates **removal** of the taint |

🔁 Example:

If your node is called `oriondevsrv-amd`, this command becomes:

```
kubectl taint nodes oriondevsrv-amd node-role.kubernetes.io/control-plane:NoSchedule-
```

This tells Kubernetes:

> *"Please remove the taint `node-role.kubernetes.io/control-plane=NoSchedule` from the node `oriondevsrv-amd`, so it can schedule normal pods."*

✅ When to Use It

- **Right after `kubeadm init`**
- Or anytime you want to run pods on a single-node (control-plane-only) cluster

After all of this Re-run the test pod:

```
kubectl delete pod testbox --ignore-not-found
kubectl run testbox --image=busybox:1.28 --restart=Never -- sleep 3600
```

Once pod is running:

```
kubectl exec -it testbox -- sh
nslookup kubernetes.default
```

you can see like

```
~$ kubectl exec -it testbox -- sh
/ # nslookup kubernetes.default
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local
```

Bingo !!!

All things are on right track I hope after reading this your life will be...

All of this what you got life?? NO!! you got !!!

✅ DNS is Working Inside the Cluster

Your pod successfully resolved `kubernetes.default` via:

- **CoreDNS IP:** `10.96.0.10`
- **Service name:** `kube-dns.kube-system.svc.cluster.local`

This means:

- CoreDNS is running and reachable
- Calico networking is properly set up
- Your airgapped Kubernetes cluster is **fully operational**

🧩 What You’ve Accomplished

| Component | Status | |------------------------|--------| | Calico CNI | ✅ Running and configured | | CoreDNS | ✅ Resolving cluster services | | Pod Scheduling | ✅ Fixed taint and launched workload | | Image Handling | ✅ Handled BusyBox via offline import | | Cluster Network (DNS) | ✅ Verified working inside pod |

---

simple guide for saving your time

Here's a **deep and complete explanation** of Kubernetes and the role of components like `kubelet` , `kubeadm` , `kubectl` , `cri-docker` , and **CNI plugins like Calico**, including why each is needed and how they interact.

🚀 What is Kubernetes?

Kubernetes is an **open-source container orchestration platform** that automates:

- **Deployment**
- **Scaling**
- **Networking**
- **Management of containerized applications**

🧩 Core Components Overview

| Component | Purpose | |---------------|---------| | **kubelet** | Runs on each node to manage containers | | **kubeadm** | Bootstraps and sets up a Kubernetes cluster | | **kubectl** | CLI tool to interact with Kubernetes API | | **cri-docker**| Connects Kubernetes to Docker runtime | | **CNI plugin**| Handles networking between Pods (like Calico) |

🔧 `kubelet` : The Node Agent

✅ Role:

- **Primary agent** on each node.
- Communicates with the **API Server**.
- Ensures containers (via container runtime) are running as defined in **PodSpecs**.

🔗 Communicates with:

- **Container runtime (e.g., Docker via CRI)**
- **API server**
- **CNI plugin** (indirectly via container runtime)

🔄 Flow:

1. Pulls Pod definition from API Server.
2. Talks to container runtime to create containers.
3. Monitors health and status.
4. Reports node and pod status back to API Server.

🧱 `kubeadm` : The Cluster Bootstrapper

✅ Role:

- Tool to **install and configure** a Kubernetes cluster.
- Sets up control plane components:
  - `kube-apiserver`
  - `kube-scheduler`
  - `kube-controller-manager`
  - `etcd`

🔧 Why needed?

- Without `kubeadm`, setting up a cluster manually is error-prone.
- Simplifies init and join processes.

🔗 Communicates with:

- The **kubelet** on each node (to configure it)
- Installs and configures **control plane components**

💻 `kubectl`: The CLI Interface

✅ Role:

- **Command-line tool** to interact with the Kubernetes API Server.

🔧 Why needed?

- Developers/admins use it to create, inspect, delete resources (pods, deployments, services).

🔗 Communicates with:

- **API server** only.
- You don't directly talk to nodes or containers.

🐳 `cri-docker`: The Runtime Interface

✅ Role:

- **Container Runtime Interface (CRI)** adapter that connects Kubernetes with **Docker**.
- Needed because Kubernetes expects a **CRI-compliant runtime**, but Docker isn't directly CRI-compliant.

> ⚠️ *Note: `cri-docker` is deprecated; modern clusters use `containerd` or `CRI-O`.*

🔗 Communicates with:

- **kubelet** (which uses CRI to talk to runtime)
- **Docker daemon**

🌐 CNI Plugin: Example - Calico

✅ Role:

- **CNI (Container Network Interface)** plugin provides **network connectivity** for Pods and Services.
- Calico adds **routing + network policies + security**.

🔧 Why needed?

- Kubernetes does not provide native Pod-to-Pod networking; it depends on **CNI plugins**.
- CNI handles:
  - Pod IP allocation
  - Routing between Pods across nodes
  - Network policies (firewalling)

🧠 How Calico Works:

1. Each node runs a Calico agent (`calico-node`).
2. When kubelet asks to create a Pod:
   - It triggers the CNI plugin (`calico` binary).
   - Calico assigns a **unique IP** and sets up a **virtual interface** for the Pod.
3. Calico adds **routes** so other Pods can reach it.
4. Optional: Calico can enforce **network policies** (who can talk to whom).

🔗 Communicates with:

- **kubelet** via CNI calls during Pod setup
- **etcd** (to store Calico's internal state)
- **Other Calico nodes** (to sync routing info)

🔄 How They Work Together: A Practical Flow

1. **Bootstrap the Cluster**:
   - You run `kubeadm init`
   - It sets up control plane and installs default configs
2. **Join Worker Nodes**:
   - You run `kubeadm join` on workers
   - It configures `kubelet` and links it to API Server
3. **Install CNI Plugin** (e.g., Calico):

   - Applied via a manifest (`kubectl apply -f calico.yaml`)
   - Enables Pod networking

4. **Deploy an App**:

   - You run `kubectl apply -f myapp.yaml`
   - `kubectl` sends it to API Server

5. **API Server notifies kubelet** on a node:

   - `kubelet` asks the runtime (via CRI-Docker) to run containers
   - CNI plugin (Calico) assigns IPs and routes

📡 Communication Summary

```
Developer ↔ kubectl ↔ kube-apiserver
                               ⬇️
                +--------------+--------------+
                |                             |
        kubelet (each node)           kube-controller-manager
                |                             |
        CRI (e.g., cri-docker)         etcd (store)
                |
        Docker or containerd
                |
           CNI plugin (e.g., Calico)
                |
          Pod virtual network setup
```

🧠 Why All Are Needed?

| Component | Why It's Needed | |----------|------------------| | `kubelet` | Actually runs and monitors containers on each node | | `kubeadm` | Automates complex cluster setup tasks | | `kubectl` | Interface to interact with the cluster | | `cri-docker` | Bridges Kubernetes with Docker runtime | | `Calico` (CNI) | Provides pod-to-pod networking and security |

Pro Tip:

Check version of kubernetes by:

```
~$ kubectl version
Client Version: v1.29.2
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
Server Version: v1.29.0
```

There's no problem using a major version of kubectl with one version difference from your cluster (like client v1.29.2 and server v1.29.0), it's not recommended that you have a big difference between them (like 3 versions).

If the situation was the opposite (server version > client version) I would recommend to update your client :)

🛠️ Crafted by Vinit — after countless debugging sessions 🐞, deep-dive research 📖, juggling 19+ open tabs 🧠💥, and fueled by just enough hope 💫!!!