# Bare-Metal Networking

For Everyone

Mateusz Kowalski
Principal Software Engineer

## $whoami

- Based in Switzerland
- Background in academia, banking and telco
- Buzzwords
  - Cloud
  - Metal
  - Network security
  - ~~Artificial intelligence~~

# Containers on Bare Metal

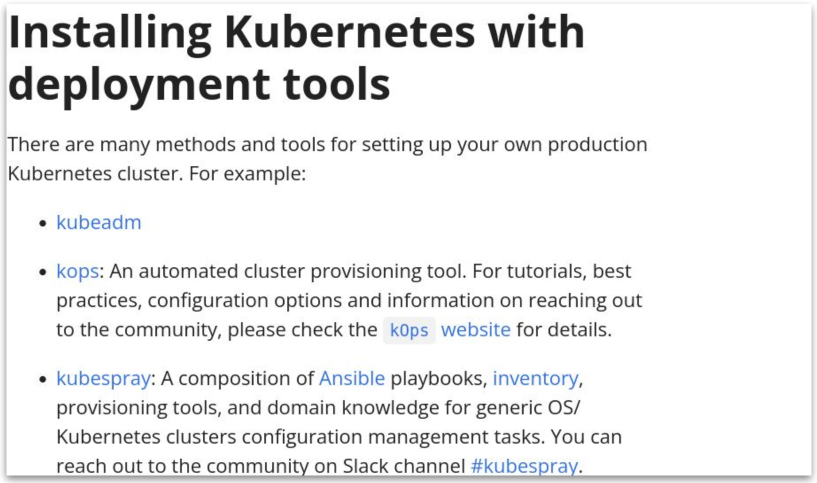
## Why?

- HPC workloads
- AI / ML
- Telco customers in the Edge
- Critical network equipment
- Specialized hardware (GPUs, NICs)
- Benchmarking

mko@redhat.com | github.com/mkowalski

# Installing a cluster

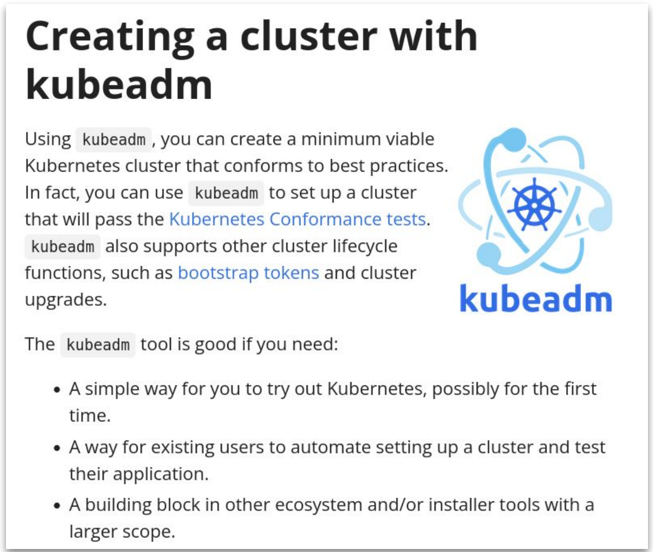
This is complicated



## Installing Kubernetes with deployment tools

There are many methods and tools for setting up your own production Kubernetes cluster. For example:

- kubeadm
- kops: An automated cluster provisioning tool. For tutorials, best practices, configuration options and information on reaching out to the community, please check the `kOps` website for details.
- kubespray: A composition of Ansible playbooks, inventory, provisioning tools, and domain knowledge for generic OS/ Kubernetes clusters configuration management tasks. You can reach out to the community on Slack channel #kubespray.

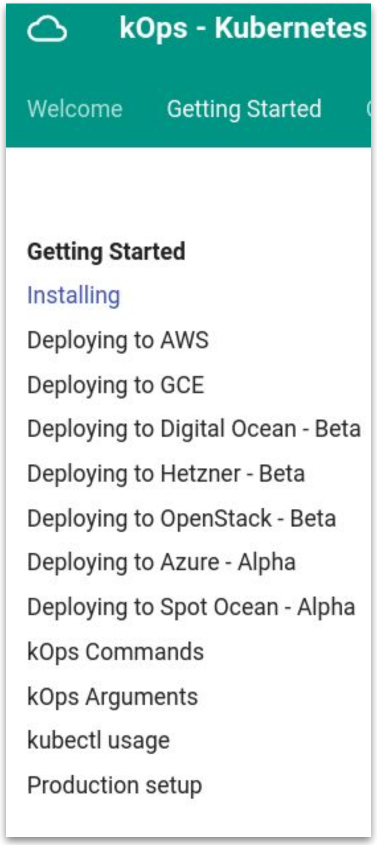


## Creating a cluster with kubeadm

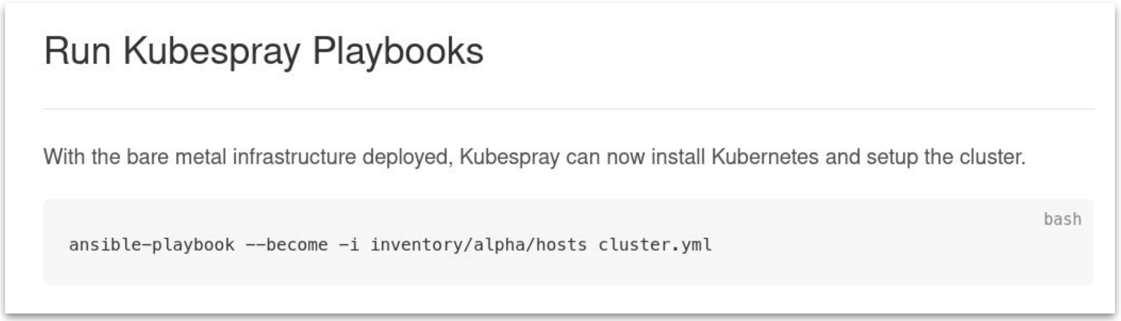
Using `kubeadm`, you can create a minimum viable Kubernetes cluster that conforms to best practices. In fact, you can use `kubeadm` to set up a cluster that will pass the Kubernetes Conformance tests. `kubeadm` also supports other cluster lifecycle functions, such as bootstrap tokens and cluster upgrades.

The `kubeadm` tool is good if you need:

- A simple way for you to try out Kubernetes, possibly for the first time.
- A way for existing users to automate setting up a cluster and test their application.
- A building block in other ecosystem and/or installer tools with a larger scope.



Getting Started
Installing
Deploying to AWS
Deploying to GCE
Deploying to Digital Ocean - Beta
Deploying to Hetzner - Beta
Deploying to OpenStack - Beta
Deploying to Azure - Alpha
Deploying to Spot Ocean - Alpha
kOps Commands
kOps Arguments
kubectl usage
Production setup



## Run Kubespray Playbooks

With the bare metal infrastructure deployed, Kubespray can now install Kubernetes and setup the cluster.

```bash
ansible-playbook --become -i inventory/alpha/hosts cluster.yml
```

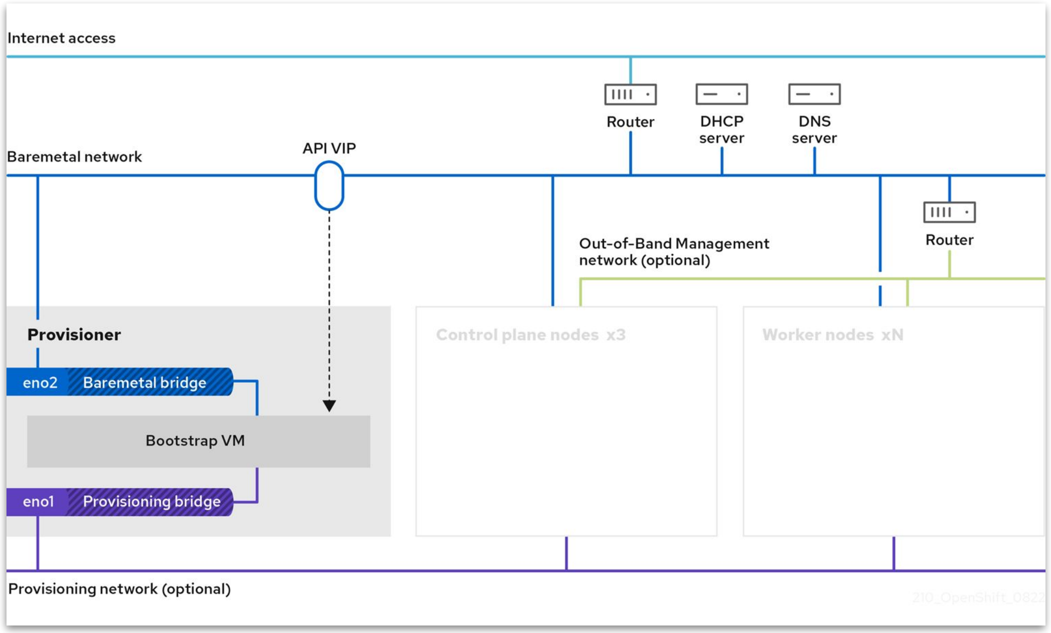
## Installing a cluster

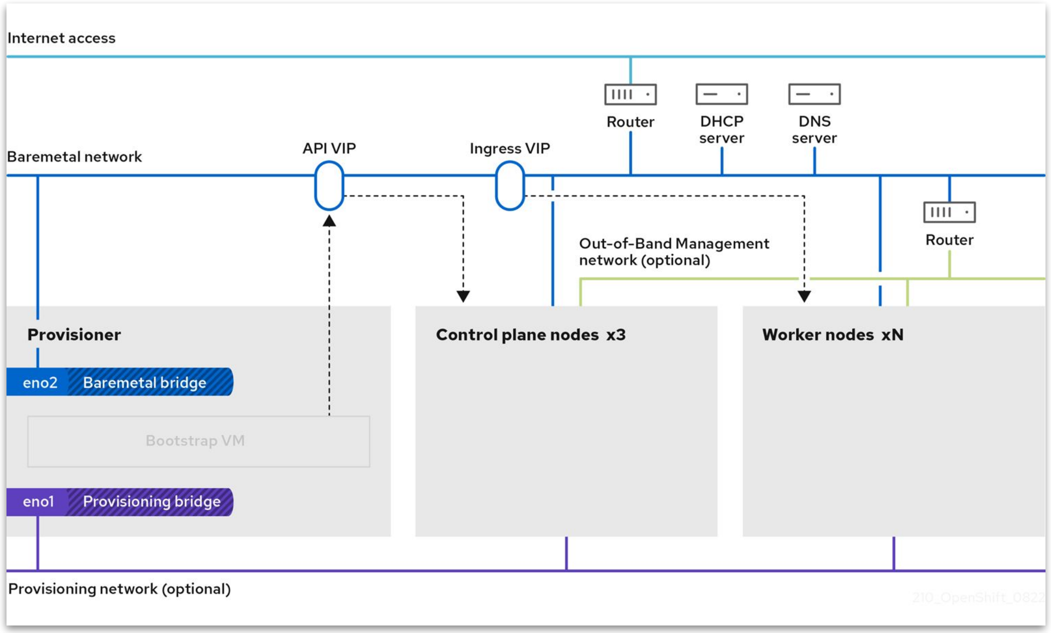
What you need to care about

- Accessing the cluster (API, workload, etc.)
  - Load balancer?
- DNS infrastructure
- Network configuration
- Other dirty tricks...

# Cluster Load Balancer

## Installation & Bootstrap

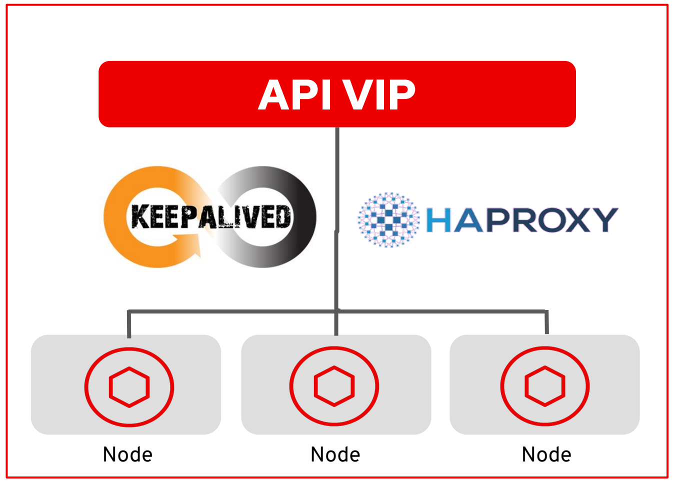


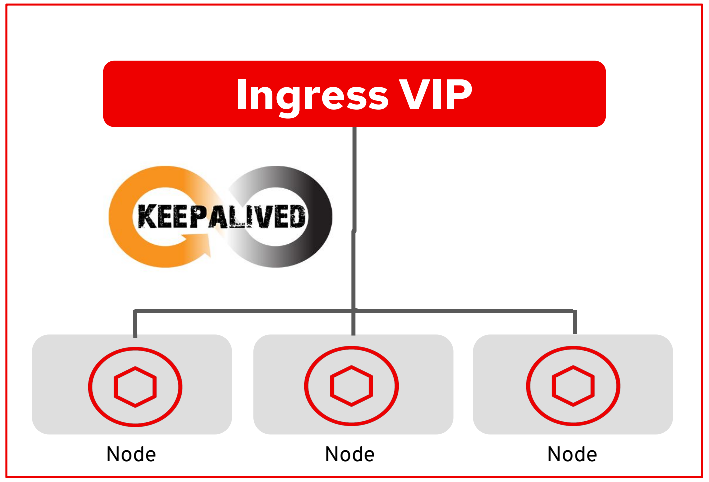


Red Hat

# Cluster Load Balancer

## Keepalived, HAProxy





## Cluster Load Balancer

Let's make it stable

- Problem - Failure of a single kubeapi-server can't cause VIP to float
  - Less floating == less connections broken
  - HAProxy also loadbalances traffic
- Keepalived keeps the VIP unless HAProxy dies
  - Kubeapi death != HAProxy death
- Healthchecks timed in a smart way

## Cluster Load Balancer

### Limitations

- Scaling
  - VIP on a single node
- "Everything" in a single L2
  - Keepalived can't work across subnets
  - With DMZ/isolated nodes, Ingress VIP needs to be tuned (placement in a single subnet)

## Cluster DNS

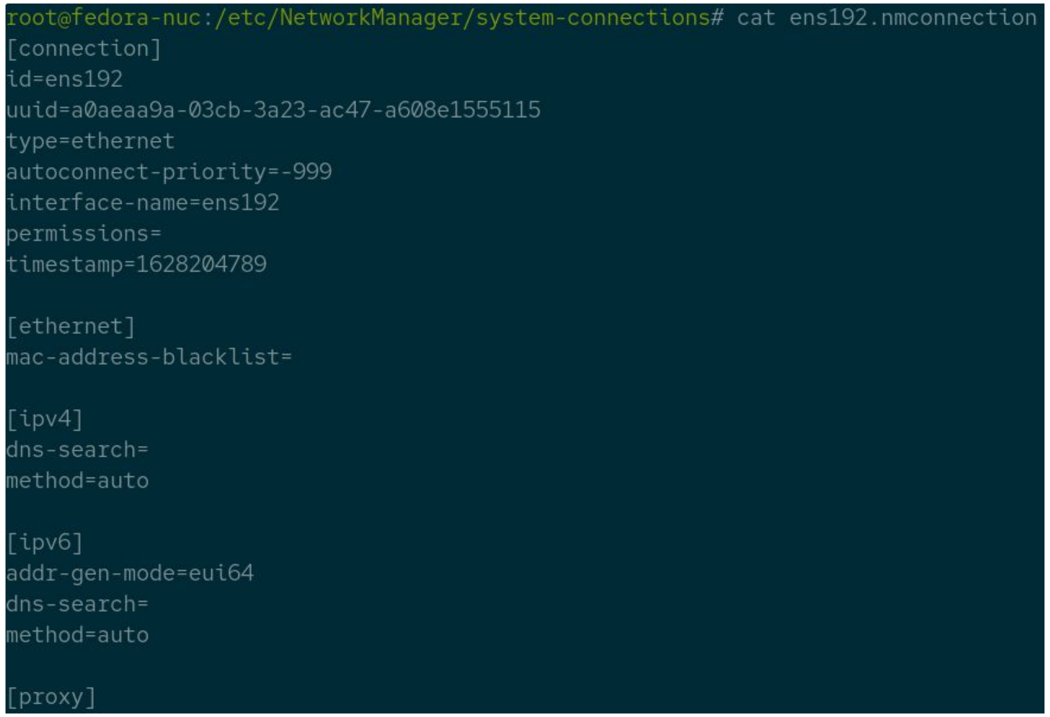
Never trust the user

CoreDNS

- We need api.example.com, *.apps.example.com, api-int.example.com
  - We could rely on user's DNS
- CoreDNS running everywhere (static Pod)
- /etc/resolv.conf configured to use it
  - Containers blindly use it too
- NetworkManager requires tuning
  - Fight over /etc/resolv.conf
  - Hooks ("dispatcher scripts") for the help

## Network Configuration

At installation-time

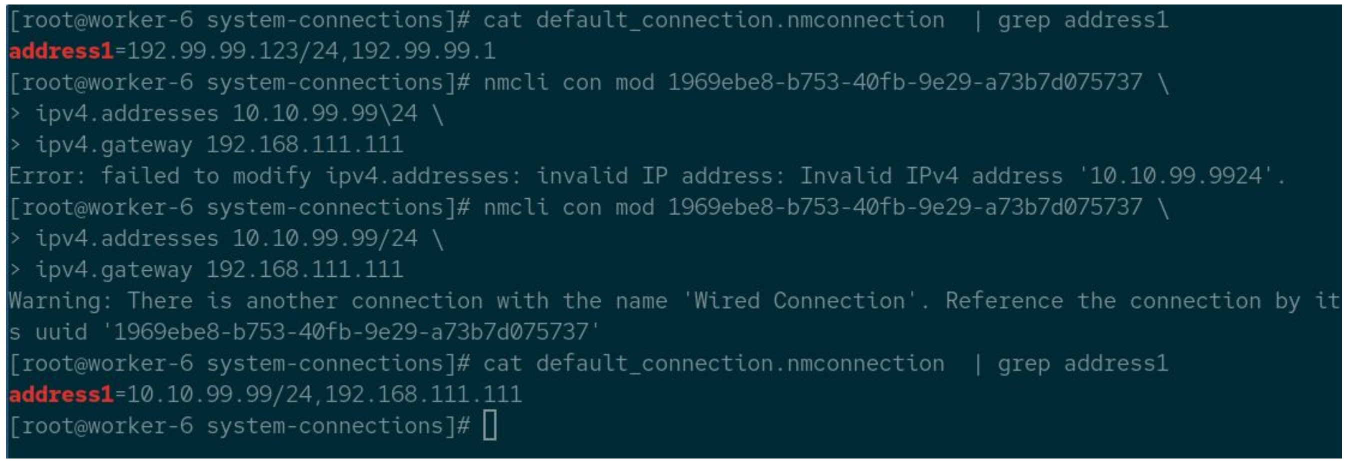


```
root@fedora-nuc:/etc/NetworkManager/system-connections# cat ens192.nmconnection
[connection]
id=ens192
uuid=a0aeaa9a-03cb-3a23-ac47-a608e1555115
type=ethernet
autoconnect-priority=-999
interface-name=ens192
permissions=
timestamp=1628204789

[ethernet]
mac-address-blacklist=

[ipv4]
dns-search=
method=auto

[ipv6]
addr-gen-mode=eui64
dns-search=
method=auto

[proxy]
```

- NetworkManager uses using static nmconnection files
- Changes not applied automatically
  - You may break your config…
  - … and not notice for ages
- Does not scale

# Network Configuration

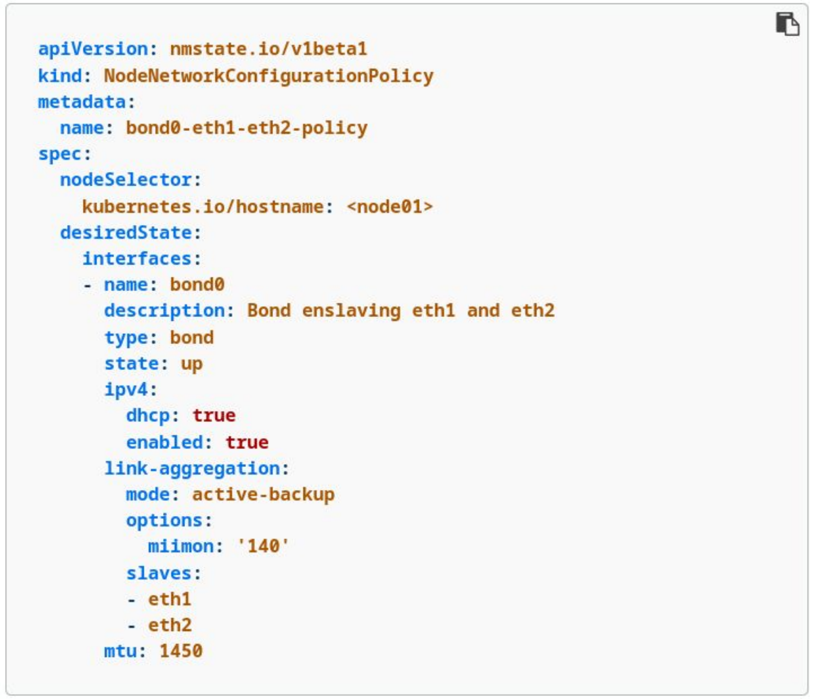
Find the bug



```
[root@worker-6 system-connections]# cat default_connection.nmconnection  | grep address1
address1=192.99.99.123/24,192.99.99.1
[root@worker-6 system-connections]# nmcli con mod 1969ebe8-b753-40fb-9e29-a73b7d075737 \
> ipv4.addresses 10.10.99.99\24 \
> ipv4.gateway 192.168.111.111
Error: failed to modify ipv4.addresses: invalid IP address: Invalid IPv4 address '10.10.99.9924'.
[root@worker-6 system-connections]# nmcli con mod 1969ebe8-b753-40fb-9e29-a73b7d075737 \
> ipv4.addresses 10.10.99.99/24 \
> ipv4.gateway 192.168.111.111
Warning: There is another connection with the name 'Wired Connection'. Reference the connection by it
s uuid '1969ebe8-b753-40fb-9e29-a73b7d075737'
[root@worker-6 system-connections]# cat default_connection.nmconnection  | grep address1
address1=10.10.99.99/24,192.168.111.111
[root@worker-6 system-connections]# 
```

# Network Configuration

## Declarative, k8s-managed



```yaml
apiVersion: nmstate.io/v1beta1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond0-eth1-eth2-policy
spec:
  nodeSelector:
    kubernetes.io/hostname: <node01>
  desiredState:
    interfaces:
    - name: bond0
      description: Bond enslaving eth1 and eth2
      type: bond
      state: up
      ipv4:
        dhcp: true
        enabled: true
      link-aggregation:
        mode: active-backup
        options:
          miimon: '140'
        slaves:
        - eth1
        - eth2
      mtu: 1450
```

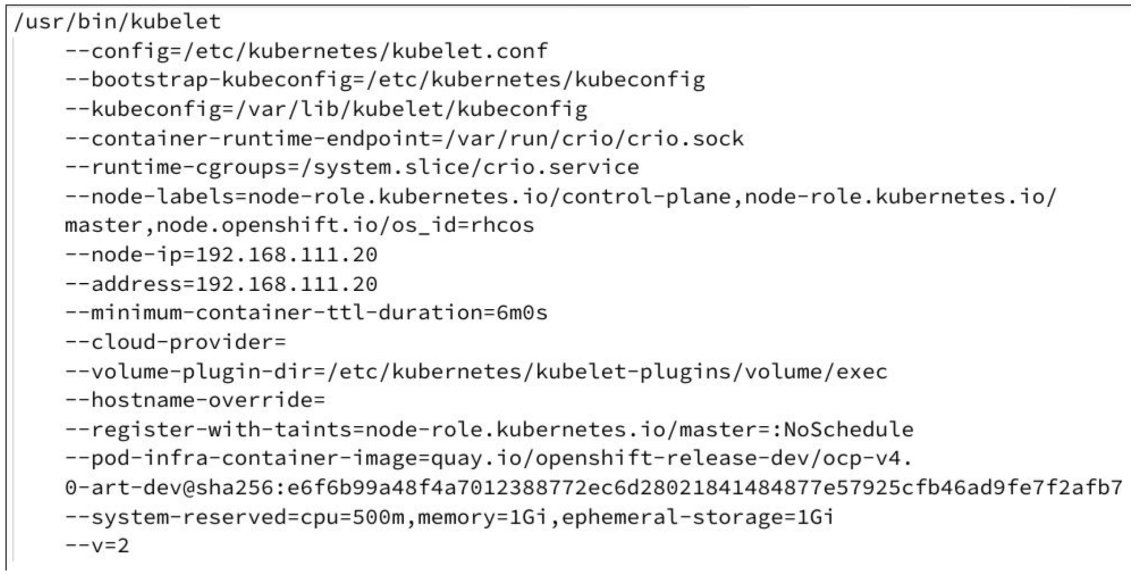
- A way to configure NICs via Kubernetes CRD
- Declarative, per-node

## Demo

…

# Node IP

## What is that?

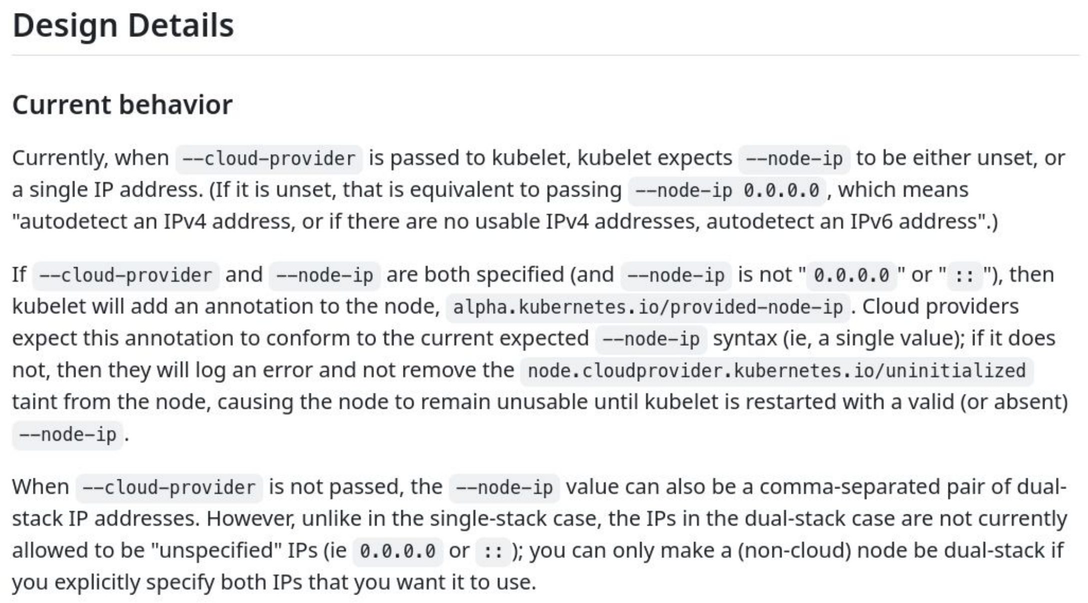


```
/usr/bin/kubelet
    --config=/etc/kubernetes/kubelet.conf
    --bootstrap-kubeconfig=/etc/kubernetes/kubeconfig
    --kubeconfig=/var/lib/kubelet/kubeconfig
    --container-runtime-endpoint=/var/run/crio/crio.sock
    --runtime-cgroups=/system.slice/crio.service
    --node-labels=node-role.kubernetes.io/control-plane,node-role.kubernetes.io/
    master,node.openshift.io/os_id=rhcos
    --node-ip=192.168.111.20
    --address=192.168.111.20
    --minimum-container-ttl-duration=6m0s
    --cloud-provider=
    --volume-plugin-dir=/etc/kubernetes/kubelet-plugins/volume/exec
    --hostname-override=
    --register-with-taints=node-role.kubernetes.io/master=:NoSchedule
    --pod-infra-container-image=quay.io/openshift-release-dev/ocp-v4.
    0-art-dev@sha256:e6f6b99a48f4a7012388772ec6d28021841484877e57925cfb46ad9fe7f2afb7
    --system-reserved=cpu=500m,memory=1Gi,ephemeral-storage=1Gi
    --v=2
```

- Kubelet needs to bind somewhere
  - Easy for single NIC setups (e.g. clouds)
- Upstream logic of selecting IP makes mistakes
- Custom component to override kubelet's logic

# Node IP

Configuration is complicated



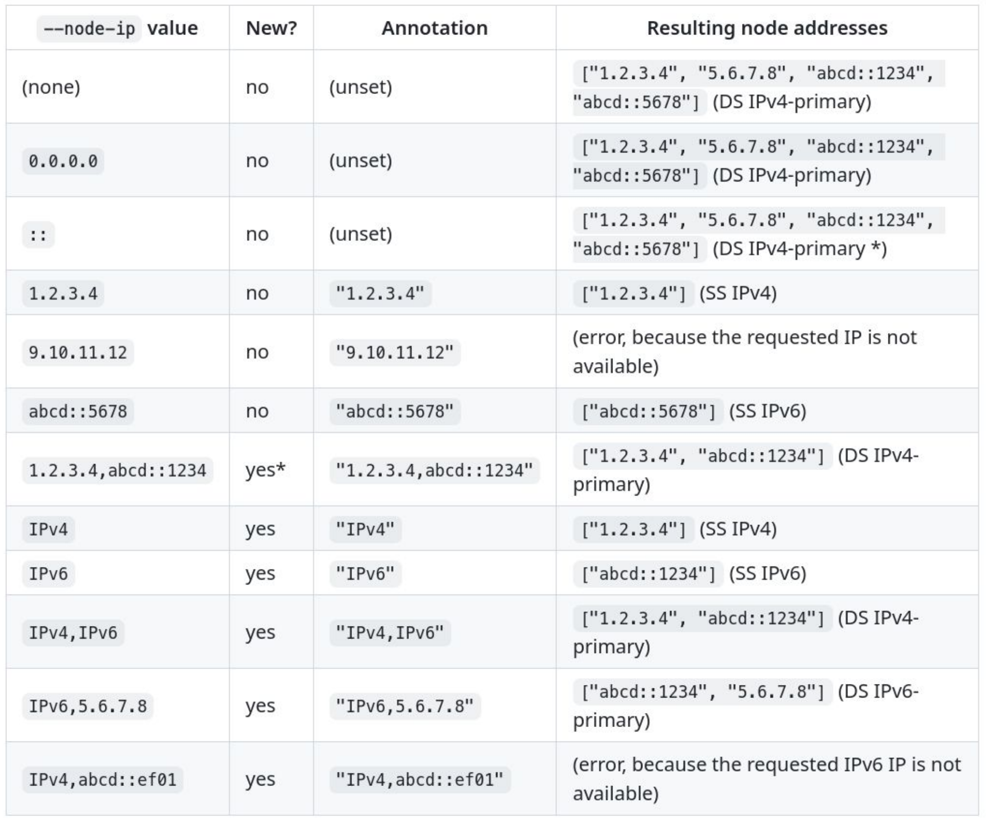
## Design Details

### Current behavior

Currently, when `--cloud-provider` is passed to kubelet, kubelet expects `--node-ip` to be either unset, or a single IP address. (If it is unset, that is equivalent to passing `--node-ip 0.0.0.0`, which means "autodetect an IPv4 address, or if there are no usable IPv4 addresses, autodetect an IPv6 address".)

If `--cloud-provider` and `--node-ip` are both specified (and `--node-ip` is not "`0.0.0.0`" or "`::`"), then kubelet will add an annotation to the node, `alpha.kubernetes.io/provided-node-ip`. Cloud providers expect this annotation to conform to the current expected `--node-ip` syntax (ie, a single value); if it does not, then they will log an error and not remove the `node.cloudprovider.kubernetes.io/uninitialized` taint from the node, causing the node to remain unusable until kubelet is restarted with a valid (or absent) `--node-ip`.

When `--cloud-provider` is not passed, the `--node-ip` value can also be a comma-separated pair of dual-stack IP addresses. However, unlike in the single-stack case, the IPs in the dual-stack case are not currently allowed to be "unspecified" IPs (ie `0.0.0.0` or `::`); you can only make a (non-cloud) node be dual-stack if you explicitly specify both IPs that you want it to use.



| `--node-ip` value | New? | Annotation | Resulting node addresses |
|---|---|---|---|
| (none) | no | (unset) | `["1.2.3.4", "5.6.7.8", "abcd::1234", "abcd::5678"]` (DS IPv4-primary) |
| `0.0.0.0` | no | (unset) | `["1.2.3.4", "5.6.7.8", "abcd::1234", "abcd::5678"]` (DS IPv4-primary) |
| `::` | no | (unset) | `["1.2.3.4", "5.6.7.8", "abcd::1234", "abcd::5678"]` (DS IPv4-primary *) |
| `1.2.3.4` | no | `"1.2.3.4"` | `["1.2.3.4"]` (SS IPv4) |
| `9.10.11.12` | no | `"9.10.11.12"` | (error, because the requested IP is not available) |
| `abcd::5678` | no | `"abcd::5678"` | `["abcd::5678"]` (SS IPv6) |
| `1.2.3.4,abcd::1234` | yes* | `"1.2.3.4,abcd::1234"` | `["1.2.3.4", "abcd::1234"]` (DS IPv4-primary) |
| `IPv4` | yes | `"IPv4"` | `["1.2.3.4"]` (SS IPv4) |
| `IPv6` | yes | `"IPv6"` | `["abcd::1234"]` (SS IPv6) |
| `IPv4,IPv6` | yes | `"IPv4,IPv6"` | `["1.2.3.4", "abcd::1234"]` (DS IPv4-primary) |
| `IPv6,5.6.7.8` | yes | `"IPv6,5.6.7.8"` | `["abcd::1234", "5.6.7.8"]` (DS IPv6-primary) |
| `IPv4,abcd::ef01` | yes | `"IPv4,abcd::ef01"` | (error, because the requested IPv6 IP is not available) |

## Node IP

### Basic principles

- Kubelet needs to listen on IP from the same subnet as VIP
- IP from VIP subnet doesn’t always exist…
    - E.g. “remote worker nodes” (some remote distant L2)
    - E.g. deployments without VIP
- … but a default route should always exist

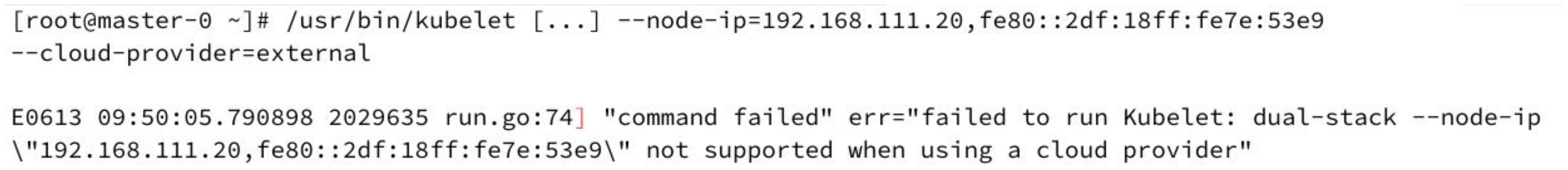
## Node IP

### The hard part

- What if...
  - your server has lots of IPs in a single subnet ?
  - your environment doesn't have any default gateway ?
  - you want kubelet to bind to >1 IP address ?
  - your IPv6 address is not really IPv6 ?

# Node IP

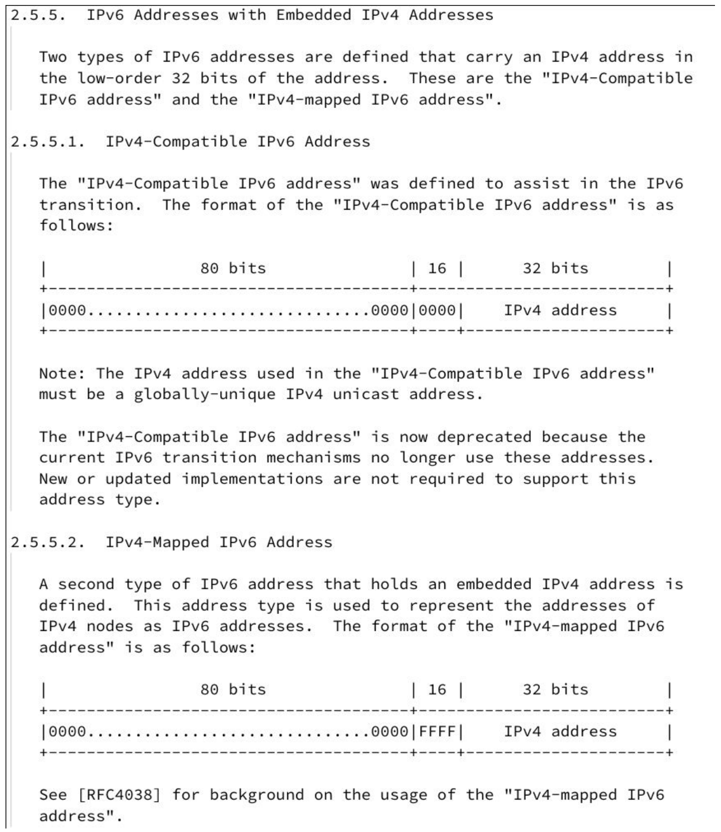
Configuration is (still) complicated



```
[root@master-0 ~]# /usr/bin/kubelet [...] --node-ip=192.168.111.20,fe80::2df:18ff:fe7e:53e9
--cloud-provider=external

E0613 09:50:05.790898 2029635 run.go:74] "command failed" err="failed to run Kubelet: dual-stack --node-ip
\"192.168.111.20,fe80::2df:18ff:fe7e:53e9\" not supported when using a cloud provider"
```

# Dual-stack addressing

IPv6, but is it really?



```
2.5.5.  IPv6 Addresses with Embedded IPv4 Addresses

   Two types of IPv6 addresses are defined that carry an IPv4 address in
   the low-order 32 bits of the address.  These are the "IPv4-Compatible
   IPv6 address" and the "IPv4-mapped IPv6 address".

2.5.5.1.  IPv4-Compatible IPv6 Address

   The "IPv4-Compatible IPv6 address" was defined to assist in the IPv6
   transition.  The format of the "IPv4-Compatible IPv6 address" is as
   follows:

   |                80 bits               | 16 |      32 bits        |
   +--------------------------------------+--------------------------+
   |0000..............................0000|0000|    IPv4 address     |
   +--------------------------------------+----+---------------------+

   Note: The IPv4 address used in the "IPv4-Compatible IPv6 address"
   must be a globally-unique IPv4 unicast address.

   The "IPv4-Compatible IPv6 address" is now deprecated because the
   current IPv6 transition mechanisms no longer use these addresses.
   New or updated implementations are not required to support this
   address type.

2.5.5.2.  IPv4-Mapped IPv6 Address

   A second type of IPv6 address that holds an embedded IPv4 address is
   defined.  This address type is used to represent the addresses of
   IPv4 nodes as IPv6 addresses.  The format of the "IPv4-mapped IPv6
   address" is as follows:

   |                80 bits               | 16 |      32 bits        |
   +--------------------------------------+--------------------------+
   |0000..............................0000|FFFF|    IPv4 address     |
   +--------------------------------------+----+---------------------+

   See [RFC4038] for background on the usage of the "IPv4-mapped IPv6
   address".
```

- ::<ipv4-address>/96
- ::FFFF:<ipv4-address>/96
- IPv6 according to most programming languages
    - they check for colon

## Dual-stack addressing

IPv6, let me break netcat

```
++ timeout 2s nc -l ::ffff:192.168.0.14 53604
Ncat: bind to ::ffff:192.168.0.14:53604: Invalid argument. QUITTING.
```

```
$ strace nc -l ::ffff:192.168.0.14 53604
[...]
socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP) = 3
setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
setsockopt(3, SOL_IPV6, IPV6_V6ONLY, [1], 4) = 0
bind(3, {sa_family=AF_INET6, sin6_port=htons(53604), sin6_flowinfo=htonl(0),
inet_pton(AF_INET6, "::ffff:192.168.0.14", &sin6_addr), sin6_scope_id=0}, 28) =
-1 EINVAL (Invalid argument)

strace nc -4 -l ::ffff:192.168.0.14 53604
[...]
socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
bind(3, {sa_family=AF_INET, sin_port=htons(53604),
sin_addr=inet_addr("192.168.1.160")}, 16) = 0
```

```
if ip.startswith("::ffff")
    timeout 2s nc -4 -l "${ip}" ${random_port};
else
    timeout 2s nc -l "${ip}" ${random_port};
```

## Cloud or on-prem?

Pick your platform, choose wisely

# The End

@mko - kubernetes.slack.com

github.com/mkowalski
linkedin.com/in/mateuszkowalski

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat