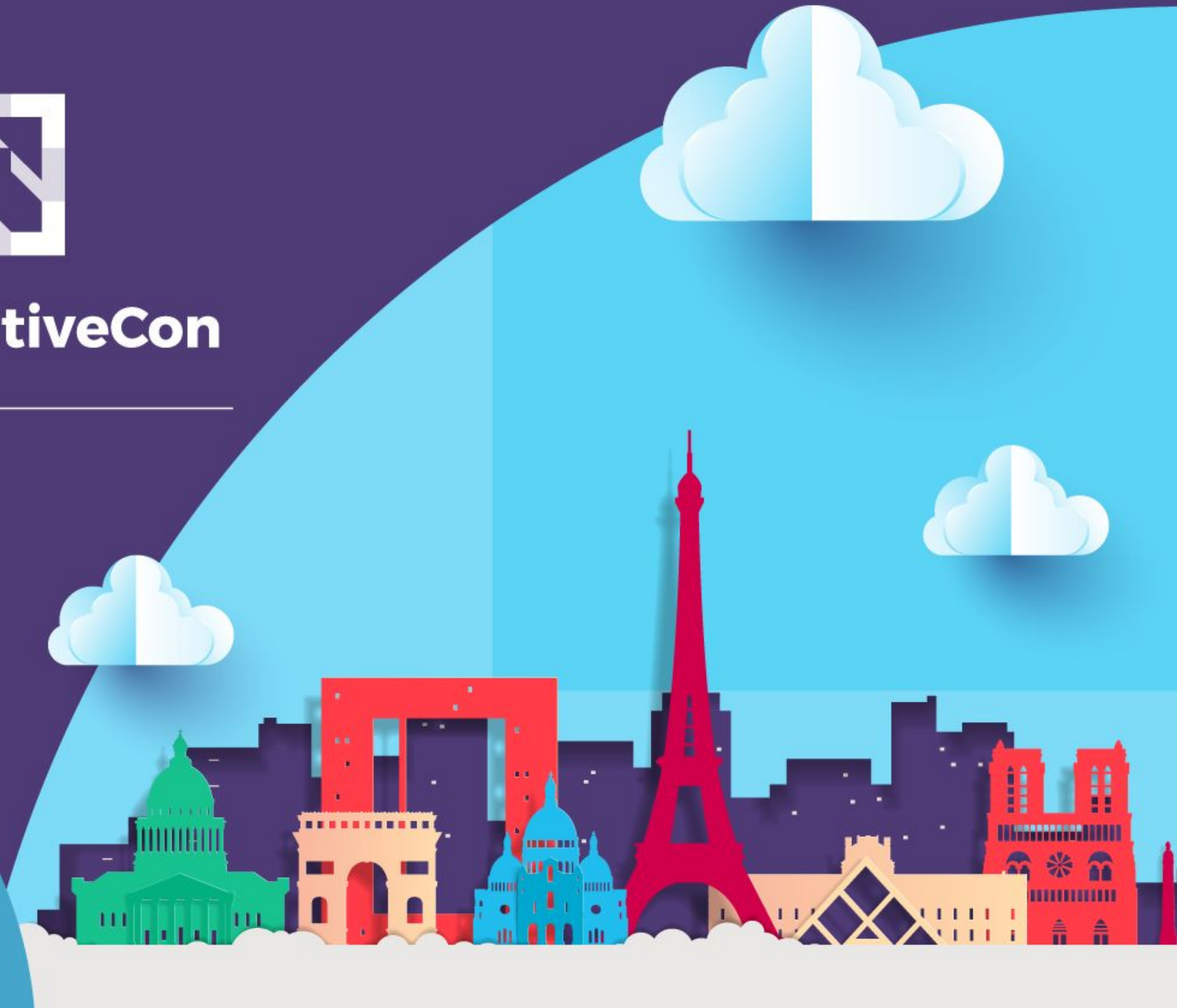
KubeCon | CloudNativeCon

Europe 2024



From CNI Zero to CNI Hero:
A Kubernetes Networking Tutorial Using CNI

Tomofumi Hayashi & Doug Smith, Red Hat

Introductions

@s1061123

Tomofumi Hayashi

- OpenShift Engineering
- CNI maintainer
- Multus CNI maintainer
- Network Plumbing Working Group member

@dougbtv

Doug Smith

- OpenShift Engineering
- Multus CNI maintainer
- Network Plumbing Working Group member
- Blog: https://dougbtv.com

What’s on the agenda Today?

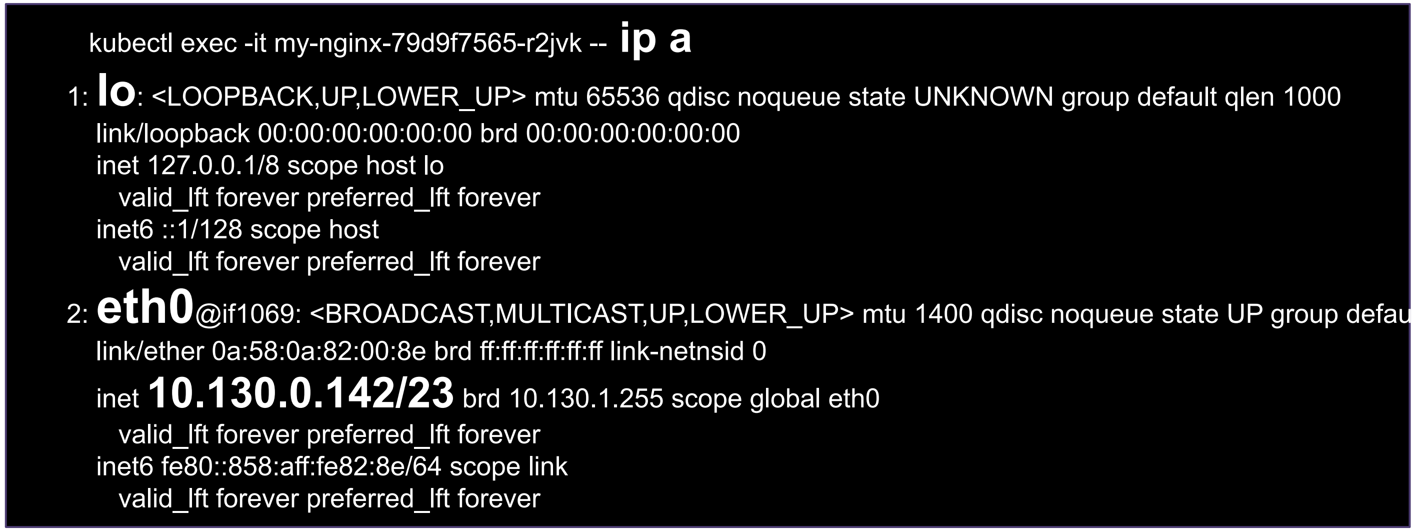
- An Intro to CNI
- All the configs!
- Developing CNI plugins
- Hands on tutorial!
- Troubleshooting
- CNI Community

An Intro to CNI

What does CNI do for you?

→ Provides network connectivity to your Kubernetes Pods!

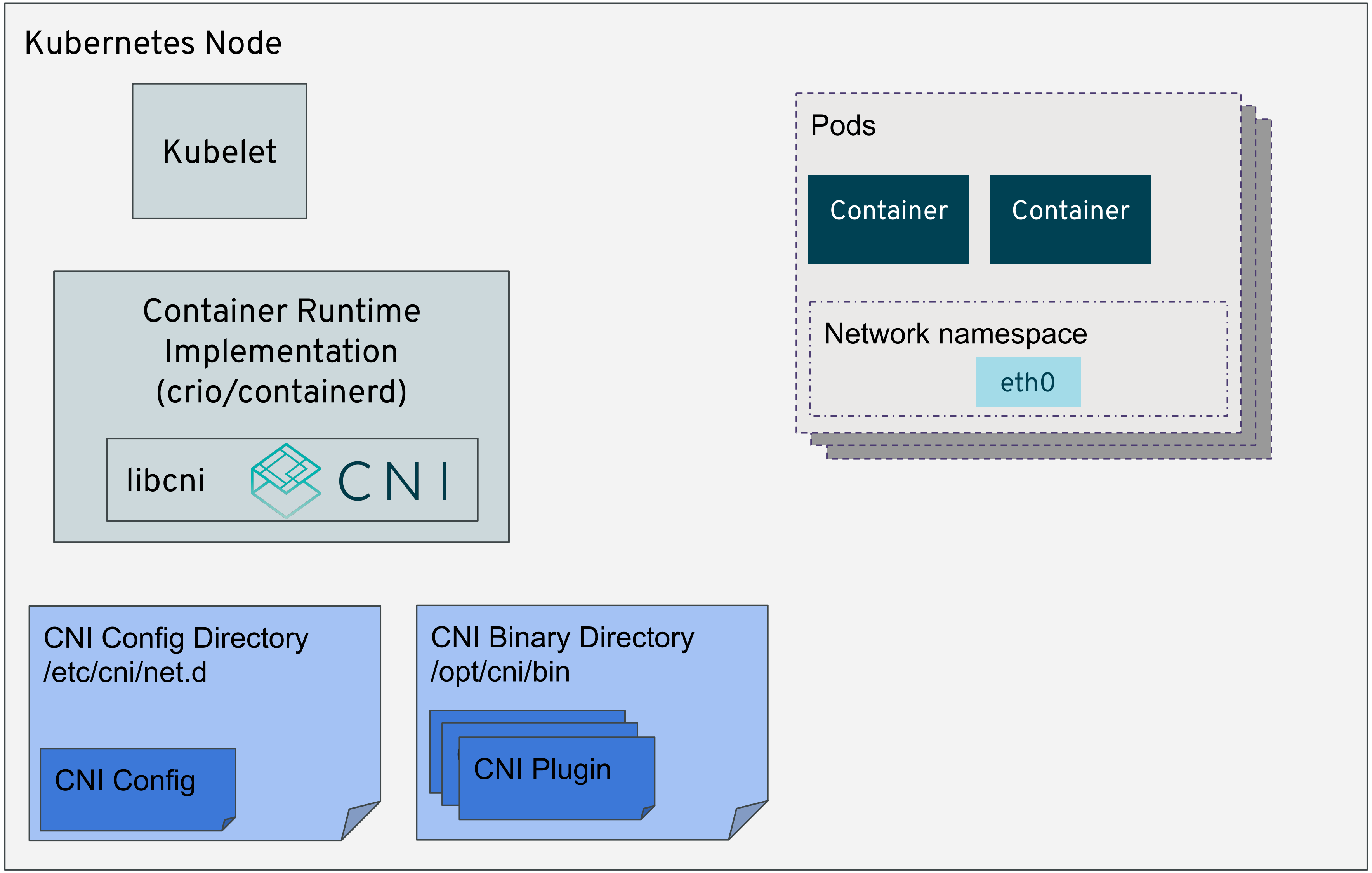
- Adding network interface to pods (i.e. eth0)



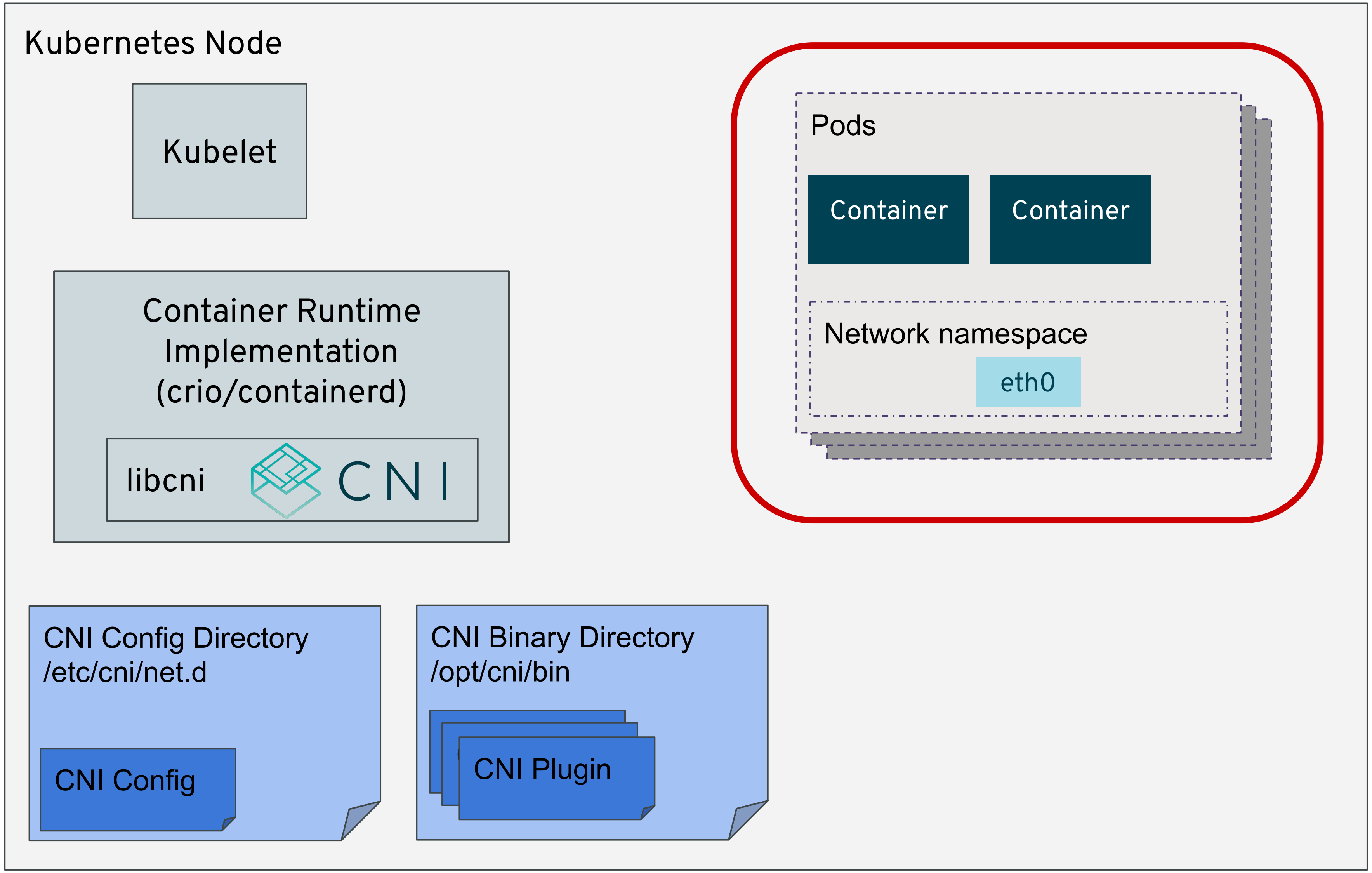
```
    kubectl exec -it my-nginx-79d9f7565-r2jvk -- ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0@if1069: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1400 qdisc noqueue state UP group defau
    link/ether 0a:58:0a:82:00:8e brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.130.0.142/23 brd 10.130.1.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::858:aff:fe82:8e/64 scope link
       valid_lft forever preferred_lft forever
```

- Assign IP addresses to pods
- Perform specialized setup required for your network!
  - Configure specific one (e.g. iptables/nft rules) to the interface
- And of course, clean up that work during pod deletion

CNI Anatomy: An overview

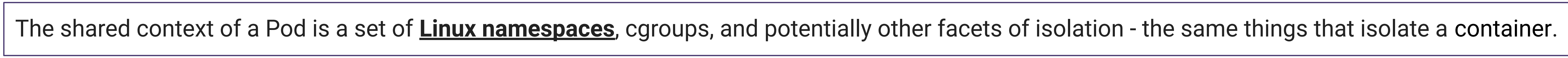

Kubernetes Node
Kubelet
Container Runtime Implementation (crio/containerd)
libcni
CNI
Pods
Container
Container
Network namespace
eth0
CNI Config Directory /etc/cni/net.d
CNI Config
CNI Binary Directory /opt/cni/bin
CNI Plugin

CNI Anatomy: An overview


Kubernetes Node
Kubelet
Container Runtime Implementation (crio/containerd)
libcni
CNI
Pods
Container
Container
Network namespace
eth0
CNI Config Directory /etc/cni/net.d
CNI Config
CNI Binary Directory /opt/cni/bin
CNI Plugin

What is the Pod, container and network?

- From https://kubernetes.io/docs/concepts/workloads/pods/



The shared context of a Pod is a set of **Linux namespaces**, cgroups, and potentially other facets of isolation - the same things that isolate a container.

Linux namespace

- Partitioning feature in Linux kernel
  - Mount (mnt)
  - Process ID (pid)
  - Network (net)
  - Inter-process communication (ipc), UTS, User ID, cgroup, Time...
- Container uses Linux namespace (e.g. pid, mnt and so on) to isolate resources from container hosts

Pod is not just a ‘Container’

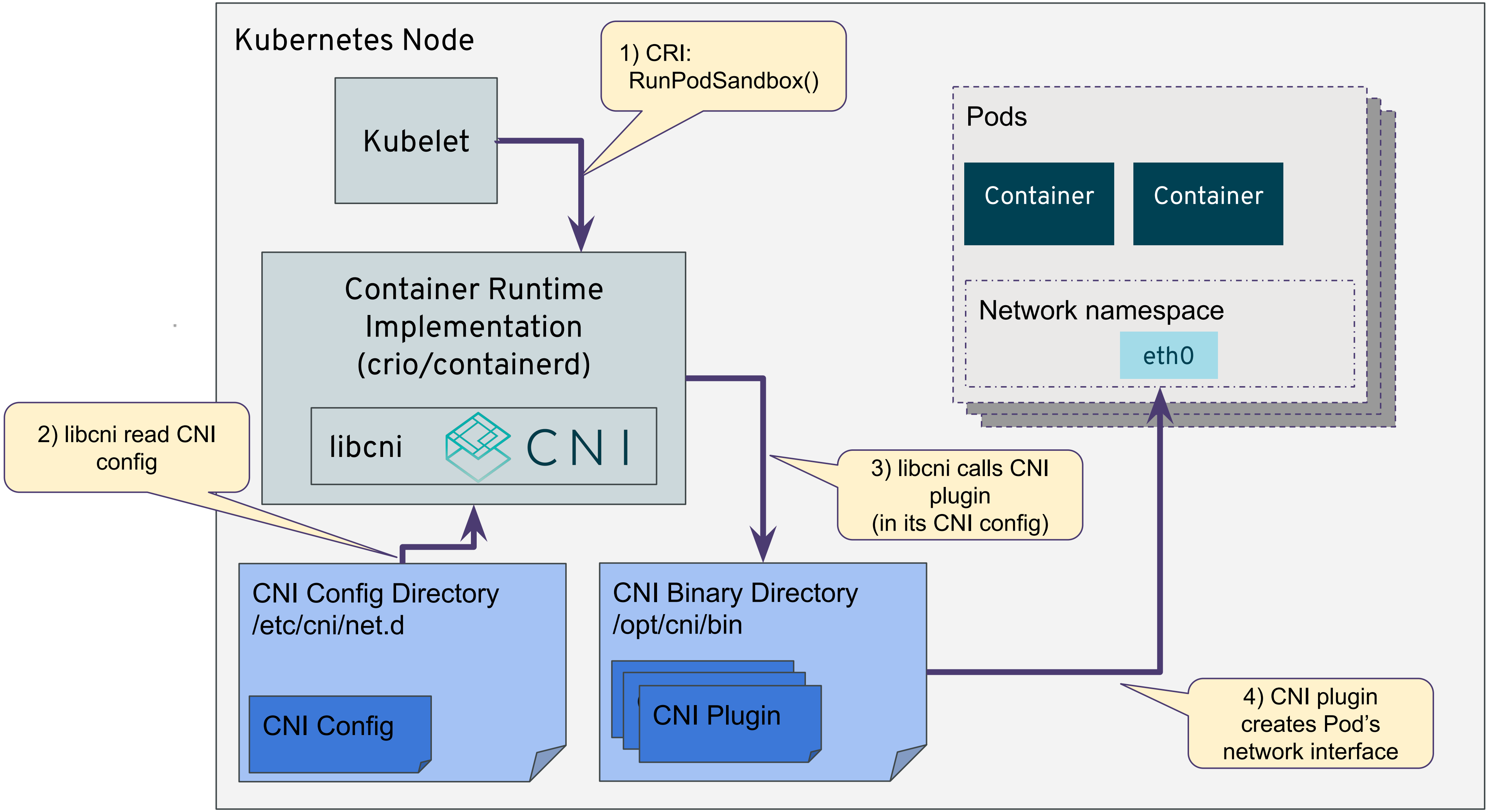
- Shares network namespaces (net)
  - Two interfaces: lo and eth0
- Multiple container is running (pid, mnt)

When does CNI Plugin work for you?

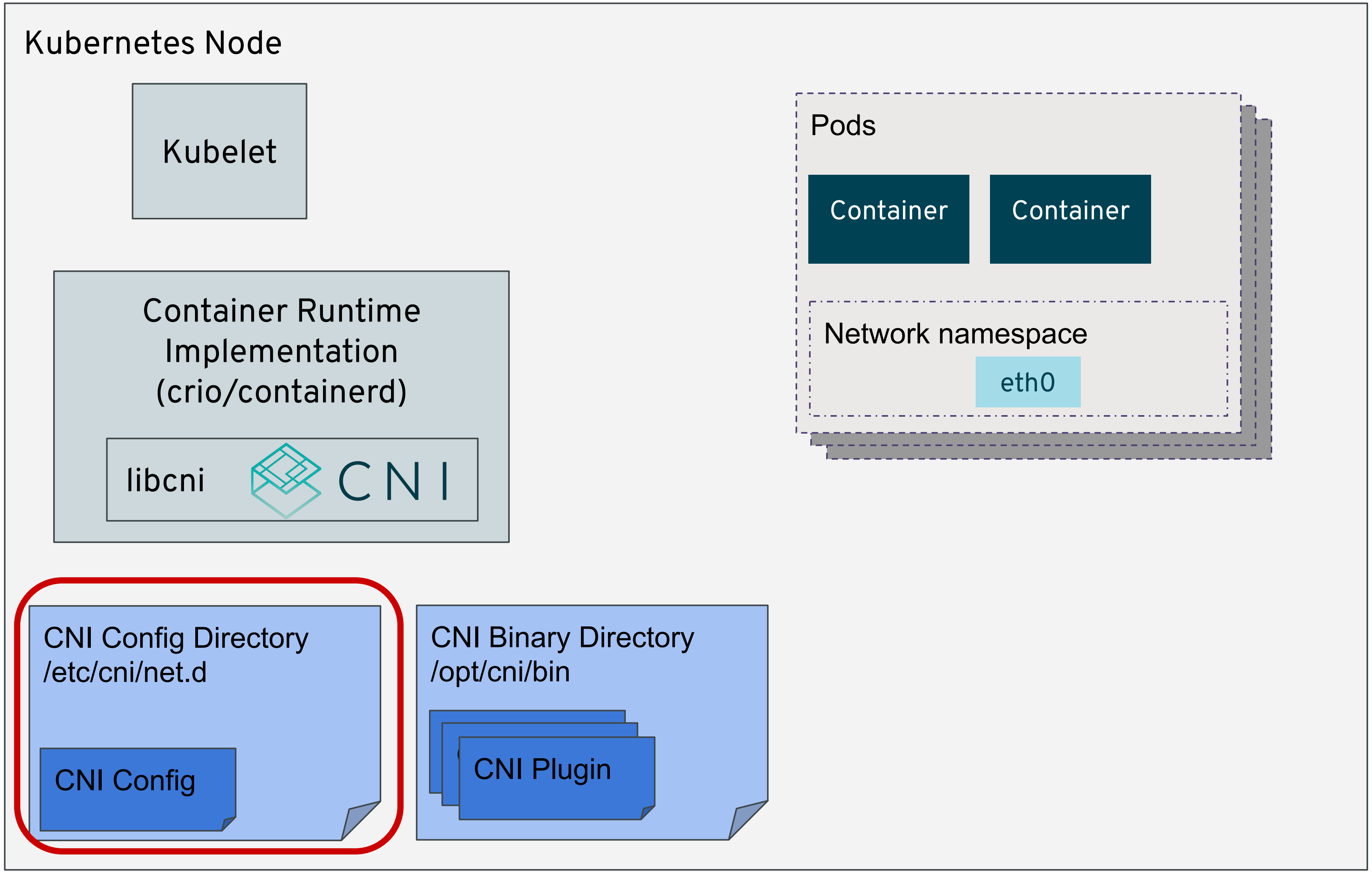
- CNI Provides “Plugin” architecture, hence network provider provides various CNI Plugins:
  - CNI Projects Reference CNI Plugins (i.e. macvlan/ipvlan/host-local/…)
  - Third Party CNI Plugins (i.e. Calico/Cilium/ovn-kubernetes/…)
- CNI Plugin will be executed by container runtime at
  - Pod creation (i.e. ADD)
  - Pod deletion (i.e. DELETE)

System flows: from kubelet to CNI

KubeCon CloudNativeCon Europe 2024


Kubernetes Node
Kubelet
1) CRI: RunPodSandbox()
Container Runtime Implementation (crio/containerd)
libcni
CNI
2) libcni read CNI config
Pods
Container
Container
Network namespace
eth0
3) libcni calls CNI plugin (in its CNI config)
CNI Config Directory /etc/cni/net.d
CNI Config
CNI Binary Directory /opt/cni/bin
CNI Plugin
4) CNI plugin creates Pod's network interface

Next: Let’s focus on CNI config!



All the configs!

What does CNI config look like?

- Text, JSON format file (case sensitive)

WAIT, WHAT!? CNI configurations are JSON!?!

Everything in else in Kubernetes is YAML – why are CNI configurations in JSON?

CNI is independent of Kubernetes and is agnostic of “container orchestration engines”

“CNI concerns itself only with network connectivity of containers and removing allocated resources when the container is deleted. Because of this focus, CNI has a wide range of support and the specification is simple to implement”

-CNI README.md

What does CNI config look like?

- Text, JSON format file (case sensitive)
- Filename should end with '**.conflist'** (sometimes .conf but we use '**.conflist'**)
  - 01-foobar.**conflist** → OK!
  - 01-foobar.json → No Good!
  - 01-foobar.conf → OK! (but a bit old style…)
- Container Runtime (containerd/crio) takes only one file from config directory
  - First file by sorted ASCII!
- CNI specification specifies CNI config and it is updated several times
  - Spec version history: 0.1.0 → 0.2.0 → 0.3.0/0.3.1 → 0.4.0 → **1.0.0**
  - Almost same syntax, but a bit different among version
  - Let's use 1.0.0 in this presentation!

Latest!

CNI Configuration example

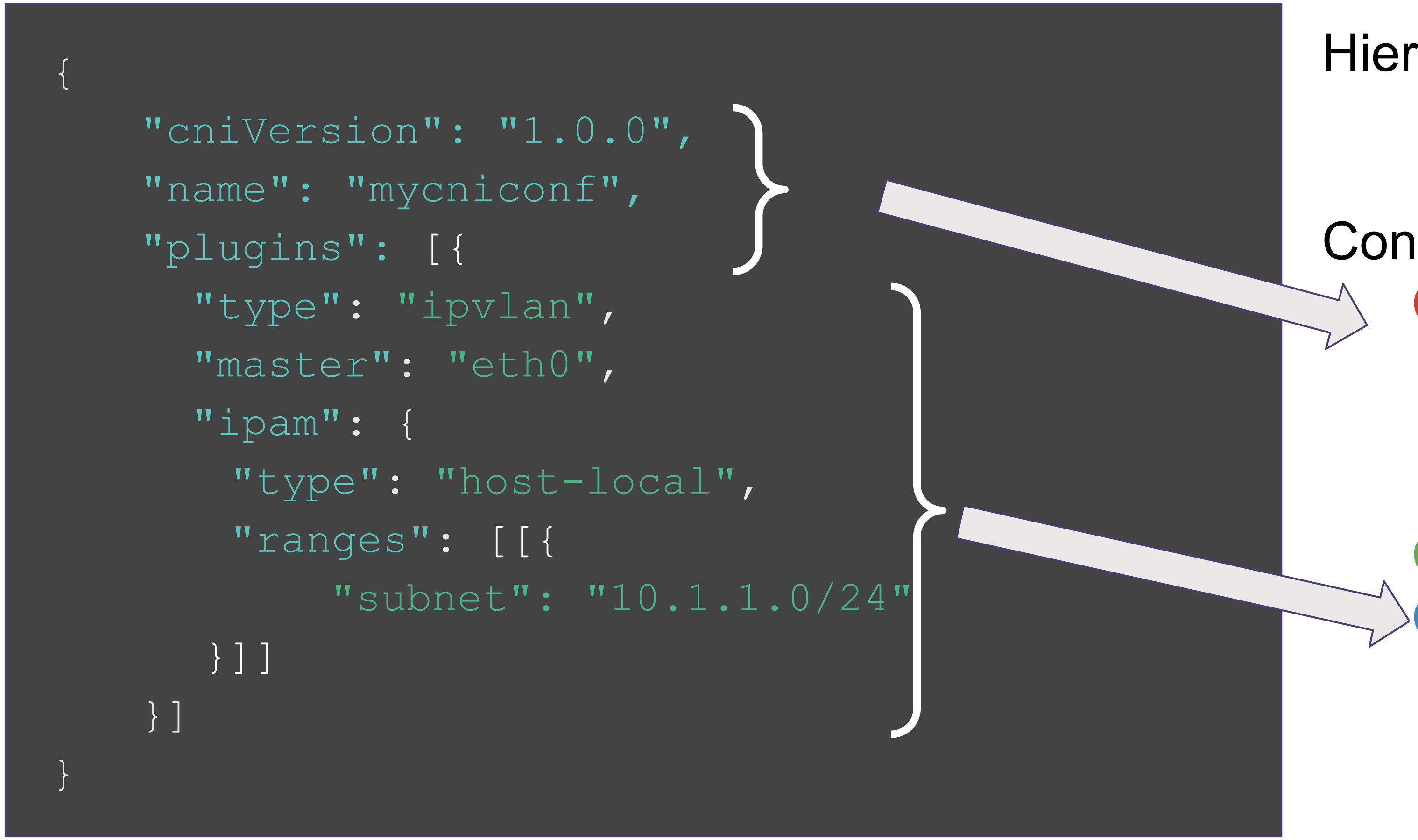
```
{
    "cniVersion": "1.0.0",
    "name": "mycniconf",
    "plugins": [{
      "type": "ipvlan",
      "master": "eth0",
      "ipam": {
        "type": "host-local",
        "ranges": [[{
            "subnet": "10.1.1.0/24"
        }]]
    }]
}
```

Hierarchical structure by json

Contains

- Config for CNI Runtime
- Config for Container Runtime
- Config for CNI Plugins

CNI Configuration example



```
{
    "cniVersion": "1.0.0",
    "name": "mycniconf",
    "plugins": [{
      "type": "ipvlan",
      "master": "eth0",
      "ipam": {
        "type": "host-local",
        "ranges": [[{
            "subnet": "10.1.1.0/24"
        }]]
    }]
}
```

Hierarchical structure by json

Contains

- Config for CNI Runtime/Plugins
  - "cniVersion": Specifies CNI version
  - "name": CNI Config name
  - "plugins": Specifies plugin configs
- Config for Container Runtime
- Config for CNI Plugins
  - "type": Which CNI Plugin is used
  - "master": master interface of ipvlan (only for ipvlan CNI plugin!)
  - "ipam": Specifies IPAM (IP Address Management)

CNI Configuration example (IPAM)

```
{
    "cniVersion": "1.0.0",
    "name": "myconflist",
    "plugins": [{
      "type": "ipvlan",
      "master": "eth0",
      "ipam": {
        "type": "host-local",
        "ranges": [[{
            "subnet": "10.1.1.0/24"
        }]]
    }]
}
```

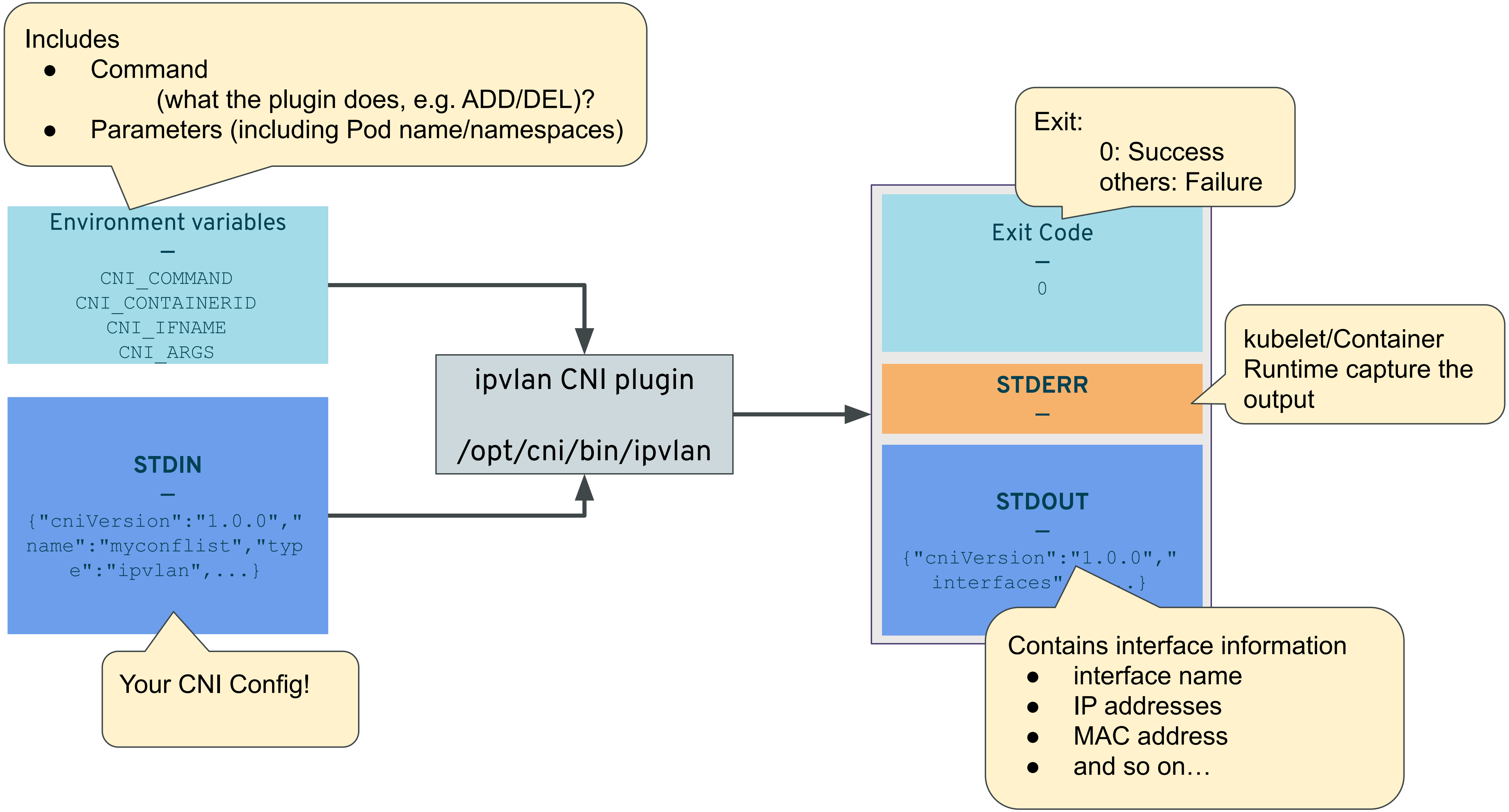
“ipvlan” CNI Plugin delegates IP address management to other IPAM CNI Plugin, hence “ipam” section is ‘another CNI config’ for its IP address configuration

Hierarchical structure by json

Contains

- Config for CNI Runtime/Plugins
- Config for Container Runtime
- Config for CNI Plugins
  - “type”: Which CNI Plugin is used for IPAM
  - “ranges”: Specify range objects (for IP ranges)
  - “subnet”: Specify subnet of IP address space (only for ‘host-local’ CNI plugin!)

What goes into a CNI plugin call?



Includes
- Command
  (what the plugin does, e.g. ADD/DEL)?
- Parameters (including Pod name/namespaces)

Environment variables
—
```
CNI_COMMAND
CNI_CONTAINERID
CNI_IFNAME
CNI_ARGS
```

STDIN
—
```
{"cniVersion":"1.0.0","name":"myconflist","type":"ipvlan",...}
```

Your CNI Config!

ipvlan CNI plugin

/opt/cni/bin/ipvlan

Exit:
    0: Success
    others: Failure

Exit Code
—
0

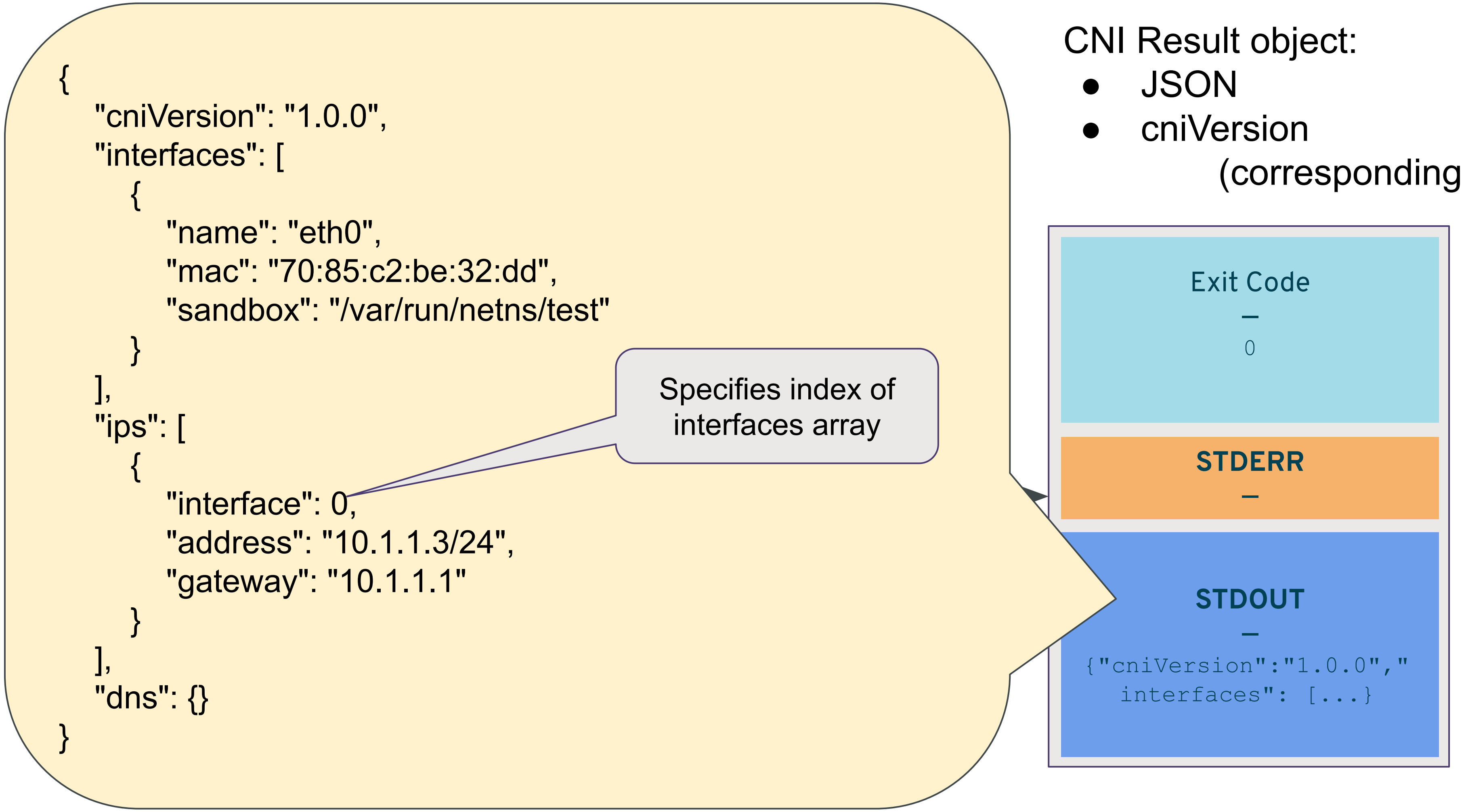
STDERR
—

kubelet/Container Runtime capture the output

STDOUT
—
```
{"cniVersion":"1.0.0","
interfaces"      .}
```

Contains interface information
- interface name
- IP addresses
- MAC address
- and so on…

What goes into a CNI plugin call?



```
{
    "cniVersion": "1.0.0",
    "interfaces": [
        {
            "name": "eth0",
            "mac": "70:85:c2:be:32:dd",
            "sandbox": "/var/run/netns/test"
        }
    ],
    "ips": [
        {
            "interface": 0,
            "address": "10.1.1.3/24",
            "gateway": "10.1.1.1"
        }
    ],
    "dns": {}
}
```

Specifies index of interfaces array

Exit Code
—
0

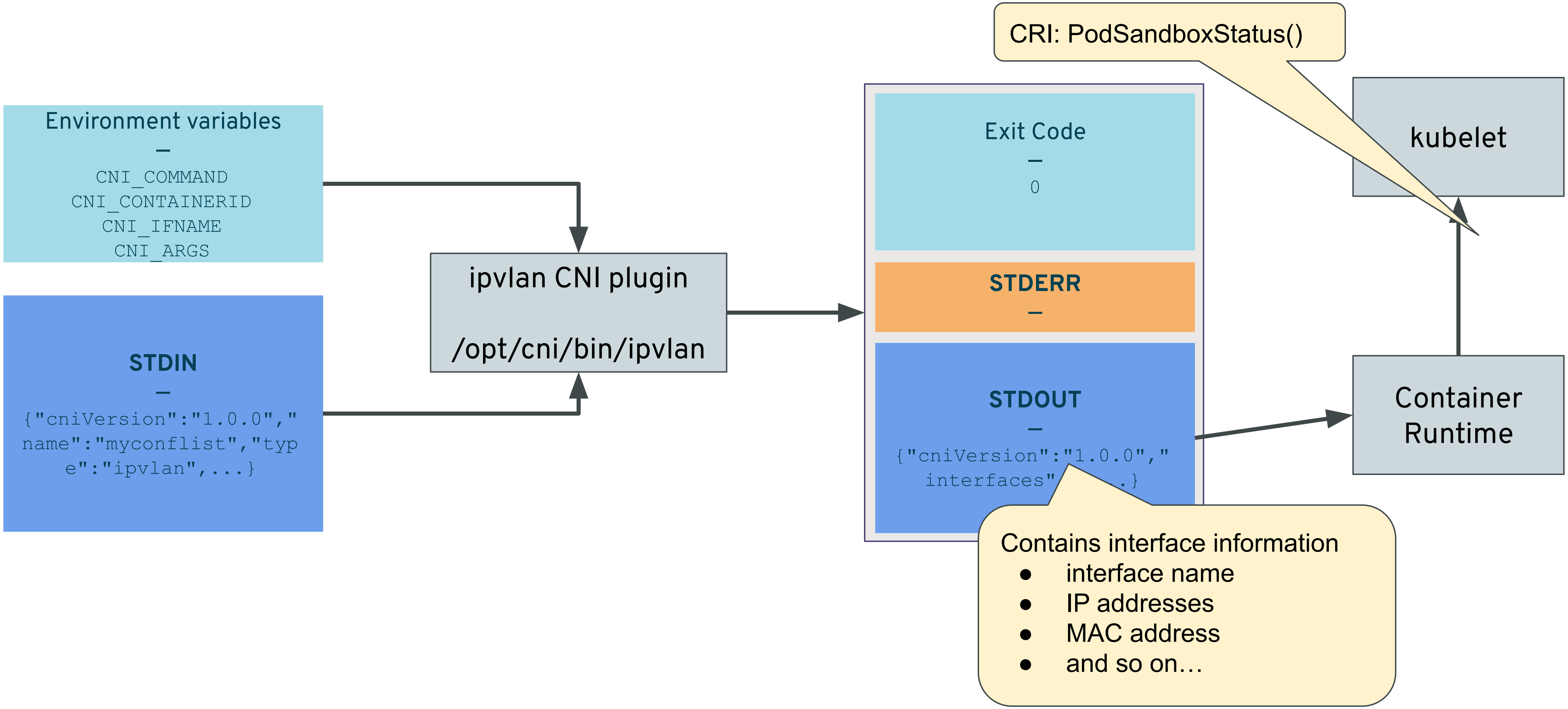
STDERR
—

STDOUT
—
{"cniVersion":"1.0.0"," interfaces": [...]}

CNI Result object:

- JSON
- cniVersion
  (corresponding to CNI config’s one)

What goes into a CNI plugin call?



Environment variables
—
CNI_COMMAND
CNI_CONTAINERID
CNI_IFNAME
CNI_ARGS

STDIN
—
{"cniVersion":"1.0.0","name":"myconflist","type":"ipvlan",...}

ipvlan CNI plugin

/opt/cni/bin/ipvlan

Exit Code
—
0

STDERR
—

STDOUT
—
{"cniVersion":"1.0.0","interfaces"...}

Container Runtime

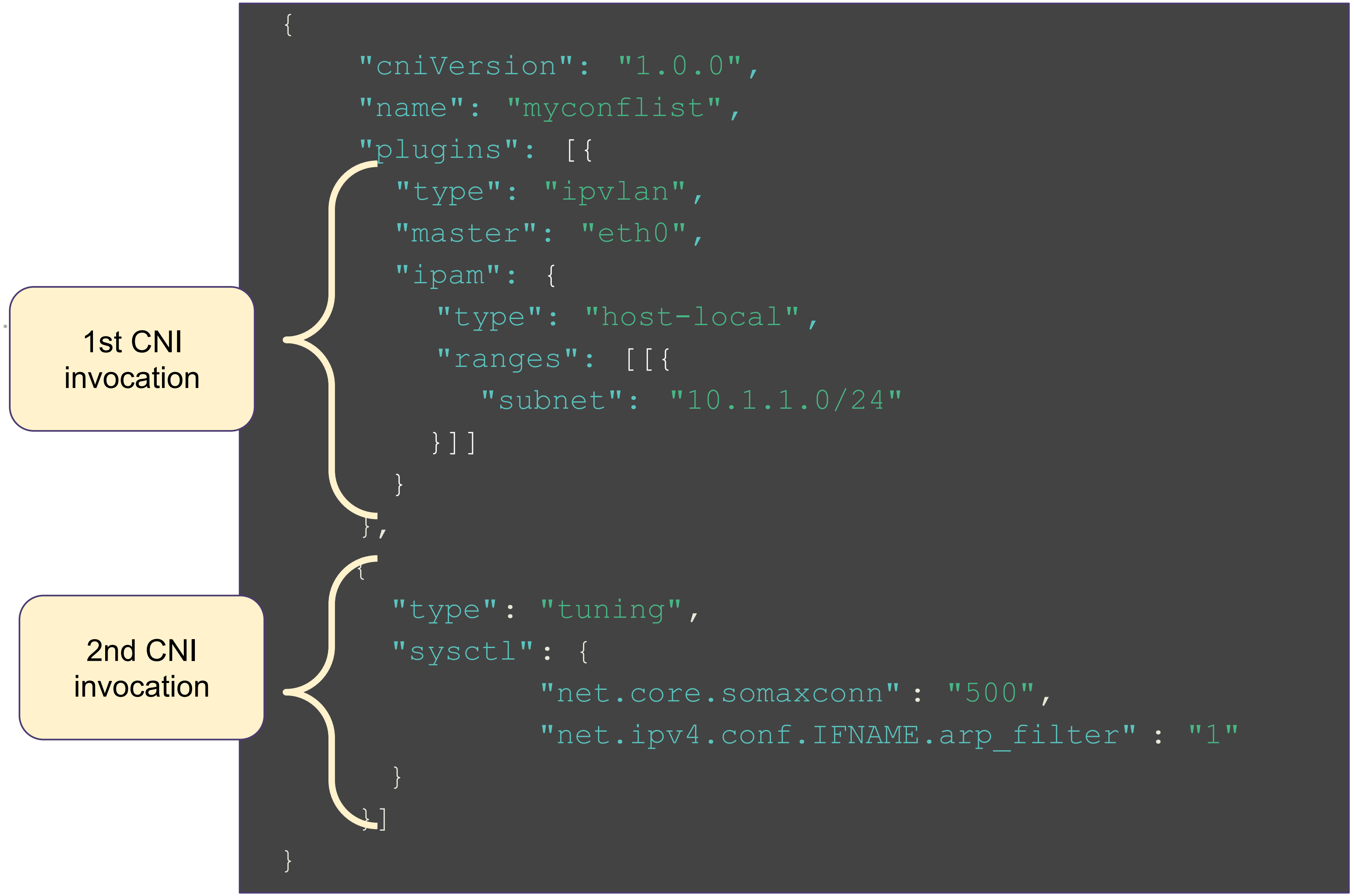
kubelet

CRI: PodSandboxStatus()

Contains interface information
- interface name
- IP addresses
- MAC address
- and so on…

CNI Configuration (plugin chains)

CNI config allows “chaining” for more config by multiple plugins!



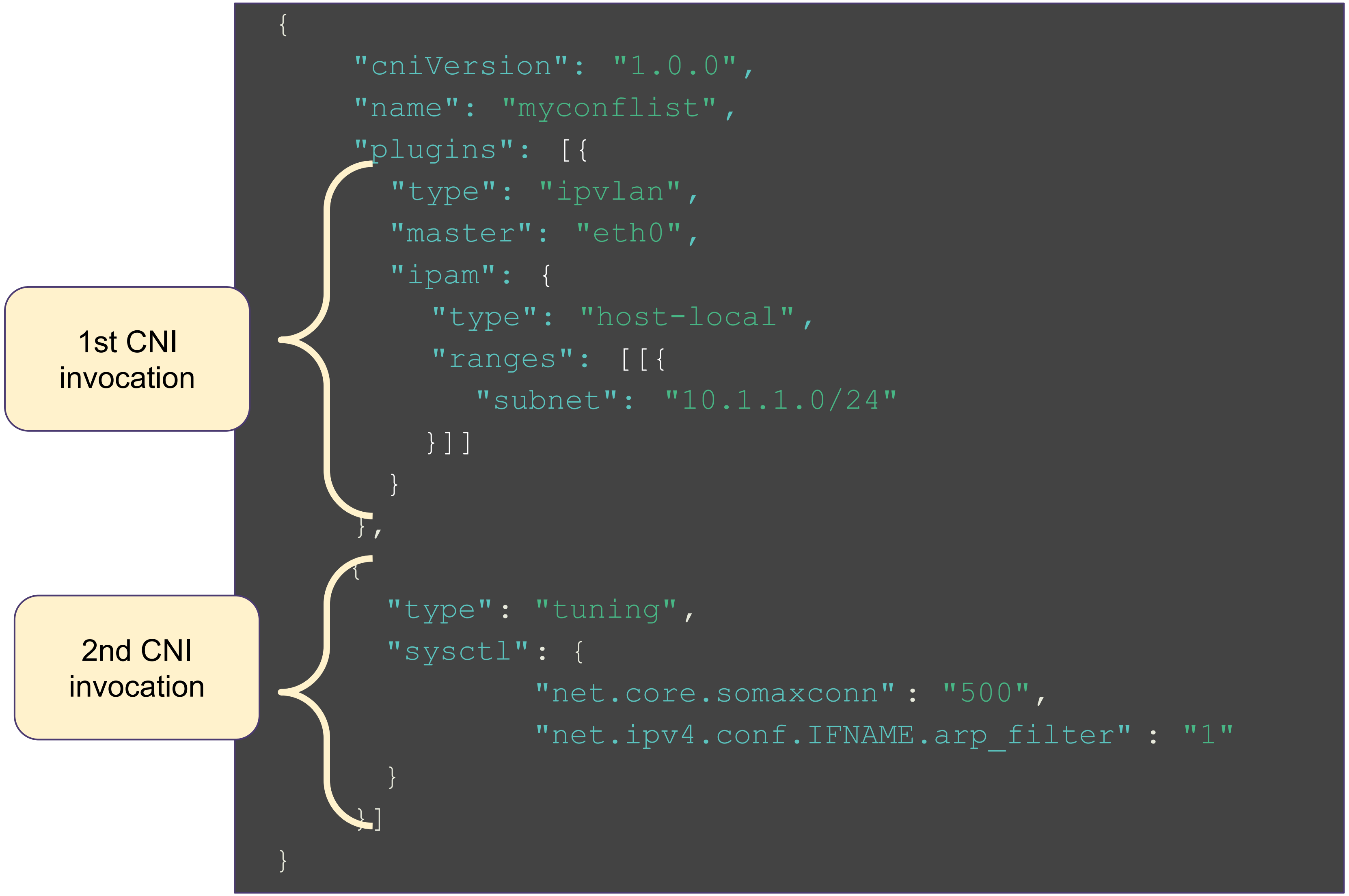
```
{
    "cniVersion": "1.0.0",
    "name": "myconflist",
    "plugins": [{
      "type": "ipvlan",
      "master": "eth0",
      "ipam": {
        "type": "host-local",
        "ranges": [[{
          "subnet": "10.1.1.0/24"
        }]]
      }
    },
    {
      "type": "tuning",
      "sysctl": {
              "net.core.somaxconn" : "500",
              "net.ipv4.conf.IFNAME.arp_filter" : "1"
      }
    }]
}
```

1st CNI invocation

2nd CNI invocation

CNI Configuration (plugin chains)

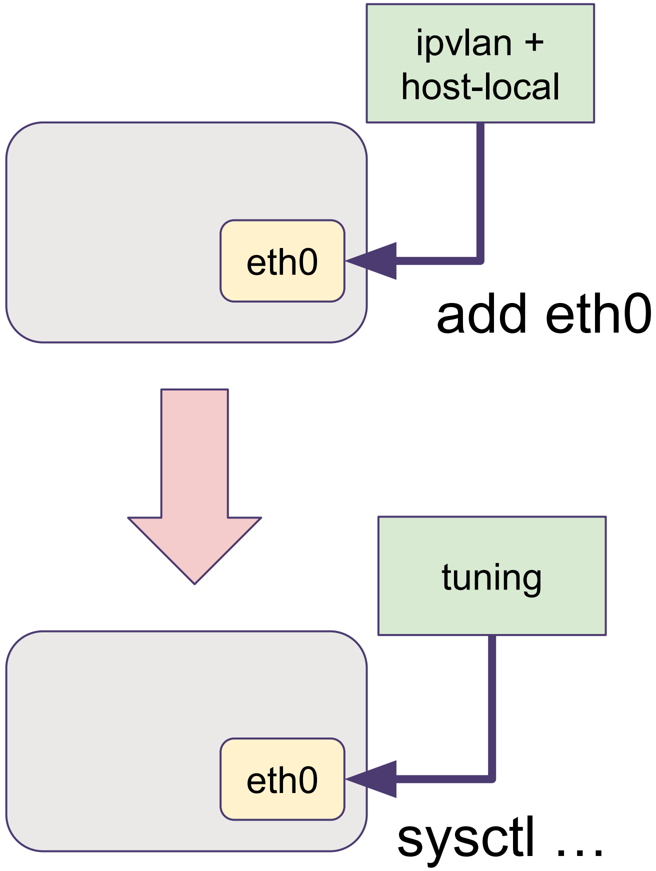
CNI config allows “chaining” for more config by multiple plugins!



```
{
    "cniVersion": "1.0.0",
    "name": "myconflist",
    "plugins": [{
      "type": "ipvlan",
      "master": "eth0",
      "ipam": {
        "type": "host-local",
        "ranges": [[{
          "subnet": "10.1.1.0/24"
        }]]
      }
    },
    {
      "type": "tuning",
      "sysctl": {
              "net.core.somaxconn" : "500",
              "net.ipv4.conf.IFNAME.arp_filter" : "1"
      }
    }]
}
```

1st CNI invocation

2nd CNI invocation

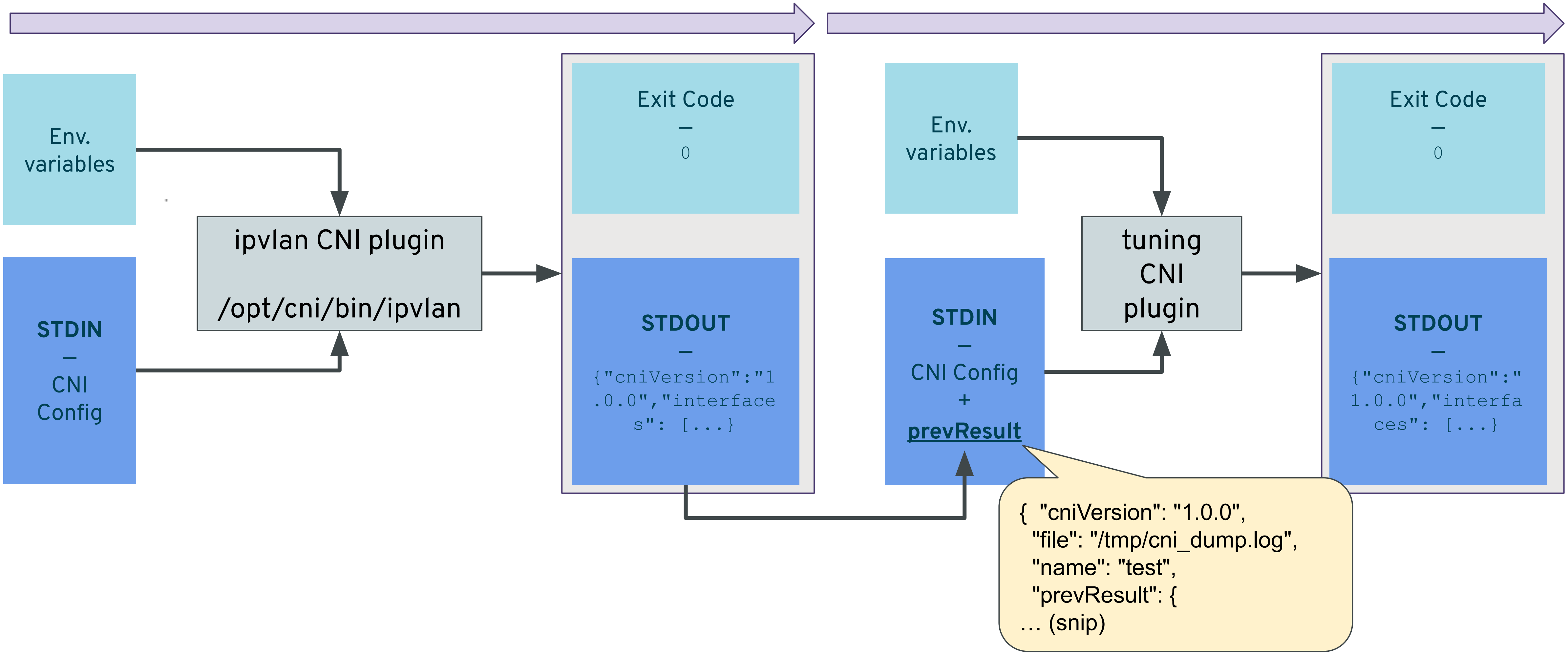


What goes into a plugin chain call?

ADD:

1st CNI call (ipvlan)

2nd CNI call (tuning)



Env. variables

STDIN – CNI Config

ipvlan CNI plugin

/opt/cni/bin/ipvlan

Exit Code – 0

STDOUT – {"cniVersion":"1.0.0","interfaces": [...]}

Env. variables

STDIN – CNI Config + prevResult

tuning CNI plugin

Exit Code – 0

STDOUT – {"cniVersion":"1.0.0","interfaces": [...]}

```
{  "cniVersion": "1.0.0",
   "file": "/tmp/cni_dump.log",
   "name": "test",
   "prevResult": {
… (snip)
```

How to call plugin chain from libcni?

CNI Config

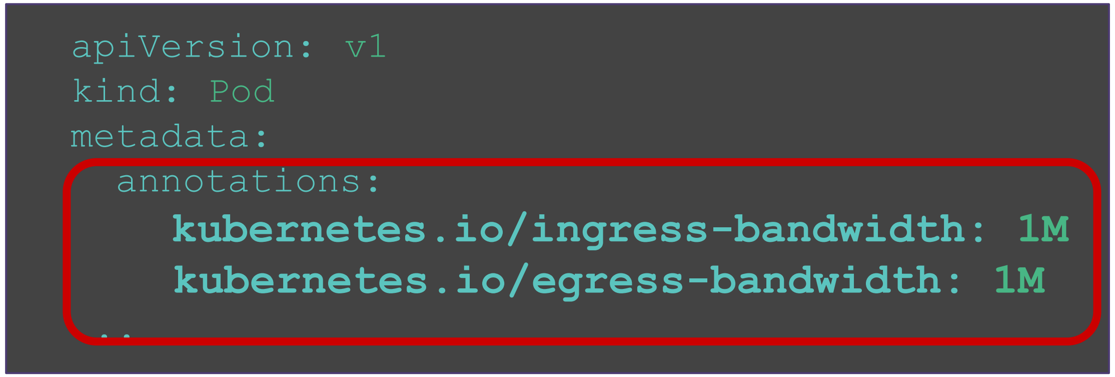
```
{
    "cniVersion": "1.0.0",
    "name": "myconflist",
    "plugins": [{
      "type": "ipvlan",
      "master": "eth0",
      "ipam": {
        "type": "host-local",
        "ranges": [[{
          "subnet": "10.1.1.0/24"
        }]]
      }
    },
    {
      "type": "tuning",
      "sysctl": {
              "net.core.somaxconn" : "500",
              "net.ipv4.conf.IFNAME.arp_filter" :
"1"
      }
    }]
```

Previous (ipvlan+host-local) results are contained in “prevResult” field!

Actual CNI config, consumed by “tuning”, given by libcni

```
{
  "cniVersion": "1.0.0",
  "name": "myconflist",
  "prevResult": {
    "cniVersion": "1.0.0",
    "interfaces": [
      {
        "name": "eth0",
        "mac": "52:54:00:1e:93:88",
        (snip)
    ],
    "ips": [
      {
        (snip)
      }
    ],
    "dns": {}
  },
  "sysctl": {
    "net.core.somaxconn": "500",
    "net.ipv4.conf.IFNAME.arp_filter": "1"
  },
  "type": "tuning"
}
```

Capabilities/RuntimeConfig



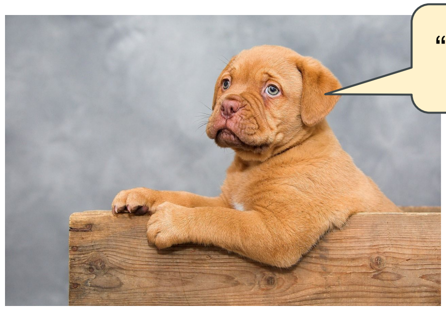
```
apiVersion: v1
kind: Pod
metadata:
  annotations:
     kubernetes.io/ingress-bandwidth: 1M
     kubernetes.io/egress-bandwidth: 1M
 ...
```

Hey, I have such Pod YAML file.
How to configure that?
It must touch with interface, right?

Capabilities/RuntimeConfig

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
     kubernetes.io/ingress-bandwidth: 1M
     kubernetes.io/egress-bandwidth: 1M
...
```

Hey, I have such Pod YAML file.
How to configure that?
It must touch with interface, right?



“Capabilities”/”RuntimeConfig” makes it possible!

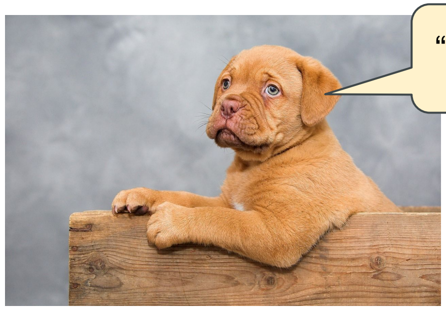
- The mechanism for container runtime/upper layer components (i.e. kubelets) to inject additional parameters to CNI Plugins
  - per-pod specific configs
  - “bandwidth”, “portmap” plugin is mainly used with Kubernetes (e.g. calico, flannel and so on)

Capabilities/RuntimeConfig

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubernetes.io/ingress-bandwidth: 1M
    kubernetes.io/egress-bandwidth: 1M
...
```

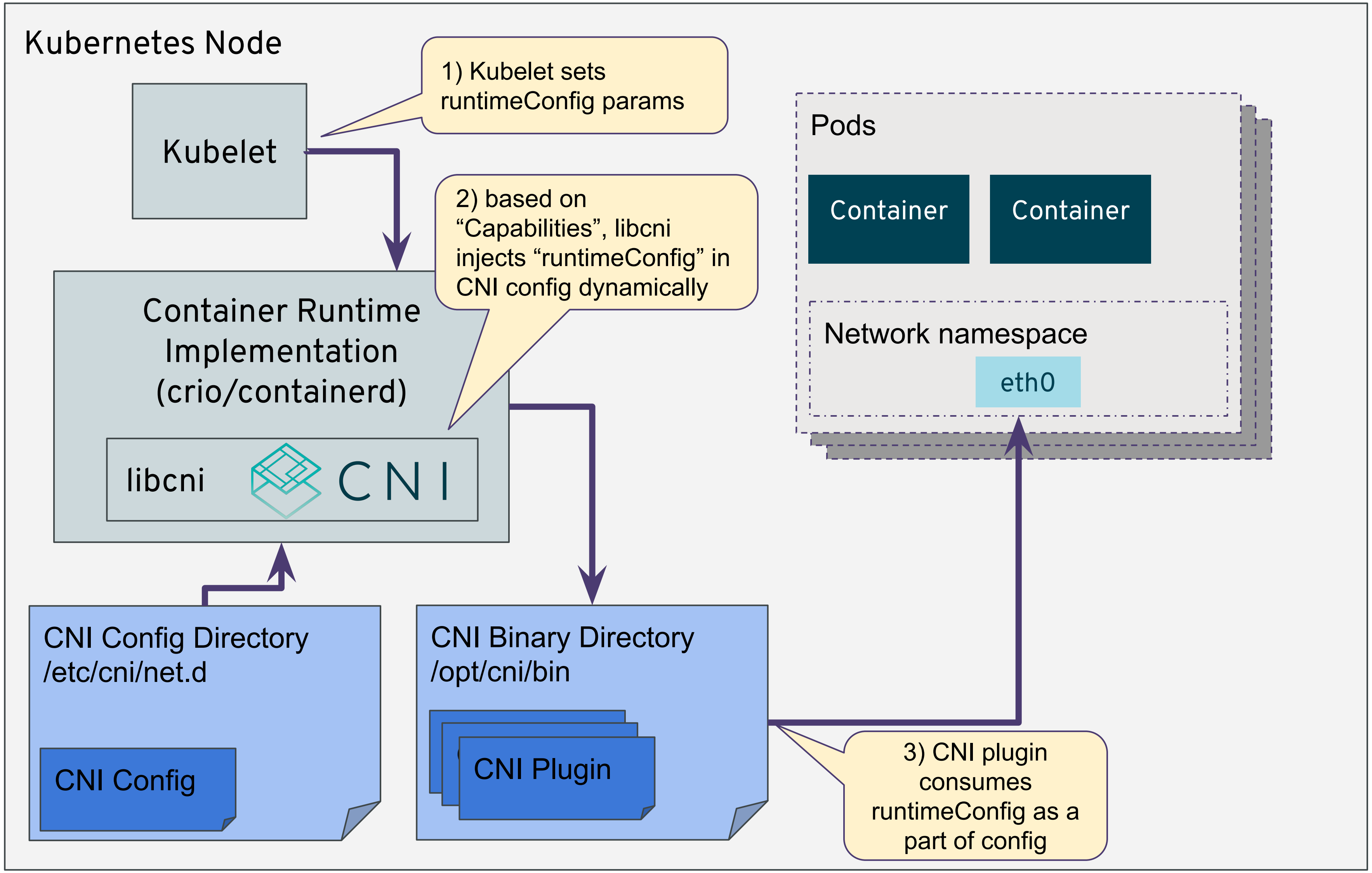
```
{
  "name": "k8s-pod-network",
  "cniVersion": "1.0.0",
  "plugins": [
    {
      "type": "calico",
       (snip)
      }
    },
    {
      "type": "portmap",
      "snat": true,
      "capabilities": {"portMappings": true}
    },
    {
      "type": "bandwidth",
      "capabilities": {"bandwidth": true}
    }
  ]
}
```

Hey, I have such Pod YAML file.
How to configure that?
It must touch with interface, right?



“Capabilities”/”RuntimeConfig” makes it possible!

System flows: “RuntimeConfig”


Kubernetes Node
Kubelet
1) Kubelet sets runtimeConfig params
2) based on “Capabilities”, libcni injects “runtimeConfig” in CNI config dynamically
Container Runtime Implementation (crio/containerd)
libcni
CNI
Pods
Container
Container
Network namespace
eth0
CNI Config Directory /etc/cni/net.d
CNI Config
CNI Binary Directory /opt/cni/bin
CNI Plugin
3) CNI plugin consumes runtimeConfig as a part of config

Capabilities/RuntimeConfig

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
     kubernetes.io/ingress-bandwidth: 1M
     kubernetes.io/egress-bandwidth: 1M
...
```

```
{
 "name": "k8s-pod-network",
 "cniVersion": "1.0.0",
 "plugins": [
  {
   "type": "calico",
    (snip)
   }
  },
  {
   "type": "portmap",
   "snat": true,
   "capabilities": {"portMappings": true}
  },
  {
   "type": "bandwidth",
   "capabilities": {"bandwidth": true}
  }
 ]
}
```

“runtimeConfig” fields are injected by libcni, based from Pod yaml annotation, through kubelet!

```
{
 "capabilities": { "bandwidth": true  },
 "cniVersion": "1.0.0",
 "name": "k8s-pod-network",
 "prevResult": {
  "cniVersion": "1.0.0",
  "dns": {},
  "interfaces": [ { "name": "caliac8fe3b11f2" } ],
  "ips": [ {
    "address": "10.244.135.129/32",
    "version": "4"
   } ]
 },
 "runtimeConfig": {
  "bandwidth": {
   "ingressRate": 1000000,
   "ingressBurst": 34359738359,
   "egressRate": 1000000,
   "egressBurst": 34359738359
  }
 },
 "type": "bandwidth"
}
```

CNI Config: Wrap-up

CNI Plugins

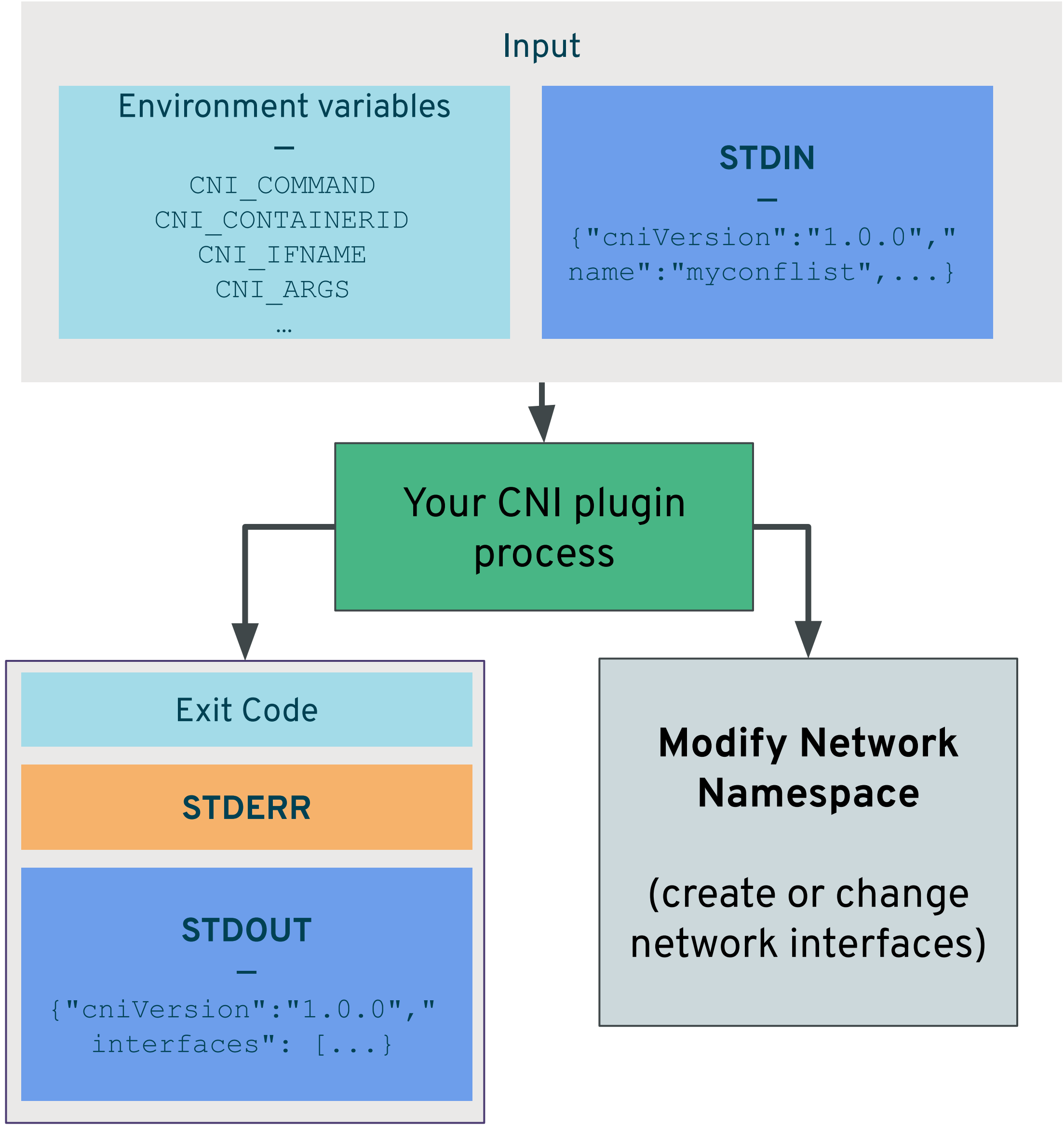
- input
  - STDENV (container specific params, including ‘**CNI command**’)
  - **CNI Configs** (from stdin) with **runtimeConfig** params
- execute
  - to create network interface, or
  - change something (previously created interface in **prevConfig**, as chain)
- and output
  - **CNI Results** (i.e. interface name, IP addresses and so on)
  - Error code
  - Error message (in stderr)

Developing CNI Plugins

CNI Plugins: Input, execution & output

CNI Plugins

- Input
  - STDENV (container specific params, including ‘CNI command’)
  - CNI Configs (from STDIN), with runtimeConfig params
- Execute
  - to create network interface, or
  - change something (previously created interface in prevConfig, as chain)
- Output
  - CNI Results (i.e. interface name, IP addresses and so on)
  - Error code
  - Error message (in stderr)



Input

Environment variables

–

```
CNI_COMMAND
CNI_CONTAINERID
CNI_IFNAME
CNI_ARGS
…
```

STDIN

–

```
{"cniVersion":"1.0.0","
name":"myconflist",...}
```

Your CNI plugin process

Exit Code

STDERR

STDOUT

–

```
{"cniVersion":"1.0.0","
interfaces": [...]}
```

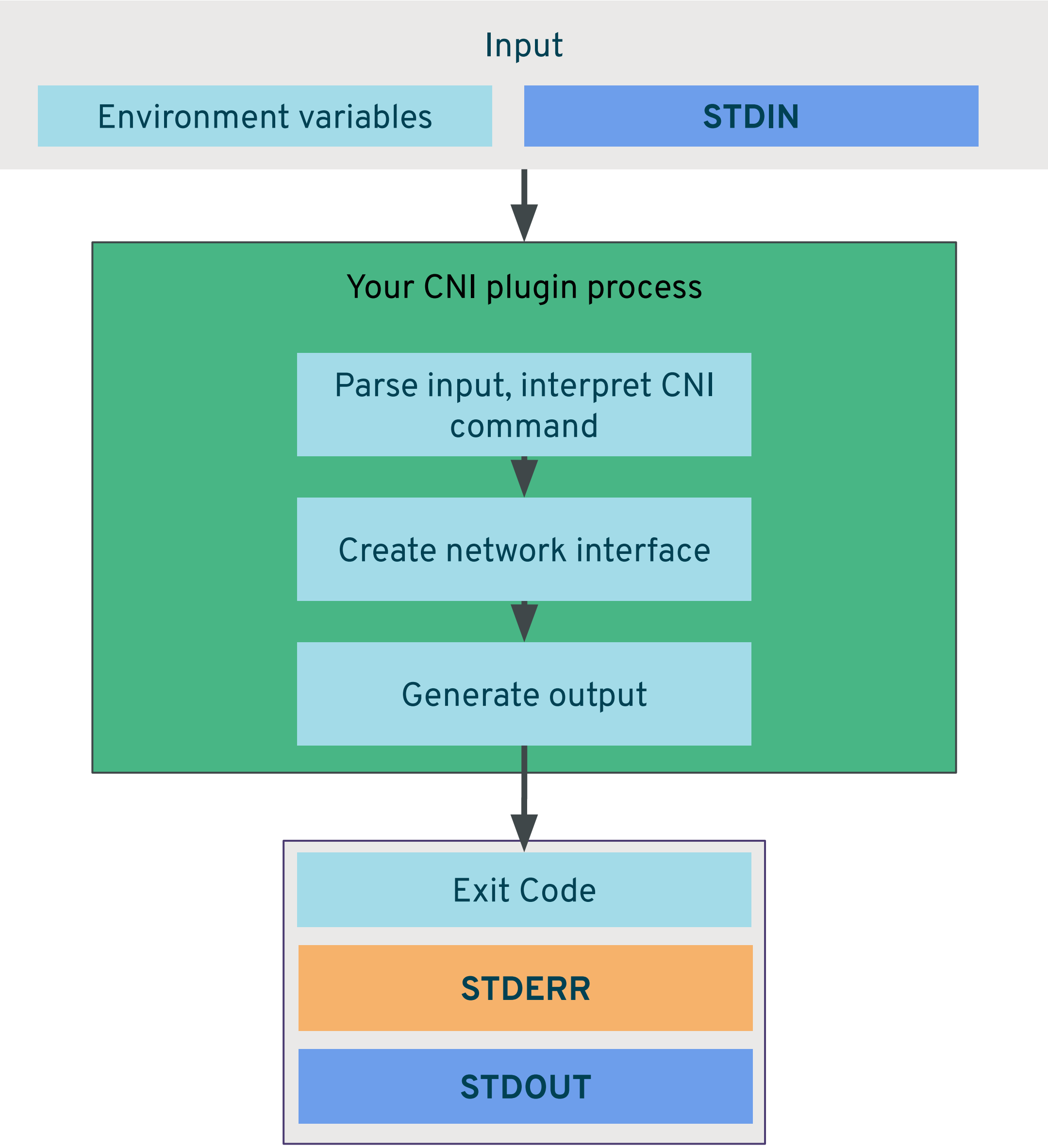
Modify Network Namespace

(create or change network interfaces)

What should your CNI plugin do?

CNI Plugins

- Parse
  - STDENV (container specific params, including 'CNI command')
  - CNI Configs (from STDIN), with runtimeConfig params
- Do your plugin's job
  - to create network interface, or
  - change something (previously created interface in prevConfig, as chain)
- ...And output
  - CNI Results (i.e. interface name, IP addresses and so on)
  - Error code
  - Error message (in stderr)



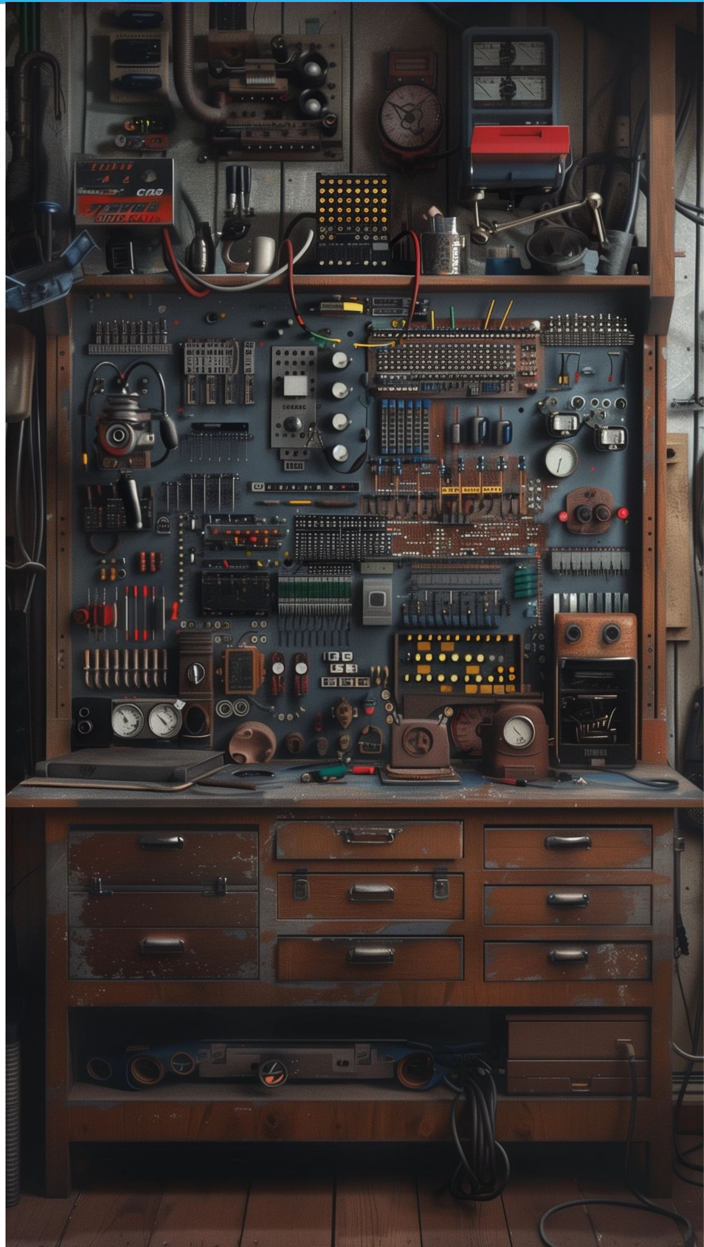
What should the CNI plugin do?

CNI Plugins

- Parse
    - stdenv (container specific params, including ‘CNI command’)
        - CNI_COMMAND: ADD/DEL/CHECK/VERSION
        - CNI_CONTAINER_ID: ‘Container Identifier’, unique ID for container, depends on implementation
            - crio/containerd: “POD ID” of ‘crictl ps’ output (not Pod UID of ‘kubectl get pod’)
        - CNI_NETNS: Linux netns path
        - CNI_IFNAME: interface name (i.e. eth0 in Kubernetes)
        - CNI_ARGS:
            - K8S_POD_NAME
            - K8S_POD_NAMESPACE
            - K8S_POD_INFRA_CONTAINER_ID (as far as I checked, it’s same as CNI_CONTAINER_ID)
            - K8S_POD_UID
        - CNI_PATH: Path for CNI plugins
    - CNI Configs (from stdin) with runtimeConfig params

Gotchas!

- **Kubernetes Integration**
  - Service account (to touch Kubernetes API objects)
- **Don't get tripped up by CNI DEL!**
  - DEL command should *NOT* return an error!
  - Plugin needs to handle multiple calls for CNI DEL!
    - Both of these are mentioned in the CNI spec.
- **Be mindful of CNI version!**
  - Return the CNI result, based on a CNI config's 'cniVersion'!
  - There are helper libraries for this in the CNI repo.



What should the CNI plugin do?

CNI Plugins

- Parse
  - stdenv (container specific params, including ‘CNI command’)
  - CNI Configs (from stdin), including runtimeConfig params
    - Need to care ‘cniVersion’!
      - 0.2.0: no ‘plugins’
      - 0.3.0/0.3.1: ‘plugins’
      - 0.4.0: CHECK command (‘disableCheck’ field is introduced)
      - 1.0.0: only ‘plugins’ is supported (non-list configurations is no longer supported)
    - There are several helper libraries (i.e. CNI project’s library, in ‘cni’ repo)

What should the CNI plugin do?

CNI Plugins

- Output
  - CNI Results (i.e. interface name, IP addresses and so on)
  - Error code
  - Error message (in stderr)

What should the CNI plugin do?

CNI Plugins

- Output
    - CNI Results (i.e. interface name, IP addresses and so on)
        - **Plugin must return appropriate CNI Result format, based on CNI config's 'cniVersion'!**
    - Error code
    - Error message (in stderr)

Be mindful of CNI version!

```
{
    "cniVersion": "0.2.0",
    "ip4": {
        "ip": "10.1.1.1/24"
    },
    "ip6": {
        "ip": "2001:db8::1/64"
    },
    "dns": {}
}
```

```
{
    "cniVersion": "0.4.0",
    "interfaces": [
        {
            "name": "eth0",
            "mac": "7a:32:9e:c4:e9:7e",
            "sandbox": "/var/run/netns/test"
        }
    ],
    "ips": [
        {
            "version": "4",
            "interface": 0,
            "address": "10.1.1.1/24"
        },
        {
            "version": "6",
            "interface": 0,
            "address": "2001:db8::1/64"
        }
    ],
    "dns": {}
}
```

```
{
    "cniVersion": "1.0.0",
    "interfaces": [
        {
            "name": "eth0",
            "mac": "96:9b:fe:87:c0:9b",
            "sandbox": "/var/run/netns/test"
        }
    ],
    "ips": [
        {
            "interface": 0,
            "address": "10.1.1.1/24"
        },
        {
            "interface": 0,
            "address": "2001:db8::1/64"
        }
    ],
    "dns": {}
}
```

…Because you might have different output formats!

```
{
   "cniVersion": "0.2.0",
   "ip4": {
      "ip": "10.1.1.1/24"
   },
   "ip6": {
      "ip": "2001:db8::1/64"
   },
   "dns": {}
}
```

```
{
   "cniVersion": "0.4.0",
   "interfaces": [
      {
         "name": "eth0",
         "mac": "7a:32:9e:c4:e9:7e",
         "sandbox": "/var/run/netns/test"
      }
   ],
   "ips": [
      {
         "version": "4",
         "interface": 0,
         "address": "10.1.1.1/24"
      },
      {
         "version": "6",
         "interface": 0,
         "address": "2001:db8::1/64"
      }
   ],
   "dns": {}
}
```

```
{
   "cniVersion": "1.0.0",
   "interfaces": [
      {
         "name": "eth0",
         "mac": "96:9b:fe:87:c0:9b",
         "sandbox": "/var/run/netns/test"
      }
   ],
   "ips": [
      {
         "interface": 0,
         "address": "10.1.1.1/24"
      },
      {
         "interface": 0,
         "address": "2001:db8::1/64"
      }
   ],
   "dns": {}
}
```

Different versions have
different IP result format!

What should the CNI plugin do? (wrap-up, again!)

CNI Plugins

- Parse
  - STDENV (container specific params, including **‘CNI command’**)
  - **CNI Configs** (from stdin), including **runtimeConfig** params
- Do your plugin job
  - to create network interface, or
  - change something (previously created interface in **prevConfig**, as chain)
- and Output
  - **CNI Results** (i.e. interface name, IP addresses and so on)
  - Error code
  - Error message (in stderr)

Hands on tutorial

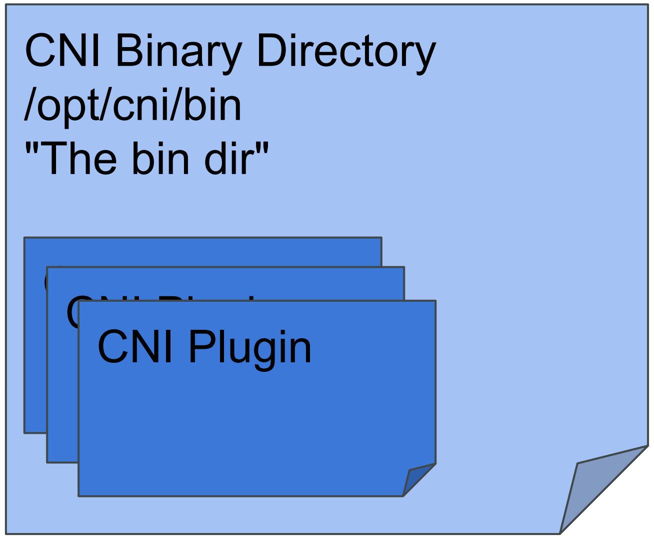
Check out the GitHub Repo!

https://github.com/dougbtv/cni-hero-hands-on

Troubleshooting

Wait, where did I put that? Know your paths

Figure out where your paths are, these are key!

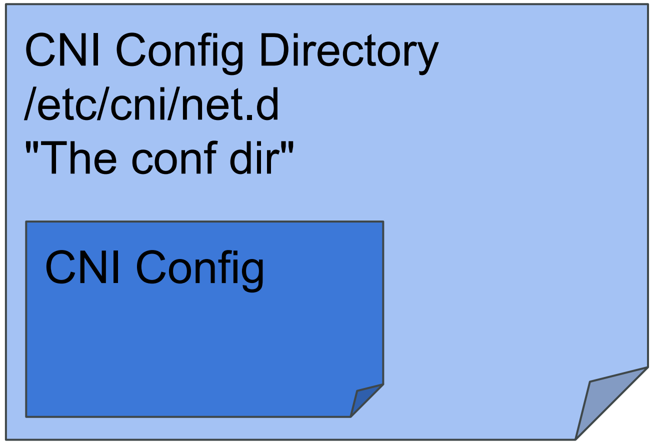


CNI Binary Directory
/opt/cni/bin
"The bin dir"

CNI Plugin

```
(containerd)
$ grep -r bin_dir /etc/containerd

(cri-o)
$ grep -r plugin_dirs /etc/crio
```



CNI Config Directory
/etc/cni/net.d
"The conf dir"

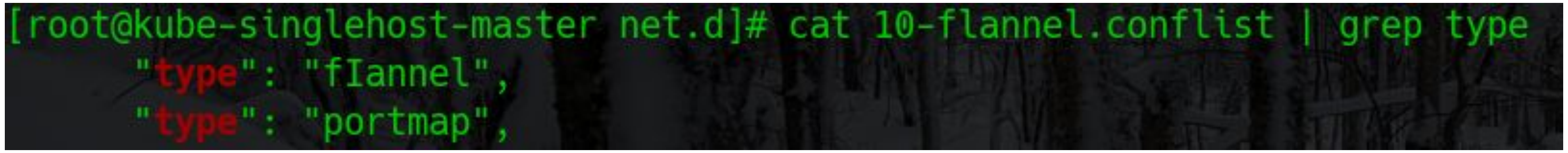
CNI Config

```
(containerd)
$ grep -r conf_dir /etc/containerd

(cri-o)
$ grep -r network_dir /etc/crio
```

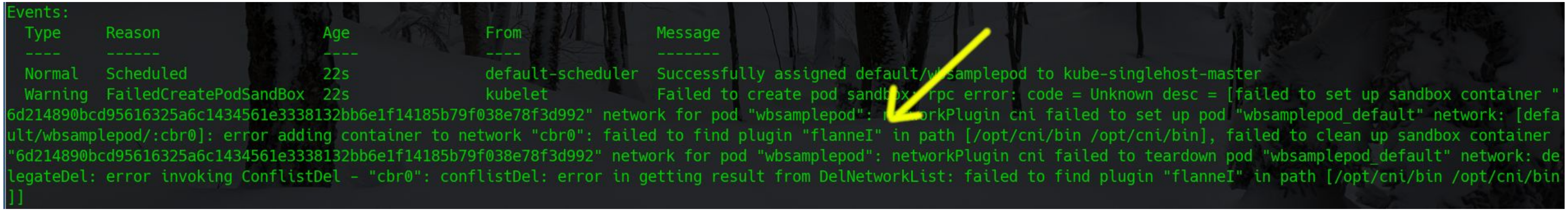
The "type" field is a binary on disk.

So, if it's wrong, and it doesn't match a filename in your CNI bin dir, it'll fail.



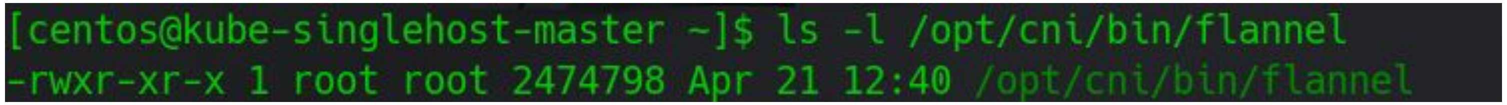
```
[root@kube-singlehost-master net.d]# cat 10-flannel.conflist | grep type
      "type": "fIannel",
      "type": "portmap",
```

When you launch a pod, and it doesn't come up, do a "kubectl describe pod" and you will likely see an event for it.



```
Events:
  Type     Reason                  Age               From               Message
  ----     ------                  ----              ----               -------
  Normal   Scheduled               22s               default-scheduler  Successfully assigned default/wbsamplepod to kube-singlehost-master
  Warning  FailedCreatePodSandBox  22s               kubelet            Failed to create pod sandbox: rpc error: code = Unknown desc = [failed to set up sandbox container "
6d214890bcd95616325a6c1434561e3338132bb6e1f14185b79f038e78f3d992" network for pod "wbsamplepod": networkPlugin cni failed to set up pod "wbsamplepod_default" network: [defa
ult/wbsamplepod/:cbr0]: error adding container to network "cbr0": failed to find plugin "flanneI" in path [/opt/cni/bin /opt/cni/bin], failed to clean up sandbox container
"6d214890bcd95616325a6c1434561e3338132bb6e1f14185b79f038e78f3d992" network for pod "wbsamplepod": networkPlugin cni failed to teardown pod "wbsamplepod_default" network: de
legateDel: error invoking ConflistDel - "cbr0": conflistDel: error in getting result from DelNetworkList: failed to find plugin "flanneI" in path [/opt/cni/bin /opt/cni/bin
]]
```

CNI searches for a file called "flannel" defined by the "type" field, and it looks for that file in your bin dir

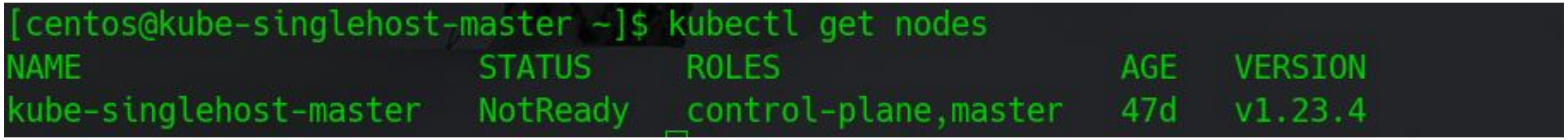


```
[centos@kube-singlehost-master ~]$ ls -l /opt/cni/bin/flannel
-rwxr-xr-x 1 root root 2474798 Apr 21 12:40 /opt/cni/bin/flannel
```

HINT: The most common cause of this is often not having the CNI plugin installed on disk (especially the reference plugins provided by the CNI contributors)

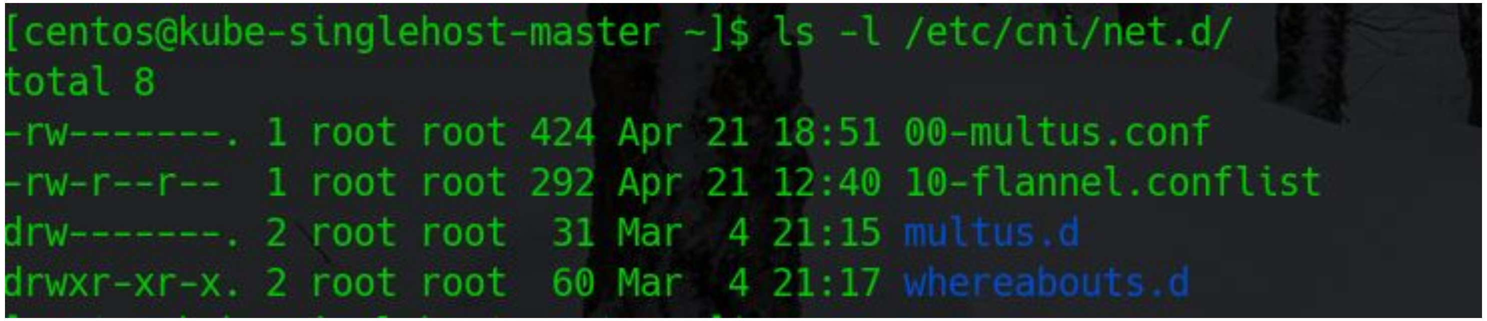
Node Readiness: It's a CNI thing.

If you're seeing a node in NotReady – you should probably check your default CNI network provider.



```
[centos@kube-singlehost-master ~]$ kubectl get nodes
NAME                       STATUS     ROLES                  AGE   VERSION
kube-singlehost-master     NotReady   control-plane,master   47d   v1.23.4
```

The first file ascii-alphabetically is the one that counts. (e.g. “010-foobar.conf” wins “1-foobar.conf”!)



```
[centos@kube-singlehost-master ~]$ ls -l /etc/cni/net.d/
total 8
-rw-------. 1 root root 424 Apr 21 18:51 00-multus.conf
-rw-r--r--  1 root root 292 Apr 21 12:40 10-flannel.conflist
drw-------. 2 root root  31 Mar  4 21:15 multus.d
drwxr-xr-x. 2 root root  60 Mar  4 21:17 whereabouts.d
```

Community

The SPEC, CNI libraries and Plugins

https://www.cni.dev/ Official web page

- ○ Web formatted document (of org repo markdown)
- ○ Reference Plugin (e.g. macvlan/ipvlan) README

https://github.com/containernetworking/ CNI's github org page

- cni/
  - SPEC.md (CONVENTIONS.md): CNI Specifications
    (Note: 'master' is "working document", not official latest!)
  - libcni/ CNI Runtime Implementation in Golang
  - pkg/ CNI Plugin helper libraries in Golang
- plugins/
  - Reference CNI Plugins (macvlan/ipvlan/host-local/dhcp …)

More CNI Update Tomorrow!

Another CNI talk tomorrow afternoon!

Title:	CNI Recap and Update
Date:	**Friday, March 22**
Time:	**14:00 - 14:35**
Place:	**Pavilion 7 | Level 7.3 | N03**

The maintainers of the CNI project will give an update as to what we've been working on, including v1.1 of the protocol. We'll then discuss where CNI may go in the near future.

Thank you!

Any questions?

Title

{ INTENTIONALLY BLANK }

REFERENCE SLIDES FOLLOW