



KubeCon



CloudNativeCon

Europe 2018

Kubernetes and the CNI: Where we are and What's Next

**Casey Callendrello
RedHat / CoreOS**



Who am I?



KubeCon



CloudNativeCon

Europe 2018

Casey Callendrello
Red Hat (née CoreOS)

github.com/squeed
@squeed

Rkt developer
CNI maintainer





0: Prologue.

1: How CNI is typically used.

2: Best practices for operators.

3: Best practices for plugin development.

4: What's next?

Prologue



KubeCon



CloudNativeCon

Europe 2018

- **What is CNI, actually?**
 - **It's OK to be confused... it's an overloaded term**
- **Why is it the way it is?**

What is CNI?



KubeCon



CloudNativeCon

Europe 2018

CNI is at least three different things:

1. A vendor-neutral protocol
 - Used by container runtimes to make request to networking providers
 - Not just for kubernetes!
2. A set of commonly used network plugins maintained by the community
3. A “kubelet network plugin”
 - E.g. “`--network-plugin=cni`” on the command line

How does it work with Kubernetes?



KubeCon



CloudNativeCon

Europe 2018

1. You write a configuration file
2. You put some binaries on disk
3. You create your pod
4. Kubelet executes the binaries
 - And passes the configuration file on stdin
5. Your pod is online!

Why is it designed the way it is?



KubeCon



CloudNativeCon

Europe 2018

or: Why isn't it a gRPC + daemon, like every other Kubernetes plugin?

CNI came out of rkt, a daemonless container runtime.

Rkt focuses on clean integration points.

Aside: network namespaces



KubeCon



CloudNativeCon

Europe 2018

Also, managing network namespaces in long-running Go processes is **unreliable!**

This was only fixed in go v1.10

No choice, you have to use subprocesses.



0: Prologue

1: How CNI is typically used.

2: Best practices for operators.

3: Best practices for plugin development.

4: What's next?

How is CNI used?



KubeCon

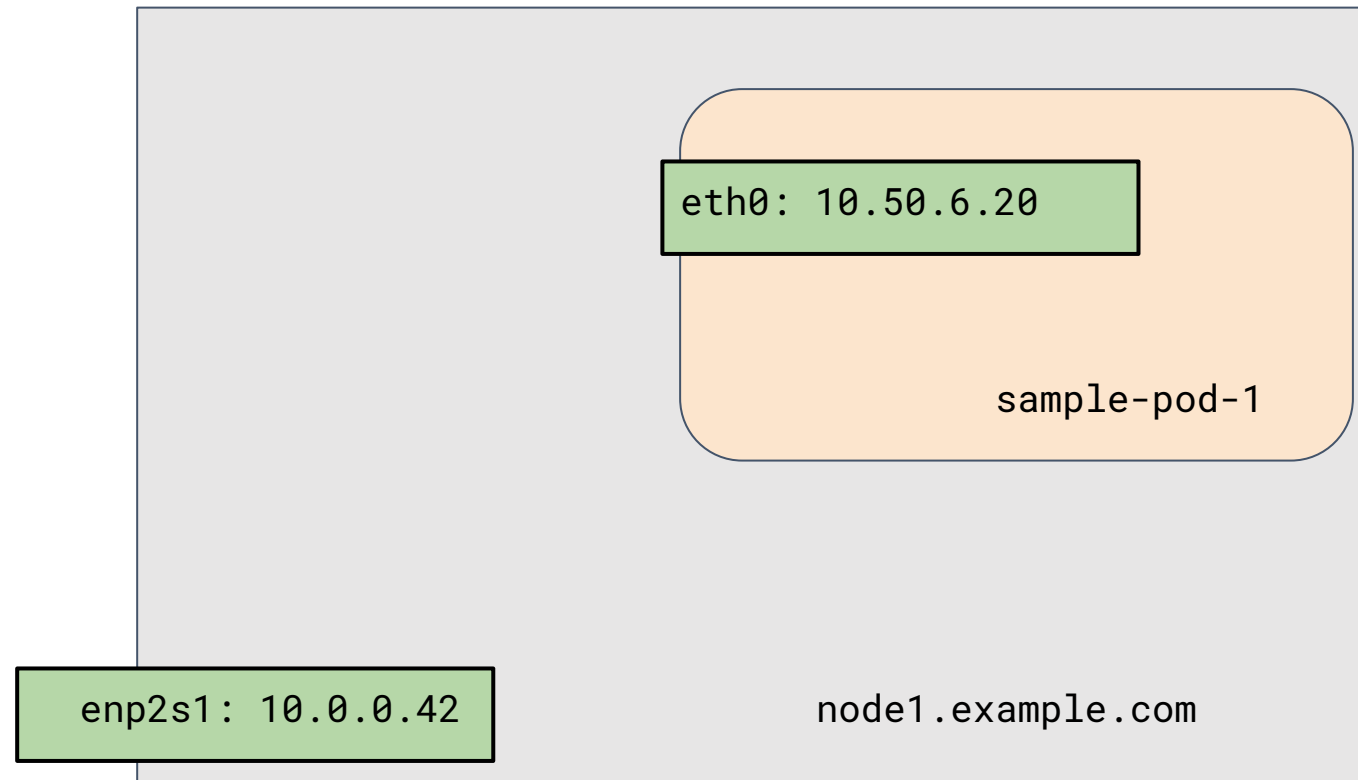


CloudNativeCon

Europe 2018

```
apiVersion: v1
kind: Node
...
status:
  addresses:
    - type: InternalIP
      address: 10.0.0.42
spec:
  podCIDR: 10.50.6.0/24
```

```
apiVersion: v1
kind: Pod
...
status:
  hostIP: 10.0.0.42
  podIP: 10.50.6.20
```



Some terminology



KubeCon



CloudNativeCon

Europe 2018

We have a node

- The node has a single network interface.
- The node has an address pool called the PodCIDR
- There is an API object that reflects this

The node has pods

- Each pod has a network interface always called eth0
- The interface has a single IP. The api object reflects this IP.

What is CNI doing?



KubeCon



CloudNativeCon

Europe 2018

When a pod is added to a node, a CNI plugin is called on to do 3 basic things:

1. Create `eth0` inside the pod's network namespace
 - In other words, connect the pod - somehow - to the network
2. Allocate the PodIP
 - Usually from the PodCIDR
3. Make this PodIP reachable by the whole cluster

What is CNI doing?



KubeCon



CloudNativeCon

Europe 2018

The CNI plugins must provide:

- 1. Connectivity**
- 2. Reachability**

Why plugins?



KubeCon



CloudNativeCon

Europe 2018

Connectivity might depend on

- Network interface hardware
- Isolation technology
- Performance requirements

Reachability depends on

- Networking hardware
 - Can you touch them?
 - Are they in “the cloud?”
- Router “interfaces”
 - BGP?
 - OSPF?
 - REST?
- Performance requirements

Reachability



KubeCon



CloudNativeCon

Europe 2018

Kubernetes doesn't match “traditional” networking environments.

Traditional environments:

- Nodes have long, predictable lifetimes
- Addressing is manual and topology-aware
- One interface = one IP

Kubernetes:

- Topology unaware
- Allocates 100+ IPs per node
- Lifetimes are short; churn is high

How is CNI used?



KubeCon



CloudNativeCon

Europe 2018

How do CNI plugins work?

(with Kubernetes)

How do CNI plugins work



KubeCon



CloudNativeCon

Europe 2018

CNI plugins for kubernetes typically have two components:

1. A CNI binary that configures the pod's interface. (Connectivity)
2. A daemon that manages routing. (Reachability)



Most CNI plugins provide connectivity in the same way:

1. Create a “veth” pair
 - Veth is a point-to-point virtual tunnel
2. Move one end of the pair in to the container’s namespace
3. Configure an ip and route in the container’s namespace

Packets leaving the container are simply routed through the host’s IP stack. They are usually masqueraded.

Connectivity



KubeCon



CloudNativeCon

Europe 2018

Play along at home: <https://lwn.net/Articles/580893/>

Namespaces in operation, by Jake Edge

(`ip netns`) is a container runtime...

Reachability



KubeCon



CloudNativeCon

Europe 2018

Goal: make every PodCIDR reachable from every Node

Reframe: Announce dynamic routes to some peers.

Sound familiar?

Essentially all CNI plugins for Kubernetes include a daemon that:

- Runs on every host
- Programs the network with learned routes

Reachability



KubeCon



CloudNativeCon

Europe 2018

Programming the network takes many forms:

- Overlay networks (e.g. Flannel, Weave, Calico)
 - Don't program your routers, program every node's route table
- Cloud provider APIs (cp-azure, Flannel)
 - Watch your resource limits!
- Routing protocols (Calico, Romana)
 - PodCIDRs are now fully reachable!
 - Watch your FIB limits



Why two components? Why can't we use the binary?

CNI only has two methods: Add Container and Delete Container

- Route lifecycle is per-node

Most networks require a long-running process:

- Overlay networks need to watch **all nodes**
- Routing protocols require persistent sessions
- Deletion is always hard



0: Prologue

1: How CNI is typically used.

2: Best practices for operators

3: Best practices for plugin development.

4: What's next?

Best Practices: Configuration



KubeCon



CloudNativeCon

Europe 2018

Kubelet learns which CNI plugin to use from a CNI configuration file.

How do you tell it where this configuration file is?

`--cni-network? --cni-configuration?`

Best Practices: Configuration



KubeCon



CloudNativeCon

Europe 2018

Answer: You don't!

The kubelet scans `--cni-conf-dir` every 5 seconds and uses whichever one has the lowest-ordered filename.

It keeps scanning and parsing forever.

Best practice: Only ever write one CNI configuration file to disk.

Best Practices: Configuration

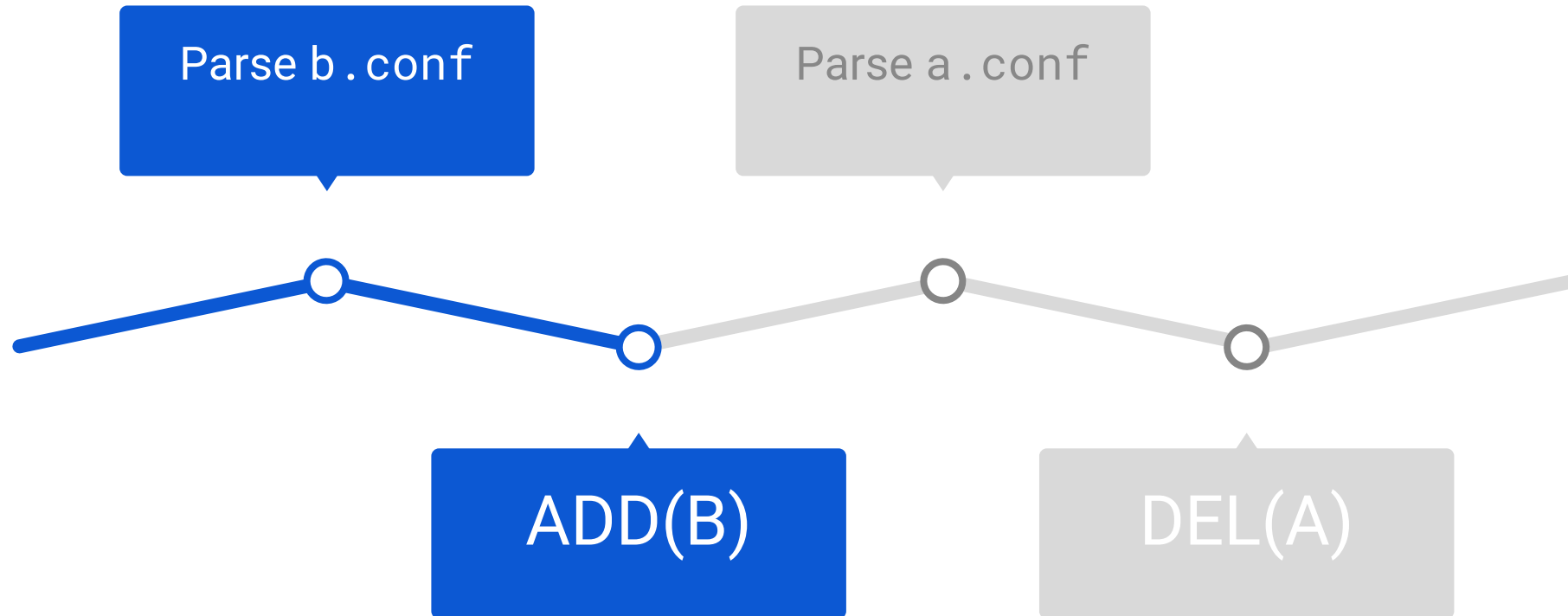


KubeCon



CloudNativeCon

Europe 2018



Best Practices: Configuration



KubeCon



CloudNativeCon

Europe 2018

CNI now supports plugin chaining.

This is a powerful tool for the operator to adjust their network.

Current chained plugins:

- Set up host-port forwarding
- Tweak sysctl parameters
- Bandwidth limits
- Write your own!

```
{ "cniVersion:" "0.3.1",  
  "name": "myNetwork",  
  "plugins": [  
    { "type": "ptp" },  
    { "type": "iptables-allow"},  
    { "type": "tuning",  
      "sysctl": {  
        "net.core.somaxconn": "9001"  
      }  
    }  
  ]  
}
```

Best Practices: Self-hosting



KubeCon



CloudNativeCon

Europe 2018

Deploy your plugins using kubernetes!

A manifest for a CNI plugin should:

1. Set up the routing daemon
2. Copy the plugin binaries to the host
3. Install the configuration file

Installing the configuration file tells the host that the network is up.
Do it last.

Best Practices: Self-hosting



KubeCon



CloudNativeCon

Europe 2018

Some important things to get right:

- Run in host network
- Ignore all taints
 - So we can run on master!
- Use magic grace period
 - Narrows the window between running daemons

```
spec :
```

```
  hostNetwork: true
```

```
  tolerations:
```

```
    - effect: NoSchedule
```

```
      operator: Exists
```

```
    - effect: NoExecute:
```

```
      operator: Exists
```

```
  terminationGracePeriodSeconds: 0
```

Best Practices: Disaster Recovery



KubeCon



CloudNativeCon

Europe 2018

Self-hosted control-planes need to be careful here.

Understand your dependencies:

- Always run your APIServer in host networking
- Make sure your daemonset will start up without any other running services



0: Prologue

1: How CNI is typically used.

2: Best practices for operators

3: Best practices for plugin development.

4: What's next?

1. Understand the kubelet lifecycle



KubeCon



CloudNativeCon

Europe 2018

The kubelet **does** guarantee that:

You will see exactly one ADD request for each CNI_CONTAINERID.

1. Understand the kubelet lifecycle



KubeCon



CloudNativeCon

Europe 2018

You **can't assume** that you will see a DELETE...

- A comfortable amount of time after the sandbox is deleted
- Before the node is rebooted
- Before 1000 crashlooping sandboxes are created and immediately stopped and you run out of IPs
- Before the node ~~catches on fire~~ is ChaosMonkey'd

Be prepared to GC resources externally where necessary.

Use /run to store state if it makes sense.

1. Understand the kubelet lifecycle



KubeCon



CloudNativeCon

Europe 2018

Any other ID (e.g. PodName) could be reused.
This has caused production bugs.

ADD A1

A1 Crash!

DELETE A1

ADD A2

(time passes)

A2 Offline!

2. Understand the Host Network



KubeCon



CloudNativeCon

Europe 2018

Kubernetes generally expects that all packets traverse the host.

Let's look at some of the implications of this.

2. Understand the Host Network



KubeCon



CloudNativeCon

Europe 2018

Most plugins will do most of their work in the host's namespace

1. Duplicating functionality in every pod is a waste
 - Especially of iptables rules and conntrack entries
2. HostNetwork pods should be (mostly) equivalent
3. Programming the pod's namespace could be a leak or security issue

2. Understand the Host Network



KubeCon



CloudNativeCon

Europe 2018

A lot of pod-facing functionality is usually implemented on the host-side:

1. Kube-proxy (ServiceIP, NodePort)
 - Uses iptables or ipvs to do DNAT
2. Most NetworkPolicy providers
3. Traffic shaping
4. Overlay networks

2. Understand the Host Network



KubeCon



CloudNativeCon

Europe 2018

If you're willing to re-implement this, or give some of it up, you can choose to have packets go directly on the wire.

E.g. macvlan, SR-IOV, etc.

As always, you can trade clean abstractions for speed.



KubeCon



CloudNativeCon

Europe 2018

3. Be wary of hairpin

Hairpin is any time a packet goes back to its destination.

CNI plugins must handle two cases.

1. Pod A -> Service -> Pod B on same host
2. Pod A -> Service -> Pod A

3. Be wary of hairpin



KubeCon



CloudNativeCon

Europe 2018

A lot of production plugins didn't handle case 2... mine included.

In general, the solution is to

1. Masquerade the source address (to the host's) (sigh)
2. Configure your bridge to allow hairpin.
3. Be sure you have a test case for this!



KubeCon



CloudNativeCon

Europe 2018

4. Use Capability args

The kubelet will fill in some dynamic configuration. This means you can reduce your dependency on the apiserver.

Current capabilities:

- Host port mappings

Future capabilities:

- PodCIDR
- Bandwidth limits

4. Use Capability args



KubeCon



CloudNativeCon

Europe 2018

```
{ "name": "myinet",  
  "type": "example",  
  "capabilities": {  
    "portMappings":  
true,  
  }  
}
```

```
{ "name": "myinet",  
  "type": "example",  
  "runtimeConfig": {  
    "portMappings": [  
      { "hostPort": ... }  
    ]  
  }  
}
```

5. Use Plugin Chaining



KubeCon



CloudNativeCon

Europe 2018

CNI configurations can include multiple plugins in a row.

Useful for common post-set-up tweaking.

(NOT multiple interfaces)

- Port-forwarding
- Iptables-allow
- Firewalld-allow
- Sysctl tweaking

5. Plugin Chaining



KubeCon



CloudNativeCon

Europe 2018

```
{ "cniVersion:" "0.3.1",  
  "name": "myNetwork",  
  "plugins": [  
    { "type": "ptp" },  
    { "type": "iptables-allow"},  
    { "type": "tuning",  
      "sysctl": {  
        "net.core.somaxconn": "9001"  
      }  
    }  
  ]  
}
```

```
{ "cniVersion:" "0.3.1",  
  "name": "myNetwork",  
  "type": "iptables-allow",  
  "prevResult": {  
    "interfaces": [  
      {"name": "veth1023"},  
      {"name": "eth0"}  
    ]  
    "ips": [  
      { "address": "10.0.56.1",  
        "interface": 0  
      }  
      { "address": "10.0.56.42",  
        "interface:" 1}  
    ]  
  }  
}
```

5. Use Plugin Chaining



KubeCon



CloudNativeCon

Europe 2018

For ADD:

- Plugins are executed first to last.
- The output of the previous plugin is passed to the next
- The last result is passed back to the kubelet.

For DELETE:

- Plugins are run in reverse order
- The “last” plugins don’t know the IP addresses
 - Can be difficult to know what to tear down
- Being fixed in spec 0.4.0

5. Use Plugin Chaining



KubeCon



CloudNativeCon

Europe 2018

If your plugin **only** touches the host-side, it could easily be a chained plugin.

Using plugin chaining allows administrators to swap out “connectivity” plugins.

Enables diverse workloads, e.g. VMs.



- 1: How CNI is typically used.**
- 2: Best practices for operators**
- 3: Best practices for plugin development.**
- 4: What's next?**

What's Next? CNI v0.4.0



KubeCon



CloudNativeCon

Europe 2018

CNI Spec 0.4.0 (2018 Q2) will include:

- A new GET verb
 - Lots of corner cases, come see me after class.
- Result caching
 - To pass in to GET and DELETE
- More capability args

What's Next? CNI v1.0



KubeCon



CloudNativeCon

Europe 2018

CNI Spec 1.0 will include:

- RFC-style spec
- Acceptance tests

What's Next? IPv6



KubeCon



CloudNativeCon

Europe 2018

Kubernetes 1.9 includes alpha IPv6 support.

CNI has supported IPv6 for years.

Dual-stack support is desired but still has some unsolved design issues:

- The Pod Object has one IP
- Services expect 1:1 IP mapping

What's Next? Multi-network



KubeCon



CloudNativeCon

Europe 2018

Attaching multiple network interfaces to a pod.

Use cases:

- Direct-on-wire interfaces
- VRF / non-uniform routing
- Special interfaces, e.g. infiniband

No CNI changes - 1 configuration = 1 network

What's Next? Multi-network



KubeCon



CloudNativeCon

Europe 2018

Lots to figure out:

- Will pods always have the “default” network?
 - Yes
- How are additional interfaces exposed? Services?
 - No, need to use the APIServer
 - This might change
- What about security?
 - Not on the plan for now
- What about hairpin?
 - Good question!

What's Next? Device Plugins



KubeCon



CloudNativeCon

Europe 2018

There is a desire to combine device plugins and CNI / multi-network.

- Device plugin allocates a device
- Device plugin manages schedulable resources
- CNI plugin initializes the network and hands it to the pod

Currently in design.

What's Next?



KubeCon



CloudNativeCon

Europe 2018

That's up to you!

CNI is an active specification, and welcomes community involvement.
Kubernetes' use of the CNI has lots of room to grow.

Thanks!



KubeCon



CloudNativeCon

Europe 2018

cdc@redhat.com

@squeed

Freenode: #containernetworking

Containernetworking.slack.com

Google Groups: cni-dev