

# Evolution of the Container Network Interface (CNI)

**Casey Callendrello**

CoreOS



**CoreOS**

# Who am I?

[github.com/squeed](https://github.com/squeed)  
[@squeed](https://twitter.com/squeed)

CNI maintainer  
Rkt developer



# The CNI

How it started

How it's grown

New developments

What's next

# What is CNI?

A standard way to connect your containers to your network.

# “Your network”

AWS? GCE? Bare Metal?

BGP? OSPF? OpenFlow? “API”?

Overlay networks?

Flat layer 2 space?

Specialized hardware?

# What is CNI?

An API for networking vendors to implement.

A small library for runtimes to consume.

Contain the complexity

# What is CNI?

A living, active open-source project with:

- 5 maintainers
- 60 contributors at 10+ companies
- 10+ plugin vendors
- 6+ runtimes

# How did it start?





# How did it start?

- Rkt is a daemonless container runtime
- No complicated data model or online DB
- Principle throughout rkt is “clean integration points”

# How did it start?

Need for a simple way to “bring this namespace online”

- Handling namespaces in golang is painful
- Let's initialize the network in a separate process
- And define an interface

**rkt**

---

**Container Networking Interface (CNI)**

---

**veth**

**macvlan**

**ipvlan**

**bridge**

# Adoption & Growth

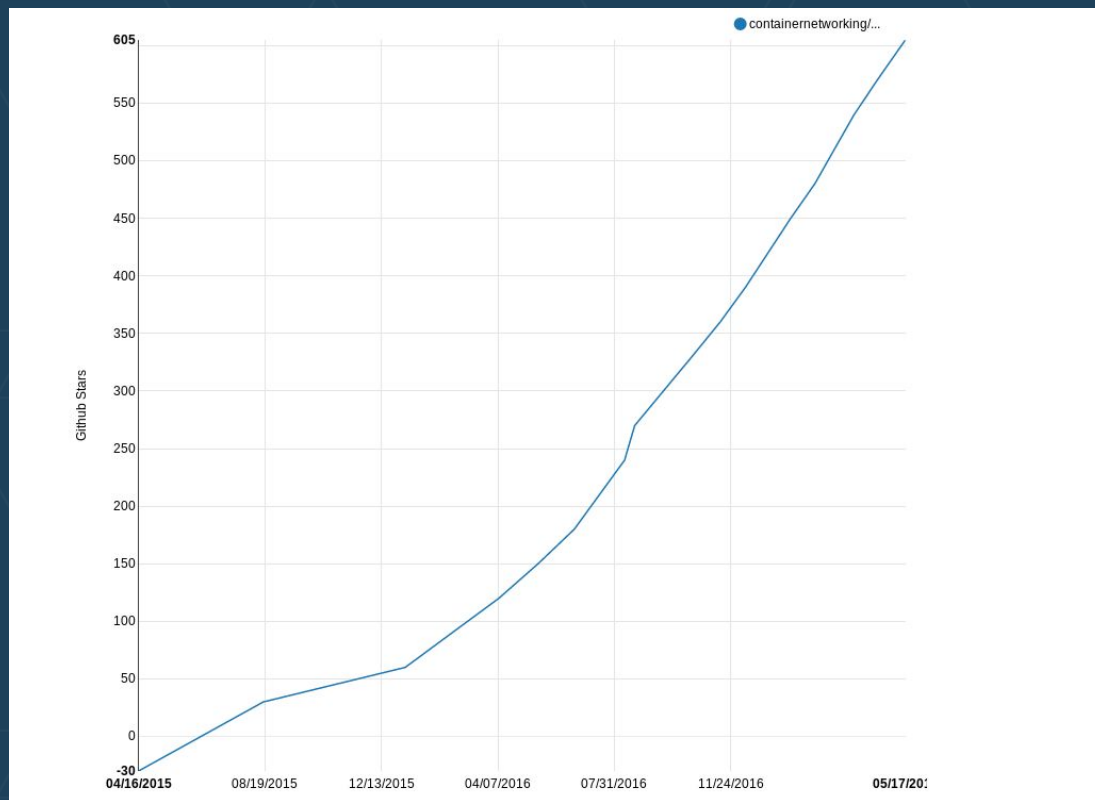
- Spec separated from rkt April 2015
- Replaced Kubernetes network plugin API ~Oct 2015
- Mesos supports CNI from May 2016



# Who uses it? - Vendors

- Canonical Kubernetes
- Tectonic by CoreOS
- Redhat Openshift
- “Kubenet” plugin

# Science!



# Technical introduction

# Technical Overview

## Simple model:

- Runtime Config
- Network Config
- Plugin
- Result

## Result:

- Interfaces
- Addresses
- Routes
- DNS



**Runtime**

**netns**

**Config file**

---

**Libcni (if you like)**

---

**plugin**

**plugin**

**plugin**

**plugin**

# Example invocation

```
$ cat /etc/cni/net.d/10-net.conf
{ "name": "mynet",
  "type": "bridge",
  "ipam": {
    "type": "host-local",
    "subnet": "10.42.0.0/24" }
}
```

# Example invocation

```
$ rkt run --net=mynet docker://redis  
- Or -  
$ kubelet ... --network-plugin=cni ...
```

# Example invocation

```
$ rkt run --net=mynet ...
```

# Example invocation

```
# ip netns new test
# ip netns exec test /bin/bash
# ip link
1: lo: <LOOPBACK> ...
#
```

# Example invocation

```
CNI_COMMAND=ADD \  
CNI_CONTAINERID=example \  
CNI_NETNS=/var/run/netns/test \  
CNI_IFNAME=eth0 \  
CNI_PATH=/opt/cni/bin:. \  
./bridge < 10-net.json
```

# Example invocation

```
# ip link
1: lo: ...
5: eth0@if9: ...
# ip -4 addr show dev eth0
5: eth0@if9: ...
   inet 10.42.0.107/24
```

# CNI spec

- 3 commands: ADD, DELETE, and VERSION
- Configuration on stdin, results on stdout
- Runtime parameters via env
  - CNI\_ARGS for arbitrary runtime params



# CNI spec

- For golang runtimes, libcni implements most boilerplate.
- Response types are versioned
  - Lossy conversion between versions

# CNI spec - IPAM

IPAM is an interesting case

- Needs to be separate plugin
- Solution: CNI!
  - Plugin uses CNI API to discover & execute IPAM plugin
- Ordering is subtle
  - DHCP: IF needs to exist to make request

# Use cases & Plugins

# What plugins exist?

Basic:

Bridge, veth

Muxing:

Macvlan, ipvlan

Overlay networks:

Flannel, Calico, Weave

IPAM:

Host-local, dhcp,  
Infoblox

# What plugins exist?

Cilium: BPF-based networking

- No more iptables & conntrack!
- I want to give Mellanox all my money

SR-IOV: PCI sub-devices

- I am doing HFT and want to give Intel even more of my money

# Common CNI setups

## Bridge + port forwarding

- “I’ll use the host IP for everything”
- Pods not otherwise reachable

# Common CNI setups

## Macvlan + DHCP

- “I already have an expensive network”
- “And an expensive network admin”
- Watch those CAM tables!

Each pod directly connected to network

# Common CNI setups

## Flannel, Weave, or Calico

- “I have a network I can’t change”
  - E.g. AWS VPC
- Routes are distributed using some other database
  - Flannel: etcd
  - Calico: bgpd



# What's new with CNI?

Spec version v0.3

# Awesome logo (Thanks, LF!)



# CNI

# What's new in v0.3

- Type enhancements
- Plugin chaining
- Capability arguments

# Type enhancements in v0.3

New result type adds:

- Multiple IPs
- Interfaces
  - Including host-side interface

# ConfigLists / Plugin Chaining

IPAM separated some common behavior.

Add post-creation plugins too!

# Plugin chaining

Motivating examples: anything that tweaks an existing interface or IP

- Port forwarding
- Sysctl tuning
- Traffic shaping
- SLAAC?

# Plugin chaining

## New configuration format

- Just a list of configuration blocks
- Runtime chains previous result

```
{ "name": "dbnet",  
  "plugins": [  
    { "type": "bridge",  
      "ipam": {  
        "type": "host-local",  
        "subnet": "10.1.0.0/16" }  
    },  
    { "type": "tuning",  
      "sysctl": {  
        "net.core.somaxconn":  
          "500" }  
      }  
    ]  
}
```

# Chained plugins

Existing:

- Tuning
- Iptables-based port forwarding

Future:

- Bridge (for virtual machines, etc.)



# Capability arguments

The problem:

I want to forward ports dynamically.

```
( curl http://127.0.0.1:8080 )
```

# Capability arguments

## The problem

- CNI\_ARGS too freeform
- Don't want to put rewrite logic in the runtime
- Don't know *which* plugin does the forwarding
  - Remember Cilium?

# Capability arguments

Need a way to pass in

- Dynamic (run-time)
- Structured
- Plugin-agnostic (sort of) configuration

To one or more plugins in a chain.

# Capability Arguments

Don't want

- Templating language
- Complicated runtime
- JSON in an environment variable

# Capability arguments

## Solution:

- Spec defines “capabilities”
- Plugins declare support
- Libcni patches arguments into config
  - Yes, it’s JSON manipulation, but it’s standardized

# Capability arguments

```
{  
  "type": "portmap-iptables",  
  "capabilities": {"portMappings": true},  
}
```

Becomes:

```
{ "type": "portmap-iptables",  
  "runtimeConfig": {  
    "portMappings": [  
      {"hostPort": 80, "containerPort": 8080}  
    ]},  
}
```

# Capability arguments

- Writing specs requires careful thought
- We all need to work together
- Many people contributed to the v0.3 upgrade
  - Esp. Kubernetes and Mesos devs

# Putting it all together

```
LoadConfList( dir, name string)  
(*NetworkConfigList, error)
```

```
AddNetworkList(  
list *NetworkConfigList,  
rt *RuntimeConf) ->(Result, error)
```



# What's Next?

# New plugins!

I want YOU to write new plugins

- FirewallD integration
- Traffic shaping
  - Oh, and this is a new capability
- Better tuning
  - Some of you have some hard-learned TCP hacks

# IPv6

Spec as written now makes v6 easy.

CNI-maintained plugins need some small tweaks.

A philosophical question: Is SLAAC an IPAM plugin?

# Virtual machines

Spec does not actually require namespaces

Hyper and Intel working on this

Plugin chaining makes this a lot easier

# CNCF

CNI (probably) going under the CNCF  
[Voting](#) is underway now.

No real change, just formal recognition.

# Splitting spec and code

Right now, all code under a single repository.

That was a mistake.

CNI-maintained plugins

# In summary

# In summary

CNI is a living, breathing project.

We're entirely dependent on the community.

We want you to participate.



# Participation

Write new plugins! It's easy!

Document your use cases

File bugs, fix bugs

Join us in IRC (or slack...)

Cni-dev mailing list.

Monthly community sync.

# CoreOS is running the world's containers

We're hiring (in Berlin!): [careers@coreos.com](mailto:careers@coreos.com)

## OPEN SOURCE

90+ Projects on GitHub, 1,000+ Contributors

 container linux

 rkt  etcd

[coreos.com](http://coreos.com)

## ENTERPRISE

Support plans, training and more

 **TECTONIC**

 **QUAY**

[sales@coreos.com](mailto:sales@coreos.com)

# Thanks!

## QUESTIONS?

[casey.callendrello@coreos.com](mailto:casey.callendrello@coreos.com)

[@squeed](#)

[github.com/squeed](https://github.com/squeed)

## LONGER CHAT?

Slack & IRC

More events: [coreos.com/community](https://coreos.com/community)

# We're hiring: [coreos.com/careers](https://coreos.com/careers)

# Bonus slides

# Example invocation

1. Rkt creates a network namespace
2. Rkt asks libcni to configure “mynet”
3. Libcni scans for file with “name: mynet”
4. Libcni parses out “type”; looks for binary
5. Libcni exec’s binary according to CNI spec
6. Binary creates network, returns result.