KubeCon | CloudNativeCon

Europe 2022

WELCOME TO VALENCIA

KubeCon CloudNativeCon Europe 2022

My CNI Plugin did… what!?: *Debugging CNI with style and aplomb*

Daniel Mellado, Red Hat, Inc.
Doug Smith, Red Hat, Inc.

@danielmellado

Daniel Mellado

- Principal Software Engineer@Red Hat
- Egress Router CNI maintainer
- SIG-OpenStack member
- Former Kuryr-Kubernetes PTL


@dougbtv

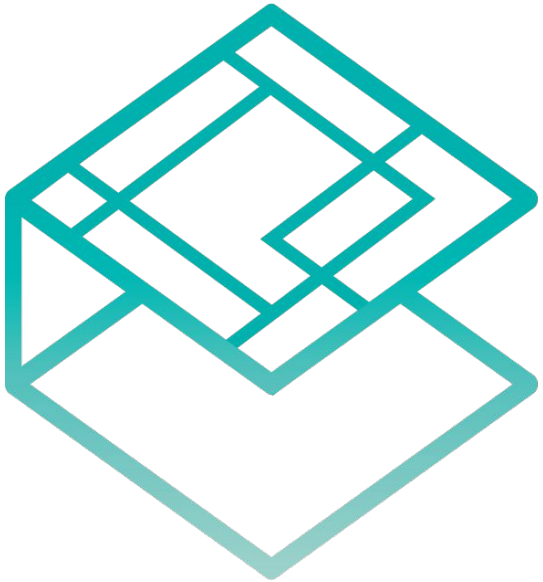
Doug Smith

- Technical lead for OpenShift Network Plumbing Team in OpenShift Engineering
- Multus CNI maintainer
- Network Plumbing Working Group member
- Blog: https://dougbtv.com

What's on the agenda?

- Intro to CNI
- Debugging CNI: Back to Basics
- CNI Tool
- Danger Demo!
- Thick vs. Thin Plugin architectures
- CNI 2.0: What the future holds.

Intro to CNI



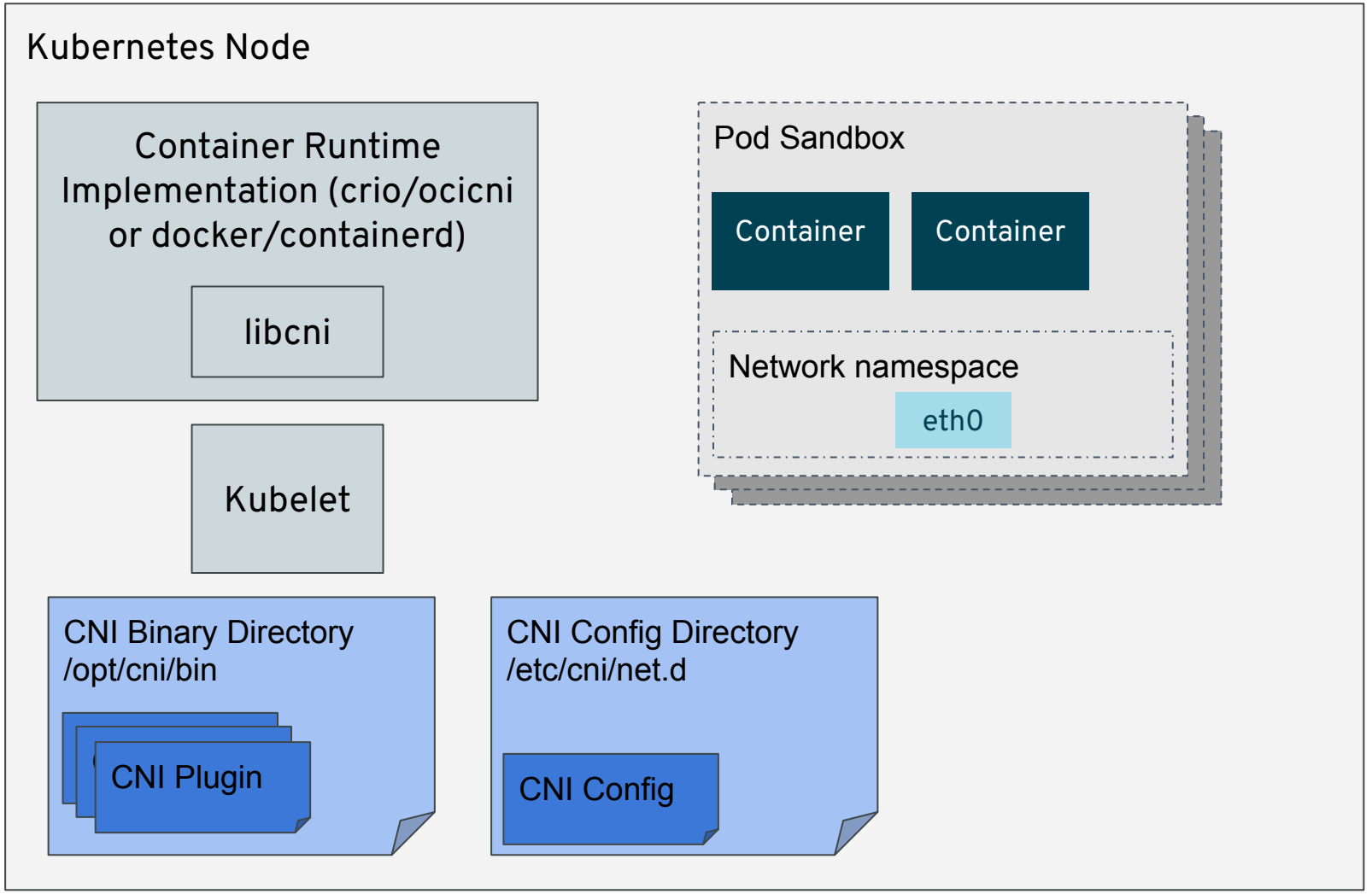
CNI

CNI (Container Network Interface), a Cloud Native Computing Foundation project, consists of a specification and libraries for writing plugins to configure network interfaces in Linux containers, along with a number of supported plugins.

GitHub Repo: https://github.com/containernetworking/cni
CNI Specification: https://github.com/containernetworking/cni/blob/main/SPEC.md

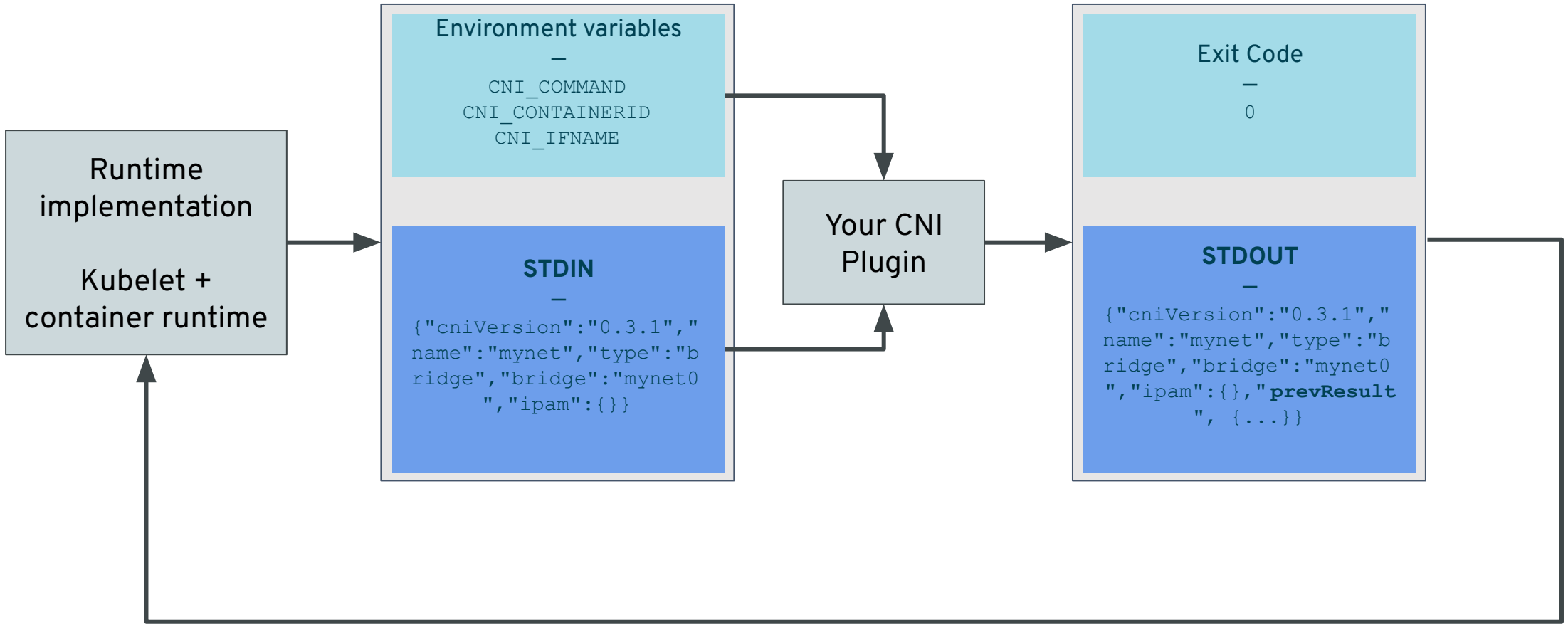
Remember: CNI isn't Kubernetes specific!

CNI Anatomy: An overview



What goes into a CNI plugin call?

CNI plugins are executed on the host, and get input from STDIN, as well as environment variables, and they give output as STDOUT (and an exit code)



CNI Operations, "Commands"

ADD — *Add container to network, or apply modifications*

DEL — *Remove container from network, or un-apply modifications*
Do garbage collection!

CHECK — *Check container's networking is as expected*
Generally called right after pod creation succeeds. Exit non-zero if check doesn't succeed.

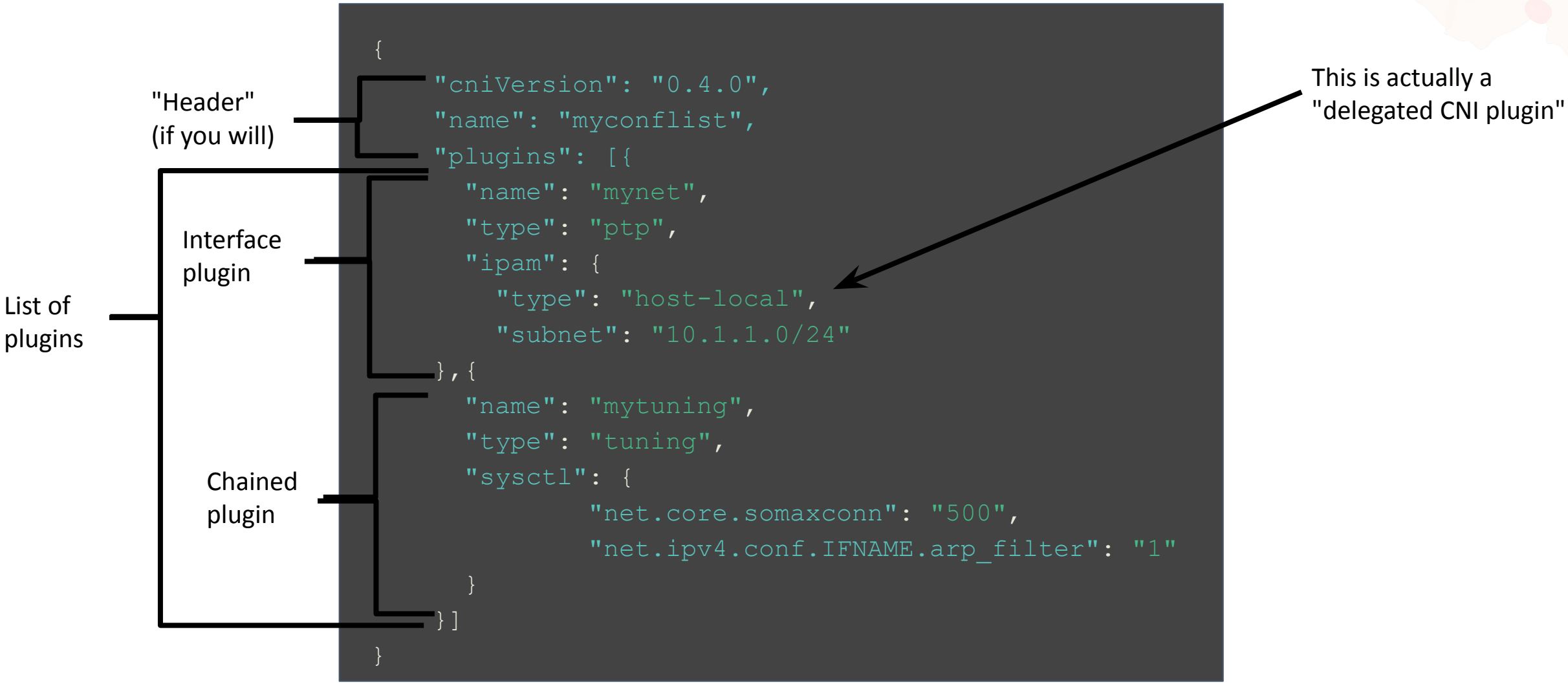
VERSION — *probe plugin version support*
Check the spec for the exact format.

CNI DEL: Gotchyas!

- This won't happen if a node is powered off! (Or possibly if you delete a pod without a grace period)
- Your plugin MUST exit 0, if you exit non-zero, CNI DEL will happen over and over again, in a crashloop where a pod isn't deleted. Even if some garbage is left behind.
- Try to think about how cleanup or reconciliation happens otherwise.

CNI Configuration Anatomy

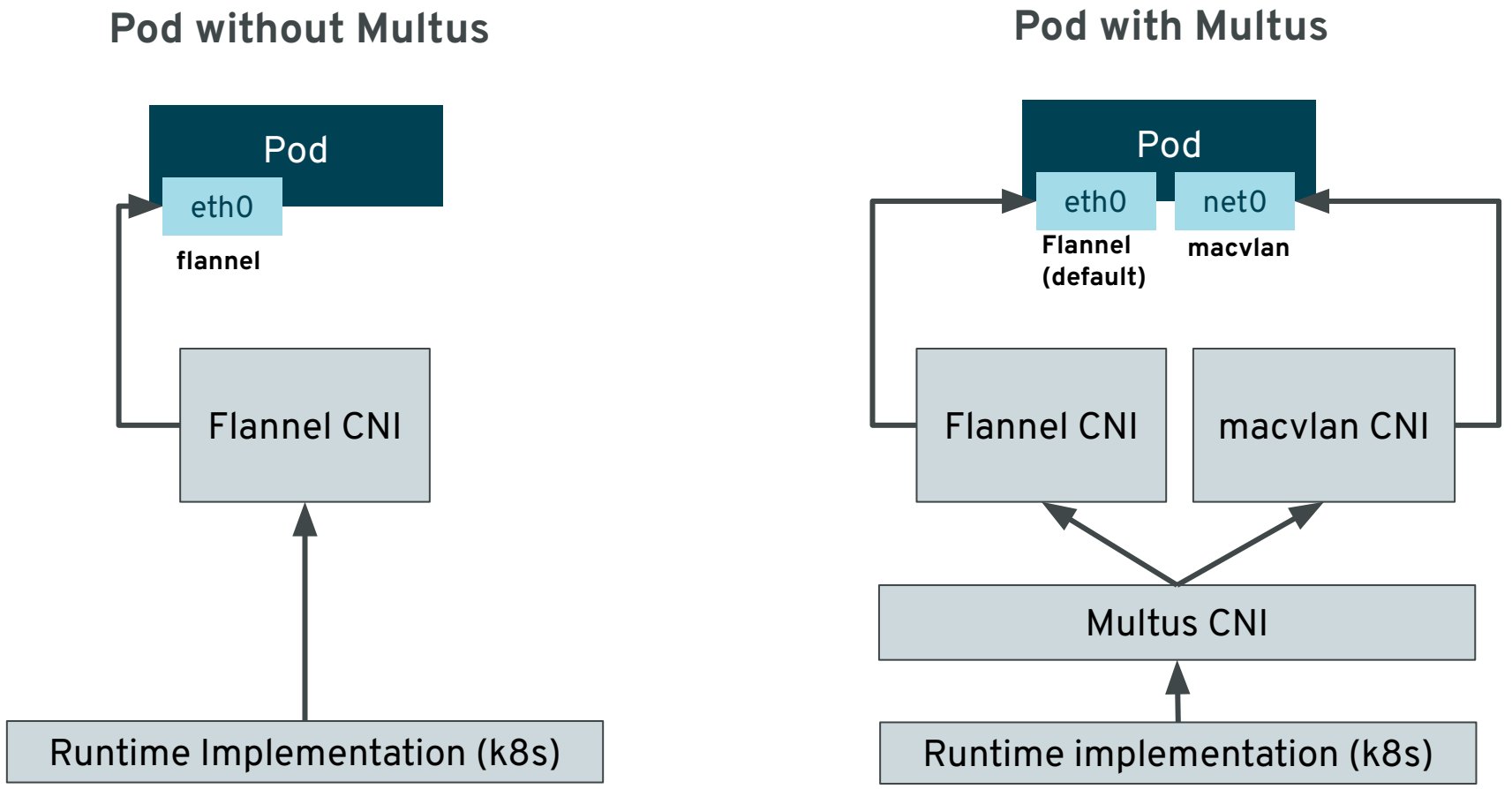
CNI Configuration: Chained plugins (or a "conflist")



Delegated Plugins

- The "ipam" section of a CNI Configuration is a "delegated plugin"
  - In this case, the idea is to assign IP addresses
- Called from within another plugin
- Multus CNI is by definition, a delegating plugin.

Multus - Multiplexing CNI ADD



Debugging CNI: Back to Basics

Wait, where did I put that? Know your paths

Two things we want to know for CNI paths:

- CNI Binary Directory (we call it "the bin dir")
- CNI Configuration Directory ("the conf dir")

The defaults are found in the K8s Network Plugins docs but are:

- Bin dir: /opt/cni/bin
- Conf dir: /etc/cni/net.d

DON'T necessarily trust the defaults! Check your kubelet configuration to double check. Check the parameters for `--cni-bin-dir` and `--cni-conf-dir`

DON'T forget your runtime! CRIO allows configuration of CNI bindir and confdir. AND CRIO allows for multiples for each of them!

```
$ ps ax | grep kubelet
$ cat /var/lib/kubelet/config.yaml
```

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
```

Check your CNI configuration directory (typically /etc/cni/net.d)

If there's not a file there, the kubelet won't mark the node in a "Ready" state. The first file in "lexigraphical order" (aka "ASCIIbetical order") is the CNI configuration used, generally you can figure it out with "ls -l" in your conf dir.

```
[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
```

Static pods are your friend

- If you're already debugging a node, try using a static pod [k8s docs]
- Saves you from having to setup nodeselectors
- Easy access for kubelet and flat file logs

```
[centos@kube-singlehost-master ~]$ sudo cp samplepod.yml /etc/kubernetes/manifests/
[centos@kube-singlehost-master ~]$ kubectl get pods | grep -Pi "name|sample"
NAME                                READY   STATUS    RESTARTS      AGE
samplepod-kube-singlehost-master    1/1     Running   1 (17s ago)   6s
[centos@kube-singlehost-master ~]$ sudo journalctl -u kubelet -n 50 | grep -i samplepod | tail -n 1
Apr 29 13:21:32 kube-singlehost-master kubelet[30386]: I0429 13:21:32.715672   11955 event.go:282] Event(v1.ObjectReference{Kind:"Pod", Namespace:
"default", Name:"samplepod-kube-singlehost-master", UID:"0c2230c4-8309-4881-a0d2-ff5faa8894a7", APIVersion:"v1", ResourceVersion:"4786400", FieldP
ath:""}): type: 'Normal' reason: 'AddedInterface' Add net1 [192.168.1.207/24] from default/macvlan-conf
[centos@kube-singlehost-master ~]$ tail -n1 /var/log/multus.log
2022-04-29T13:21:32Z [debug] GetK8sClient: /etc/cni/net.d/multus.d/multus.kubeconfig, &{0xc0003aa000 0xc0002c6380 0xc0004275c0 0xc00033ec80}
[centos@kube-singlehost-master ~]$
```

Use a JSON Linter!

- Often the errors from bad JSON – just aren't very helpful.
- Check early, check often.
- Our favorite linters
  - jq is ideal as you can use it from the command line
  - Or use a web linter like jsonlint.com
- Don't forget the requirements: Especially CNI Version!
  - Not having a version field will cause problems between minor versions, e.g. check is not introduced until CNI 0.4.0.

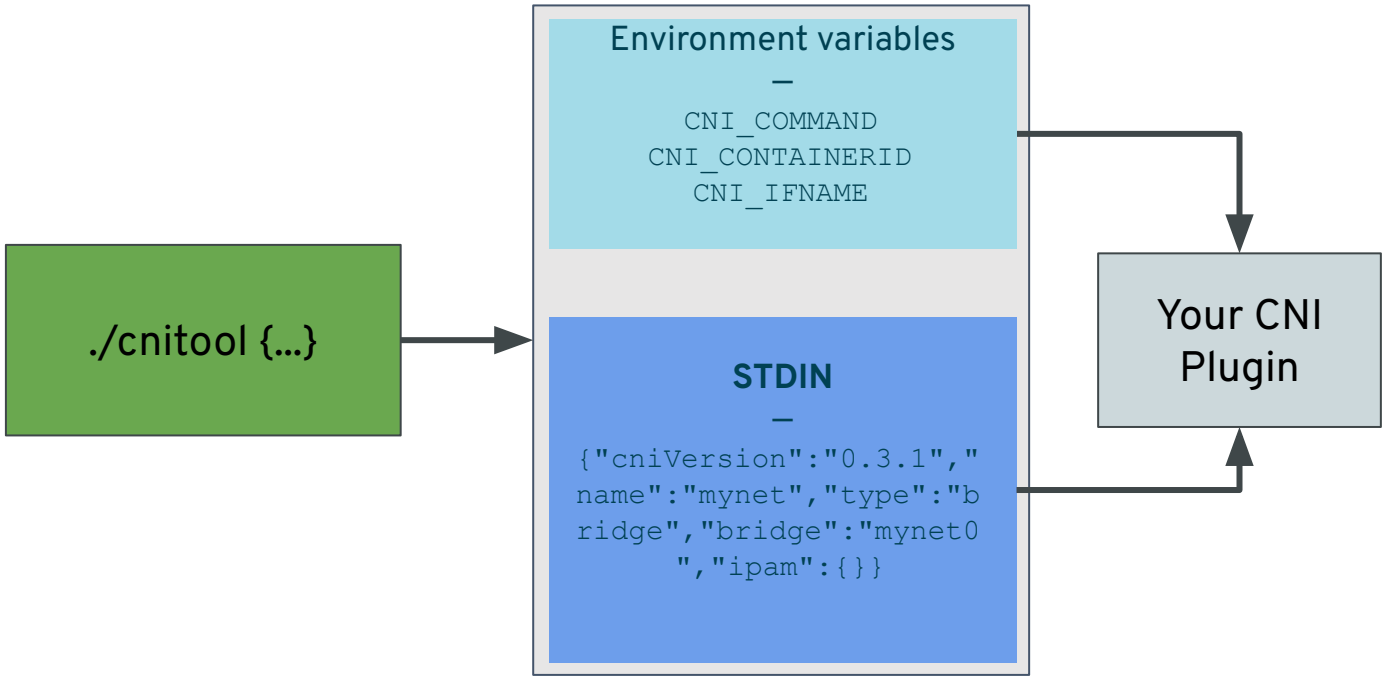
All the logging!

- Things to check first
  - The "kubectl describe" for a pod
    - If you're using Multus CNI: See the "events" section first
  - The kubelet logs
  - Container runtime logs (crio/containerd)
- Check to see if the plugin you're using has discrete logging
  - Then! Be mindful if the logs are in a pod, or on a host.
- If you're a developer: Add some debug-level logging.
  - Include the CNI input in your logging, helps you replay and rebuild what happened.

CNI Tool

CNI Tool: Your CNI swiss army knife

- Full tutorial / DIY workshop @ https://dougbtv.com/nfvpe/2021/05/14/using-cnitool/
- It allows you to execute your plugins without having to launch a pod, cnitool calls your binary with the ENV variables and CNI configs.

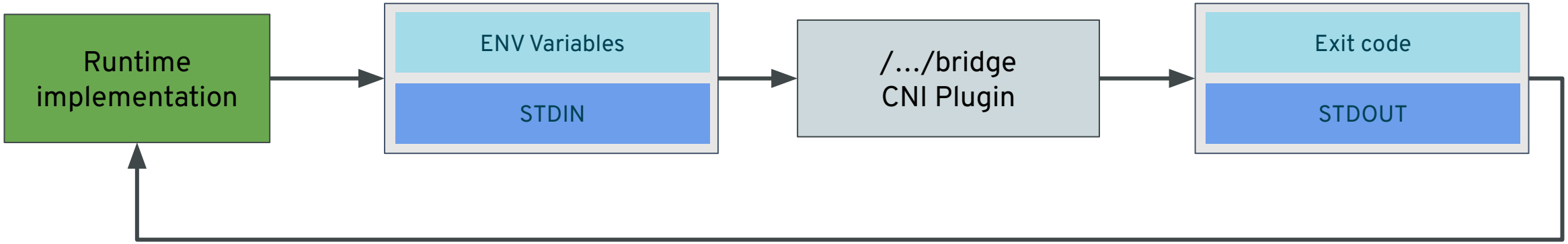


Danger Demo!

Thick Plugin vs. Thin Plugin architectures

Thin Plugins

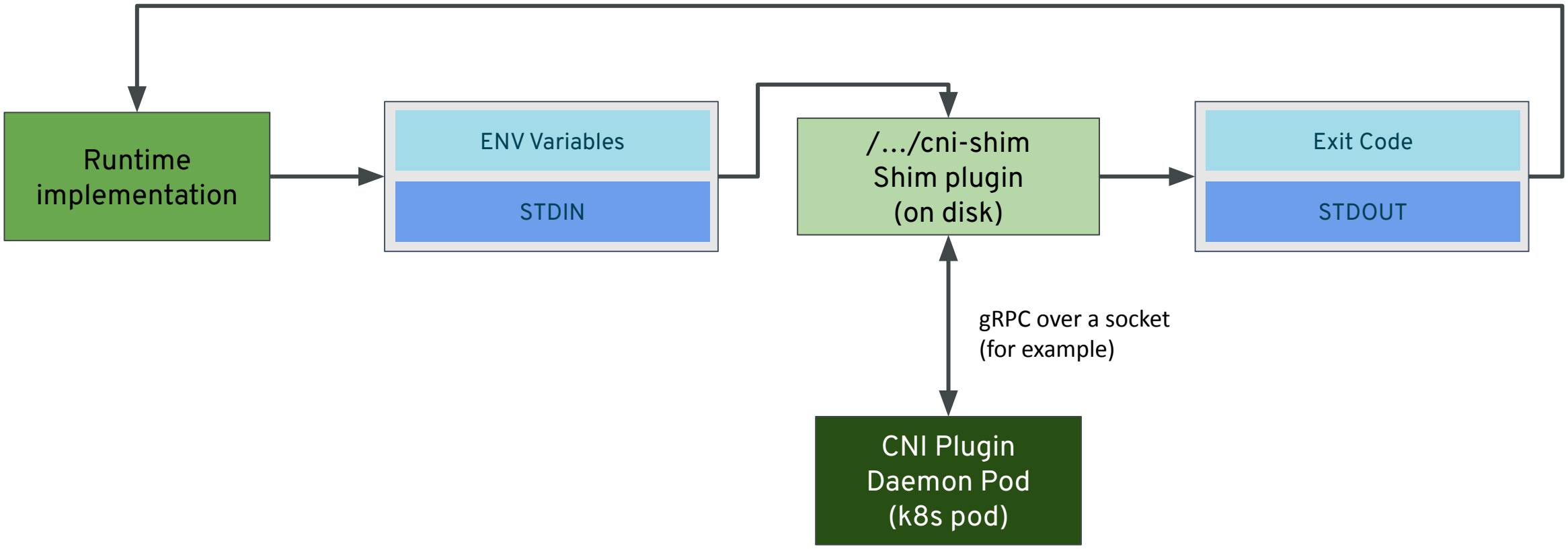
Run as a one-shot process.



Thick Plugins

Run with a "CNI shim" that lives on disk and a daemon that runs as a pod.

The shim's job is just to handle CNI input/output and let the daemon do the bulk of the work.




```
$ kubectl logs -f my-cni-daemon
```

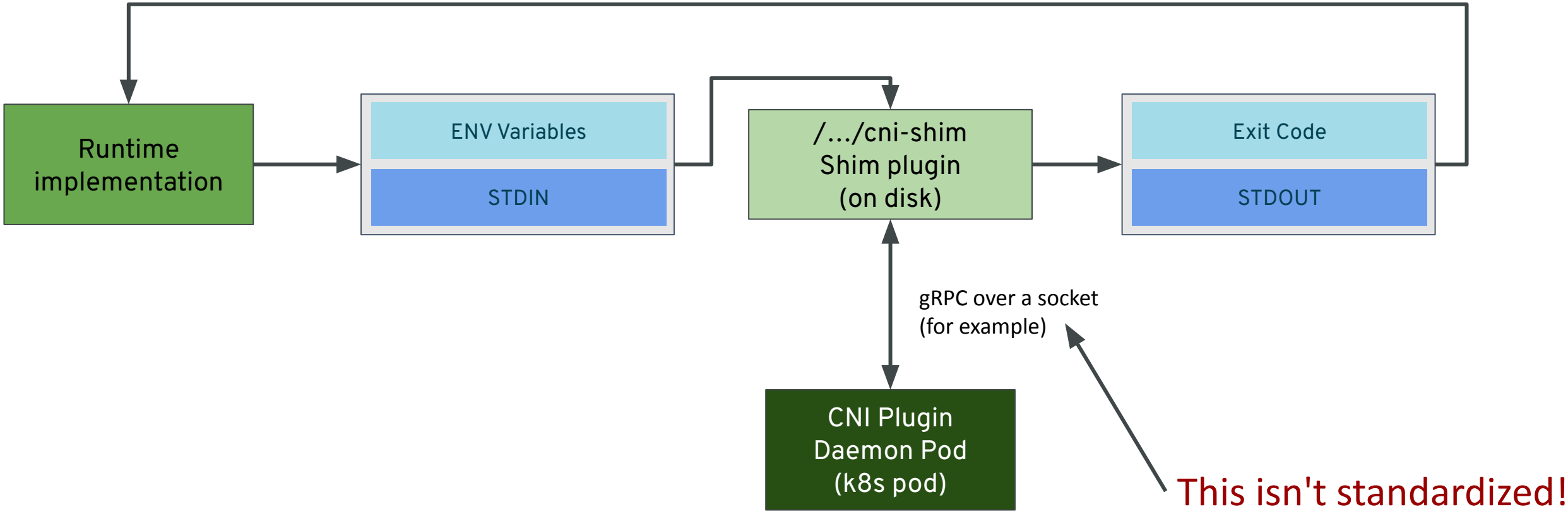
…so much better than logging into the node.

CNI 2.0: What the future holds

What do we want in CNI 2.0?

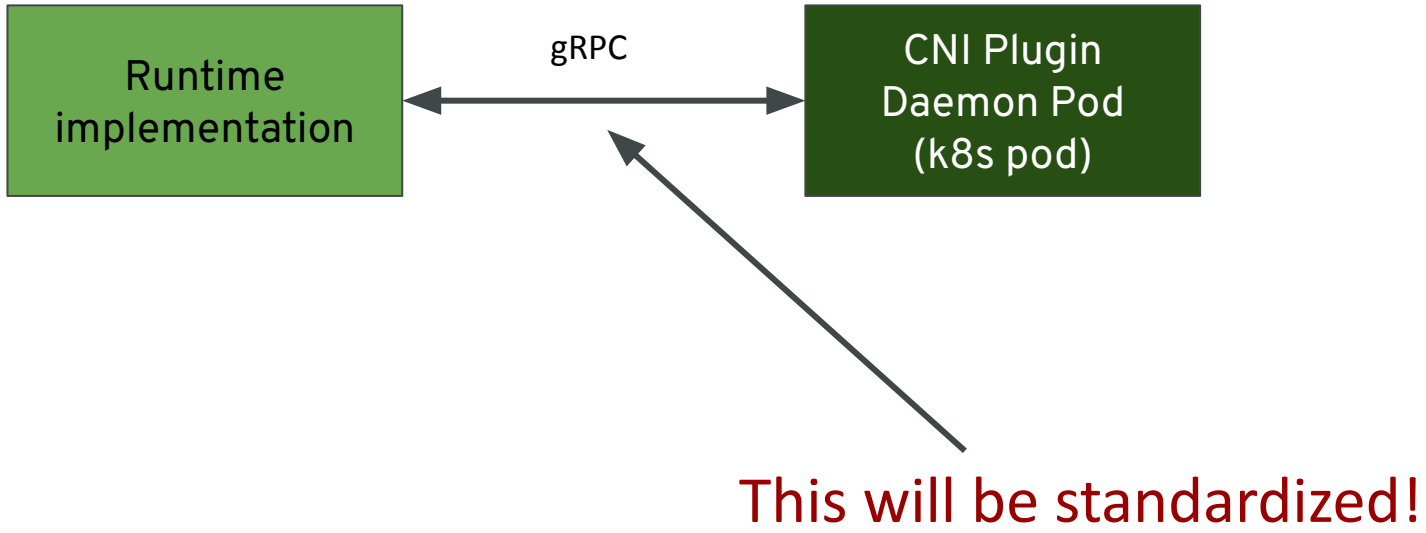
- CNI plugins running in pods, instead of on the host.
  - …They still need host access, though. Can we standardize this?
  - Today most plugins are installed by writing binaries to disk, there must be a better way?
- JSON, we think it's better for computers than it is for humans.
- A better way to handle devices.
- Richer runtime capability
  - That is, not just ADD/DEL/CHECK
  - We can validate and correct state during pod lifecycle if we have resident daemons that
  - Otherwise, we could kill the pod like a failed CHECK does today.
- CNI 1.0 doesn't capture network lifecycle
  - For example: Bridge CNI creates a linux bridge, but can't delete them if all networks are torn down.
  - CNI 1.0 doesn't consider race conditions for shared resources.
  - CNI 1.0 doesn't provide an official uninstall path (see: running in pods above)

Thick Plugins today



Thick Plugins in CNI 2.0

(A theoretical simplification, also see upstream discussion)



Join us in the Network Plumbing Working Group or CNI Maintainers for further discussion!

github.com/k8snetworkplumbingwg/community

Thank you! Any questions?