redhat.

Everyone Gets a Data Plane! Multi-Networking Kubernetes with the NPWG Spec

December 2018

Dan Williams & Doug Smith
Red Hat, Inc.

@dcbw

Dan Williams

- Member of Networking Services Team at Red Hat
- Focus on container networking and orchestration in OpenShift, Kubernetes, CNI, and related projects
- Co-chair of Kubernetes SIG Network, lead of the Network Plumbing Working Group, CNI maintainer

@dougbtv

Doug Smith

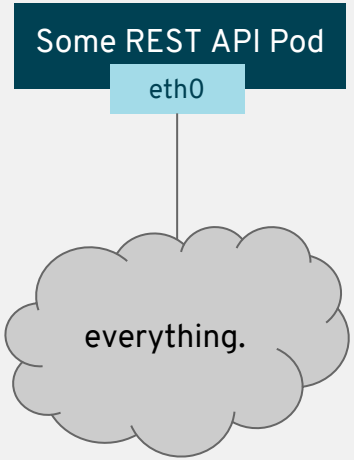
- Member of the NFV Partner Engineering team in Red Hat's Office of the CTO
- Focus on analyzing gaps in containerized workloads for NFV, including container networking & orchestration (e.g. Kube & OpenShift)
- Blog: https://dougbtv.com

AGENDA

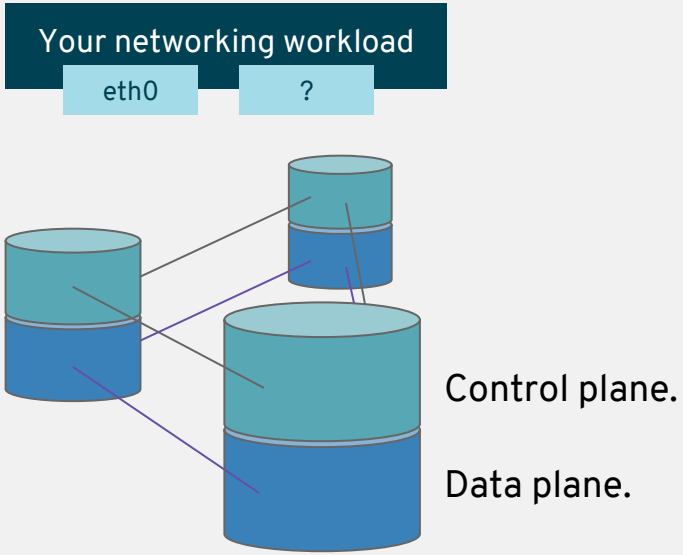
- About the Network Plumbing Working Group
- Overview of the Multi-Network specification
- Multi-networked pods with a CNI meta plugin
- Key concept overview
- Configuration overview
- What's next?

WHAT IF YOU NEED A DATA PLANE IN KUBERNETES?

It’s straightforward for web scale.



But what about for special networking workloads?



WHAT KIND OF WORKLOADS DO WE MEAN?

- High bandwidth
- Specific latency requirements or QoS
- Segregated networks
- Legacy network resources

NETWORK PLUMBING WORKING GROUP

Network Plumbing Working Group

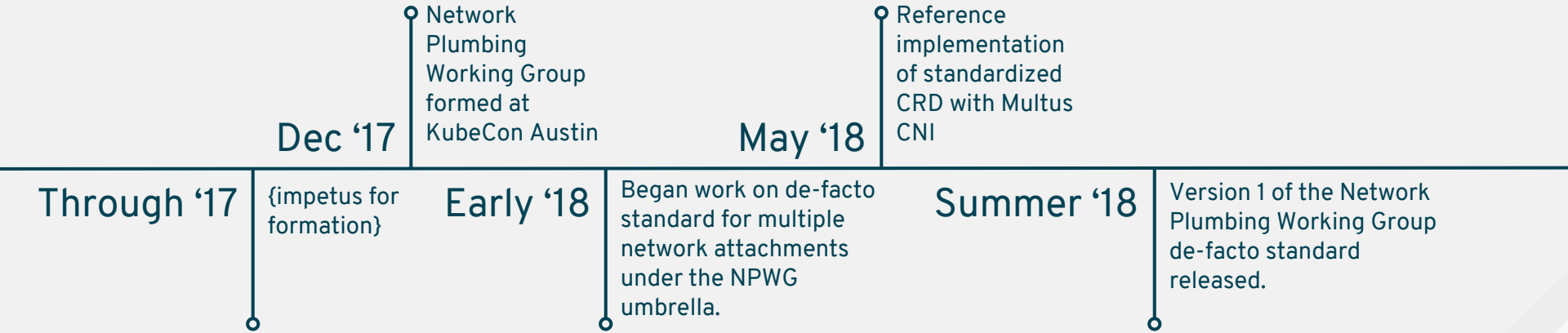
An informal offshoot of Kubernetes SIG-Network

Red Hat helped found the group during Kubecon 2017 to address lower level networking issues in Kubernetes

- Currently focused on multiple network attachments using an out-of-tree solution.
- Gather use-cases and propose standard specification
- Provide reference CNI plugin implementing standard specification
- Refine proposals/PoCs internally before sending up to Kubernetes SIG-Network
- Plan to expand to further advanced networking use-cases
- Meets every other Thursday opposite SIG-Network (same Zoom channel)
- Meeting recordings on YouTube.

What's been happening with multi-networking?

A brief history of how our upstream collaboration has evolved.



NPWG Multi-Network Specification v1

Something actually happens...

Goals:

- Short-term solution for multiple network attachments per pod
- No changes to the Kubernetes API or expected network behavior
- Light-weight standard for network attachment definitions and status reporting
- Specify behavior of CNI "meta-plugins" for multiple network attachments
- Coordinate with Resource Management WG on network resource management

Specification:
https://github.com/K8sNetworkPlumbingWG/multi-net-spec/blob/master/%5Bv1%5D%20Kubernetes%20Network%20Custom%20Resource%20Definition%20De-facto%20Standard.md

NPWG Multi-Network Specification v1

Overview:

- Pod annotation to select network attachments

```
k8s.v1.cni.cncf.io/networks: foobar
```

- Pod annotation reporting network attachment status

```
k8s.v1.cni.cncf.io/network-status: |
    {
      "name": "foobar",
      "interface": "eth5",
      "ips": [ "1.2.3.1/24", "2001:abba::2230/64" ],
      "mac": "02:11:22:33:44:54",
    },
```

- Custom Resource Definition (CRD) describing network attachments

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: foobar
```

- Requirements for CNI Delegating Plugins ("meta plugins")

NPWG Additional Components

- Go client code for the NPWG CRD object
- Admission controller for CRD and annotation validation
- In-progress/upcoming
  - Access control for pod network annotations
  - Go library code for selection/status annotations

MULTI-NETWORKED PODS WITH A CNI META PLUGIN

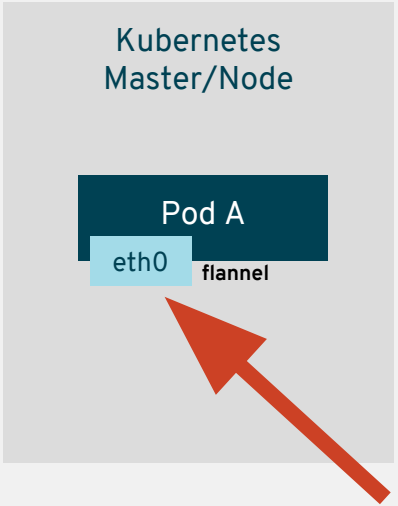
https://github.com/Intel-Corp/multus-cni

Red Hat in collaboration with Intel & the Network Plumbing Working Group is using Multus as part of a reference implementation.

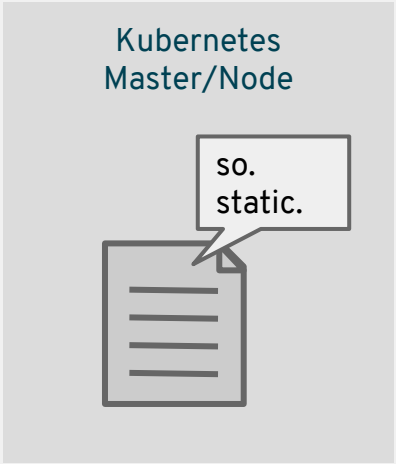
Multus CNI is a “meta plugin” for Kubernetes CNI which enables one to create multiple network interfaces per pod. It allows one to assign a CNI plugin to each interface created in the pod.

THE PROBLEM

#1 Each pod only has one network interface

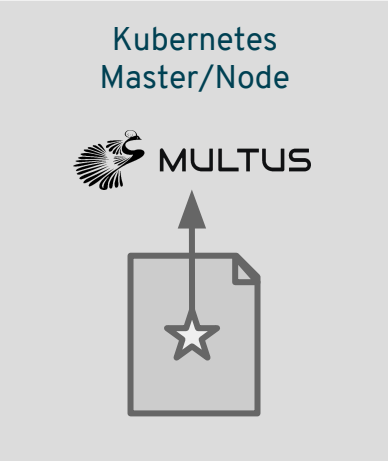


#2 Each master/node has only one static CNI configuration

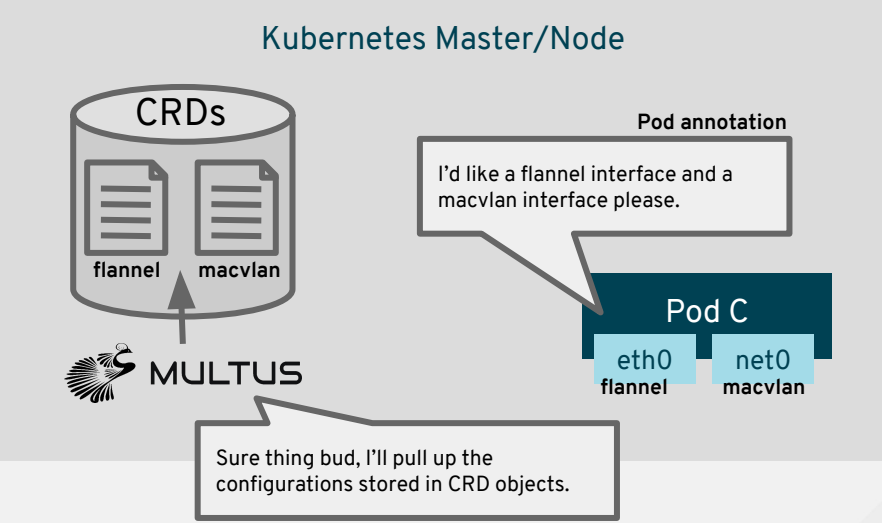


THE SOLUTION

Static CNI configuration points to Multus

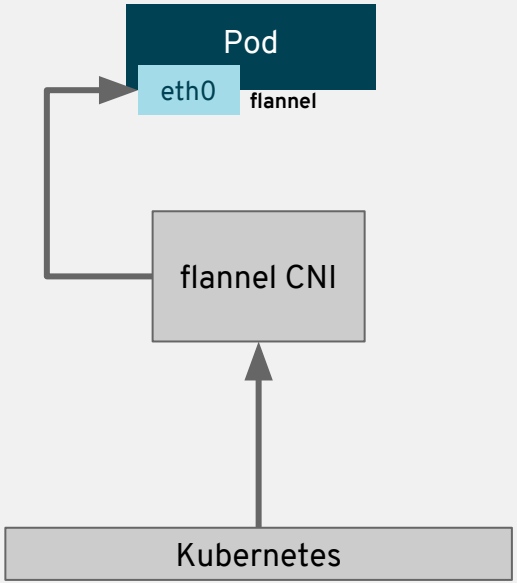


Each subsequent CNI plugin, as called by Multus, has configurations which are defined in CRD objects

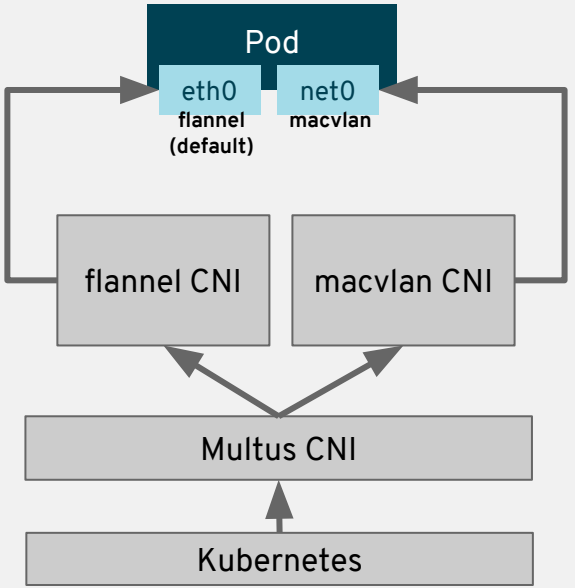


WHAT MULTUS DOES

Pod without Multus



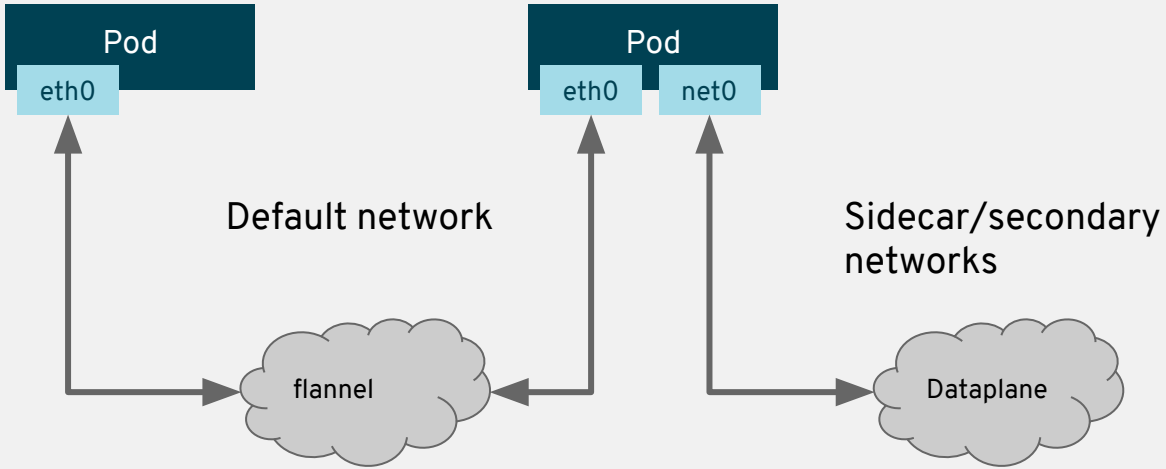
Pod with Multus



KEY CONCEPTS

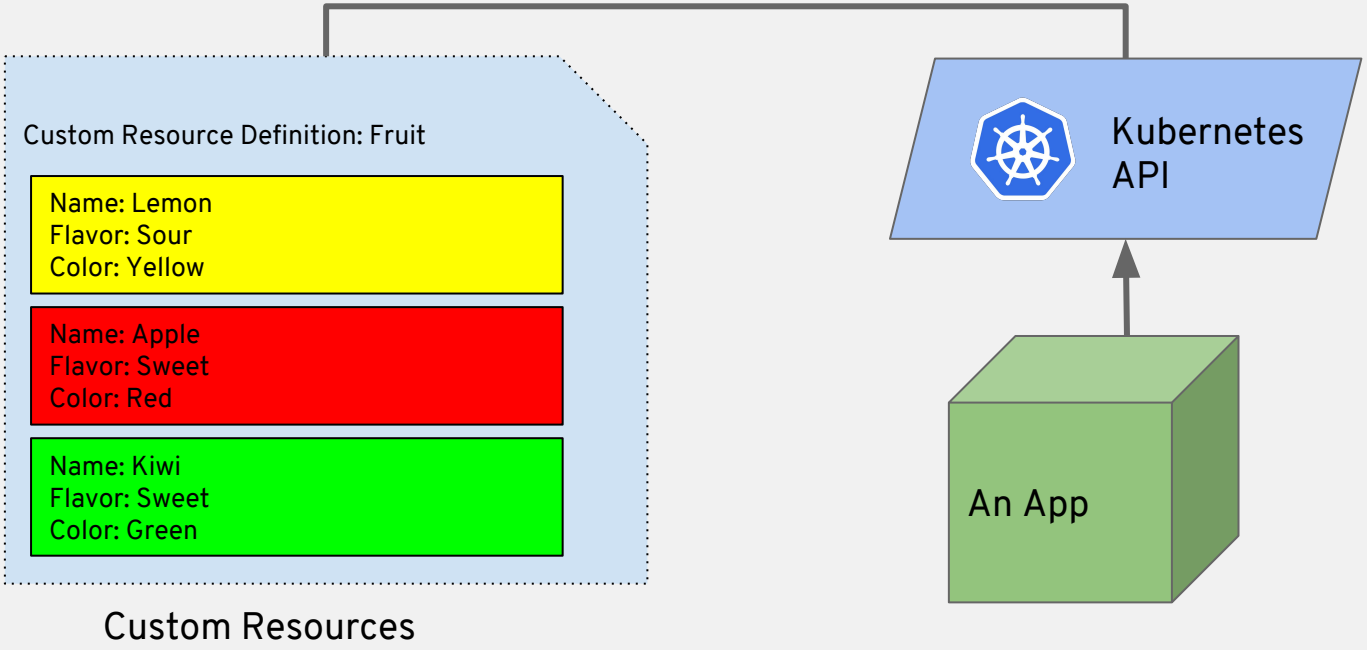
Default Network

Pod-to-pod communication is always available.



CRDs - Custom Resource Definitions

A way to customize the Kubernetes API to store data for applications.



Standardized CRD

As created by the Network Plumbing Working Group.

Pod annotations

```
apiVersion: v1
kind: Pod
metadata:
  name: pod_c
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
        { "name": "control-plane" },
        { "name": "data-plane" }
    ]'
spec:
  containers: [...]
```

Maps to...

CRD Object

```
Name:         data-plane
Namespace:    default
Labels:       <none>
Annotations:  <none>
API Version:  cni.cncf.io/v1
Args:         [ { "master": "eth0", "mode": "bridge", ...
Kind:         Network
Plugin:       macvlan
Metadata: [...]
```

The specification uses annotations to call out a list of intended network attachments as “sidecar networks”

CNI network configurations are packed inside CRD objects.

CONFIGURATION OVERVIEW

How do I add a configuration for an additional interface?

```
$ $ cat <<EOF | kubectl create -f -
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: macvlan-conf
spec:
  config: '{
            "cniVersion": "0.3.0",
            "type": "macvlan",
            "master": "eth0",
            "mode": "bridge",
            "ipam": {
                "type": "dhcp"
            }
        }'
EOF
```

You pack up a JSON CNI configuration and create it as a CRD object. We'll reference the name when we create a pod that wants to use this configuration.

How do I start a pod with an additional interface?

```
$ cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Pod
metadata:
  name: bothpod
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-conf
spec:
  containers:
  - name: bothpod
    image: dougbtv/nginx-toolbox
    ports:
    - containerPort: 8080
EOF
```

Add a comma delimited list of names from the previously loaded configurations, and additional interfaces will be attached to each pod referencing this annotation.

What's the result after I've started that pod?

```
$ kubectl exec -it bothpod -- ip a | grep -A2 "@"
3: eth0@if10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue
state UP group default
    link/ether 0a:58:0a:81:00:04 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.129.0.4/23 brd 10.129.1.255 scope global eth0
--
  4: net0@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UNKNOWN group default
    link/ether 9e:1d:f5:03:2f:17 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.1.201/24 scope global net0
```

eth0 will always be attached to your "default network" / pod network, in this case flannel.

Additional interfaces named net*N* will attach with the CNI configuration as defined in each CRD object.

WHAT'S NEXT?

Iteration!

- Develop proposal and components for Services on additional networks
- Enhanced security for additional networks
- Refinements to the NPWG specification
- Conformance test framework
- Joint efforts on resource management and device plugins

THAT'S WHERE WE NEED YOU!

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHat

youtube.com/user/RedHatVideos