Large-scale Network Simulations in Kubernetes
$ whoami - Michael Kashin

Network Engineer/Architect/Developer @Arista Networks

Github: @networkop

Twitter: @networkop1

Blog: networkop.co.uk
Problem Statement
Why simulating physical network?

• Studying/Labbing
• Network design validation
• Network testing
Lay of the land

• Multiple commercial and open-source products exist:
  • GNS3
  • EVE-NG
  • vrnetlab
• Most of them are single-node-only deployments
• Work well for studying/labbing
• Vendor’s virtual devices are huge VMs (~10GB+2vCPU)
Kubernetes as a network simulation platform

- Compute resources are cheap and abundant
- Kubernetes is the pervasive “cloud-native operating system”

but

- Kubernetes IP-per-pod networking model is restrictive
- Existing CNI plugins address the NFV use-cases (e.g. Multus)
- Somewhere there are still bridges
meshnet
Main goals

• Connect arbitrary number links inside a pod
  • **veth** - if pods are on the same node
  • **vxlan** - if pods are on different nodes
  • **macvlan** - to connect to the underlay
• All links will be point-to-point, without any bridges (thanks [koko](#))
• Non-disruptive to other Kubernetes resources
• Asynchronous operation
How to build a custom CNI plugin

• Talks, blogs and existing examples:
  • Kubernetes and CNI - Kubecon 2018
  • Writing a CNI plugin with Bash
  • EVPN CNI plugin
  • Workflow for writing CNI plugins
  • Flannel, Weave, AWS CNI

• CNI specification
• CNI standard plugins
CNI plugin ADD call

CNI_COMMAND=ADD \ 
CNI_CONTAINERID=$id \ 
CNI_NETNS=/proc/$pid/ns/net \ 
CNI_ARGS=K8S_POD_NAMESPACE=$ns;K8S_POD_NAME=$name

/opt/cni/bin/my-plugin < /etc/cni/net.d/my-config
Choosing a backend

Backend is used to synchronise runtime state:
  • Namespaces
  • Nodes where pods are deployed
  • Read the topology data

1. etcd seems like an obvious choice but external etcd cluster requires an operator
2. CRDs require RBAC and user credentials
7.2 gRPC Update { IP(node-1), VNI }
2-node topology CRDs

- apiVersion: networkop.co.uk/v1beta1
  kind: Topology
  metadata:
    name: r1
  spec:
    links:
      - uid: 1
        peer_pod: r2
        local_intf: eth1
        local_ip: 169.254.0.1/24
        peer_intf: eth1
        peer_ip: 169.254.0.2/24

- apiVersion: networkop.co.uk/v1beta1
  kind: Topology
  metadata:
    name: r2
  spec:
    links:
      - uid: 1
        peer_pod: r1
        local_intf: eth1
        local_ip: 159.254.0.2/24
        peer_intf: eth1
        peer_ip: 169.254.0.1/24
k8s-topo - topology orchestrator
k8s-topo 2-node topology definition

---

links:
- endpoints: ["r-1:eth1:169.254.0.1/16", "r-2:eth1:169.254.0.2/16"]
End-to-end workflow

1. Start with a working Kubernetes cluster
2. Install meshnet-cni plugin
3. Install k8s-topo
4. Create the topology definition YAML file
5. (Optional) Create startup configurations
6. Instantiate the topology