Peeling Layers: A Deep Dive into Kubernetes Networking

Angela Chin
Senior Software Engineer, Pivotal
@angelaschin
Overview

- Kubernetes Networking 101
  - Container/Pod Network
  - Service Network
  - Kube Proxy
- Kubernetes Networking 301
  - aka Service Meshes
Overview

● Kubernetes Networking 101
  ○ Container/Pod Network
  ○ Service Network
  ○ Kube Proxy

● Kubernetes Networking 301
  ○ aka Service Meshes
Let’s say we have a cluster...
Let’s deploy a pod!
Terminology

- **Pod**: group of one or more containers that share some resources, including a network namespace
Pod Networking

- Every single pod gets an unique ip
- This ip is allocated from the pod network
IP ≠ Routability
**Terminology**

- **Pod**: group of one or more containers that share some resources, including a network namespace.
- **Pod Network**: virtual network that allows pods to communicate with one another.
CNI (Container Network Interface)

- **CNI** is a standard API between container runtimes and container network implementations.
- Kubernetes allows you to “bring-your-own” CNI plugin to set up your own container network.
- The most common CNI plugin in Kubernetes is **Flannel**, but each cloud engine has their own CNI plugin implementation.
Flannel-specific

worker

10.0.10.2

Flannel VXLAN

bridge

veth0

eth0

10.100.0.10

pod

10.100.0.10

pod
Flannel-specific
Differences Between CNI Plugins

- Performance
- Network isolation
- Policy
  - Flannel doesn’t support NetworkPolicy
CNI Plugins

- Project Calico - a layer 3 virtual network
- Weave - a multi-host Docker network
- Contiv Networking - policy networking for various use cases
- SR-IOV
- Cilium - BPF & XDP for containers
- Infoblox - enterprise IP address management for containers
- Multus - a Multi plugin
- Romana - Layer 3 CNI plugin supporting network policy for Kubernetes
- CNI-Genie - generic CNI network plugin
- Nuage CNI - Nuage Networks SDN plugin for network policy-kubernetes support
- Silk - a CNI plugin designed for Cloud Foundry
- Linen - a CNI plugin designed for overlay networks with Open vSwitch and fit in SDN/OpenFlow network environment
- Vhostuser - a Dataplane network plugin - Supports OVS-DPDK & VPP
- Amazon ECS CNI Plugins - a collection of CNI Plugins to configure containers with Amazon EC2 elastic network interfaces (ENIs)
- Bonding CNI - a Link aggregating plugin to address failover and high availability network
- ovn-kubernetes - an container network plugin built on Open vSwitch (OVS) and Open Virtual Networking (OVN) with support for both Linux and Windows
- Juniper Contrail / TungstenFabric - Provides overlay SDN solution, delivering multicloud networking, hybrid cloud networking, simultaneous overlay-underlay support, network policy enforcement, network isolation, service chaining and flexible load balancing
- Knitter - a CNI plugin supporting multiple networking for Kubernetes
- DANM - a CNI-compliant networking solution for Telco workloads running on Kubernetes
- VMware NSX - a CNI plugin that enables automated NSX L2/L3 networking and L4/L7 Load Balancing; network isolation at the pod, node, and cluster level; and zero-trust security policy for your Kubernetes cluster.
- cni-route-override - a meta CNI plugin that override route information
- Terway - a collection of CNI Plugins based on alibaba cloud VPC/ECS network product
... but pods are the building block for Kubernetes
Pods are ephemeral
How should we be discovering and referencing our applications?
Overview

- **Kubernetes Networking 101**
  - Container/Pod Network
  - **Service Network**
  - Kube Proxy

- **Kubernetes Networking 301**
  - aka Service Meshes
Let’s create a service!
Terminology

- **Pod**: group of one or more containers that share some resources, including a network namespace
- **Pod Network**: virtual network that allows pods to communicate with one another
- **Service**: resource that represents a logical grouping of pods
Pods & Services

```
kind: Pod
apiVersion: v1
metadata:
  name: oss-app
  labels:
    app: oss
spec:
  containers:
    - name: oss-app
      image: pivotal/dummy-image

kind: Service
apiVersion: v1
metadata:
  name: oss-service
spec:
  selector:
    app: oss
```
Pods & Services

kind: Pod
apiVersion: v1
metadata:
  name: oss-app
labels:
  app: oss
spec:
  containers:
  - name: oss-app
    image: pivotal/dummy-image

kind: Service
apiVersion: v1
metadata:
  name: oss-service
spec:
  selector:
    app: oss
Pods & Services

kind: Pod
apiVersion: v1
metadata:
  name: oss-app
  labels:
    app: oss
spec:
  containers:
    - name: oss-app
      image: pivotal/dummy-image

kind: Service
apiVersion: v1
metadata:
  name: oss-service
spec:
  selector:
    app: oss
Pods & Services

kind: Pod
apiVersion: v1
metadata:
  name: oss-app
  labels:
    app: oss
spec:
  containers:
    - name: oss-app
      image: pivotal/dummy-image

kind: Service
apiVersion: v1
metadata:
  name: oss-service
spec:
  selector:
    app: oss
Services

Three types of services:
Services

Three types of services:

1. ClusterIP
2. NodePort
3. LoadBalancer
Services

Three types of services:

1. **ClusterIP**
2. **NodePort**
3. **LoadBalancer**
Services

Three types of services:

1. ClusterIP - gets an unique IP from the service network
2. NodePort
3. LoadBalancer
Services

Three types of services:

1. **ClusterIP** - gets an unique **IP** from the **service network**
2. **NodePort** - gets an unique **port** open on **every worker node**
3. **LoadBalancer**
Services

Three types of services:

1. ClusterIP - gets an unique **IP** from the **service network**
2. NodePort - gets an unique **port** open on **every worker node**
3. LoadBalancer - gets an unique **external** **ip**
Services

Three types of services:

1. ClusterIP - gets an unique IP from the service network
2. NodePort - gets an unique port open on every worker node
3. LoadBalancer - gets an unique external ip

By default, services are created with type ClusterIP.
When would I want to use each?
<table>
<thead>
<tr>
<th>Cluster IP</th>
<th>Node Port</th>
<th>Load Balancer</th>
</tr>
</thead>
</table>
| ● Address is only reachable from other pods  
● Use for sensitive applications that shouldn’t be accessible from anything but other pods | ● Address is only reachable from within the private network  
● Can use in conjunction with a single load balancer to expose to public while cutting down on LB costs | ● Address is reachable from entire internet  
● Use if you want a dedicated load balancer for that service |
Service Network
## Terminology

- **Pod**: group of one or more containers that share some resources, including a network namespace.
- **Pod Network**: virtual network that allows pods to communicate with one another.
- **Service**: resource that represents a logical grouping of pods.
- **Service Network**: virtual network that allows communication between services.
Cluster IP

10.100.0.10

podA

10.200.0.53

svc

podB

10.100.0.11

10.0.10.2

worker

10.100.0.10

10.100.0.11
Node Port

- podA: 10.100.0.10
- podB: 10.100.0.11
- svc: 10.200.0.53
- 10.0.10.2 (worker)

Node Port

- podA: 10.100.0.10
- podB: 10.100.0.11
- svc: 10.200.0.53
- 10.0.10.2 (worker)
Load Balancer

10.0.10.2

10.200.0.53

podA: 10.100.0.10

podB: 10.100.0.11

worker

LB: 10.0.10.2

svc: :31000
Services can span nodes too!
How does this all get set up?
Overview

- Kubernetes Networking 101
  - Container/Pod Network
  - Service Network
  - Kube Proxy
- Kubernetes Networking 301
  - aka Service Meshes
Kube Proxy

- Runs on every VM
- Sets up the virtual networking
- 3 different modes it can run in
  - Userspace
  - Iptables
  - IP Virtual Server (IPVS)
**Iptables**

- Uses netfilter tables (specifically NAT) to re-direct traffic
- Packets make their way through iptables rules linearly
  - Doesn’t scale well
- Basic load balancing
- Plays more nicely with other programs that use iptables (CNI plugins)

**IPVS**

- Uses IPVS load balancer (built on netfilter)
- In-kernel hash tables make packets have optimized lookup
  - Lends to scale
- More sophisticated load balancing
- Sometimes doesn’t work with programs that use iptables
- IPVS kernel modules must be installed on the node!
Networking Caveats
Caveats

- Type Load Balancer is not supported by default
  - vSphere w/o NSX-T and OpenStack do not support out-of-the-box
- Depending on how you are provisioning your cluster, you may have a limited say in your network topology!
Kubernetes Networking 101
Overview

- Kubernetes Networking 101
  - Container/Pod Network
  - Service Network
  - Kube Proxy
- Kubernetes Networking 301
  - aka Service Meshes
Wait, where’s 201?
201: Ingress and NetworkPolicy
For the purposes of today’s talk, there is not enough time
Sorry 😞
Overview

- Kubernetes Networking 101
  - Container/Pod Network
  - Service Network
  - Kube Proxy
- Kubernetes Networking 301
  - aka Service Meshes
The dream
The dream: a resilient, production ready network
Considerations for production networking

- Retries
- Circuit breaking
- Load balancing
- Mutual TLS
- Configurable timeouts
- Metrics collection
How are these features typically implemented?
In-process architecture

A Library!
Multiple languages, Multiple libraries

Each with…

• Different features
• Different configuration
• Different quirks

*Polyglot shouldn't be painful!*
What if this was provided by the platform?
Out-of-process architecture

Your App

Separate process!

Some service
Out-of-process architecture

- retries
- load balancing
- mutual TLS
- timeouts
- metric collection
- etc.

Your business logic here
Out-of-process architecture

Client app — Your App (acting as a service)
Sidecars

- A separate process
- Runs as a container in each pod
- A proxy reachable via localhost
- Layer 4 (TCP) and Layer 7 (HTTP)
- Can proxy Ingress and Egress
- Provides all those features we want!
When every component has a sidecar... we get a SERVICE MESH
By having a common control plane, we can set sane defaults for all applications.
Retries

“If an error occurs, retry up to 3 times”
We can also utilize service meshes for more fine-grained control.
Load Balancing

“Send 80% traffic to V1, 20% V2”
Load Balancing

Control Plane

backend request

Sidecar

Frontend

Sidecar

Backend V1

Sidecar

Backend V2
Service Meshes

- Istio
- Linkerd
- Consul Connect
- Weave Mesh
- etc...
In summary
Summary

- There’s a lot to know about networking in Kubernetes!
- Customizations are abound with choices in **CNI Plugins** and **Service Meshes**
  - Allows for extensibility in regard to implementation of Kubernetes constructs such as NetworkPolicy and Ingress
- Different IAASes will have different constraints around networking
We’re Hiring!

pivotal.io/careers
Kube Proxy Mode

- Check what mode you are running in by looking at interface on VM
- If ipvs, `ip addr` should return an interface that references ipvs (default name is `kube-ipvs0`)