Apply the Can Opener of Enlightenment: Lifting the Lid Off Kubernetes Networking

Joe Thompson

Staff Solutions Engineer, HashiCorp

Who even is this guy?

Pronouns: he/him
Blood type: Caffeine-positive

How to get in touch:
- Kubernetes Slack: @kensey
- LinkedIn

In IT since my first job helping out with computers in my high school in 1994

Past employers: Mesosphere, Capital One, CoreOS, Red Hat, among others

Exposed to Kubernetes in early 2015 and working with it full-time since late 2015

Currently a Staff Solutions Engineer for HashiCorp

Why are we here?

- First and foremost, to learn, of course
  - Both about the foundations of Kubernetes networking and how we can apply knowledge we already have
- But also to understand that just because tools are old and well-worn doesn't mean they must be thrown away

Credit: FFD Restorations

The general situation

- Kubernetes lives ~~in a society~~ on a set of hosts
- It enables some very advanced network models...
- ...but those models are built on the same basic tools that have been around for a long... long... long... **long** time

Aren't there Kubernetes-level tools for all this?

- There sure are!
- You should definitely learn and use native tools like ephemeral debug containers, add-ons like Weave Scope, etc.
- But:
  - Sometimes you have to debug installing or using those tools
  - Sometimes high-level tools aren't telling you what you need to know
  - You might be in a situation where you're responsible to fix something and you don't have actual API access

Kubernetes Networking in Brief

A brief history note: the OSI model

- (Not the Open Source Initiative -- OSI = Open System Interconnection in this case)
- Grab Marty and the Doc because we're going back to the '80s!
- The OSI model was an attempt by ISO to create a standard abstraction of network comunications
  - 7 layers from physical to application
  - Not widely implemented in practice but still makes a convenient reference -- we still talk today about "layer 4" vs. "layer 7" proxies/etc.
  - Kubernetes networking is layered as well in ways that somewhat map to the OSI model

Kubernetes network layers

- At the bottom: the host network itself
- Host interface addresses are how the cluster components initially connect to each other
- We could call this the "physical layer" in OSI terms

Kubernetes network layers

- Next up, the container network
- The container runtime handles this, which is configured by CNI plugins, which are often intalled... with kubectl?!
  - It sounds like a circular situation but the answer is privileged containers running with host networking
- CNI plugins also have to set up some method for traffic between hosts to traverse the host layer
  - Lots of options here, and many plugins support more than one

Kubernetes network layers

- Next up the stack is the Service network
- Services are simple(...ish) abstractions to load balance traffic between pods in a set that matches certain labels
  - The "...ish" is doing a lot of work there -- see Tim Hockin's lightning talk about it linked at the end of this presentation
- Services have a set of layers all their own, from ClusterIP to NodePort to LoadBalancer

Kubernetes network layers

- Finally, there are higher-level network abstractions that build on Services
  - Ingress
  - Gateway API
  - Pretty much every service mesh
- Many of these things include (or are) layer-7 (there goes that OSI model talk again!) proxies
  - Service meshes are, in a sense, merely very complex applications

What all this is built on top of

Credit: Aysegul Alp

Host networking: the basic stuff

- Interfaces - you'll see them both inside and outside containers
- Routing - this is often where inter-node communication is set up by the CNI plugin
- `iptables` - the `nat` table is populated by the Service API to route traffic to pods backing each service
  - Sometimes Services show up in IPVS depending on how the cluster was configured

Host networking: namespaces

- Kernel namespaces were added to the 2.4 Linux kernel in 2002
- There are (currently) ~~seven~~ **eight** namespaces; mostly you will only need three:
  - network
  - pid
  - mount
- The last two will come in handy for some types of debugging

Host networking: more exotic things...

- A lot of service meshes and other network tooling are beginning to use eBPF for routing instead of sidecars
  - This is actually a (relatively) new thing!
  - A tiny VM in the kernel that validates code before running it
  - This enables a lot of things that used to require sidecars or special host config
  - This does makes things more opaque though, since this is **compiled code**, not human-readable manifests or output

How you can apply your can opener

Credit: Alexander Lesnitsky

Use the basic host tools

- They still work the same way and usually you won't have to do anything special to make these produce useful output
- No surprises here: `ip`/`ifconfig`, `route`, `iptables`
- You may also be able to use things like `openssl` at this level to debug TLS certs; sometimes you'll need to use that inside a container
- Downside: some things live in container namespaces so tools like `ss`/`netstat` may not show you what you need to know

Exec into a container

- Usually you will be trying to run a shell, but that's not a hard requirement
- Downside: What you run needs to either already exist in the container, or you need to install it -- in a minimized container you may not have a way to get what you need

Run a debug container

- A new container attached to an existing container's namespaces -- run anything you want, install anything you want
- Especially useful if you're trying to debug something that lives inside a minimized container
- Kubernetes now supports this through `kubectl debug`
- Downside: Requires you to have permission to spawn containers; can be pretty heavyweight

Use nsenter to enter a namespace directly

- Pick the namespaces you want to attach
- Run a shell/other command
- Downside: Somewhat less friendly than just doing a quick `docker run -it ...`

Demo: debugging a faulty Ingress

Wrapping up

Final Thoughts

- If there is one thing I want you to leave here with today, it is the belief that what you already know is not obsolete
  - You will need to learn new tools as time goes on (and so will I!)
  - You will sometimes still want to use the same tools you're comfortable with (and that's probably fine)

The right tool is the one that gets the job done most efficiently.

Further reading/etc.

KubeCon CloudNativeCon North America 2023

- Kubernetes the Hard Way
- Kubernetes docs on cluster networking, Services networking and service proxying
- Jérôme Petazzoni's Cgroups, namespaces, and beyond: what are containers made from?

Other sessions at this KubeCon worth checking out:

Monday: Lightning Talk: Why Service Is the Worst API in Kubernetes, and What We're Doing About It

Tuesday:

- Multi-Network, New Level of Cluster Multi-Tenancy
- Demystifying Service Mesh: Separating Hype from Practicality
- AdminNetworkPolicy: A New Kubernetes-Native API for Comprehensive Cluster-Wide Network Security

Wednesday:

- 11:55 – 12:30, W178: Gateway API: The Most Collaborative API in Kubernetes History Is GA
- 2:30 – 3:05, W181: Learning Kubernetes by Chaos - Breaking a Kubernetes Cluster to Understand the Components
- 3:25 – 4:00, W187: Network Policy API: Intro and Project Update
- 4:30 – 5:05, W175: Dungeons and Deployments: Leveling up in Kubernetes
- 5:25 – 6:00, W181: Service Proxy, API Gateway, Service Mesh: What's the Difference? What Do I Need?

Thank you!

KubeCon CloudNativeCon North America 2023

Slides:

tinyurl.com/apply-the-can-opener

Feedback: