SUSEcon digital '20

Container Networking: From The Shallow End To The Diving Pool

TUT-1294

Mark Darnell, Senior Product Manager (mdarnell@suse.com)

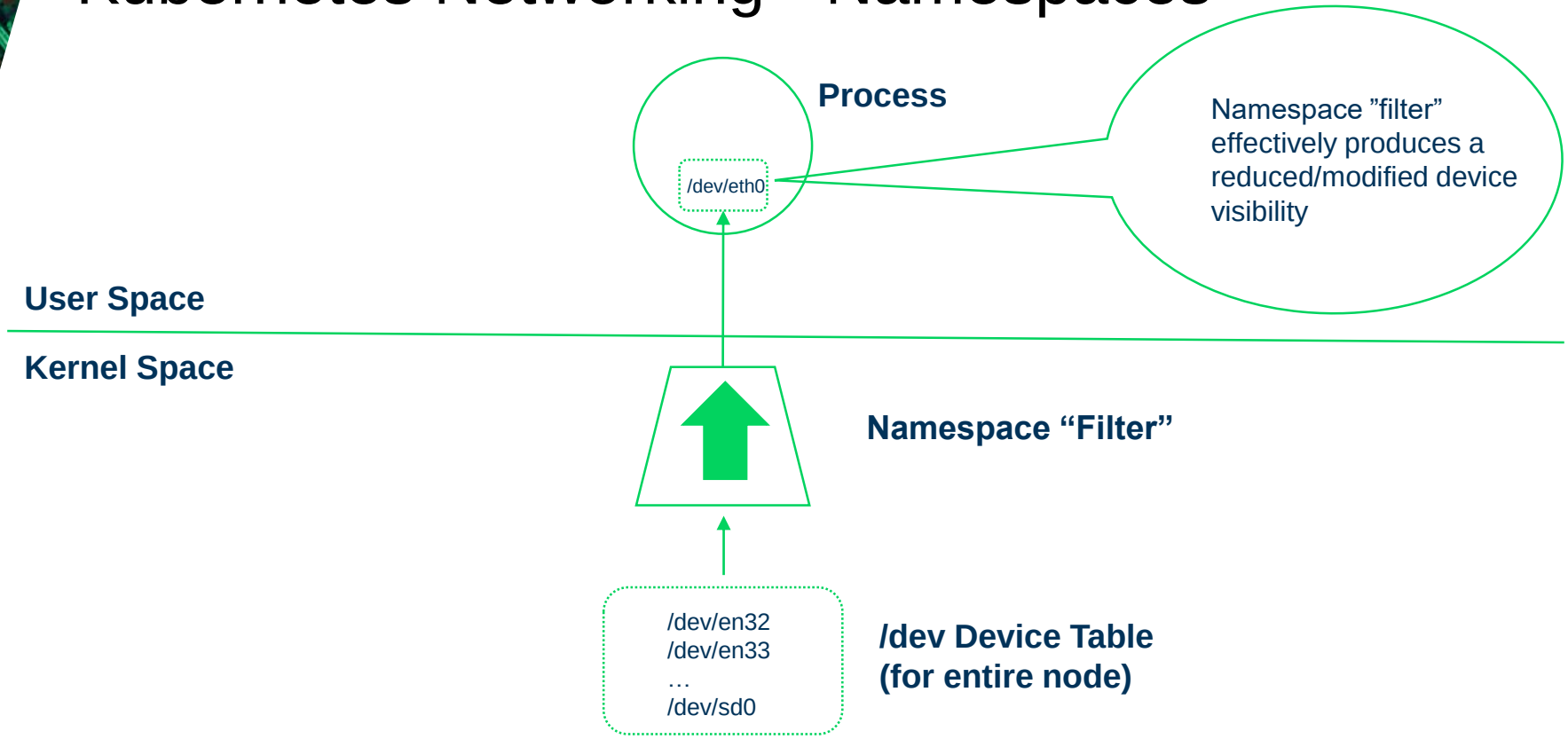
SUSE

Agenda

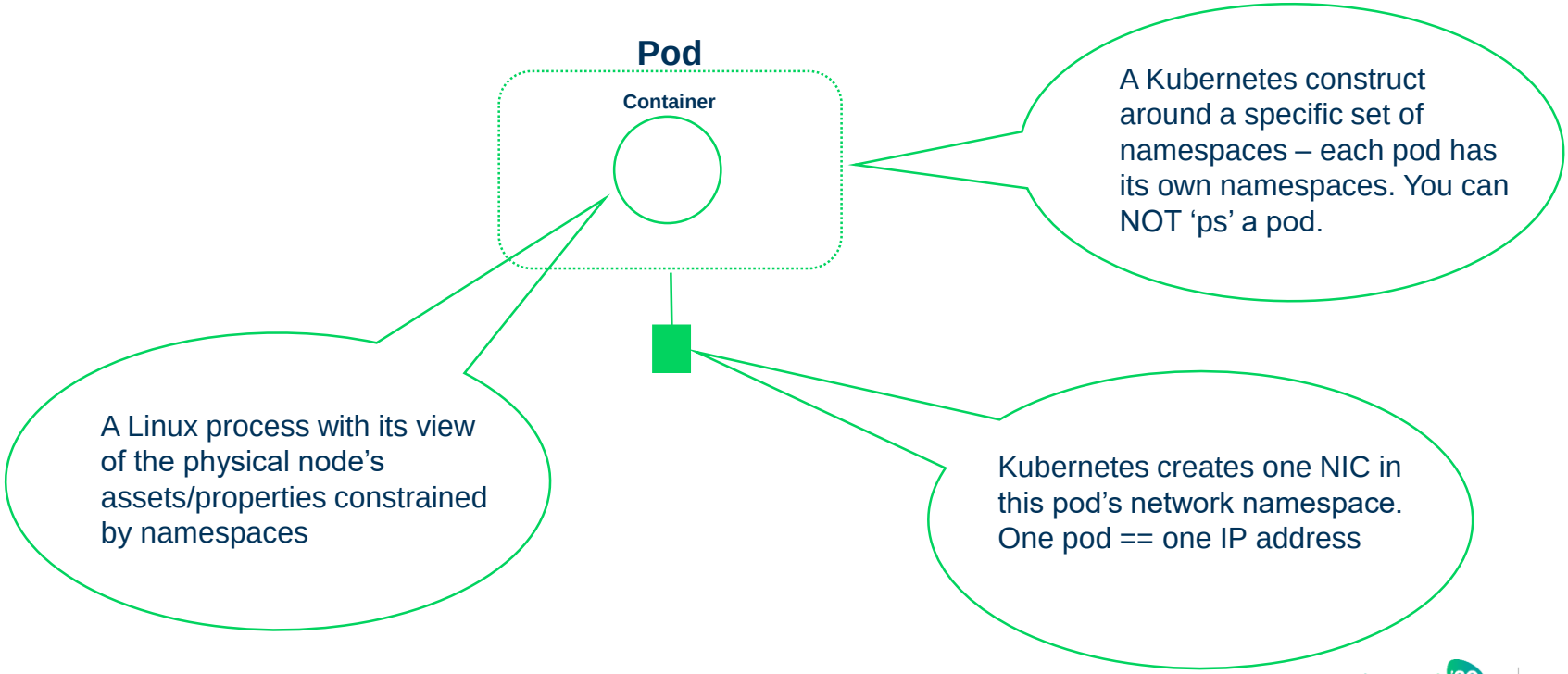
1. Defining pods and containers
2. Drilling into namespaces and cgroups
3. Basic Kubernetes networking
4. A primer in Cilium policy enforcement
5. Service mesh

Defining Pods and Containers

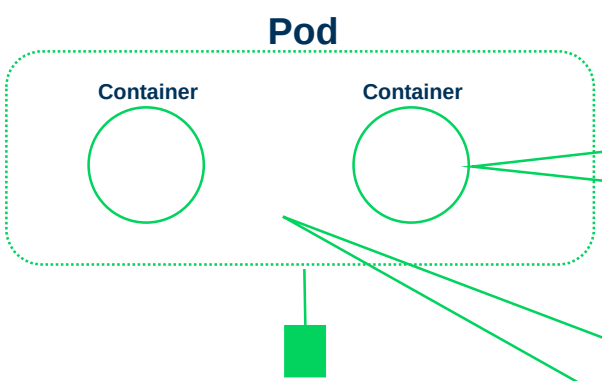
Kubernetes Networking - Namespaces



Kubernetes Networking – Pods And Containers



Kubernetes Networking – Multi-Container Pods



Both containers share localhost (127.0.0.1) port space and can talk to each other via localhost.
They also share port space of pod IP.

If one of these containers was, say, a web server, the other one might be loading content into the local pod filesystems for the web server to serve.
The populator would be a **sidecar**.

Drilling Into Namespaces And cgroups

Linux Namespaces – Isolating A Process's View

Namespace*	Isolates
Cgroup	Cgroup root directory
IPC	System V IPC, Posix message queues
Network	Network devices, stacks, ports, iptables, routes
Mount	Mount points
PID	Process IDs
Time	Boot and monotonic clocks
User	User and group ids
UTS	Hostname and NIS domain name

* - from namespaces(7) man page

Kubernetes And Namespaces

- Unix processes by default will run in default (root) namespace
- A Kubernetes pod is a set of Linux namespaces
- Think of this a pod as a circle of eight filters which will determine what processes (containers) in this pod will be able to see or access
- Kubernetes creates namespaces for each pod to isolate them from each other
- Kubernetes host networking means that a process(container) is running in the root network namespace
- The root network namespace can see all network devices in the physical host/node*

cgroups – Manage and Report Resource Usage

cgroup Controller*	Manages
cpu	Guarantees a minimum/maximum of cpu shares
cpuacct	Provides cpu usage accounting
cpuset	Bind processes to a set of CPU and NUMA nodes
memory	Supports reporting and limiting of process memory
freezer	Suspend and resume all processes in a cgroup
blkio	Control/limit access to devices (throttling)
net_prio	Specify priorities per network interface
pids	Limit number of processes in a cgroup

* - from cgroups(7) man page, partial listing of cgroup capabilities

Kubernetes And cgroups

- cgroups can be applied to a set of processes (containers) in a namespace
- CPU, memory, block I/O utilization, network I/O utilization can be throttled using cgroups
- In more typical terminology, cgroups allow you to implement quotas or Quality of Service (QOS)

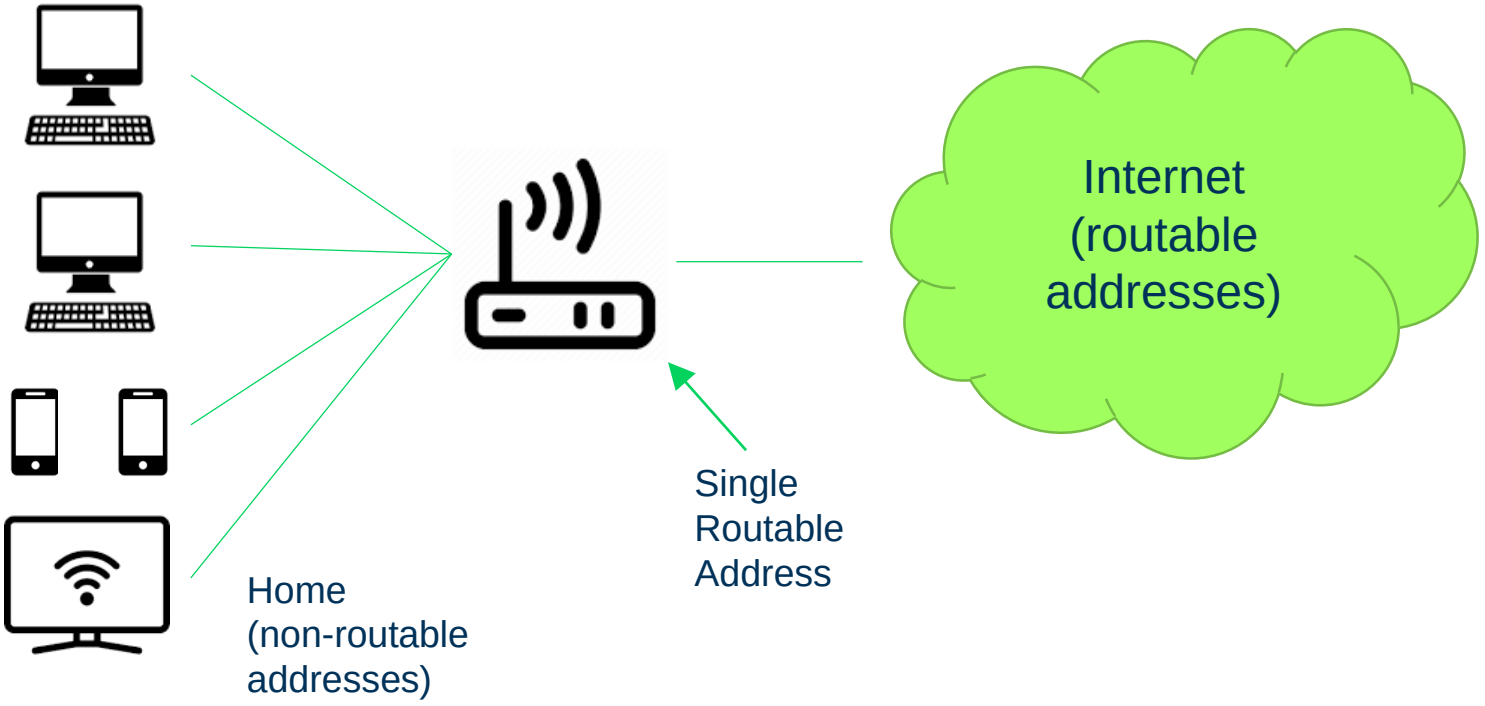
Basic Kubernetes Networking

Kubernetes Networking Rules*

- Pods on a node can communicate with all pods on all nodes without NAT
- Agents on a node (system daemons, kubelet) can communicate with all pods on that node
- Pods in the host network of a node can communicate with all pods on all nodes without NAT

* - from https://kubernetes.io/docs/concepts/cluster-administration/networking/

A Quick Primer/Refresher/RabbitHole On NAT



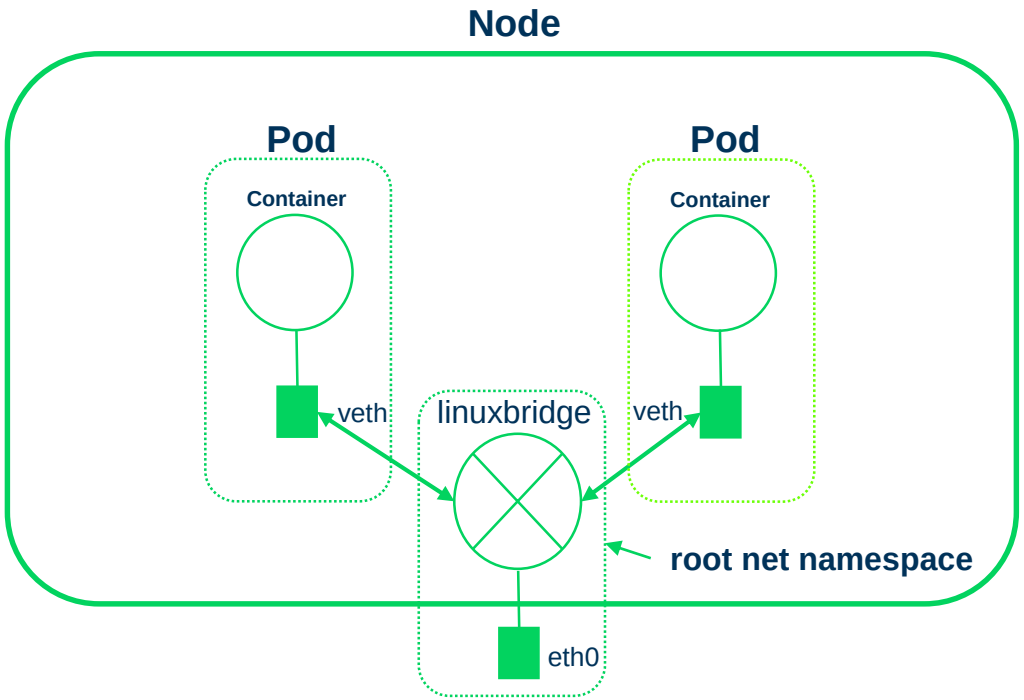
Kubernetes And NAT

- Many (most?) Kubernetes clusters are set up with non-routable addresses
- But clients/users from the Internet need to talk to services hosted by the cluster
- This drives the need for an ingress engine and/or load balancer to translate from Internet routable addresses to cluster non-routable addresses

Kubernetes Networking Models

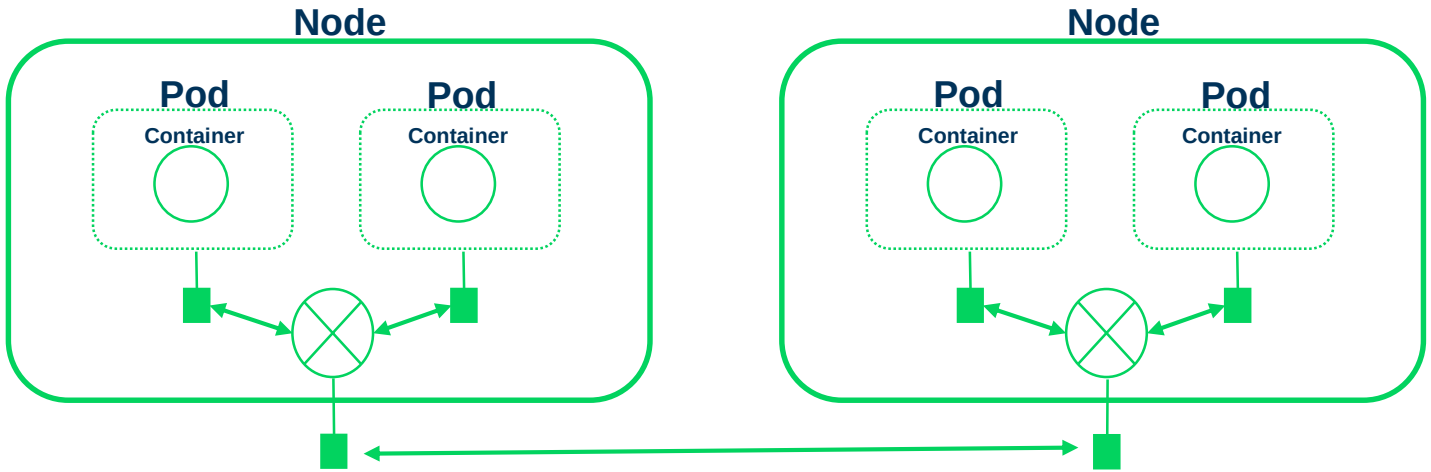
- hostNetwork: true ➔ All pods will run in root network namespace. Efficient, but a little…ahem…insecure
- kubenet ➔ Simple model, using a bit of CNI under-the-covers, more secure than host networking
- CNI ➔ Extremely flexible and powerful, a good number of different CNI options available

kubenet Networking – Intra-node Pod-to-Pod



- A 10km view of kubenet CNI
- A linuxbridge in root namespace joins eth0 with all pods
- A veth (pipe) is injected in each pod as its primary nic
- The other end of each veth is joined to the linuxbridge

kubenet Networking – Inter-node Pod Communication

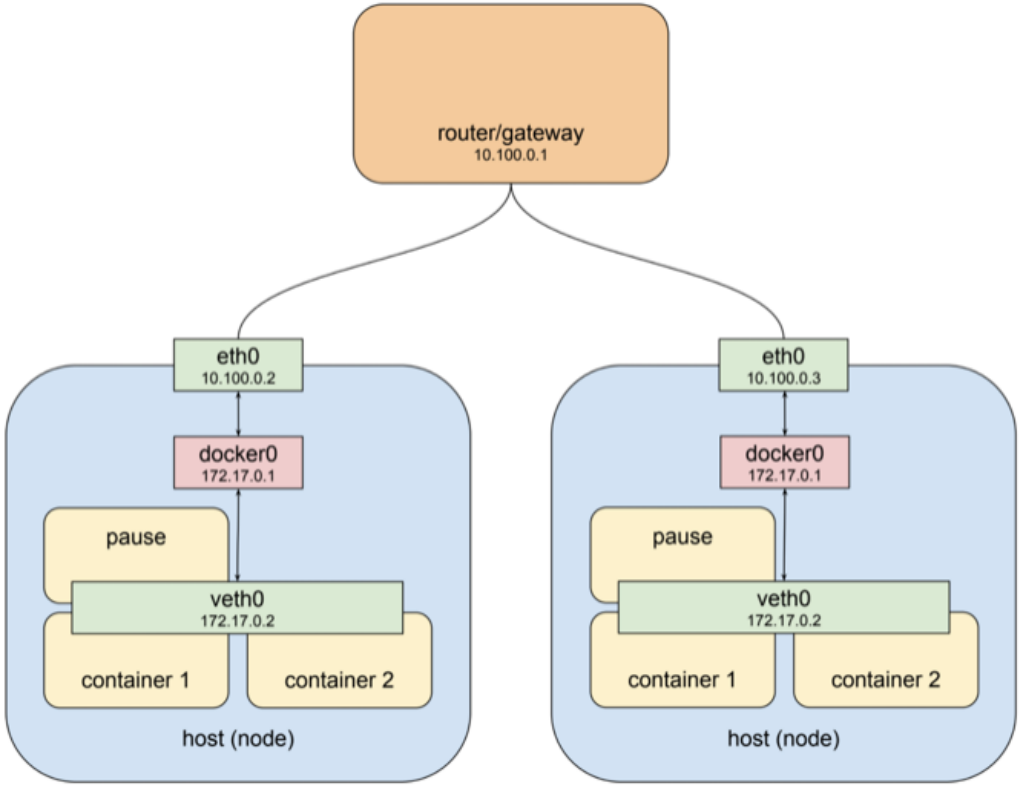


From veth/linuxbridge to node's physical NIC to the other node and reverse your way back up the chain

EMPHASIS

Primary policy of initial K8s networking was hiding the complexity and enabling everything to see everything else with no user effort – *easier than Docker*

Docker Runtime Networking Example



Cilium Policy Enforcement

Cilium – A Quick Sandwich Primer

- Kubernetes is our tool and primary view of the world – the top piece of bread
- Berkeley Packet Filter (BPF) is a high-performance/stable Linux/BSD tech
- XDP is a network-specific Linux kernel-layer technology
- BPF and XDP are the bottom bit of bread

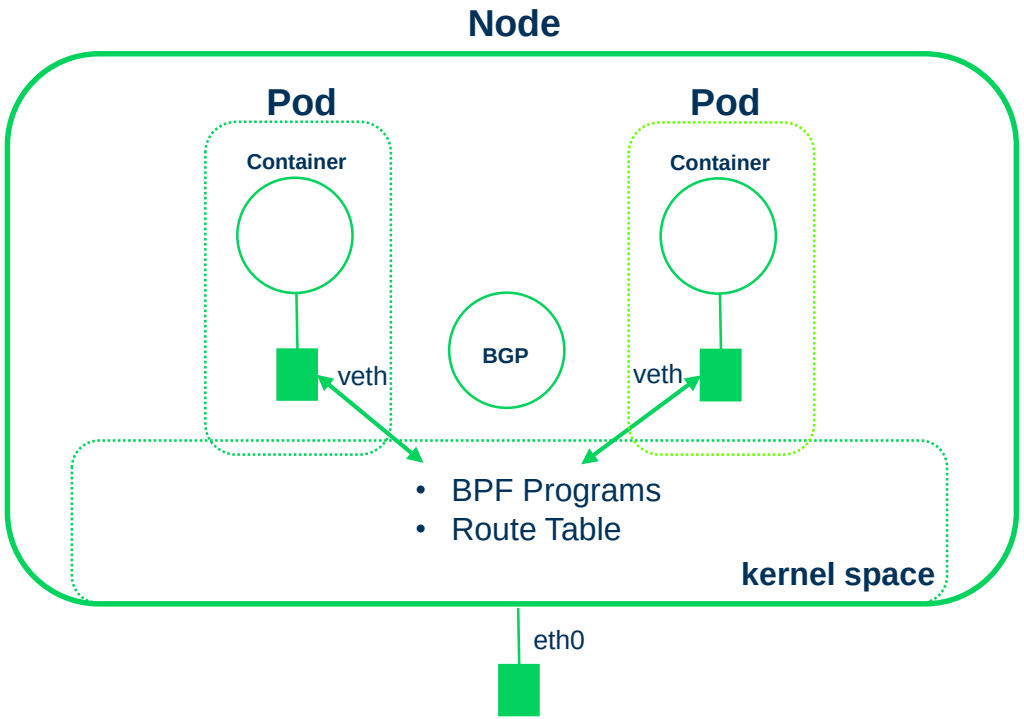
Kubernetes

Cilium

BPF/XDP

- Cilium – translates from Kubernetes networking constructs to BPF/XDP

Cilium – Architecture And Why It's Faster



- A 10km view of Cilium
- veth from pod directly to kernel
- No iptables; replaced with BPF
- No userspace/context-switching overhead
- XDP requires no sk_buff and major iptables copy/processing overhead is eliminated

Kubernetes Policy – Why Do We Need It?

- Remember all pods can see each other? What about security?
- Developers LOVE the Kubernetes networking model, but…
- Security officers, IT managers, CIOs would like to see something…more
- And thus, we require policy enforcement – limit who can see what
- But we have to do it in an agile, cloud-native, dev-ops kind of way

Cilium Label-Aware Policy Declaration

Create This ONE Time and Scale Out

```
endpointSelector:
    matchLabels:
      role = “backend”
ingress:
    matchLabels:
      role = “frontend”
```

OR...modify and grow this each time a pod is added

```
# Allow Apache to access Vitess/Mysql in pods [1,2]-2
iptables -i eth0 -p tcp -dport 3306 -s ${IP1-1} -d ${IP1-2}
iptables -i eth0 -p tcp -dport 3306 -s ${IP2-1} -d ${IP1-2}
iptables -i eth0 -p tcp -dport 3306 -s ${IP1-1} -d ${IP2-2}
iptables -i eth0 -p tcp -dport 3306 -s ${IP2-1} -d ${IP2-2}
…
```

A Quick Shout-out To SDN

- Software Defined Networking (SDN)
- Multiple definitions from network hardware vendors, commercial and open source vendors, and analysts
- My favorite definition - “SDN provides the ability to centrally manage and deploy network policy to disparate network devices”
- Cilium provides the ability to centrally manage and deploy network policy in your Kubernetes cluster

Service Mesh

Application Architecture Changes In Kubernetes

Telco Appliance

- Often x64-based
- Often Linux-based
- *ASICs and FPGAs*
- Stateful, monolithic, long-running

Easier

Mode 1 Application

- Migrated code base
- Stateful
- Monolithic
- Long-running
- State of the Union

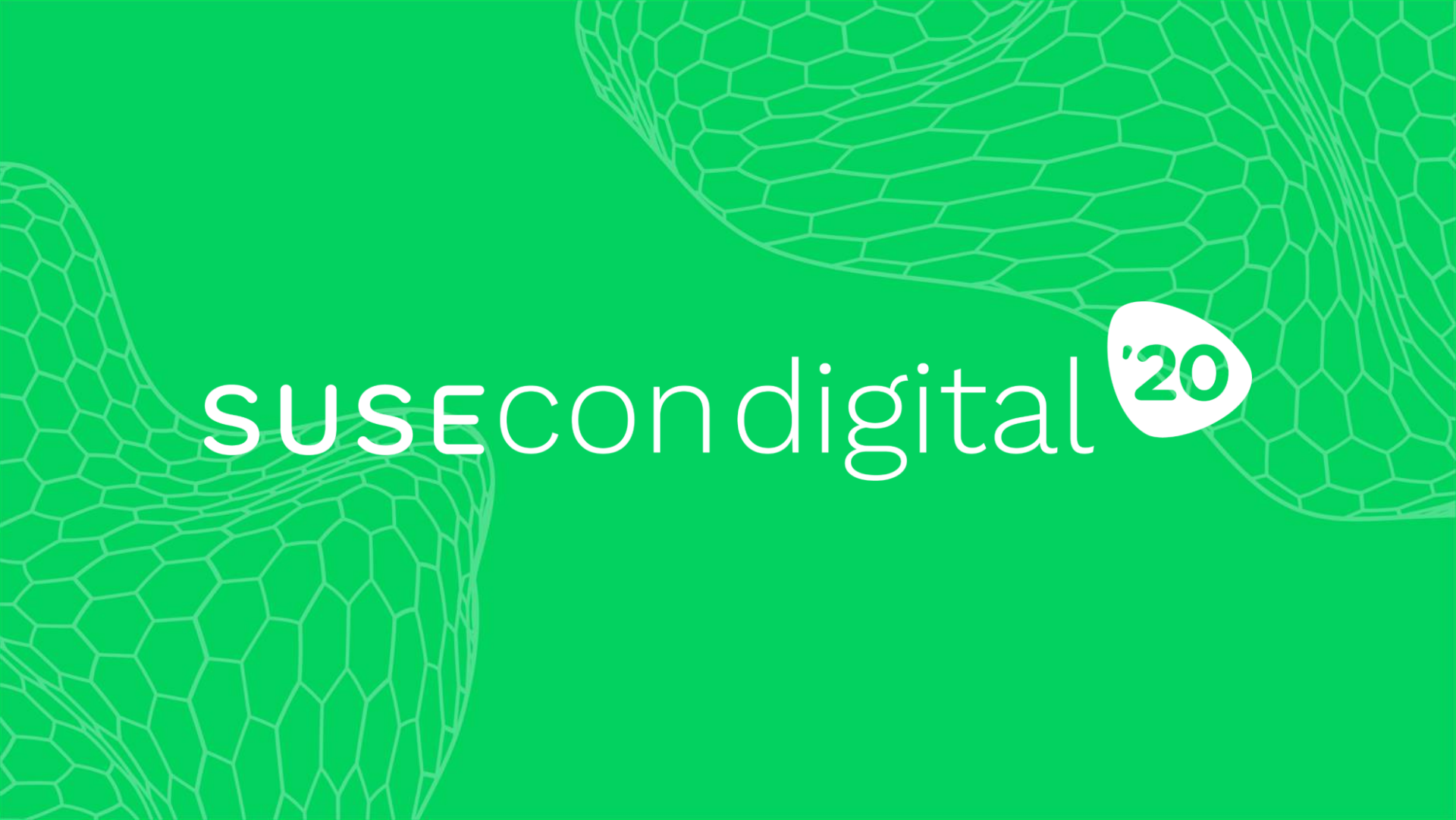
Harder

Mode 2 Application

- Redesigned code base
- “Stateless”
- Scale-out
- Possibly long-running but generally not
- Coming Soon to a Theater Near You

The Result Of Lots Of M2's

- Many endpoints to talk to a single service
- Need to "hide" all of those individual endpoints behind a single address
- Non-routable addresses used internally to Kubernetes along with specific TCP ports – these are ugly/difficult to type into a browser
- Ingress (part of service mesh) provides an easier-to-reach service
- This is just ONE benefit provided by service mesh, but we're out of time for this talk. More information in a follow-up talk and service mesh coming to SUSE CaaS Platform soon!


SUSEcon digital '20

General Disclaimer

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. SUSE makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. The development, release, and timing of features or functionality described for SUSE products remains at the sole discretion of SUSE. Further, SUSE reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All SUSE marks referenced in this presentation are trademarks or registered trademarks of SUSE, LLC, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.