Whose Packet Is It Anyways?

Kevin Leimkuhler & Douglas Jordan
KubeCon NA 2022

Douglas Jordan

Airbnb: Istio at scale

Microsoft: Bare metal in Azure

@dwj300

Kevin Leimkuhler

Software Engineer @Buoyant

Linkerd Maintainer

@kleimkuhler

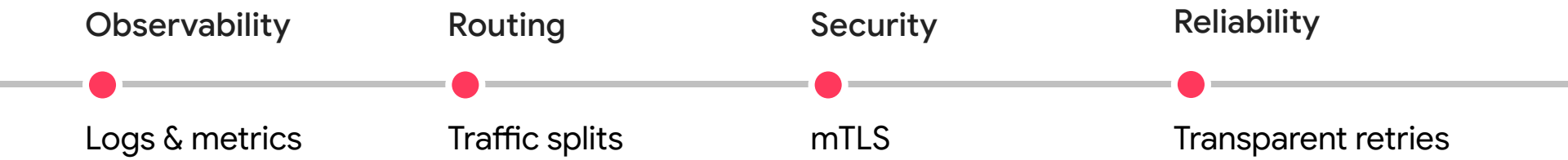
Agenda

1. How is a packet routed in a service mesh?
2. TCP Debugging
3. tcpdump + wireshark

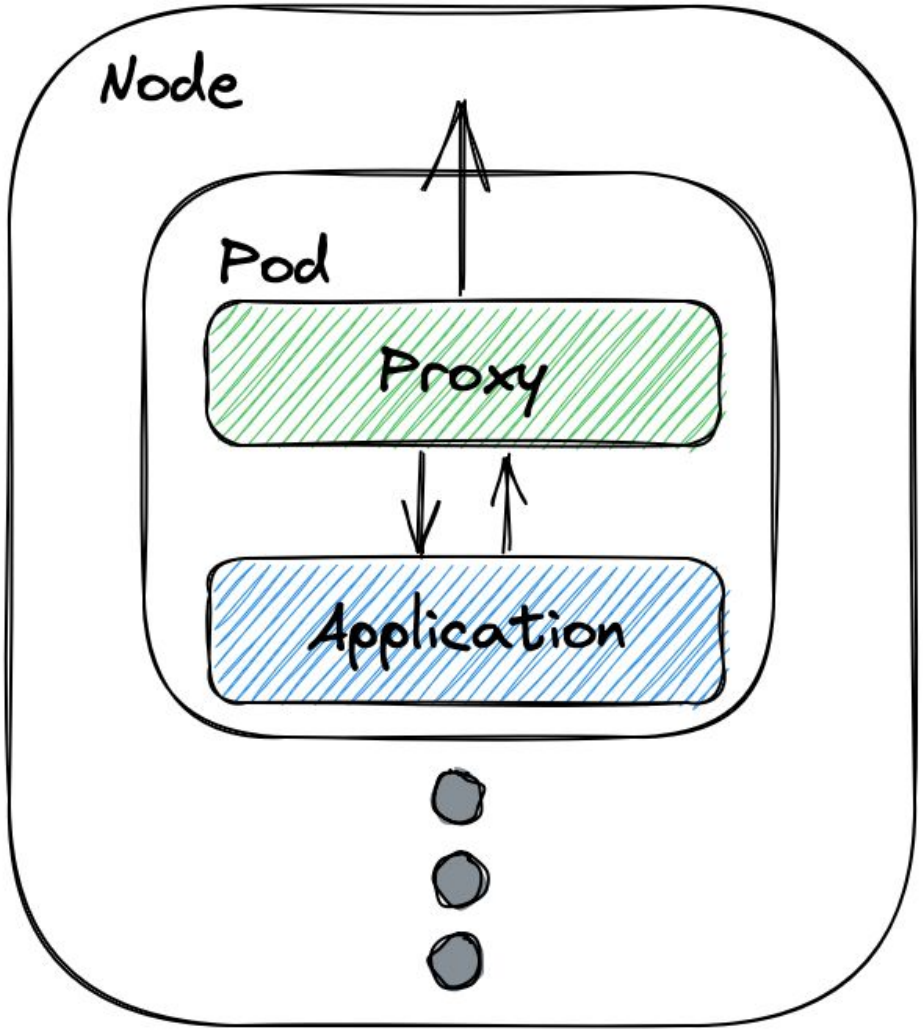
How is a packet routed in a service mesh?

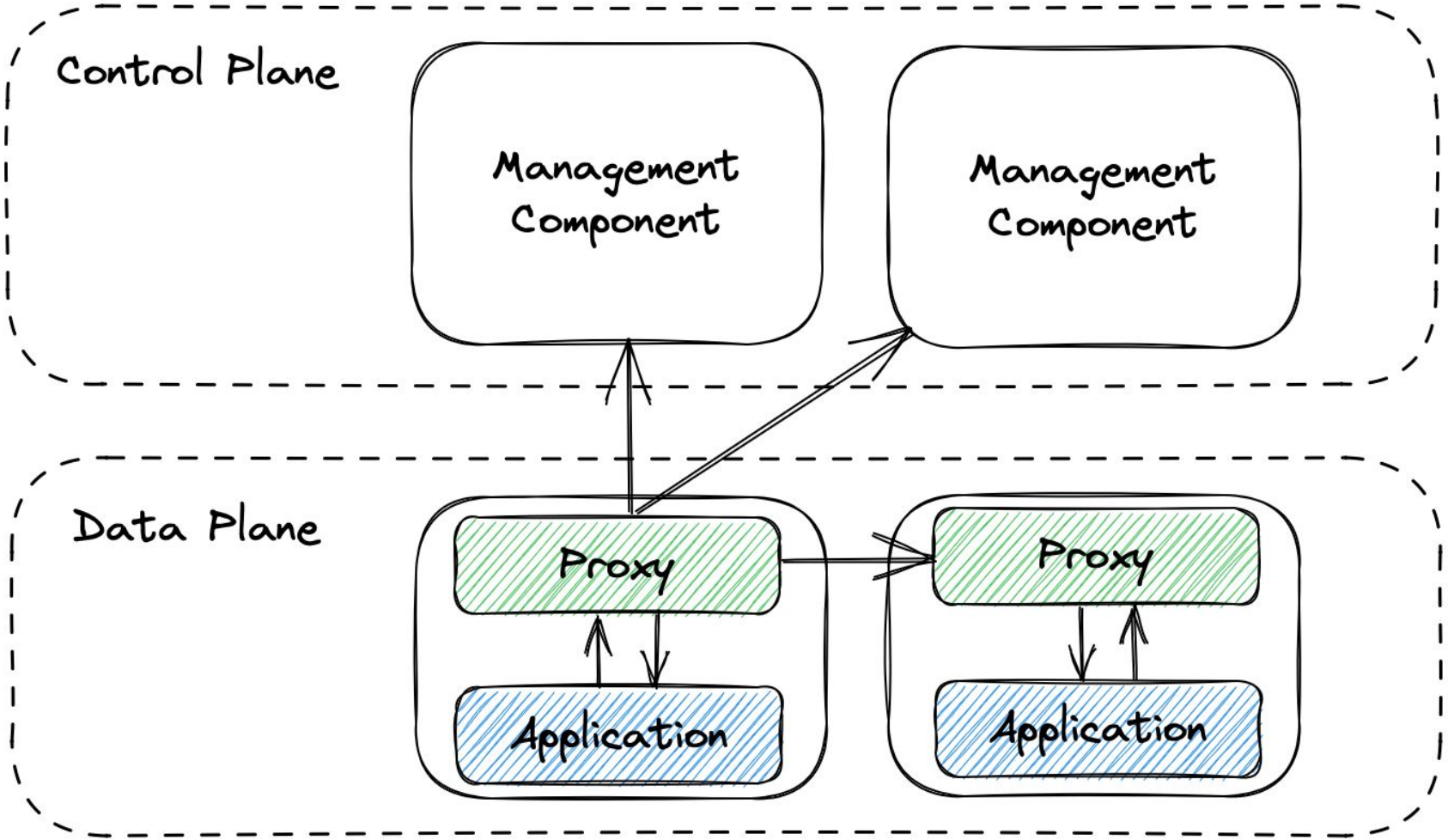
What is a service mesh?

Key Properties of a Service Mesh



What is the sidecar proxy model

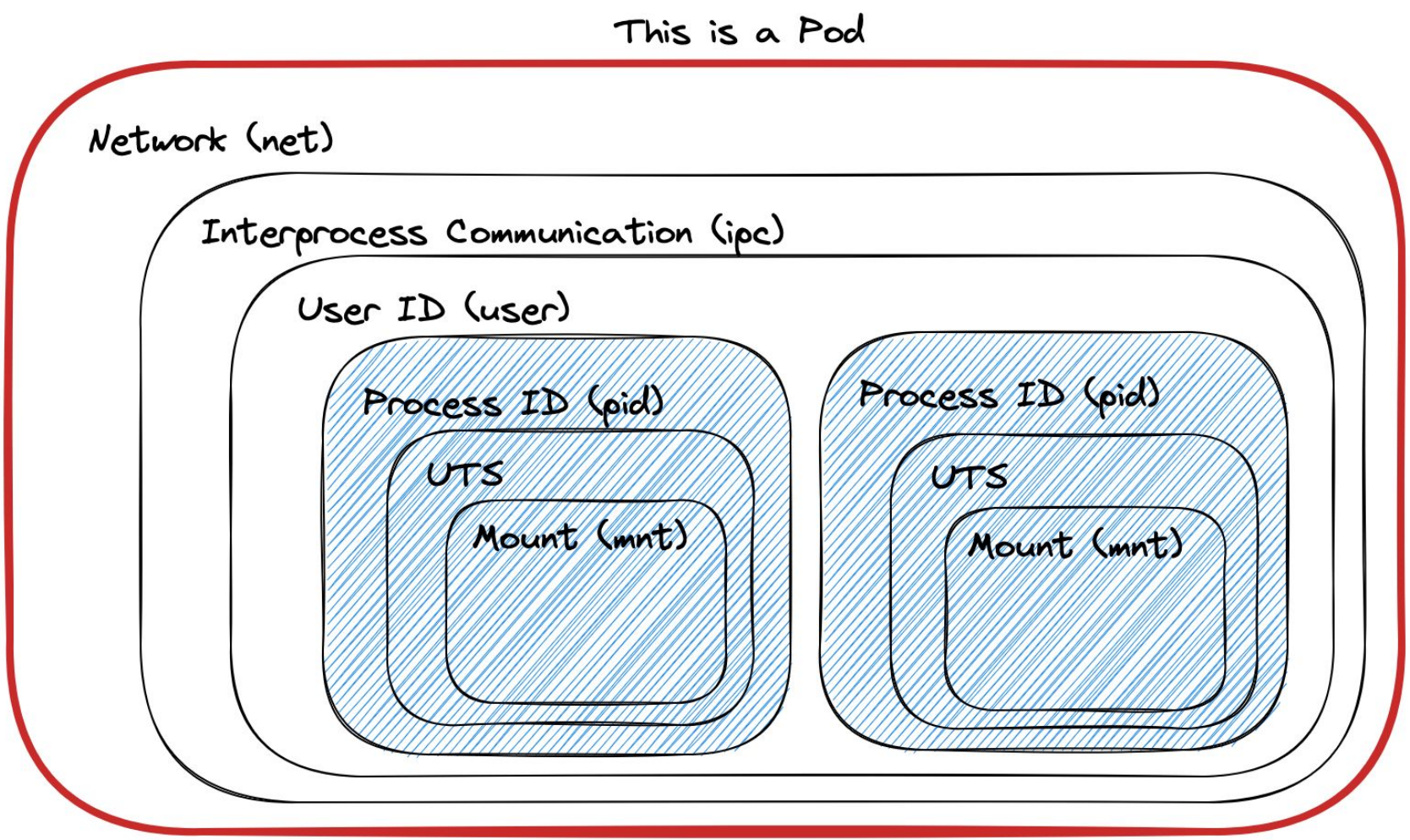


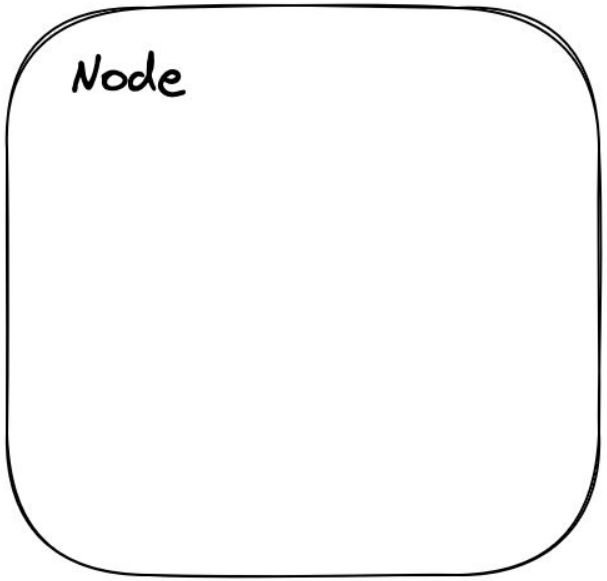

Control Plane
Management Component
Management Component
Data Plane
Proxy
Application
Proxy
Application

What is a container?

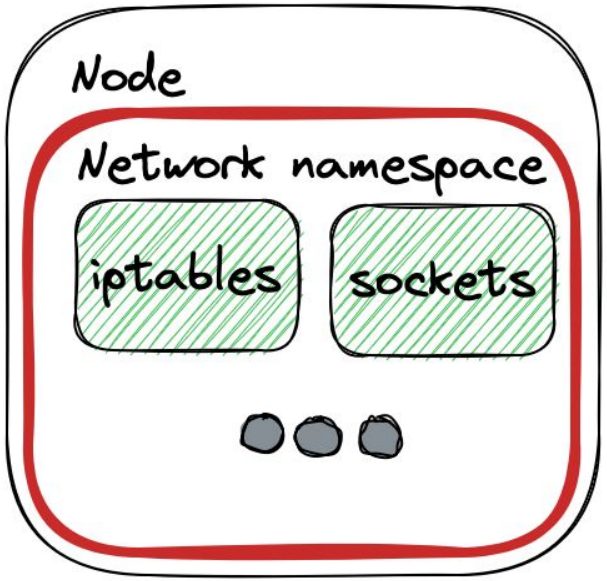
Linux doesn’t have containers

It has namespaces


This is a Pod
Network (net)
Interprocess Communication (ipc)
User ID (user)
Process ID (pid)
UTS
Mount (mnt)
Process ID (pid)
UTS
Mount (mnt)

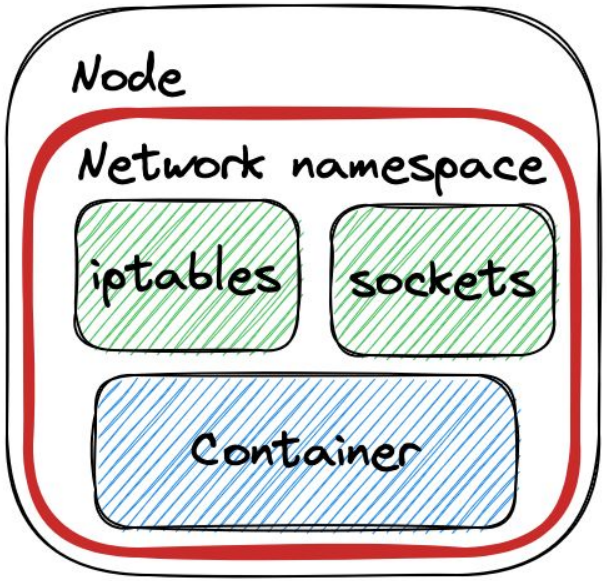


We start with a host, often referred to as a Node

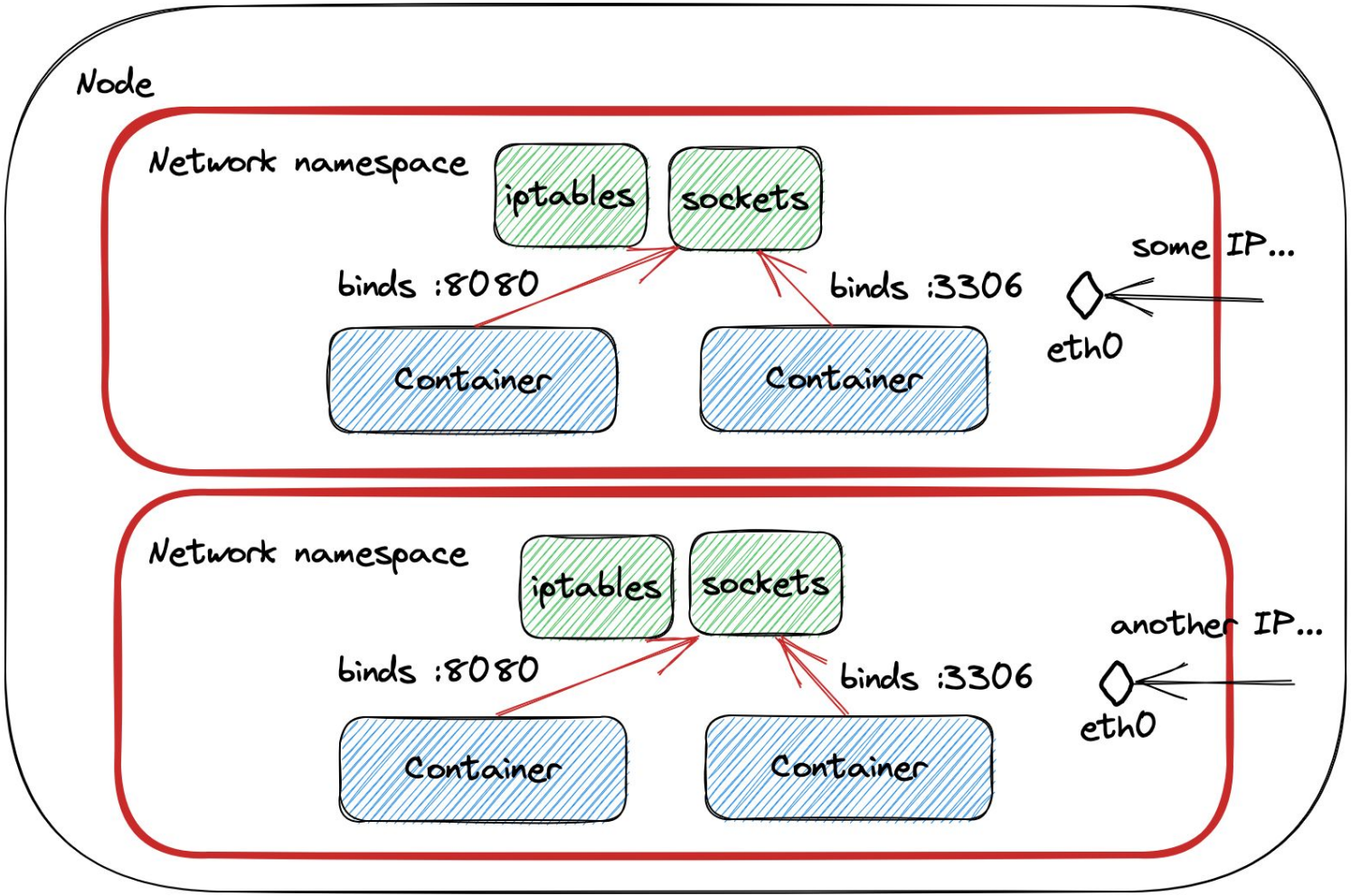


We create a network

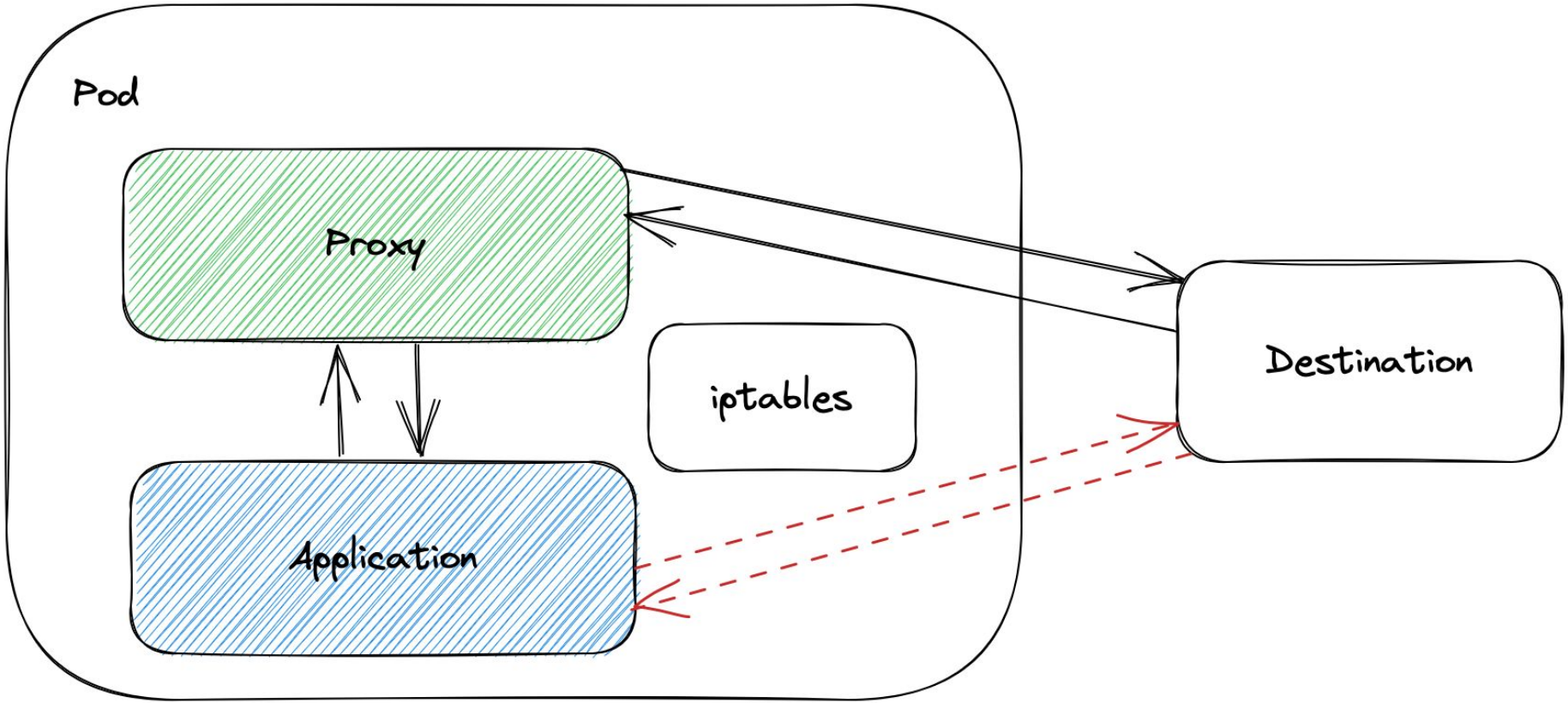
namespace on the host

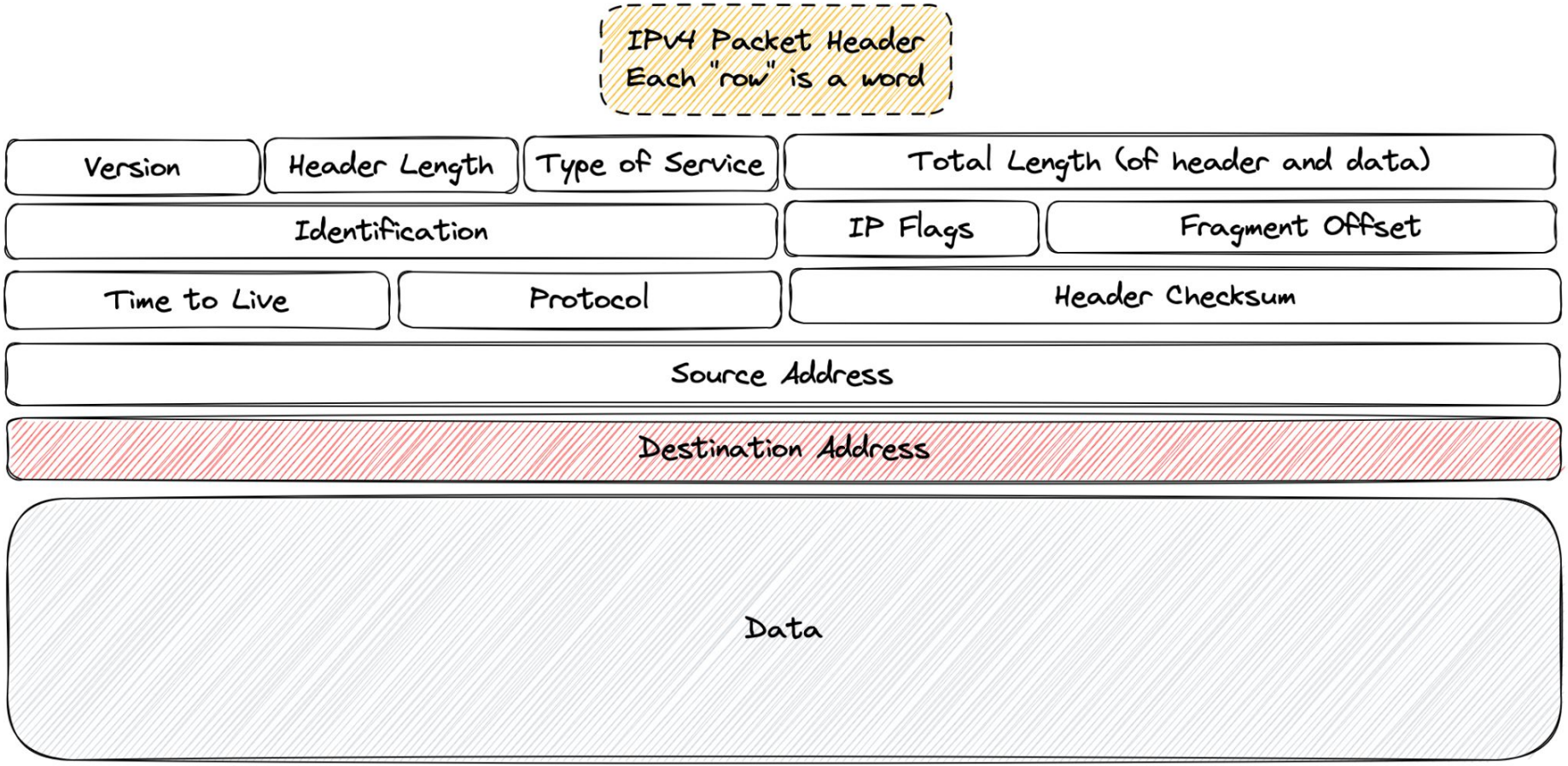


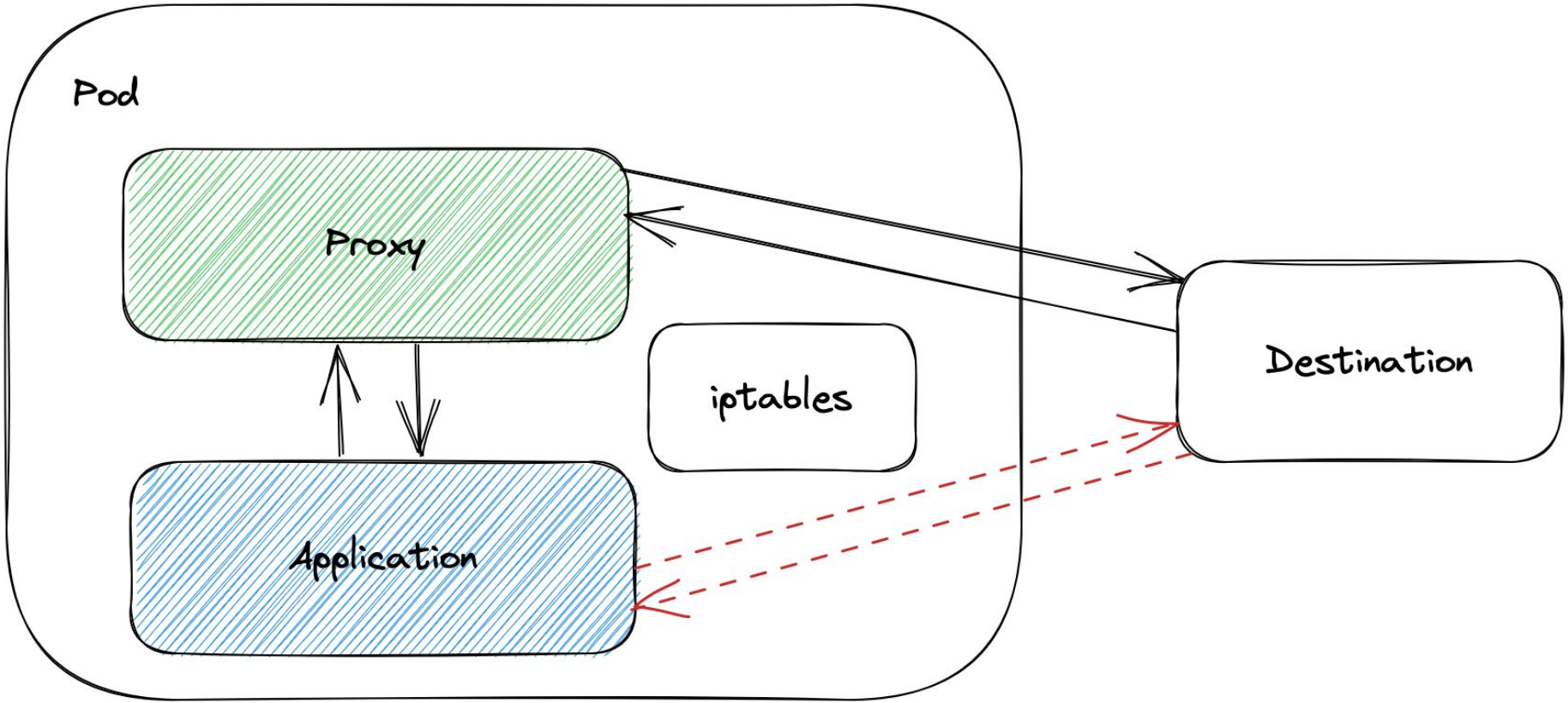
Each “container” is a process that shares this view of network resources

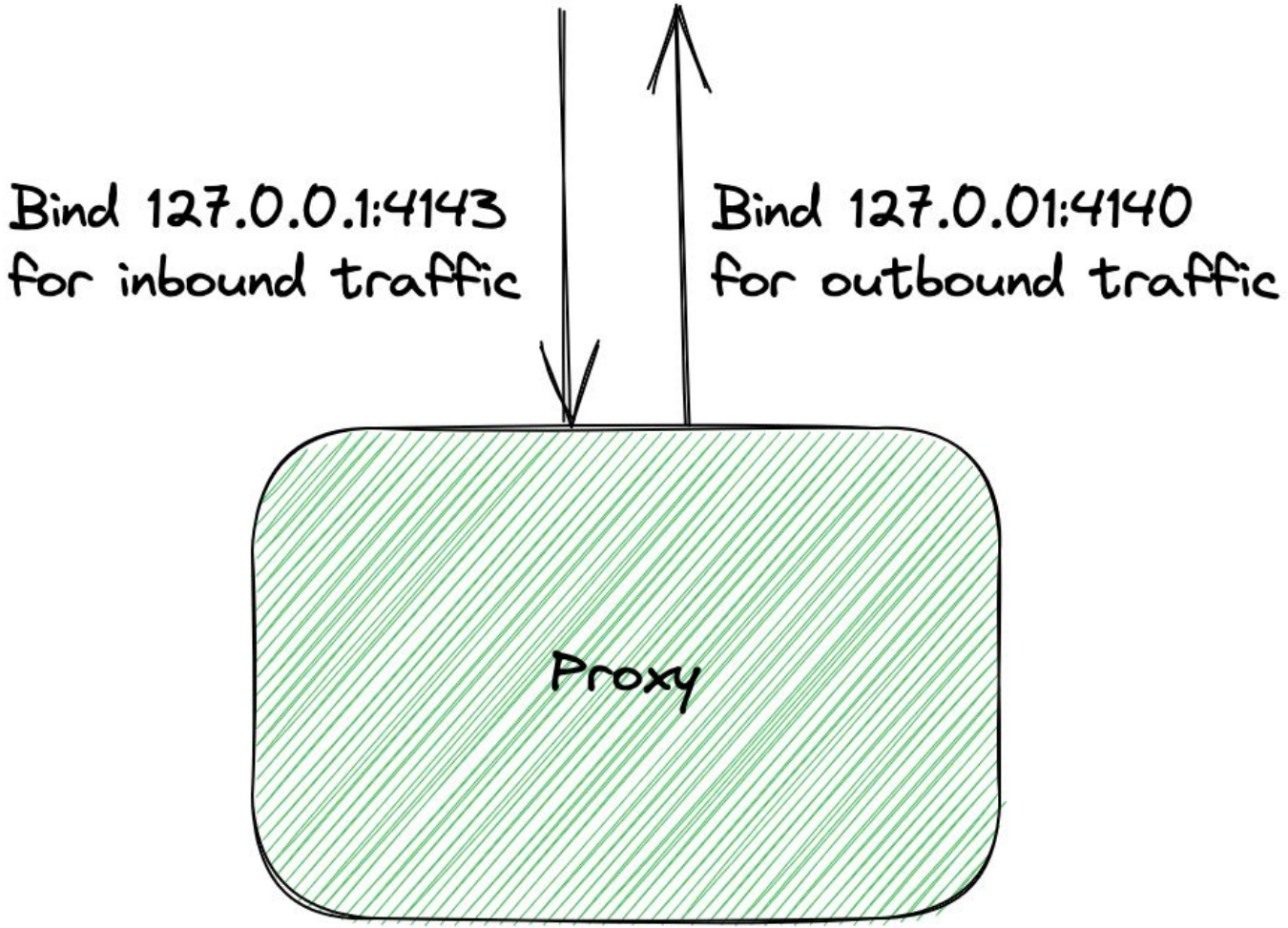

Node
Network namespace
iptables
sockets
binds :8080
binds :3306
Container
Container
some IP...
eth0
Network namespace
iptables
sockets
binds :8080
binds :3306
Container
Container
another IP...
eth0

Why do we care about iptables?


Pod
Proxy
iptables
Application
Destination


IPV4 Packet Header
Each "row" is a word
Version
Header Length
Type of Service
Total Length (of header and data)
Identification
IP Flags
Fragment Offset
Time to Live
Protocol
Header Checksum
Source Address
Destination Address
Data

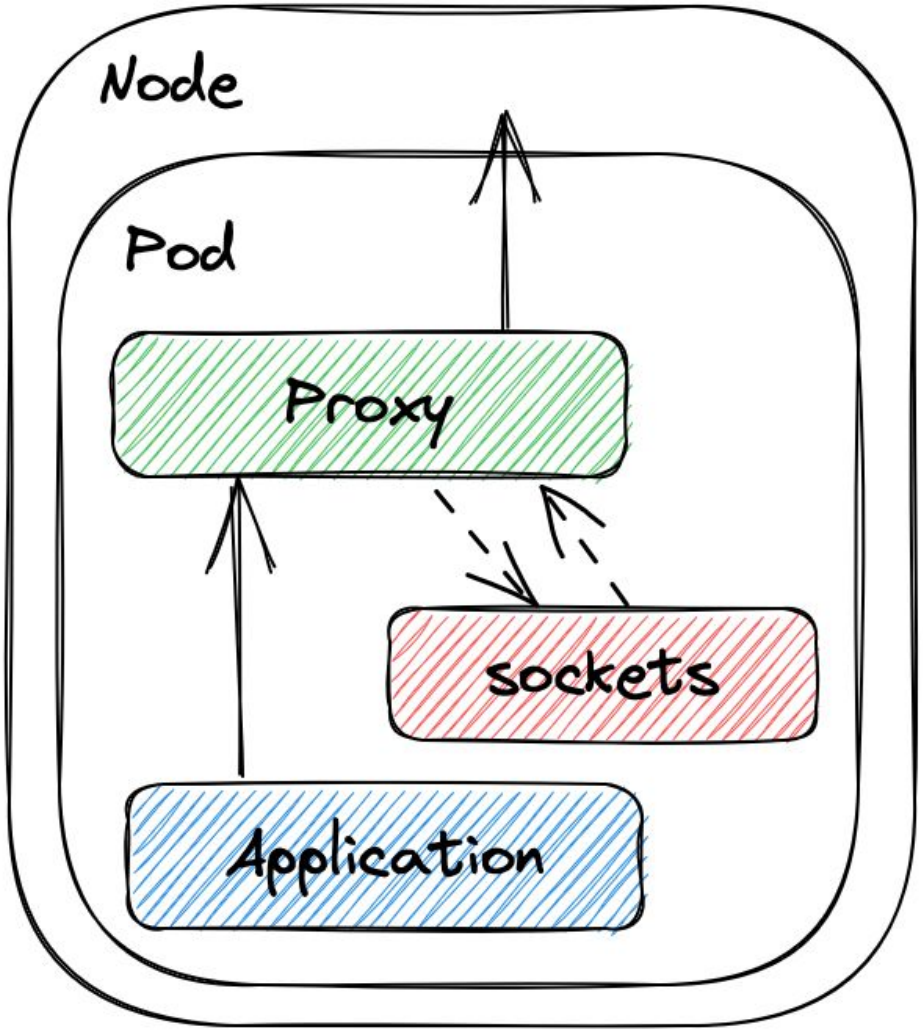

Pod
Proxy
iptables
Application
Destination


Bind 127.0.0.1:4143
for inbound traffic
Bind 127.0.01:4140
for outbound traffic
Proxy

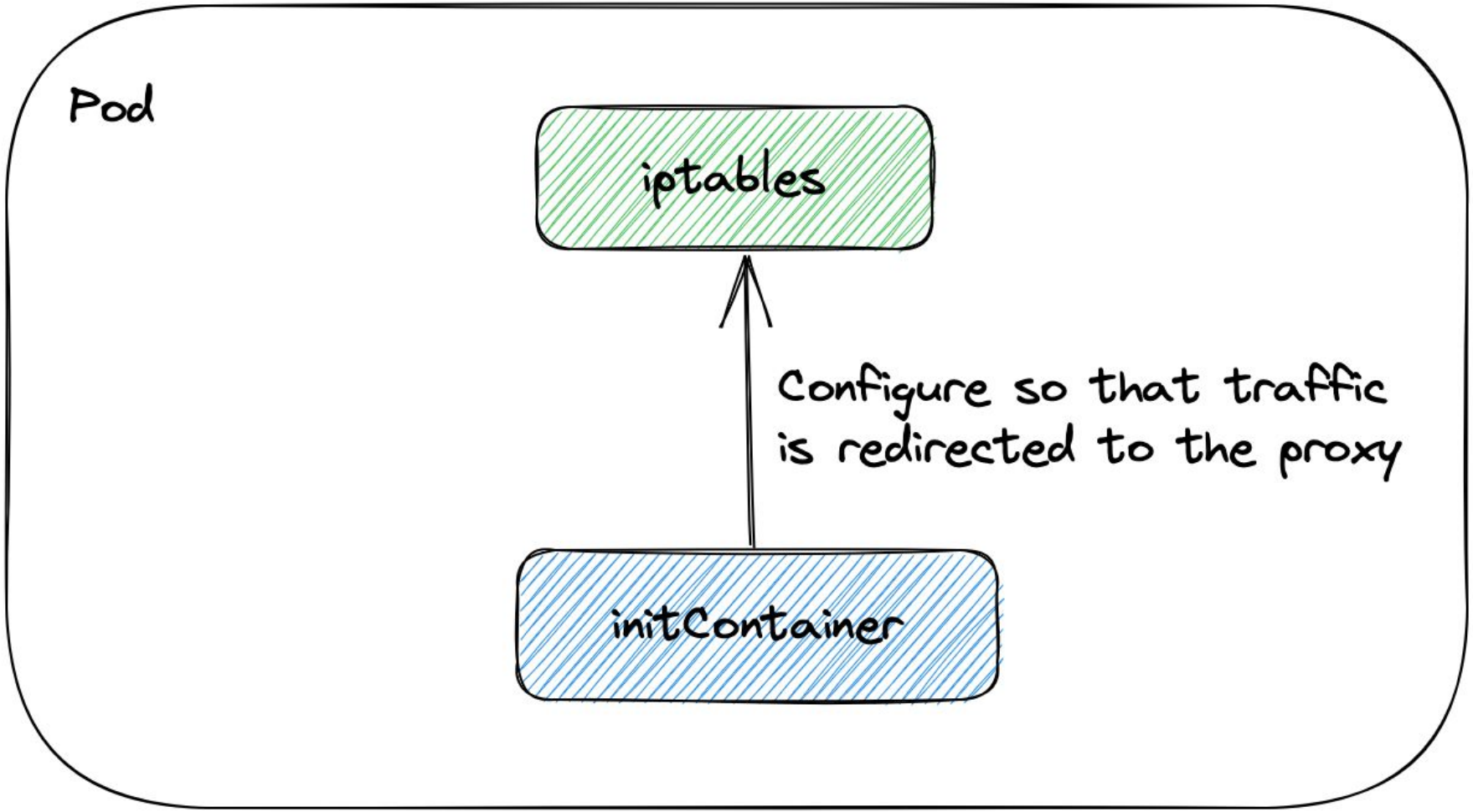
How does a proxy redirect a packet?

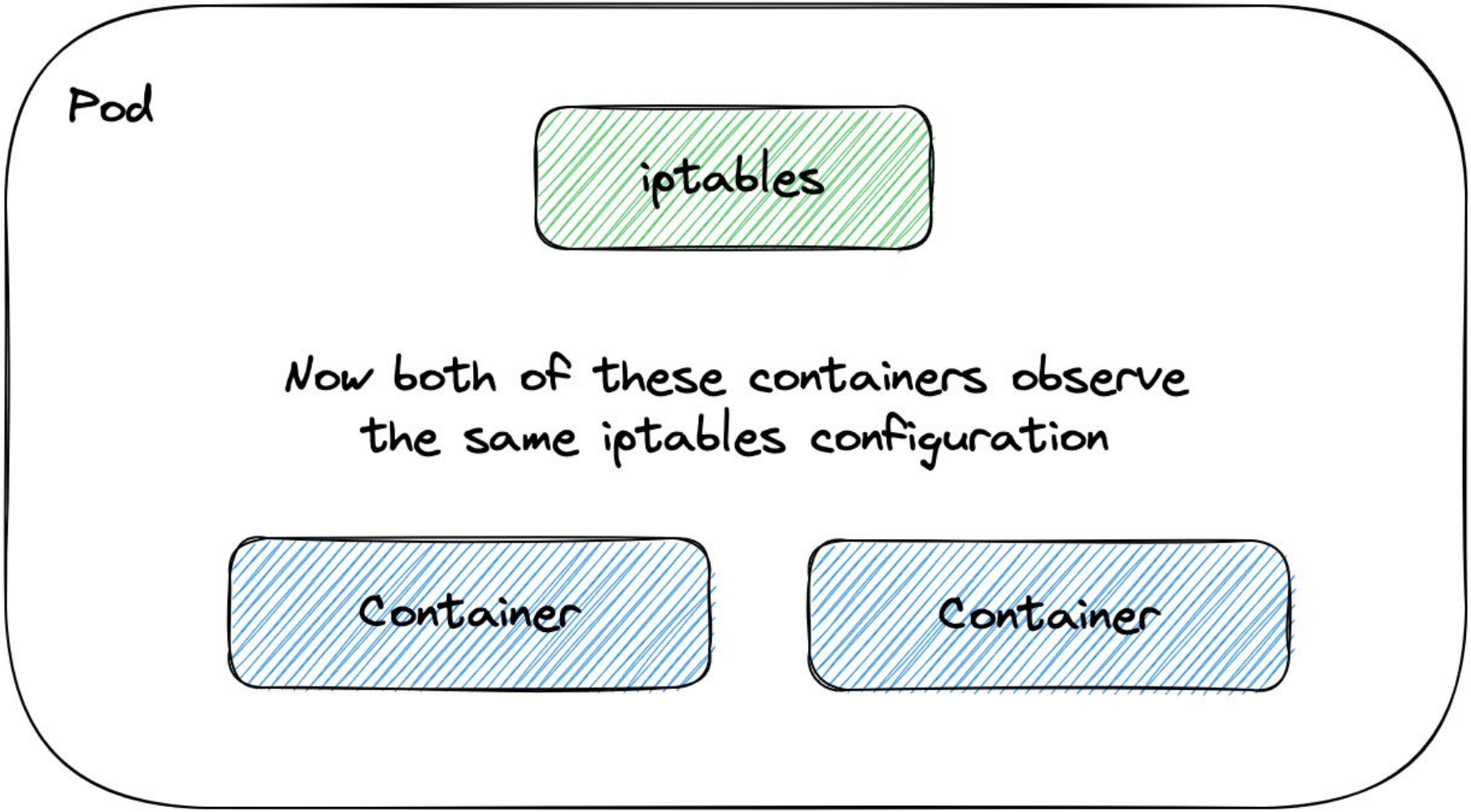
The packet headers were changed by iptables

A proxy checks the TCP stream's socket options



What is responsible for configuring iptables?


Pod
iptables
Configure so that traffic
is redirected to the proxy
initContainer


Pod
iptables
Now both of these containers observe
the same iptables configuration
Container
Container

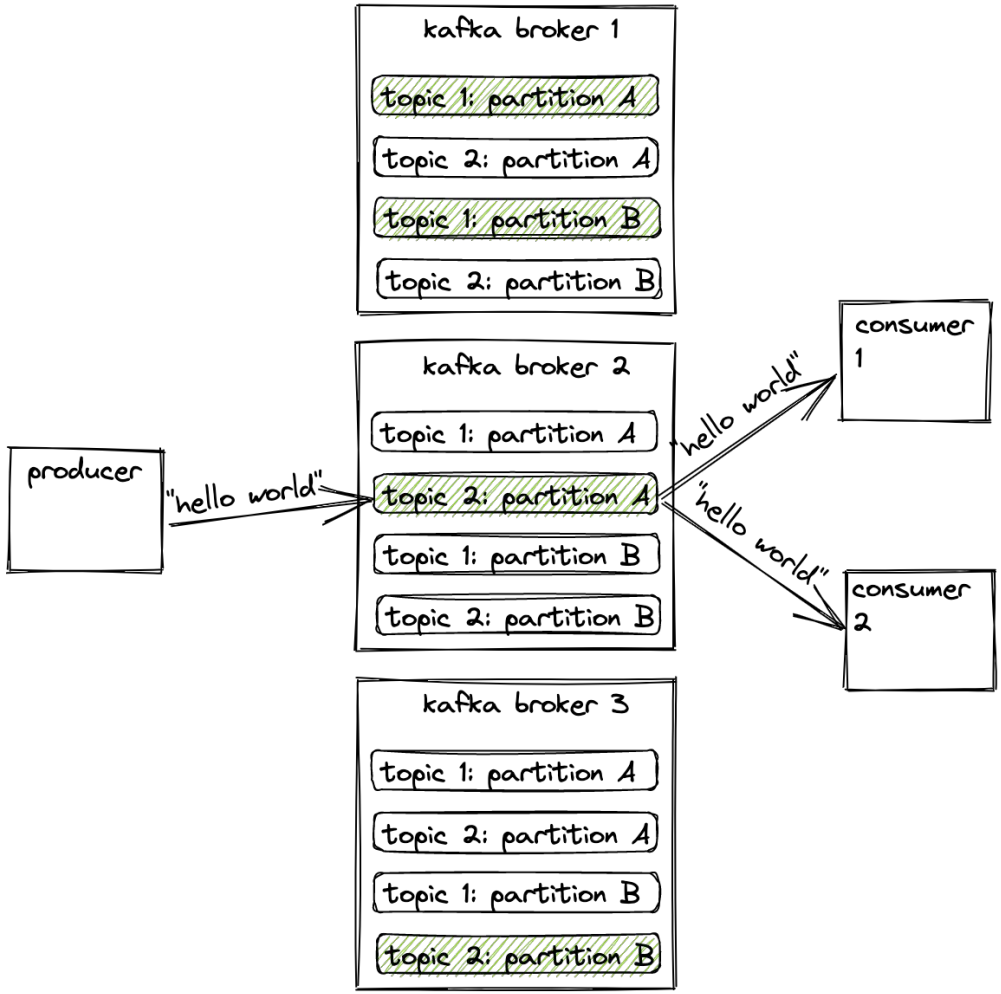
Still don’t want to give an initContainer elevated permissions?

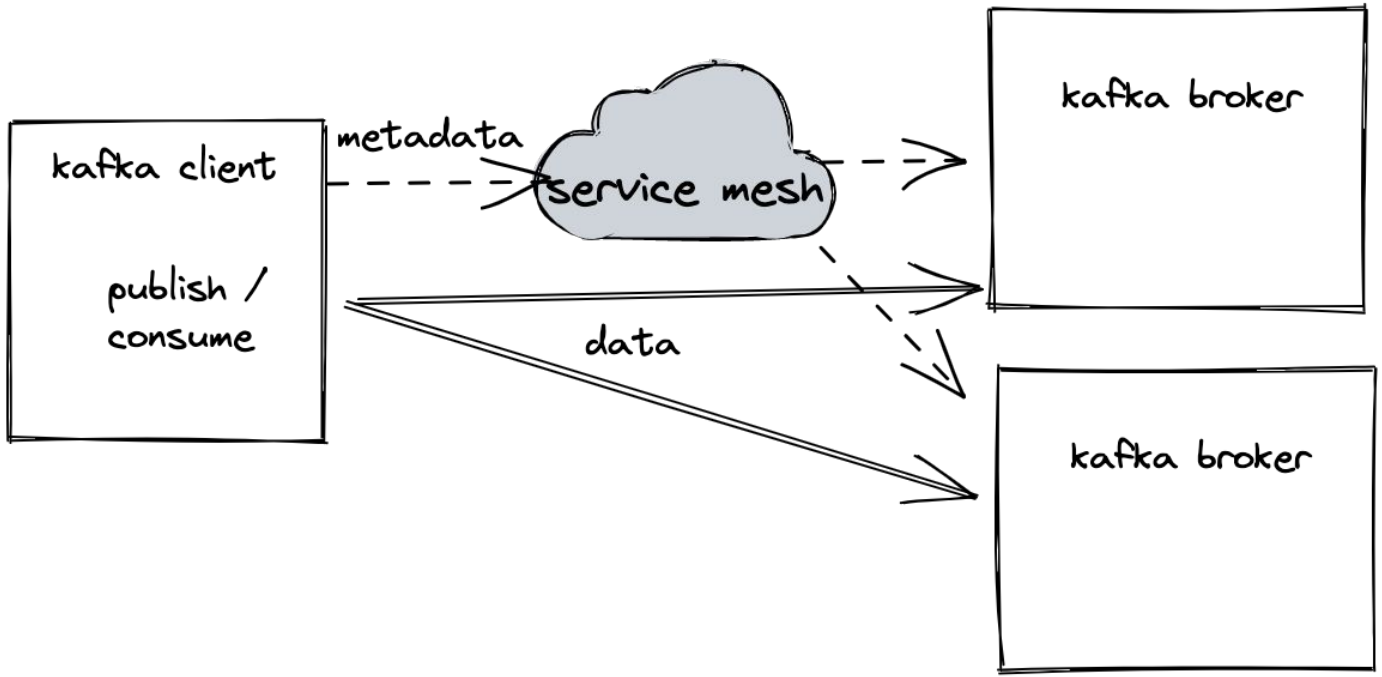
TCP Debugging

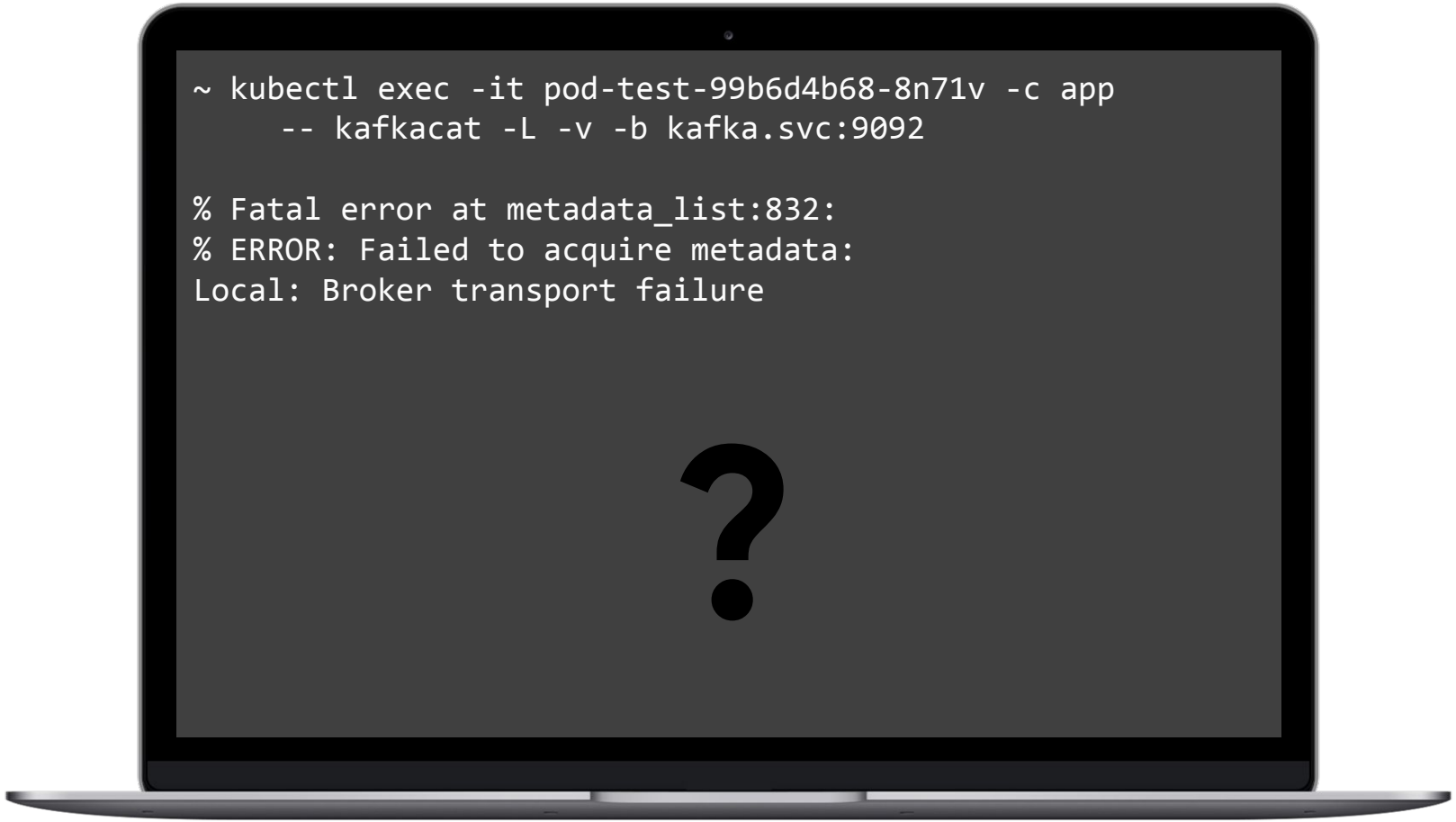
Assumptions

1. CNI: AWS VPC CNI
2. CRI: CRI-O
3. SSH access to nodes

kafka?


kafka broker 1
topic 1: partition A
topic 2: partition A
topic 1: partition B
topic 2: partition B
kafka broker 2
topic 1: partition A
topic 2: partition A
topic 1: partition B
topic 2: partition B
kafka broker 3
topic 1: partition A
topic 2: partition A
topic 1: partition B
topic 2: partition B
producer
"hello world"
"hello world"
"hello world"
consumer 1
consumer 2


kafka client
publish /
consume
metadata
service mesh
data
kafka broker
kafka broker


~ kubectl exec -it pod-test-99b6d4b68-8n71v -c app
-- kafkacat -L -v -b kafka.svc:9092
% Fatal error at metadata_list:832:
% ERROR: Failed to acquire metadata:
Local: Broker transport failure
?

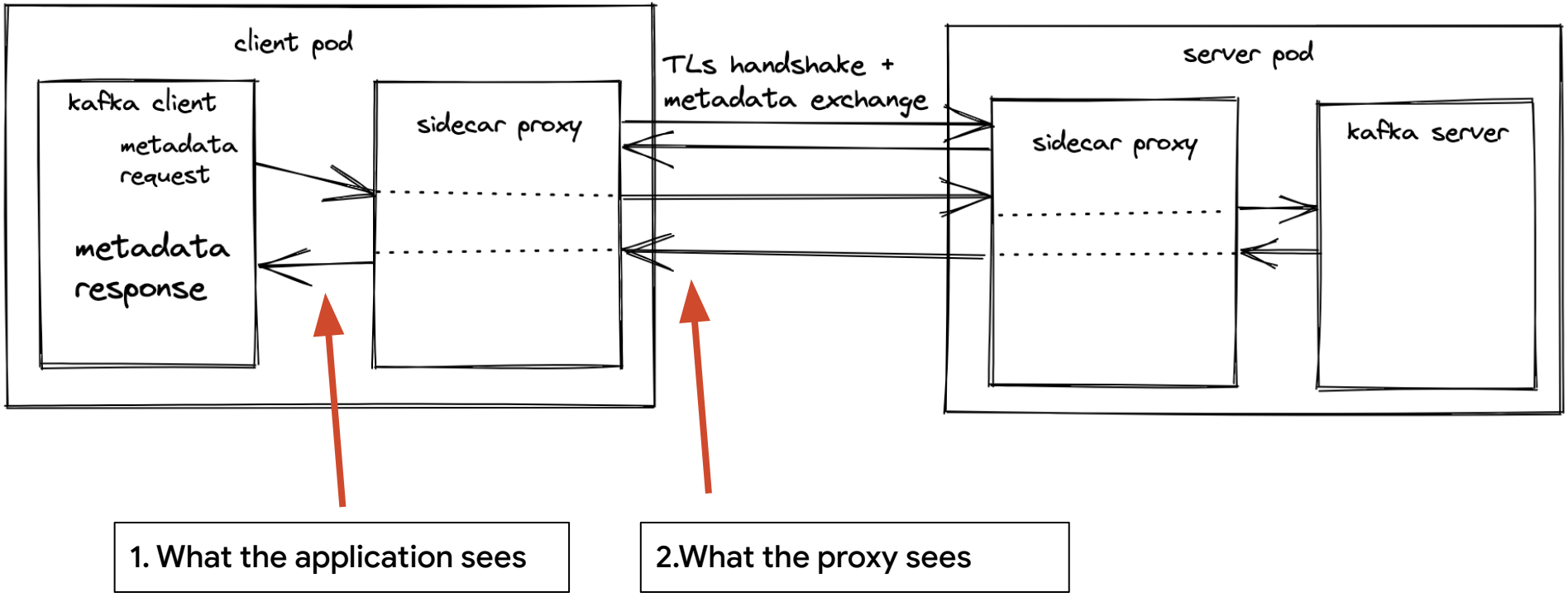
HTTP (25) vs TCP (7) Response Flags

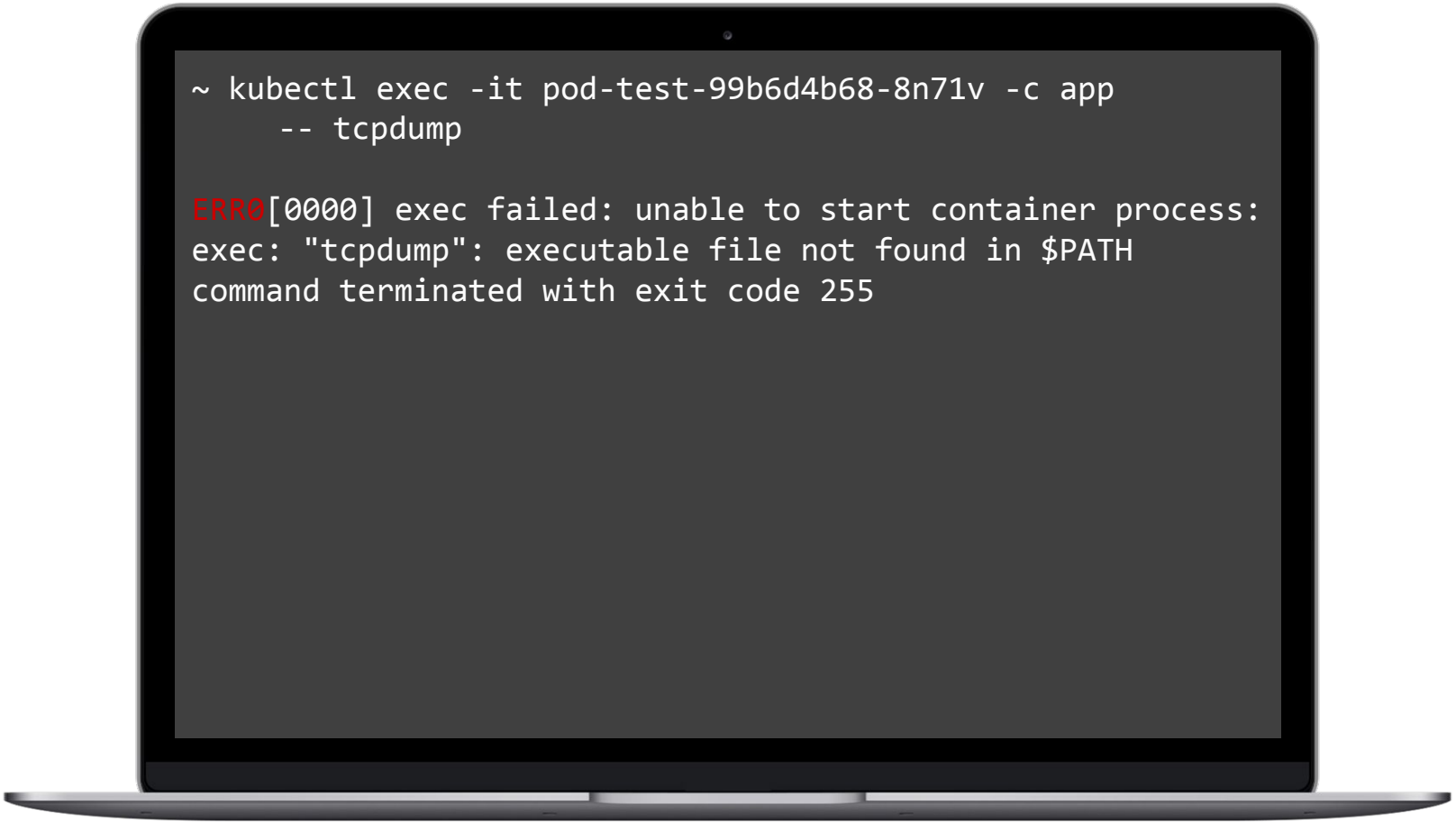
DC: Downstream connection termination.
LH: Local service failed health check request in addition to 503 response code.
UT: Upstream request timeout in addition to 504 response code.
LR: Connection local reset in addition to 503 response code.
UR: Upstream remote reset in addition to 503 response code.
UC: Upstream connection termination in addition to 503 response code.
DI: The request processing was delayed for a period specified via fault injection.
FI: The request was aborted with a response code specified via fault injection.
RL: The request was ratelimited locally by the HTTP rate limit filter in addition to 429 response code.
UAEX: The request was denied by the external authorization service.
RLSE: The request was rejected because there was an error in rate limit service.
IH: The request was rejected because it set an invalid value for a strictly-checked header in addition to 400 response code.
SI: Stream idle timeout in addition to 408 response code.
DPE: The downstream request had an HTTP protocol error.
UPE: The upstream response had an HTTP protocol error.
UMSDR: The upstream request reached max stream duration.
OM: Overload Manager terminated the request.
DF: The request was terminated due to DNS resolution failure.

UH: No healthy upstream hosts in upstream cluster in addition to 503 response code.
UF: Upstream connection failure in addition to 503 response code.
UO: Upstream overflow (circuit breaking) in addition to 503 response code.
NR: No route configured for a given request in addition to 404 response code, or no matching filter chain for a downstream connection.
URX: The request was rejected because the upstream retry limit (HTTP) or maximum connect attempts (TCP) was reached.
NC: Upstream cluster not found.
DT: When a request or connection exceeded max_connection_duration or max_downstream_connection_duration.

https://www.envoyproxy.io/docs/envoy/latest/configuration/observability/access_log/usage

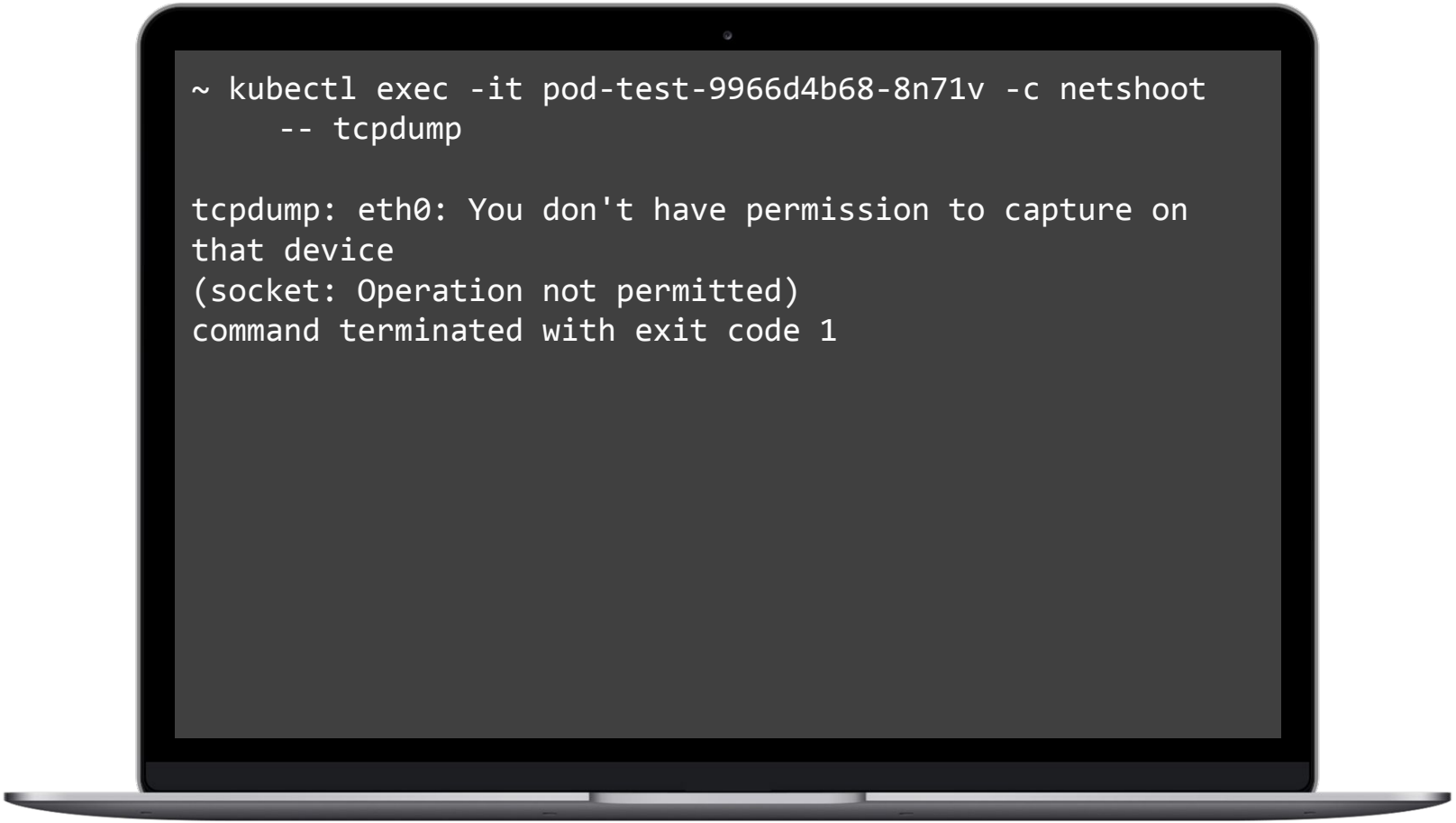
tcpdump + wireshark


client pod
kafka client
metadata request
metadata response
sidecar proxy
TLs handshake + metadata exchange
server pod
sidecar proxy
kafka server
1. What the application sees
2.What the proxy sees


~ kubectl exec -it pod-test-99b6d4b68-8n71v -c app
-- tcpdump
ERRO[0000] exec failed: unable to start container process:
exec: "tcpdump": executable file not found in $PATH
command terminated with exit code 255

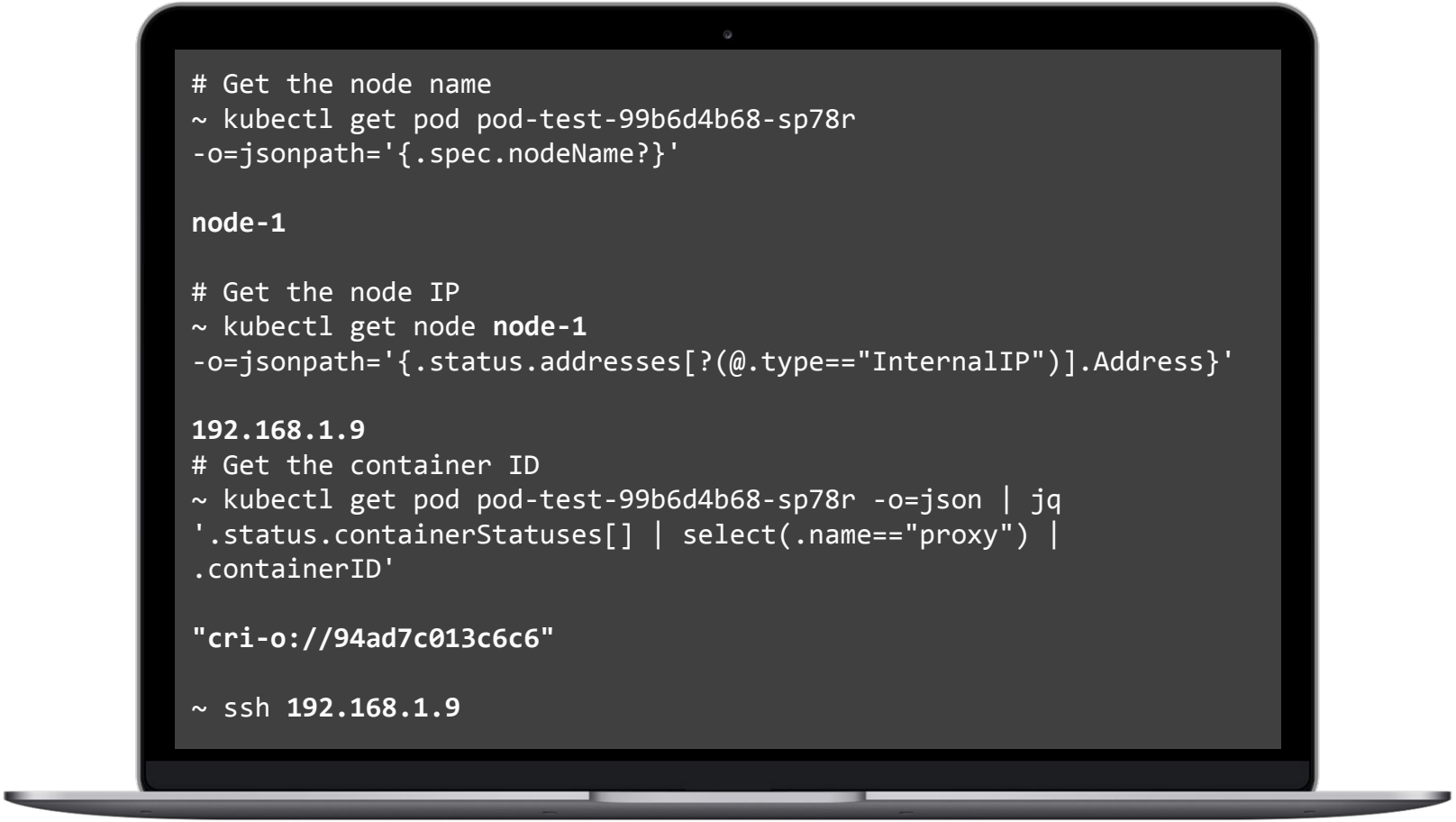
Just install tcpdump

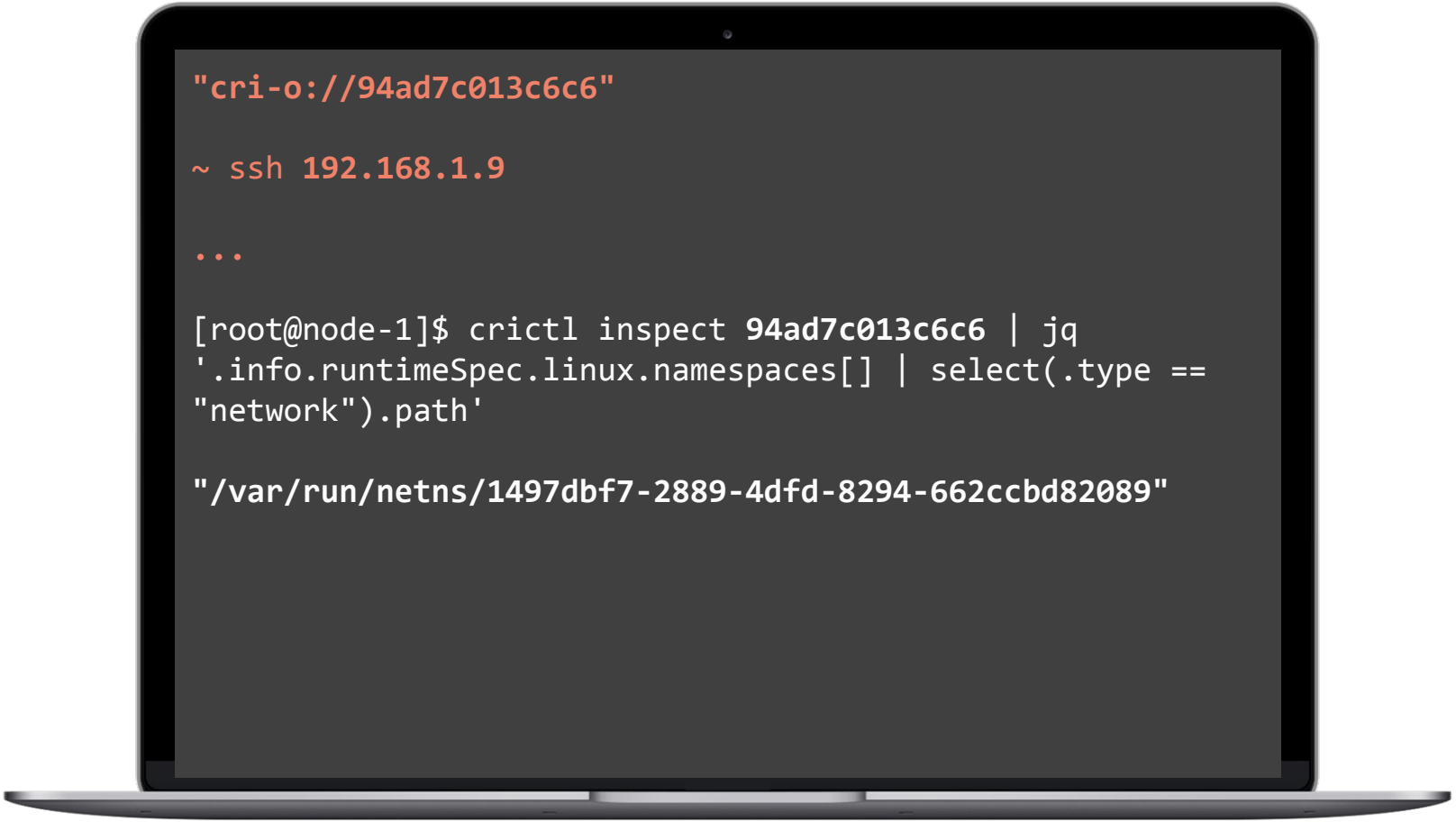
```
$ apt update && apt install tcpdump
```

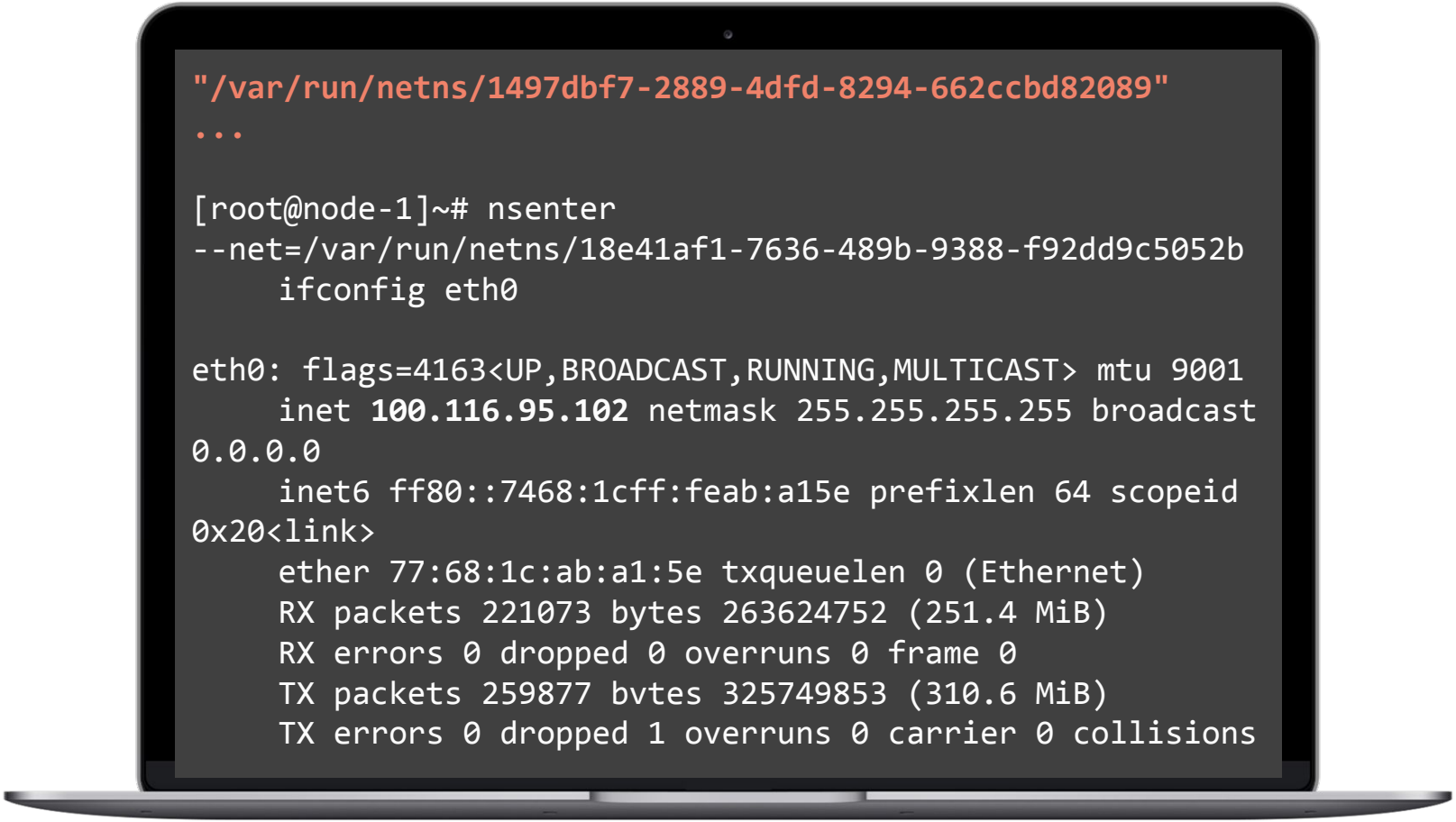

~ kubectl exec -it pod-test-9966d4b68-8n71v -c netshoot
-- tcpdump
tcpdump: eth0: You don't have permission to capture on
that device
(socket: Operation not permitted)
command terminated with exit code 1

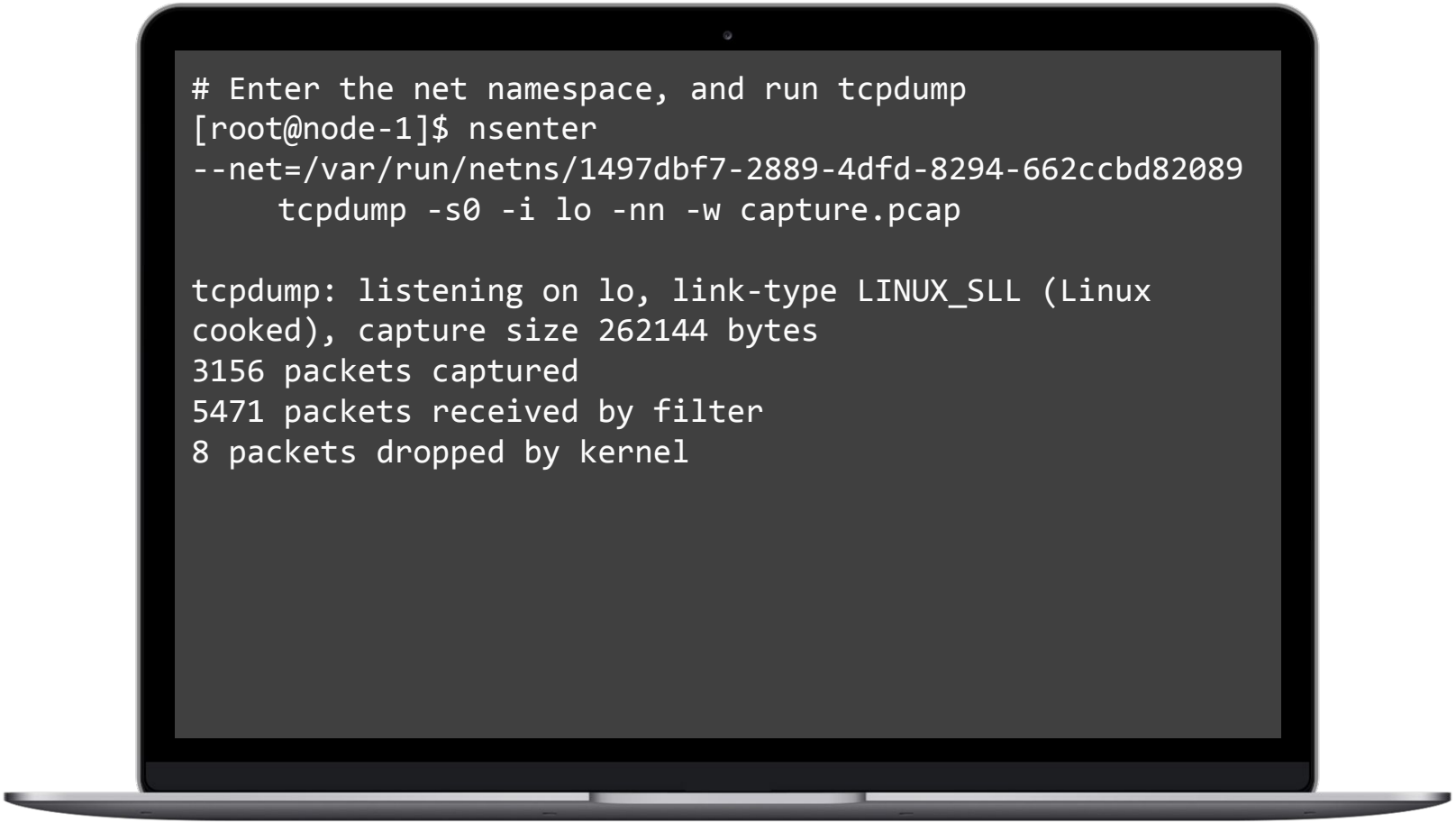
rootless

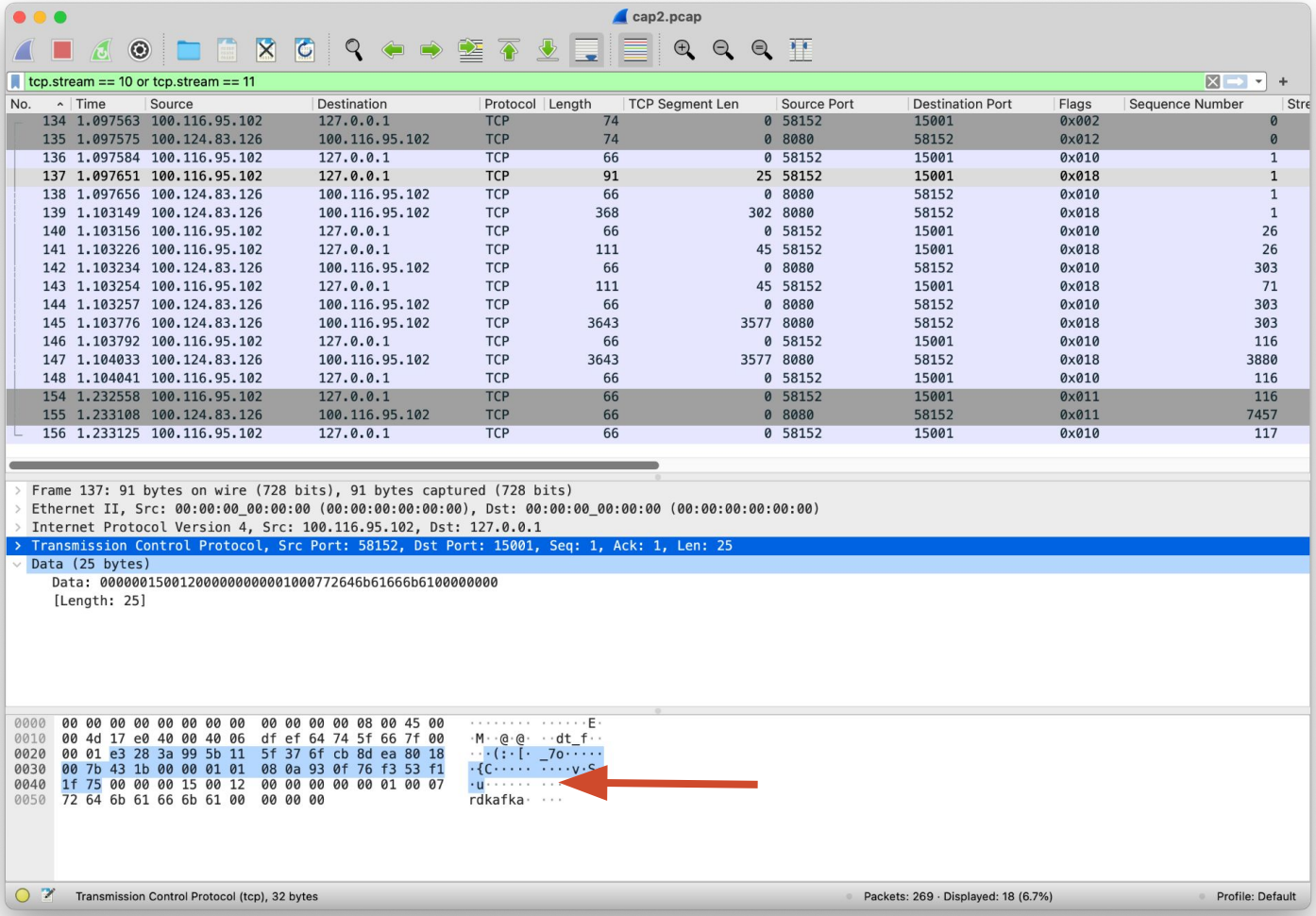
1. Add NET_ADMIN / NET_RAW to the pod's security context
2. **SSH to the node**

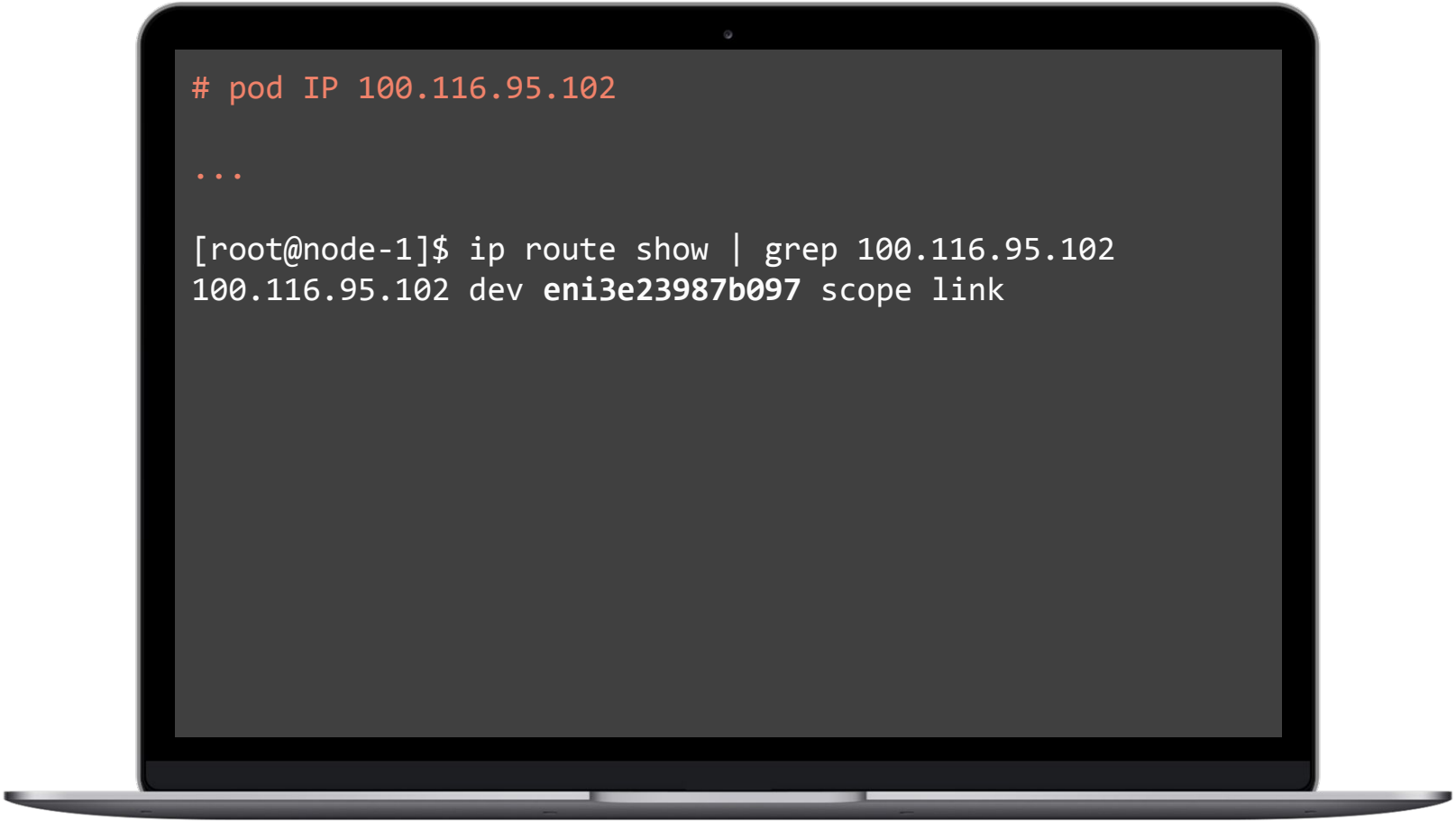

Get the node name
~ kubectl get pod pod-test-99b6d4b68-sp78r
-o=jsonpath='{.spec.nodeName?}'
node-1
Get the node IP
~ kubectl get node node-1
-o=jsonpath='{.status.addresses[?(@.type=="InternalIP")].Address}'
192.168.1.9
Get the container ID
~ kubectl get pod pod-test-99b6d4b68-sp78r -o=json | jq
'.status.containerStatuses[] | select(.name=="proxy") |
.containerID'
"cri-o://94ad7c013c6c6"
~ ssh 192.168.1.9

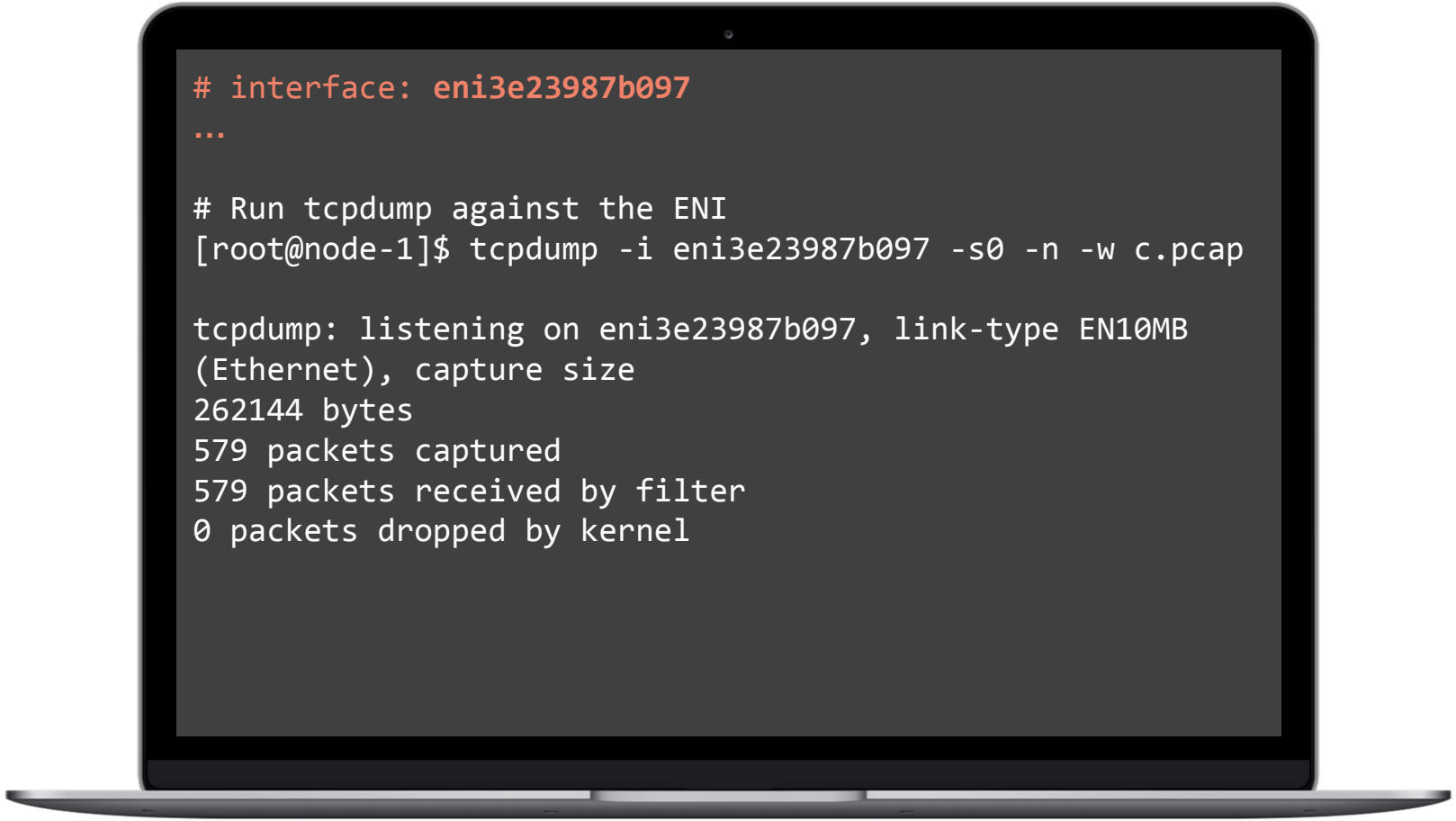

"cri-o://94ad7c013c6c6"
~ ssh 192.168.1.9
...
[root@node-1]$ crictl inspect 94ad7c013c6c6 | jq '.info.runtimeSpec.linux.namespaces[] | select(.type == "network").path'
"/var/run/netns/1497dbf7-2889-4dfd-8294-662ccbd82089"


"/var/run/netns/1497dbf7-2889-4dfd-8294-662ccbd82089"
...
[root@node-1]~# nsenter
--net=/var/run/netns/18e41af1-7636-489b-9388-f92dd9c5052b
ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 9001
inet 100.116.95.102 netmask 255.255.255.255 broadcast
0.0.0.0
inet6 ff80::7468:1cff:feab:a15e prefixlen 64 scopeid
0x20<link>
ether 77:68:1c:ab:a1:5e txqueuelen 0 (Ethernet)
RX packets 221073 bytes 263624752 (251.4 MiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 259877 bvtes 325749853 (310.6 MiB)
TX errors 0 dropped 1 overruns 0 carrier 0 collisions


Enter the net namespace, and run tcpdump
[root@node-1]$ nsenter
--net=/var/run/netns/1497dbf7-2889-4dfd-8294-662ccbd82089
tcpdump -s0 -i lo -nn -w capture.pcap
tcpdump: listening on lo, link-type LINUX_SLL (Linux
cooked), capture size 262144 bytes
3156 packets captured
5471 packets received by filter
8 packets dropped by kernel


cap2.pcap
tcp.stream == 10 or tcp.stream == 11
No. Time Source Destination Protocol Length TCP Segment Len Source Port Destination Port Flags Sequence Number Stre
134 1.097563 100.116.95.102 127.0.0.1 TCP 74 0 58152 15001 0x002 0
135 1.097575 100.124.83.126 100.116.95.102 TCP 74 0 8080 58152 0x012 0
136 1.097584 100.116.95.102 127.0.0.1 TCP 66 0 58152 15001 0x010 1
137 1.097651 100.116.95.102 127.0.0.1 TCP 91 25 58152 15001 0x018 1
138 1.097656 100.124.83.126 100.116.95.102 TCP 66 0 8080 58152 0x010 1
139 1.103149 100.124.83.126 100.116.95.102 TCP 368 302 8080 58152 0x018 1
140 1.103156 100.116.95.102 127.0.0.1 TCP 66 0 58152 15001 0x010 26
141 1.103226 100.116.95.102 127.0.0.1 TCP 111 45 58152 15001 0x018 26
142 1.103234 100.124.83.126 100.116.95.102 TCP 66 0 8080 58152 0x010 303
143 1.103254 100.116.95.102 127.0.0.1 TCP 111 45 58152 15001 0x018 71
144 1.103257 100.124.83.126 100.116.95.102 TCP 66 0 8080 58152 0x010 303
145 1.103776 100.124.83.126 100.116.95.102 TCP 3643 3577 8080 58152 0x018 303
146 1.103792 100.116.95.102 127.0.0.1 TCP 66 0 58152 15001 0x010 116
147 1.104033 100.124.83.126 100.116.95.102 TCP 3643 3577 8080 58152 0x018 3880
148 1.104041 100.116.95.102 127.0.0.1 TCP 66 0 58152 15001 0x010 116
154 1.232558 100.116.95.102 127.0.0.1 TCP 66 0 58152 15001 0x011 116
155 1.233108 100.124.83.126 100.116.95.102 TCP 66 0 8080 58152 0x011 7457
156 1.233125 100.116.95.102 127.0.0.1 TCP 66 0 58152 15001 0x010 117
Frame 137: 91 bytes on wire (728 bits), 91 bytes captured (728 bits)
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 100.116.95.102, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 58152, Dst Port: 15001, Seq: 1, Ack: 1, Len: 25
Data (25 bytes)
Data: 000000150012000000000001000772646b61666b6100000000
[Length: 25]
0000 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00
0010 00 4d 17 e0 40 00 40 06 df ef 64 74 5f 66 7f 00
0020 00 01 e3 28 3a 99 5b 11 5f 37 6f cb 8d ea 80 18
0030 00 7b 43 1b 00 00 01 01 08 0a 93 0f 76 f3 53 f1
0040 1f 75 00 00 00 15 00 12 00 00 00 00 00 01 00 07
0050 72 64 6b 61 66 6b 61 00 00 00 00
Transmission Control Protocol (tcp), 32 bytes
Packets: 269 · Displayed: 18 (6.7%)
Profile: Default


pod IP 100.116.95.102
...
[root@node-1]$ ip route show | grep 100.116.95.102
100.116.95.102 dev eni3e23987b097 scope link

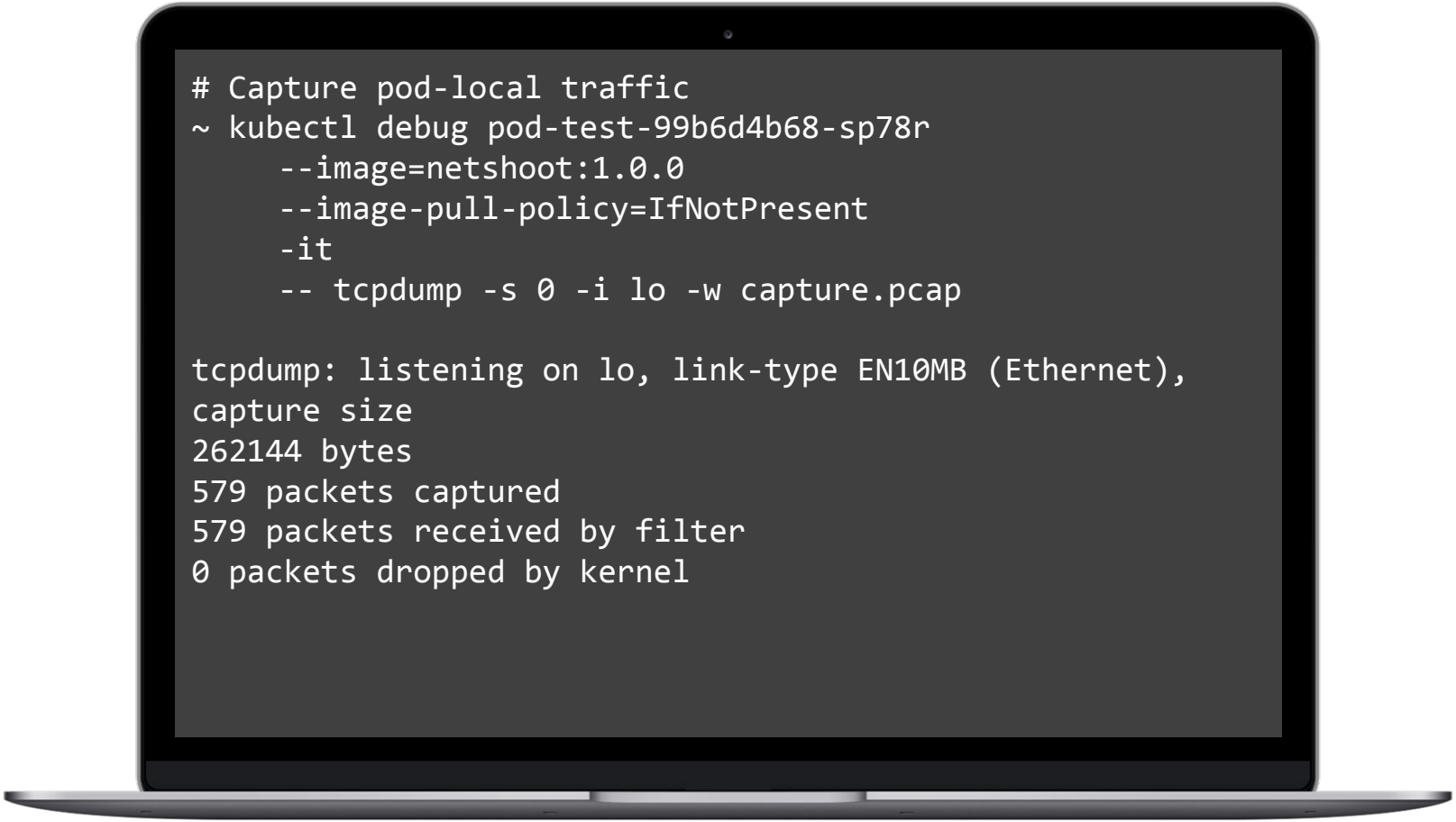

interface: eni3e23987b097
...

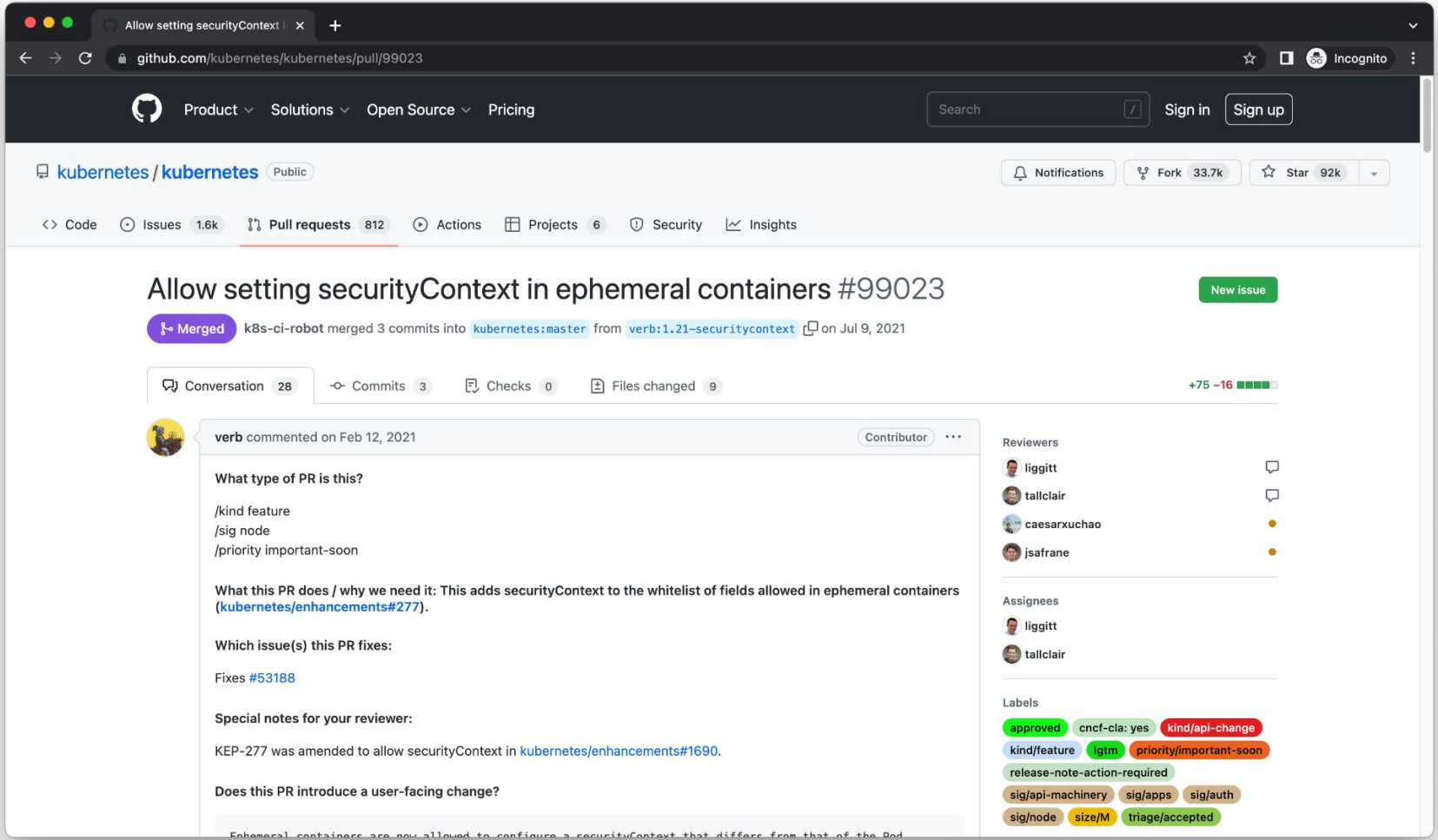
Run tcpdump against the ENI
[root@node-1]$ tcpdump -i eni3e23987b097 -s0 -n -w c.pcap

tcpdump: listening on eni3e23987b097, link-type EN10MB
(Ethernet), capture size
262144 bytes
579 packets captured
579 packets received by filter
0 packets dropped by kernel

Ephemeral containers

Currently don’t support changing the security context (NET_RAW + NET_ADMIN)


Capture pod-local traffic
~ kubectl debug pod-test-99b6d4b68-sp78r
--image=netshoot:1.0.0
--image-pull-policy=IfNotPresent
-it
-- tcpdump -s 0 -i lo -w capture.pcap
tcpdump: listening on lo, link-type EN10MB (Ethernet),
capture size
262144 bytes
579 packets captured
579 packets received by filter
0 packets dropped by kernel


Allow setting securityContext
github.com/kubernetes/kubernetes/pull/99023
Incognito
Product
Solutions
Open Source
Pricing
Search
Sign in
Sign up
kubernetes / kubernetes
Public
Notifications
Fork 33.7k
Star 92k
Code
Issues 1.6k
Pull requests 812
Actions
Projects 6
Security
Insights
Allow setting securityContext in ephemeral containers #99023
New issue
Merged
k8s-ci-robot merged 3 commits into kubernetes:master from verb:1.21-securitycontext on Jul 9, 2021
Conversation 28
Commits 3
Checks 0
Files changed 9
+75 −16
verb commented on Feb 12, 2021
Contributor
What type of PR is this?
/kind feature
/sig node
/priority important-soon
What this PR does / why we need it: This adds securityContext to the whitelist of fields allowed in ephemeral containers (kubernetes/enhancements#277).
Which issue(s) this PR fixes:
Fixes #53188
Special notes for your reviewer:
KEP-277 was amended to allow securityContext in kubernetes/enhancements#1690.
Does this PR introduce a user-facing change?
Reviewers
liggitt
tallclair
caesarxuchao
jsafrane
Assignees
liggitt
tallclair
Labels
approved
cncf-cla: yes
kind/api-change
kind/feature
lgtm
priority/important-soon
release-note-action-required
sig/api-machinery
sig/apps
sig/auth
sig/node
size/M
triage/accepted

Recap

- Linux doesn’t have containers
- The network namespace isolates network resources
- iptables rewrite the packet header
- The proxy looks at the socket table

- TCP observability is limited
- tcpdump the pod on loopback via nsenter
- tcpdump the proxy via host + interface
- Ephemeral containers will save us

airbnb.com/careers

BUOYANT'S SERVICE MESH ACADEMY

The creators of LINKERD

Monthly hands-on, engineer-focused training from the creators of the service mesh

Nov 11-17: What really happens at startup: a deep dive into Linkerd, init containers, CNI plugins, and more

SIGN UP TODAY!

buoyant.io/sma

The end!