



Resiliency Superpowers with eBPF

Liz Rice

Chief Open Source Officer, Isovalent

Ambassador & board member, OpenUK

Emeritus chair, CNCF Technical Oversight Committee

@lizrice



ISOVALENT





What is eBPF?

extended
Berkeley
Packet
Filter

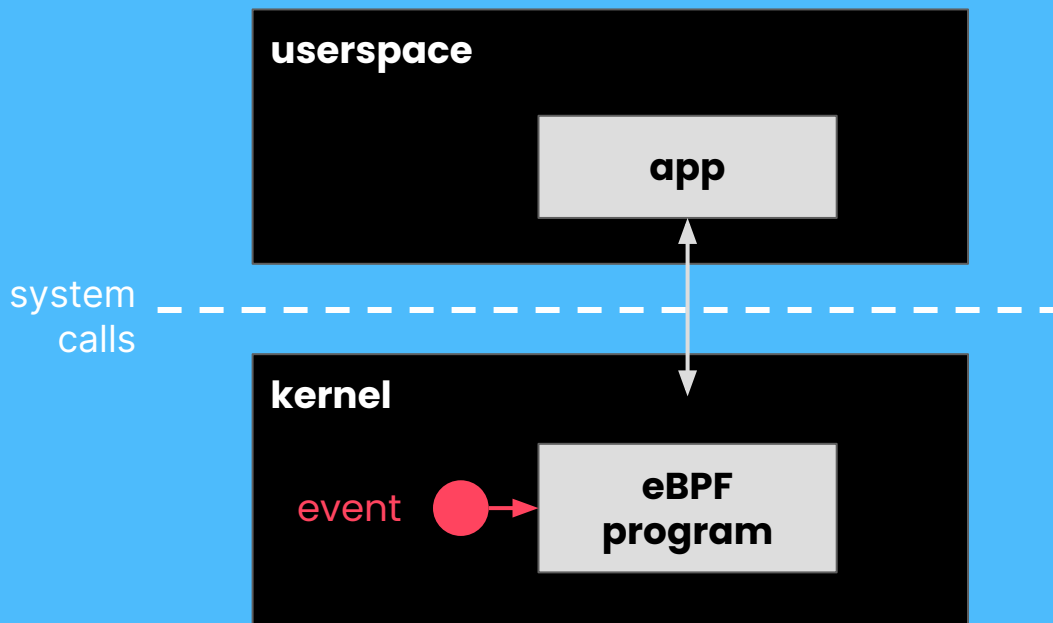




What is eBPF?

Makes the kernel **programmable**

Run custom code in the kernel



eBPF Hello World

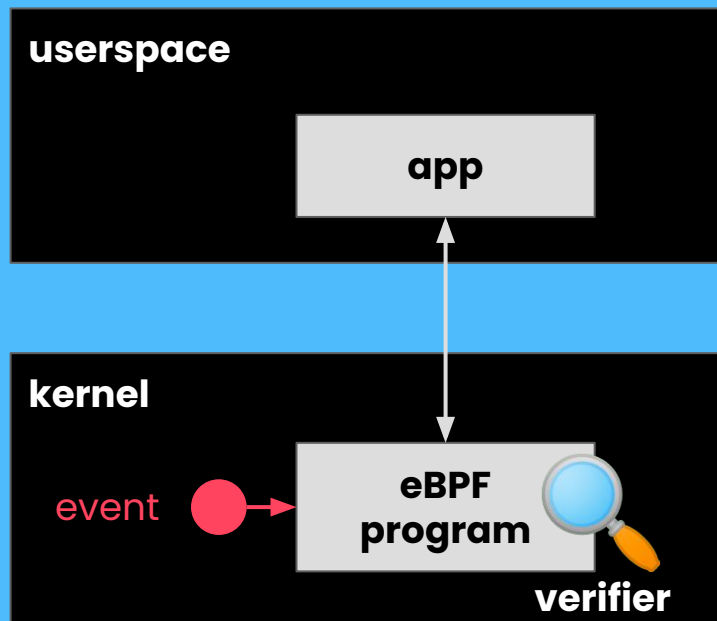
```
SEC("kprobe/sys_execve")
int hello(void *ctx)
{
    bpf_printk("I'm alive!");
    return 0;
}
```

+ userspace code to load eBPF program & attach to event

```
$ sudo ./hello
bash-20241 [004] d... 84210.752785: 0: I'm alive!
bash-20242 [004] d... 84216.321993: 0: I'm alive!
bash-20243 [004] d... 84225.858880: 0: I'm alive!
```

Info about process that called execve syscall

eBPF code has to be safe





Dynamically change kernel behavior



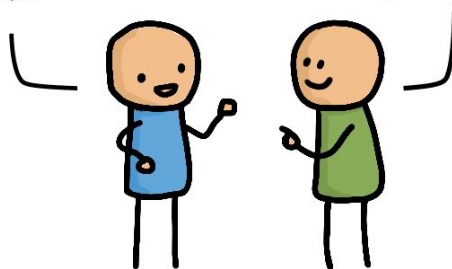
Application Developer:

I want this new feature to observe my app



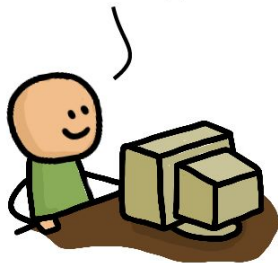
Hey kernel developer! Please add this new feature to the Linux kernel

OK! Just give me a year to convince the entire community that this is good for everyone.

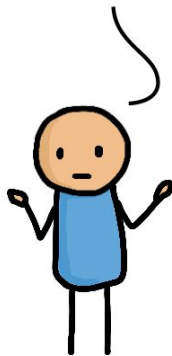


1 year later...

I'm done. The upstream kernel now supports this.



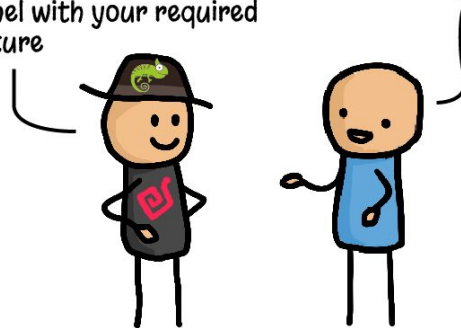
But I need this in my Linux distro



5 years later...

Good news. Our Linux distribution now ships a kernel with your required feature

OK but my requirements have changed since...



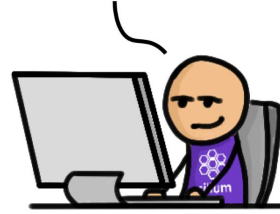
Application Developer:

i want this new feature to observe my app



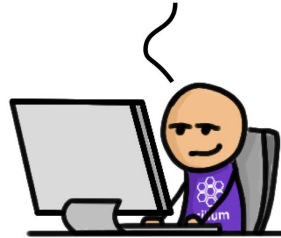
eBPF Developer:

OK! The kernel can't do this so let me quickly solve this with eBPF.



A couple of days later...

Here is a release of our eBPF project that has this feature now. BTW, you don't have to reboot your machine.

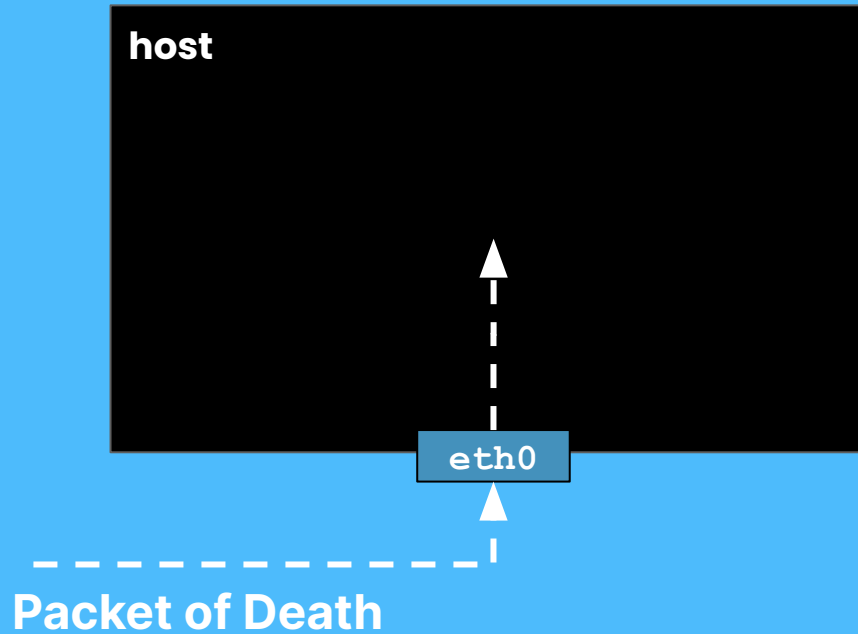




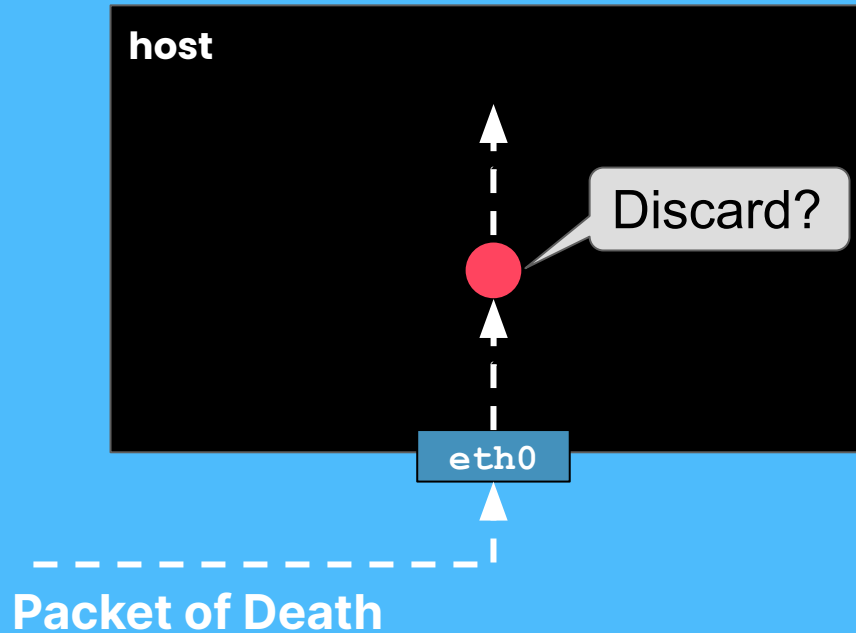
Resilience to exploits

Dynamic vulnerability patching

Packet of Death



Packet of Death





eBPF Packet Drop

```
SEC("xdp/bye")
int goodbye_ping(struct xdp_md *ctx)
{
    ...
    if (iph->protocol == IPPROTO_ICMP)
        return XDP_DROP;

    return XDP_PASS;
}
```

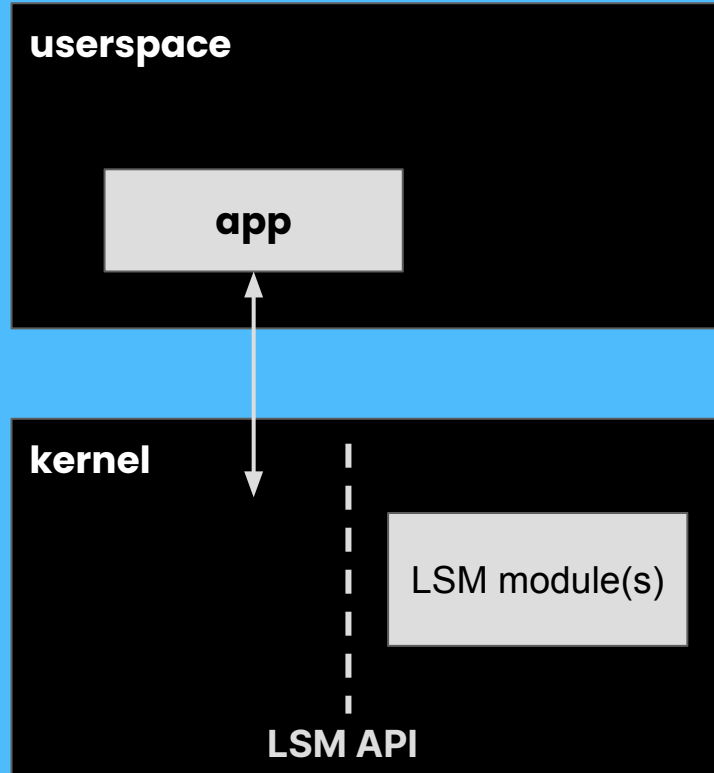


Resilience to exploits

BPF Linux Security Module

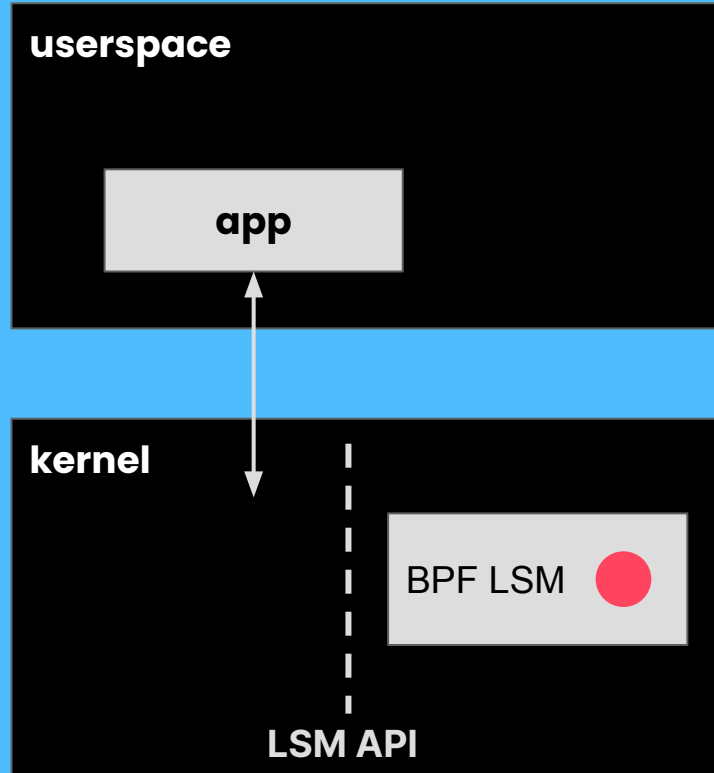


LSM



**Inflexible
security policies**

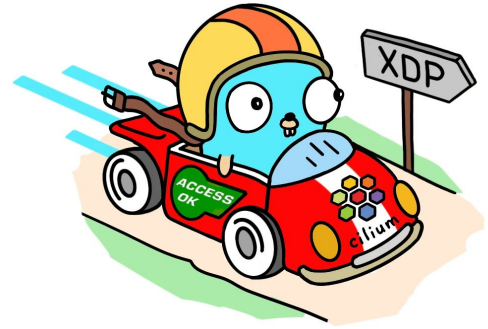
BPF LSM



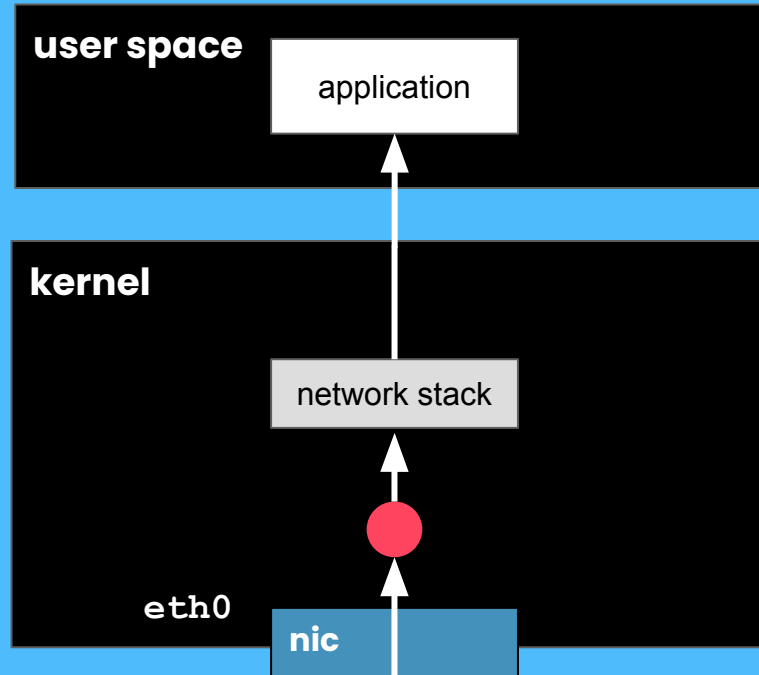
**Dynamic,
custom security
policies**

Resilience to failure

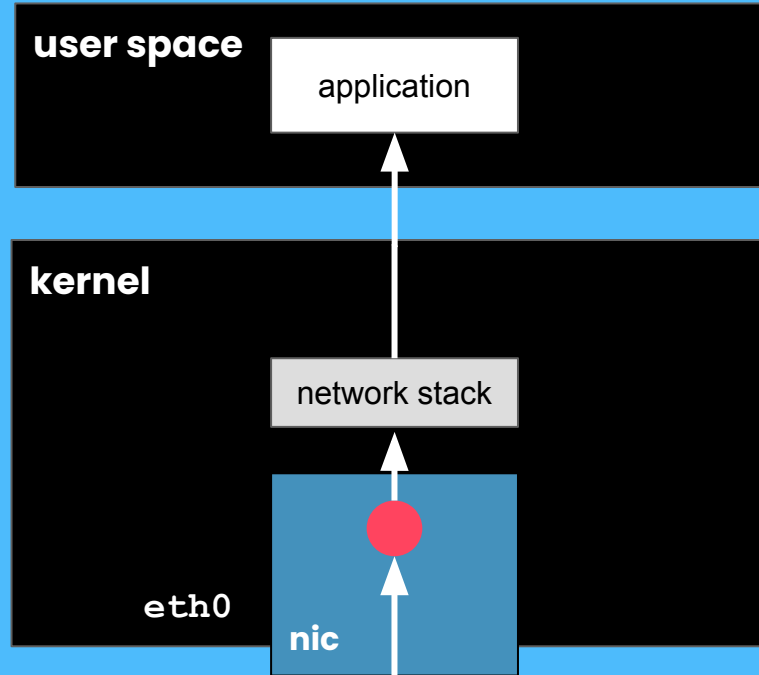
Super-fast load balancing



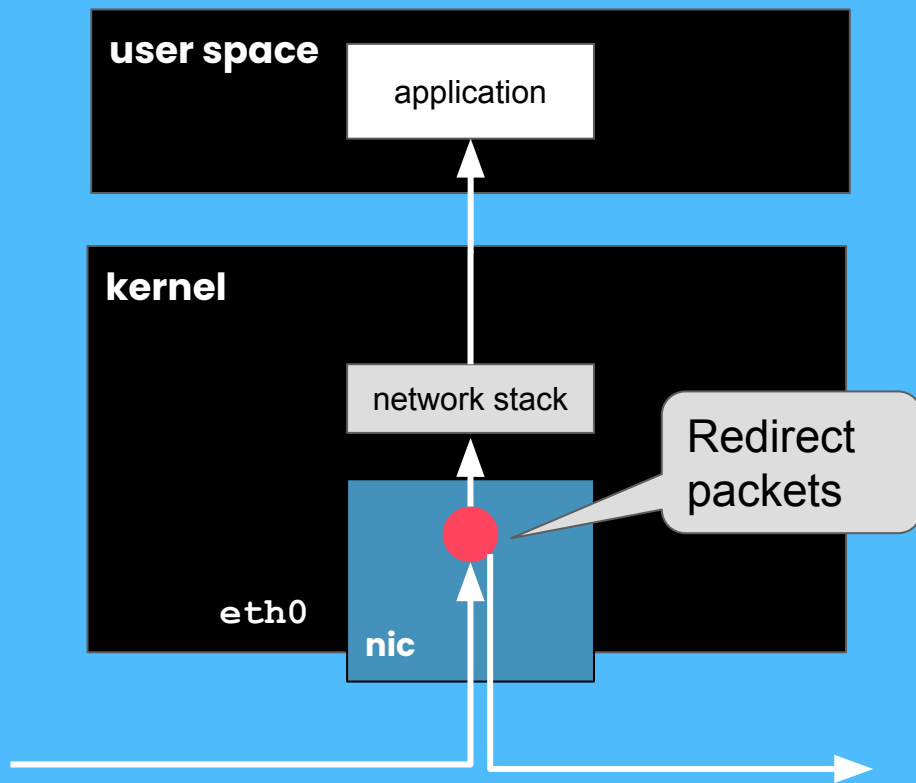
eXpress Data Path



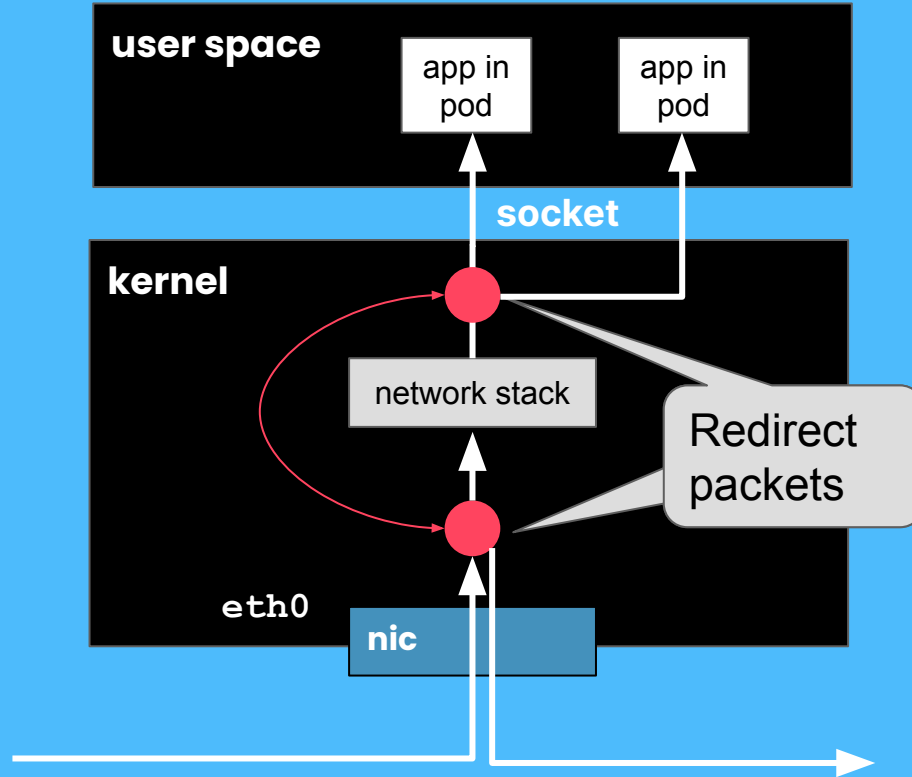
eXpress Data Path



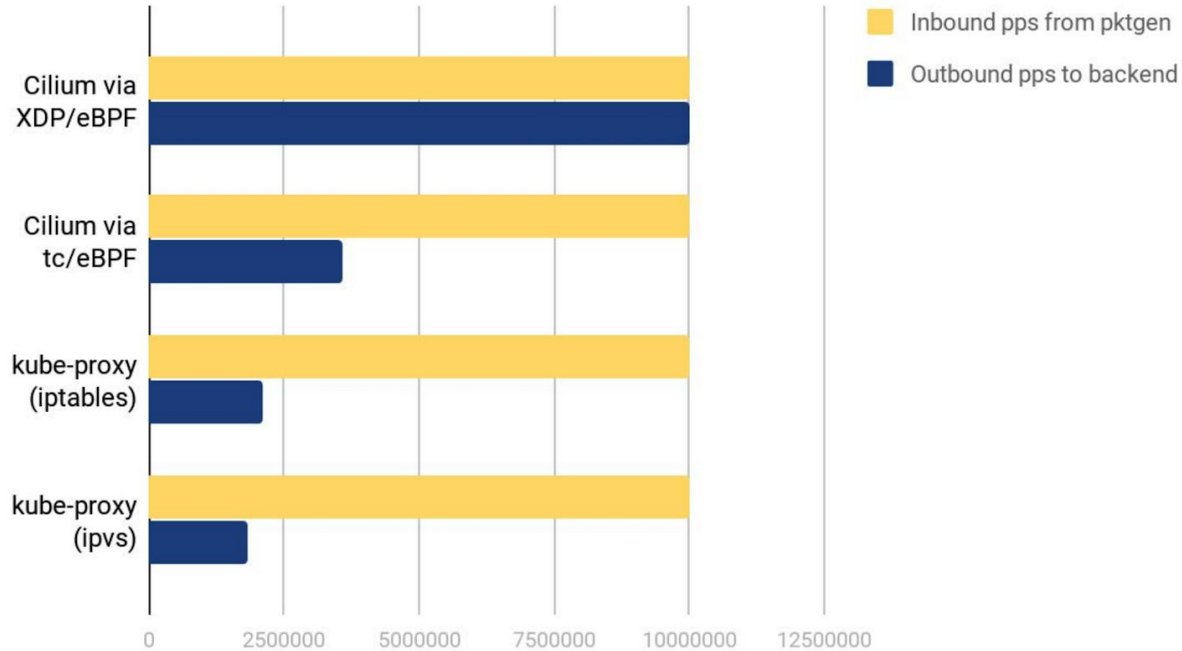
L3/4 load balancing



kube-proxy replacement



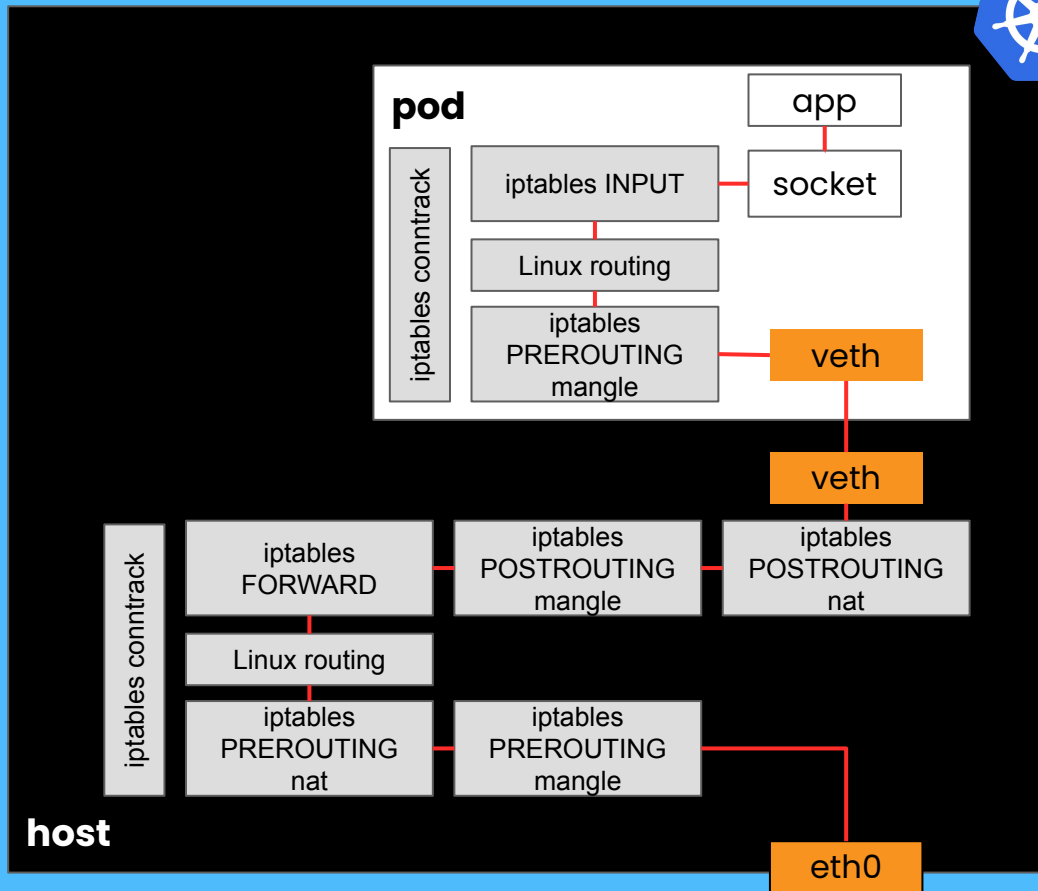
kube-proxy replacement performance

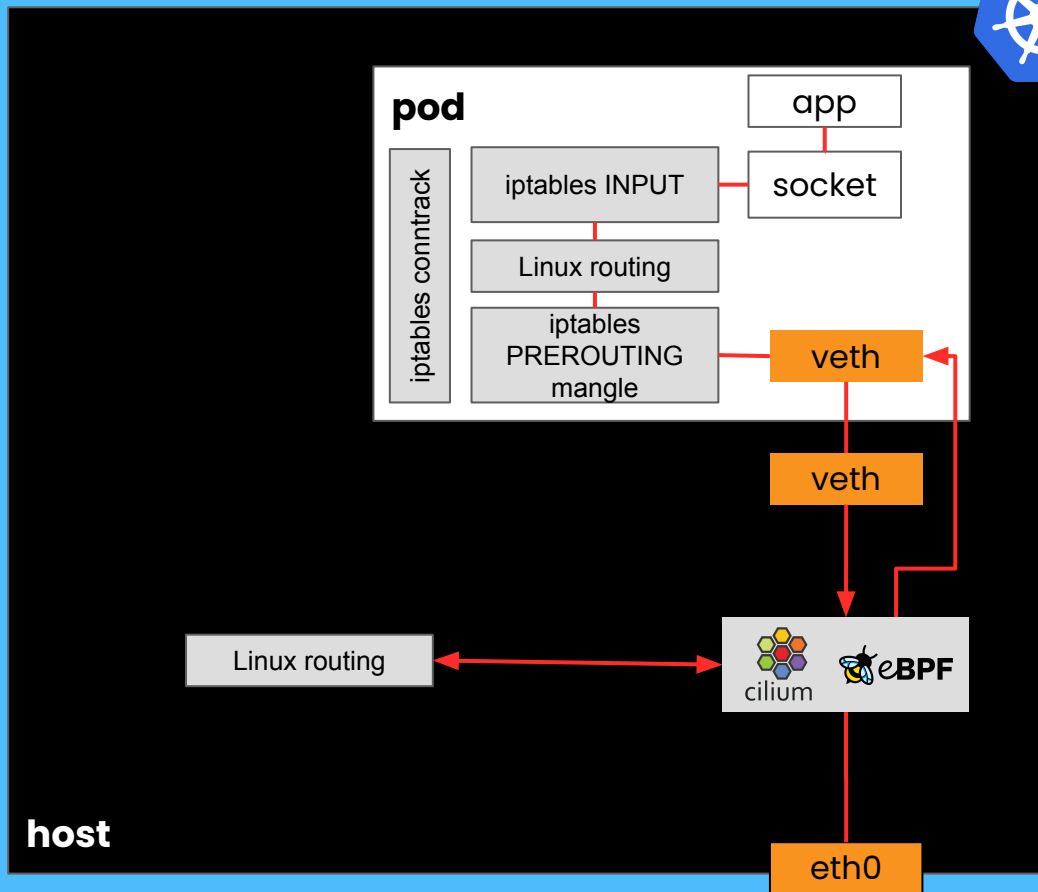




eBPF enables efficient **Kubernetes-aware** networking







Linux routing

cilium eBPF

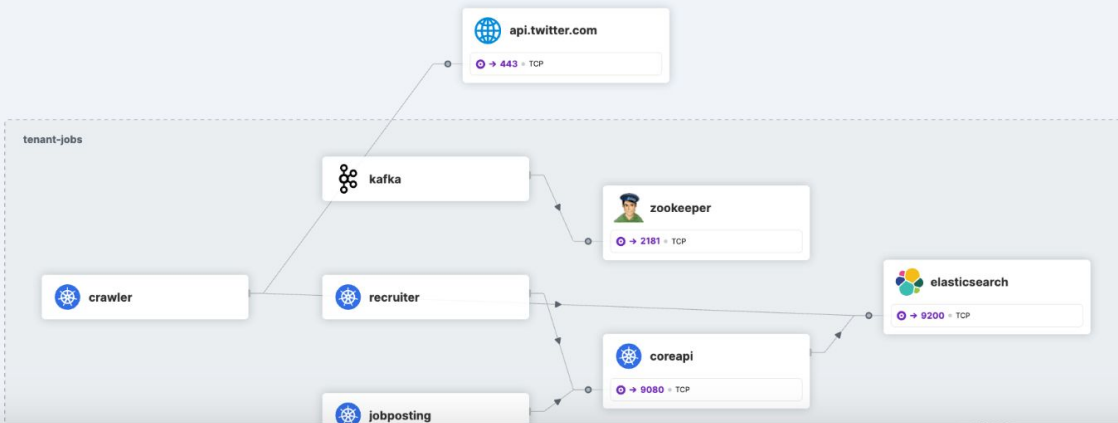
host

eth0



for example

Kubernetes-aware network flows



Columns ▾

Source Pod	Source Service	Destination Service	Destination
kafka-0	kafka tenant-jobs	zookeeper tenant-jobs	
kafka-0	kafka tenant-jobs	zookeeper tenant-jobs	
—	coreapi tenant-jobs	elasticsearch tenant-jobs	
—	coreapi tenant-jobs	elasticsearch tenant-jobs	
recruiter-f44d5f778-86qkv	recruiter tenant-jobs	coreapi tenant-jobs	
recruiter-f44d5f778-86qkv	recruiter tenant-jobs	coreapi tenant-jobs	
—	recruiter tenant-jobs	coreapi tenant-jobs	
—	recruiter tenant-jobs	coreapi tenant-jobs	
—	recruiter tenant-jobs	coreapi tenant-jobs	
coreapi-79568ff848-k98zq	coreapi tenant-jobs	elasticsearch tenant-jobs	
recruiter-f44d5f778-86qkv	recruiter tenant-jobs	coreapi tenant-jobs	
—	recruiter tenant-jobs	coreapi tenant-jobs	
—	recruiter tenant-jobs	coreapi tenant-jobs	
—	recruiter tenant-jobs	coreapi tenant-jobs	
—	recruiter tenant-jobs	coreapi tenant-jobs	

01007

Destination labels

```
app=elasticsearch
io.cilium.k8s.policy.cluster=default
io.cilium.k8s.policy.serviceaccount=default
namespace=tenant-jobs
```

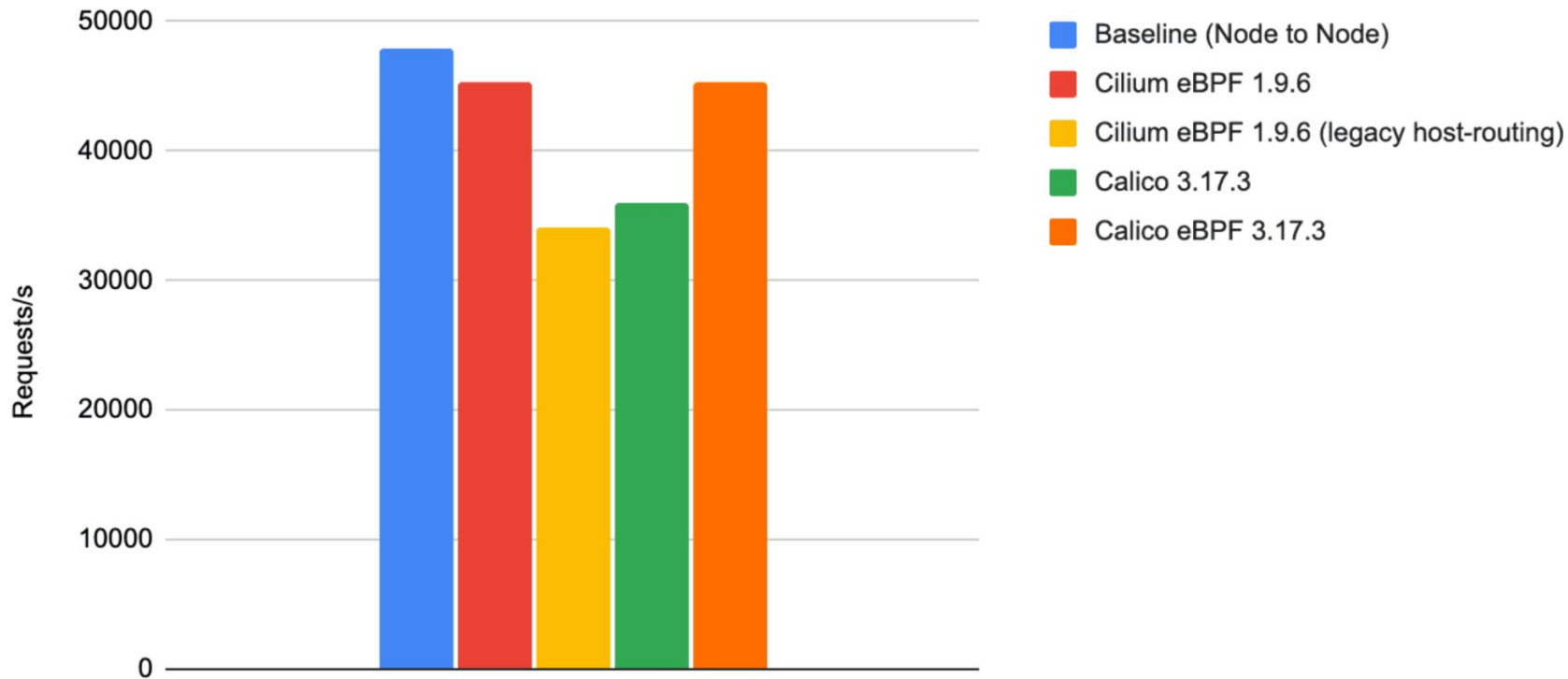
Destination IP
192.168.60.86

Destination port
9200

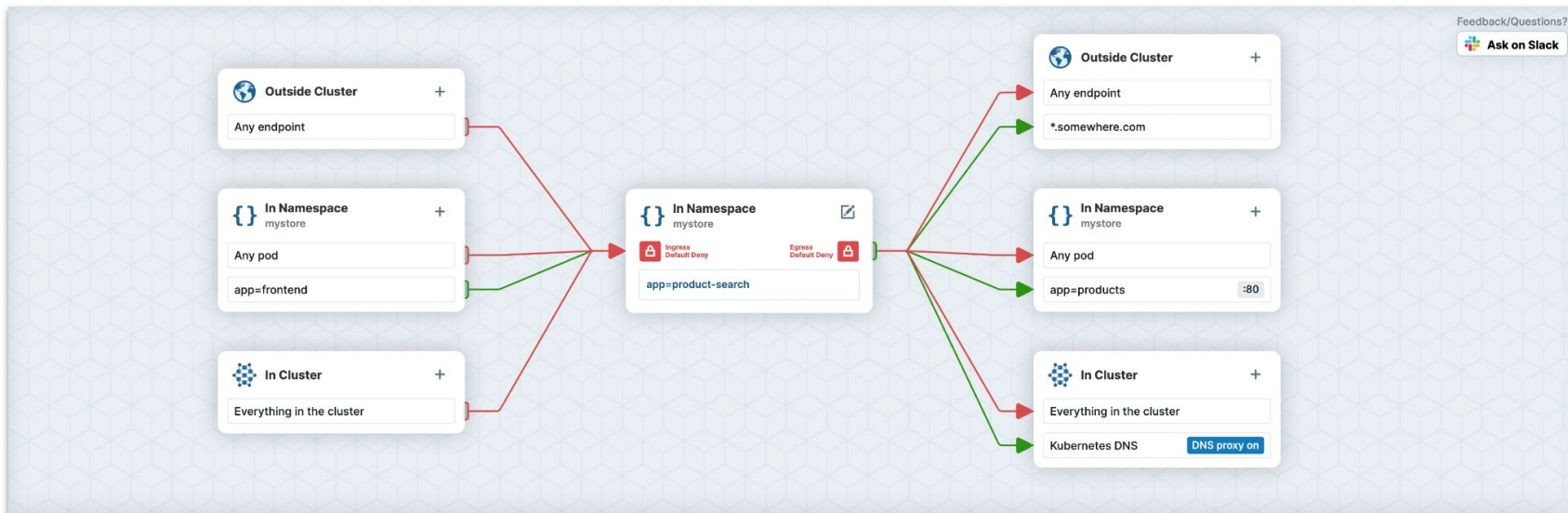
HTTP Request Method
GET

HTTP Request Protocol
HTTP/1.1

TCP RR - higher is better



eBPF network policy decisions



```
Kubernetes Network Policy | Cilium Network Policy | Policy Rating [||||] | Download | Share
```

```
18   toPorts:
19     - ports:
20       - port: "80"
21   - toFQDNs:
22     - matchPattern: "*.somewhere.com"
23   - toEndpoints:
24     - matchLabels:
25       io.kubernetes.pod.namespace: kube-system
26       k8s-app: kube-dns
27   toPorts:
28     - ports:
```

Main tutorial | Flows upload

Welcome to the Network Policy Editor! *Beta*

This tutorial will teach you how to create a network policy using the Editor. It explains basic network policy concepts and guides you through the steps needed to achieve the desired least-privilege security and zero-trust concepts.

Step 1. What pods do you want to secure?

The diagram shows a 'Cluster' containing two namespaces: 'namespaceA' and 'namespaceB'. Each namespace contains several pod icons, representing the scope of network policy enforcement.

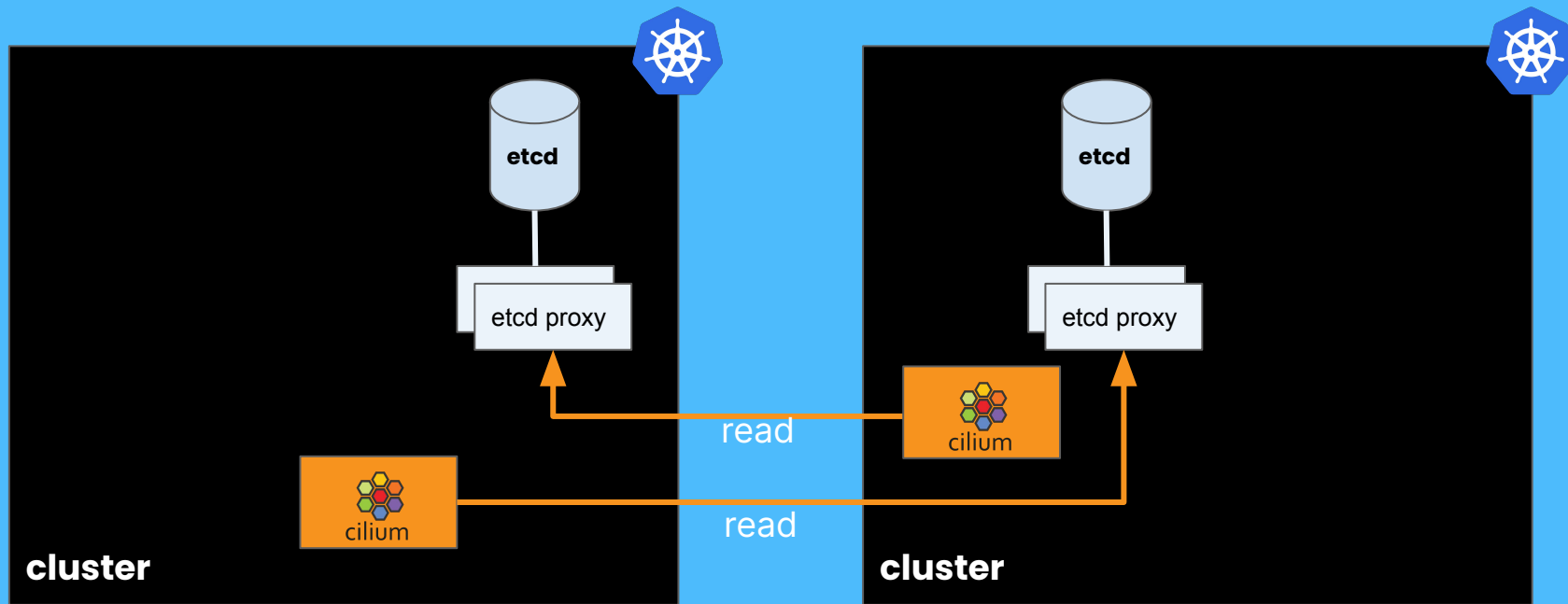


Resilience to failure

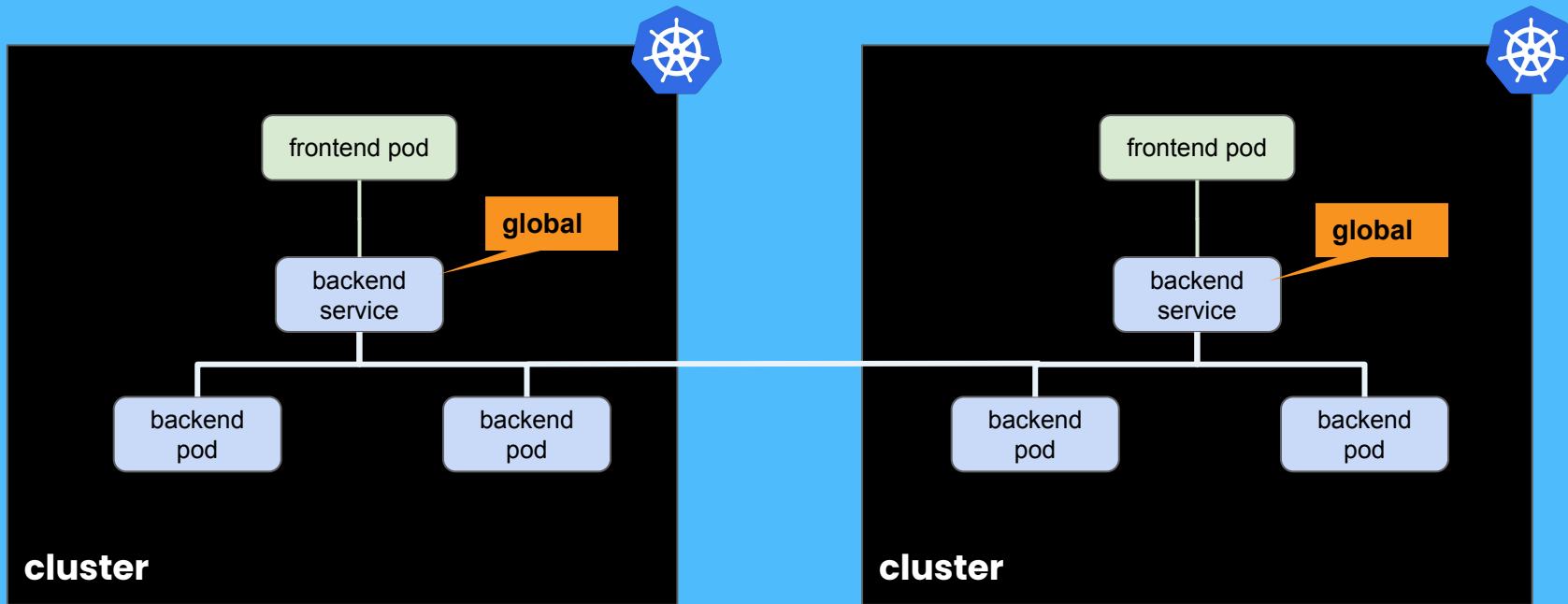
Clustermesh



Share multi-cluster state



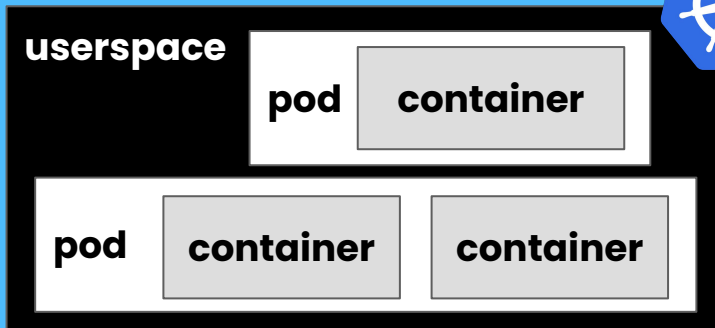
Highly available services





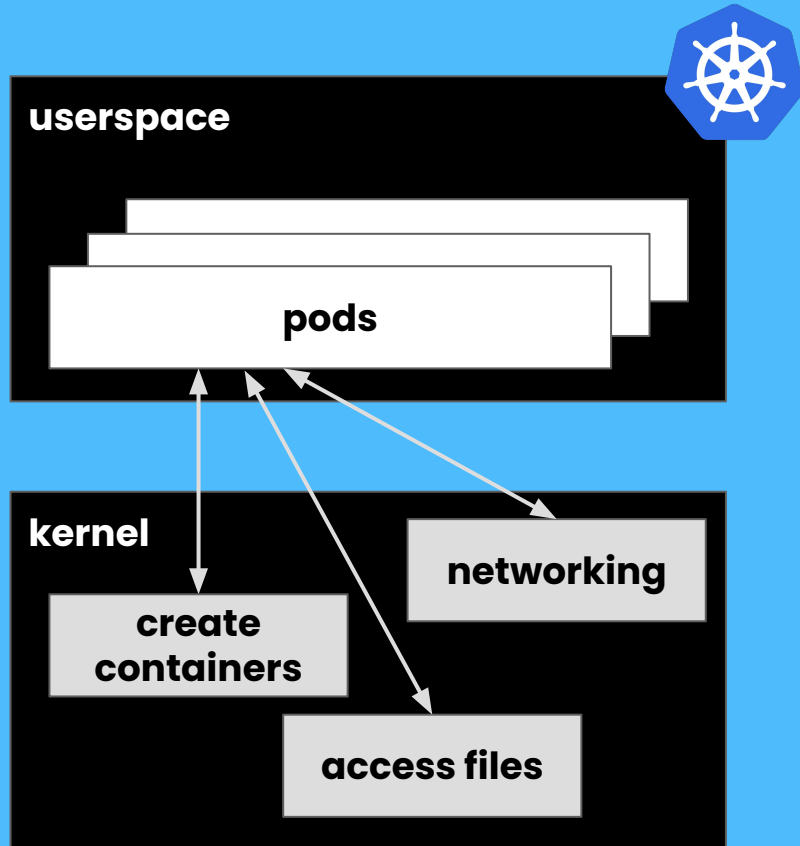
Visibility into failures

eBPF observability instrumentation

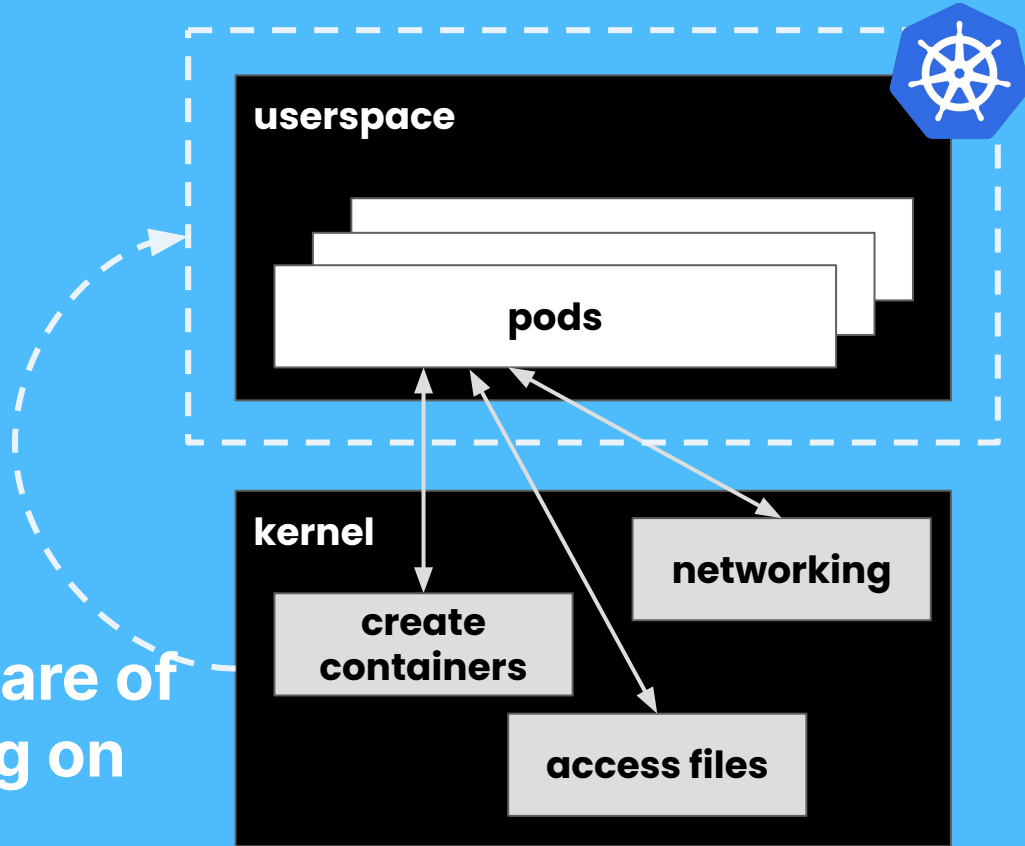


One kernel per host

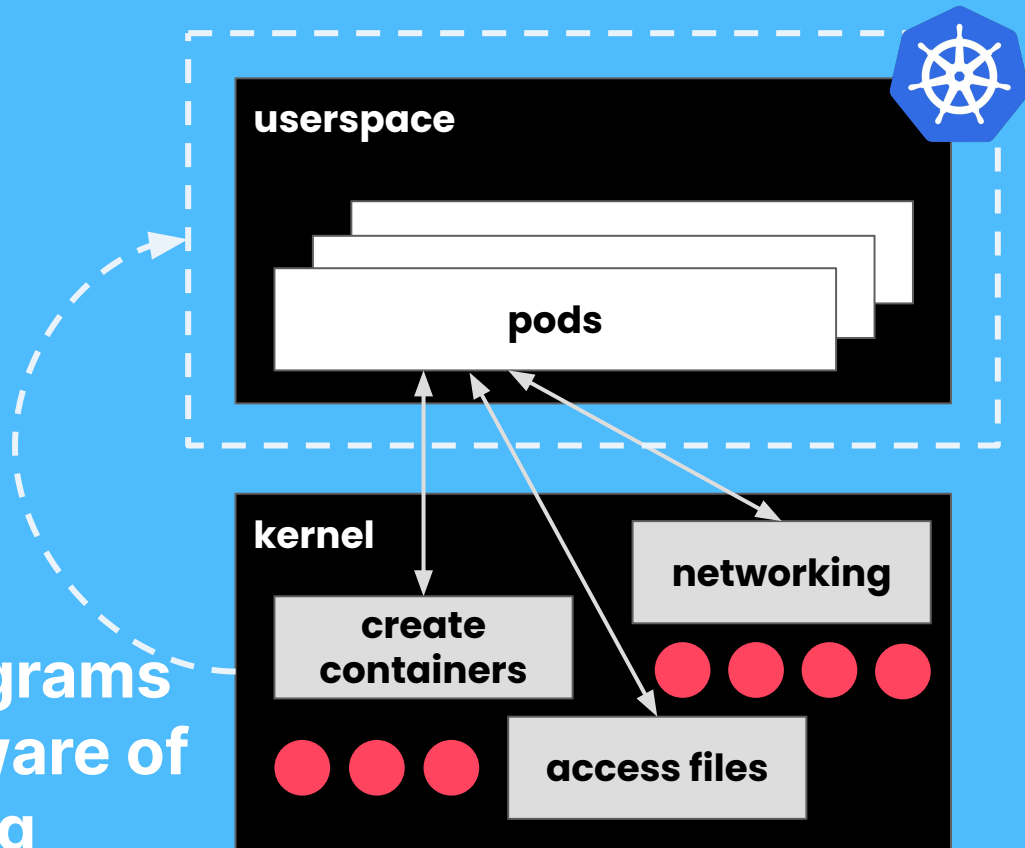
One kernel per host



Kernel aware of everything on the host



eBPF programs
can be aware of
everything





for example Pixie flamegraph

Cluster: gke:fgs-demo

script: px/perf_flamegraph node: namespace: pod: pct_basis_entity: node start_time: -5m

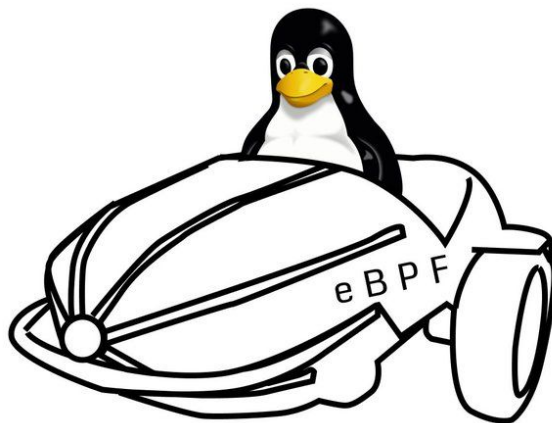
Flamegraph

pid: c...	pid: /f...	pid: /bin/bash /usr/loca...	pid: ...	pid: java -Xms64m -Xmx128m -XX:P...	pid: m...	pid: /usr/b...	pid: /app/src/operator/operator
containe...	contain...	container: elasticsearch	cont...	container: orders	contai...	container: ...	container: app
pod: cil...	pod: fluentb...	pod: elasticsearch-56f8f...	pod: ...	pod: orders-599865f7b7-bst5t	pod: c...	pod: falco...	pod: vizier-operator-5857c76859-1x9hb
namespace: kube-system		namespace: tenant-jobs	name...	namespace: px-sock-shop	namespace: ...	namespace: ...	namespace: px-operator
all							



**eBPF programs have a view
across the entire node
without any app or config changes**

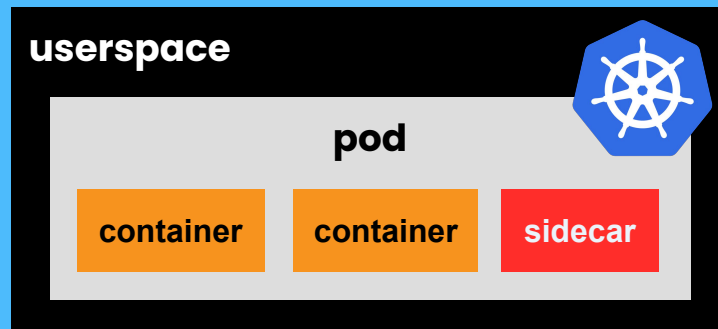
My other
sidecar
is a **kernel**



"Get in loser. We're going tracing"

- **Nathan LeClaire** [@dotpem](#)

A sidecar has a view across one pod



Sidecars need YAML

```
my-app.yaml
containers:
- name: my-app
  ...
- name: my-app-init
  ...
- name: my-sidecar
  ...
```

userspace

pod

container

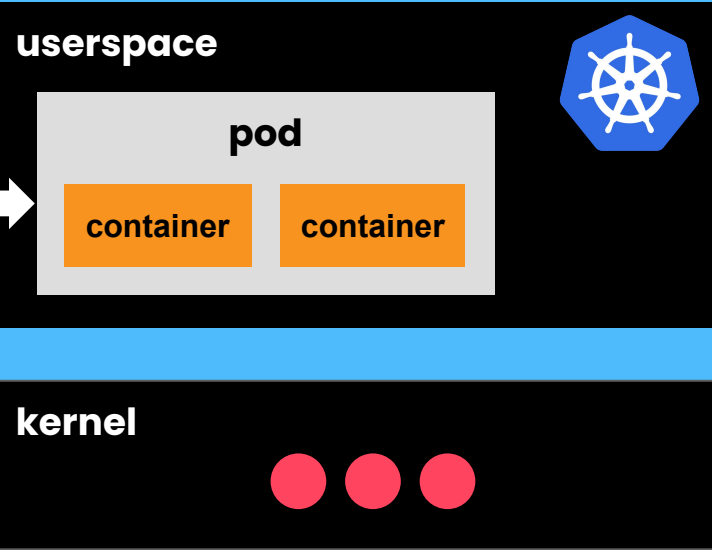
container

sidecar



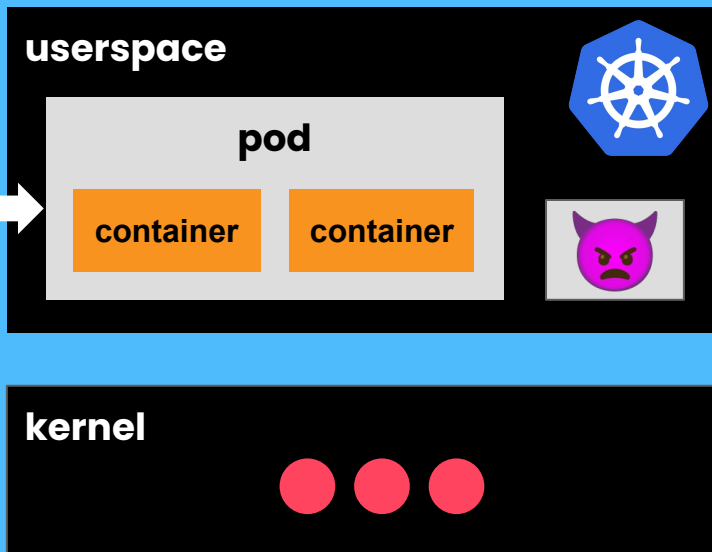
eBPF does not need any app changes

```
my-app.yaml
containers:
- name: my-app
  ...
- name: my-app-init
  ...
```



eBPF can see ALL activity on the node

```
my-app.yaml
containers:
- name: my-app
  ...
- name: my-app-init
  ...
```



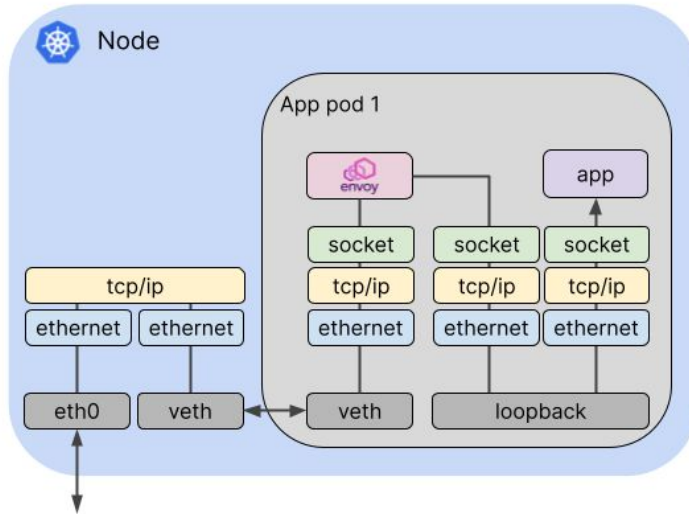


Resilient, observable, secure deployments

Sidecarless service mesh

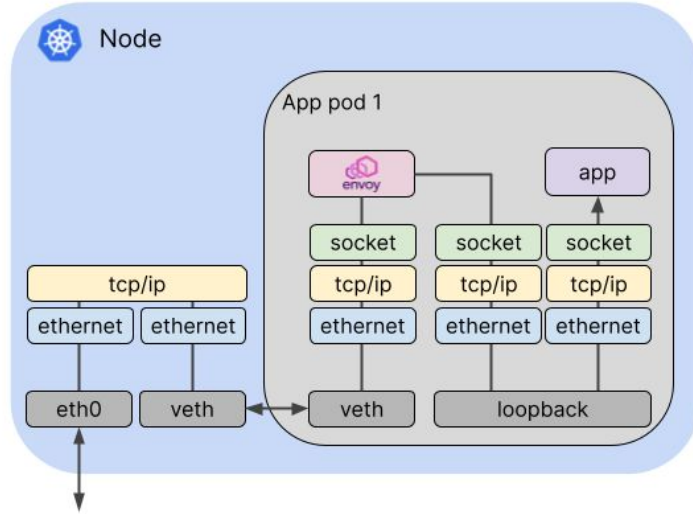
Service mesh

Service mesh with traditional networking

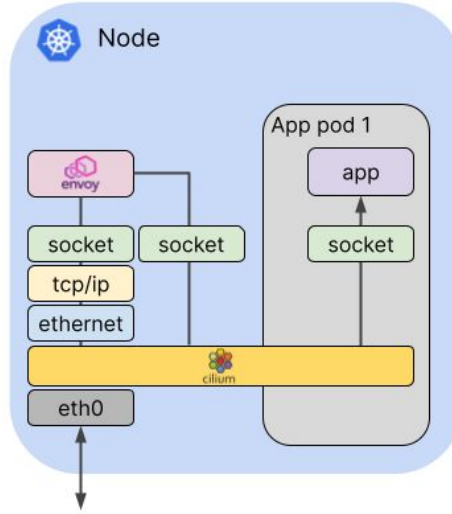


eBPF-accelerated service mesh

Service mesh with traditional networking



Sidecarless model, eBPF acceleration





**eBPF makes the kernel programmable
enabling networking, observability &
security tools for resilient deployment**





Not just for Linux...

☰ README.md

eBPF on Windows

eBPF is a well-known technology for providing programmability and agility, especially for extending an OS kernel, for use cases such as DoS protection and observability. This project is a work-in-progress that allows using existing eBPF toolchains and APIs familiar in the Linux ecosystem to be used on top of Windows. That is, this project takes existing eBPF projects as submodules and adds the layer in between to make them run on top of Windows.



eBPF Foundation

Founding Members

FACEBOOK

Google



ISOVALENT™



Microsoft

NETFLIX



Thank you

ebpf.io | cilium.io | isovalent.com

[@lizrice](#)



ISOVALENT

