
KubeCon
CloudNativeCon
Europe 2021
Virtual
Forward Together »

Breaking your Kubernetes Cluster with Networking

Thomas Graf, Isovalent

KubeCon CloudNativeCon Europe 2021

Virtual


Networking is
Everywhere


Networking is Everywhere
... even in the KubeCon Title Slide


KubeCon
CloudNativeCon
Europe 2021
Virtual
Liveness probes without network awareness
App team scheduling 5K services
kube-proxy
Platform team ignoring crashing CoreDNS pods
iptables
Forward Together
CNI Chaining + kube-proxy + Ingress + CoreDNS + Service Mesh + Cloud Networking


Kubernetes Networking
The Dark Side

Kubernetes Networking The Dark Side

Special Appearance: DNS

Context:
Where are the stories coming from

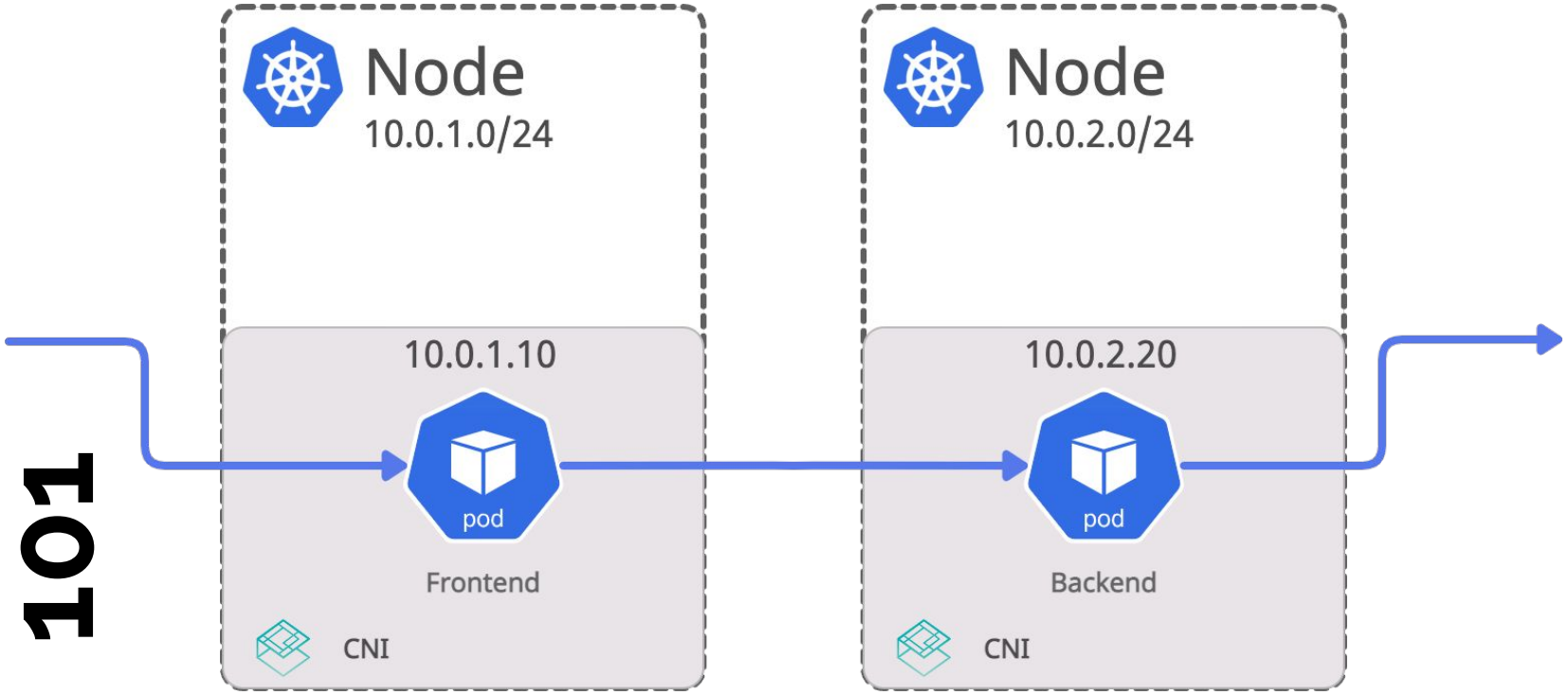
- I'm a Cilium Maintainer
- These are stories from our users

cilium

eBPF-based Networking, Security, and Observability

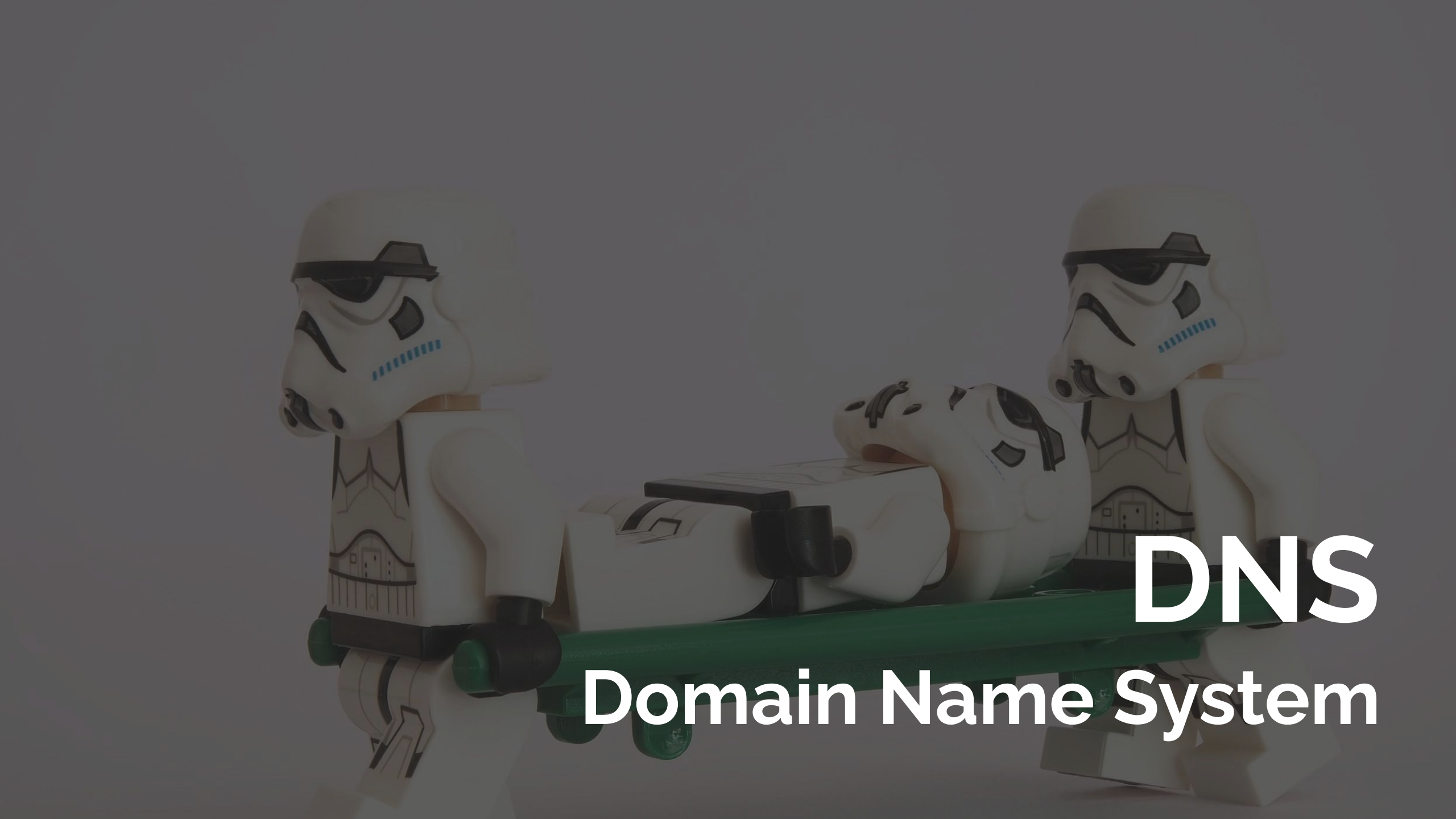
Learn more: cilium.io

Kubernetes Networking 101



- All Pods have IPs
- All Pods can talk
- PodCIDR[s] per node

- Services for load-balancing
- DNS for service-discovery
- Network Policy for segmentation



DNS

Domain Name System

Kubernetes DNS

- Used for service discovery
- (Usually) Implemented CoreDNS
- Multi-replica Deployment
- No App changes needed
- Looks Simple

Kubernetes DNS

- Used for service discovery
- (Usually) Implemented CoreDNS
- Multi-replica Deployment
- No App changes needed
- Looks Simple

The ndots Default

- kubelet injects a bunch of options into /etc/resolv.conf of pods
- `search` will contain something like this:
  `search namespace.svc.cluster.local svc.cluster.local cluster.local eu-west-1.compute.internal`
- `ndots` defaults to 5

→ **Any non-FQDN lookup really results in >=5 lookups (v4+v6)**

DNS Rate Limiting

- Most cloud providers rate limit DNS (e.g. AWS: 1K pps/ENI)
- It's hard to notice
- You've likely been limited, you never knew.

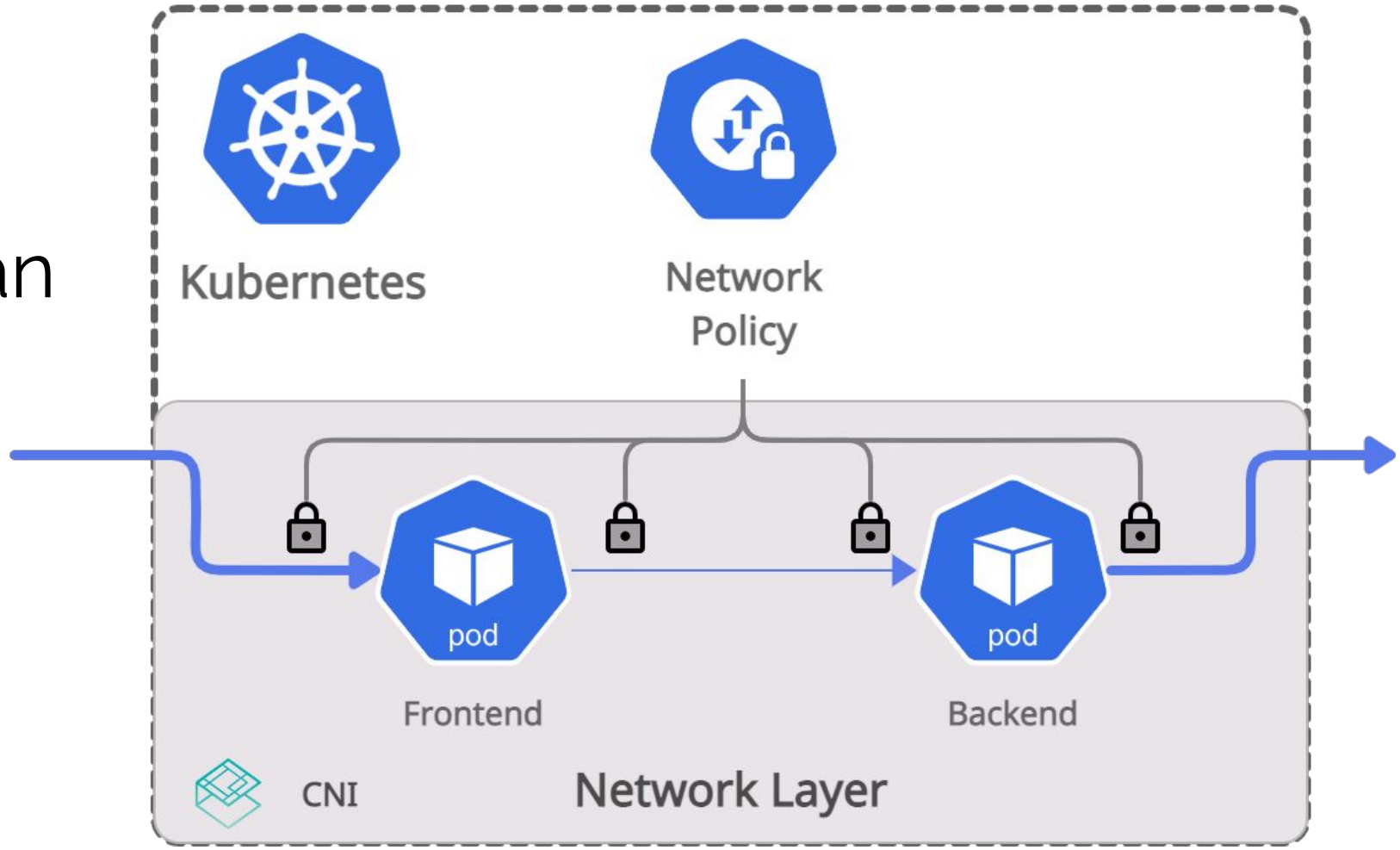
→ **Random connectivity errors**

→ **Often hidden in P99 because it doesn't cover DNS**

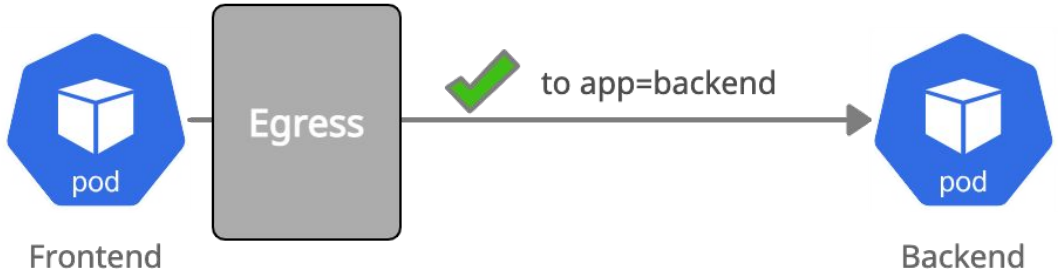
Network Policy

Network Policy

Declares who can talk to whom

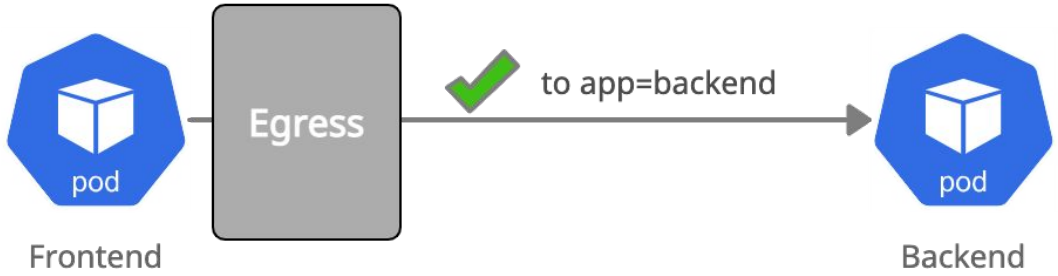


Most Common Fail

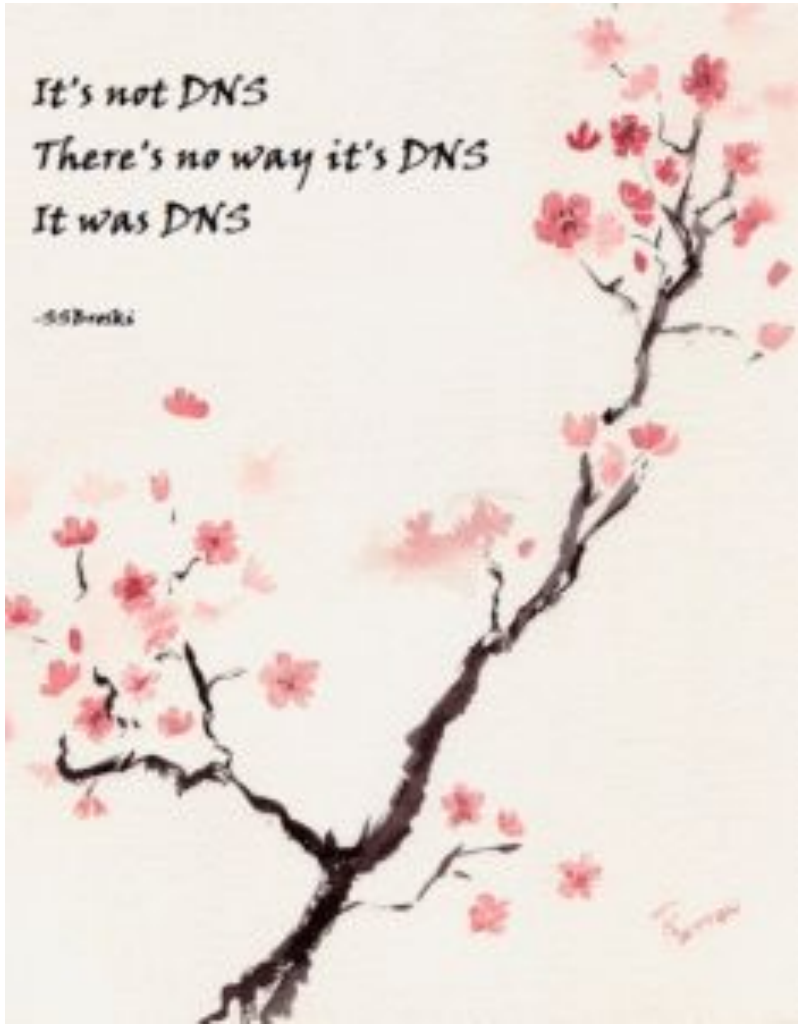



```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: frontend-egress-allow-to-backend
spec:
  podSelector:
    matchLabels:
      app: frontend
  egress:
    - to:
      - podSelector:
          matchLabels:
            app: backend
```

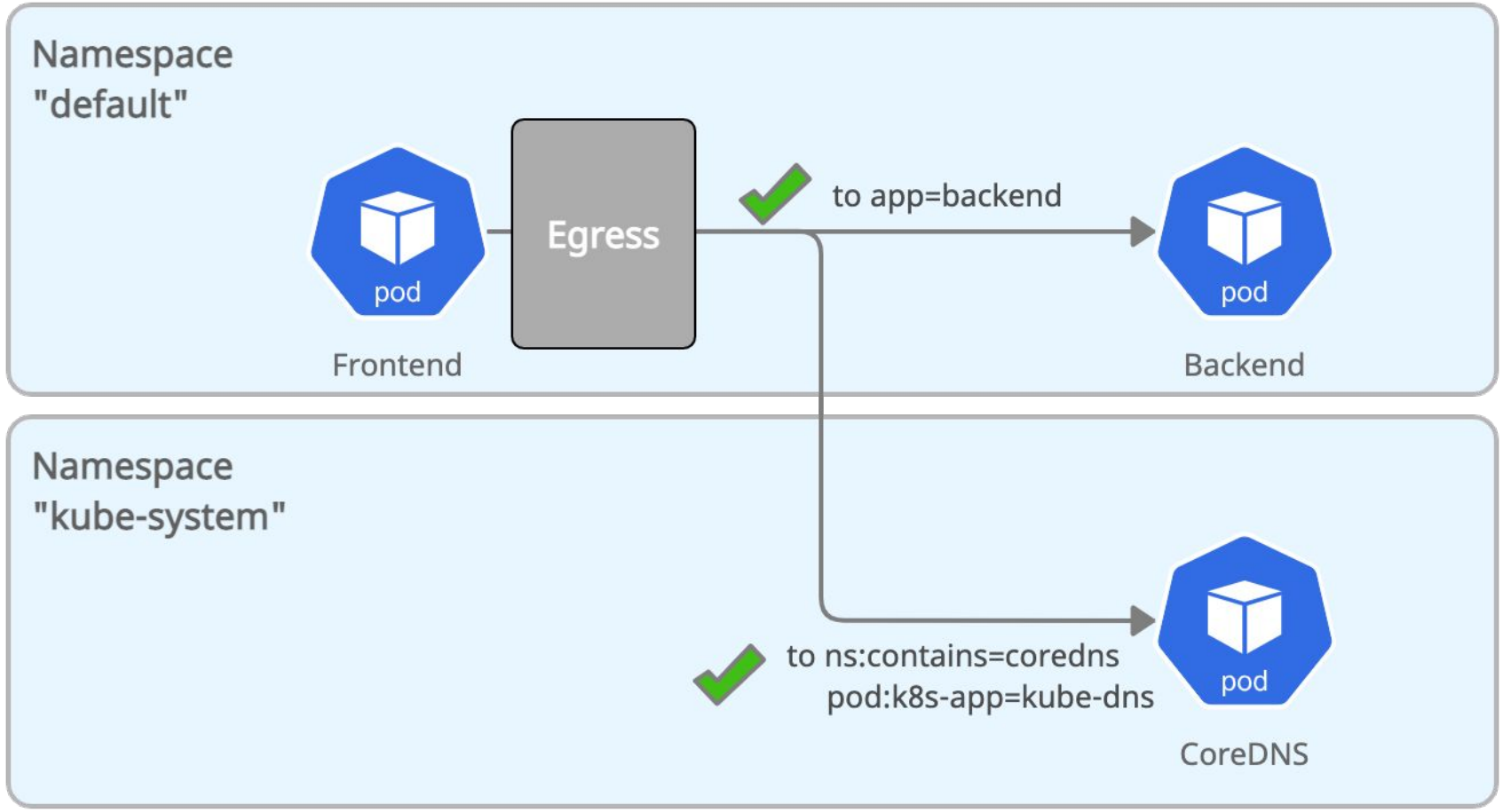
Most Common Fail




```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: frontend-egress-allow-to-backend
spec:
  podSelector:
    matchLabels:
      app: frontend
  egress:
    - to:
      - podSelector:
          matchLabels:
            app: backend
```

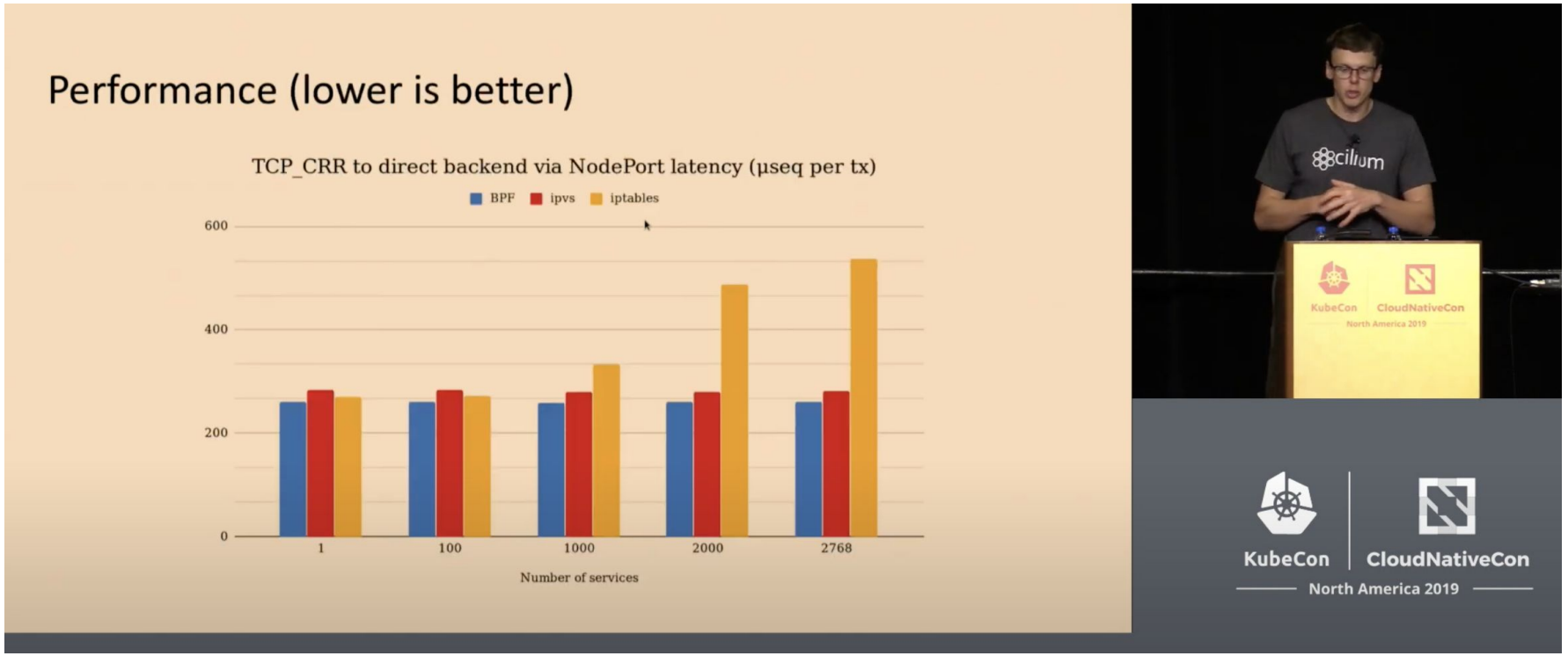


Most Common Fail



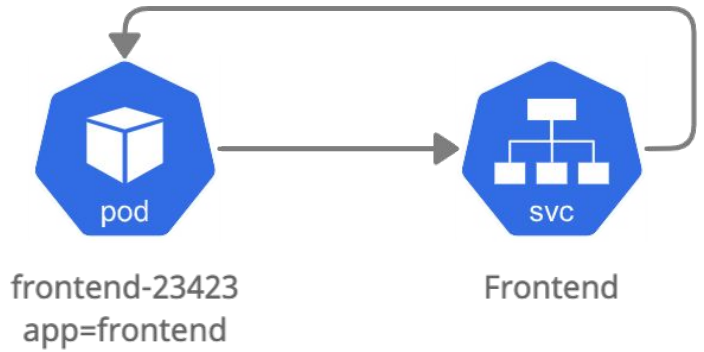
Kubernetes Services & Load-Balancing

Scaling Services



- Default kube-proxy uses iptables
- Latency grows as you grow # services + endpoints

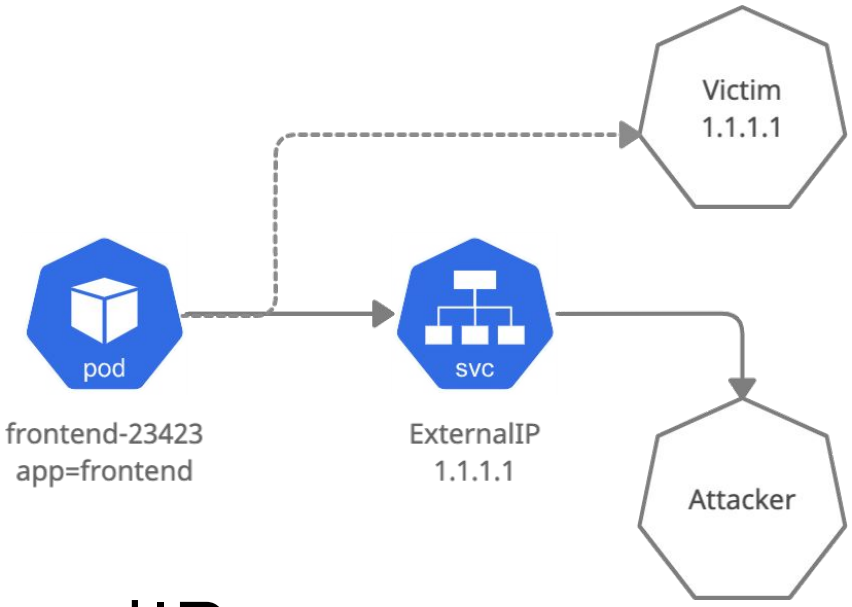
Service Loopback



- Pod talking to itself via Service
- With many CNIs, this will fail silently
- Why: Linux accepts SIP → SIP only on `lo` device

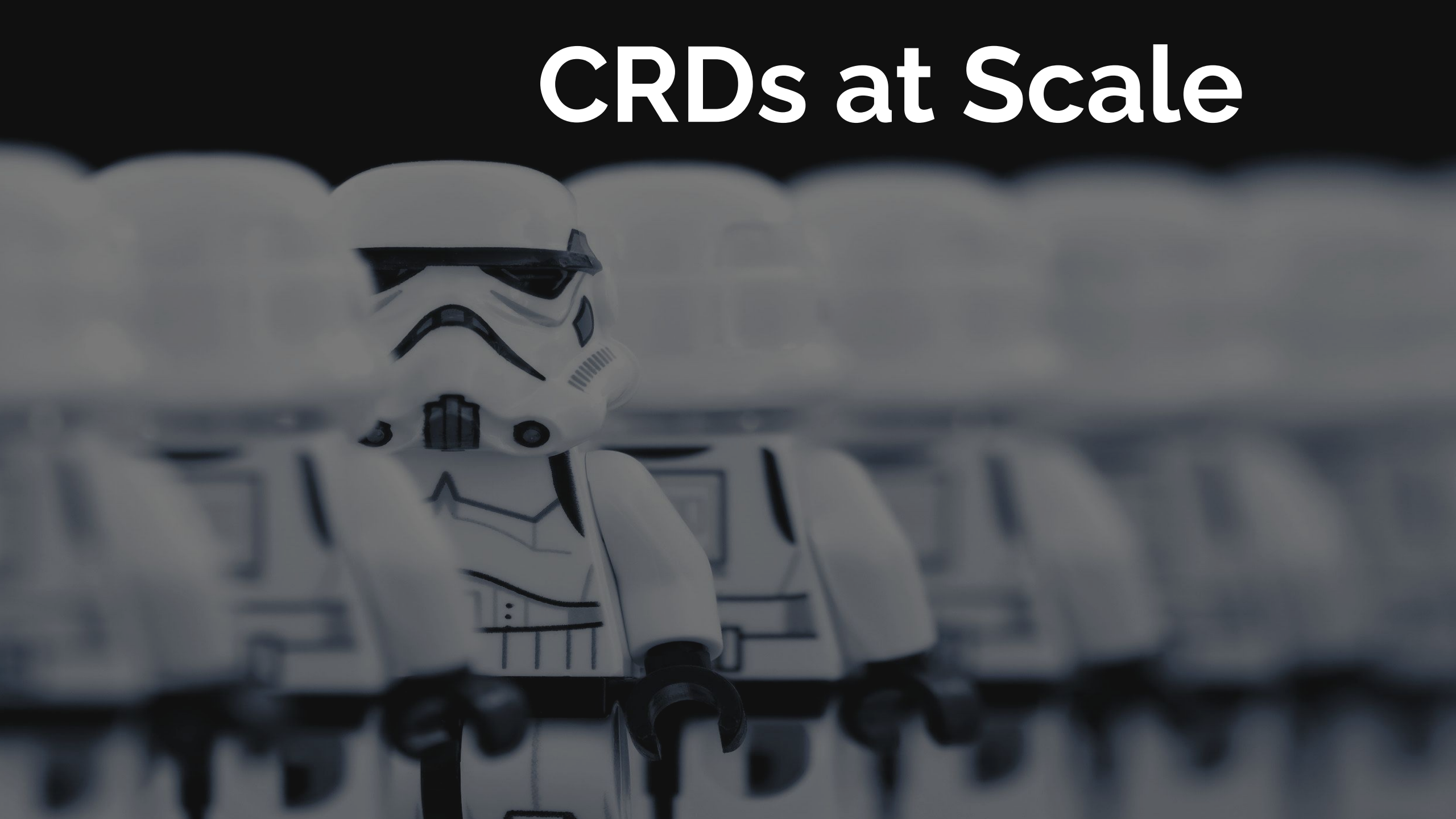
→ **Random connections breaking**

CVE-2020-8554
ExternalIP MITM



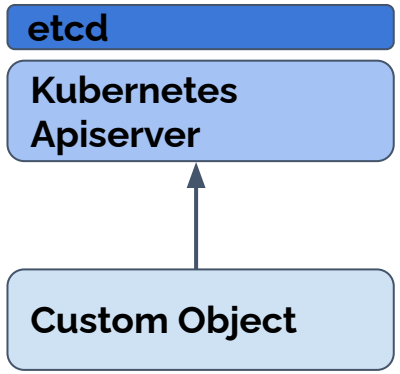
- Redirect any traffic with an ExternalIP
- Quick Demo

CRDs at Scale



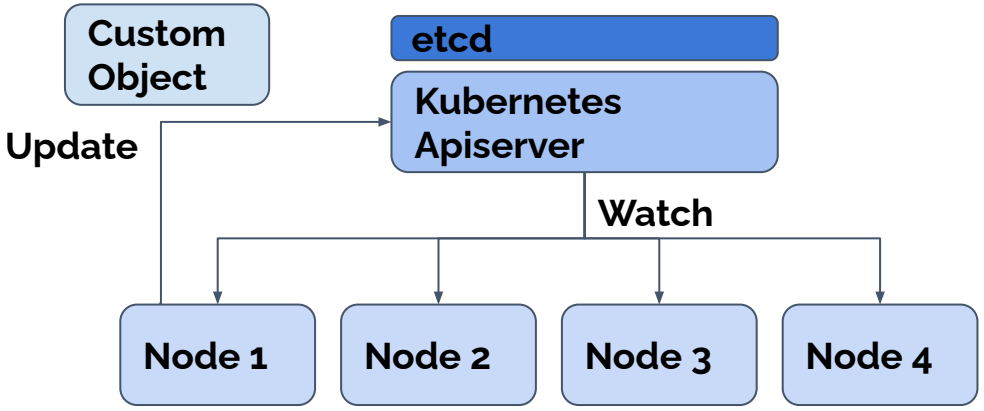
CRD

Custom Resource Definitions



- Custom objects in Kubernetes
- Stored in etcd of the apiserver
- Can be created, watched, deleted, …
- (Mis)used for anything (configuration, state, storage)

CRD Watchers and the Network



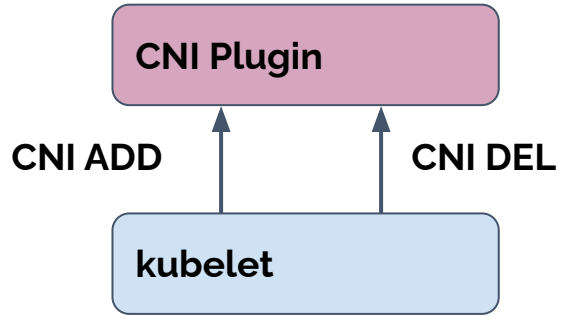
- ~50KB CRD and 5,000 nodes
- CRD is updated by each node every 10min
- 1 update/10min * 5,000 nodes = 8 updates/s
- 8 updates/s * 50KB * 5,000 watchers = **16 GBit/s (or 2GB/s)**

→ **Single apiserver needs to push out 15.6 GBit/s on average**

Perfect Storm: DaemonSet updates CRD on startup & all pods of DaemonSet are restarted simultaneously (within 10s): 977Gbit/s (or 131GB/s)

CNI Configuration Wars

CNI Basics



- kubelet reads CNI configuration from /etc/cni/net.d (or similar)
- CNI Plugins (DaemonSets) drop in their config file (e.g. 05-cilium.conf)
- First file in alphabetical order wins
- Node becomes ready when CNI configuration is found & valid

The Uninstall Leftover Surprise

- CNI plugins typically drop the CNI configuration as they get deployed onto a node (`postStart` or init container)
- CNI plugins can't remove the CNI configuration on `preStart`
  - If they would, fall back to other CNI during restarts
- Thus, CNI plugins leave configuration file behind and only remove the binary

→ **Uninstall a CNI and your networking will be broken**

The Boostrap Race

- User deploys a CNI via DaemonSet with `system-node-critical`
- Another CNI plugin is pre-installed (managed Kubernetes)
- Node is immediately ready due to pre-installed CNI
- DaemonSet races to be scheduled first on new node to replace CNI configuration
- If race is lost, intended CNI plugins misses CNI ADD event

→ **Random new pods have no connectivity**

Bonus: Scheduled != Running: Even if scheduled first, another pod may already get scheduled while DaemonSet writes CNI configuration

The Asymmetric Cleanup

1. CNI configuration X is present
2. Pods get scheduled
3. New CNI configuration file is written
4. Pods are deleted to restart them
5. Old CNI is not invoked with **CNI DEL** when pods are deleted

   → **Routes, interfaces, and other resources are leaked**

   (It will bite you two weeks later)

[3] Lessons Learned


Shiny Objects
are Cool
but keep
it simple

Visibility Matters

Connectivity is not enough,
Visibility is what matters on day 2+

Get Yourself some Superpowers

eBPF

Learn more: ebpf.io

Thank You

Contact Thomas:
@tgraf__

Learn about Cilium:
cilium.io
github.com/cilium/cilium