

Secure and monitor your service mesh
(Praparn Lueangphoonaip)



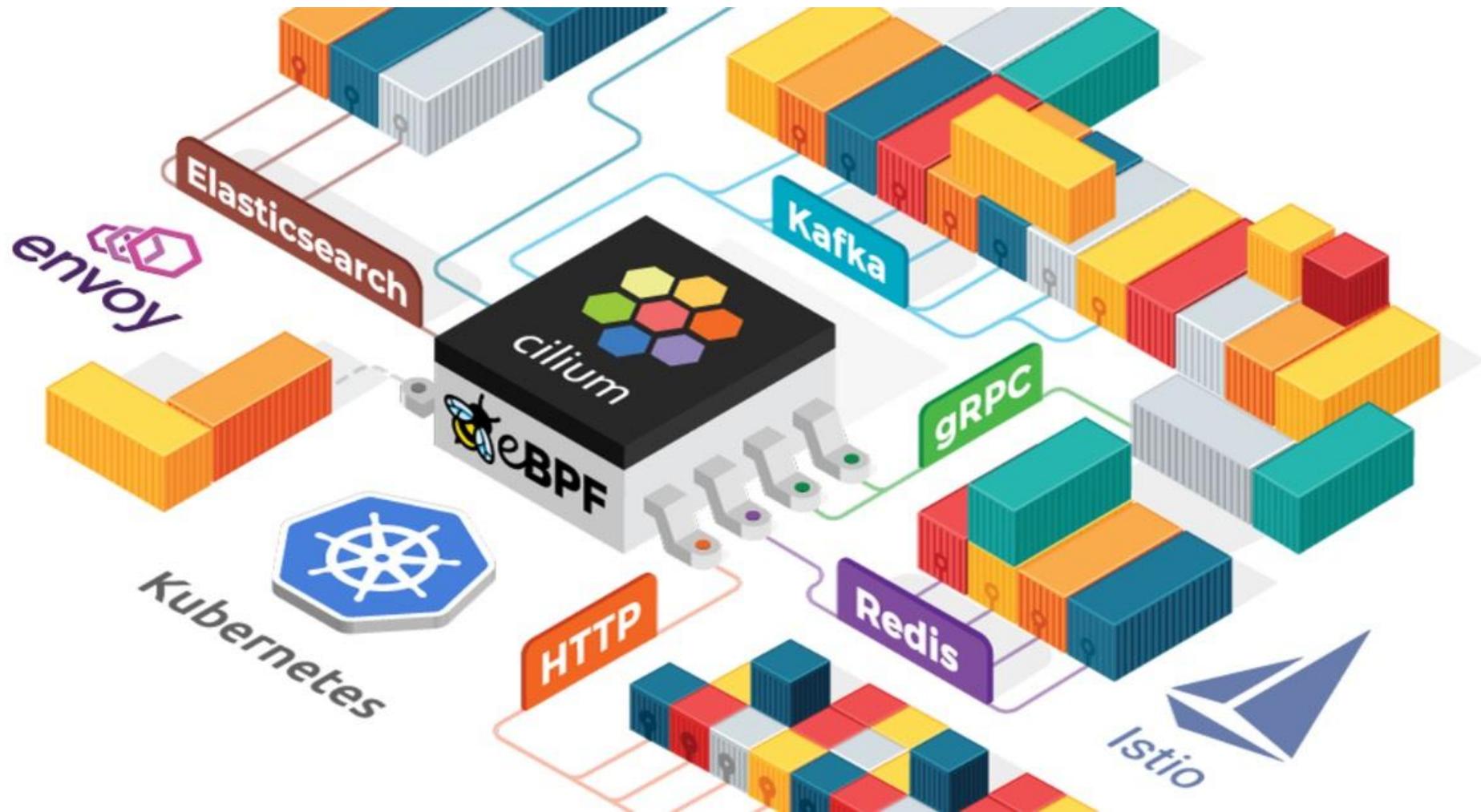
kubernetes
by Google

Agenda

- Why microservice connectivity is matter ?
 - Secure and Visibility is the Key
 - What the solution (Istio ?) and Problem
- Introduction to eBPF and Cilium
 - What is and Why eBPF?
 - Cilium on Kubernetes as data plane
 - Networking
 - Observability
 - Security
 - No kube-proxy with Cilium !!!
 - Hubble observability for all
- Demo session



Why microservice connectivity is matter ?



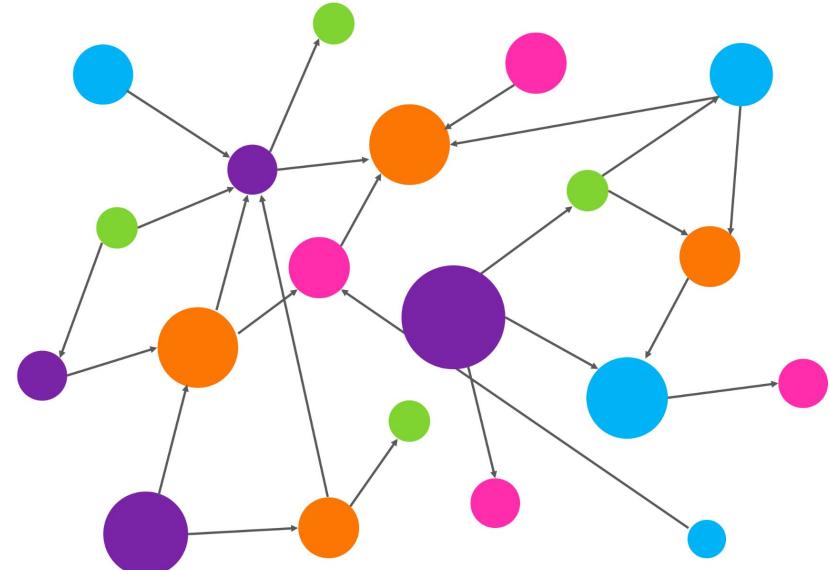
Kubernetes for enterprise business



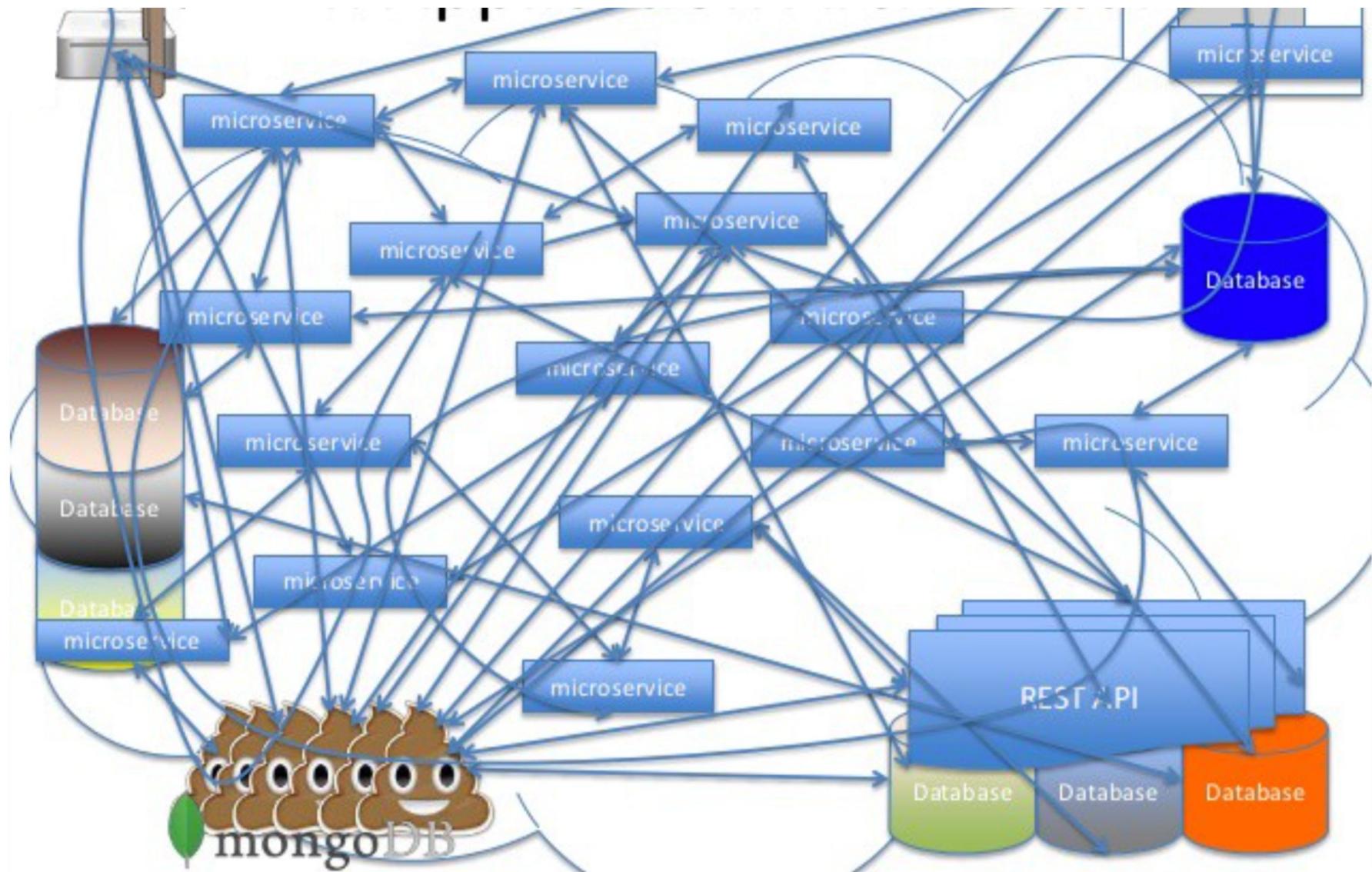
kubernetes
by Google

Why microservice connectivity is matter ?

- Basically when we design application in microservice. We also have multiple microservice...
- Each microservice/pods will handle connection from...
 - Client (via front-end)
 - BFF (Back from Front)
 - 3rd party call
 - Other microservice
 - Stranger connection withc
 - etc.



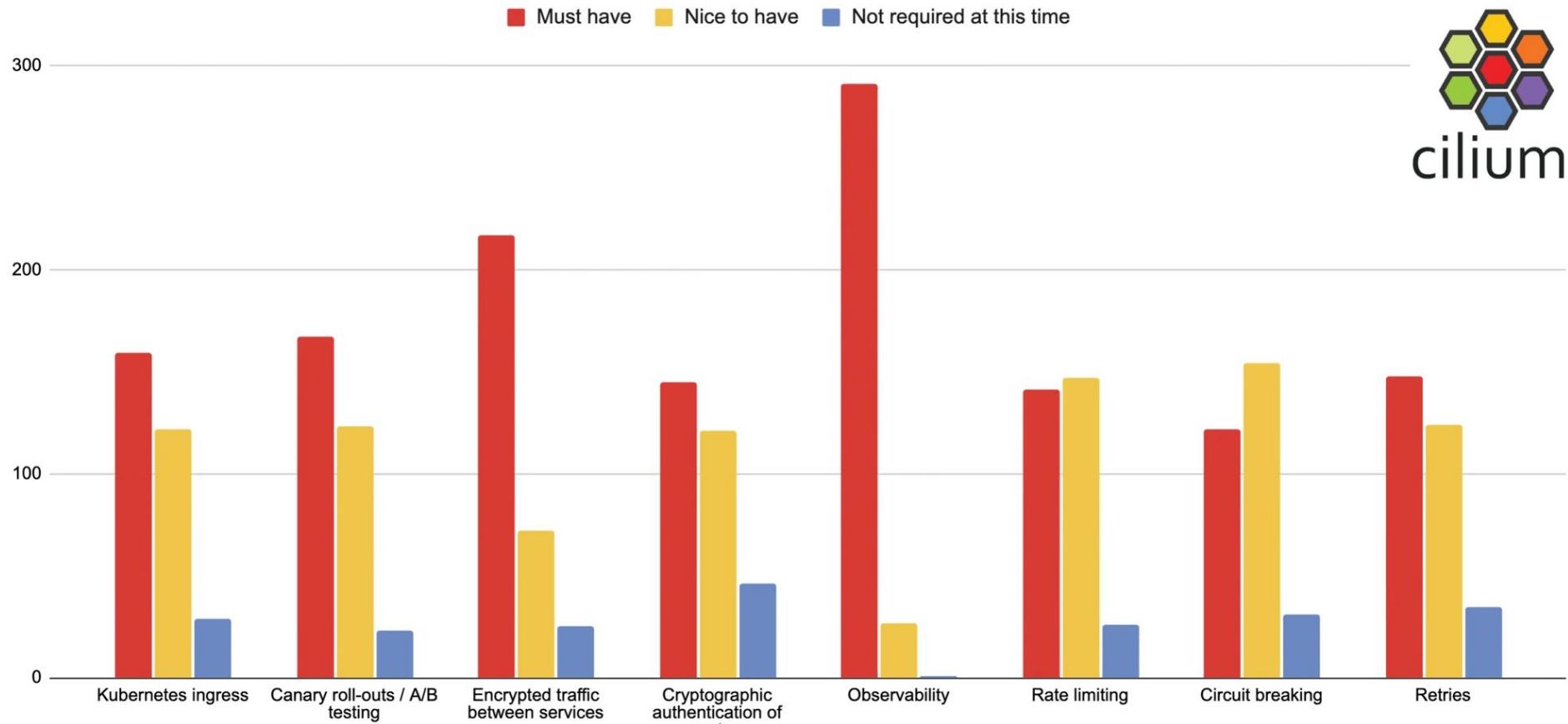
Why microservice connectivity is matter ?



kubernetes
by Google

Why microservice connectivity is matter ?

What features of a Service Mesh interest you most?



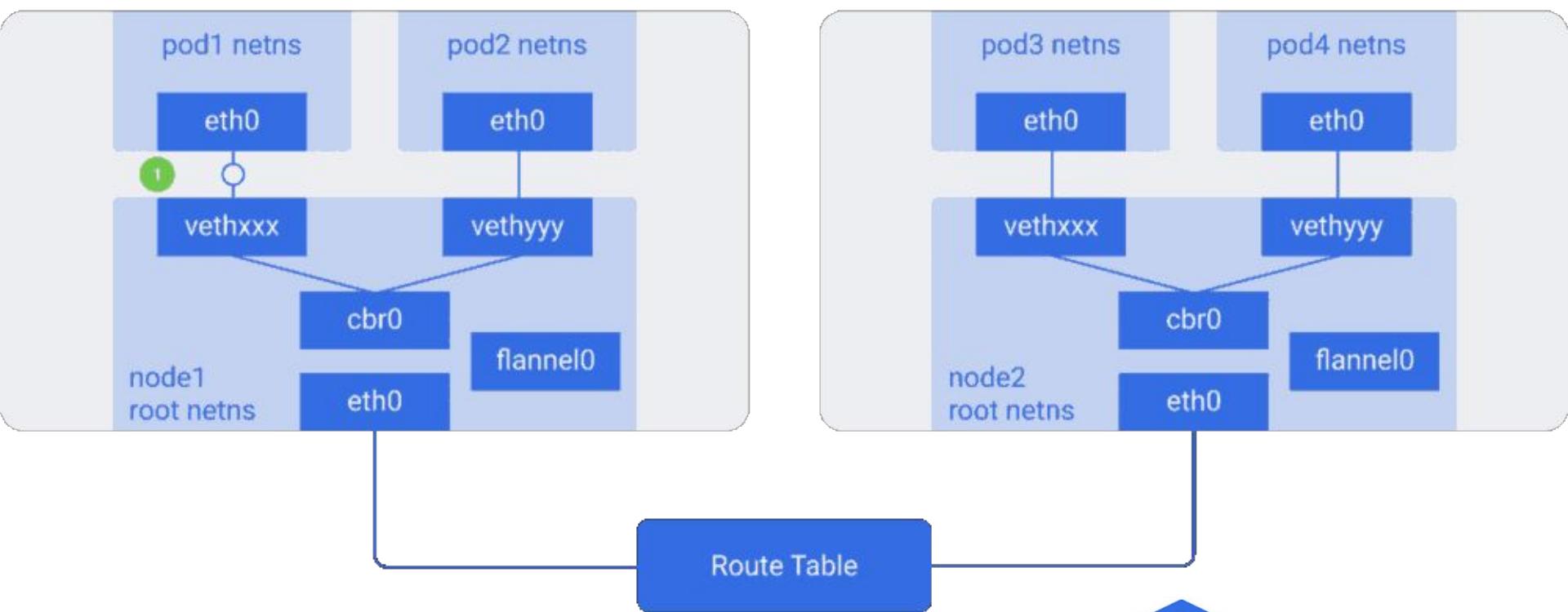
Ref: <https://cilium.io/blog/2022/01/25/cilium-service-mesh-beta-feedback>



kubernetes
by Google

Secure and Visibility is Key

- When we had been deploy our microservice on Kubernetes. By default all microservice can accessible to any microservice that they know the service name !!!



Secure and Visibility is Key

- And the exactly happen is... We never have chance to see that is going on
 - Who try to connect our application (pods)
 - Our pods is try to reach other pods or not ? (If there got compromised ?)
 - Any connection is normal
 - Which source connection ?
 - How frequency ?
 - Any connection is abnormal
 - Which source try to connect ?
 - Is it success ?
 - Any try from unknown source ?

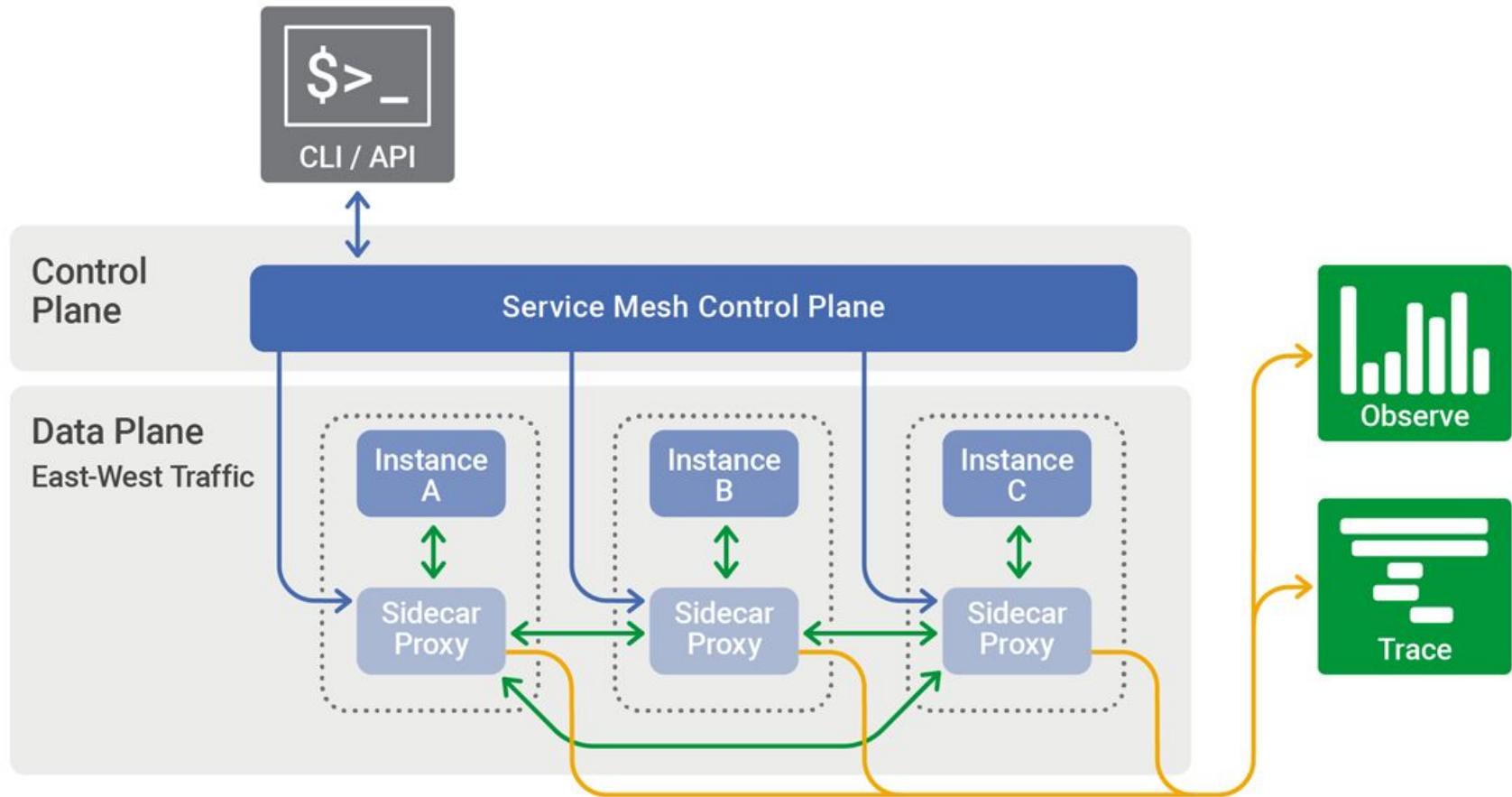


What the solution? (Istio ?)

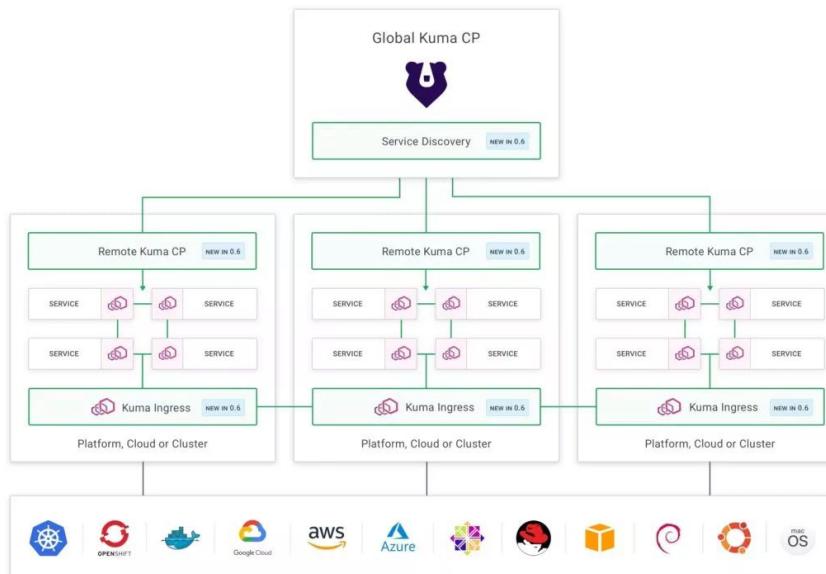
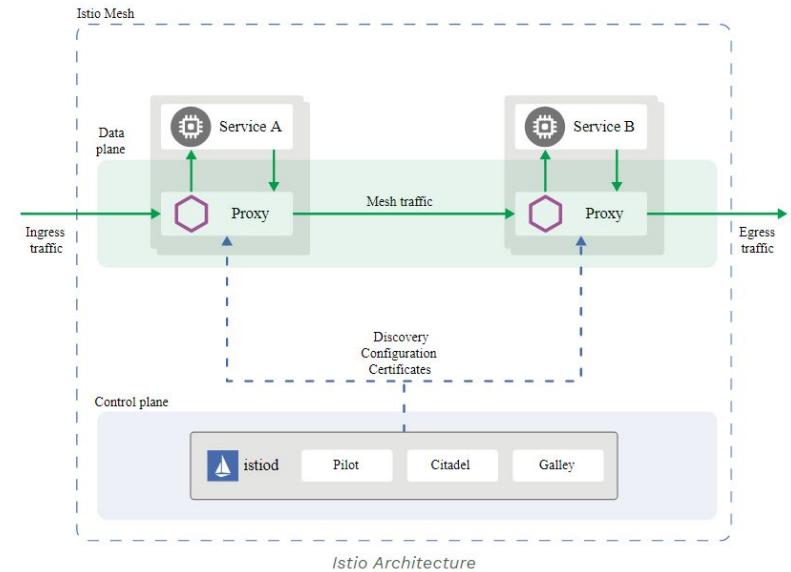
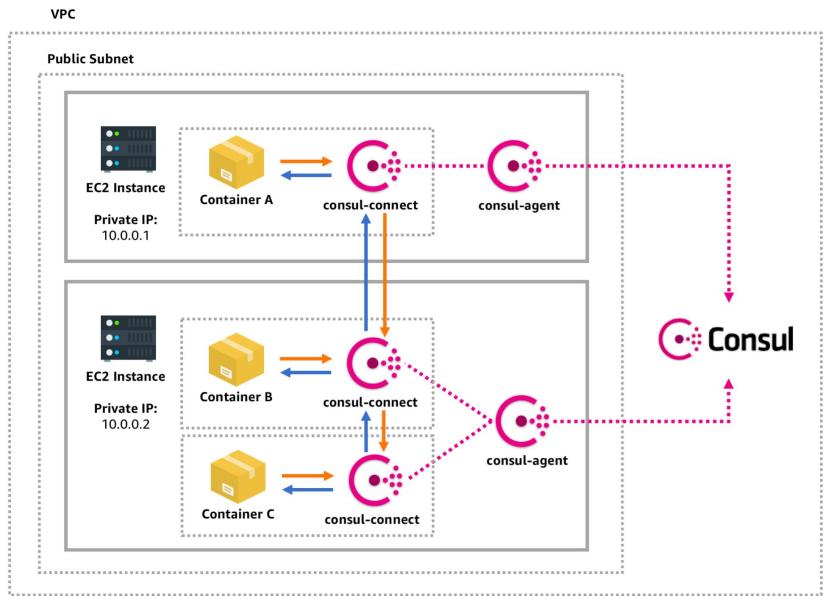
- **Yes (1)**, Service Mesh. Believe that 1st, 2nd solution bring in the table.
- Istio, Linkerd, Kuma, Console, Dynatrace etc or In-house with similar concept
 - Pass all connection in pods via “sidecar proxy” for observe/trace all connection and manageable service mesh or send all connect to service mesh control plane
 - All connection will pass on service mesh control plane.
 - So service mesh management can generate flow-map and control from this concept



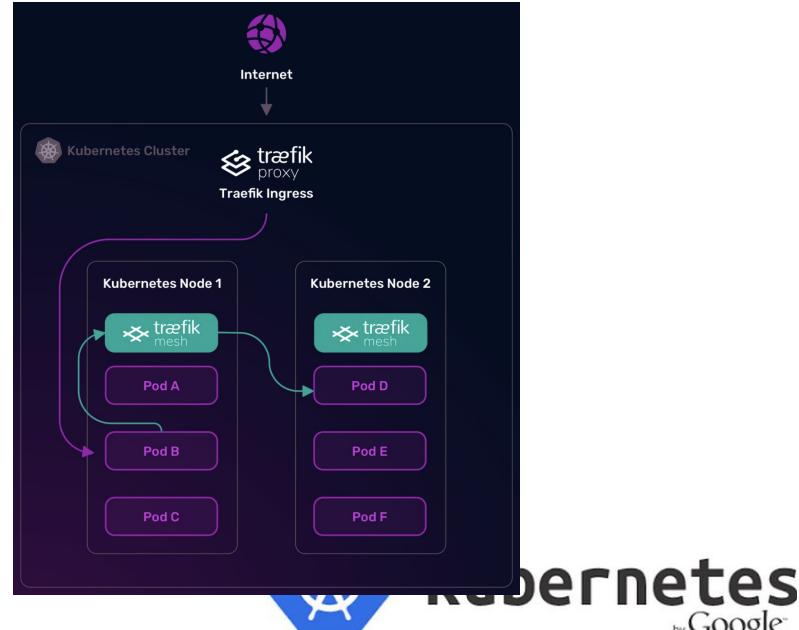
What the solution? (Istio ?)



What the solution? (Istio ?)

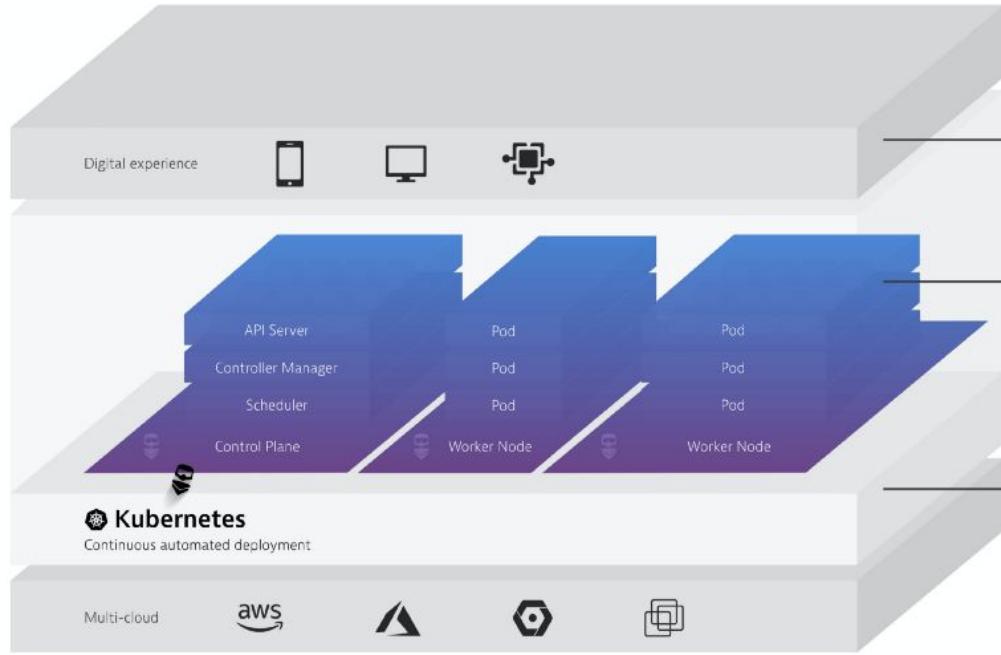


Kubernetes for enterprise business



What the solution? (Istio ?)

OneAgent deploys automatically via Operator to all layers and technologies in your environment



Monitor, analyze and optimize every digital interaction

Real-time auto discovery through OneAgent injection into containers without code or image changes

Automatic and continuous deployment of Dynatrace OneAgent to all cluster nodes

Full integration with all major cloud platforms



What the solution? (Istio ?)

The image displays three screenshots of service mesh tools:

- Kiali:** A screenshot of the Kiali interface showing a service graph for the bookinfo namespace. It visualizes the flow of requests between services like istio-ingressgateway, productpage, reviews, ratings, and mongodb.
- Linkerd:** A screenshot of the Linkerd dashboard for the deployment/product-gateway-api-deploy. It shows metrics for SR (Success Rate), RPS (Requests Per Second), and P99 (99th Percentile Response Time) for three services: deploy/price-api-deploy, deploy/product-gateway-api-deploy, and deploy/product-api-deploy.
- Jaeger:** A screenshot of the Jaeger Service Flow interface for nginxForMicroservices. It traces a request path from nginxForMicroservices through EasyTravel(BackendWebserver), JourneyService, and easyTravel-Business, providing detailed response time contributions at each step.

Kubernetes for enterprise business



kubernetes
by Google

What the solution? (Istio ?)

- From this concept every microservice/pods need to have sidecar proxy ?
- **So what is the problem ?**
 - Sidecar proxy/Control Plane (The other hand: another proxy for our request in/out) at principle will slightly impact performance in several way
 - Complicate: More layer is make complicate for troubleshooting and harder to investigate
 - Resource overhead: With sidecar and service mesh control plane implement. It will increase more on resource consumption (CPU/Memory/IO) for their work for service mesh. This mean more cost in project (Aka: Pay more for same result)



What the solution? (Istio ?)



Dave Strebler
@dave_strebler

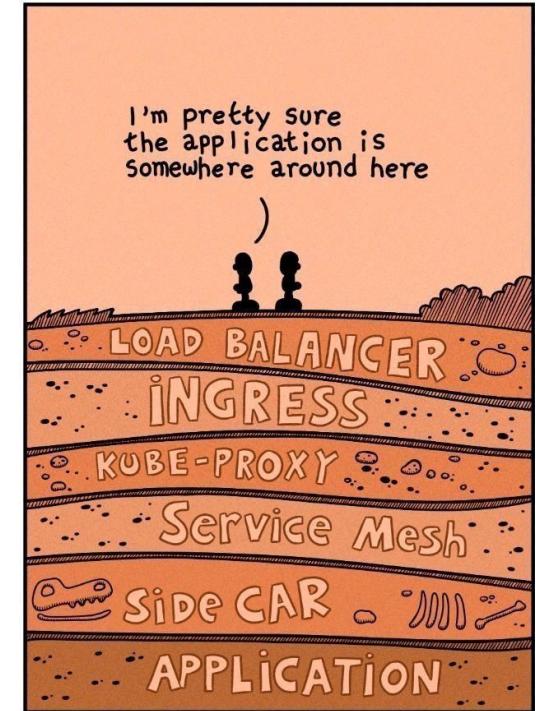
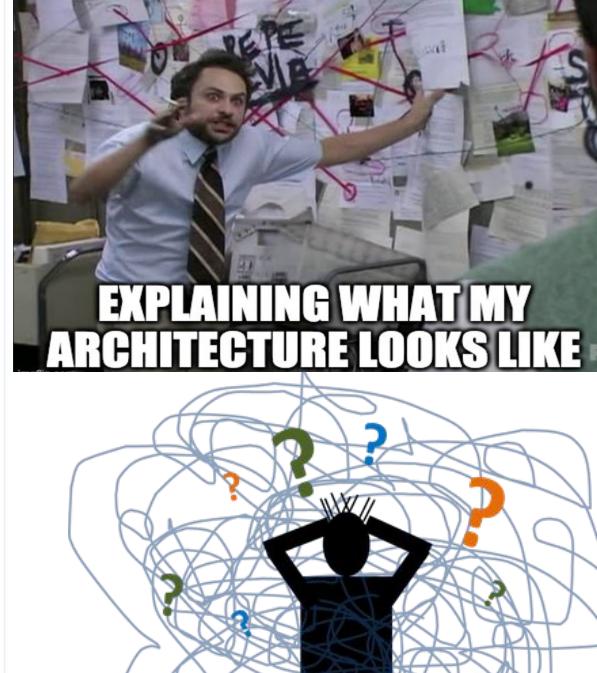
"Finally got Istio into production"



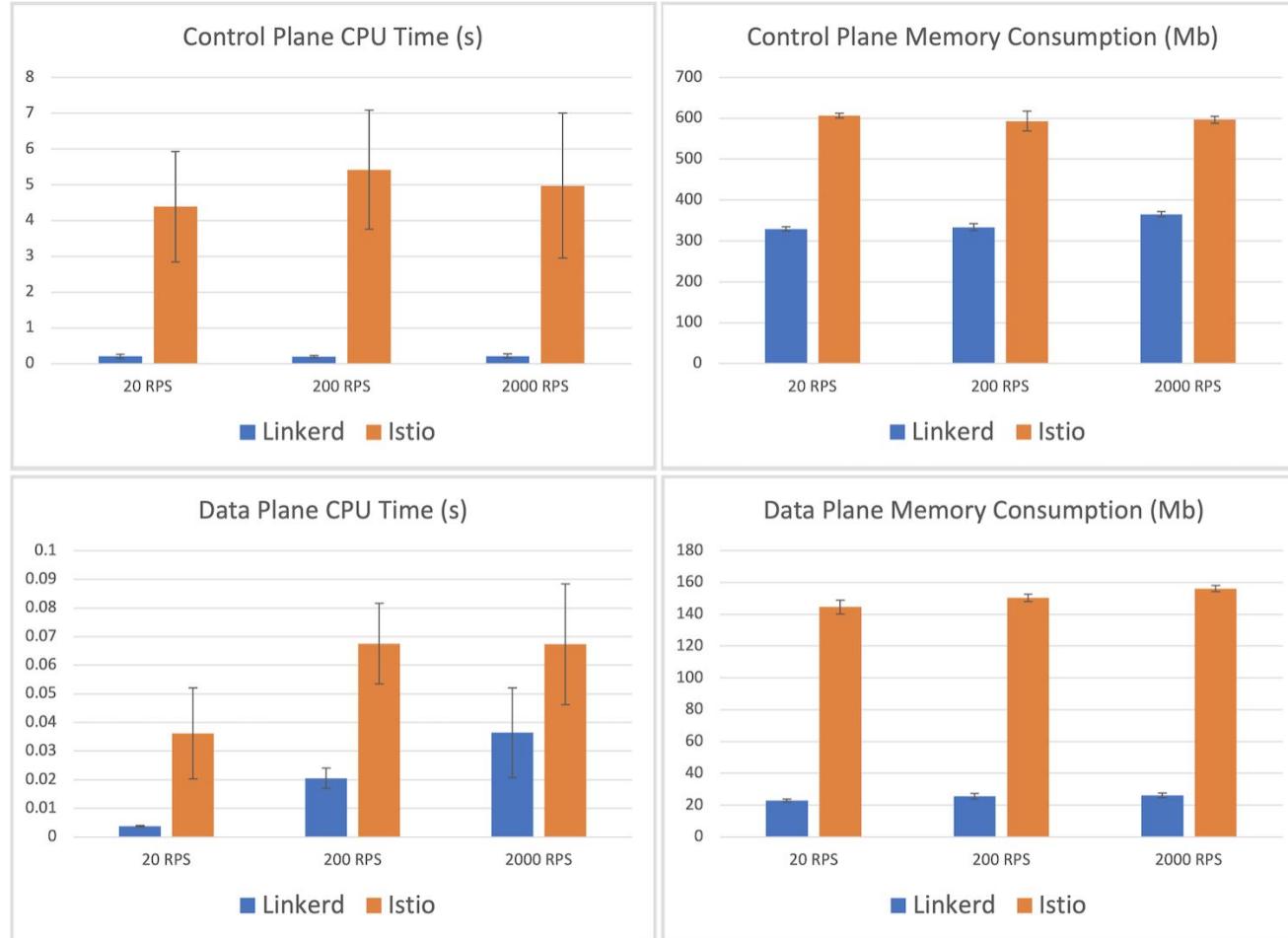
6:21 PM · Sep 17, 2019 from Apple Valley, MN · Tweetbot for iOS



SPRING-BOOT RUNNING UNDER ISTIO
SIDECAR PROXY THROW HTTP 403 ERROR



What the solution? (Istio ?)



Ref:<https://linkerd.io/2021/11/29/linkerd-vs-istio-benchmarks-2021/>

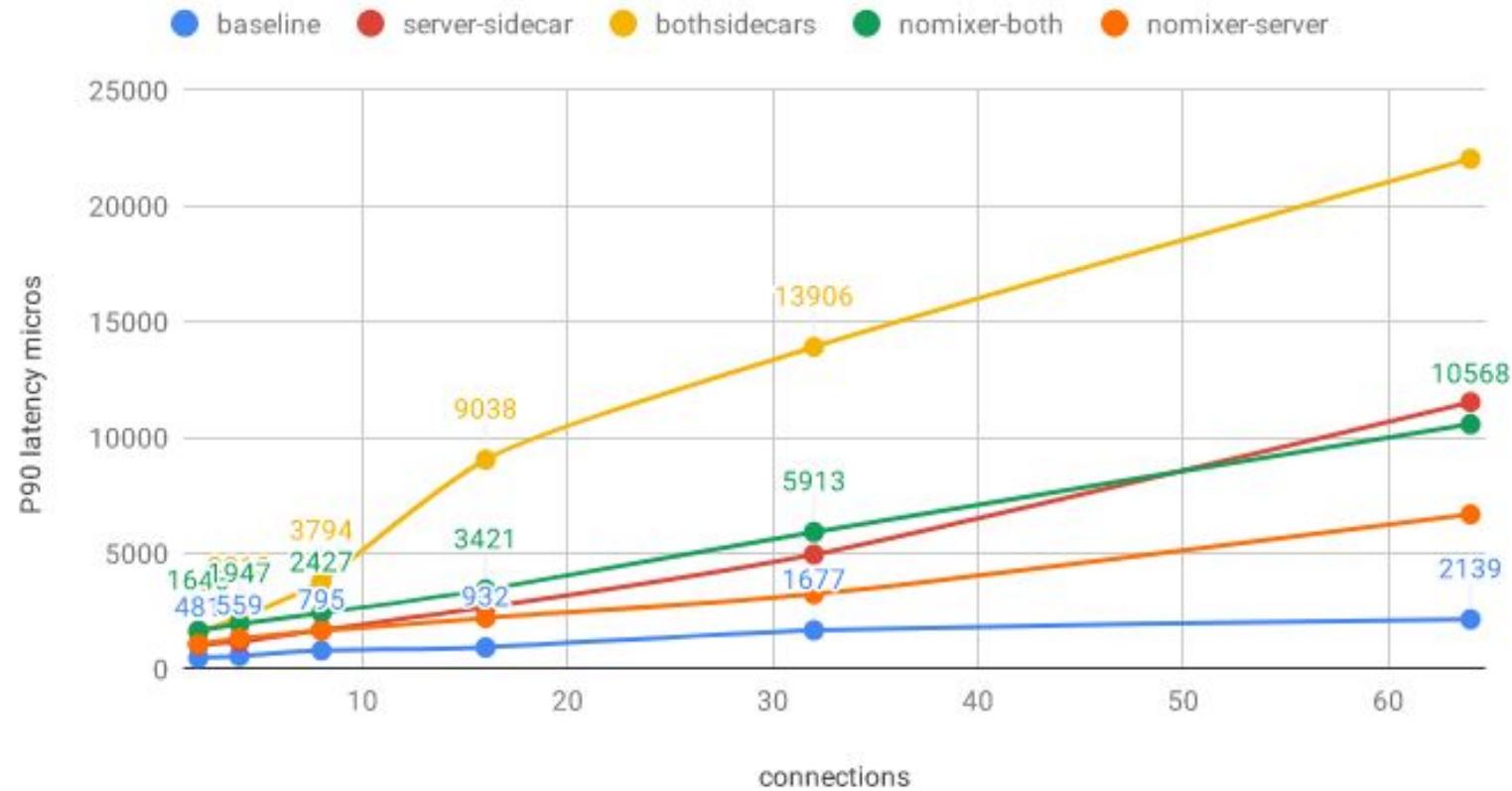
What the solution? (Istio ?)

- **So what is the problem ?**
 - Increase latency: With fact that you just add more “hop” in connection. This will slightly increase network latency more and effect to application (Ex: 400 ms x 10 hop call ~ 4000 ms. What if you have 5 call this api per page of application ?)
 - Slow performance: In concept all service mesh/control plane are running in “user space” is this cannot be avoid this or make it faster like non-sidecar proxy T T



What the solution? (Istio ?)

Latency at 1000 rps



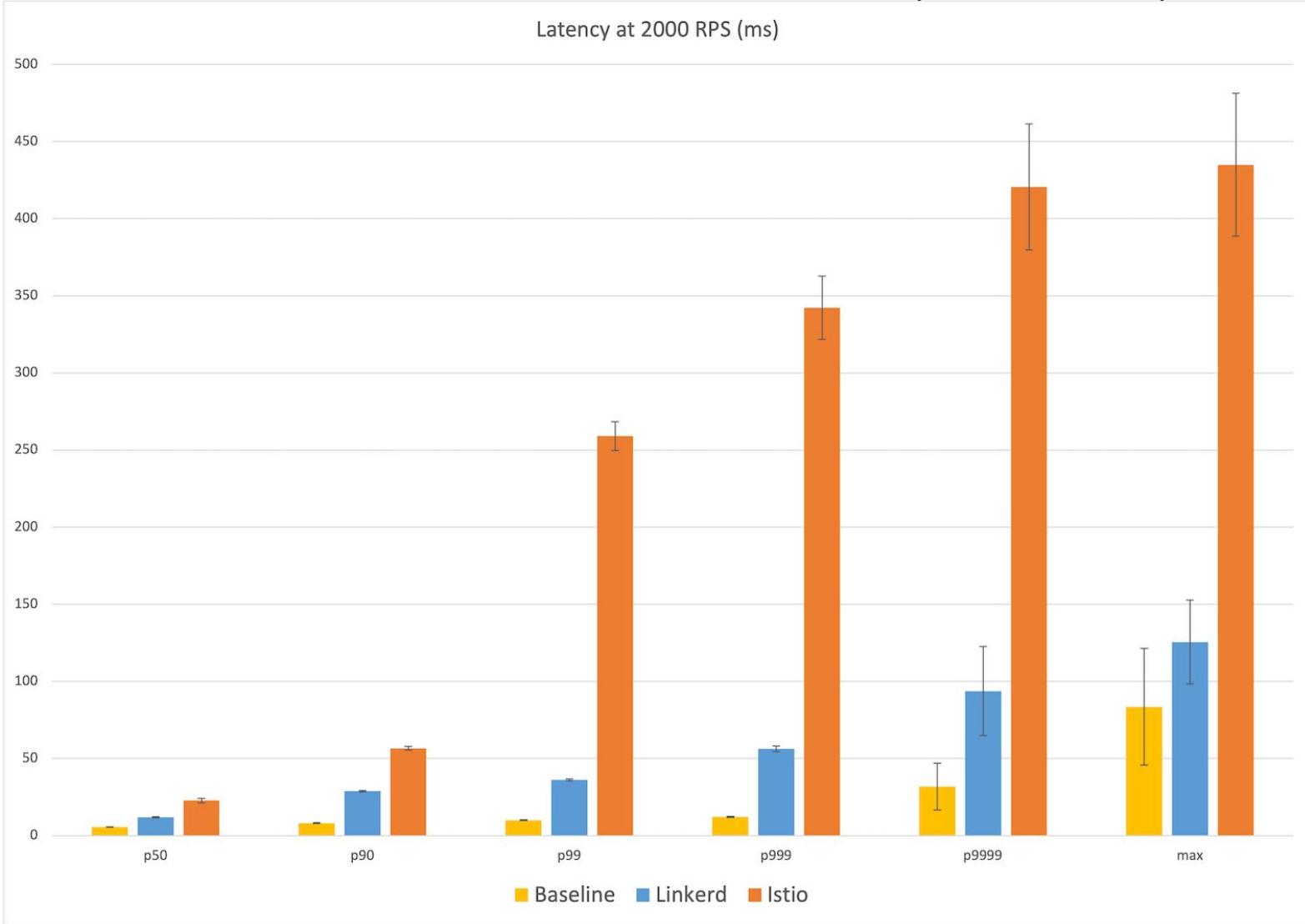
Ref:<https://istio.io/v1.2/docs/concepts/performance-and-scalability/>

Kubernetes for enterprise business



kubernetes
by Google

What the solution? (Istio ?)



<https://linkerd.io/2021/11/29/linkerd-vs-istio-benchmarks-2021/>

Kubernetes for enterprise business



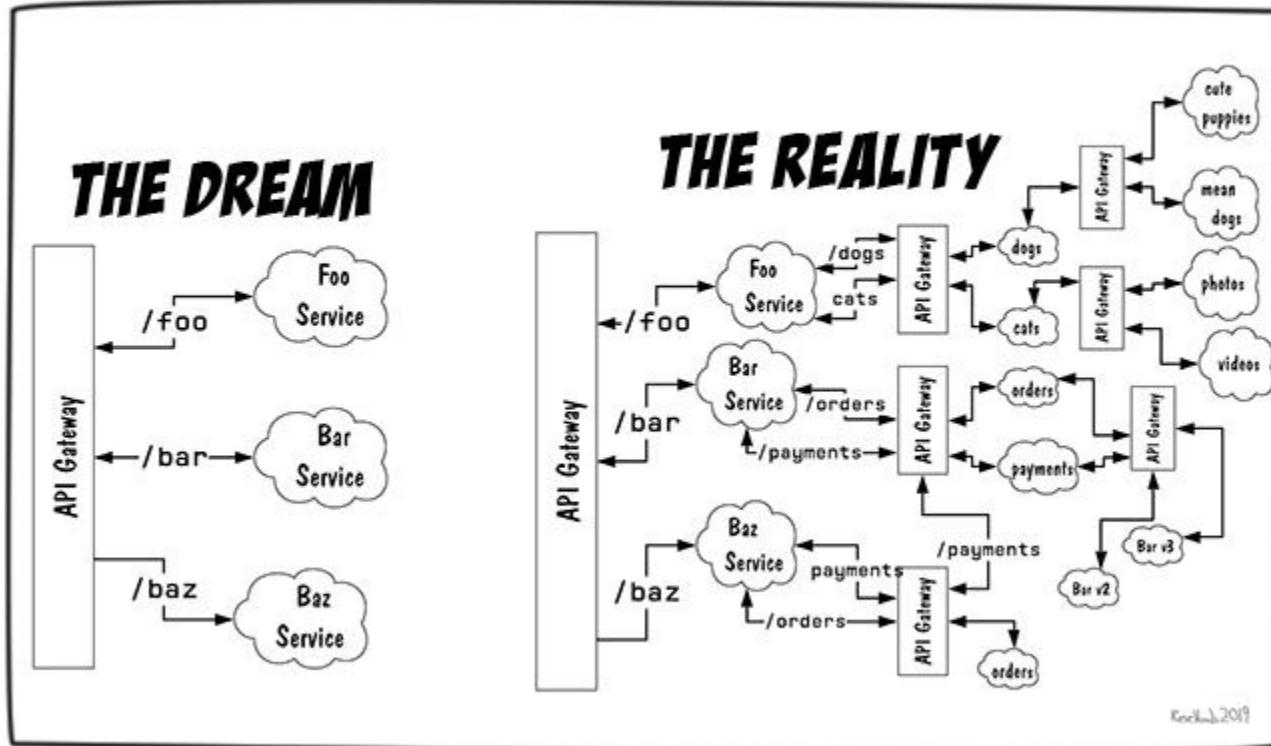
kubernetes
by Google

What the solution? (Istio ?)

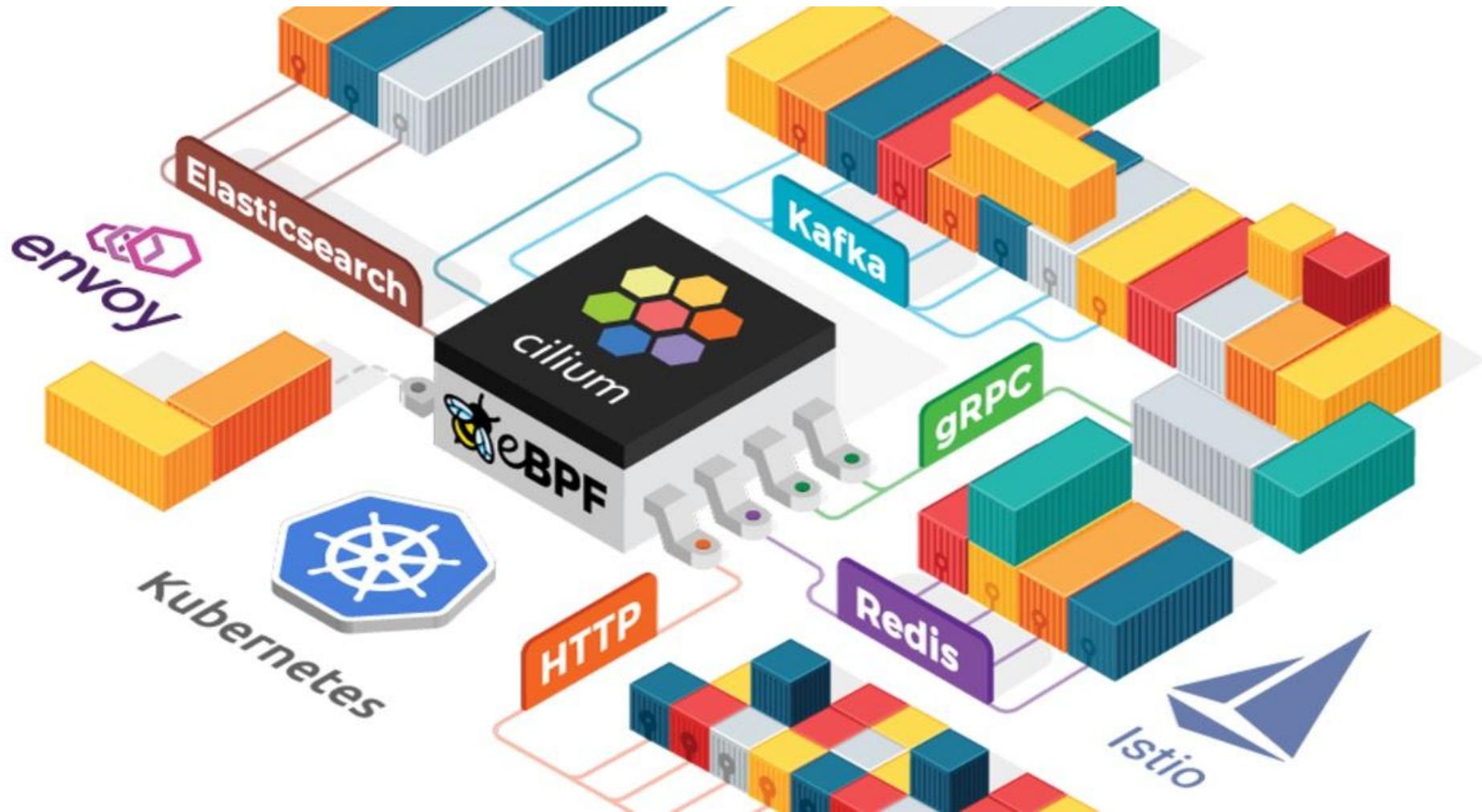
- **Yes (2)**, API Gateway ?
- Idea is enforce all microservice will communicate via API Gateway for secure and manage as centralize. This good idea in term of operation and make benefit for security but ...
 - How can we know all microservice is connect to api gateway ?
 - Overload in API gateway will make it need more resource overhead
 - So service mesh management can generate flow-map and control from this concept



What the solution? (Istio ?)



Introduction to eBPF and Cilium



Kubernetes for enterprise business



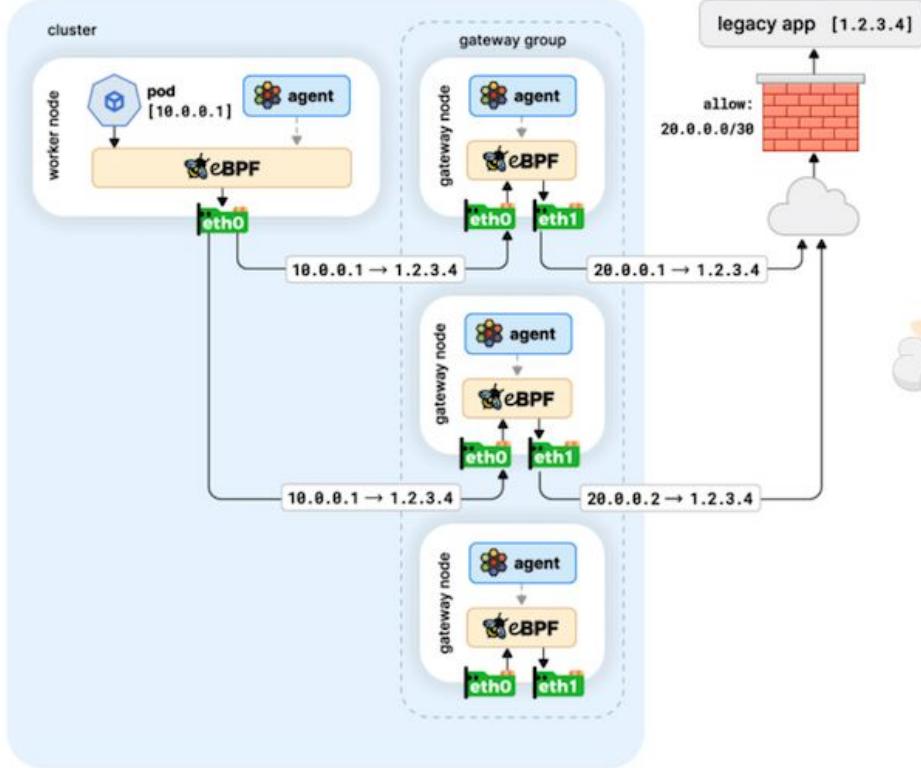
kubernetes
by Google

Introduction to eBPF and Cilium

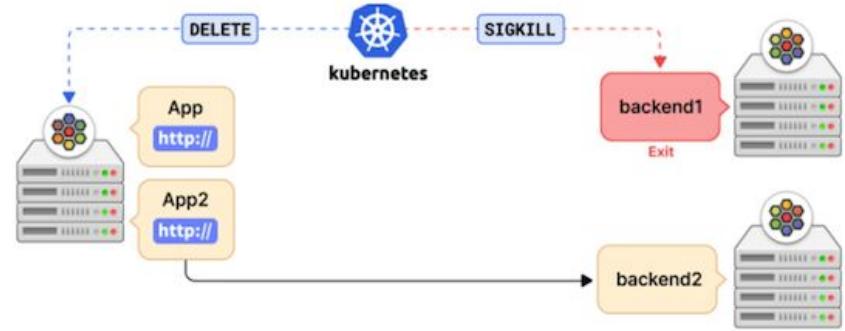
- Many problem of sidecar proxy and service mesh today is on world scale problem !!!
- Many year since istio launch and the performance issue is hard to avoid from architecture design
- **Since Oct,2021.** Cilium join CNCF as incubating project and apply “next generation” of service mesh management with eBPF (since version 1.10 and above)
- This concept is eliminate all sidecar proxy that we familiar with new technique to operate directly in “kernel space” that more effective than “user space”



Introduction to eBPF and Cilium



What's new in 1.11?

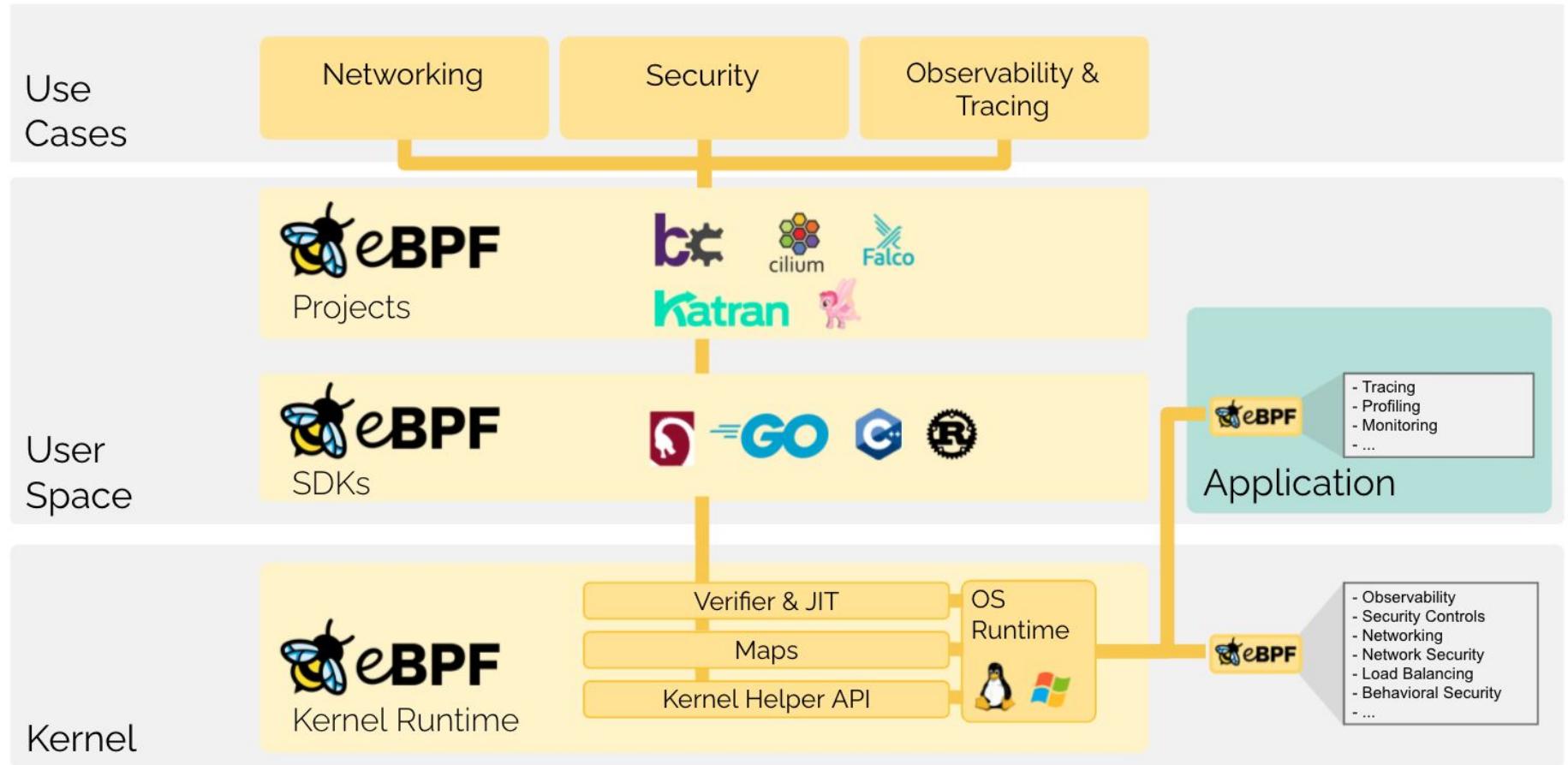


What is and Why eBPF ?

- eBPF is one technology that operate in linux kernel by kernel sandbox program feature
- This make us to run some program without need to upgrade kernel or install module
- eBPF provide SDK for developer can run eBPF problem for additional feature on kernel space
- When we run our application via sdk. eBPF runtime will be verify and JIT compile before send to kernel via “Kernel helper API”
- With this process. Running program via operating system then guarantees safety and execution efficiency as if natively compiled



What is and Why eBPF ?



Ref: <https://ebpf.io/>

Kubernetes for enterprise business

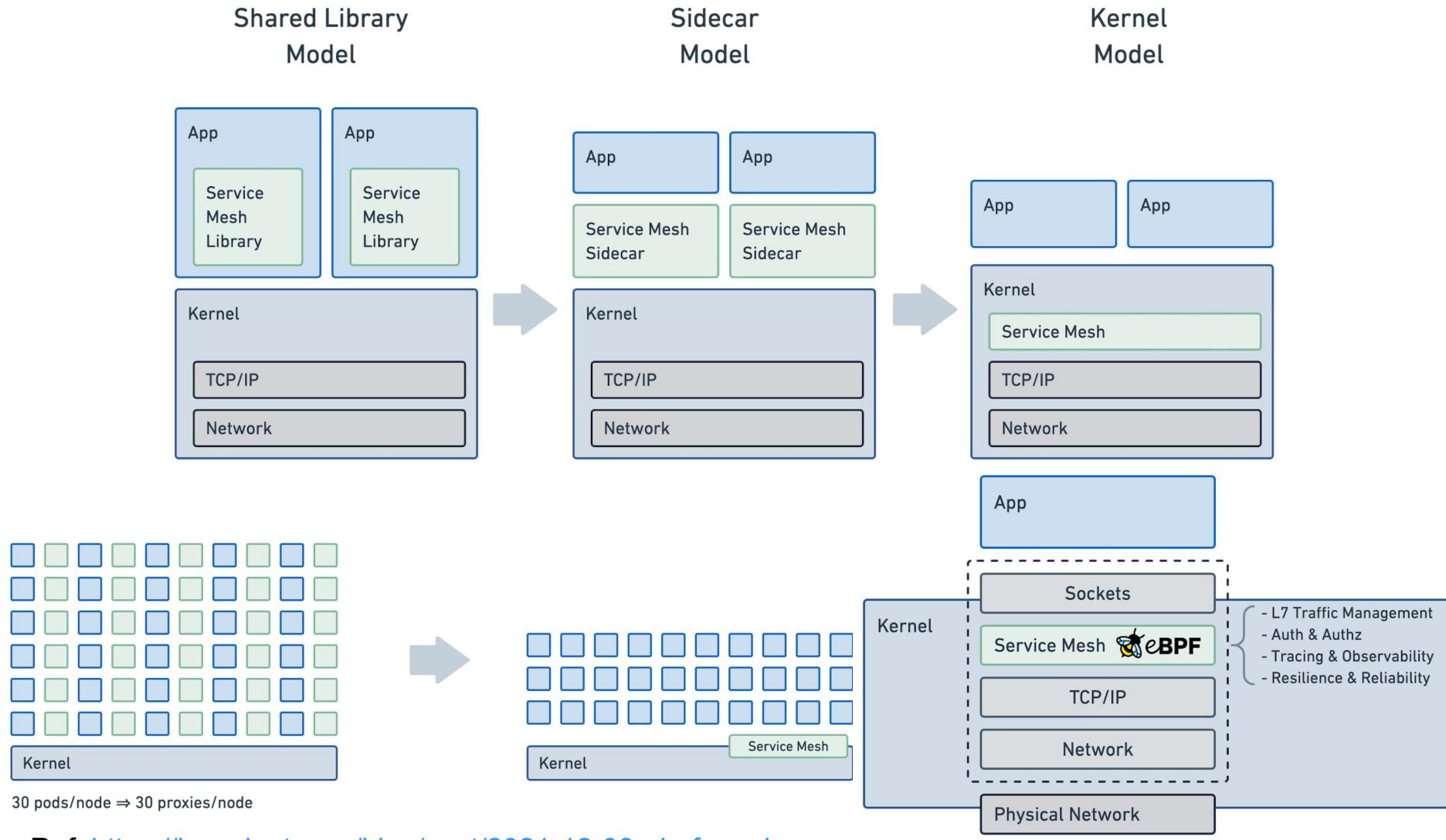


What is and Why eBPF ?

- From capability of eBPF. Cilium find the new way for enhancement the service mesh for make visibility and manage connectivity within “kernel space”. That not make overhead like traditional service mesh on “user space”.
- With common process that all traffic need to service via kernel process. So in new way of Cilium use with eBPF with embed service mesh on eBPF and running there without any latency from sidecar proxy. Only pure kube-proxy operation
- So this why eBPF is answer with native and highly efficient service mesh implementation



What is and Why eBPF ?



Ref: <https://isovalent.com/blog/post/2021-12-08-ebpf-servicemesh>

Kubernetes for enterprise business



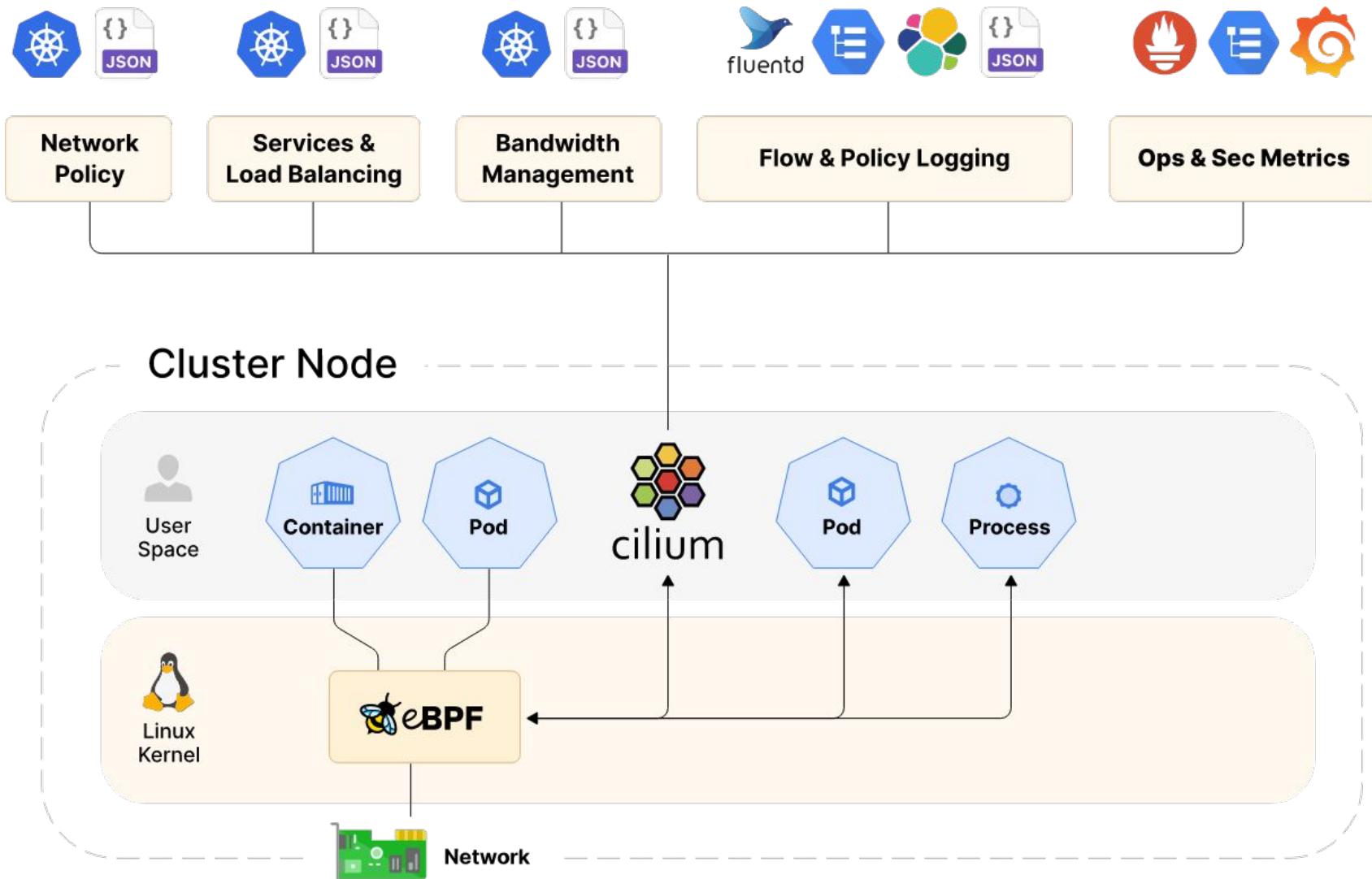
kubernetes
by Google

Cilium on Kubernetes Dataplane

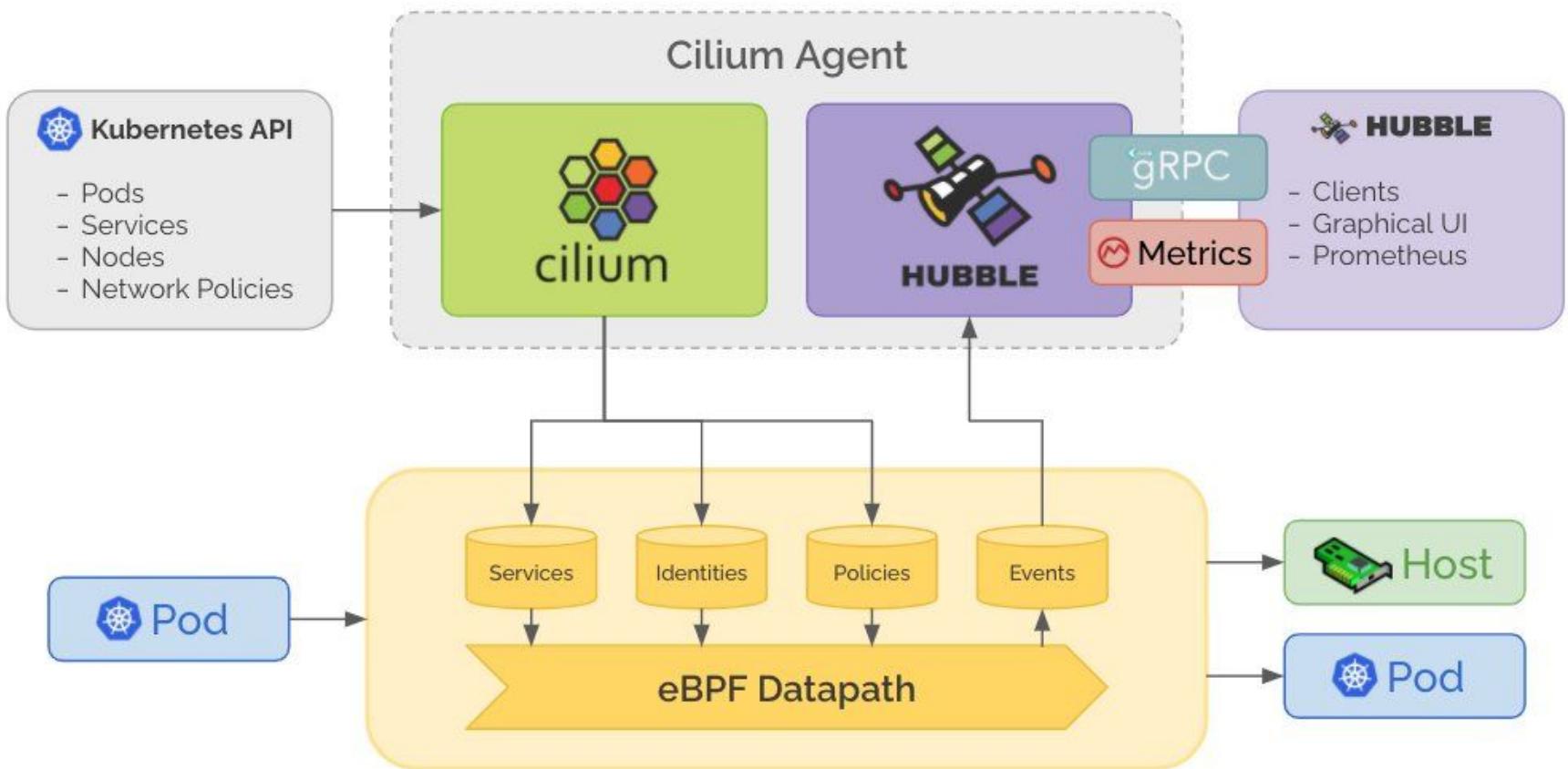
- Cilium had been joined with CNCF since Oct 2021 with concept “eBPF-based Networking, Observability, and Security”
- As CNCF member (incubating). Cilium is now open source project for provide networking, security, observability in cloud native environment (Kubernetes as CNI standard)
- With incredible performance in Cilium. Many cloud provider had been added cilium to their Kubernetes platform already since year 2021



Cilium on Kubernetes Dataplane



Cilium on Kubernetes Dataplane



Cilium on Kubernetes Dataplane



September 9, 2021

Author: Thomas Graf, CTO & Co-Founder Isovalent, Co-Creator Cilium

AWS has just announced the availability of EKS Anywhere to manage on-premises Kubernetes clusters. As part of this, AWS picked Cilium as the built-in default for networking and security. So, as you create your first EKS-A cluster, you will automatically have Cilium installed and benefit from the powers of eBPF.

Google Cloud

Blog Latest Stories What's New Product News Solutions & Technologies Topics

Gobind Johar
Product Manager, Google
Kubernetes Engine

Varun Marupadi
Software Engineer, Google
Kubernetes Engine

August 19, 2020

Editor's note: As of May 10, 2021, GKE Dataplane V2 is generally available starting with GKE version 1.20.6-gke.700. We're also using Dataplane V2 to make Kubernetes Network Policy logging generally available on Google Kubernetes Engine (GKE).

One of Kubernetes' true superpowers is its developer-first networking model. It provides easy-to-use features such as L3/L4 services and L7 Ingress to bring traffic into your cluster as well as network policies for isolating multi-tenant workloads. As more and more enterprises adopt Kubernetes, the gamut of use cases is widening with new requirements around multi-cloud, security, visibility and scalability. In addition, new technologies such as service mesh and serverless demand more customization from the underlying Kubernetes layer. These new requirements all have something in common: they need a more programmable dataplane that can perform Kubernetes-aware packet manipulations without sacrificing performance.

Enter [Extended Berkeley Packet Filter \(eBPF\)](#), a new Linux networking paradigm that exposes programmable hooks to the network stack inside the Linux kernel. The ability to enrich the kernel with user-space information—without jumping back and forth between user and kernel spaces—enables context-aware operations on network packets at high speeds.

Today, we're introducing GKE Dataplane V2, an opinionated dataplane that harnesses the power of eBPF and [Cilium](#), an open source project that makes the Linux kernel Kubernetes-aware using eBPF. Now in beta, we're also using Dataplane V2 to bring Kubernetes Network Policy logging to Google Kubernetes Engine (GKE).

What are eBPF and Cilium?



2020

2021

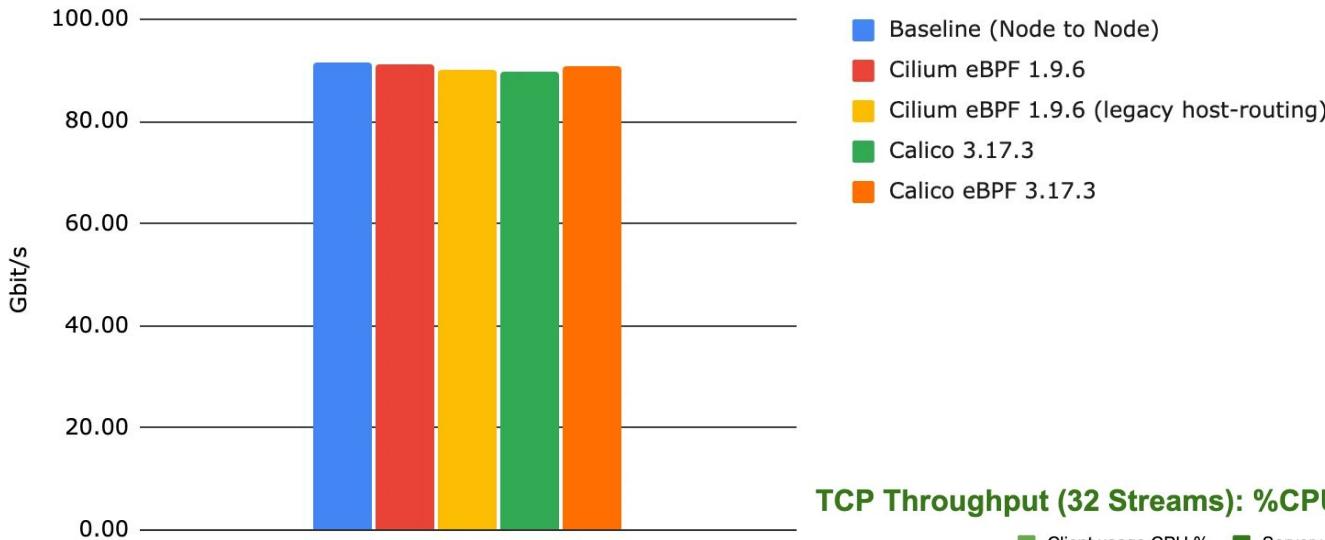
Kubernetes for enterprise business



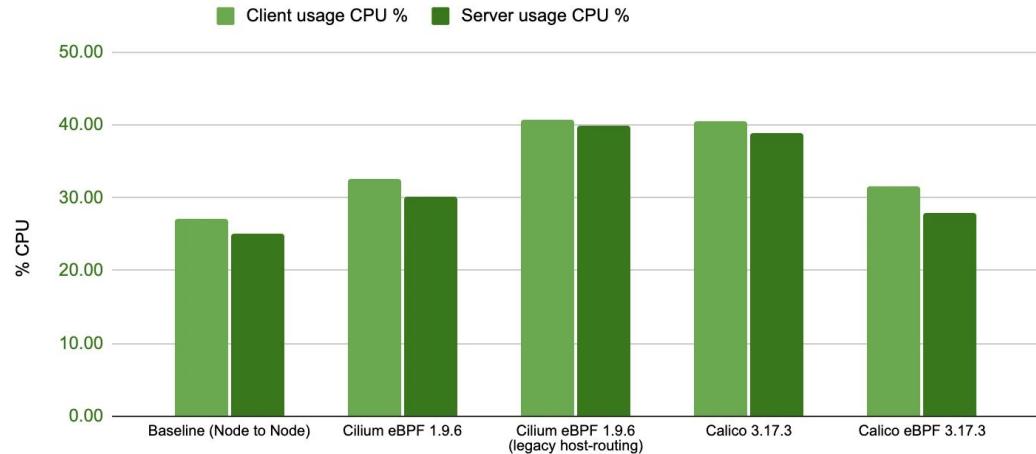
kubernetes
by Google

Cilium on Kubernetes Dataplane

TCP Throughput (32 Streams) - Higher is better



TCP Throughput (32 Streams): %CPU for 100Gbit/s - Lower is better



Cilium on Kubernetes Dataplane

Networking

Observability

Security



Native support for service type Load Balancer and Egress



Identity-aware Visibility



Transparent Encryption



Scalable Kubernetes CNI



Advanced Self Service Observability



Security Forensics + Audit



Multi-cluster Connectivity



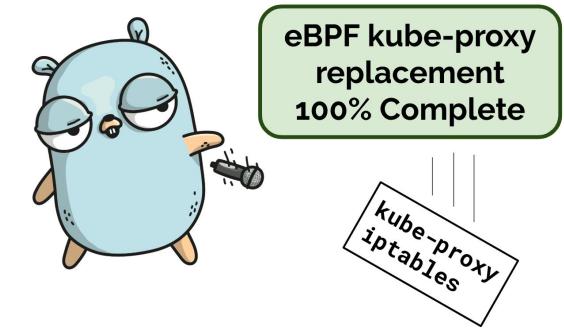
Network Metrics + Policy Troubleshooting



Advanced Network Policy

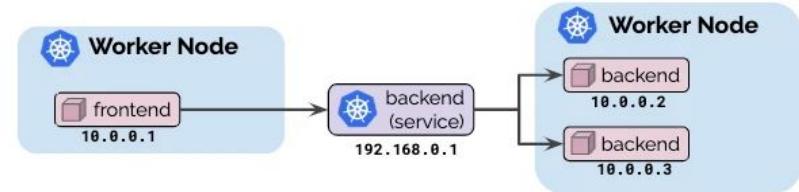
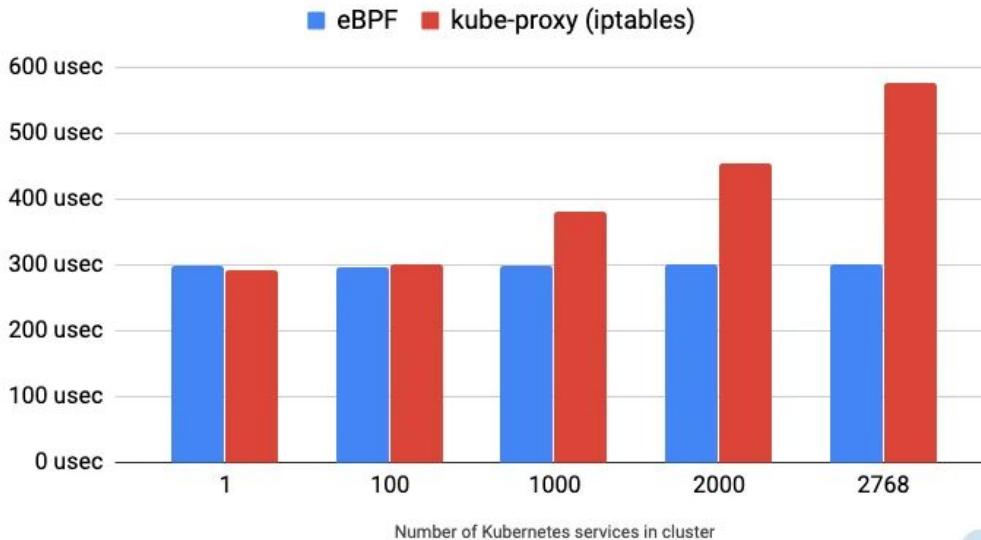
No kube-proxy with Cilium !!!

- Normally Kubernetes will handle network under hood with kube proxy/iptables.
- Base on idea of iptables operation. This also have some performance concern in case we have multiple service in single of node (Mean huge of iptables).
- As Cilium is base on eBPF and run on kernel. So it can have capability to full replacement with “kube-proxy” (Support with kernel v4.19.57, v5.1.16, v5.2.0)
- *Remark: This topic also need to test before operate in production env.



No kube-proxy with Cilium !!!

HTTP request latency via k8s service (usec)



Network-based load-balancing

→ connect("192.168.0.1")

Local Socket

```
#1 10.0.0.1→192.168.0.1  
#2 10.0.0.1←192.168.0.1  
#3 10.0.0.1→192.168.0.1  
#n [...]
```

Network DNAT

```
10.0.0.1→10.0.0.2  
10.0.0.1←10.0.0.2  
10.0.0.1→10.0.0.2  
[...]
```

Remote Socket

```
10.0.0.1→10.0.0.2  
10.0.0.1←10.0.0.2  
10.0.0.1→10.0.0.2  
[...]
```

DNAT

Socket-based load-balancing

→ connect("192.168.0.1")

DNAT

Local Socket

```
#1 10.0.0.1→10.0.0.2  
#2 10.0.0.1←10.0.0.2  
#3 10.0.0.1→10.0.0.2  
#n [...]
```

Remote Socket

```
10.0.0.1→10.0.0.2  
10.0.0.1←10.0.0.2  
10.0.0.1→10.0.0.2  
[...]
```

Ref:https://cilium.io/blog/2019/08/20/cilium-1_

Kubernetes for enterprise business

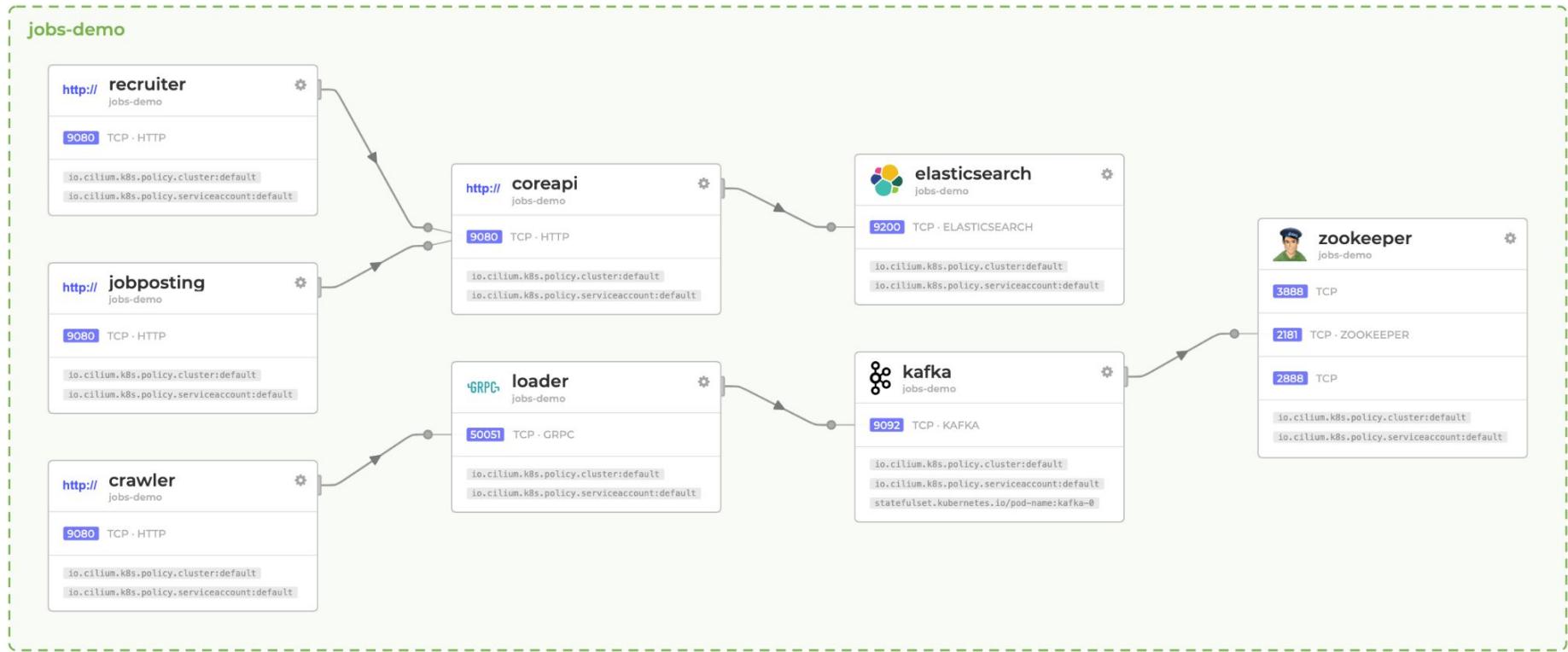


kubernetes
by Google

Hubble observability for all

- Hubble is the module in Cilium for observability network distribution and security
- With capability of eBPF. Hubble can retrieve connection between microservice as “Service Dependency Graph” (Like kiali in istio). This will help developer to have visibility about what is going to microservice.
- Hubble also can observability about network policy, network behavior etc.

Hubble observability for all



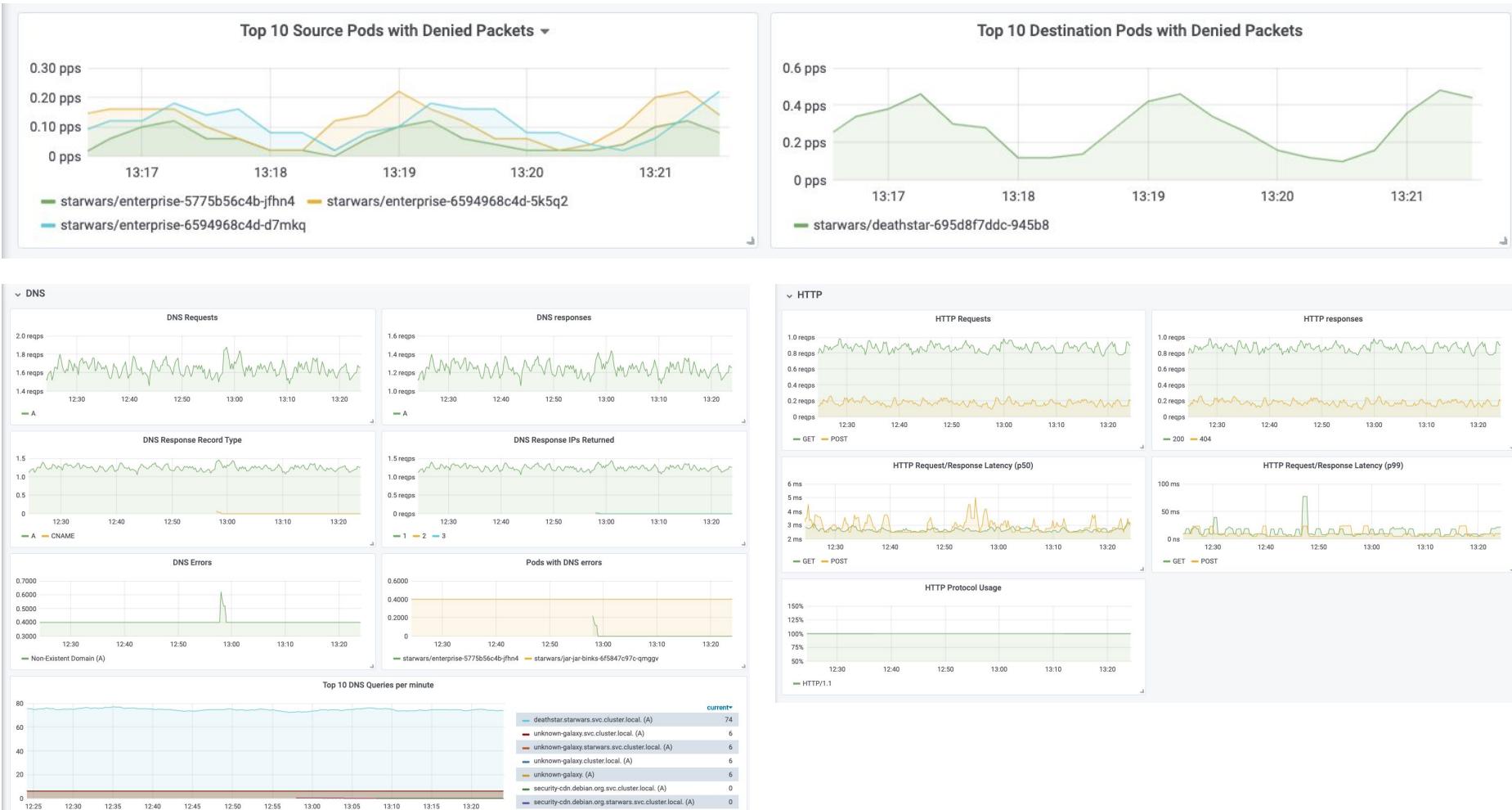
Ref:<https://github.com/cilium/hubble>

Kubernetes for enterprise business



kubernetes
by Google

Hubble observability for all



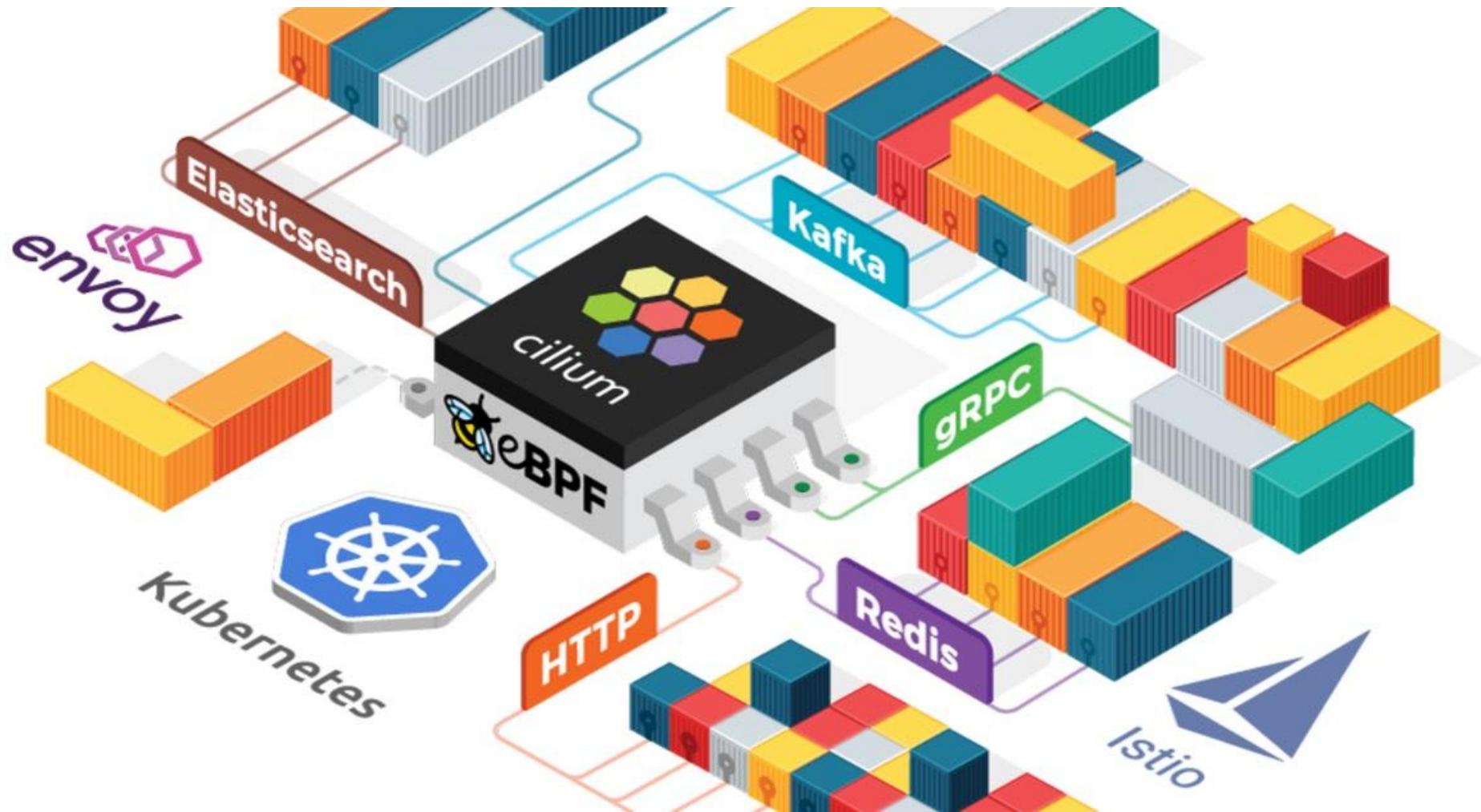
Ref: <https://github.com/cilium/hubble>

Kubernetes for enterprise business



kubernetes
by Google

Demo Session



Kubernetes for enterprise business

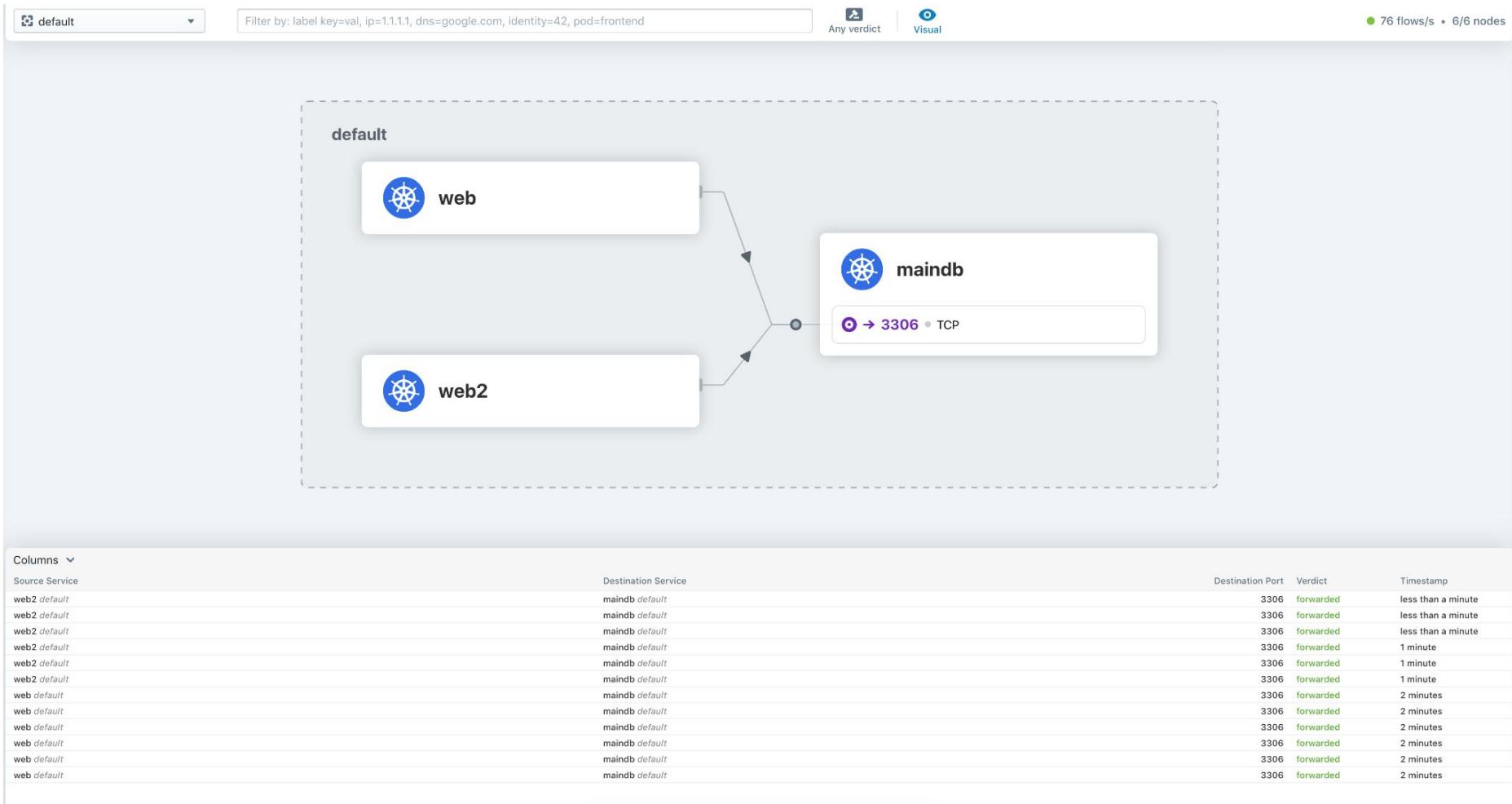


kubernetes
by Google

Demo Session

- In Demo session. We will setup 2 scenario for demo hubble observability
- Scenario 1 (Single Namespace): Basic rest api with 2 rest api connect in same database
- Scenario 2 (Multiple Namespace): Setup application with 3 namespaces and connect to each other:
 - Client namespace:
 - Frontend/Backend namespace:
 - Management namespace:

Scenario 1: Single Namespace



Scenario 2: Multiple Namespace

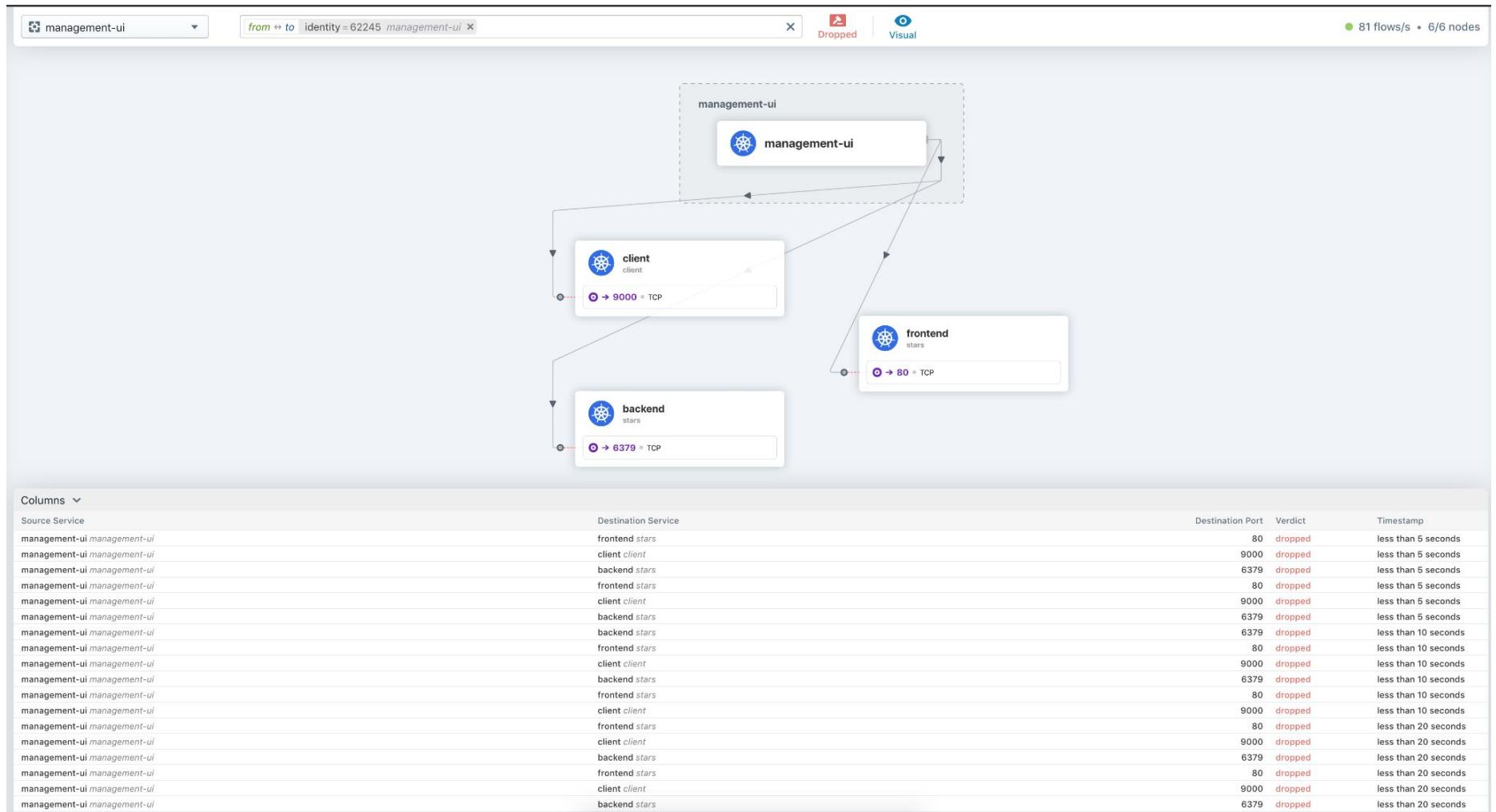
Screenshot of a network flow visualization tool showing traffic between multiple services across namespaces.

The interface includes:

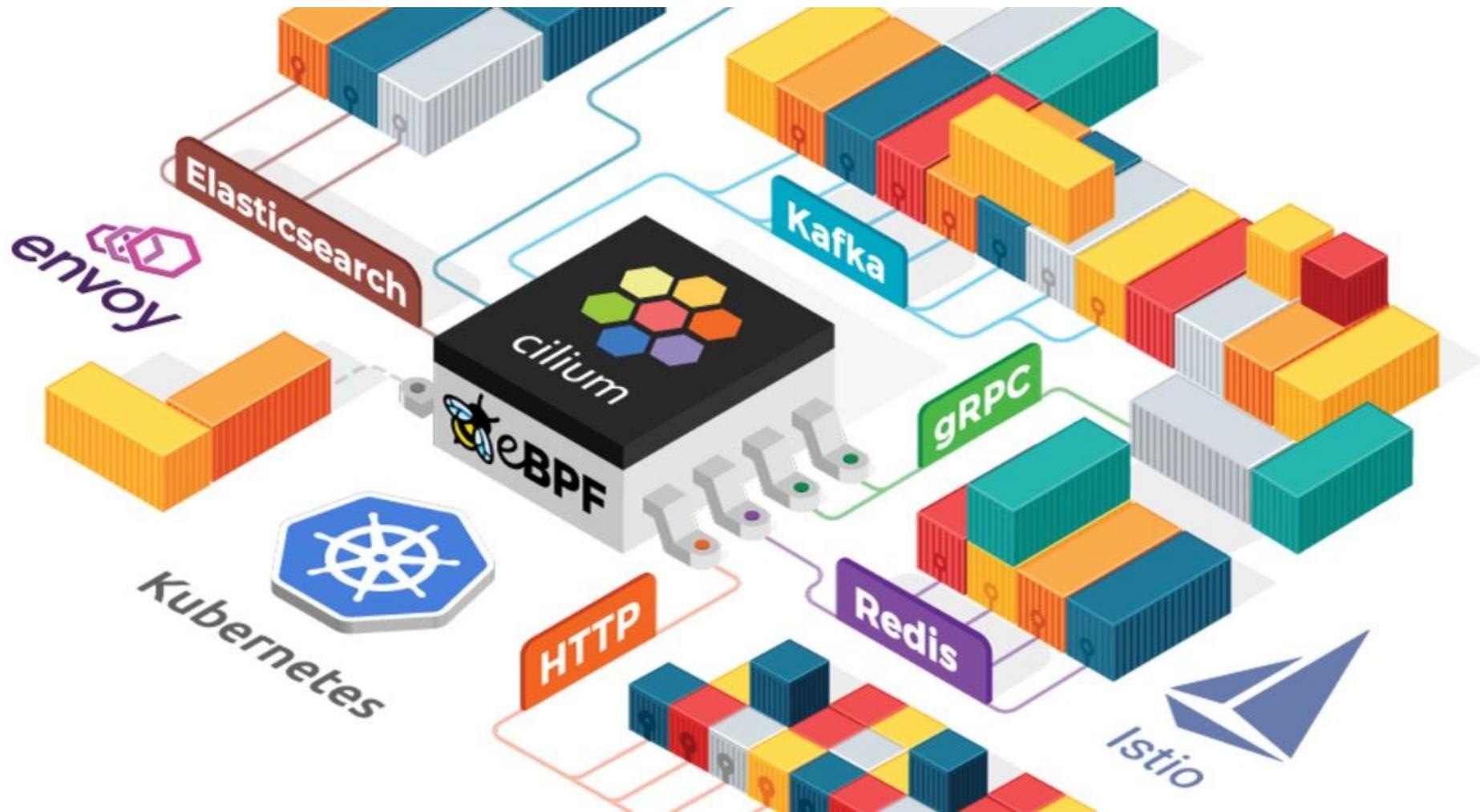
- Top bar: "management-ui" dropdown, "from ↔ to identity=62245 management-ui X", "Forwarded" button, "Visual" button, and status "83 flows/s • 6/6 nodes".
- Diagram: A network graph with nodes "management-ui", "client", "frontend", and "backend". "management-ui" has a self-loop arrow. "client" has an outgoing arrow to "backend" and an incoming arrow from "management-ui". "frontend" has an outgoing arrow to "backend" and an incoming arrow from "management-ui". "backend" has an outgoing arrow to "client".
- Table: A detailed view of the 83 flows. It lists Source Service (all entries are "management-ui"), Destination Service (various entries like "frontend stars", "backend stars", etc.), Destination Port (all 80), Verdict ("forwarded" in green), and Timestamp (ranging from 6 minutes ago to 7 minutes ago).



Scenario 2: Multiple Namespace



Q&A



Kubernetes for enterprise business



kubernetes
by Google