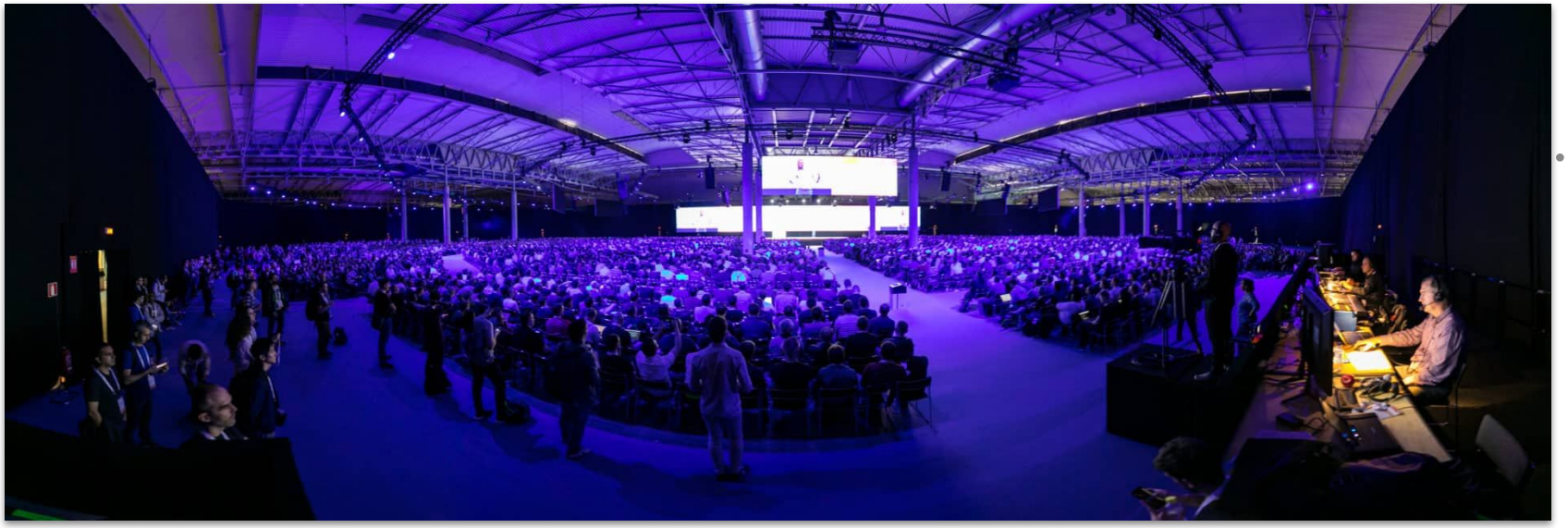
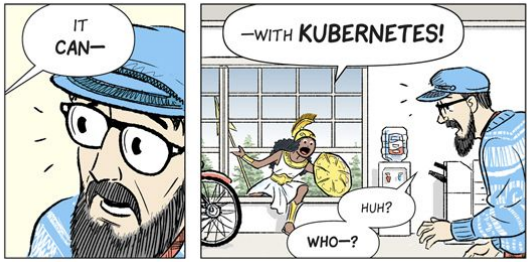




Secure and monitor your service connectivity without service mesh

(Praparn Lueangphoonlap)

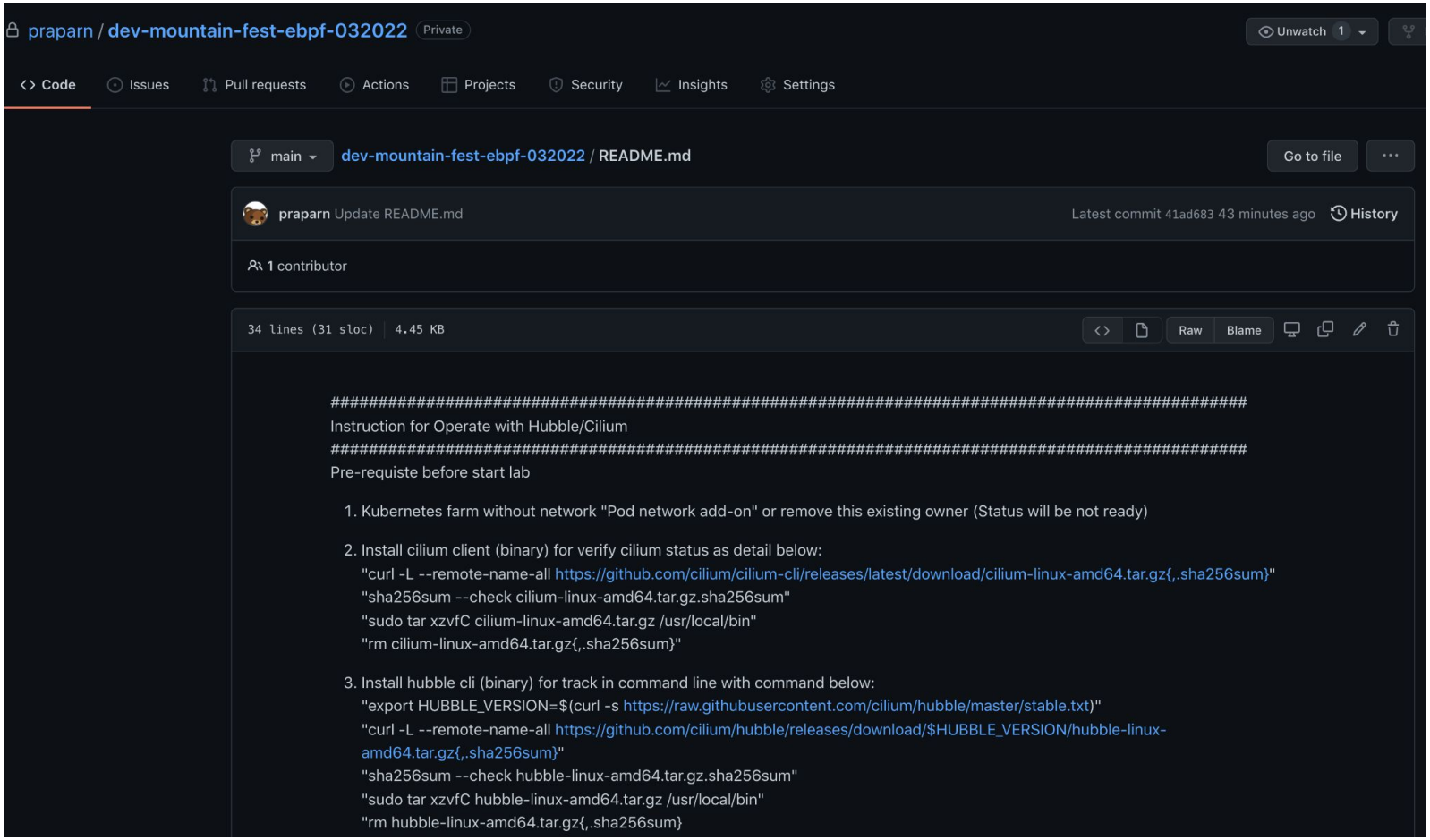
kubernetes by Google

Agenda

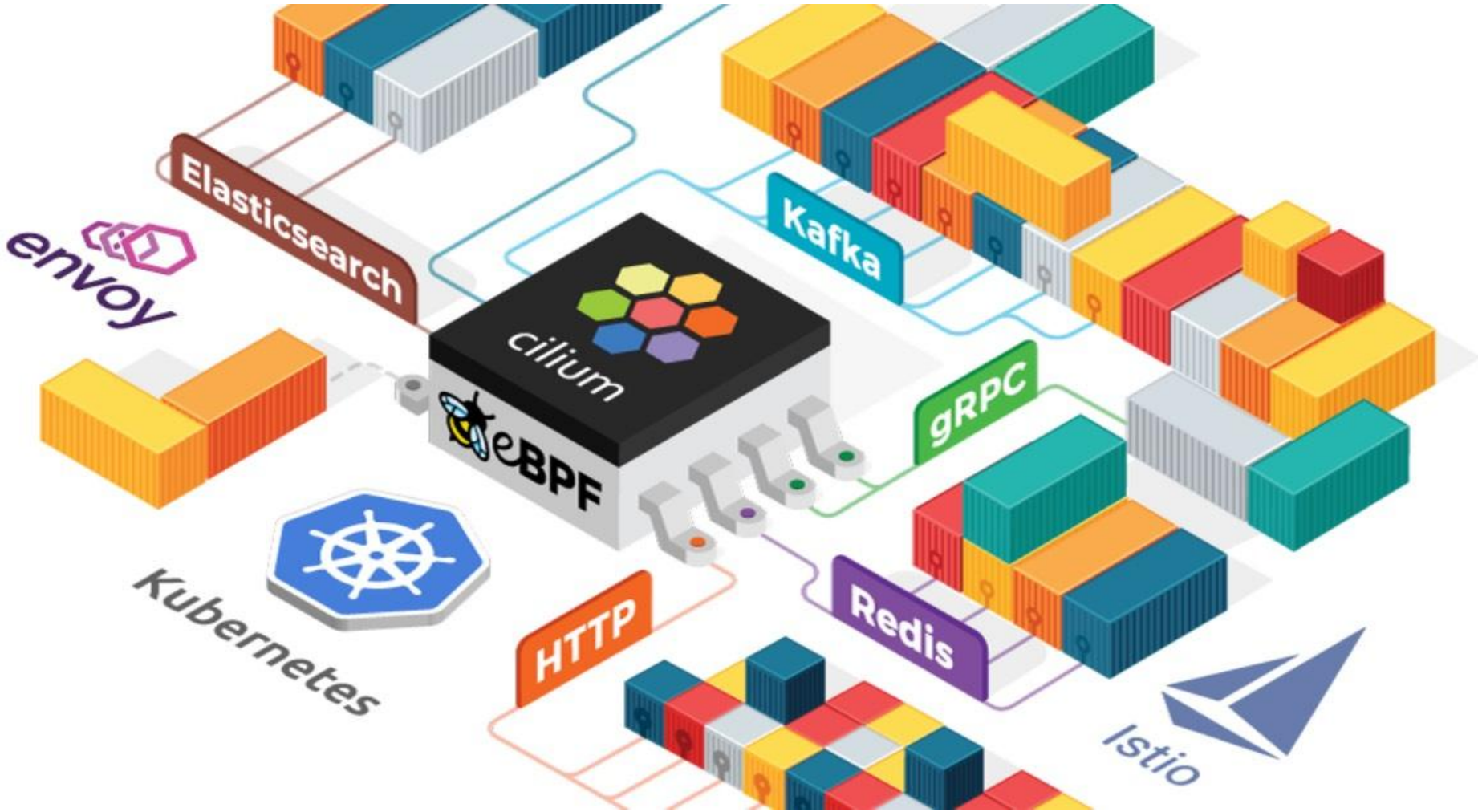
- Why microservice connectivity is matter ?
  - Secure and Visibility is the Key
  - What the solution (Istio ?) and Problem
- Introduction to eBPF and Cilium
  - What is and Why eBPF?
  - Cilium on Kubernetes as data plane
    - Networking
    - Observability
    - Security
    - No kube-proxy with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

Git Resource

- https://github.com/praparn/dev-mountain-fest-ebpf-032022



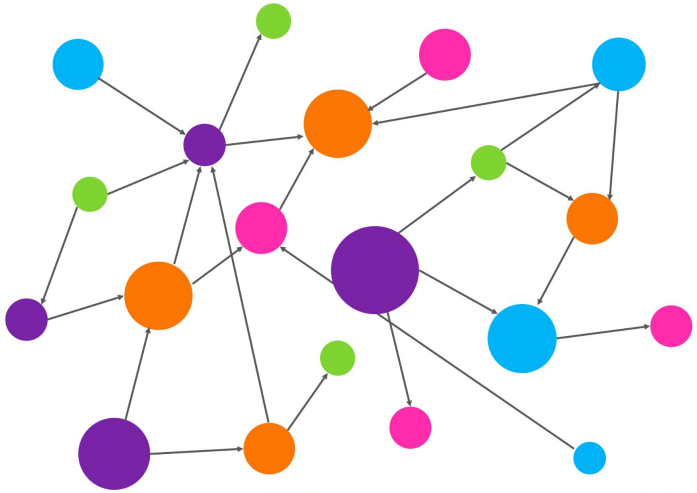
Why microservice connectivity is matter ?



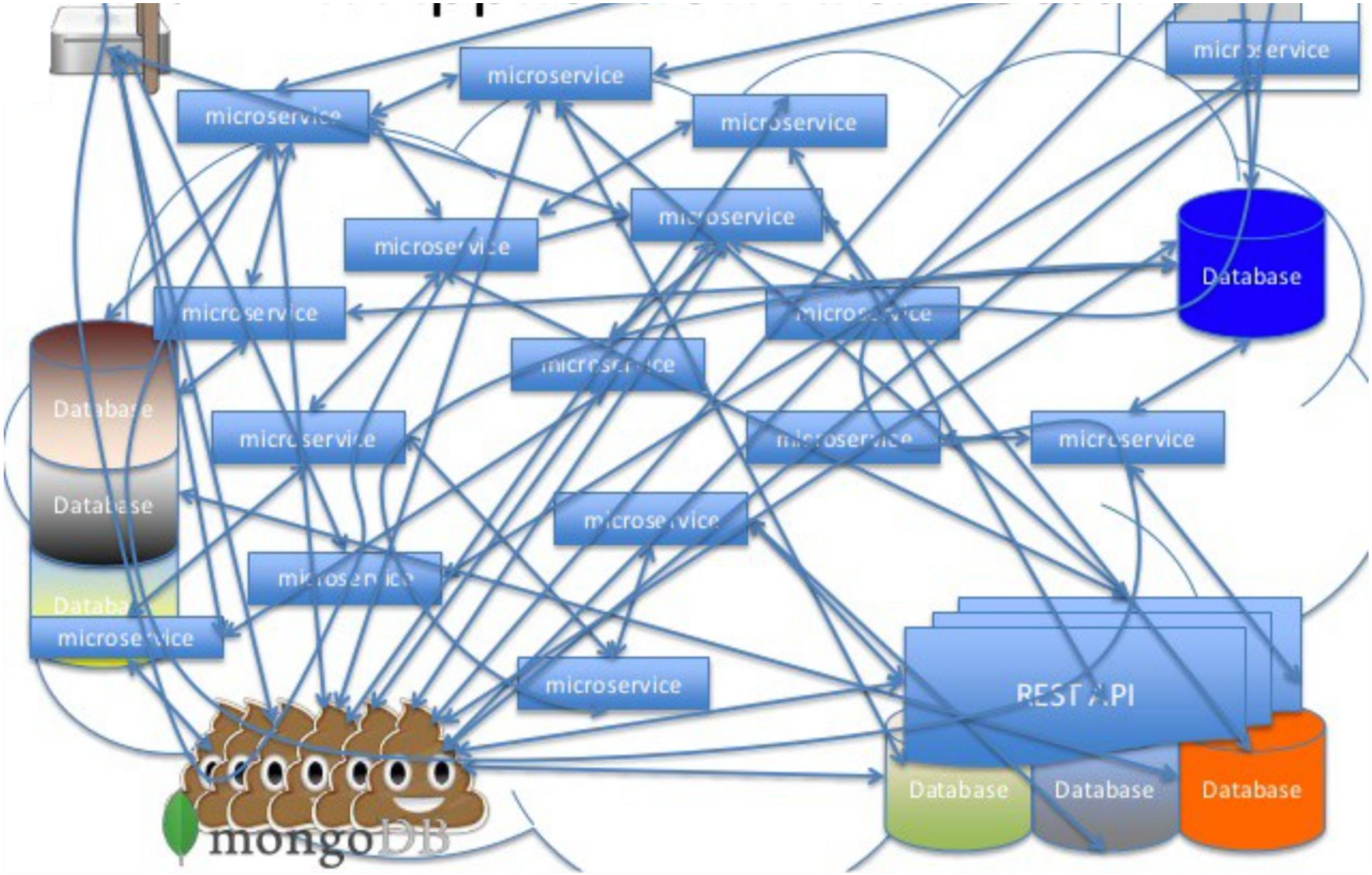
kubernetes by Google

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 without expect !!!
  - etc.

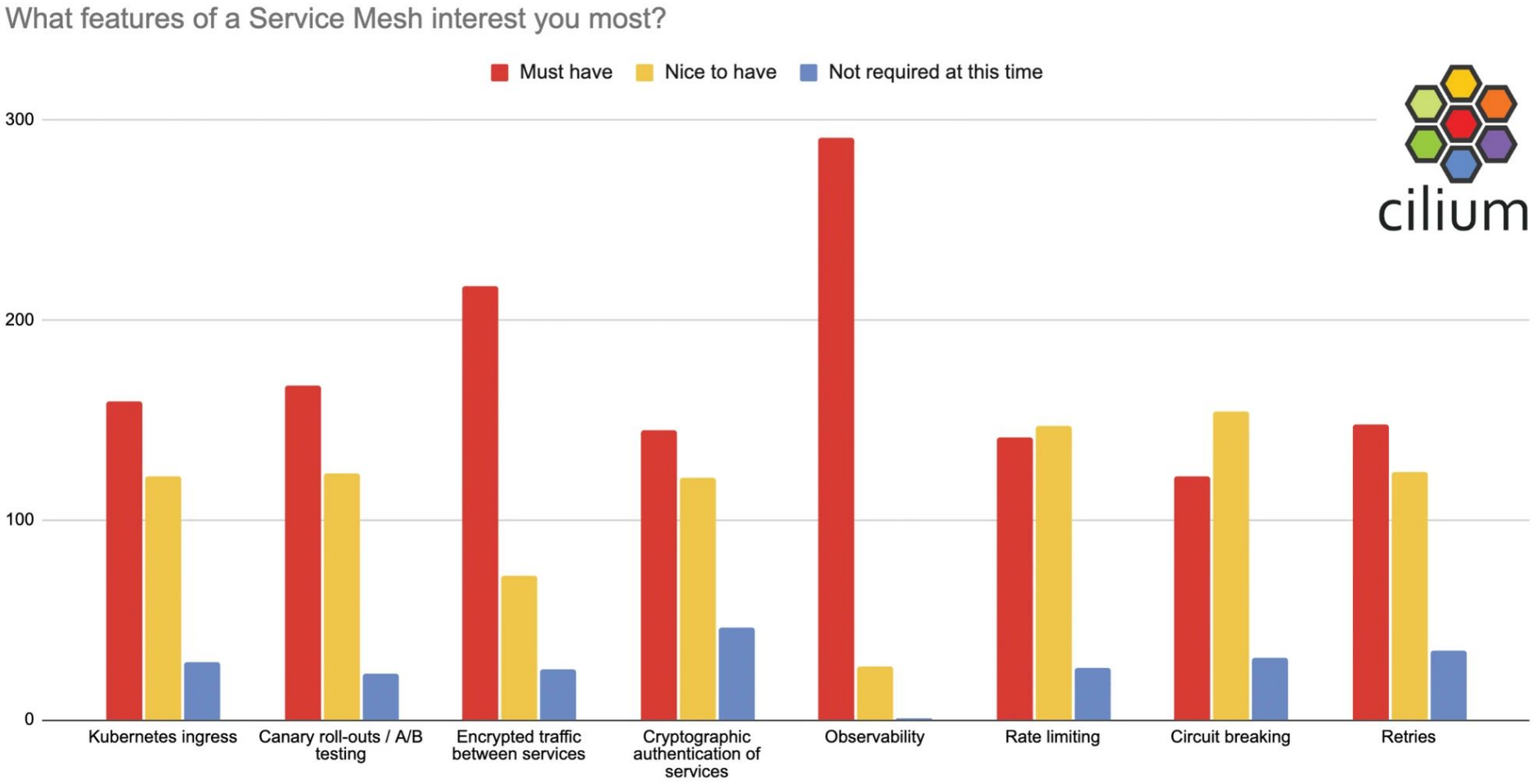


Why microservice connectivity is matter ?



kubernetes by Google

Why microservice connectivity is matter ?



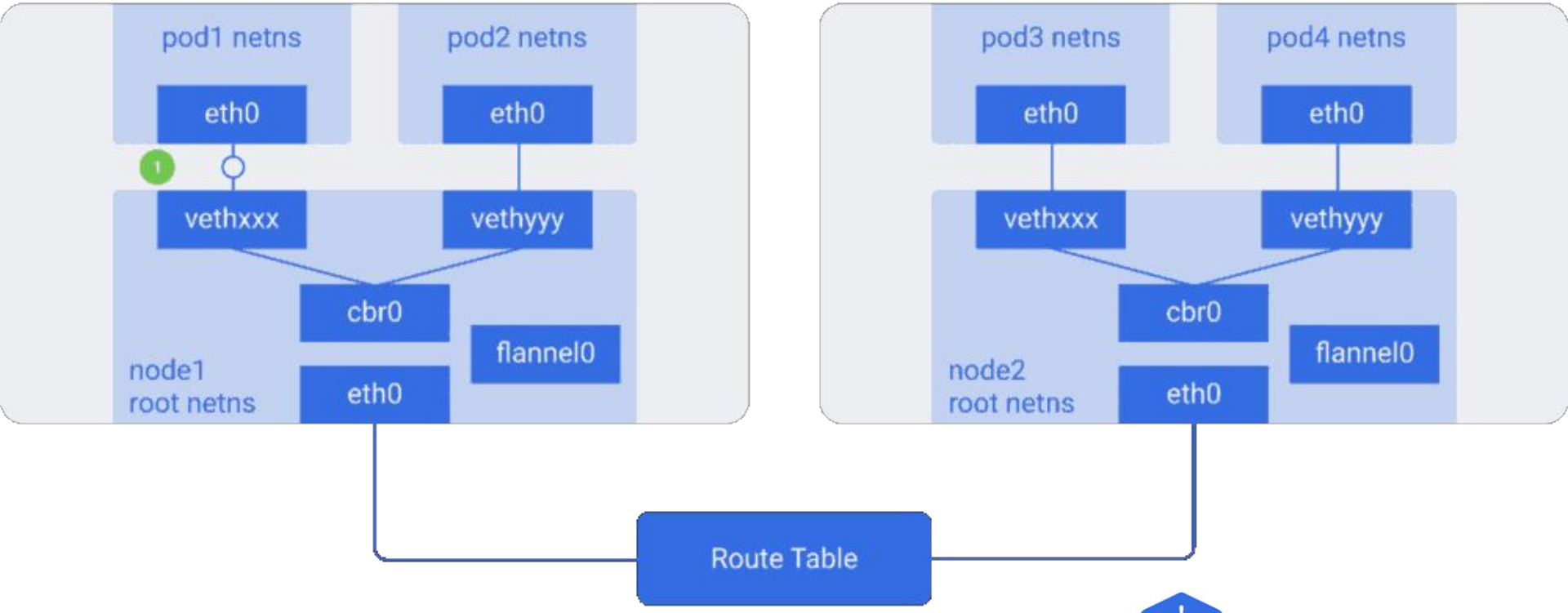
cilium

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

kubernetes by Google

Secure and Visibility is Key

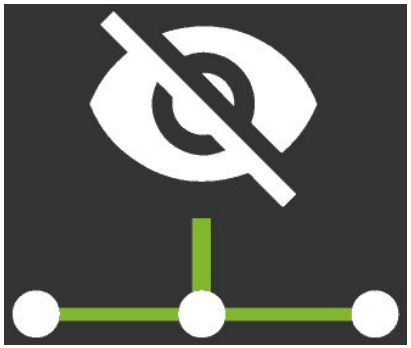
- When we had been deploy our microservice on Kubernetes. By default all microservice can accessible to any microservice that they know the service name !!!



kubernetes by Google

Secure and Visibility is Key

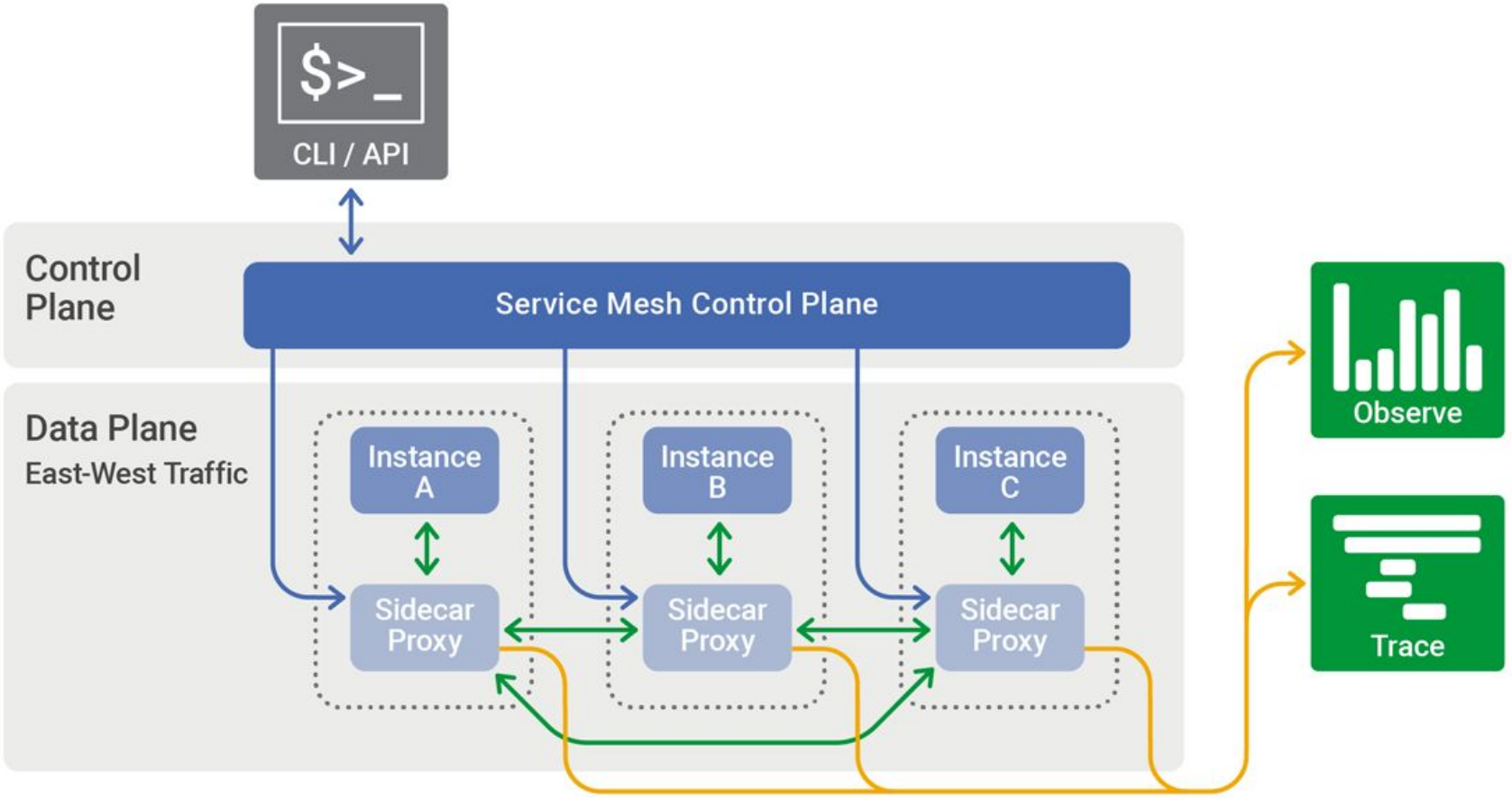
- And the exactly happen is… We never have chance to see that is going on
  - Who try to connect our application (pods)
  - Our pods is try to reach other pods or not ? (If there got compromised ?)
  - Any connection is normal
    - Which source connection ?
    - How frequency ?
  - Any connection is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



What the solution? (Istio ?)

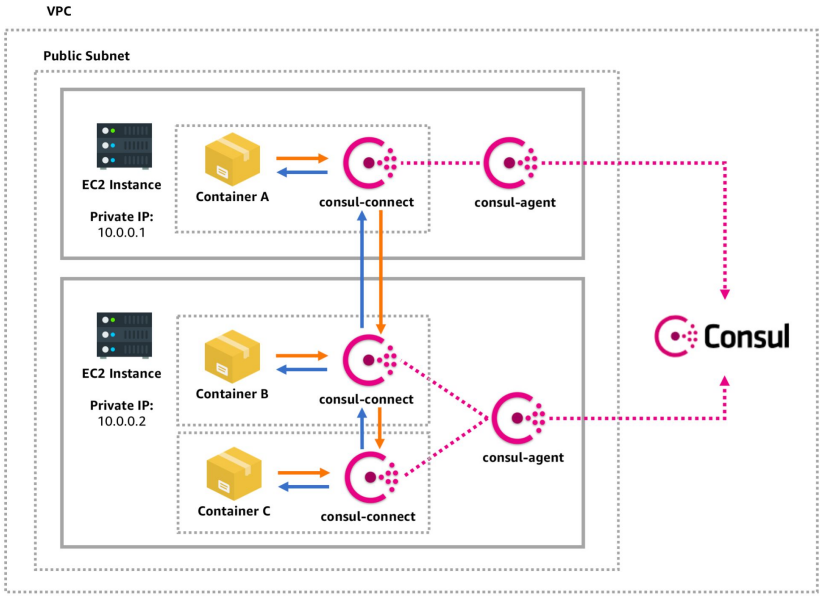
- **Yes (1)**, Service Mesh. Believe that 1st, 2nd solution bring in the table.
- Istio, Linked, Kuma, Console, Dynatrace etc or In-house with similar concept
  - Pass all connection in pods via “sidecar proxy” for observe/trace all connection and manable service mesh or send all connect to control plane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

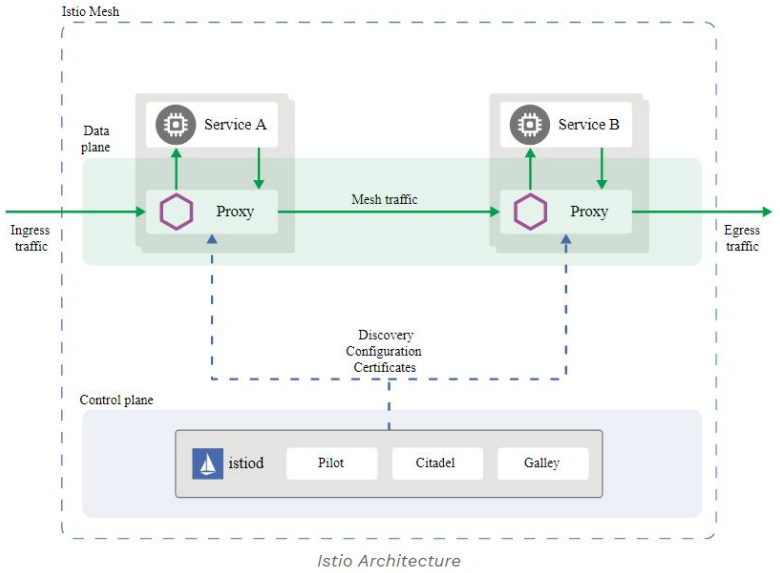
What the solution? (Istio ?)



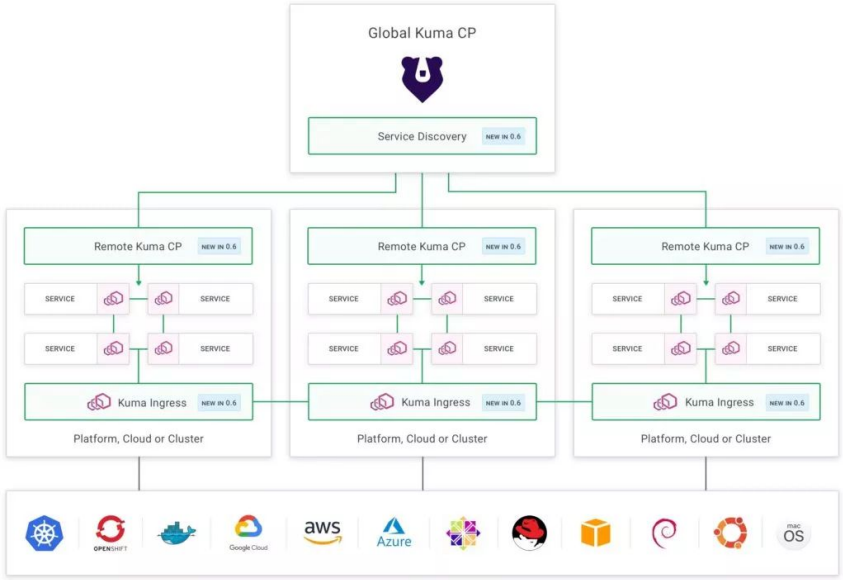
kubernetes by Google

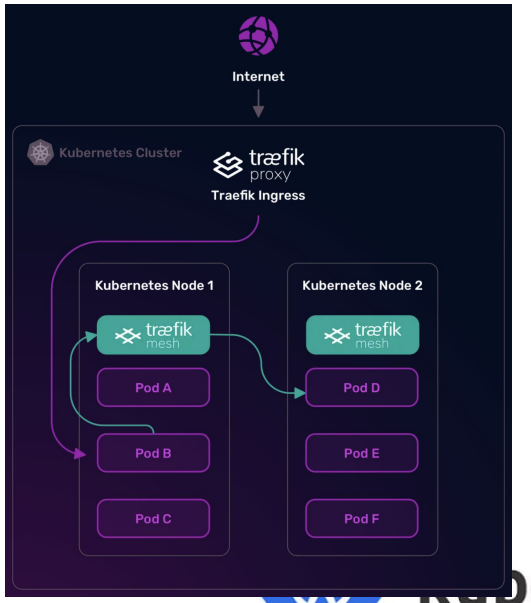
What the solution? (Istio ?)





Istio Architecture

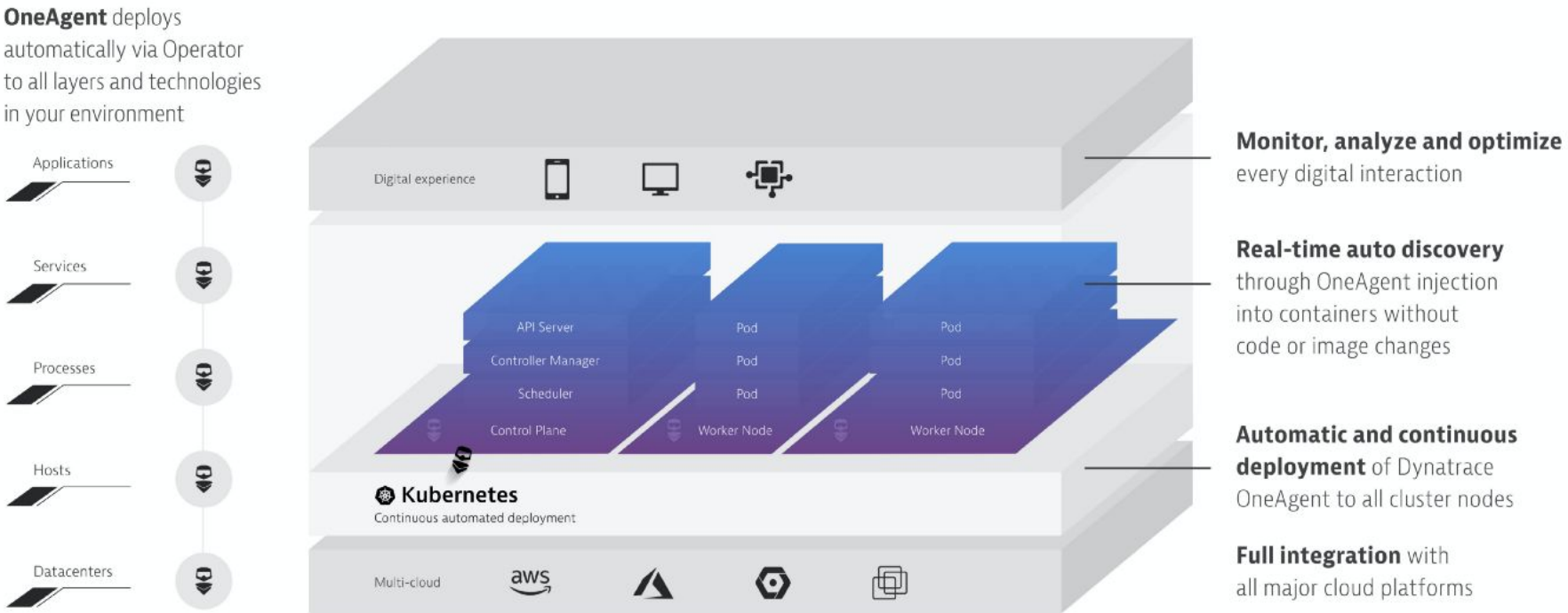




Kubernetes for enterprise business

Kubernetes by Google

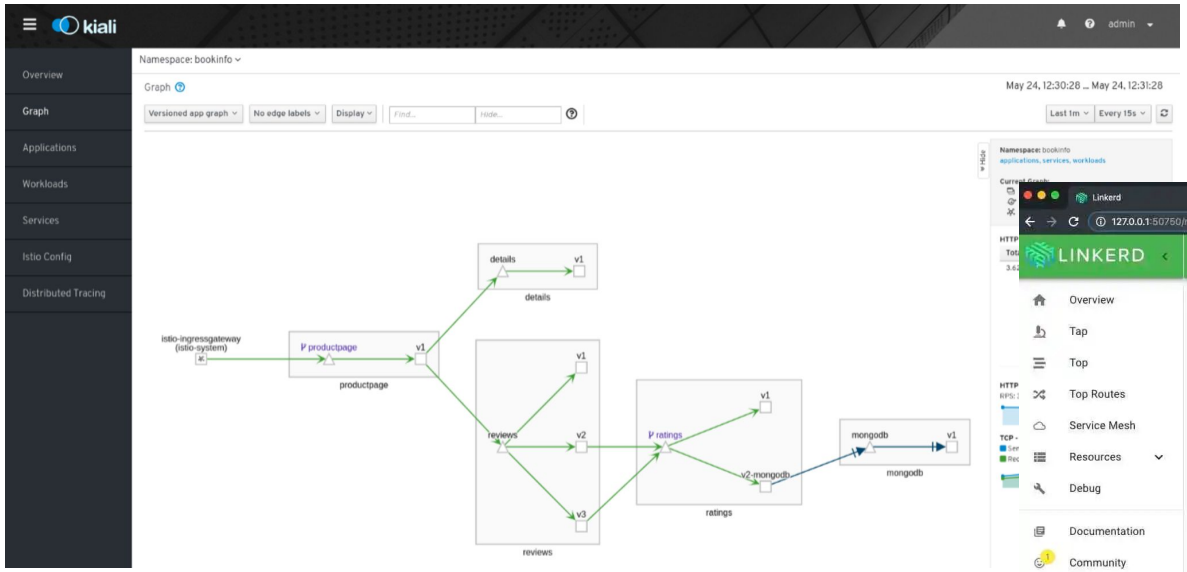
What the solution? (Istio ?)

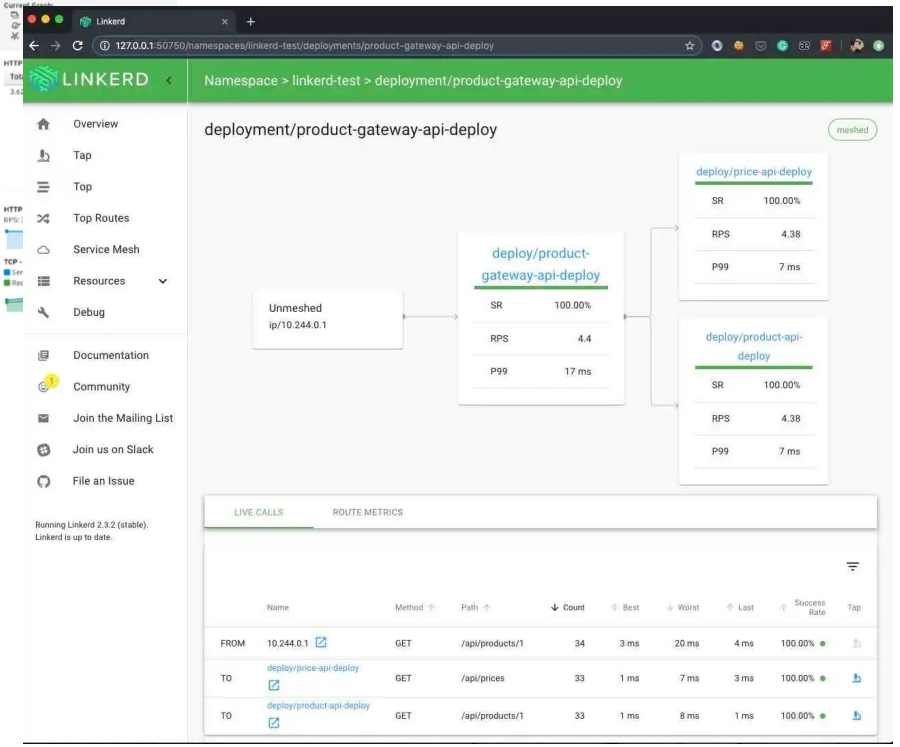


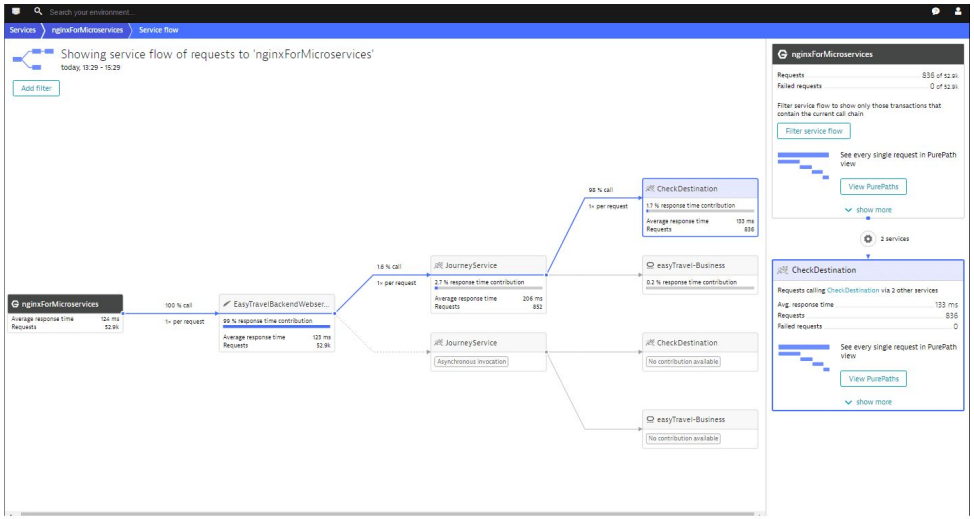
Kubernetes for enterprise business

kubernetes by Google

What the solution? (Istio ?)







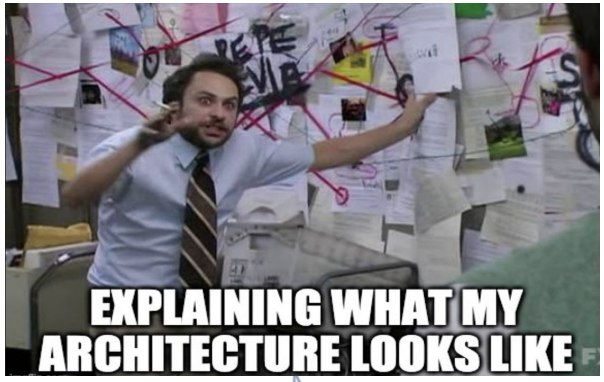
Kubernetes for enterprise business

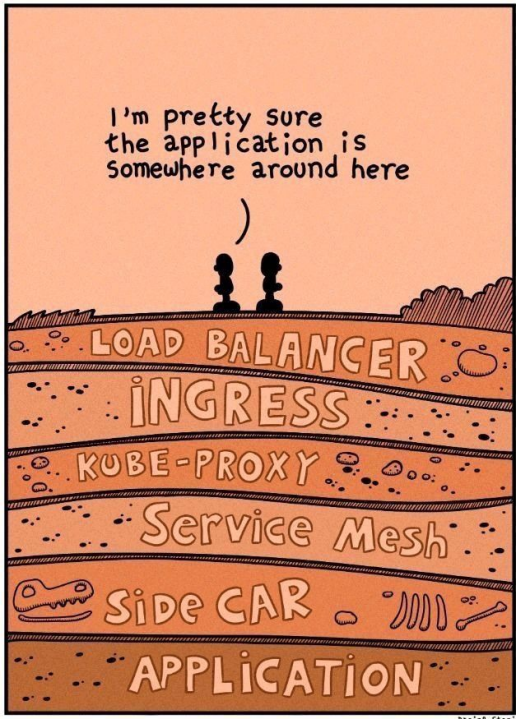
kubernetes by Google

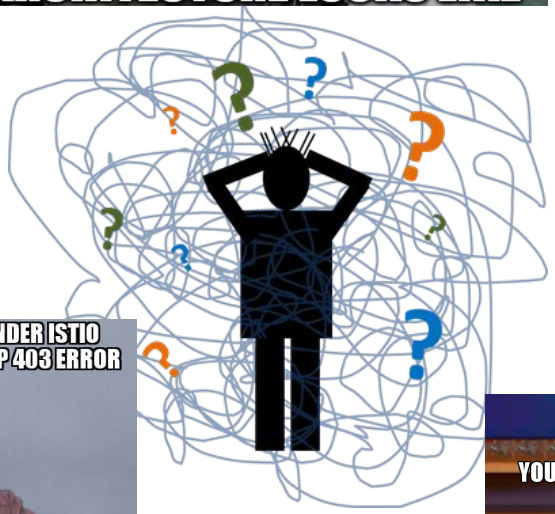
What the solution? (Istio ?)

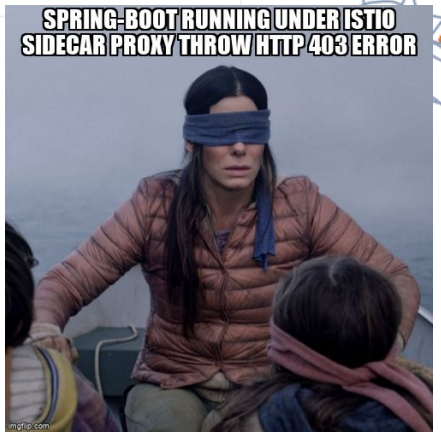
- From this concept every microservice/pods need to have sidecar proxy ?
- **So what is the problem ?**
  - Sidecar proxy/Control Plane (The other hand: another proxy for our request in/out) at principle will slightly impact performance in several way
  - Complicate: More layer is make complicate for 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)

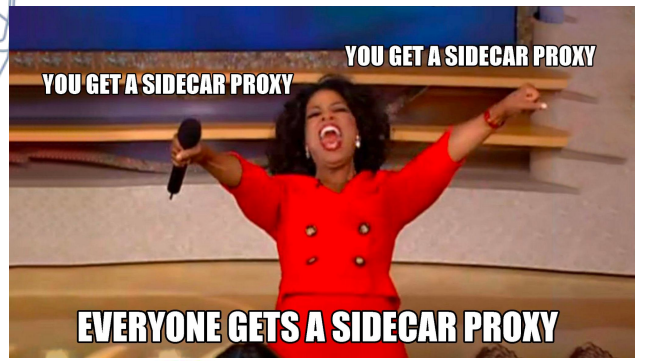
What the solution? (Istio ?)







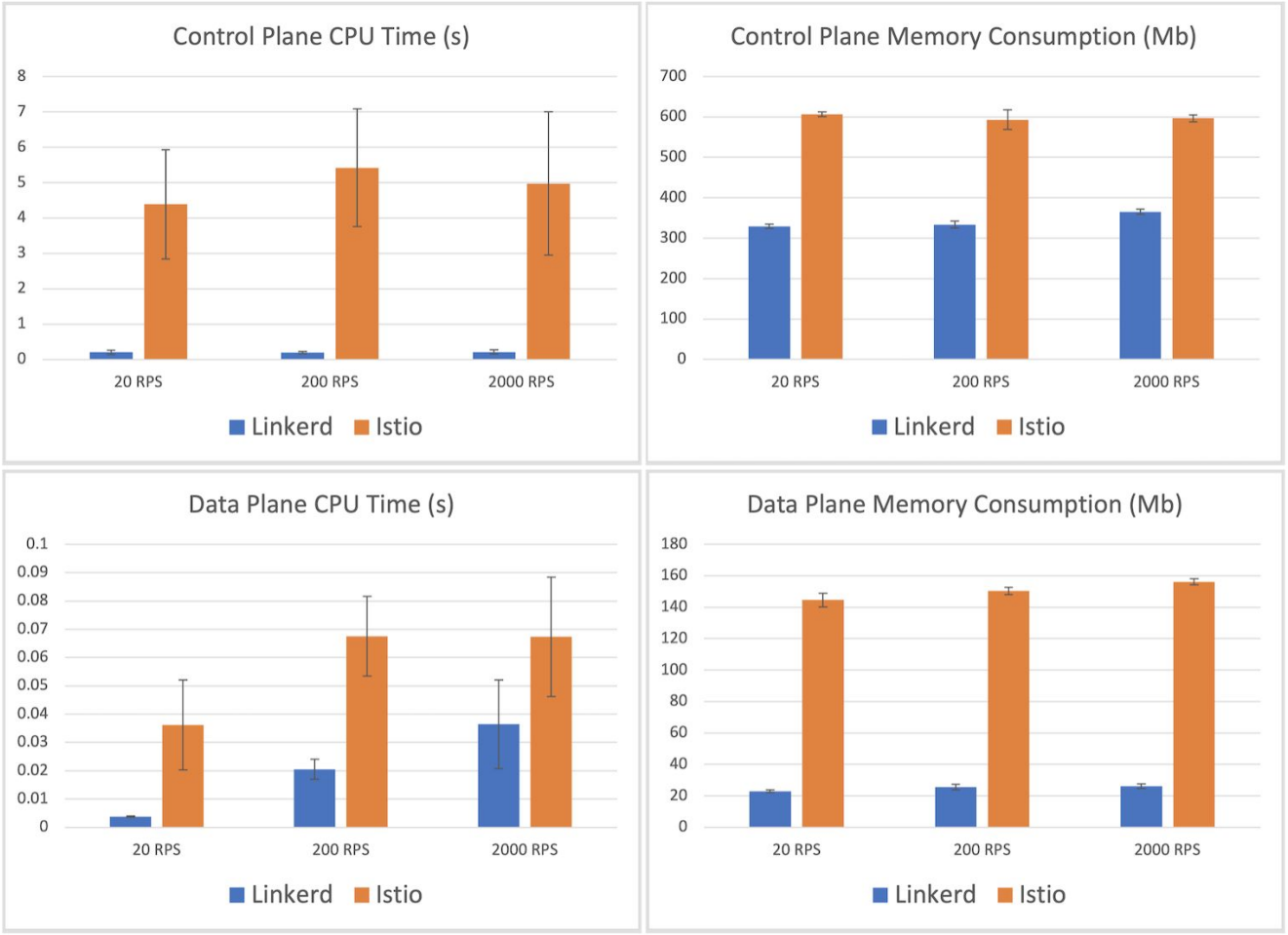




Kubernetes for enterprise business

kubernetes by Google

What the solution? (Istio ?)



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

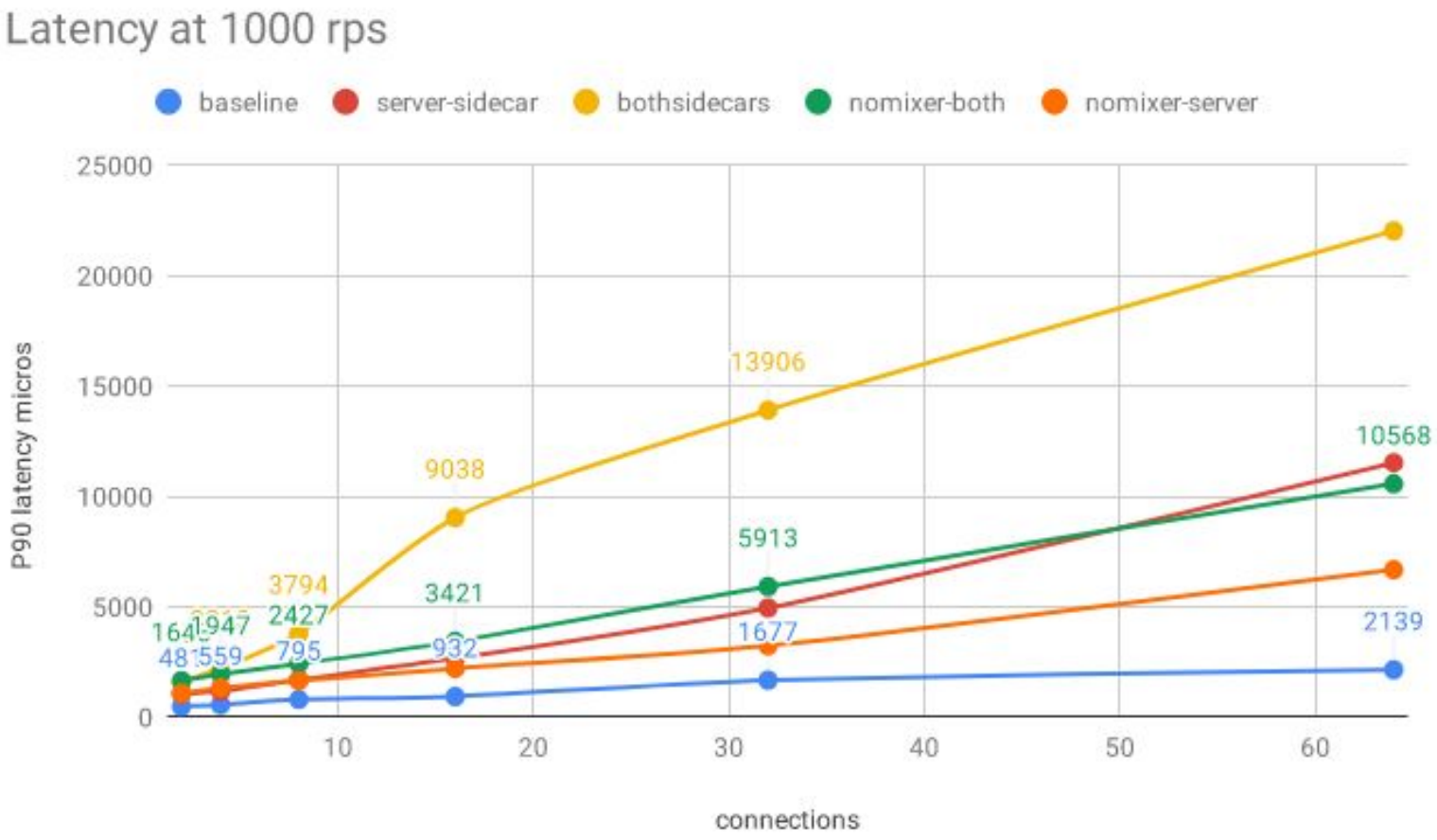
Kubernetes for enterprise business

kubernetes by Google

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 ?)

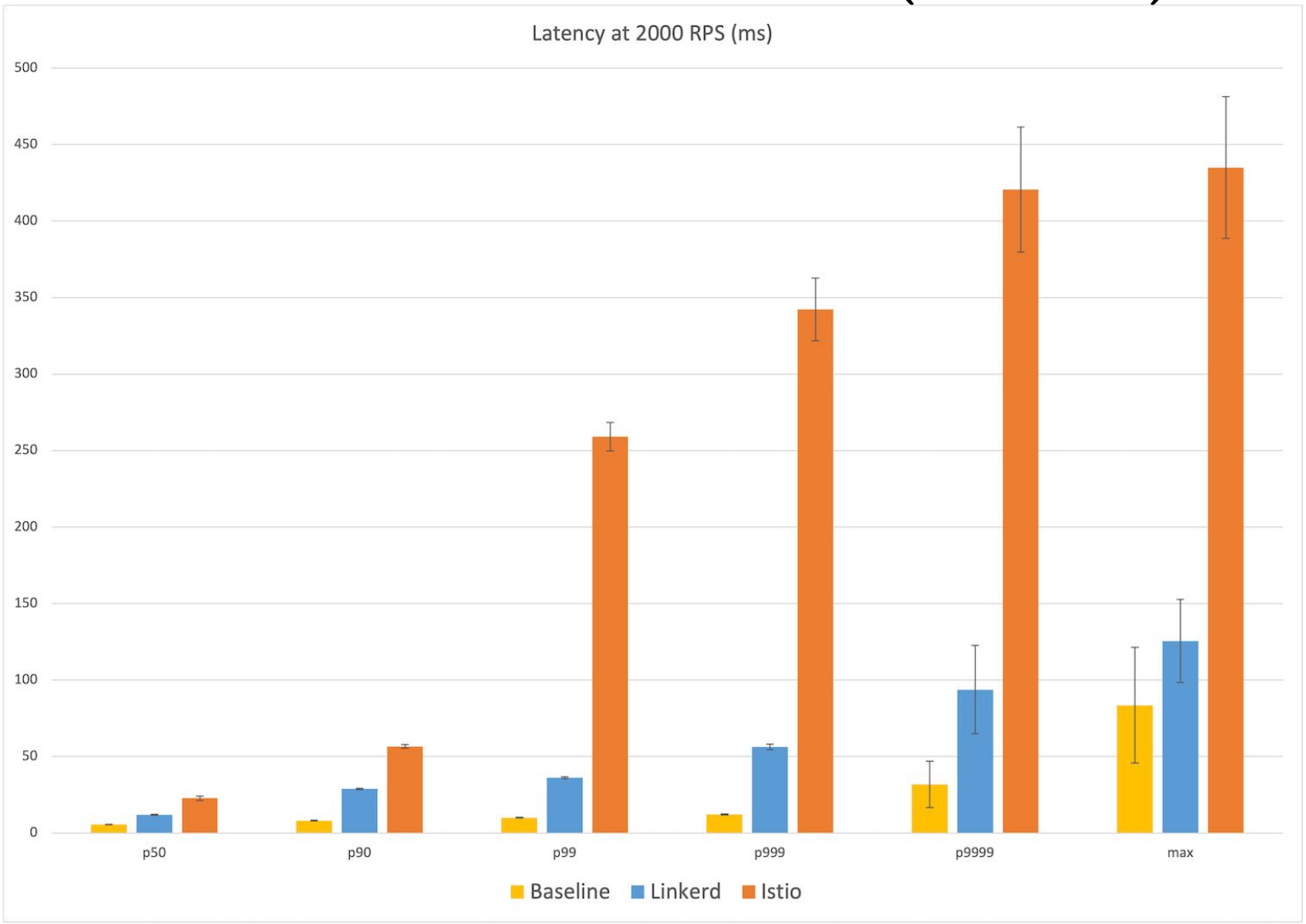


Ref:https://istio.io/v1.2/docs/concepts/performance-and-scalability/

Kubernetes for enterprise business

kubernetes by Google

What the solution? (Istio ?)



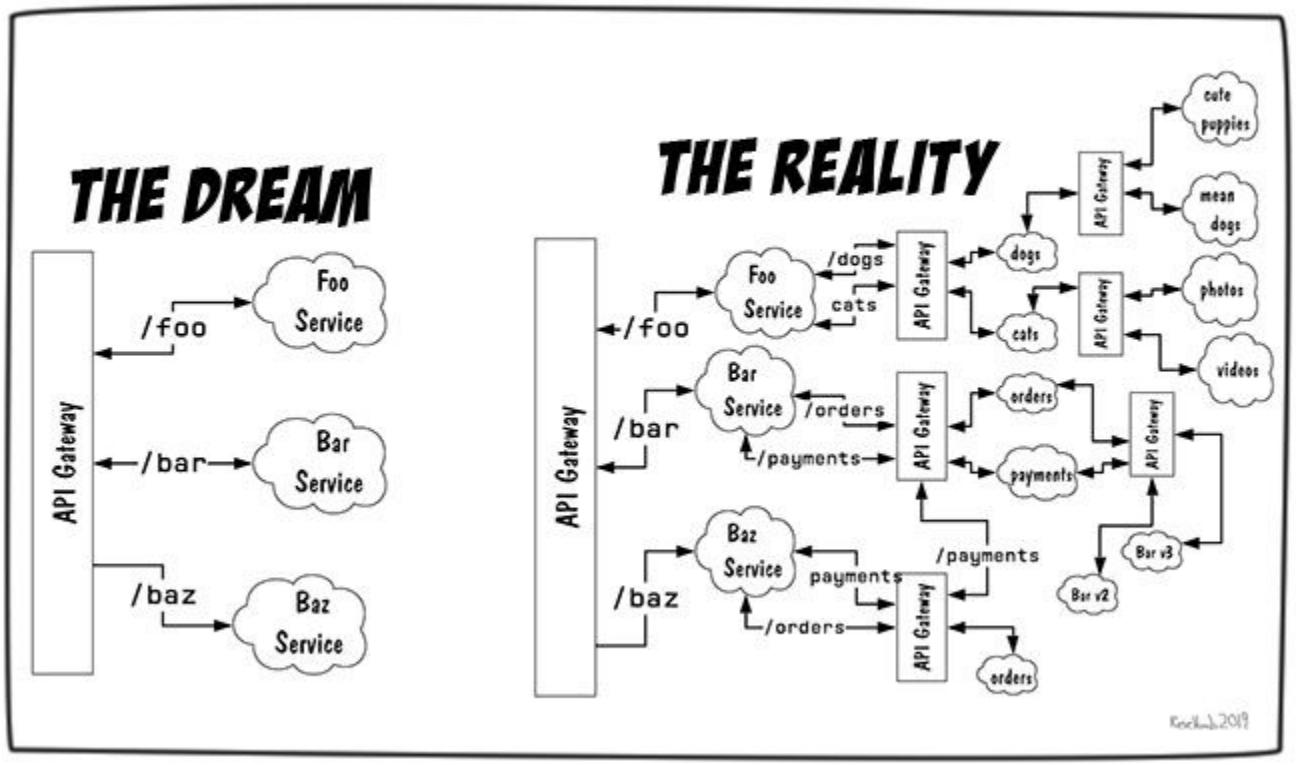
https://linkerd.io/2021/11/29/linkerd-vs-istio-benchmarks-2021/

kubernetes 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

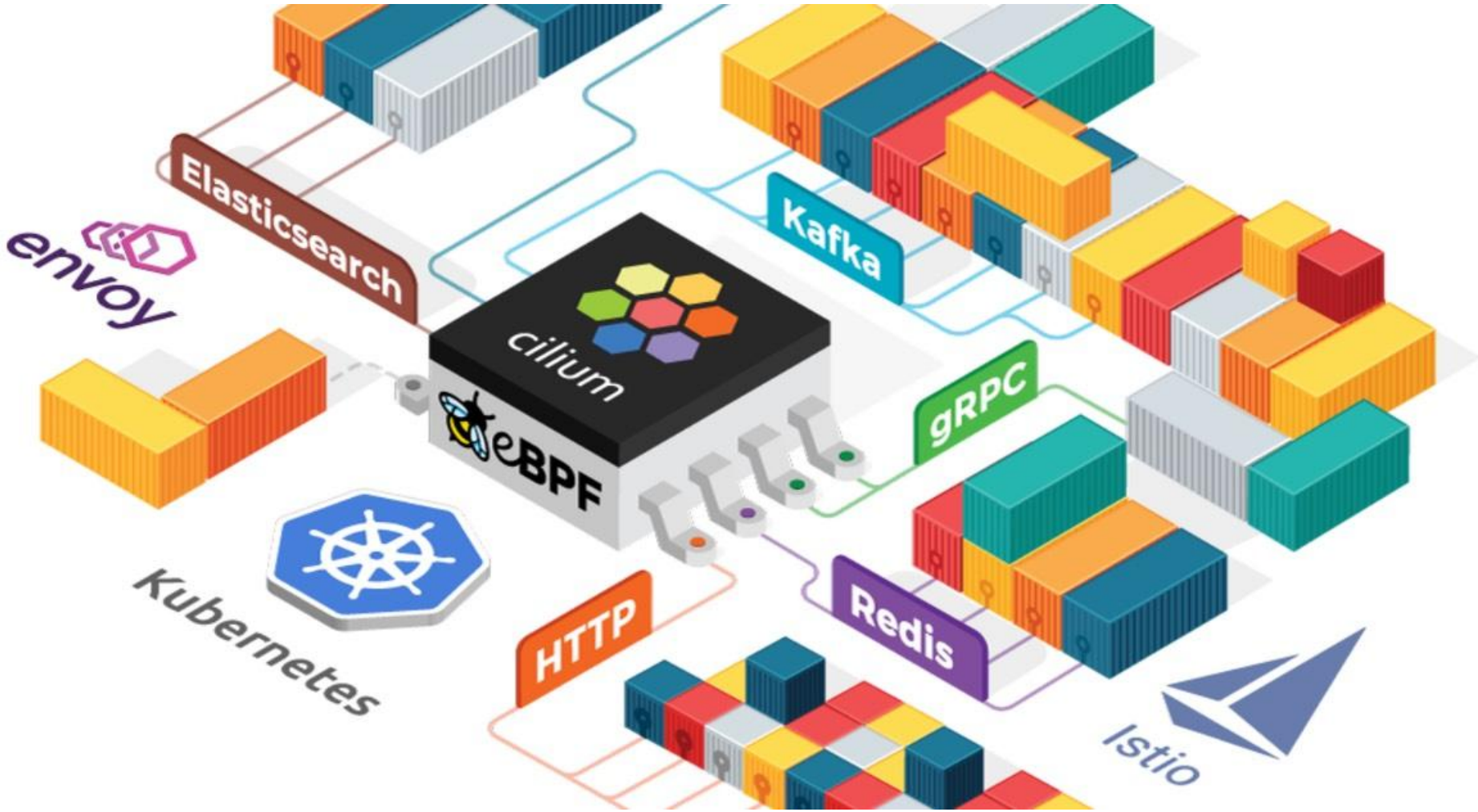
What the solution? (Istio ?)



Kubernetes for enterprise business

kubernetes by Google

Introduction to eBPF and Cilium

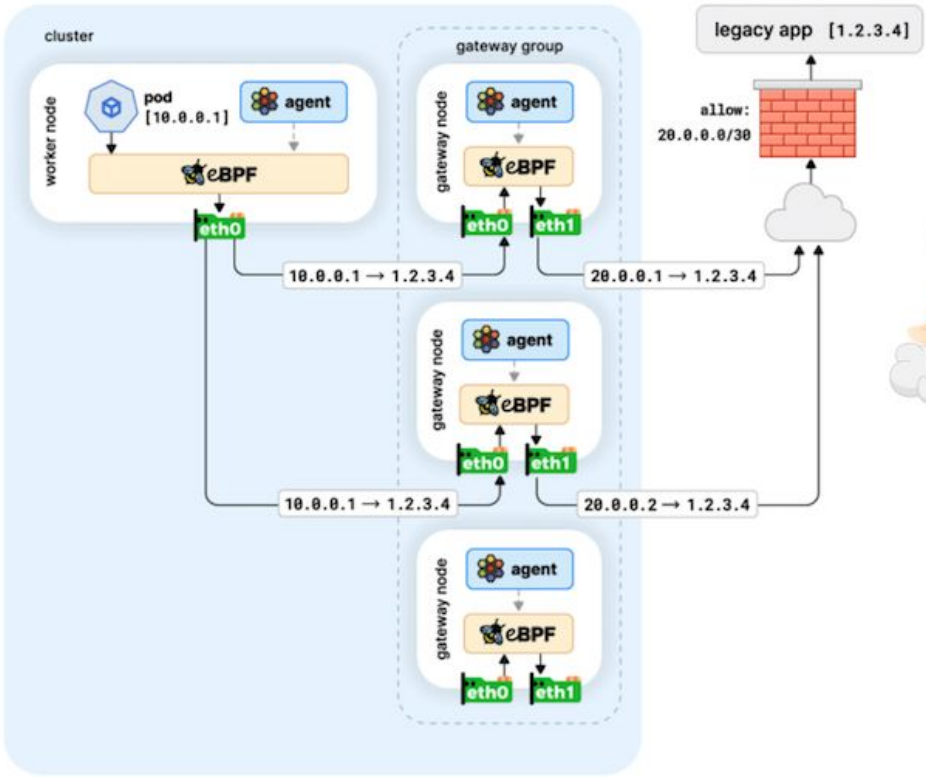


kubernetes by Google

Introduction to eBPF and Cilium

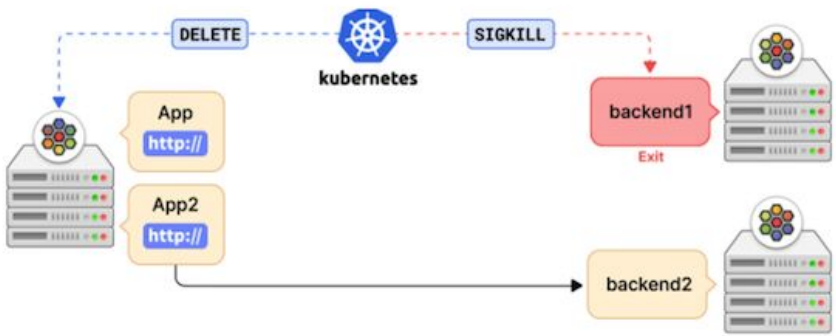
- Many problem of sidecar proxy and service mesh today is on world scale problem !!!
- Many year since istio launch and the performance issue is hard to avoid from architecture design
- **Since Oct,2021.** Cilium join CNCF as incubating project and apply "next generation" of service mesh management with 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



cilium

What's new in 1.11?



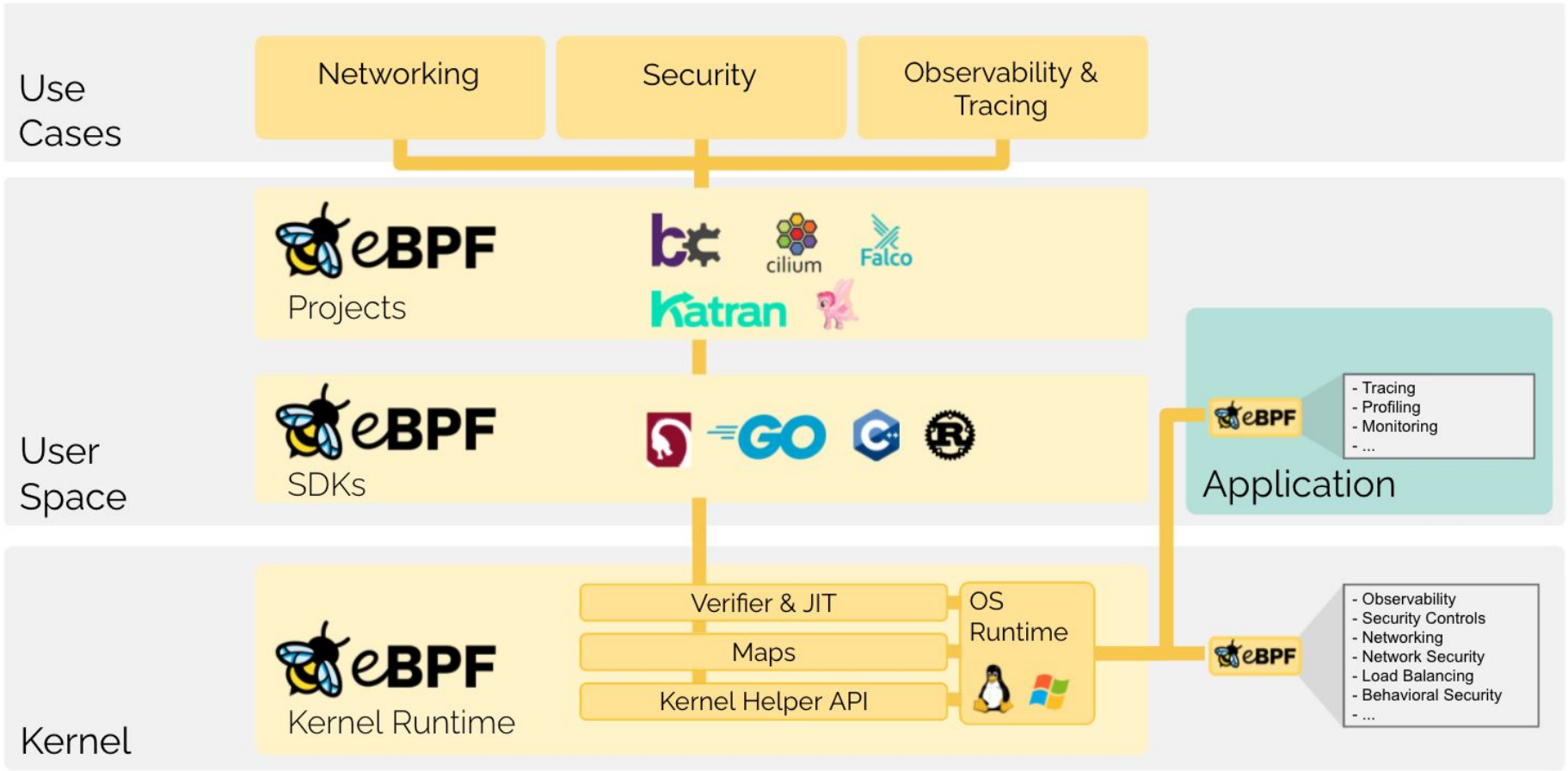
Kubernetes for enterprise business

kubernetes by Google

What is and Why eBPF ?

- eBPF is one technology that operation in linux kernel by kernel sandbox program feature
- This make us to run some program without need to upgrade kernel or install module
- eBPF are 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 ?

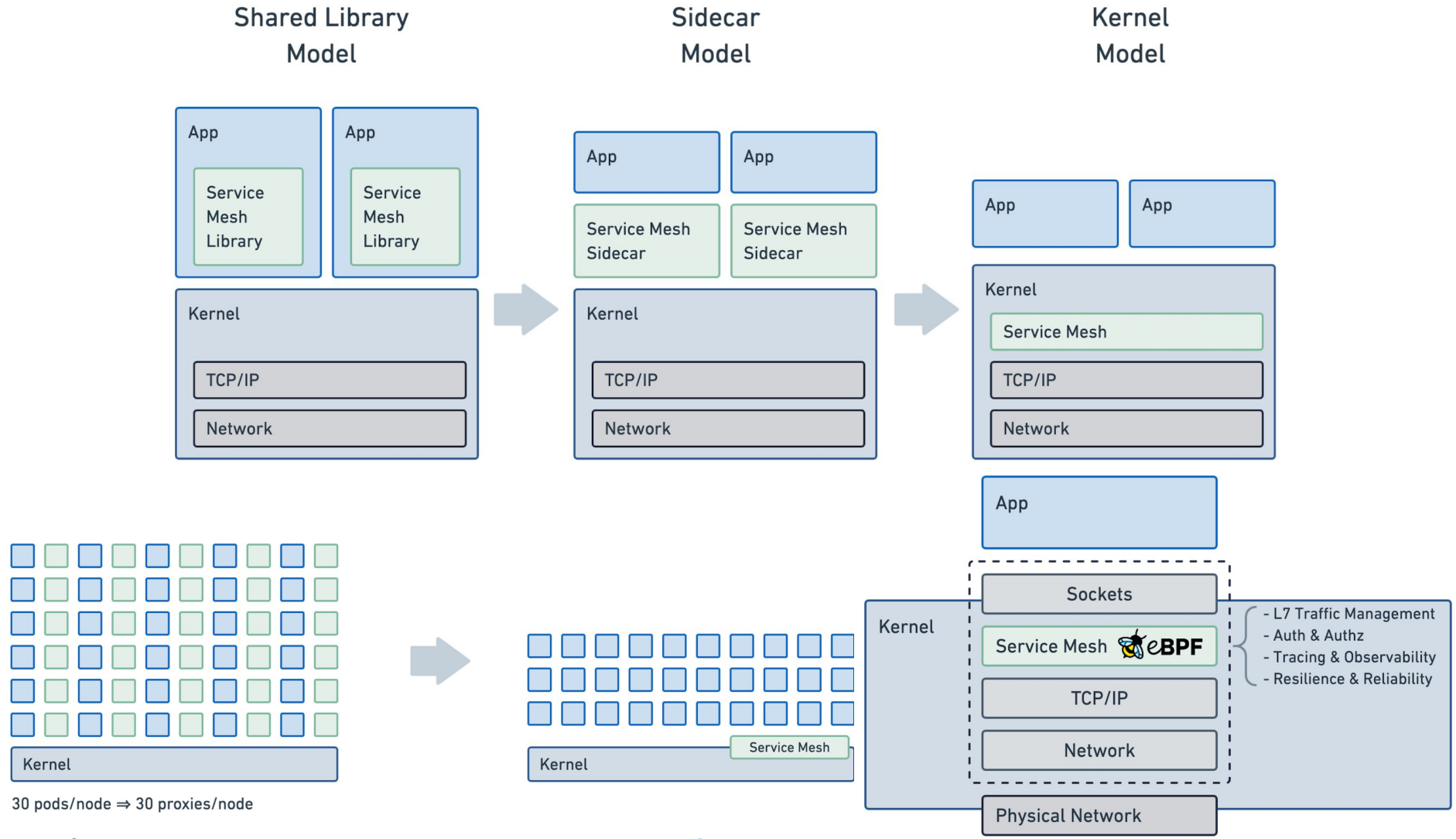


Ref: https://ebpf.io/

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

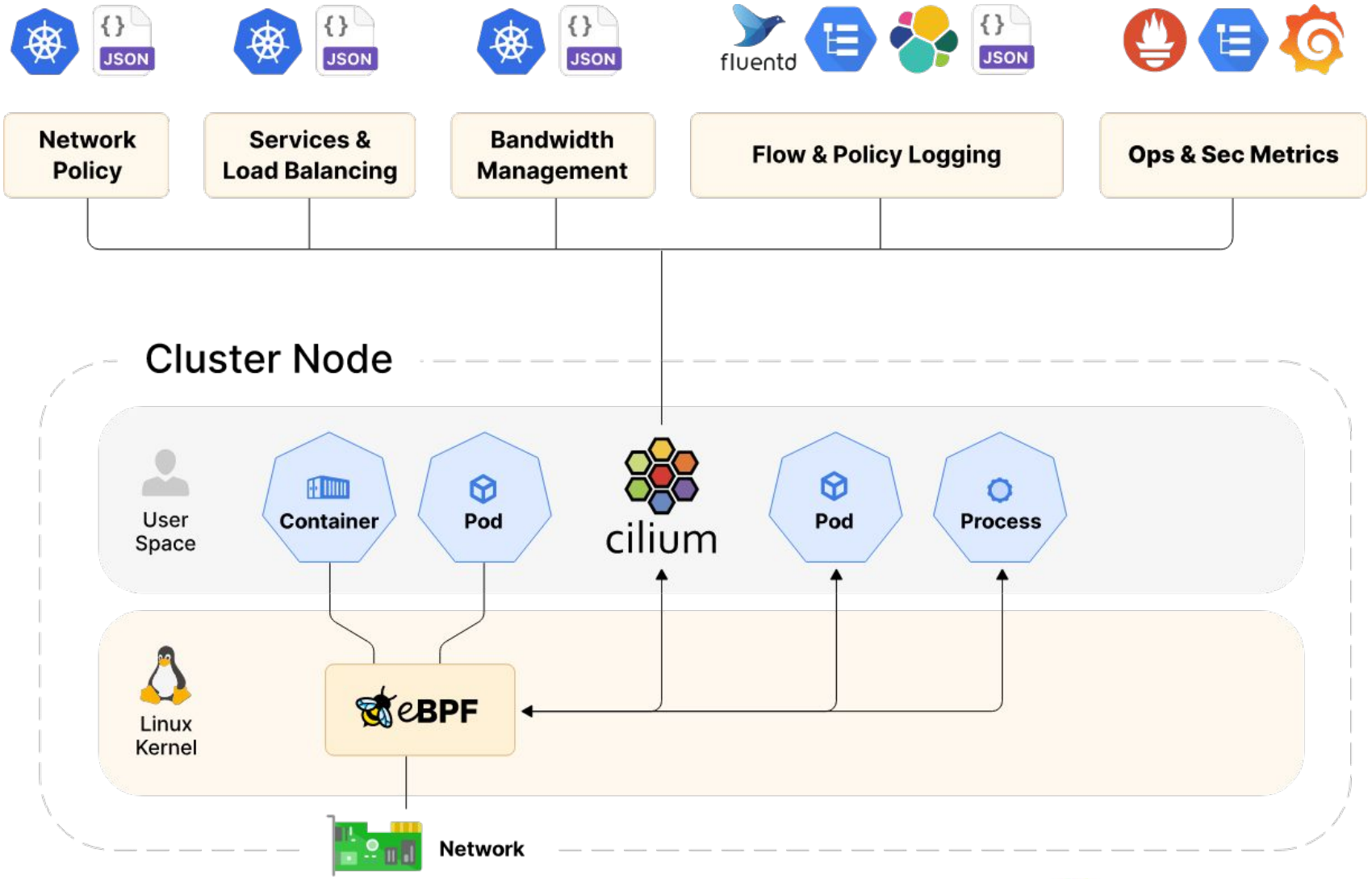
Kubernetes for enterprise business

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

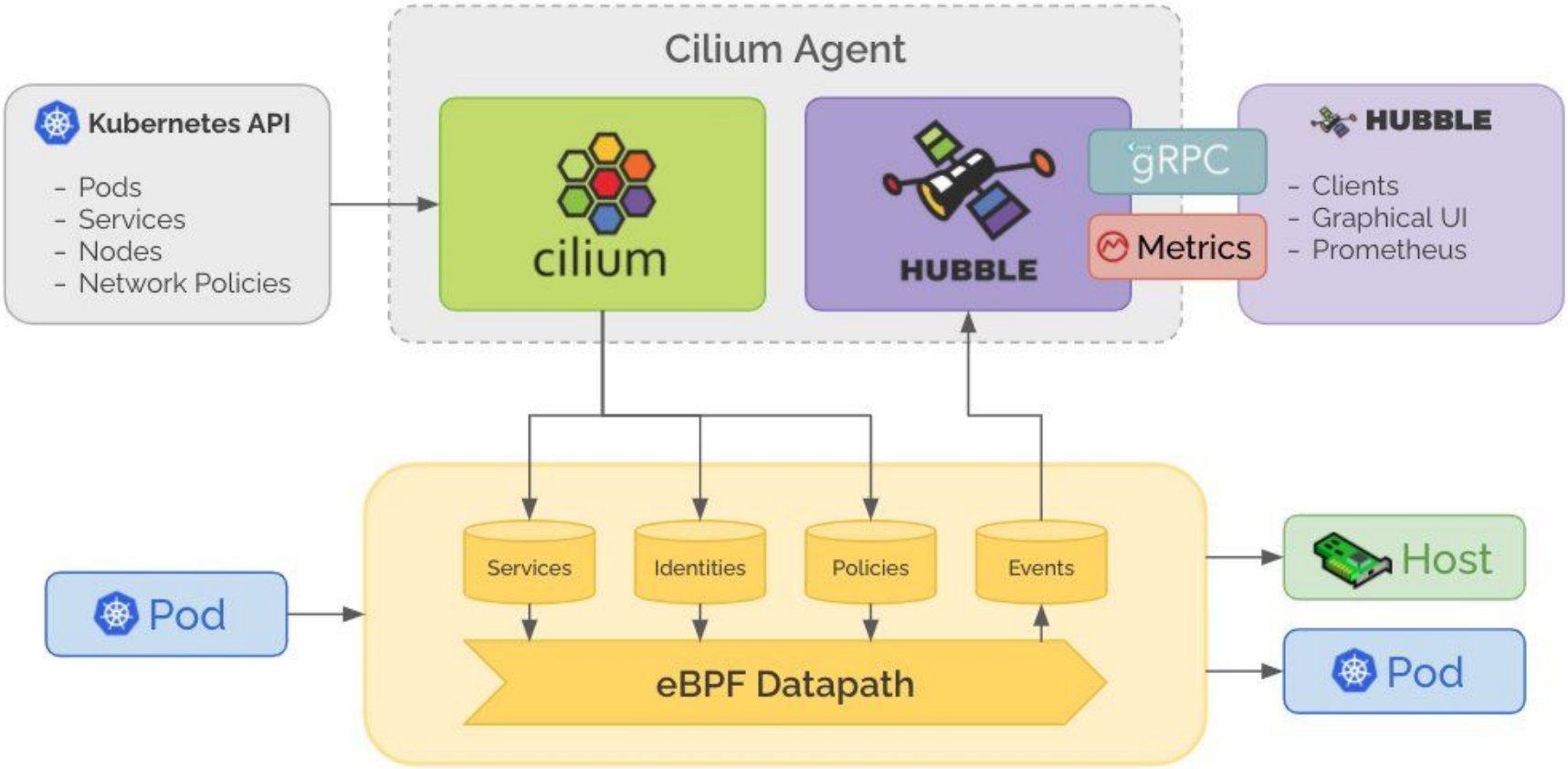
Cilium on Kubernetes Dataplane



Kubernetes for enterprise business

kubernetes by Google

Cilium on Kubernetes Dataplane



kubernetes by Google

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

Try Google Cloud

Start building on Google Cloud with $300 in free credits and 20+ always free products.

FREE TRIAL

***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.

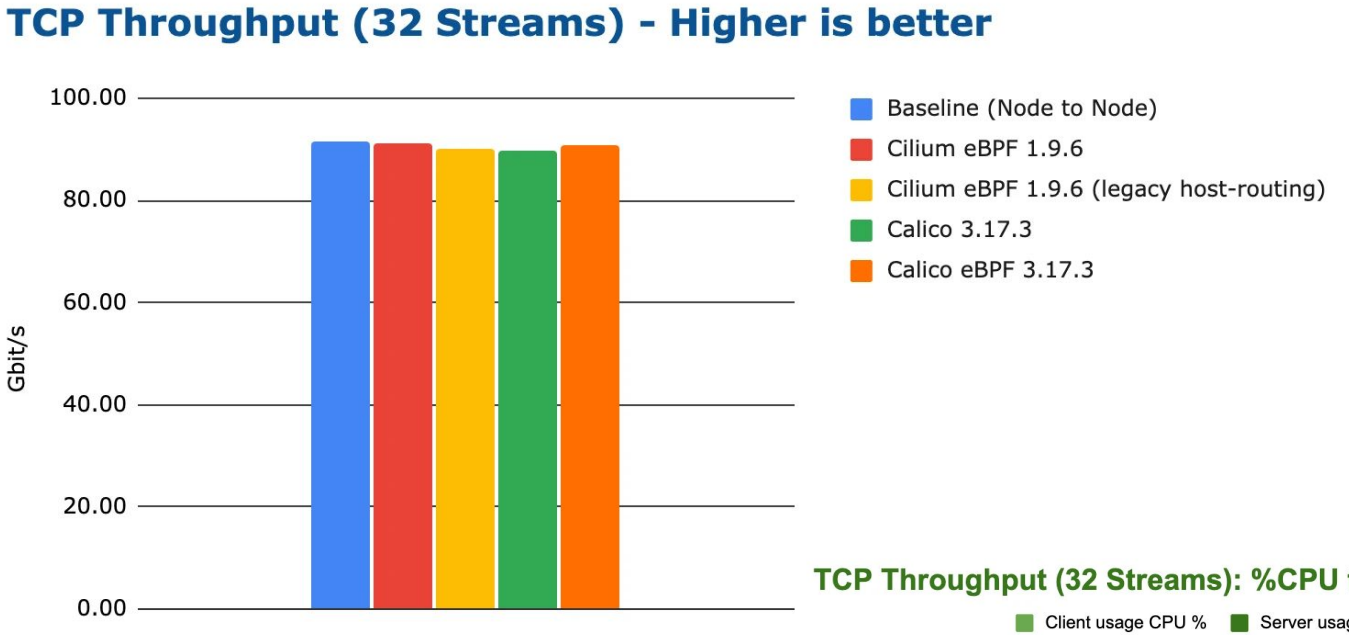
Today, we're introducing GKE Dataplane V2, an opinionated dataplane that harnesses the power of eBPF and Cilium, an open source project that makes the Linux kernel Kubernetes-aware using eBPF. Now in beta, we're also using Dataplane V2 to bring Kubernetes Network Policy logging to Google Kubernetes Engine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

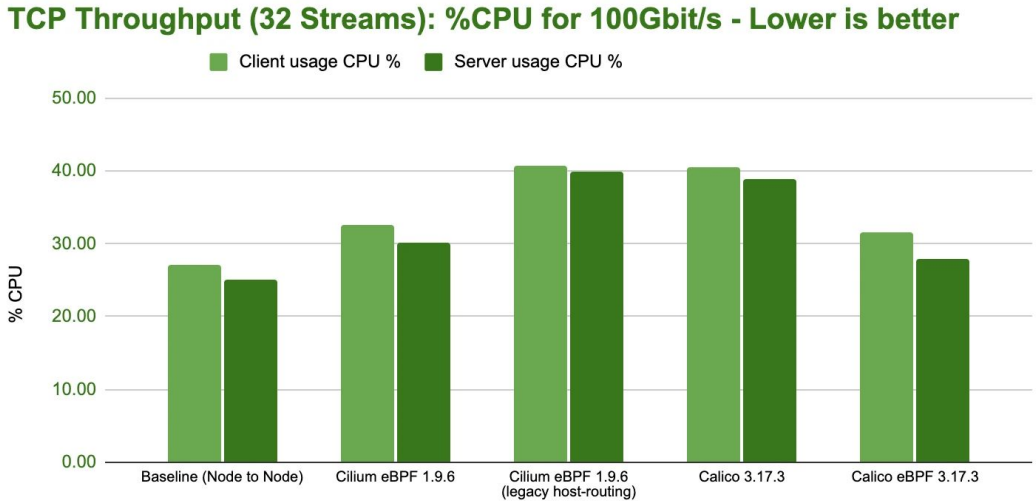
What are eBPF and Cilium?

Kubernetes for enterprise business

kubernetes by Google

Cilium on Kubernetes Dataplane





Kubernetes for enterprise business

kubernetes by Google

Cilium on Kubernetes Dataplane

Networking

Native support for service type Load Balancer and Egress

Scalable Kubernetes CNI

Multi-cluster Connectivity

Observability

Identity-aware Visibility

Advanced Self Service Observability

Network Metrics + Policy Troubleshooting

Security

Transparent Encryption

Security Forensics + Audit

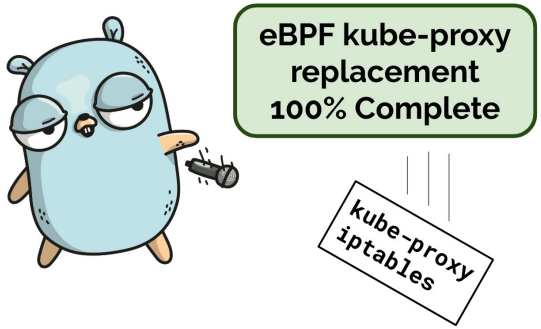
Advanced Network Policy

Kubernetes for enterprise business

kubernetes by Google

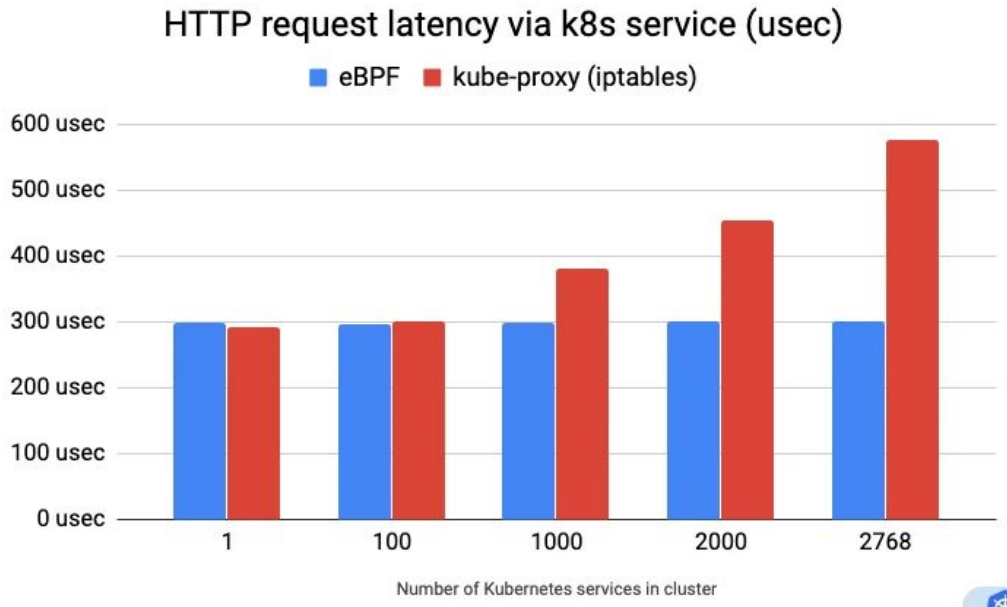
No kube-proxy with Cilium !!!

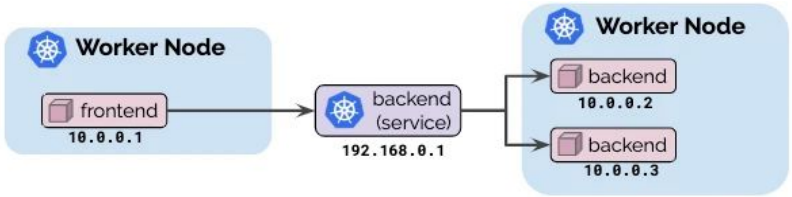
- Normally Kubernetes will handle network under hood with kube proxy/iptables.
- Base on idea of iptables operation. This also have some performance concern in case we have multiple service in single of node (Mean huge of iptables).
- As Cilium is base on eBPF and run on kernel. So it can have capability 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.

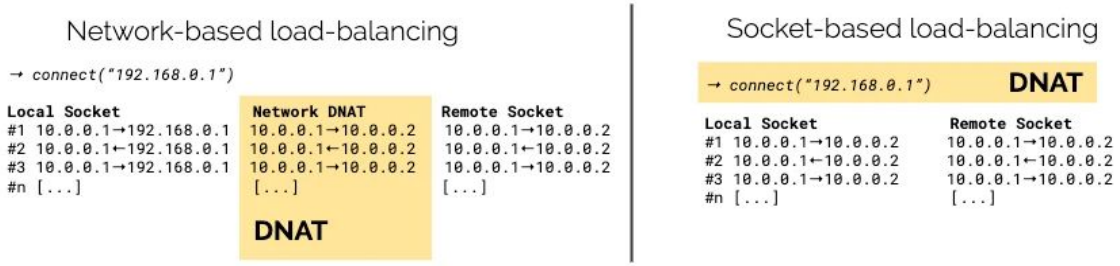


kubernetes by Google

No kube-proxy with Cilium !!!







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

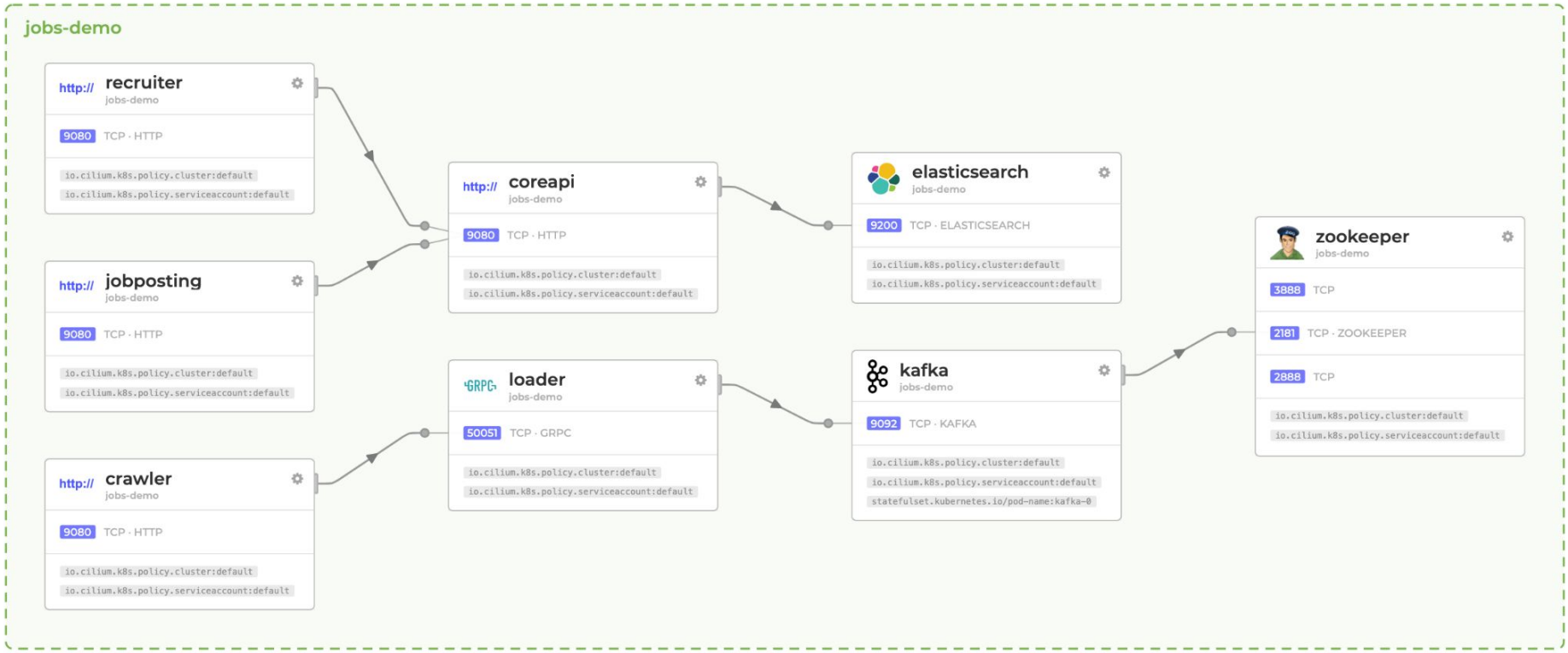
```
ubuntu@ip-10-21-1-233:~$ hubble observe
Feb 12 01:36:06.300: 10.0.4.83:14888 -> 10.0.3.10:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:06.556: 10.0.3.10:4240 <> 10.0.5.110:52348 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:06.556: 10.0.5.110:52348 -> 10.0.3.10:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:06.864: 10.0.3.142:15558 <> 10.0.0.25:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:06.870: 10.0.3.142:15558 -> 10.0.0.25:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:06.927: 10.0.3.142:58984 <> 10.0.4.148:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:06.935: 10.0.1.57:52076 <> 10.0.4.148:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:06.946: ingress-nginx/ingress-nginx-controller-8cf5559f8-h2sr7:443 <> 10.0.1.57:46590 to-overlay FORWARDED (TCP Flags: SYN, ACK)
Feb 12 01:36:06.946: 10.0.1.57:46590 -> ingress-nginx/ingress-nginx-controller-8cf5559f8-h2sr7:443 to-endpoint FORWARDED (TCP Flags: ACK, RST)
Feb 12 01:36:06.946: 10.0.1.57:47310 -> ingress-nginx/ingress-nginx-controller-8cf5559f8-h2sr7:80 to-endpoint FORWARDED (TCP Flags: ACK, RST)
Feb 12 01:36:06.959: 10.0.1.57:47310 <> ingress-nginx/ingress-nginx-controller-8cf5559f8-h2sr7:80 to-overlay FORWARDED (TCP Flags: SYN)
Feb 12 01:36:06.959: 10.0.1.57:46590 <> ingress-nginx/ingress-nginx-controller-8cf5559f8-h2sr7:443 to-overlay FORWARDED (TCP Flags: SYN)
Feb 12 01:36:06.960: 10.0.1.57:46590 <> ingress-nginx/ingress-nginx-controller-8cf5559f8-h2sr7:443 to-overlay FORWARDED (TCP Flags: ACK, RST)
Feb 12 01:36:06.960: 10.0.1.57:47310 <> ingress-nginx/ingress-nginx-controller-8cf5559f8-h2sr7:80 to-overlay FORWARDED (TCP Flags: ACK, RST)
Feb 12 01:36:07.019: 10.0.3.142:16908 <> 10.0.5.215:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.026: 10.0.1.57:48222 <> 10.0.5.215:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.036: 10.0.3.142:24910 <> 10.0.1.248:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.045: 10.0.1.248:4240 <> 10.0.3.142:24910 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.045: 10.0.1.57:14258 <- 10.0.1.248:4240 to-stack FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.045: 10.0.1.57:14258 -> 10.0.1.248:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.045: 10.0.3.142:24910 -> 10.0.1.248:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.068: 10.0.3.10:4240 <> 10.0.1.57:39218 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.068: 10.0.3.142:55422 <- 10.0.3.10:4240 to-stack FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.068: 10.0.3.142:55422 -> 10.0.3.10:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.068: 10.0.1.57:39218 -> 10.0.3.10:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.077: 10.0.1.57:39218 <> 10.0.3.10:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.083: 10.0.2.154:4240 <> 10.0.3.142:30542 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.083: 10.0.1.57:36562 -> 10.0.2.154:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.083: 10.0.3.142:30542 -> 10.0.2.154:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.084: 10.0.3.142:30542 <> 10.0.2.154:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.092: 10.0.1.57:36562 <> 10.0.2.154:4240 to-overlay FORWARDED (TCP Flags: ACK)
```

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

Kubernetes for enterprise business

kubernetes by Google

Hubble observability for all

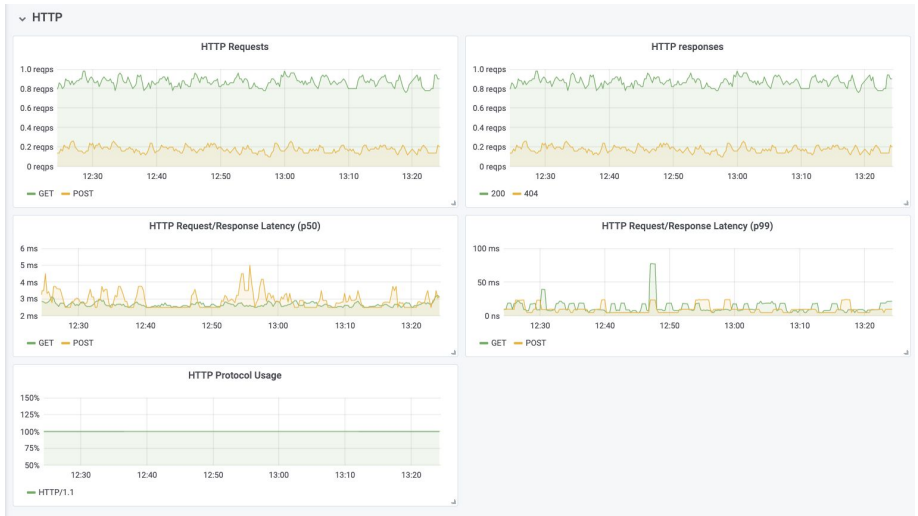


Ref:https://github.com/cilium/hubble

Kubernetes for enterprise business

kubernetes by Google

Hubble observability for all

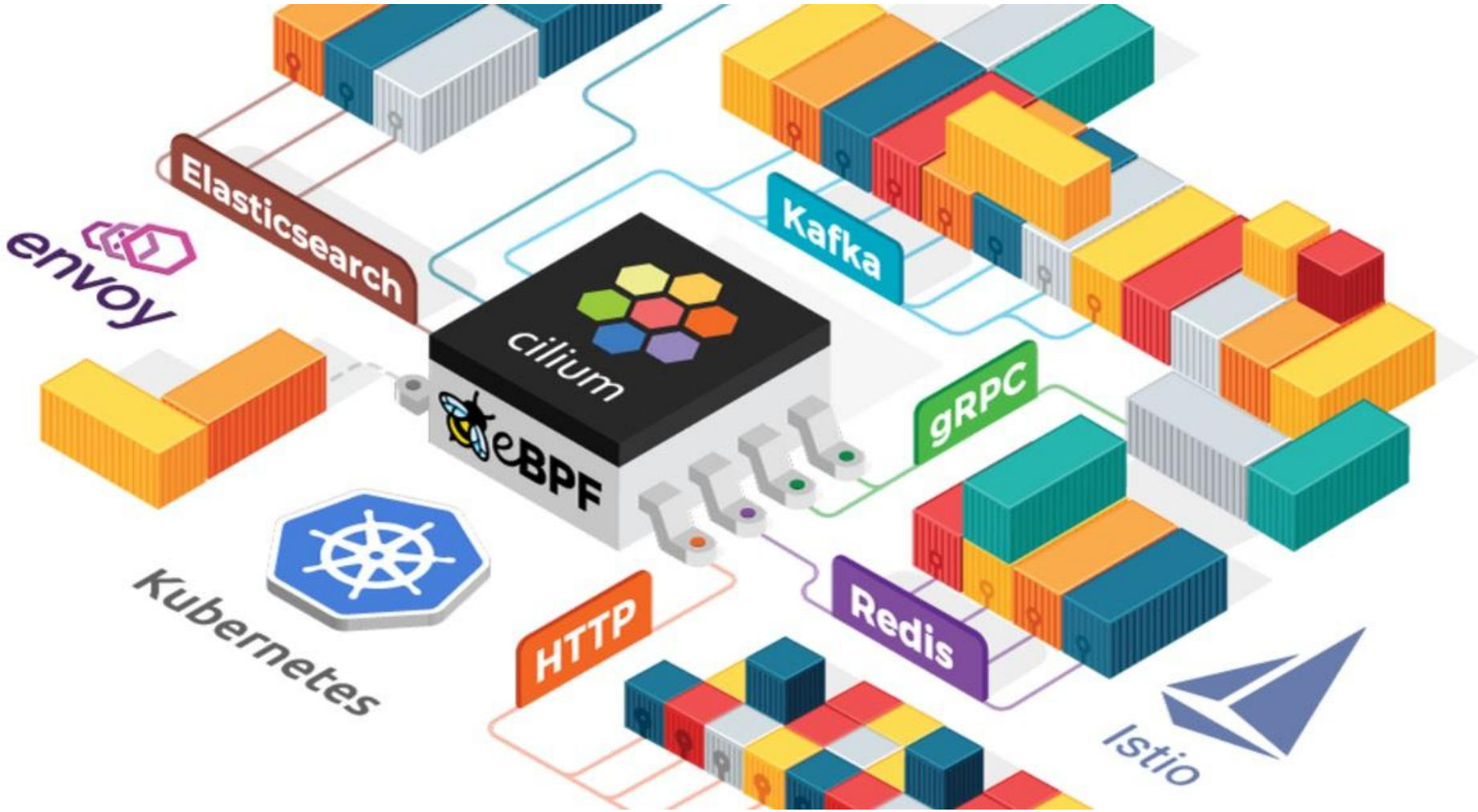


Ref:https://github.com/cilium/hubble

Kubernetes for enterprise business

kubernetes by Google

Demo Session



kubernetes by Google

Demo Session

- In Demo session. We will setup 3 part for demo hubble observability
- **Part 1: Setup Cilium & Hubble**
- **Part 2: Deploy application with single namespace:**
  - Basic rest api with 2 rest api connect in same database
- **Part 3: Deploy application with multiple namespace and apply security policy:**
  - Client namespace:
  - Frontend/Backend namespace:
  - Management namespace:

Part1: Install cilium & hubble

```
ubuntu@ip-10-21-1-233:~$ cilium install
ℹ️  using Cilium version "v1.11.0"
🔮 Auto-detected cluster name: kubernetesforenterpriseteacher
🔮 Auto-detected IPAM mode: cluster-pool
🔑 Found CA in secret cilium-ca
🔑 Generating certificates for Hubble...
🚀 Creating Service accounts...
🚀 Creating Cluster roles...
🚀 Creating ConfigMap for Cilium version 1.11.0...
🚀 Creating Agent DaemonSet...
🚀 Creating Operator Deployment...
⌛ Waiting for Cilium to be installed and ready...
✅ Cilium was successfully installed! Run 'cilium status' to view installation health
```

```
ubuntu@ip-10-21-1-233:~$ cilium status
    /¯¯\
 /¯¯\__/¯¯\    Cilium:         OK
 \__/¯¯\__/    Operator:       OK
 /¯¯\__/¯¯\    Hubble:         disabled
 \__/¯¯\__/    ClusterMesh:    disabled
    \__/

DaemonSet         cilium             Desired: 6, Ready: 6/6, Available: 6/6
Deployment        cilium-operator    Desired: 1, Ready: 1/1, Available: 1/1
Containers:       cilium             Running: 6
                  cilium-operator    Running: 1
Cluster Pods:     14/16 managed by Cilium
Image versions    cilium             quay.io/cilium/cilium:v1.11.0: 6
                  cilium-operator    quay.io/cilium/operator-generic:v1.11.0: 1
ubuntu@ip-10-21-1-233:~$ kubectl get nodes
NAME                                              STATUS   ROLES                  AGE   VERSION
ip-10-21-1-115.ap-southeast-1.compute.internal    Ready    worker                 27d   v1.22.1
ip-10-21-1-160.ap-southeast-1.compute.internal    Ready    control-plane,master   27d   v1.22.1
ip-10-21-1-195.ap-southeast-1.compute.internal    Ready    control-plane,master   27d   v1.22.1
ip-10-21-1-233.ap-southeast-1.compute.internal    Ready    control-plane,master   27d   v1.22.1
ip-10-21-1-44.ap-southeast-1.compute.internal     Ready    worker                 27d   v1.22.1
ip-10-21-1-53.ap-southeast-1.compute.internal     Ready    worker                 27d   v1.22.1
```

Part1: Install cilium & hubble

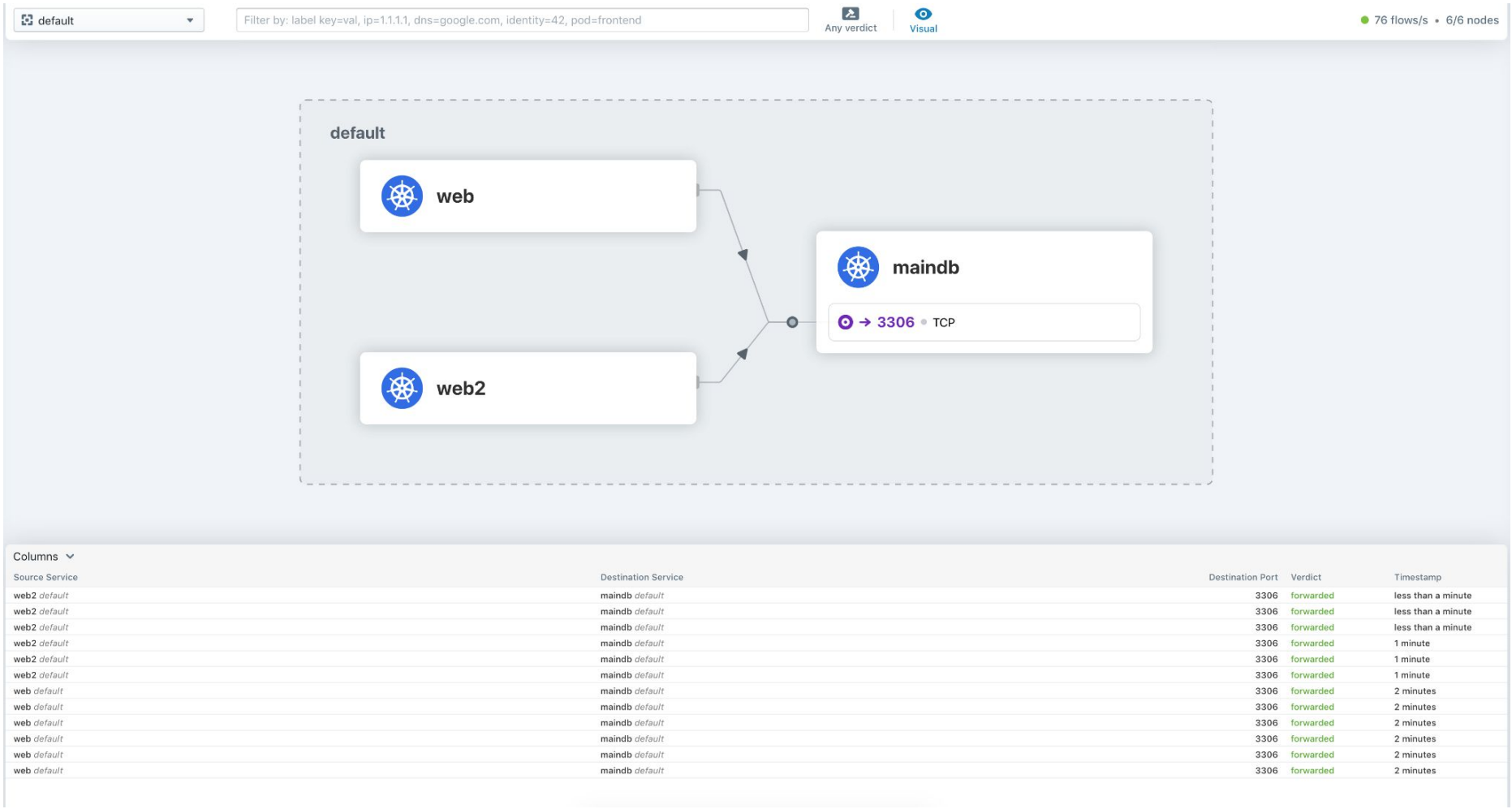
```
ubuntu@ip-10-21-1-233:~$ cilium hubble enable --ui
🔑 Found CA in secret cilium-ca
✨ Patching ConfigMap cilium-config to enable Hubble...
♻️ Restarted Cilium pods
⌛ Waiting for Cilium to become ready before deploying other Hubble component(s)...
🔑 Generating certificates for Relay...
✨ Deploying Relay from quay.io/cilium/hubble-relay:v1.11.0...
✨ Deploying Hubble UI from quay.io/cilium/hubble-ui:v0.8.3 and Hubble UI Backend from quay.io/cilium/hubble-ui-backend:v0.8.3...
⌛ Waiting for Hubble to be installed...
✅ Hubble was successfully enabled!
ubuntu@ip-10-21-1-233:~$ cilium status
    /¯¯\
 /¯¯\__/¯¯\    Cilium:         OK
 \__/¯¯\__/    Operator:       OK
 /¯¯\__/¯¯\    Hubble:         OK
 \__/¯¯\__/    ClusterMesh:    disabled
    \__/

DaemonSet         cilium             Desired: 6, Ready: 6/6, Available: 6/6
Deployment        cilium-operator    Desired: 1, Ready: 1/1, Available: 1/1
Deployment        hubble-relay       Desired: 1, Ready: 1/1, Available: 1/1
Deployment        hubble-ui          Desired: 1, Ready: 1/1, Available: 1/1
Containers:       cilium-operator    Running: 1
                  hubble-relay       Running: 1
                  hubble-ui          Running: 1
                  cilium             Running: 6
Cluster Pods:     16/18 managed by Cilium
Image versions    cilium-operator    quay.io/cilium/operator-generic:v1.11.0: 1
                  hubble-relay       quay.io/cilium/hubble-relay:v1.11.0: 1
                  hubble-ui          quay.io/cilium/hubble-ui:v0.8.3: 1
                  hubble-ui          quay.io/cilium/hubble-ui-backend:v0.8.3: 1
                  hubble-ui          docker.io/envoyproxy/envoy:v1.18.2@sha256:e8b37c1d75787dd1e712ff389b0d37337dc8a174a63bed9c34ba73359dc67da7: 1
                  cilium             quay.io/cilium/cilium:v1.11.0: 6
```

Part1: Install cilium & hubble

```
ubuntu@ip-10-21-1-233:~$ hubble observe
Feb 12 01:36:06.300: 10.0.4.83:14888 -> 10.0.3.10:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:06.556: 10.0.3.10:4240 <> 10.0.5.110:52348 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:06.556: 10.0.5.110:52348 -> 10.0.3.10:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:06.864: 10.0.3.142:15558 <> 10.0.0.25:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:06.870: 10.0.3.142:15558 -> 10.0.0.25:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:06.927: 10.0.3.142:58984 <> 10.0.4.148:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:06.935: 10.0.1.57:52076 <> 10.0.4.148:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:06.946: ingress-nginx/ingress-nginx-controller-8cf5559f8-h2sr7:443 <> 10.0.1.57:46590 to-overlay FORWARDED (TCP Flags: SYN, ACK)
Feb 12 01:36:06.946: 10.0.1.57:46590 -> ingress-nginx/ingress-nginx-controller-8cf5559f8-h2sr7:443 to-endpoint FORWARDED (TCP Flags: ACK, RST)
Feb 12 01:36:06.946: 10.0.1.57:47310 -> ingress-nginx/ingress-nginx-controller-8cf5559f8-h2sr7:80 to-endpoint FORWARDED (TCP Flags: ACK, RST)
Feb 12 01:36:06.959: 10.0.1.57:47310 <> ingress-nginx/ingress-nginx-controller-8cf5559f8-h2sr7:80 to-overlay FORWARDED (TCP Flags: SYN)
Feb 12 01:36:06.959: 10.0.1.57:46590 <> ingress-nginx/ingress-nginx-controller-8cf5559f8-h2sr7:443 to-overlay FORWARDED (TCP Flags: SYN)
Feb 12 01:36:06.960: 10.0.1.57:46590 <> ingress-nginx/ingress-nginx-controller-8cf5559f8-h2sr7:443 to-overlay FORWARDED (TCP Flags: ACK, RST)
Feb 12 01:36:06.960: 10.0.1.57:47310 <> ingress-nginx/ingress-nginx-controller-8cf5559f8-h2sr7:80 to-overlay FORWARDED (TCP Flags: ACK, RST)
Feb 12 01:36:07.019: 10.0.3.142:16908 <> 10.0.5.215:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.026: 10.0.1.57:48222 <> 10.0.5.215:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.036: 10.0.3.142:24910 <> 10.0.1.248:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.045: 10.0.1.248:4240 <> 10.0.3.142:24910 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.045: 10.0.1.57:14258 <- 10.0.1.248:4240 to-stack FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.045: 10.0.1.57:14258 -> 10.0.1.248:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.045: 10.0.3.142:24910 -> 10.0.1.248:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.068: 10.0.3.10:4240 <> 10.0.1.57:39218 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.068: 10.0.3.142:55422 <- 10.0.3.10:4240 to-stack FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.068: 10.0.3.142:55422 -> 10.0.3.10:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.068: 10.0.1.57:39218 -> 10.0.3.10:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.077: 10.0.1.57:39218 <> 10.0.3.10:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.083: 10.0.2.154:4240 <> 10.0.3.142:30542 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.083: 10.0.1.57:36562 -> 10.0.2.154:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.083: 10.0.3.142:30542 -> 10.0.2.154:4240 to-endpoint FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.084: 10.0.3.142:30542 <> 10.0.2.154:4240 to-overlay FORWARDED (TCP Flags: ACK)
Feb 12 01:36:07.092: 10.0.1.57:36562 <> 10.0.2.154:4240 to-overlay FORWARDED (TCP Flags: ACK)
```

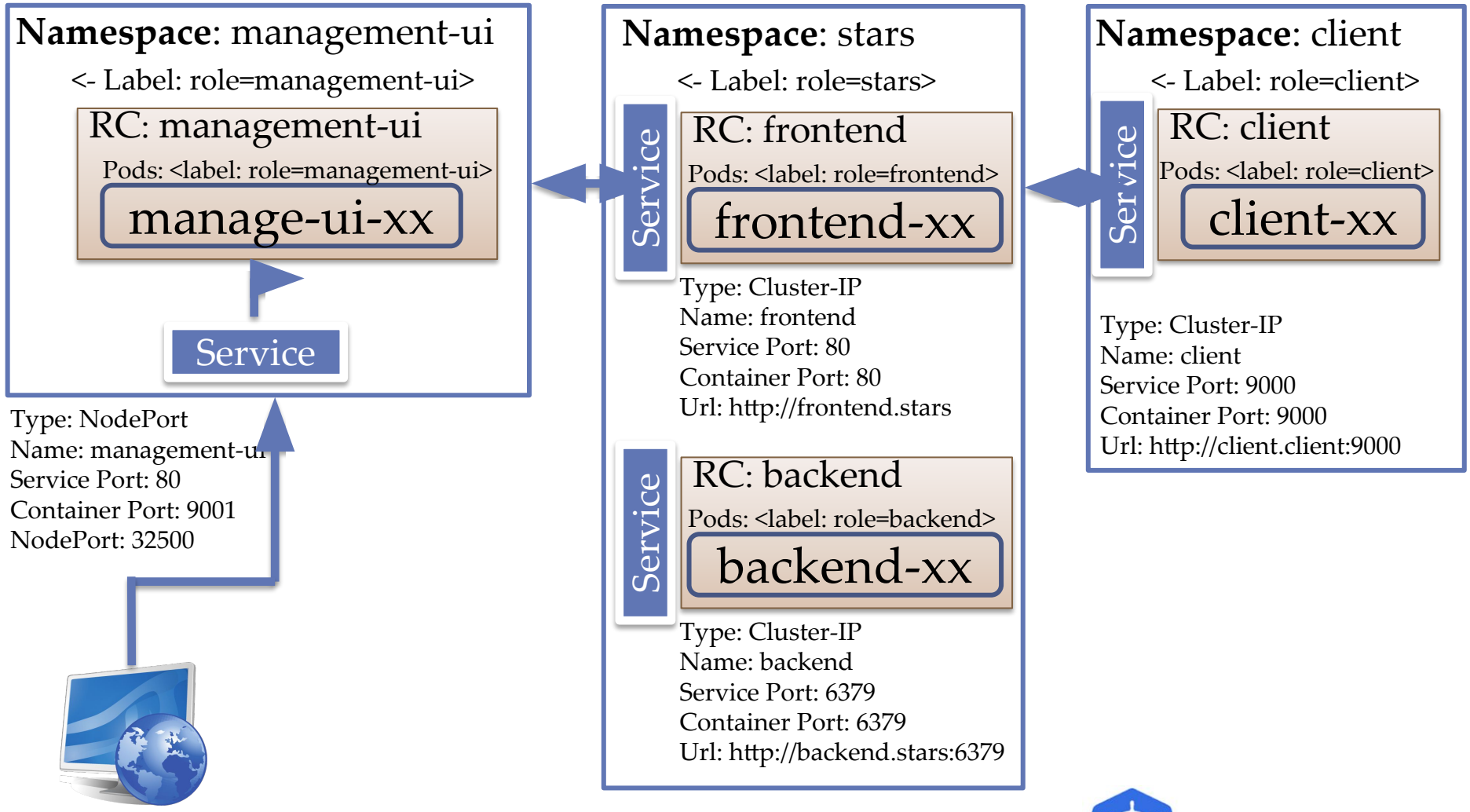
Part 2: Deploy application with single ns



Kubernetes for enterprise business

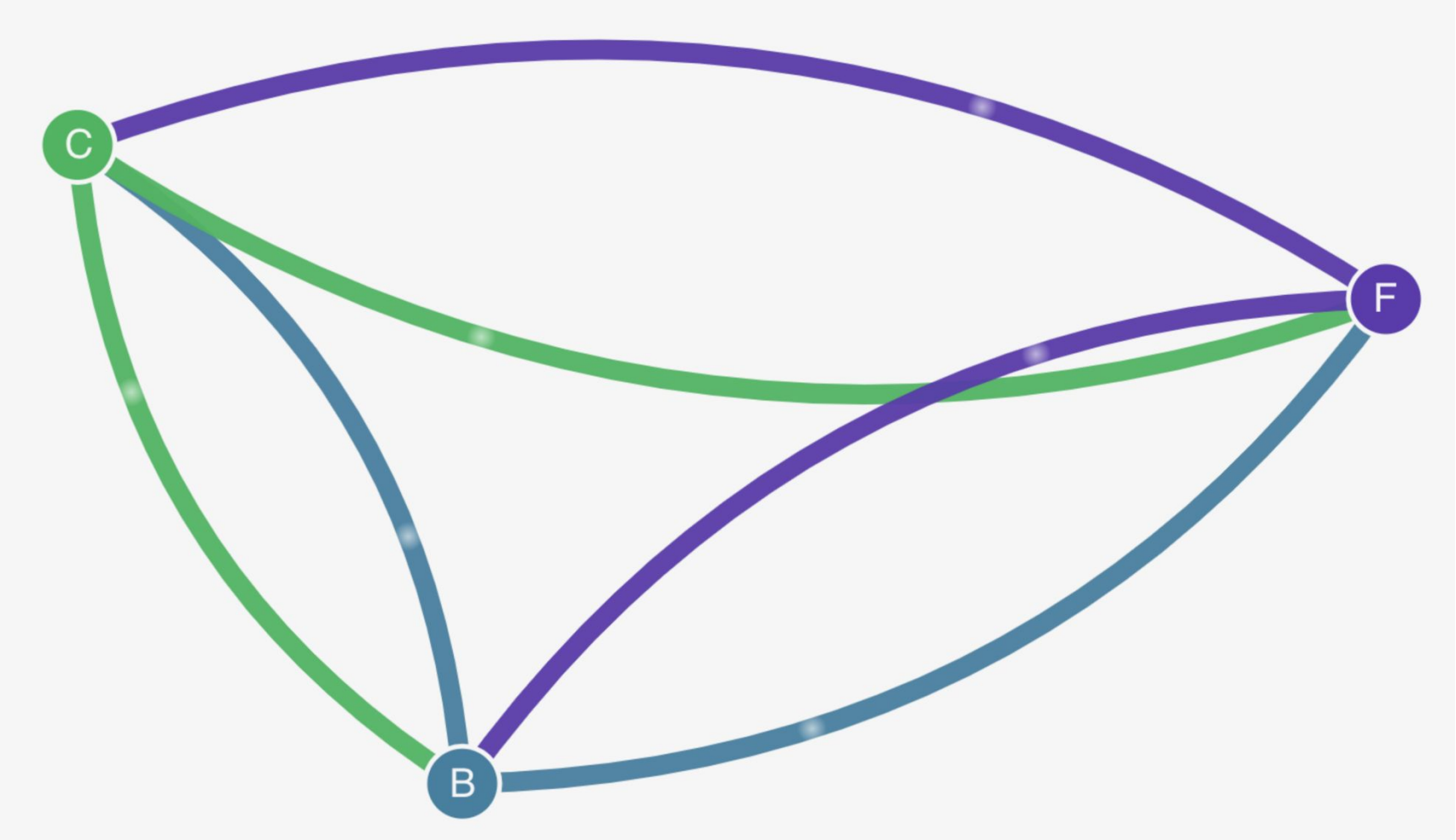
kubernetes by Google

Part 3: Deploy app with multiple namespace



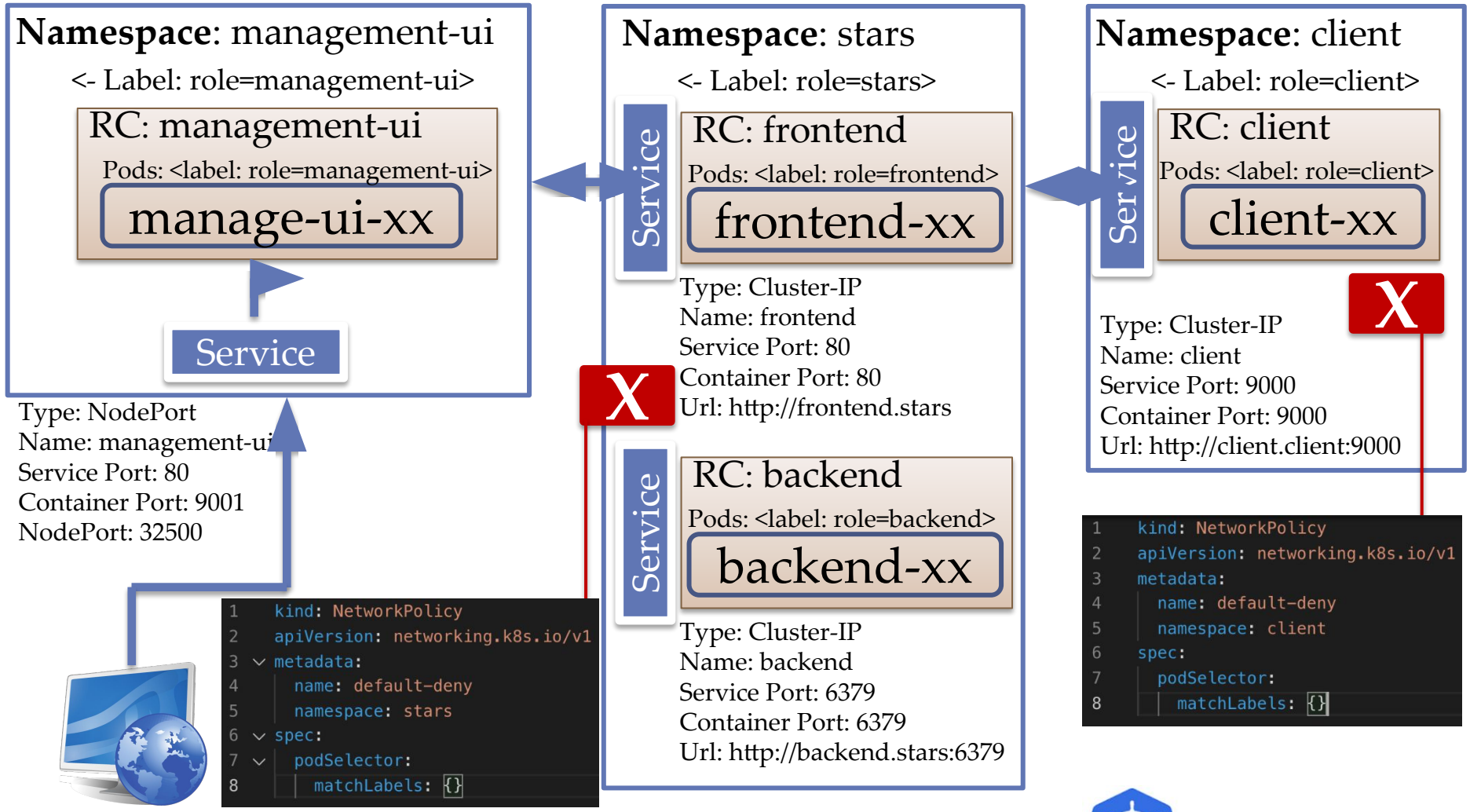
kubernetes by Google

Part 3: Deploy app with multiple namespace



kubernetes by Google

Part 3: Deploy app with multiple namespace



Kubernetes: Production Workload Orchestration

kubernetes by Google

Part 3: Deploy app with multiple namespace

kubernetes by Google

Part 3: Deploy app with multiple namespace

Namespace: management-ui

<- Label: role=management-ui>

RC: management-ui

Pods: <label: role=management-ui>

manage-ui-xx

Service

Type: NodePort
Name: management-ui
Service Port: 80
Container Port: 9001
NodePort: 32500

```
kind: NetworkPolicy
apiVersion: extensions/v1beta1
metadata:
  namespace: stars
  name: allow-ui
spec:
  podSelector:
    matchLabels: {}
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              role: management-ui
```

Namespace: stars

<- Label: role=stars>

Service

RC: frontend

Pods: <label: role=frontend>

frontend-xx

X

Type: Cluster-IP
Name: frontend
Service Port: 80
Container Port: 80
Url: http://frontend.stars

Y

Service

RC: backend

Pods: <label: role=backend>

backend-xx

Type: Cluster-IP
Name: backend
Service Port: 6379
Container Port: 6379
Url: http://backend.stars:6379

Namespace: client

<- Label: role=client>

Service

RC: client

Pods: <label: role=client>

client-xx

X

Type: Cluster-IP
Name: client
Service Port: 9000
Container Port: 9000
Url: http://client.client:9000

Y

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: client
  name: allow-ui
spec:
  podSelector:
    matchLabels: {}
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              role: management-ui
```

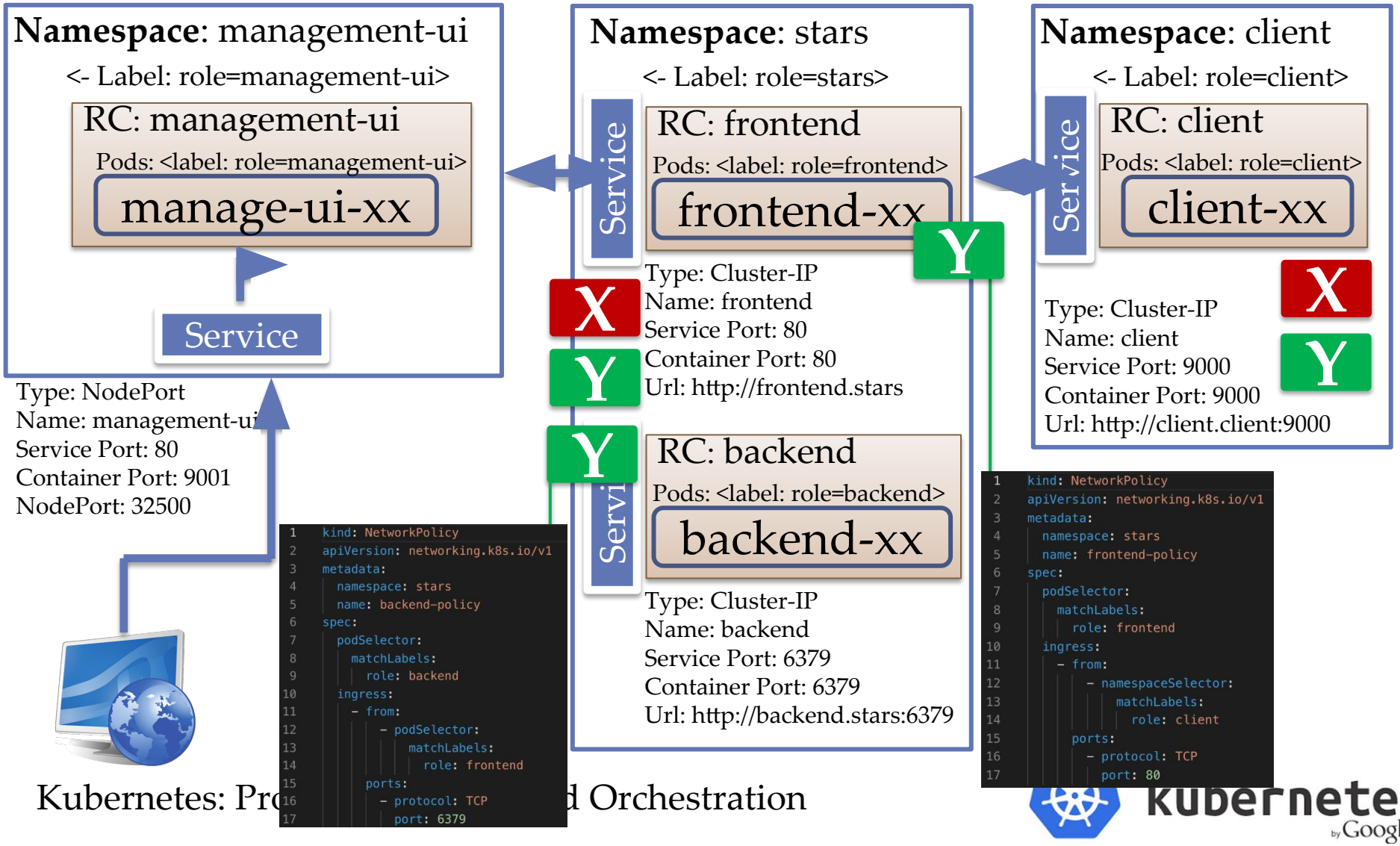
Kubernetes: P d Orchestration

kubernetes by Google

Part 3: Deploy app with multiple namespace

kubernetes by Google

Part 3: Deploy app with multiple namespace



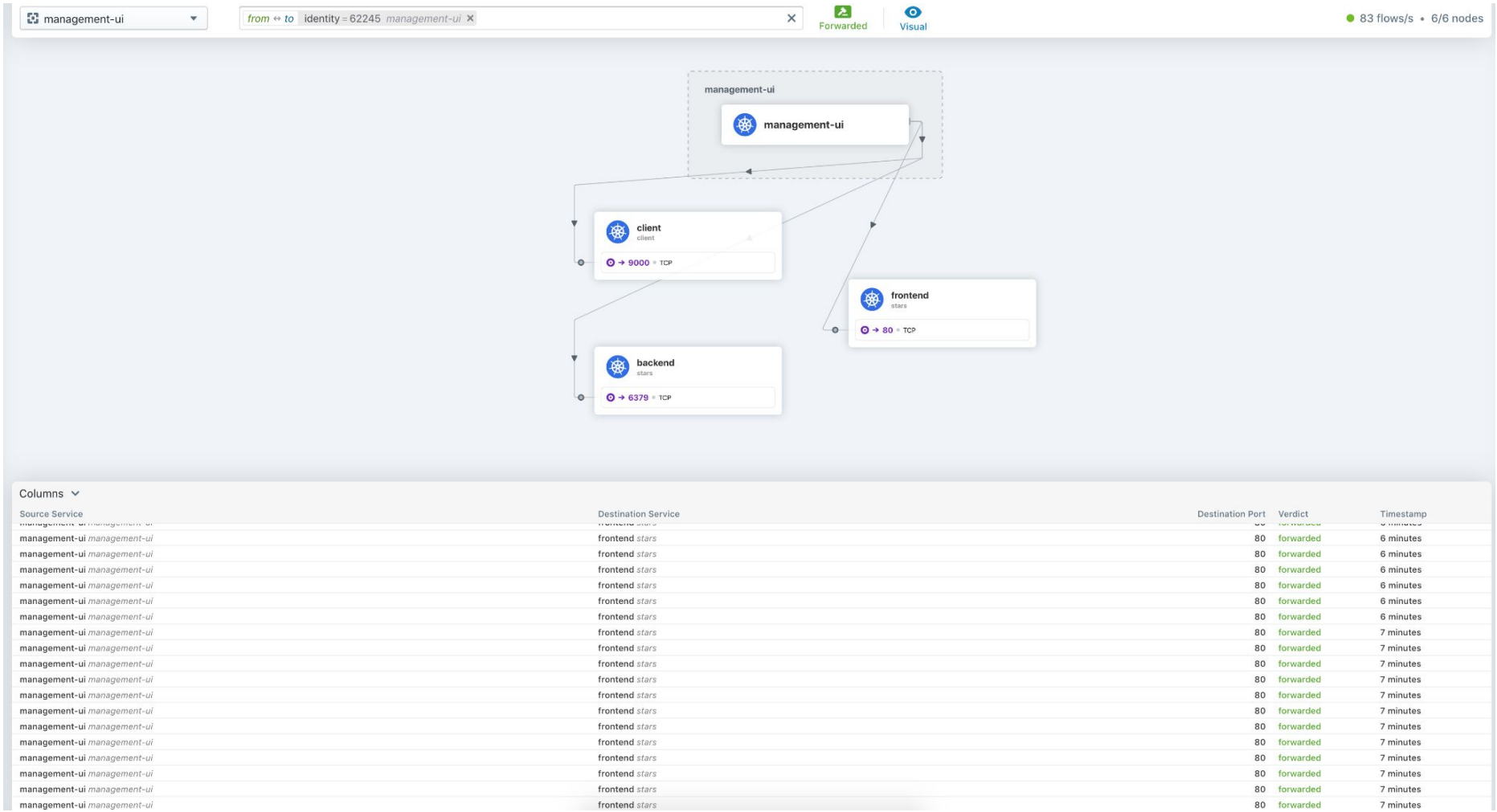
Kubernetes: Pro d Orchestration

kubernetes by Google

Part 3: Deploy app with multiple namespace

kubernetes by Google

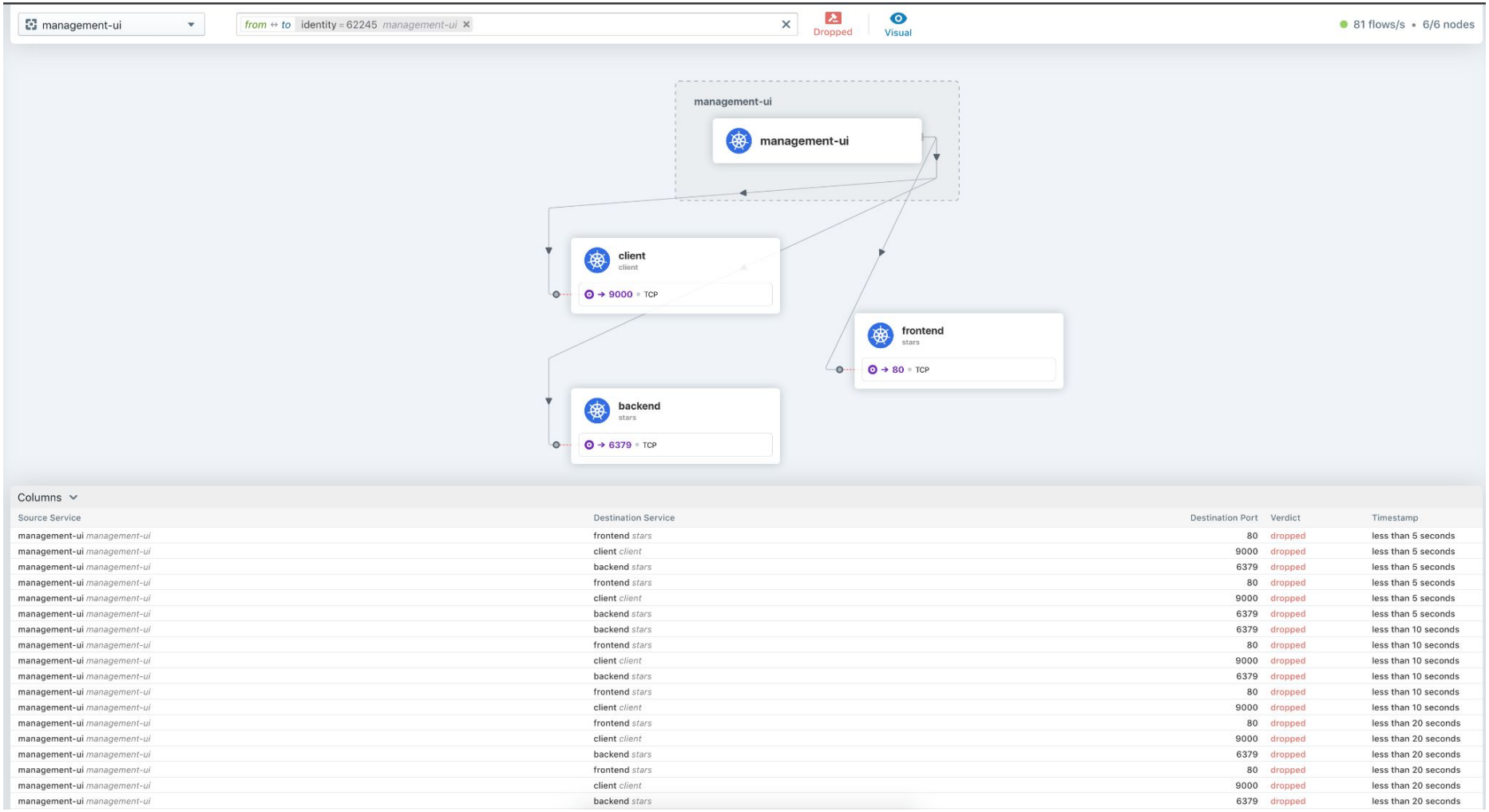
Part 3: Deploy app with multiple namespace



Kubernetes for enterprise business

kubernetes by Google

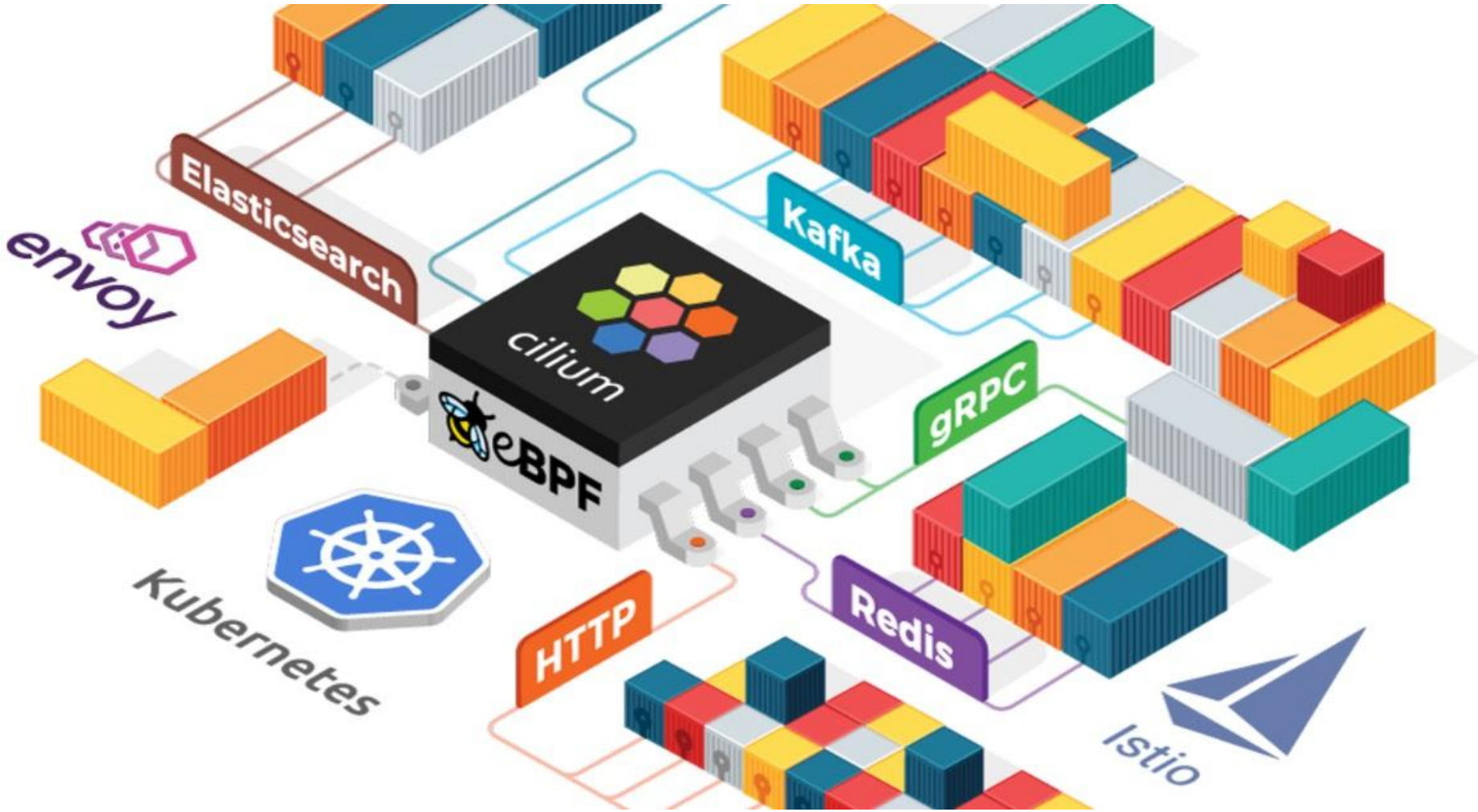
Part 3: Deploy app with multiple namespace



Kubernetes for enterprise business

kubernetes by Google

Q&A



kubernetes by Google