
KubeCon
CloudNativeCon
North America 2024

KubeCon

CloudNativeCon

North America 2024

Understanding How OpenTelemetry Network Uses eBPF for Network Observability

Jonathan Perry, PerfPod
Shivanshu Raj Shrivastava, SigNoz

KubeCon CloudNativeCon
North America 2024

About the speakers

Jonathan Perry

Founder & CEO PerfPod
Prev. CEO FlowMill
Maintainer opentelemetry-network

Shivanshu Raj Shrivastava

Founding Engineer SigNoz (YC W21)
CNCF ambasador
Member & contributor opentelemetry

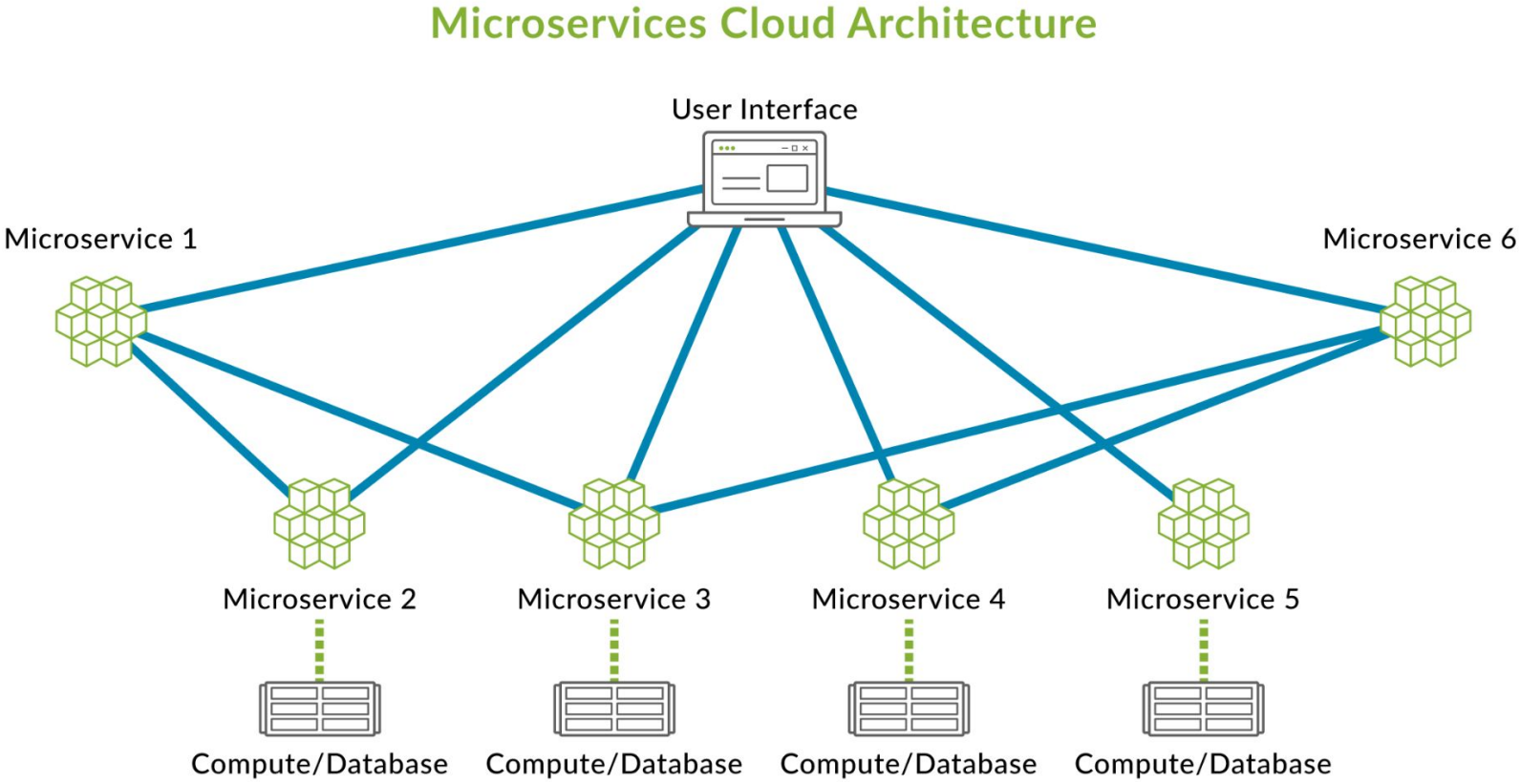
Talk Outline

Talk Outline

- Why **Network observability** is important, and why use **eBPF**?
- **What is *eBPF*** and how to use it for **network observability**?
- ***Tooling*** available to facilitate network observability via eBPF
- Understanding Open Telemetry Network ***architecture***
- ***Deep dive*** into individual components and agents of OTeL Network
- Demo
- How to get involved?

Why is Network observability important, and why use eBPF?

Why Network observability is important, and why use eBPF?



Why Network observability is important, and why use eBPF?

-> Modern distributed microservice and hybrid system architecture complex

-> They are introducing novel challenges

-> Traditional Telemetry is not sufficient to get complete visibility into your system

Why Network observability is important, and why use eBPF?

-> **Complex** Troubleshooting and Prolonged Downtime

-> **Limited Visibility** into Network Performance

-> Ineffective Detection of Anomalies and **Security Threats**

-> **Lack of Insight** into User Experience and Application Performance

-> **Compliance** and Audit Challenges

-> Less visibility into **open network ports**

-> **Network Cost Analysis is hard**

-> Detecting **networking failures at the cloud provider** and at **external vendor's apis is hard**

What's possible with Network observability?

Why Network observability is important, and why use eBPF?

-> Enhanced Security Posture

-> Optimize Network Performance

-> Diagnosing Packet Loss

-> Resolving High Latency Issues

-> User and Device Identification

-> Protocol-Specific Information (TCP Metrics, UDP Statistics)

-> Flow Data (NetFlow/sFlow/IPFIX Record)

-> Visibility into third party API failures

Why use eBPF for Network observability?

Why Network observability is important, and why use eBPF?

-> Low Overhead Monitoring

- Efficient Data Collection
- Minimal System Impact

-> Deep Visibility into Kernel and Application Behavior

- Access to Kernel Events
- Application-Level Insights

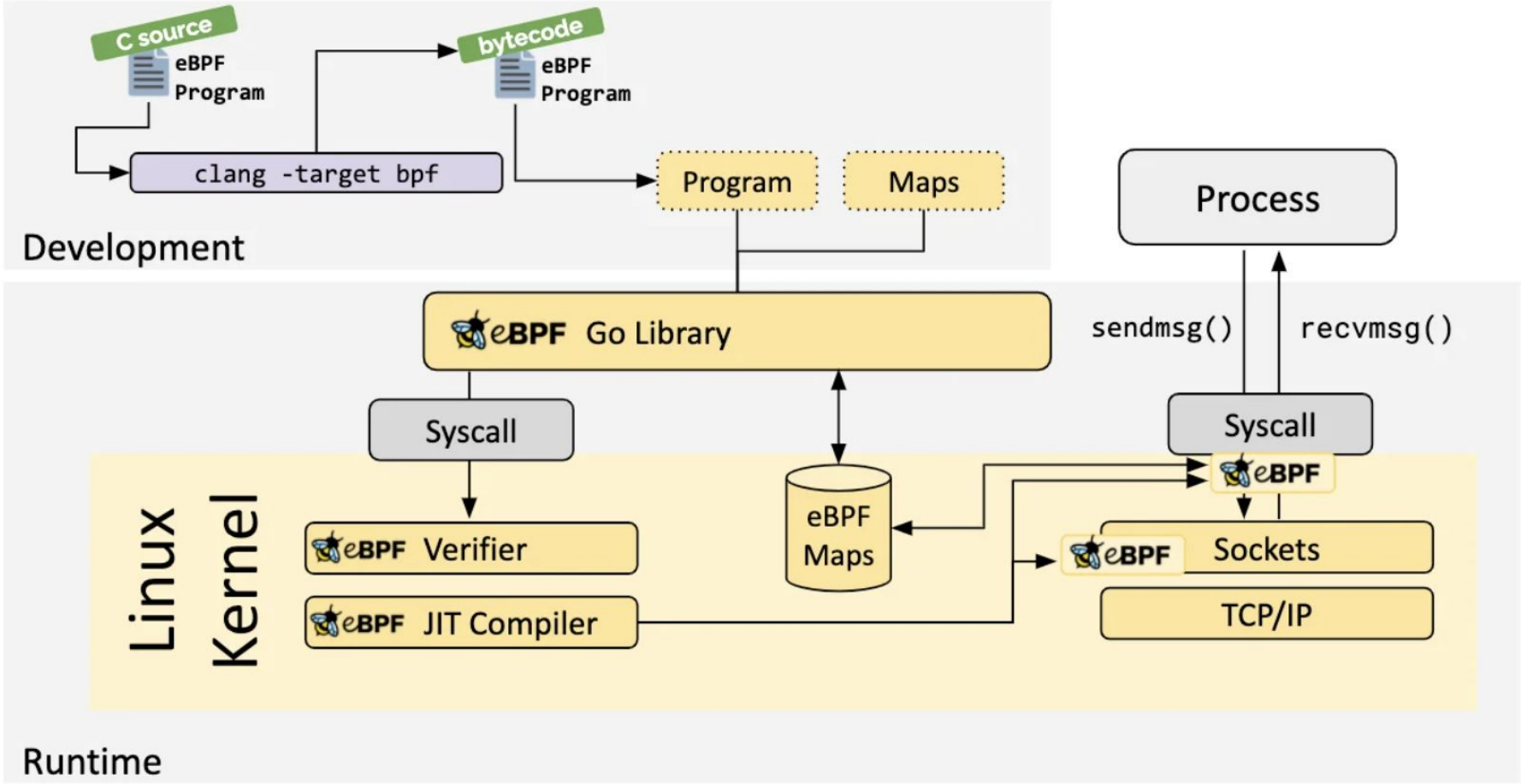
-> Programmability and Flexibility

-> Real-Time Data and High-Frequency Metrics

-> Rich Telemetry available for correlation

What is eBPF?

What is eBPF?


C source
eBPF Program
bytecode
eBPF Program
clang -target bpf
Program
Maps
Development
Process
eBPF Go Library
sendmsg()
recvmsg()
Syscall
Syscall
eBPF
eBPF Maps
eBPF Verifier
eBPF JIT Compiler
eBPF Sockets
TCP/IP
Linux Kernel
Runtime

Tooling available to facilitate network observability via eBPF

Essential tooling

- iovisor /bcc:

  Tools for BPF-based Linux IO analysis, networking, monitoring, and more
- LLVM Compiler Infrastructure
- bpftrace

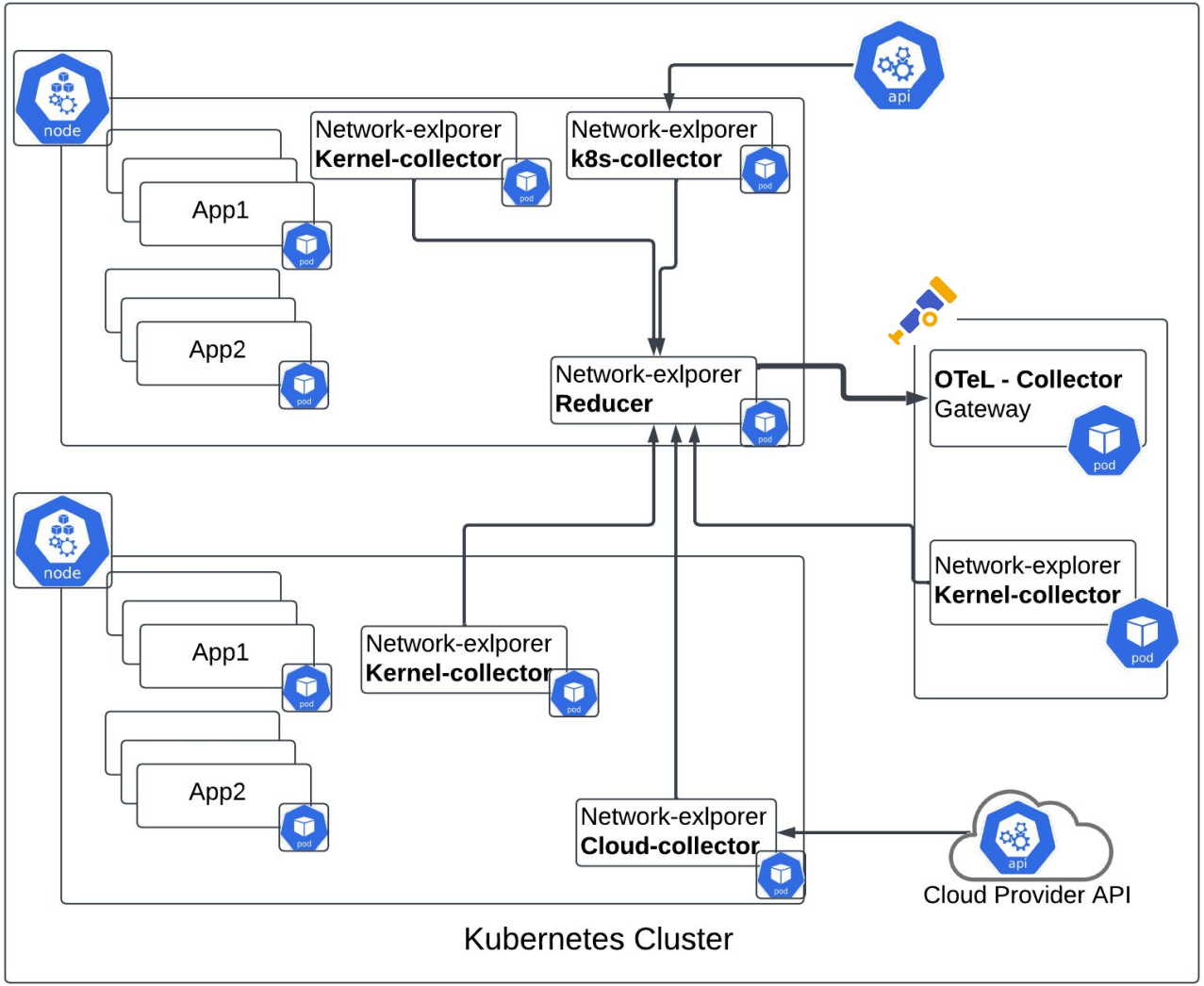
and many more...

Checkout:

https://ebpf.io/applications/

https://ebpf.io/infrastructure/

Understanding Open Telemetry Network architecture



Kernel-collector:

- usually a DaemonSet
- collects *low level telemetry* using eBPF

k8s-collector:

- usually a Deployment
- collects events from k8s *API server*

Cloud-collector:

- usually a Deployment
- gathers metadata from a *cloud provider*

Reducer:

- usually a Deployment
- combines collected data from different agents, and enrich them

Deep dive into individual components and agents of OTeL Network

Deep dive

1) Kernel-collector

2) k8s-collector

3) Cloud-collector

4) Reducer

Kernel Collector

Kernel Collector

eBPF programs:
- heart of the Kernel Collector
- captures data from various kernel subsystems
- BPF Handler

BPF Handler:
- Manages the lifecycle of eBPF probes

Data Collection Modules:
- focuses on a specific aspect of system telemetry
- DNS monitoring, kernel symbols, and process activity
- Reads kernel symbols from */proc/kallsyms* for address resolution
- Monitors control groups (cgroups)

Kernel Collector

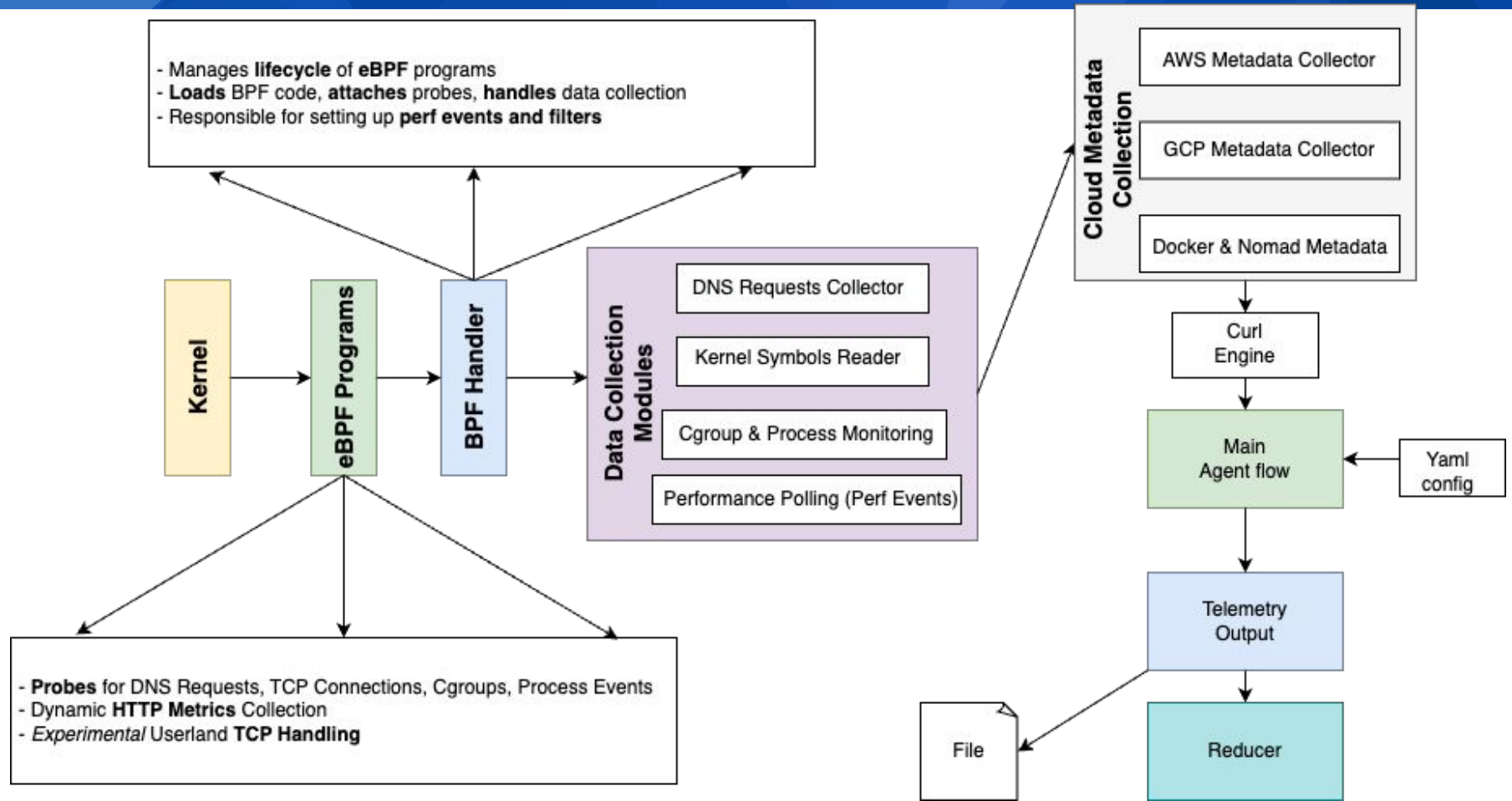
Cloud Metadata Collection:
- Fetching cloud metadata helps contextualize the telemetry data
- Helps in correlating system performance with cloud environment configurations.

Curl Engine:
- handles both metadata fetching and telemetry output
- ensuring that data is collected and sent efficiently

Main Application Flow:
- Initialization, configuration, and signal management using the libuv event loop
- non-blocking framework for managing the application's lifecycle

Telemetry Output and Data Transmission:
- Supports dual output modes
- write to local files or stream to upstream reducer server

Kernel Collector

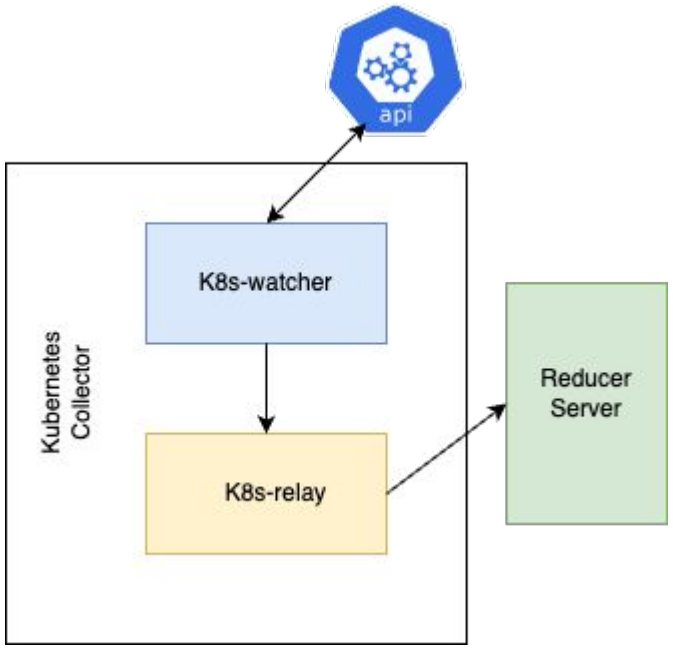


Kernel Collector


```
➜  ~ k logs -n default       example-opentelemetry-ebpf-kernel-collector-9vwxk
resolving kernel headers...
cleaning up stale kprobes...
launching kernel collector...
+ exec /srv/kernel-collector --host-distro amazon --kernel-headers-source
pre_installed --config-file=/etc/network-explorer/config.yaml
--disable-nomad-metadata --debug --trace --log-console
2024-11-11 14:24:58.231031+00:00 debug [p:1024038 t:1024038] setting up breakpad...
FAIDDWUPQAQFRAU0JRMATS3KPA0S8GRAIPUF
2024-11-11 14:24:58.231270+00:00 info [p:1024038 t:1024038] Running on:
   sysname: Linux
  nodename: ip-172-31-90-170.ec2.internal
   release: 5.10.227-219.884.amzn2.x86_64
   version: #1 SMP Tue Oct 22 16:38:23 UTC 2024
   machine: x86_64
2024-11-11 14:24:58.231290+00:00 info [p:1024038 t:1024038] HTTP Metrics: Enabled
2024-11-11 14:24:58.231290+00:00 info [p:1024038 t:1024038] Socket stats interval
in seconds: 10
2024-11-11 14:24:58.231291+00:00 info [p:1024038 t:1024038] Userland TCP: Disabled
2024-11-11 14:24:58.239933+00:00 info [p:1024038 t:1024038]   AZ: AWS-Meta(valid=1
value='us-east-1c')
```

Kubernetes Collector

Kubernetes Collector



k8s watcher:

- gathers information from a Kubernetes API server

- collects events like pod creation and deletion

K8s relay:

- collects data from k8s-watcher and forwards that to reducer

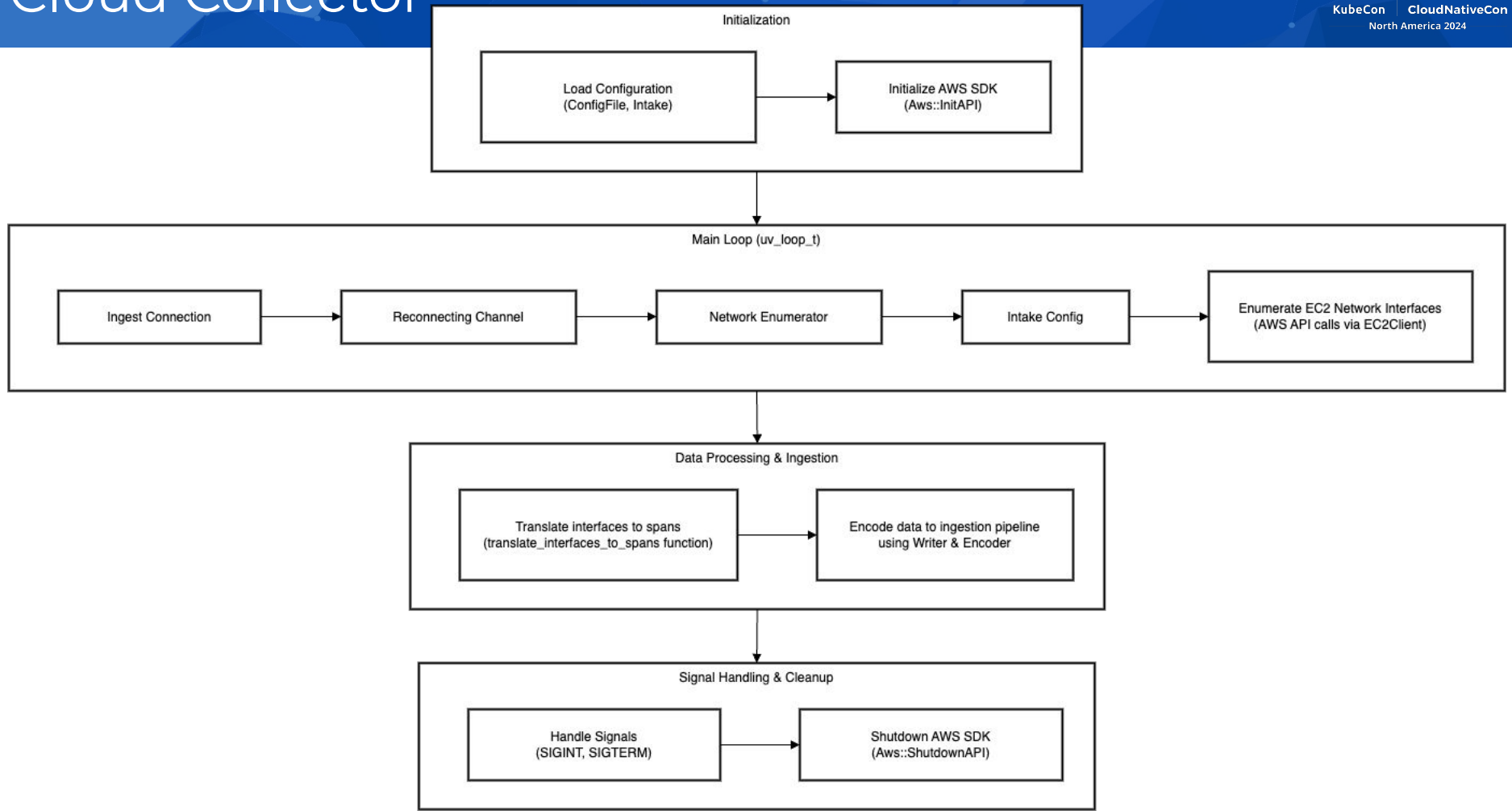
Cloud Collector

Cloud Collector

- Network Interfaces Enumerator: (component of CCA)
  - Queries AWS EC2 APIs
  - Retrieves network interfaces, regions, & metadata
  - Handles throttling & API errors

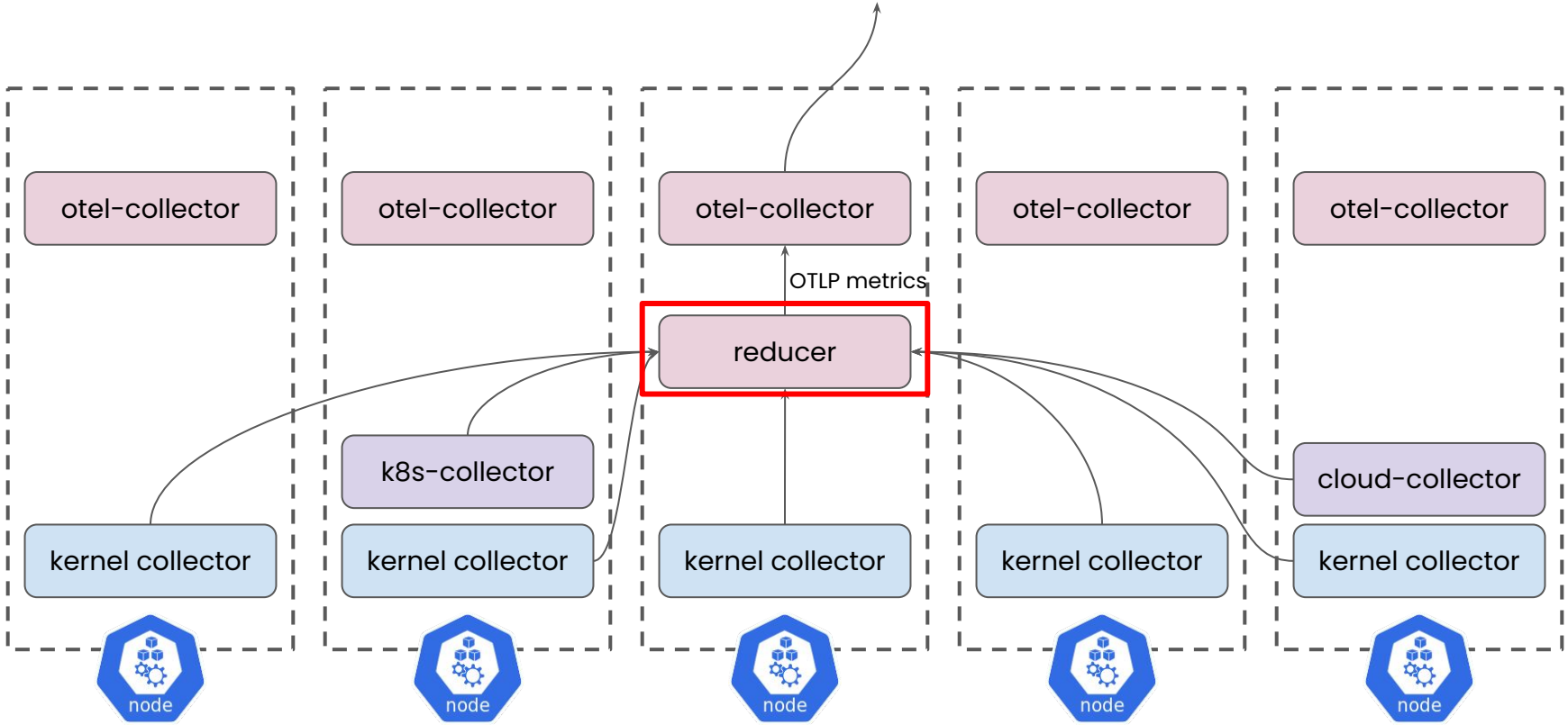
- Cloud Collector Agent:
  - Schedules metadata collection
  - Handles error recovery
  - Sends Data to Ingestion Pipeline
  - Formats data as spans & logs using eBPF ingestion writer
  - Transmits cloud metadata for processing and enrichment to reducer

Cloud Collector

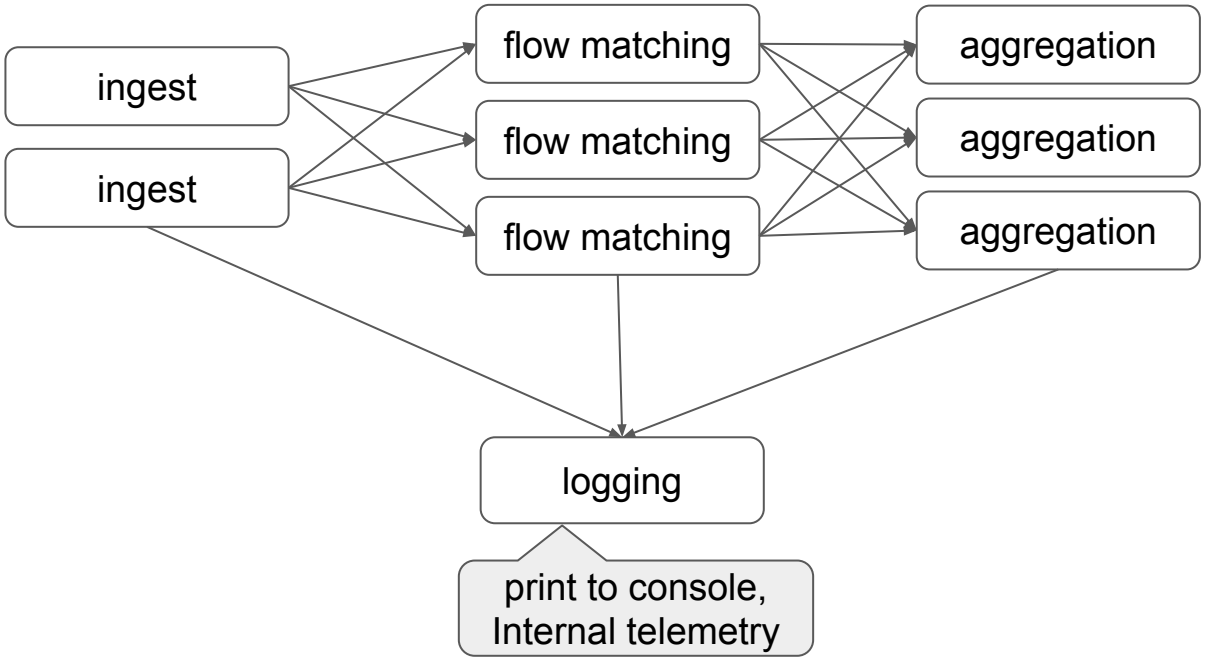


Reducer

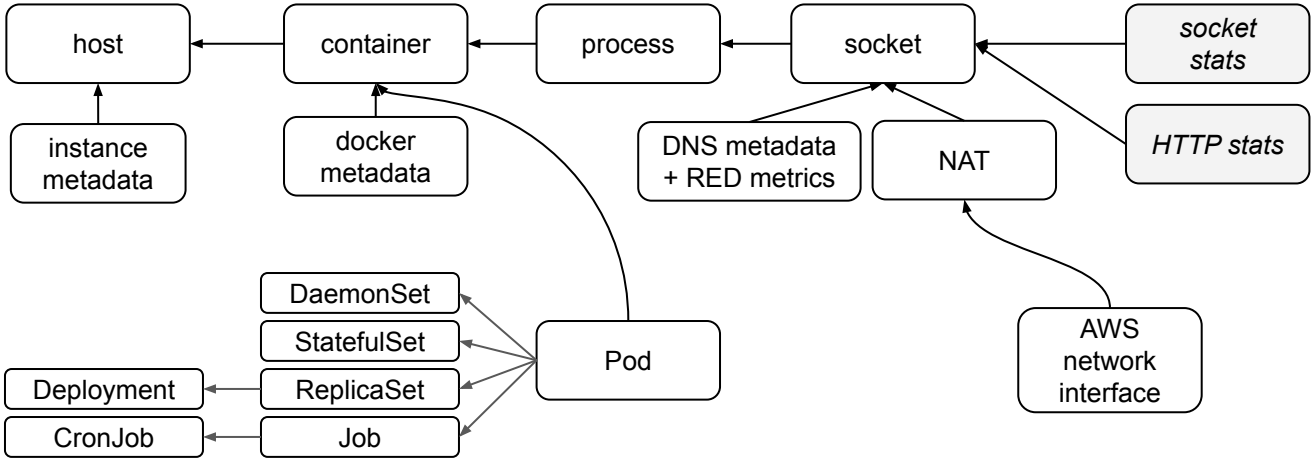
Kubernetes Deployment



Reducer multicore architecture

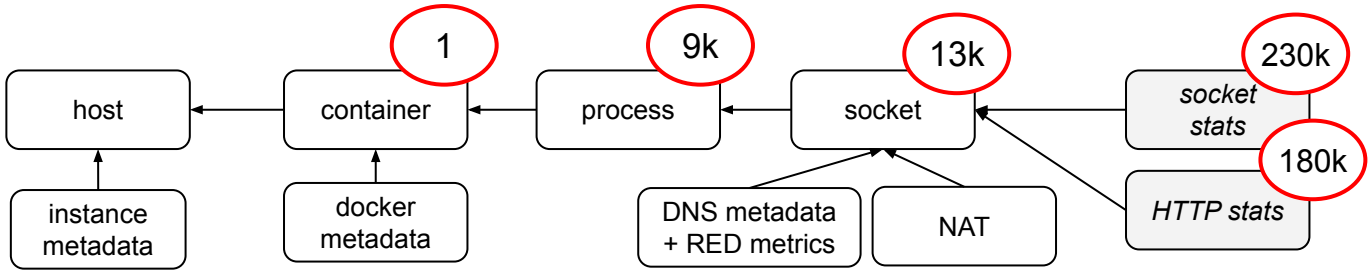


Ingest: Context Drives Value, So Collect It



Ingest maintains a **digital twin**

Controlling Overhead



- DRY: Don’t Repeat Yourself
  - Send metadata on update (for container, process, socket)
- Avoids *lots* of CPU encoding/decoding!
- Keeps size small

* performance depends on the workload observed

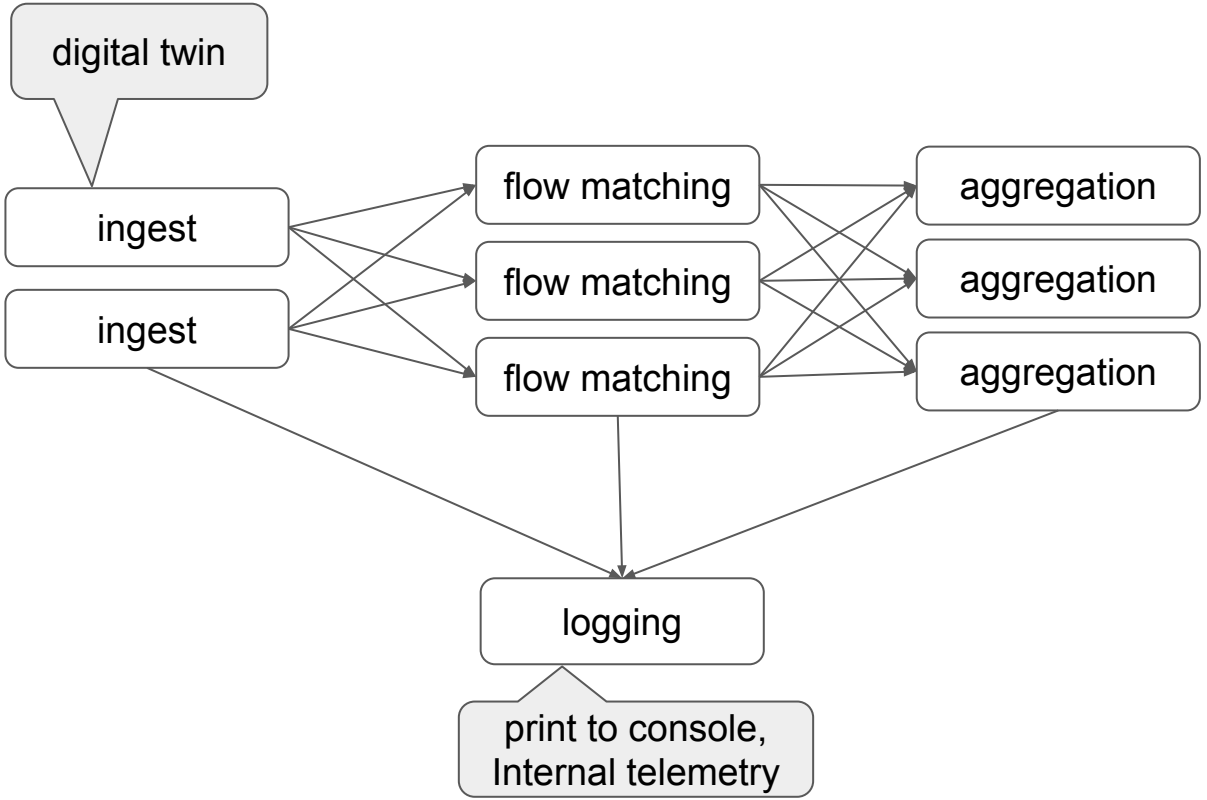
Evaluation: Network overhead

	App throughput (MB / second)	Flow telemetry (MB / second)	**% Network Overhead**
Cluster 1	186.2	0.85	0.46%
Cluster 2	217.1	2.49	1.15%
Cluster 3	249.6	0.25	0.10%
Cluster 4 (batch)	522.0	0.16	0.031%
Cluster 5	183.0	0.02	0.013%

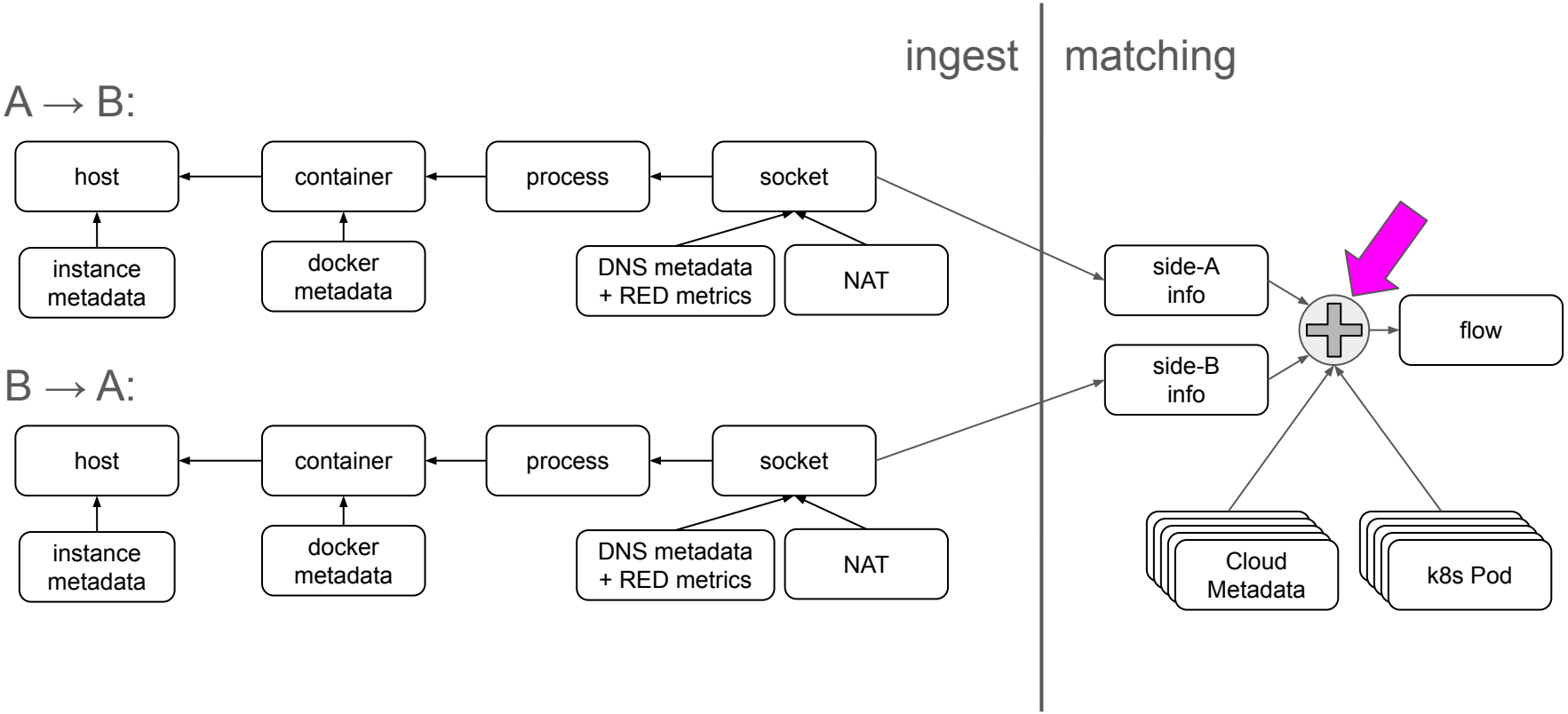
→ Usually < 0.5% network overhead, outliers ~1%

Lucky we use DRY!

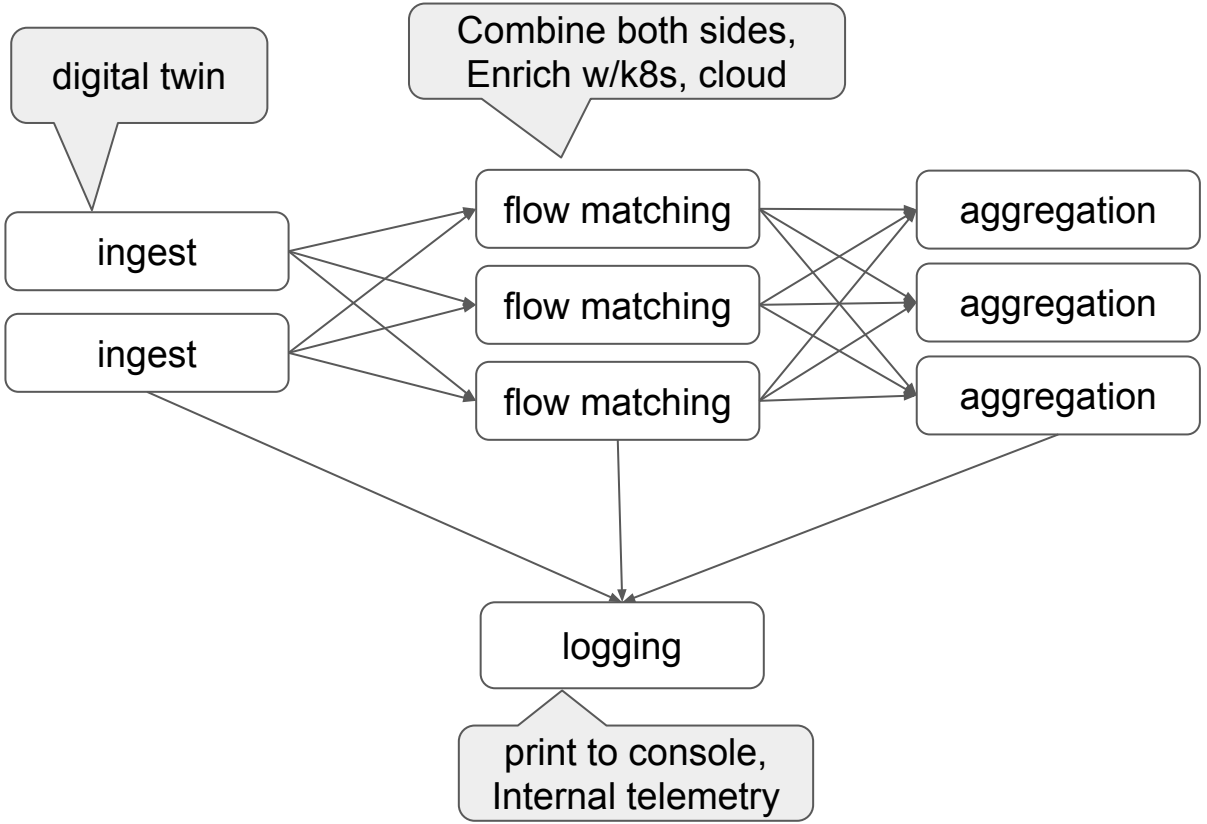
Reducer multicore architecture



Matching and Enriching with metadata



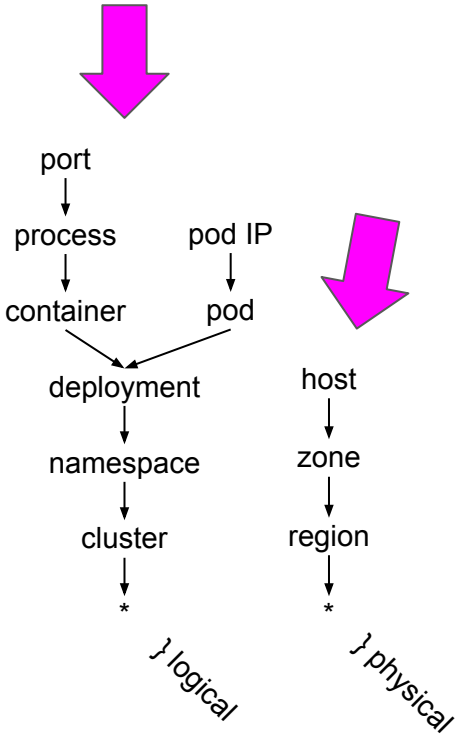
Reducer multicore architecture



Dimensions from Enrichment

But there are too many flows!

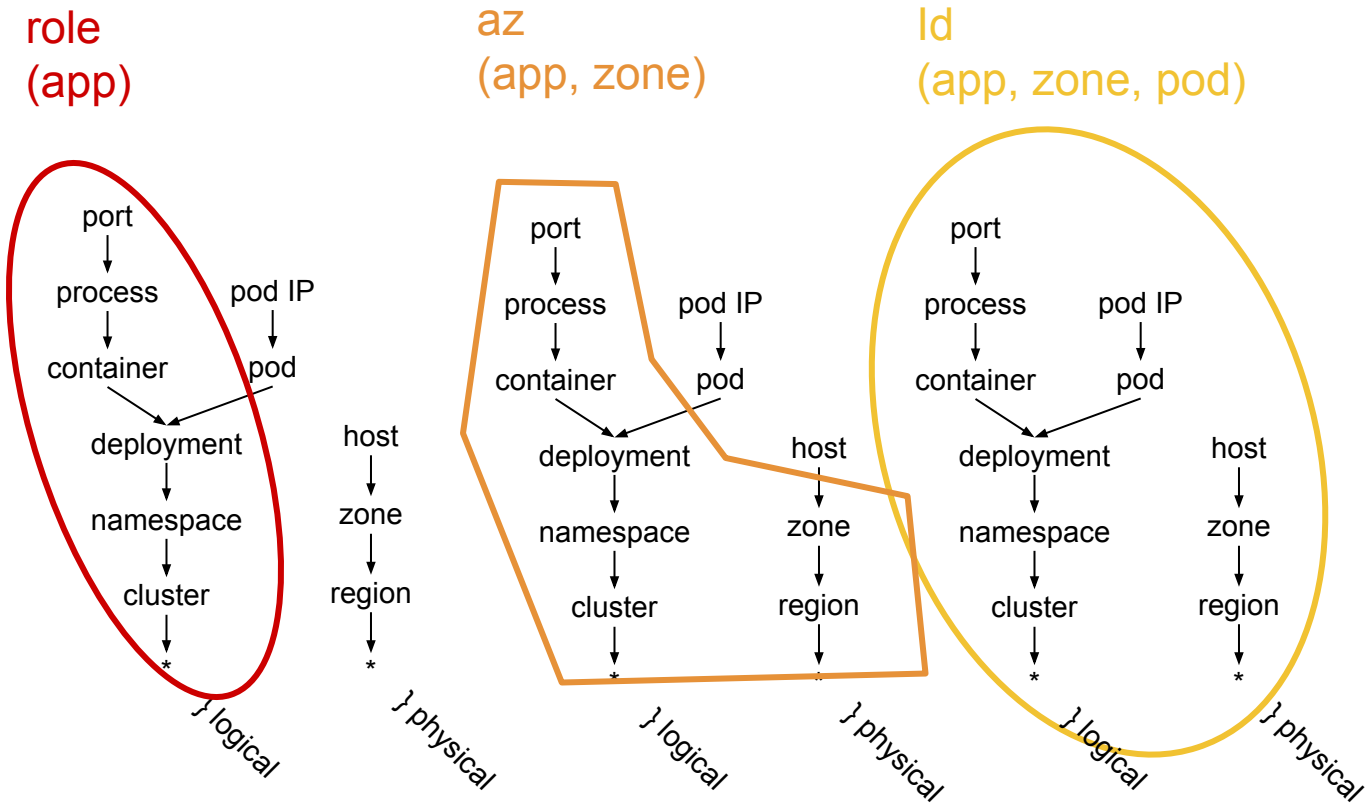
→ aggregate!



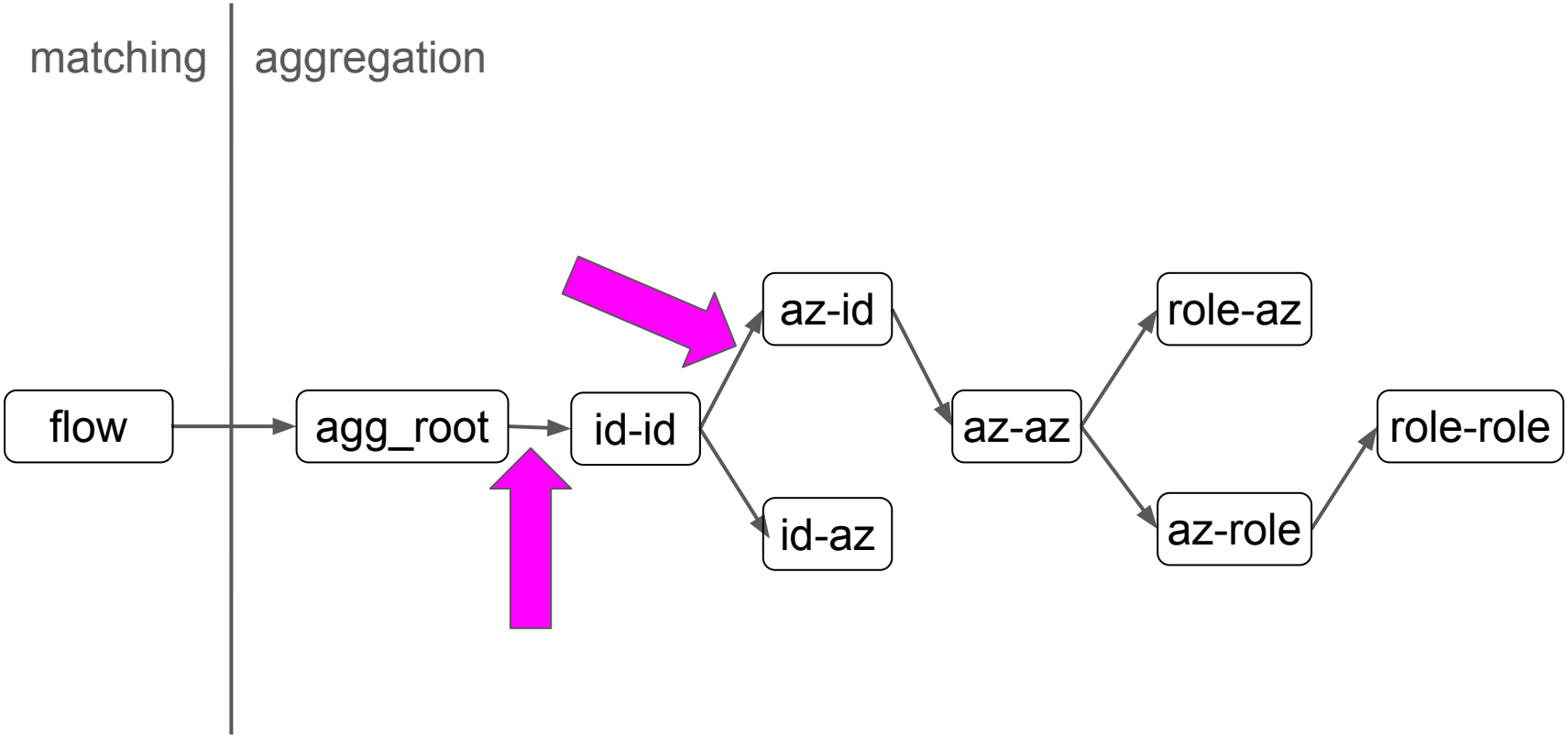
Potential aggregations

sender dimensions,
receiver dimensions

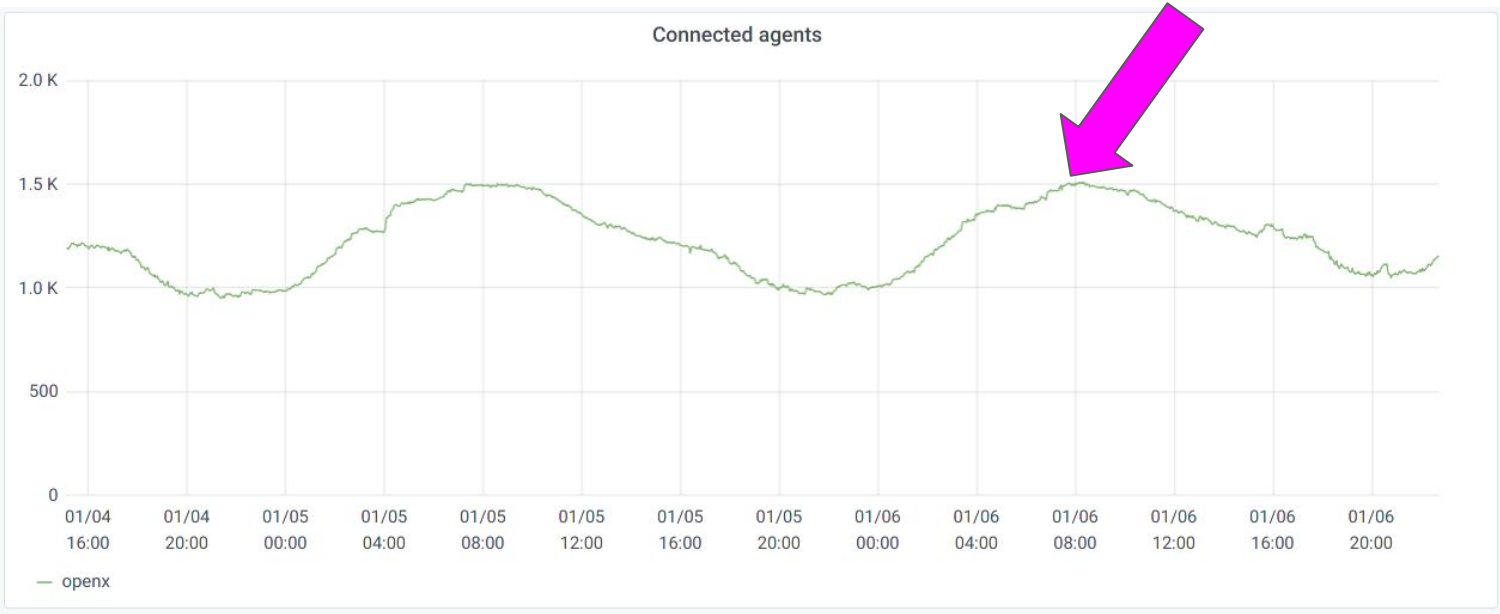
- role-role
- role-az, az-role
- az-az
- az-id, id-az
- id-id



Solution: aggregation tree

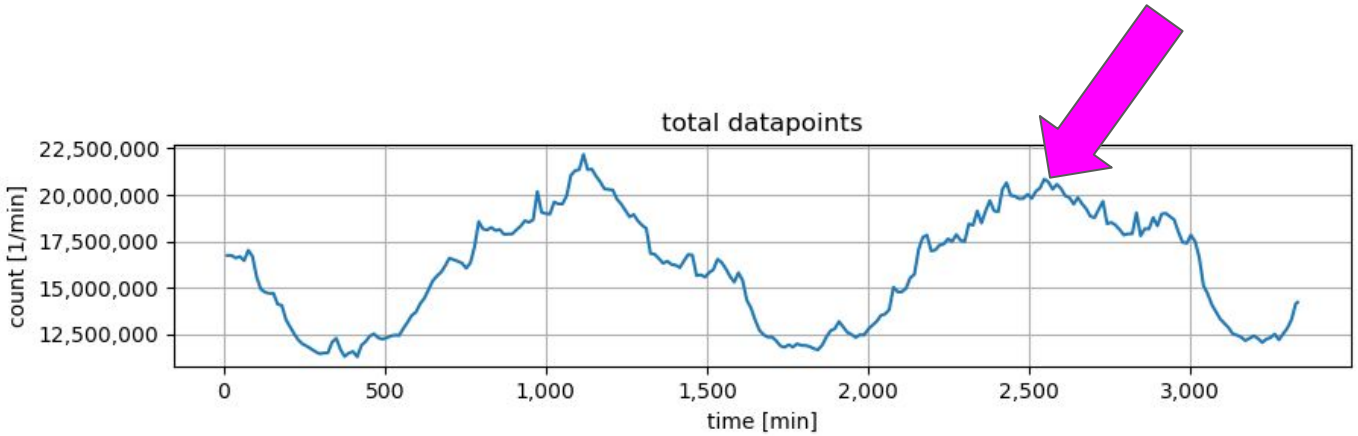


Connected agents

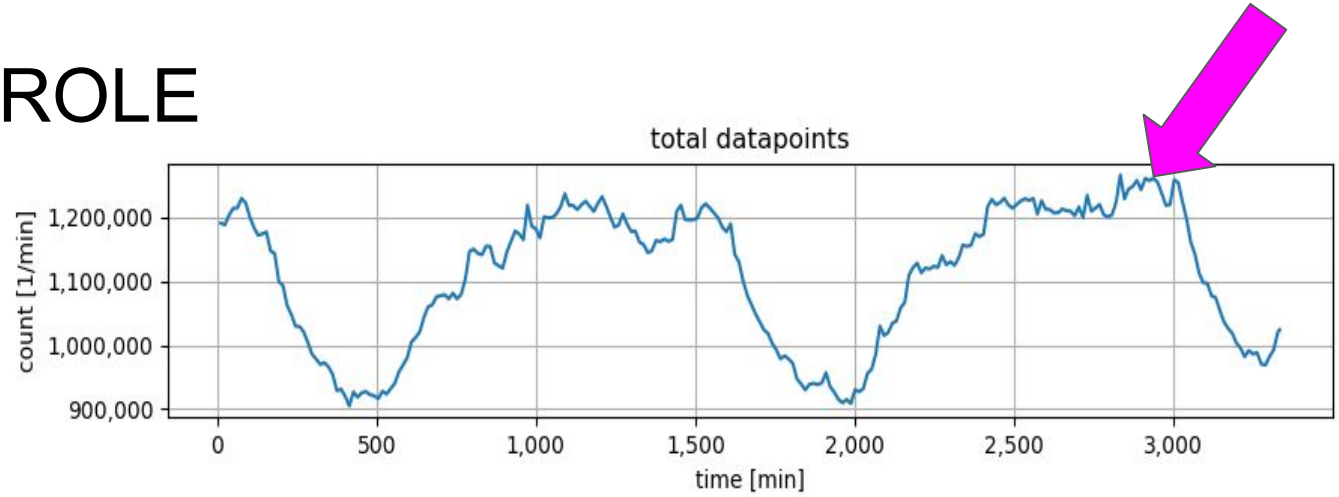


next few slides based on this doc

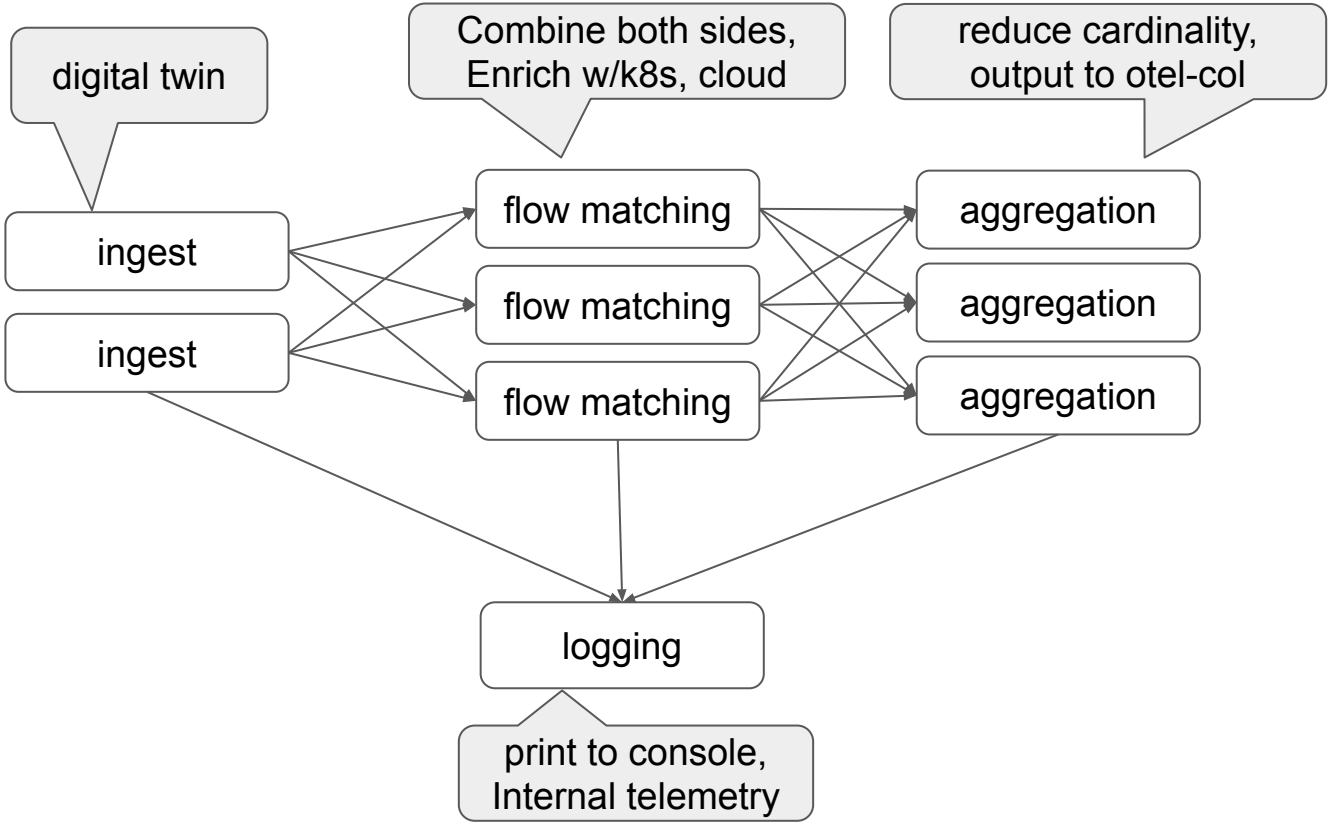
AZ-ID



ROLE-ROLE



Reducer multicore architecture

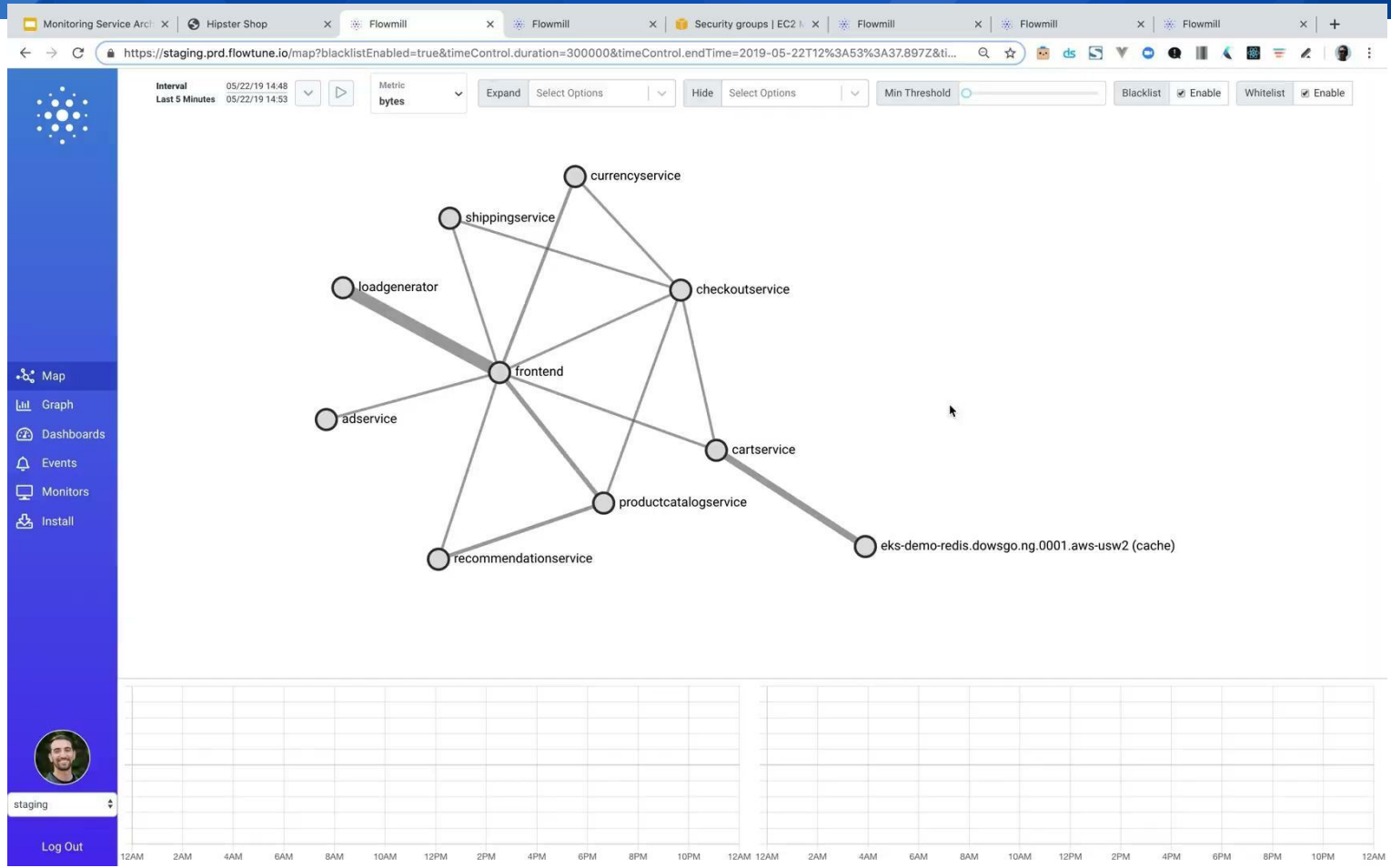


Demo

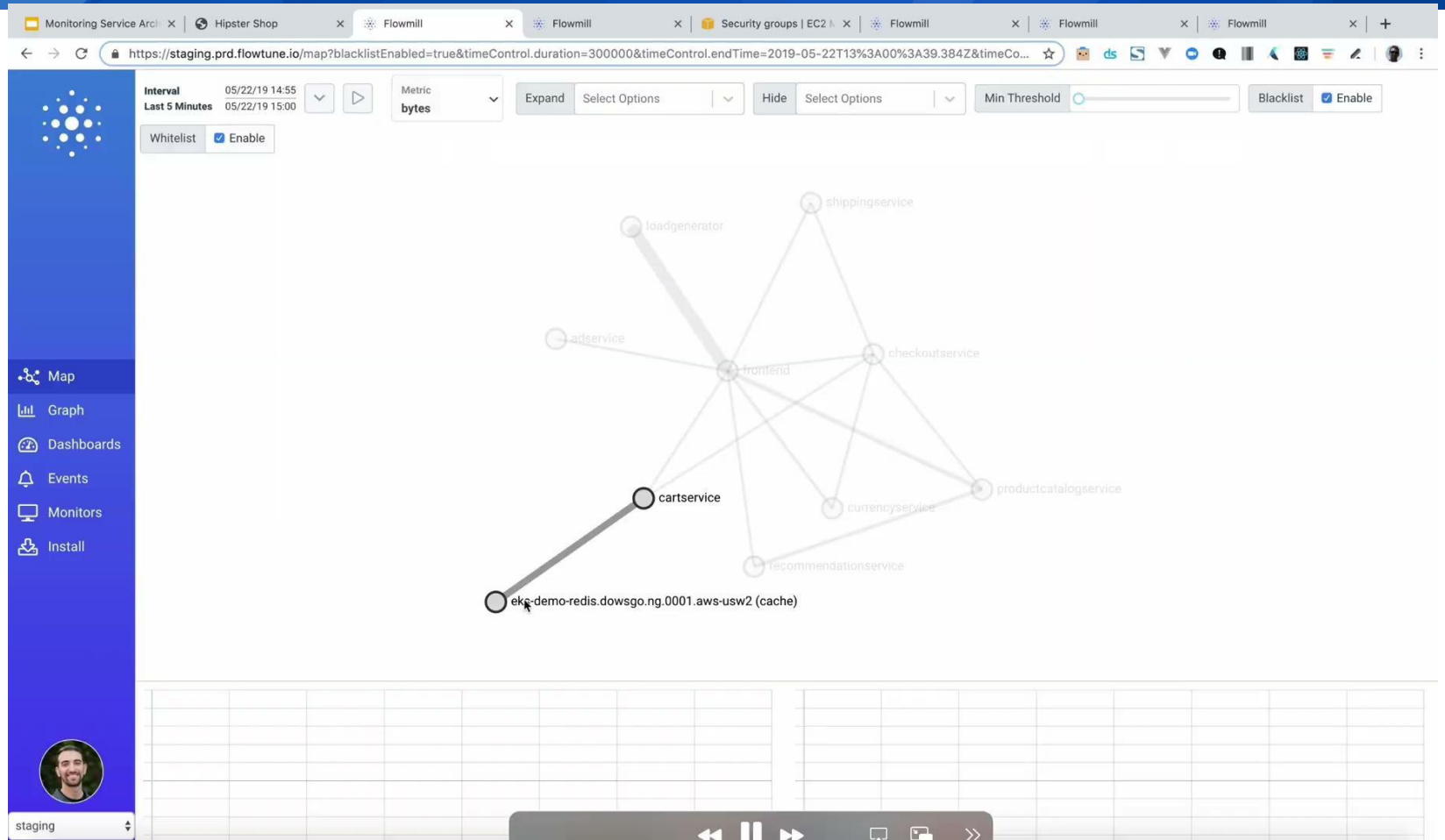
UI Demos

- Next demos: old UI
- Not part of the project
- Can build similar with OSS tools
- Vendor: Splunk NPM
- Others? lmk

Use Case: Audit Architecture



Use Case: Security Group Debugging



How to get involved?

Roadmap (on GitHub)

- **Short-term:**
  - Packaging
  - Test coverage
- **Medium-term:**
  - CO-RE support
  - ARM support
  - Refactoring the k8s-collector
  - Layer 7 collection and parsing
  - Replacing Xtend with plain Java
- **Long-term:**
  - Load-rebalanced reducer components
  - Rust: developer productivity and compiler-assisted code stability
  - Modular collection
  - Module convergence
  - Collection breadth
  - Userspace instrumentation
  - Linking application trace IDs to kernel events
  - High cardinality storage
  - Serving queries from within user deployments

Get Involved

Deploy

- Helm chart in opentelemetry-helm-charts

Engage

- CNCF Slack: #otel-ebpf
- Weekly meeting Tuesday 9am PT
  - (see opentelemetry-community for calendar, up to date time)

Contribute

- Collectors, Reducer: opentelemetry-network
- Build environment: opentelemetry-network-build-tools

KubeCon

CloudNativeCon

North America 2024

Thanks for joining!

Questions?

We appreciate your feedback!