Kubernetes...
Minus the Blind Spots

Solving K8s Observability Challenges with Real-Time Topology

Conf42 DevOps
January 2025

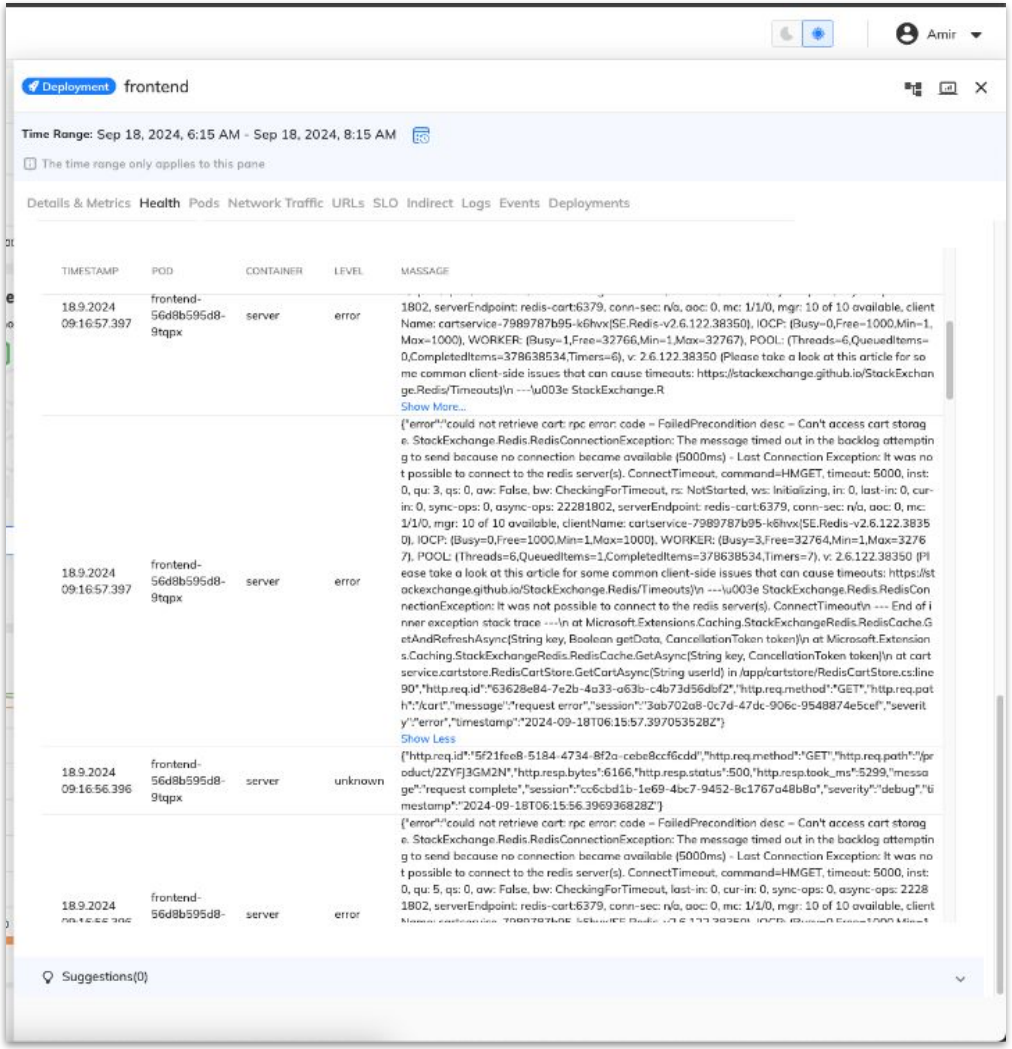
Today's Roadmap

- Kubernetes is great (until you have to troubleshoot)
- Challenges with the traditional observability toolkit
- Real-time topology mapping: what, why, how
- Real-world use cases
- eBPF as real-time topology enabler
- Getting started

SENSER

Kubernetes is great (until you have to troubleshoot)

- ❗ Blurred line between app and infra issues
- ❗ Ephemeral environments
- ❗ Complex networking (CoreDNS, kube-proxy, CNI plugins, etc.)
- ❗ Shared resources and noisy neighbors
- ❗ Black box containers
- ❗ Automation complexity



"The problem with logs is that they require more logs, and rarely from a single place"

Challenges with the traditional observability toolkit

Metrics

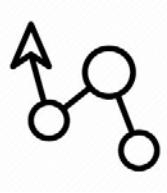
Track trends but lack system views and context.

What's causing this spike in CPU usage?

Logs

"Coarse," provide granular detail but are fragmented.

What are the downstream effects of an error in this pod?

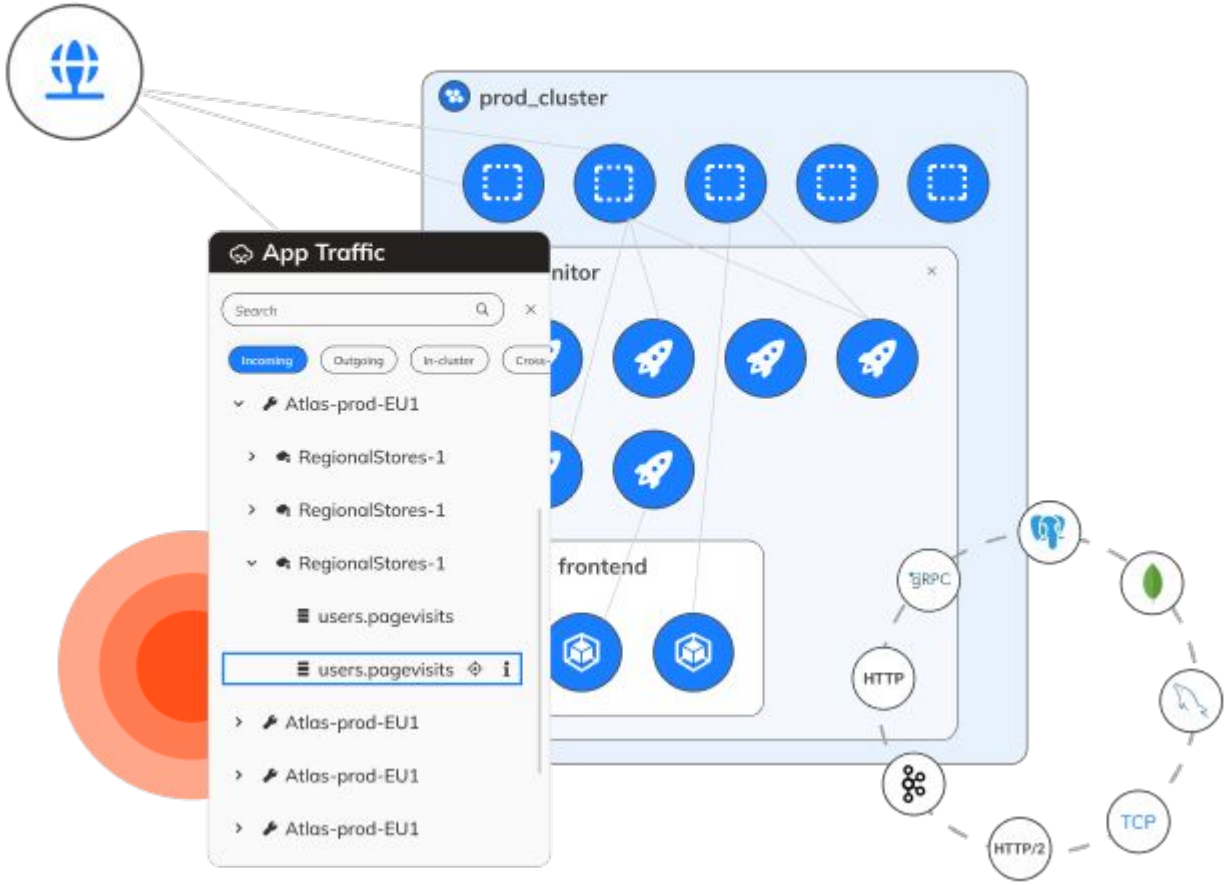


Traces

Show service interactions but need instrumentation.

We're flying blind on the services we haven't instrumented...

Real-time topology provides the context required to eliminate blind spots and troubleshoot, fast.



What it is

A live map of system relationships (pods, services, networks)

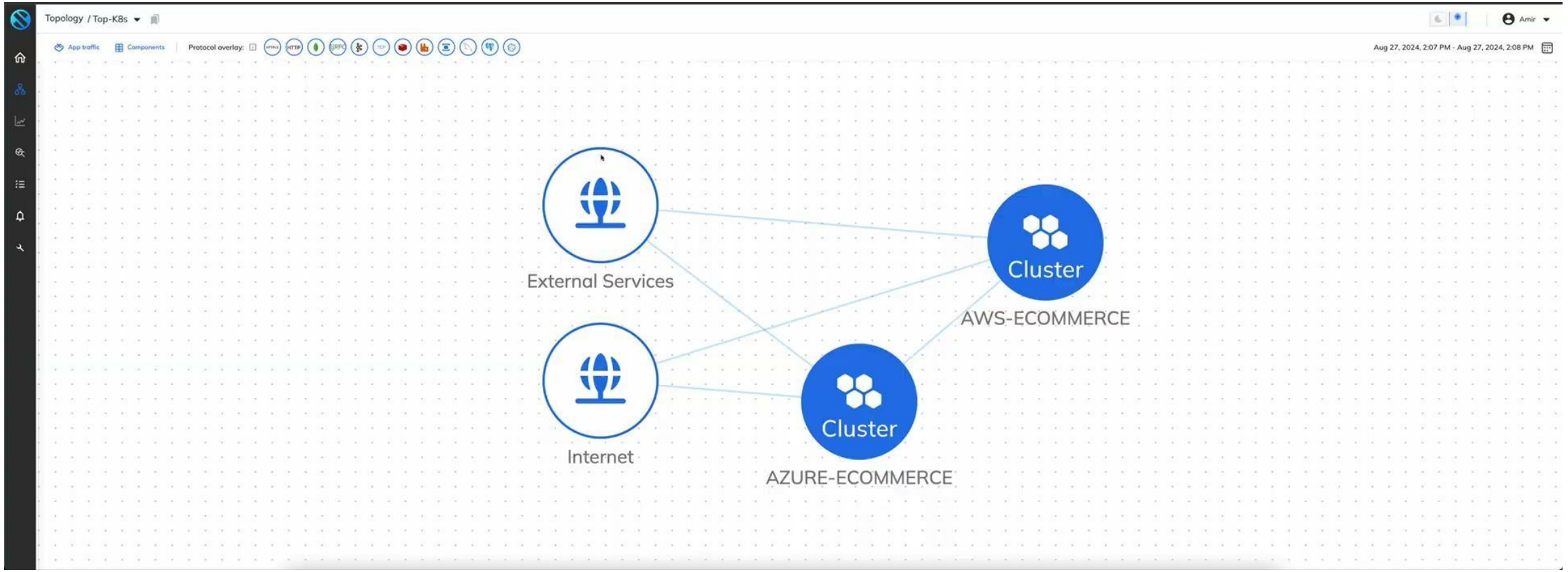
Why it matters

Connect individual data points into a full-system view

How it works

Continuously updates to reflect real-time changes (every new service, every deployment)

Real-time topology provides the context required to eliminate blind spots and troubleshoot, fast.



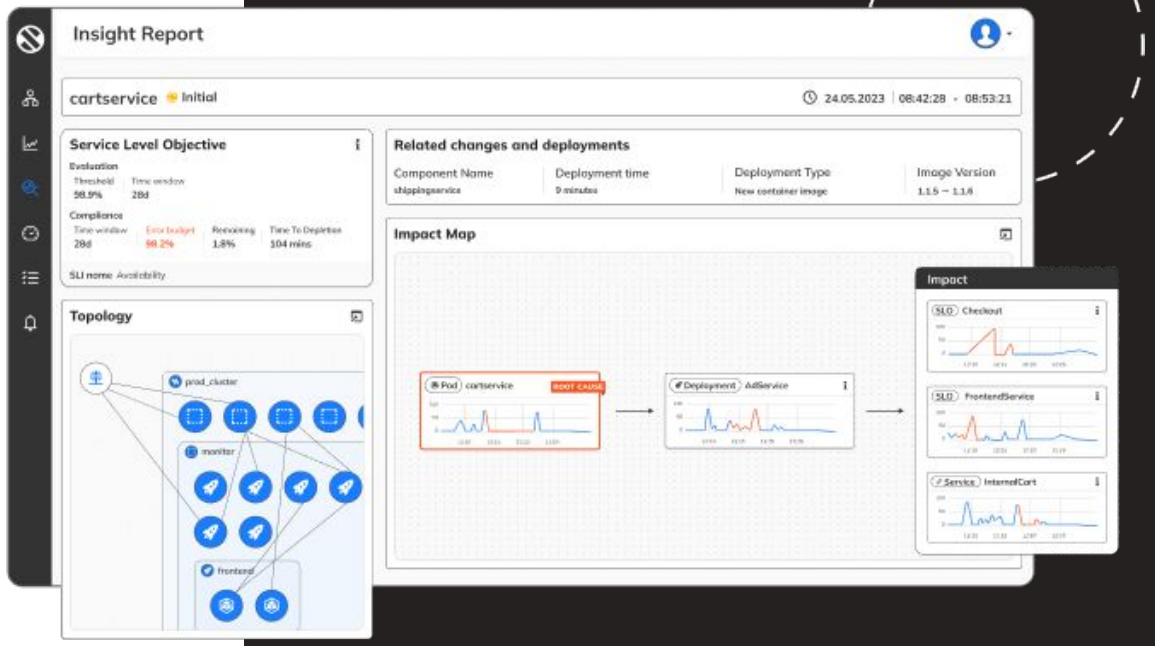
Use case #1: Probe readiness failure

The scenario

A stateful application in Kubernetes fails readiness probes due to high disk I/O, leading to pods being repeatedly restarted by the kubelet. Without proper visibility, it's difficult to correlate the high disk usage with other factors like database queries or backup operations.

How real-time topology helps

Mapping pod dependencies and resource usage, enabling teams to identify the root cause of the disk I/O spike and adjust the probe configurations or workload scheduling.



SENSER

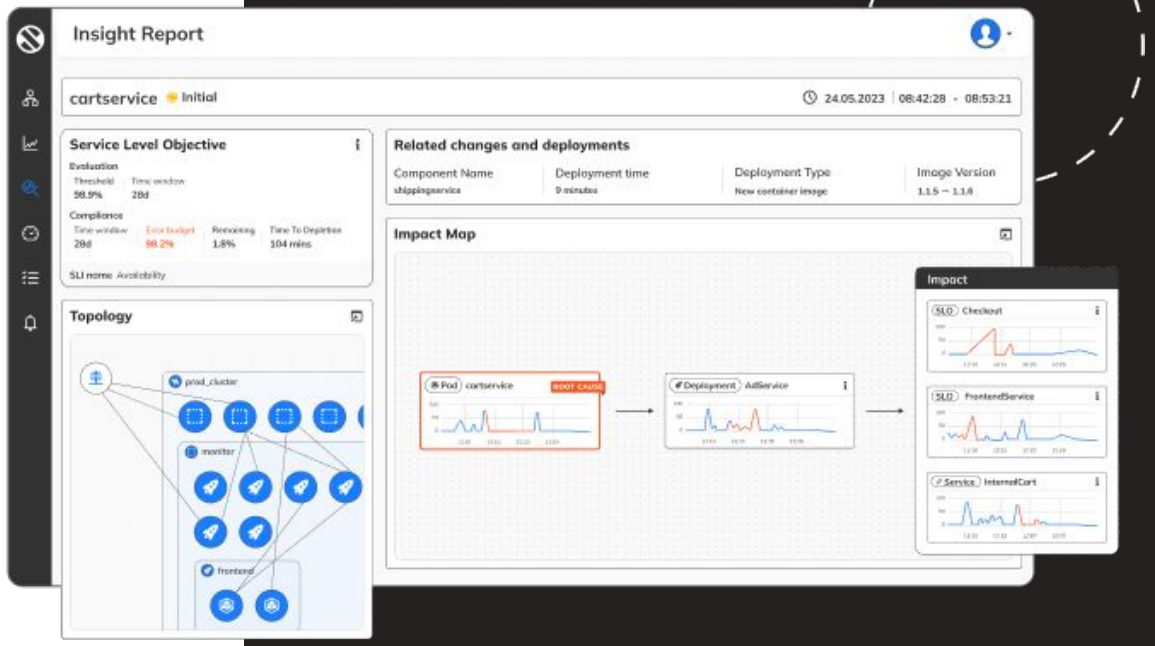
Use case #2: Throttling in autoscaling

The scenario

An application under high traffic triggers Horizontal Pod Autoscaling (HPA), but new pods experience throttling due to CPU limits set too low. This slows down traffic handling and results in degraded performance.

How real-time topology helps

Visualizing resource allocation and pod scaling behavior, enabling teams to adjust resource requests and limits to align with the workload and improve scaling efficiency.



SENSER

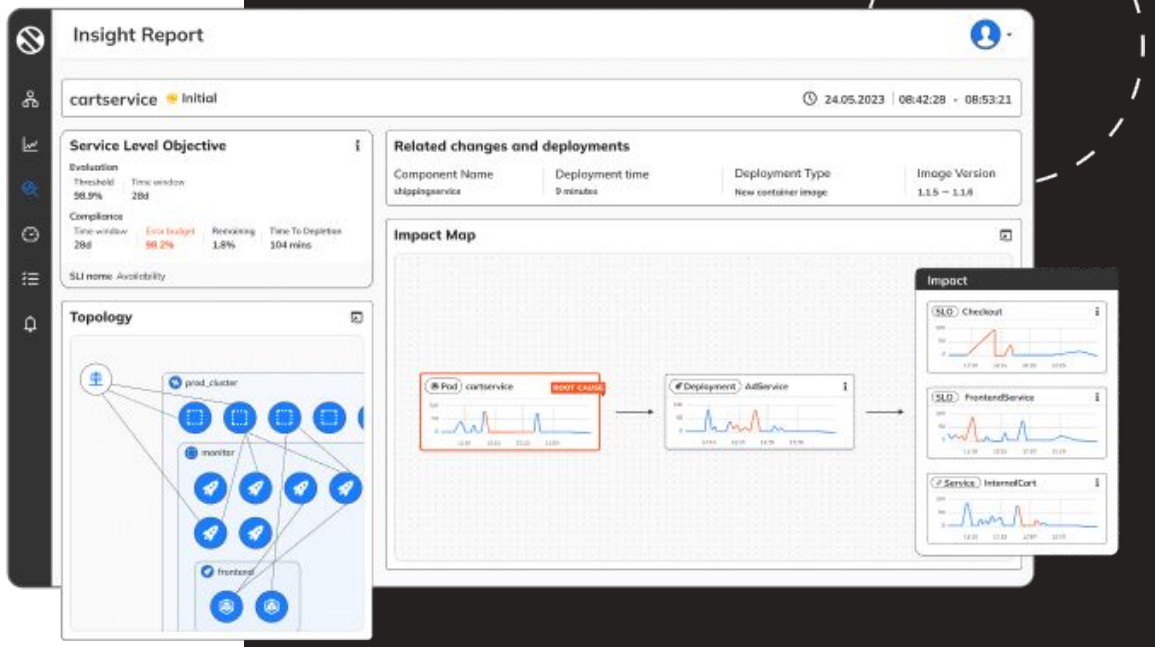
Use case #3: Misconfigured network policies

The scenario

A microservice deployed in Kubernetes fails to communicate with its dependent services due to overly restrictive Network Policies. The lack of detailed logs or visibility into blocked connections makes it difficult to diagnose the issue.

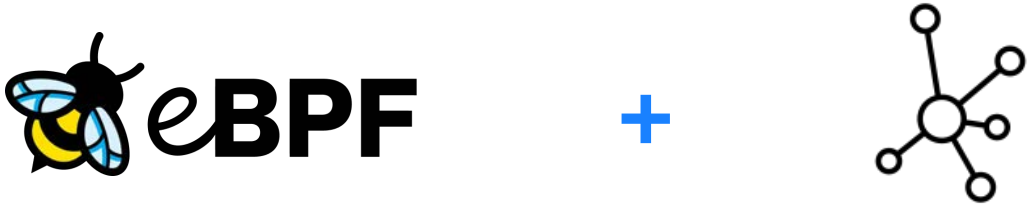
How real-time topology helps

Mapping service-to-service communication paths and highlighting blocked connections, enabling teams to update Network Policies and restore communication between services.



SENSER

What enables real-time topology



What is eBPF?

eBPF (extended Berkeley Packet Filter) is a Linux kernel feature that allows safe, low-level instrumentation and monitoring.

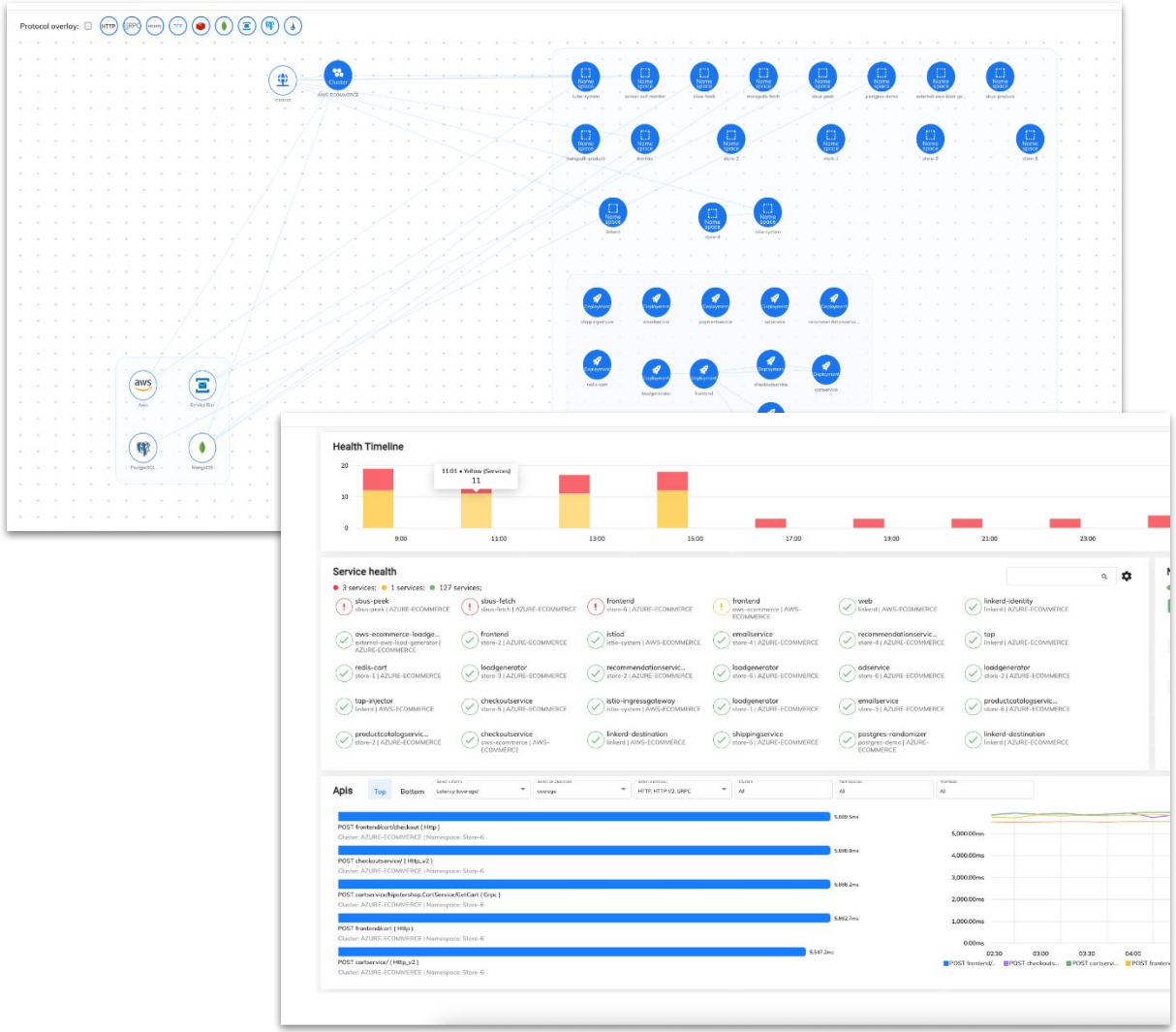
- **Traffic analysis**: Trace network requests, messages, and data flows.
- **Infra, App & Performance monitoring**: Observe system calls, resource usage, and latency in real-time.
- **Security auditing**: Detect anomalies or unauthorized access at a granular level.

Why it matters

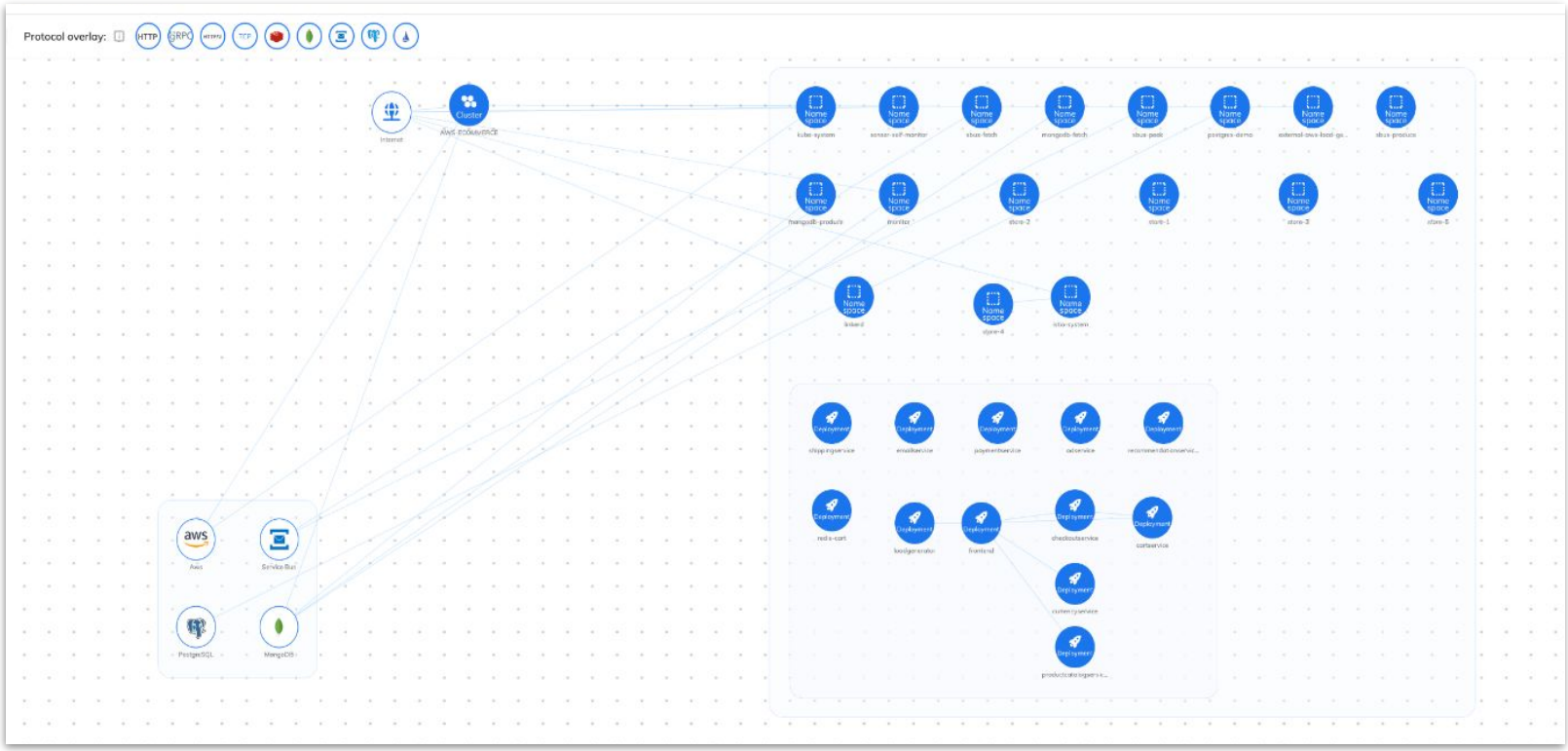
*Enables **deep observability** directly from the kernel, making it invaluable for **complex, distributed environments**.*

Why eBPF is ideal for distributed systems

- **Low-latency monitoring**. Captures and displays real-time insights without hindering system performance.
- **Enhanced visibility**. Sees all network interactions and system calls to provide comprehensive insights.
- **Improved efficiency**. Reduces need for multiple diagnostic tools, centralizing observability with eBPF.
- **Scalability**. Adapts to cloud-native and microservices architectures, providing a consistent observability approach.



Auto-discovery and mapping of your environment



AI capabilities that use Kubernetes metadata and traffic inspection to discover dependencies and create a dynamic service graph – without requiring manual updates.

Common pitfalls (and how to avoid them)

- **Incomplete instrumentation:** Missing critical data like kubelet logs, API server events, or container runtime metrics leaves significant gaps in the topology, reducing its usefulness. (Technologies like eBPF can help.)
- **Stale topology maps.** Kubernetes environments are constantly changing. If topology updates aren't automatic and real-time, you risk basing decisions on outdated information.
- **Assuming topology alone solves everything.** Topology mapping is powerful for troubleshooting – but by itself rarely a "smoking gun" in root cause analysis. (This is where AIOps shines.)

Tips for getting started

- **Leverage Kubernetes metadata.** Use tools that automatically pull data from Kubernetes-native sources like pod labels, ConfigMaps, and events to build accurate, context-rich maps.
- **Automate topology maps.** Choose solutions that dynamically update the topology to keep up with Kubernetes' rapid scaling and frequent state changes.
- **Avoid context-building pitfalls.** Begin with critical applications where better visibility will have the greatest operational benefit, then expand as you refine your process.

SENSER

Thank you!