

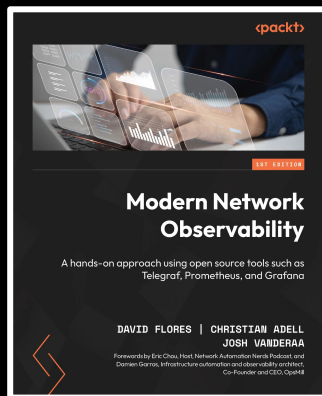
Rethinking Network Monitoring

Christian Adell | David Flores
ESNOG36



- Network Automation and Observability at CoreWeave, former NTC and Cisco network automation architect
- Co-Author of “**Modern Network Observability**”, “Network Programmability & Automation v2”, “Network Automation Cookbook v2”, and other books
- Currently, writing an open book at

<https://designingnetworkautomation.com/>



- Network Automation and Observability at CoreWeave
- Experience in Service Provider, Financial, Hyperscalers, and Integrators.
- Focused on building open source automation and network observability platforms
- Co-Author of “**Modern Network Observability**”

The Vision

- A new device gets added to the SoT.
 - Monitoring starts automatically. No ticket. No manual config.
- An interface starts behaving poorly at 3am.
 - A workflow runs, verifies, remediates. Nobody wakes up.

*This is not a roadmap. This is achievable today.
But only if the data underneath is ready for it.*

You Probably Have Most of This

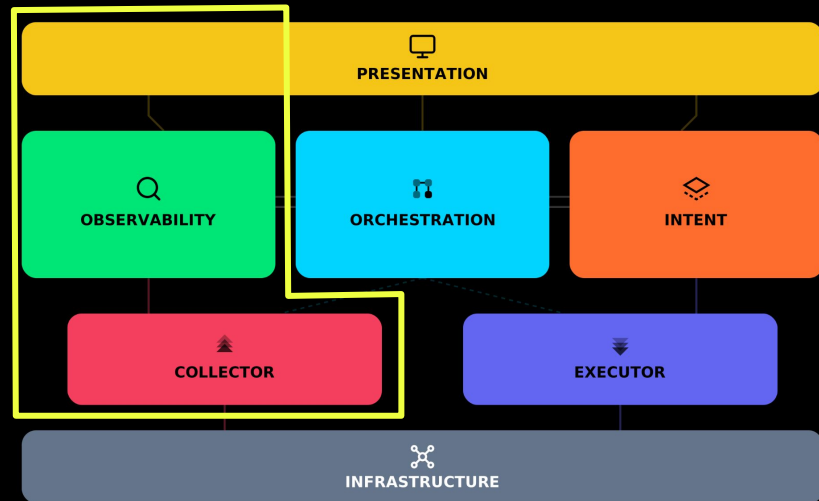
- Solid dashboards ✓
- Thresholds and alerts ✓
- Maybe some Automation/AI wrapped around it ✓
- But...
 - Alerts end at Slack. Nobody reads Slack at 3am.
 - The loop (observe, detect, respond, verify) is not closed
 - The requirements have quietly evolved
 - Most implementations have not kept pace

The gap isn't tooling. It's architecture.

That's what this talk is about.

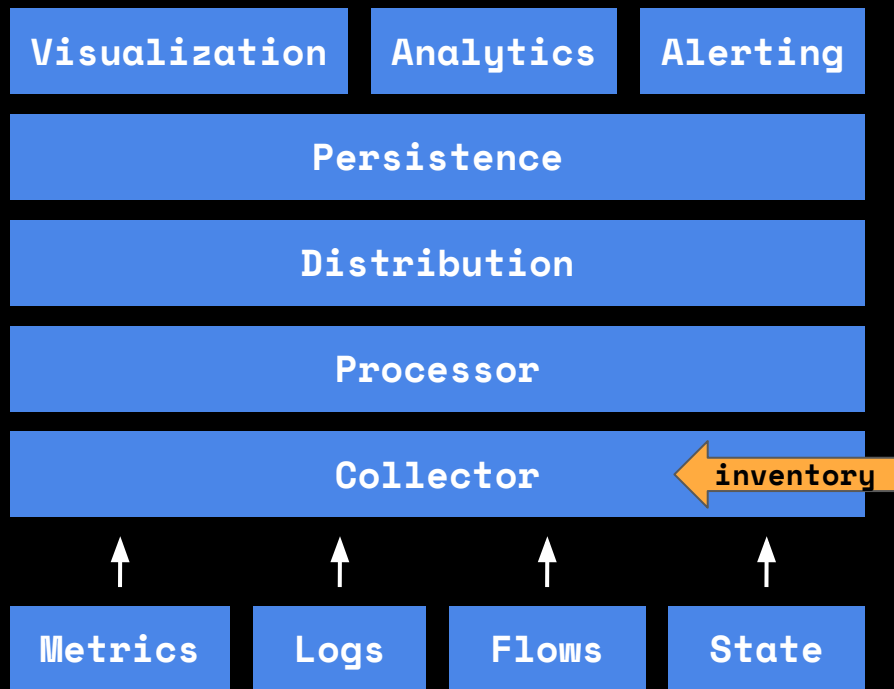
Monitoring in the NAF Framework

- A first-class architectural block, not just a monitoring tool
- Two roles running in parallel:
 - **Human role**: insight, dashboards, capacity planning, decisions
 - **Automation role**: the feedback signal that closes the loop
- **Without monitoring, automation is blind**
 - It can execute, but it cannot verify
 - Intent and execution mean nothing without feedback



Observability Is a Pipeline, Not a Product

- Each block has a job. Each job feeds the next.
- Choosing a tool is step 3. Understanding the architecture is step 1.
- Whatever your tool of choice:
 - The underlying pattern is the same
 - Understanding it helps you **choose, scale, and evolve** your stack



Three Lenses for What Monitoring Needs to Be in 2026

- **DIVERSE**
 - All data planes, all signal types, enriched with context
- **INTEGRABLE**
 - Connected to SoT, Presentation, and Orchestrator (closing the loop)
- **SCALABLE**
 - Architectural decisions that let it grow without degrading operations

Diversity and integration without scale will eventually collapse under load.

DIVERSE

The Dream (as a human or an AI agent)

- Raise an alert when an uplink interface for a border router from different vendors:
 - Has more than 1% of input errors
 - More than 2 critical events have been seen in last 15 mins
 - It transports traffic for network 192.0.2.0/24

The Dream (as a human or an AI agent)

- Raise an alert when an uplink interface for a border router from different vendors:
 - Has more than 1% of input errors
 - More than 2 critical events have been seen in last 15 mins
 - It transports traffic for network 192.0.2.0/24

Diverse Data

The Dream (as a human or an AI agent)

- Raise an alert when an uplink interface for a border router from different vendors:
 - Has more than 1% of input errors
 - More than 2 critical events have been seen in last 15 mins
 - It transports traffic for network 192.0.2.0/24

Diverse Data

Normalized Data

The Dream (as a human or an AI agent)

- Raise an alert when an **uplink interface** for a **border router** from **different vendors**:
 - Has more than 1% of **input errors**
 - More than 2 **critical events** have been seen in last 15 mins
 - It transports **traffic for network 192.0.2.0/24**

Diverse Data

Normalized Data

Enriched Data

The Dream (as a human or an AI agent)

- Raise an alert when an **uplink interface** for a **border router** from **different vendors**:
 - Has more than 1% of **input errors**
 - More than 2 **critical events** have been seen in last 15 mins
 - It transports **traffic for network 192.0.2.0/24**

Diverse Data

Normalized Data

Enriched Data

*... and now, run it
for hundreds of
thousands of devices*

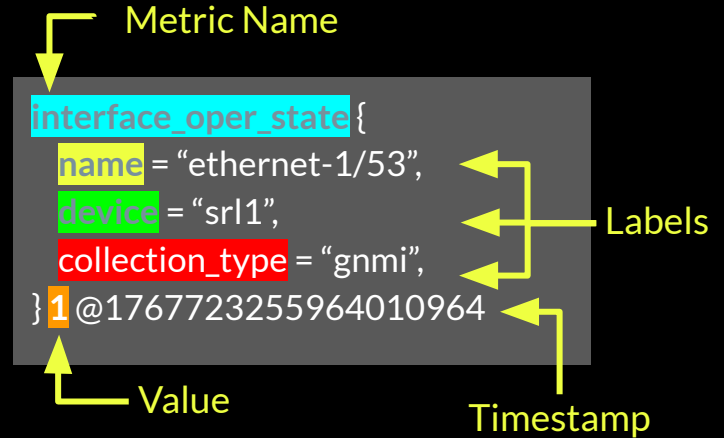
Your Network Generates More Than Metrics

	Metrics	Events / Logs	Flows	Protocol / Routing State	Topology / Inventory	Packet Capture
Management plane	SNMP gNMI NETCONF HTTP scraping OTLP SSH (scraping) Interface counters, CPU, memory, queue depth	Syslog OTLP SNMP traps gNMI ON_CHANGE Config changes, auth events, threshold crossings, interface up/down	—	—	SNMP (ifTable) NETCONF HTTP / REST SSH (show commands) Device inventory, interfaces, vendor, model, OS version	—
Control plane	gNMI SNMP BGP prefix counts, OSPF neighbor count, convergence timers, route churn rate	Syslog gNMI ON_CHANGE SNMP traps Adjacency up/down, route flaps, LDP/RSVP events, BFD state change	—	BGP (BMP) OSPF IS-IS ARP / NDP BFD LDP RSVP SSH (show bgp, show route) RIB/adj-RIB, LSP state, FEC bindings, neighbor state, routing table snapshots	LLDP / CDP BGP (BMP) SSH (show lldp, show arp) Physical topology, peer relationships, AS paths	SPAN / TAP BGP, OSPF, BFD packet-level capture for debugging
Forwarding plane	Synthetic probes eBPF gNMI (FIB counters) Drop counters, ASIC queues, latency, jitter, loss	Syslog eBPF SNMP traps ACL hits, FIB drops, QoS events, ASIC hardware exceptions	IPFIX NetFlow v9 sFlow Per-flow traffic, top talkers, traffic matrix	FIB snapshot LACP SSH (show fib, show lacp) Active forwarding entries, LAG member state	MAC table ARP / NDP cache SSH (show mac, show arp) L2 adjacency, host location, endpoint tracking	PCAP SPAN / TAP eBPF (XDP) Full packet inspection, header analysis

Data Normalization

```
interface,collection_type=gnmi,host=telegra  
f-01,name=ethernet-1/53,source=srl1,  
admin_state="disable",oper_state="up"  
1767723255964010964
```

```
interface,collection_type=snmp,host=telegr  
af-01,agent_host=srl1,  
ifName=ethernet-1/53,ifIndex=101  
ifAdminStatus=1i,ifOperStatus=2i  
1767723255964010964
```



Enrichment: Data Without Context Is Noise

- Normalization aligns the schema. Enrichment adds meaning.
- Add from the SoT: role, location, owner, downstream services
- This is what makes data actionable for both humans and automation.

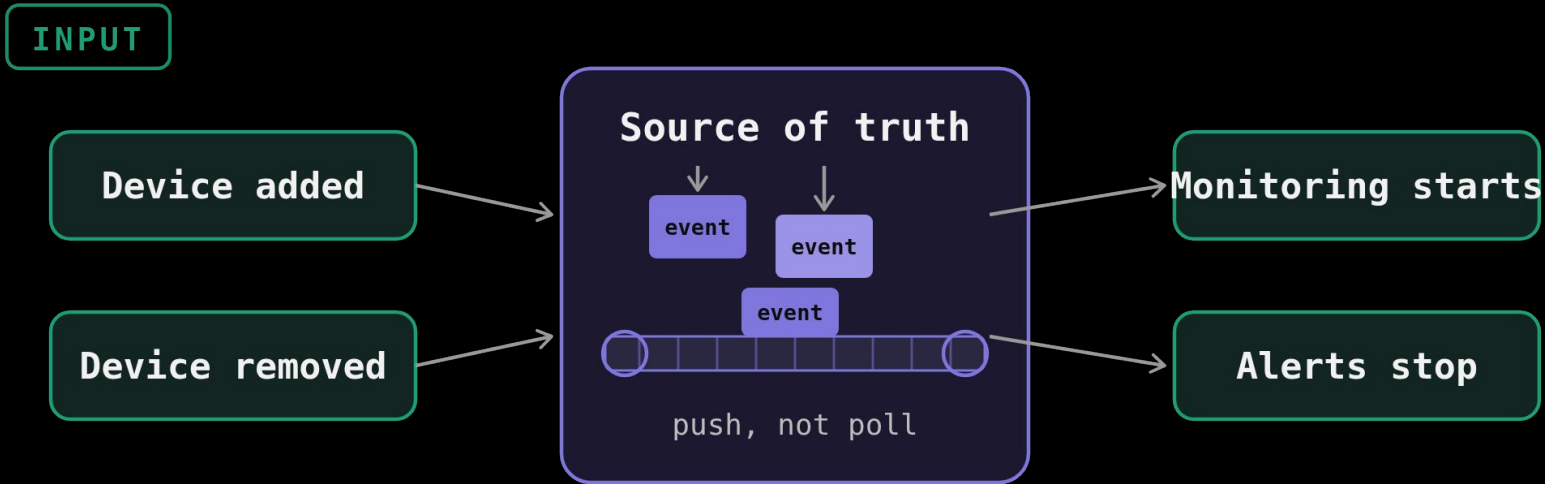
```
interface_oper_state {  
  name = "ethernet-1/53",  
  device = "sr11",  
} 1 @1767723255964010964
```



```
interface_oper_state {  
  name = "ethernet-1/53",  
  device = "sr11",  
  role = "uplink",  
  device_role = "border-router",  
  status = "maintenance",  
  location = "BCN-01",  
} 1 @1767723255964010964
```

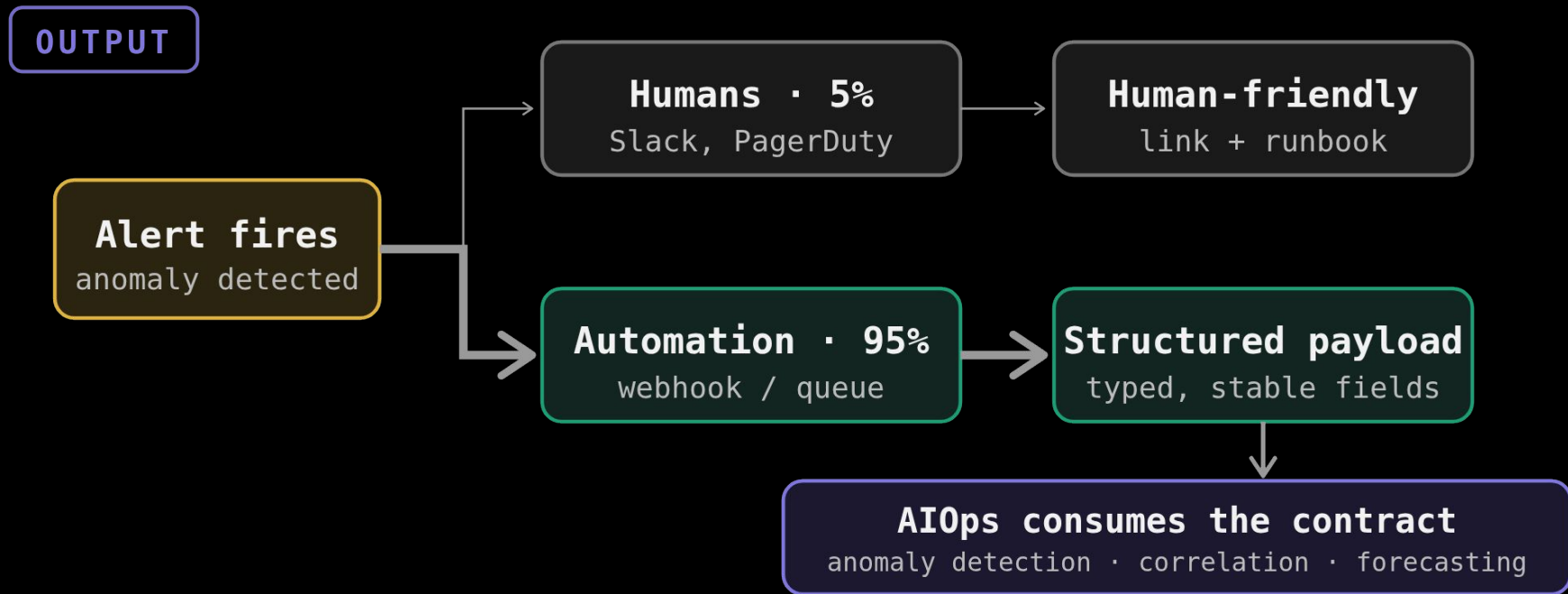
INTEGRABLE

The Source of Truth decides what gets monitored. Not you.



- No manual device lists that go stale
- Push (webhook, events) over pull (periodic sync misses mid-interval changes)

An alert is a contract, not a notification.



Observability doesn't stop at the alert.

THE LOOP

scenario: uplink errors climbing on a border router



MTTR: hours → minutes

SCALABLE

THE BETTER
YOU DO THIS,
THE MORE DATA
YOU MAKE.



More signal types, more
devices, more context =
more data, fast

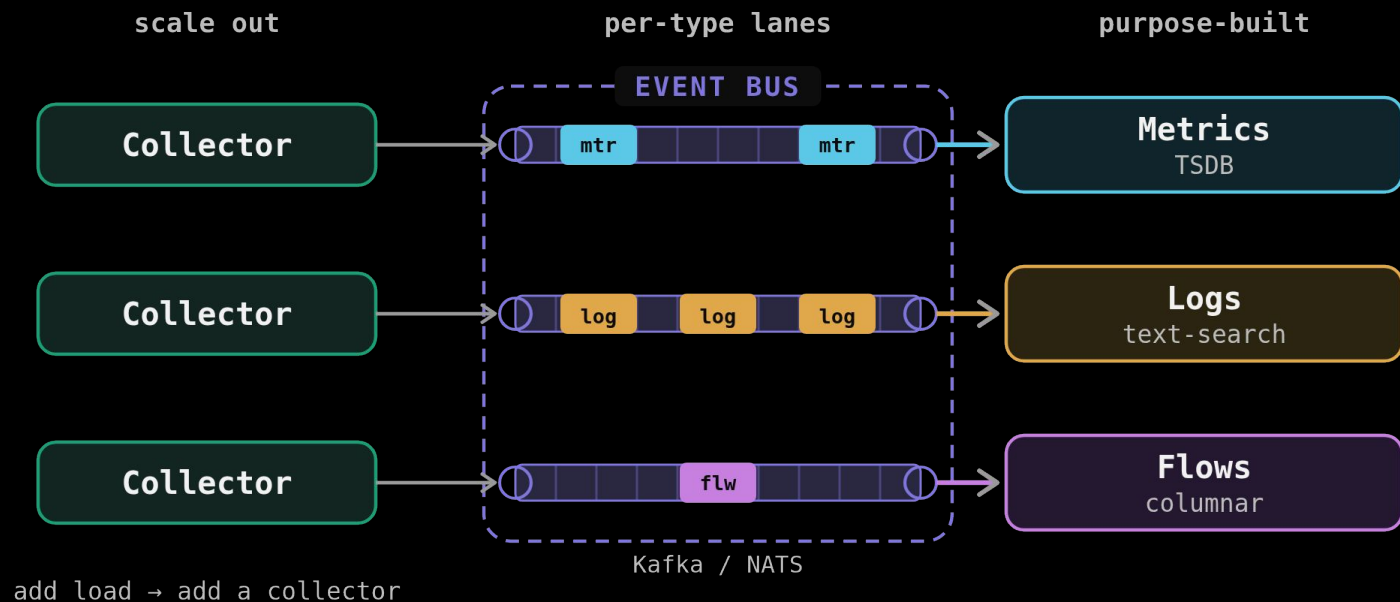


A setup fine today quietly
falls over at **10x** the load



Scaling isn't a bolt-on –
it's a choice you make **early**

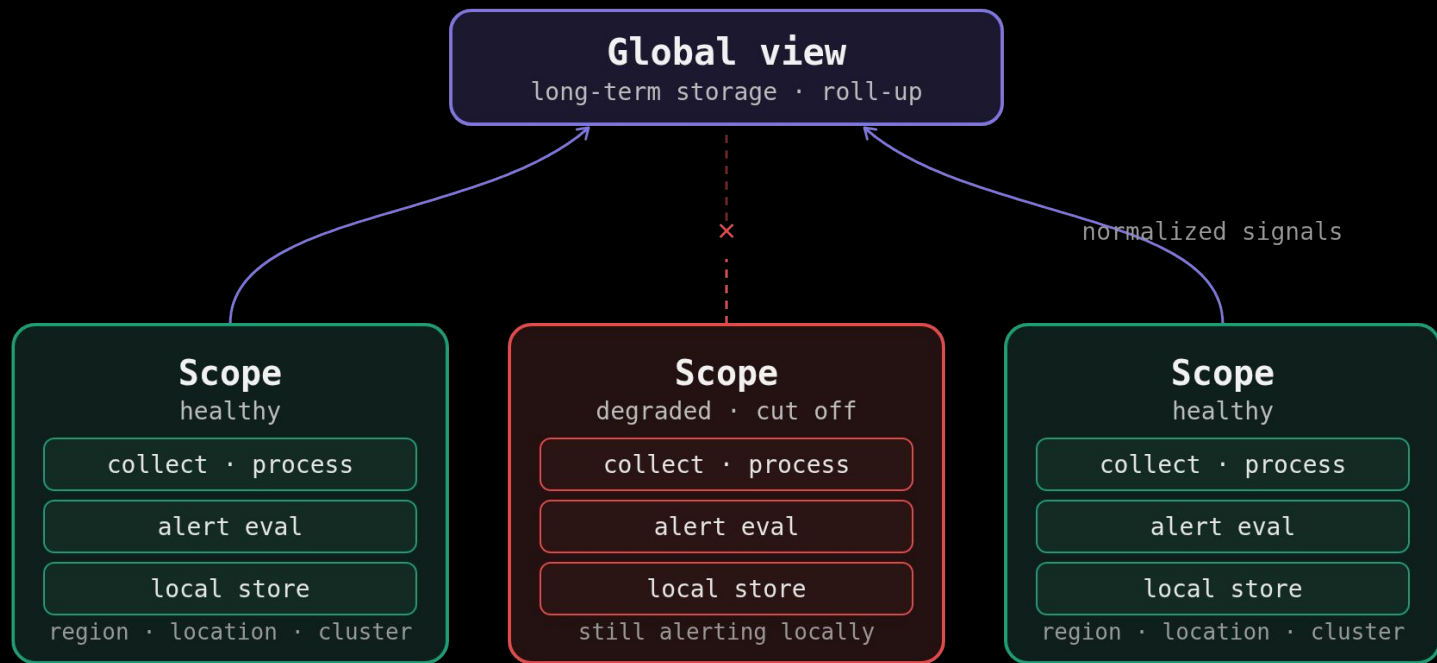
The pipeline grows by addition, not redesign.



Scale collectors out · one bus · routed to purpose-built stores

grows by addition, not redesign

Don't centralize. Federate by scope.



autonomous scopes · localized degradation · global rolls up the long term

FOUNDATION · makes change cheap

- 1 **Dynamic inventory, driven by the SoT**
never miss a device; enrichment injected across the pipeline
- 2 **Decouple the pipeline so it can expand**
you can only "add as needed" if the architecture allows it

START SMALL · don't boil the ocean

- 3 **Model your metrics, enrich with 2–3 dimensions**
a modelled approach first; only the dimensions your answers need
- 4 **Promote events into first-class alerts**
events first → promotion gate → alerts that drive workflows

ITERATE · the engine

- 5 **Iterate as new answers are needed**
collect → normalize + enrich → alert + visualize → back to collect



refine
& repeat

Observability IS the feedback loop.

- Intent → Execute → Observe → Detect → Respond → Verify
- Every system that doesn't observe is eventually flying blind
- The one part of the loop you cannot delegate to the executor

Thank You
Questions?