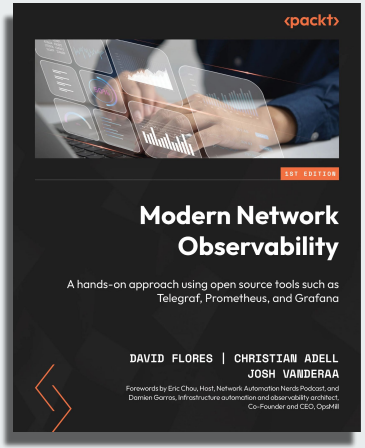
Modern Network Observability Workshop

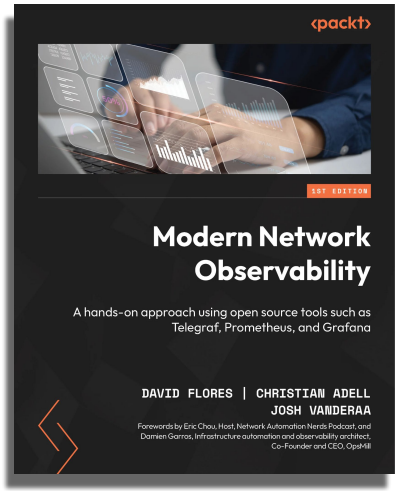
CfgMgmtCamp (04/02/2026)
Ghent

David Flores & Christian Adell



About Us: David Flores

- 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”



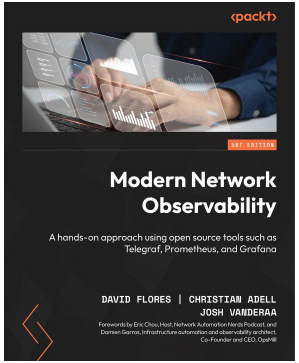
@netpanda.bsky.social - Bluesky

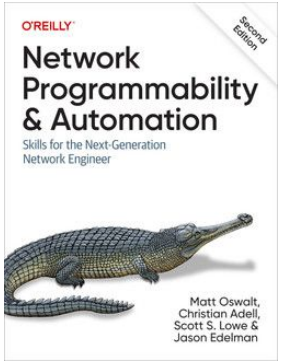
@davidban77 - Github / X

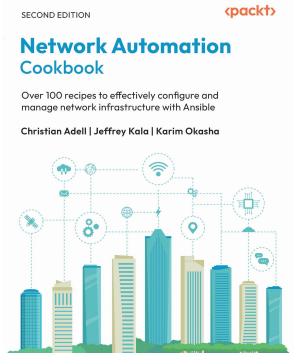
David Flores - LinkedIn

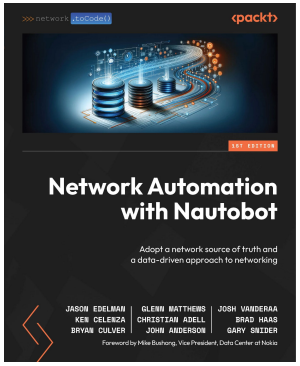
About Us: Christian Adell

- Network Automation and Observability at CoreWeave
- Broad Experience: Service and Content Provider, Enteprises, Hyperscalers, and Integrators.
- Co-Author of “Modern Network Observability” and other books
- Currently, writing an open book at https://designingnetworkautomation.com/









@chadell - Github

Christian Adell - LinkedIn

Workshop Agenda

1. From Monitoring to Observability: Understanding the Framework
2. Data Collection & Normalization: Building Trustworthy Signals
3. Turning Data into Insight: Alerting, Queries & Visualization
4. Day-2 Automation: From Observability to Intelligent Workflows

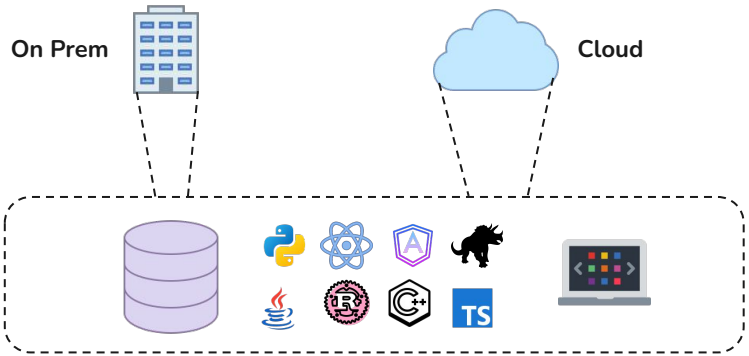
All the code is at: https://github.com/network-observability/network-observability-lab

- Base scenario: **webinar**
- Scenario solved (spoiler alert): **webinar-completed**

1

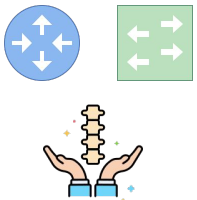
From Monitoring to Observability: Understanding the Framework

Network Monitoring Today

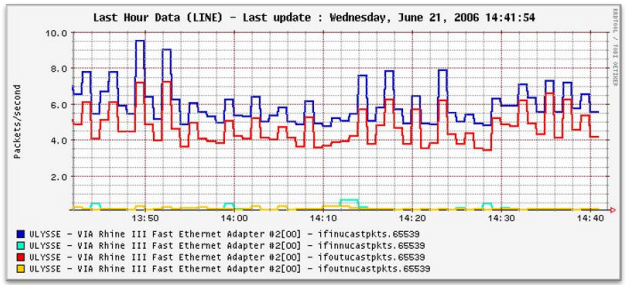


- MRTG, PRTG, Cacti – Familiar but limited.
- **Networks: The backbone of modern infrastructure.**
- Complexity is growing: cloud, on-premises, hybrid, microservices.
- Challenge? some metrics and disparate logging isn't enough for deep insights.

Servers / Virtual Machines / Containers / Kubernetes

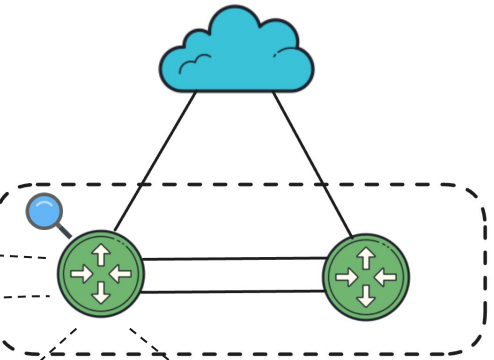


Networks



What Is Observability?

- **Observability vs. Monitoring**: “What happened” vs. “Why it happened.”
- Extracting actionable insights from **multiple system signals**.
- Distributed systems and modular architectures require new strategies.



Metrics

```
ifOutOctets=3814i, ifInOctets=469i
```

Flows

```
{
  "netflow": {
    "header": { "version": 9, "sysUptime": 3600000 },
    "template": {
      "id": 1001,
      "fields": [
        "srcIPv4",
        "dstIPv4",
        { "bgp": ["srcAS", "dstAS"] }
      ]
    },
    "data": [
      {
        "srcIPv4": "192.168.1.1",
        "dstIPv4": "10.0.0.1",
        "bgp": { "srcAS": 65001, "dstAS": 65002 }
      }
    ]
  }
}
```

Logs

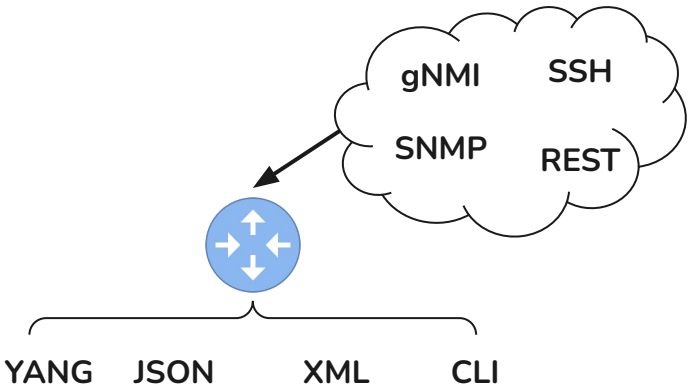
```
07:01:134 %LINEPROTO-5-UPDOWN: Line protocol on
Interface FastEthernet0/5, changed state to up
```

Synthetic Monitoring

```
Router# ping 192.0.2.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.0.2.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/6/8 ms
```

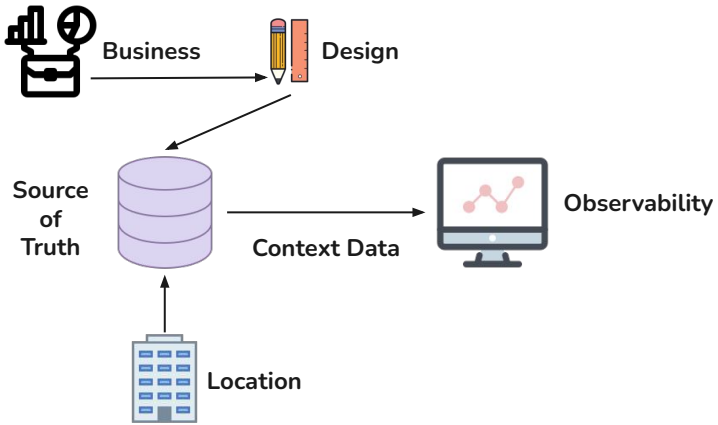
The Problem with Current Tools

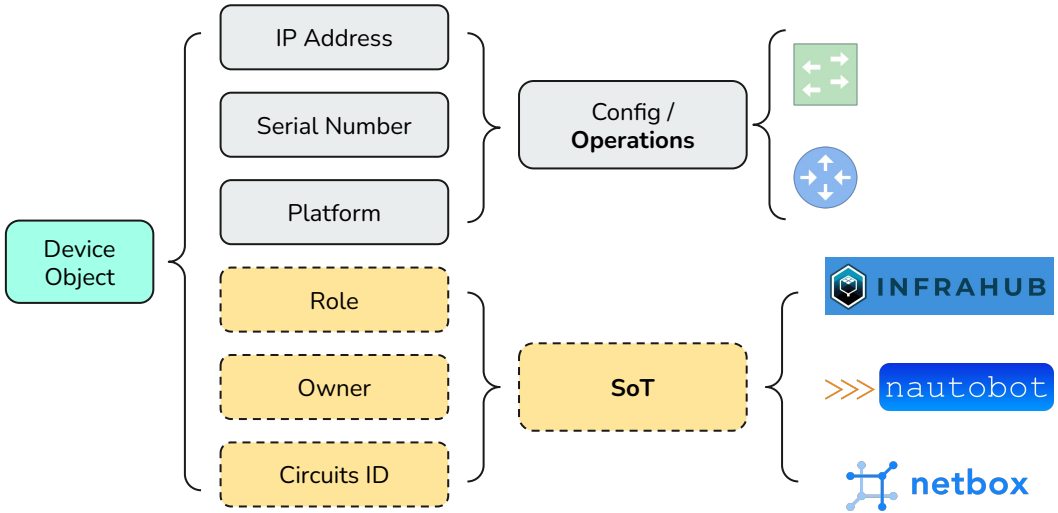
- **Vendor Lock-In**: Limited scalability and flexibility.
- **Static Dashboards**: Outdated visualizations that don’t adapt.
- **Inflexible Data Collection and Processing**: Data comes in many many many forms...



The Problem with Current Tools

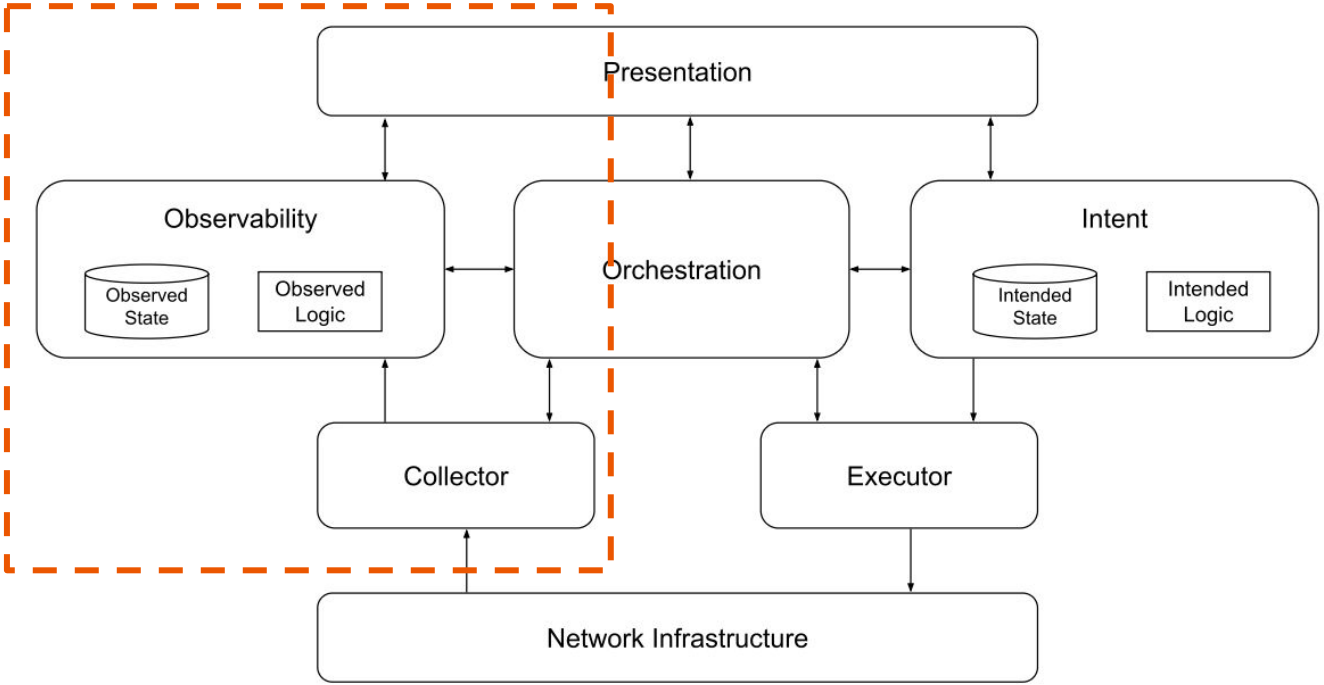
- **Missing Context**: Lack of contextual (**enrichment**) data for smarter alerts.





https://reference.networkautomation.forum/Framework/Framework/
https://nanog.org/events/nanog-96/content/5625/

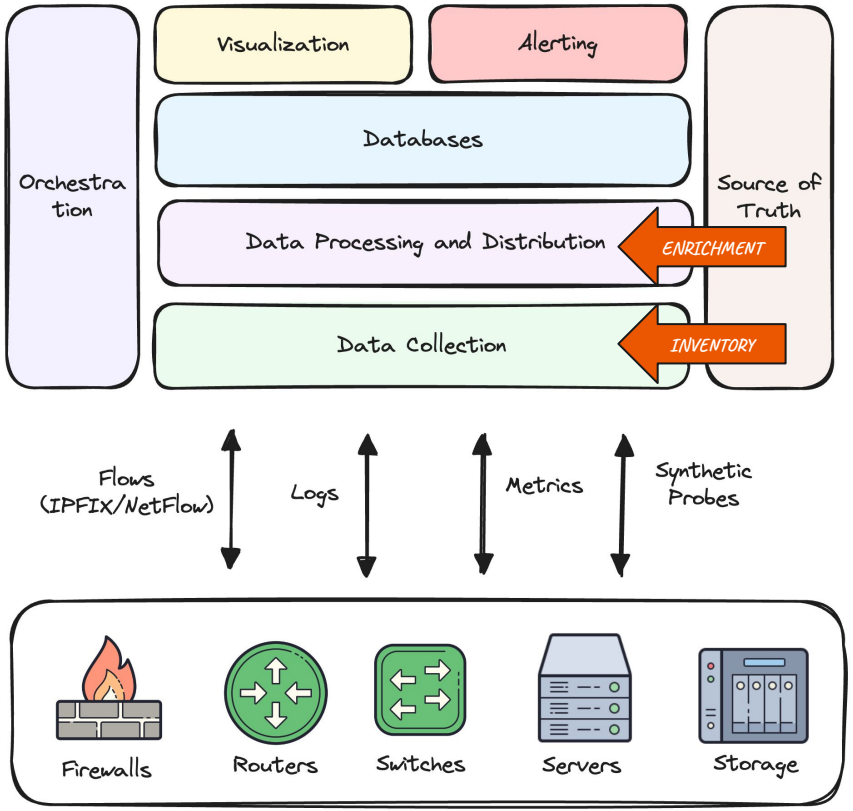
Let's talk about Architecture



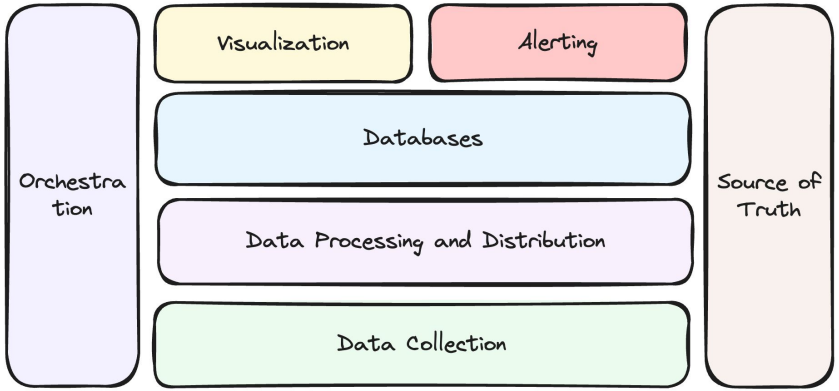
The Observability Architecture

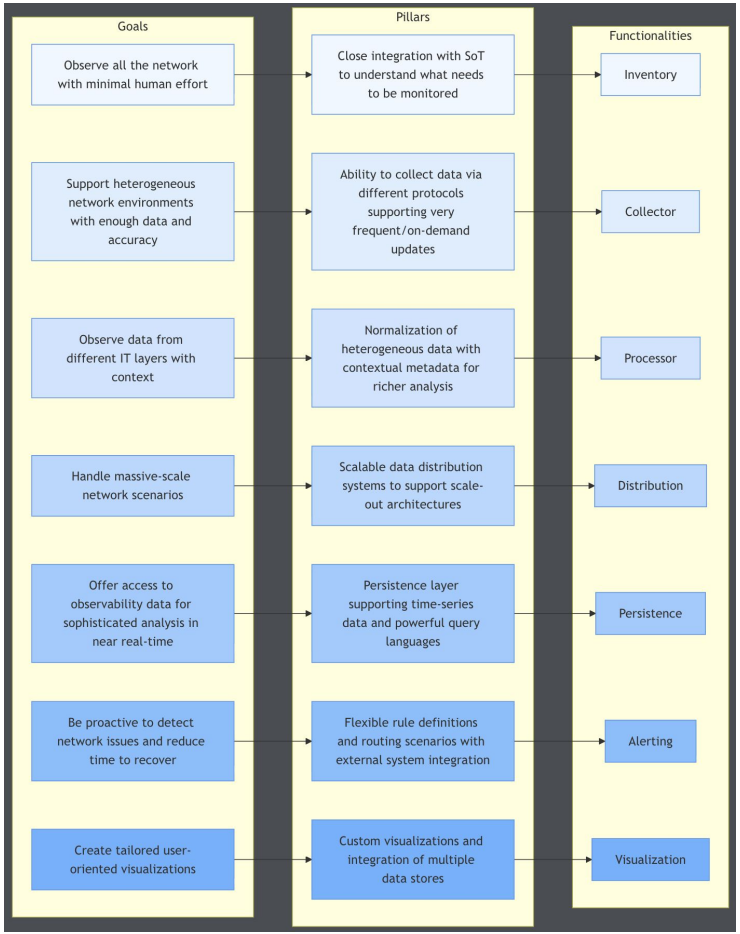
Modular architecture and **specialized** components helps addressing the challenges.

- **Collection**: Gather data from SNMP, streaming telemetry, logs.
- **Processing & Distribution**: Clean and unify diverse data.
- **Storage**: Leverage specialized databases (time-series, log storage).
- **Visualization & Alerts**: Turn data into actionable insights with tools like Grafana.



Goals - Pillars - Components



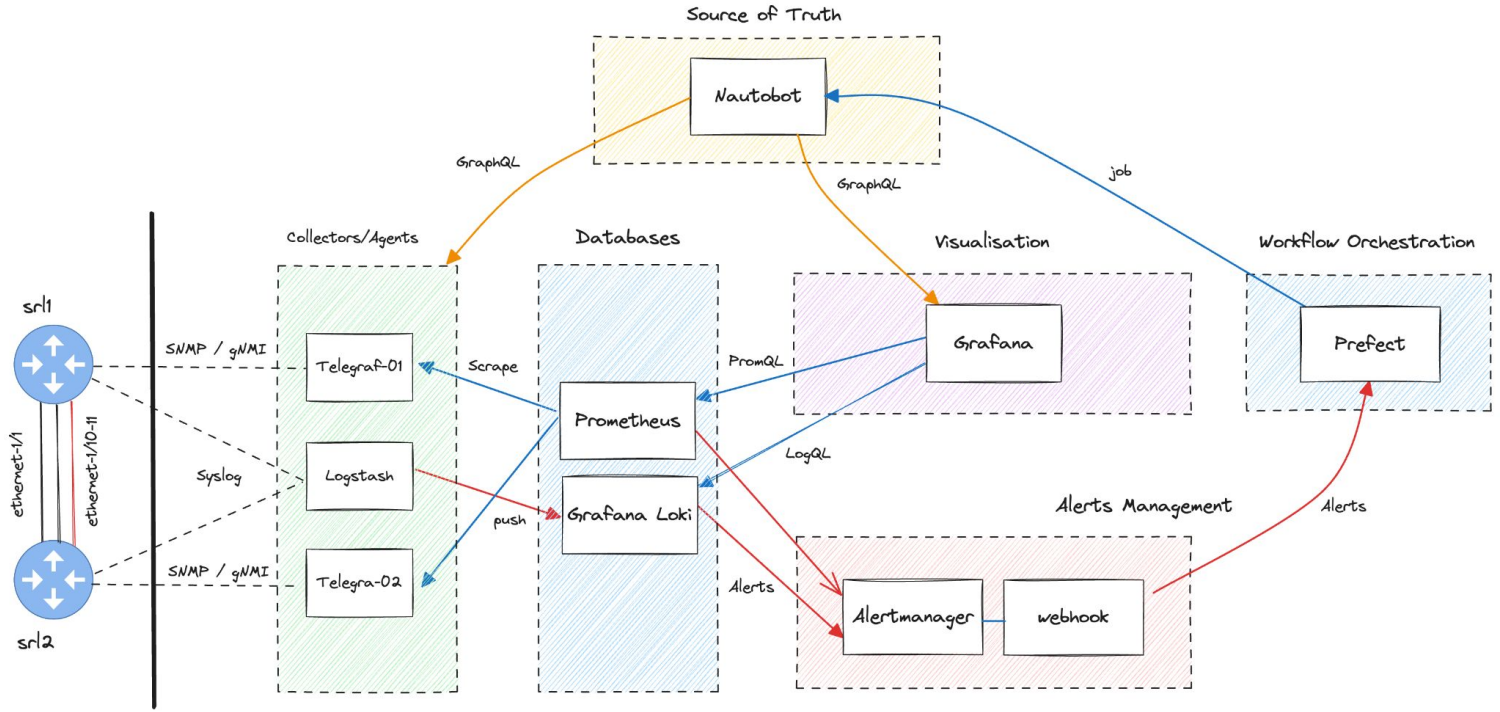


Exercise 1: Setting Up the Stack

Workshop Exercises

1. **Setting Up the Stack**
2. Build Your First Normalized Telemetry Pipeline
3. Build Alerts & Explore Data
4. Build an Observability-Driven Automation Flow

Workshop Scenario



Disclaimer: None of the elements of the scenario should be seen as a recommendation. They are used as examples, and are completely interchangeable by the tools you prefer

Setup Options

All the code: https://github.com/network-observability/network-observability-lab

Two main options:

1. Setup in Linux with Ansible
2. Unattended install in Digital Ocean

Setup in Linux with Ansible (1/2)

1. Fork (in GitHub) and Clone the Git Repository

```
$ git clone https://github.com/<your-user>/network-observability-lab.git
$ cd network-observability-lab
```

2. Install the netobs tool (use a virtualenv!)

```
(.venv)$ pip install .
```

3. Setup environment files

```
$ cp example.env .env
```

Setup in Linux with Ansible (2/2)

4. Deploy the stack with Docker (dependency)

```
(.venv)$ netobs lab deploy \
    --scenario webinar \
    --topology ./chapters/webinar/containerlab/lab.yml \
    --vars-topology ./chapters/webinar/containerlab/lab_vars.yml
```

Unattended install in Digital Ocean (1/3)

1. Create a Digital Ocean account: https://www.digitalocean.com/try/free-trial-offer
2. Fork (in GitHub) and Clone the Git Repository

```
$ git clone https://github.com/<your-user>/network-observability-lab.git
$ cd network-observability-lab
```

3. Install the netobs tool (use a virtualenv!)

```
(.venv)$ pip install .
```

Unattended install in Digital Ocean (2/3)

4. Create a Digital Ocean API token

5. Create an SSH key pair and retrieve its fingerprint

```
$ ssh-keygen -t rsa -b 4096 -C "network-observability-lab" -f ~/.ssh/id_rsa_do
```

6. Setup environment files

```
$ cp example.setup.env .setup.env

$ cp example.env .env

SSH_KEY_PATH="~/.ssh/id_rsa_do"

NETOBS_REPO="https://github.com/<your-username>/network-observability-lab.git"
```

Unattended install in Digital Ocean (3/3)

7. Create a Digital Ocean droplet (it takes some minutes)

```
(.venv)$ netobs setup deploy --scenario webinar \

  --topology ./chapters/webinar/containerlab/lab.yml \
  --vars-topology ./chapters/webinar/containerlab/lab_vars.yml
[17:15:12] Running command: ansible-playbook setup/create_droplet.yml -i
setup/inventory/localhost.yaml
Enter the droplet image [ubuntu-22-04-x64]:
Enter the droplet size [s-8vcpu-16gb]: s-4vcpu-8gb
Enter the droplet region [fra1]:
PLAY [Stand up netobs-droplet]
**********************************************************************
```

Verifying the Stack (1/4)

- (if using DO) Connect to remote server

```
$ netobs setup show
[08:45:17] Running command: ansible-playbook setup/list_droplet.yml -i
setup/inventory/do hosts.yaml
PLAY [Show Inventory] ***********************************************************
TASK [Show SSH command] *********************************************************
ok: [netobs-droplet] => {}
MSG:
ssh -o StrictHostKeyChecking=no -i ~/.ssh/id_rsa_do root@192.0.2.1
```

- Connect via SSH

```
$ ssh -o StrictHostKeyChecking=no -i ~/.ssh/id_rsa_do root@192.0.2.1
```

Verifying the Stack (2/4)

```
root@netobs-droplet:~/network-observability-lab# netobs lab show --scenario webinar  --topology chapters/webinar/containerlab/lab.yml
[17:57:23] Showing lab environment for scenario: webinar
           Showing containers for service(s): []
           Running command: docker compose --project-name netobs -f chapters/webinar/docker-compose.yml --verbose ps
NAME                        IMAGE                                       COMMAND                  SERVICE             CREATED          STATUS                    PORTS
alertmanager                docker.io/prom/alertmanager:v0.26.0         "/bin/alertmanager -…"   alertmanager        39 minutes ago   Up 39 minutes             0.0.0.0:9093->9093/tcp,
[::]:9093->9093/tcp
grafana                     docker.io/grafana/grafana:10.4.4            "/run.sh"                grafana             39 minutes ago   Up 39 minutes             0.0.0.0:3000->3000/tcp,
[::]:3000->3000/tcp
logstash                    docker.io/grafana/logstash-output-loki:3.1.1   "/usr/local/bin/dock…"   logstash            39 minutes ago   Up 39 minutes             0.0.0.0:1515->1515/tcp,
[::]:1515->1515/tcp, 0.0.0.0:9600->9600/tcp, [::]:9600->9600/tcp, 5044/tcp, 0.0.0.0:12201->12201/udp, [::]:12201->12201/udp
loki                        docker.io/grafana/loki:3.1.1                "/usr/bin/loki -conf…"   loki                39 minutes ago   Up 39 minutes             0.0.0.0:3001->3001/tcp,
[::]:3001->3001/tcp, 3100/tcp
nautobot                    docker.io/networktocode/nautobot:2.4-py3.10   "/docker-entrypoint.…"   nautobot            39 minutes ago   Up 39 minutes (unhealthy)   0.0.0.0:8080->8080/tcp,
[::]:8080->8080/tcp, 0.0.0.0:8443->8443/tcp, [::]:8443->8443/tcp
nautobot-postgres           postgres:14                                 "docker-entrypoint.s…"   nautobot-postgres   39 minutes ago   Up 39 minutes             5432/tcp
nautobot-redis              redis:7.2-alpine                            "docker-entrypoint.s…"   nautobot-redis      39 minutes ago   Up 39 minutes             6379/tcp
netobs-postgres-1           postgres:14                                 "docker-entrypoint.s…"   postgres            39 minutes ago   Up 39 minutes (healthy)   5432/tcp
netobs-prefect-server-1     prefecthq/prefect:3-latest                  "/usr/bin/tini -g --…"   prefect-server      39 minutes ago   Up 39 minutes             0.0.0.0:4200->4200/tcp,
[::]:4200->4200/tcp
netobs-prefect-services-1   prefecthq/prefect:3-latest                  "/usr/bin/tini -g --…"   prefect-services    39 minutes ago   Up 39 minutes
netobs-redis-1              redis:7                                     "docker-entrypoint.s…"   redis               39 minutes ago   Up 39 minutes (healthy)   6379/tcp
prometheus                  docker.io/prom/prometheus:v2.52.0           "/bin/prometheus --s…"   prometheus          39 minutes ago   Up 39 minutes             0.0.0.0:9090->9090/tcp,
[::]:9090->9090/tcp
telegraf-01                 netobs-telegraf-01                          "/entrypoint.sh tele…"   telegraf-01         39 minutes ago   Up 39 minutes             8092/udp, 8125/udp, 8094
/tcp, 0.0.0.0:9004->9004/tcp, [::]:9004->9004/tcp
telegraf-02                 netobs-telegraf-02                          "/entrypoint.sh tele…"   telegraf-02         39 minutes ago   Up 39 minutes             8092/udp, 8125/udp, 8094
/tcp, 0.0.0.0:9005->9005/tcp, [::]:9005->9005/tcp
webhook                     netobs-webhook                              "python -m app.main"     webhook             39 minutes ago   Up 39 minutes             0.0.0.0:9997->9997/tcp,
[::]:9997->9997/tcp
           Successfully ran: show containers
──────────────────────────────────────────────── End of task: show containers ────────────────────────────────────────────────

           Showing containerlab topology
           Topology file: chapters/webinar/containerlab/lab.yml
           Running command: sudo containerlab inspect -t chapters/webinar/containerlab/lab.yml
17:57:23 INFO Parsing & checking topology file=lab.yml
```

Name	Kind/Image	State	IPv4/6 Address
clab-mno-webinar-srl1	nokia_srlinux ghcr.io/nokia/srlinux	running	198.51.100.3 N/A
clab-mno-webinar-srl2	nokia_srlinux ghcr.io/nokia/srlinux	running	198.51.100.2 N/A

```
           Successfully ran: Inspect containerlab topology
```

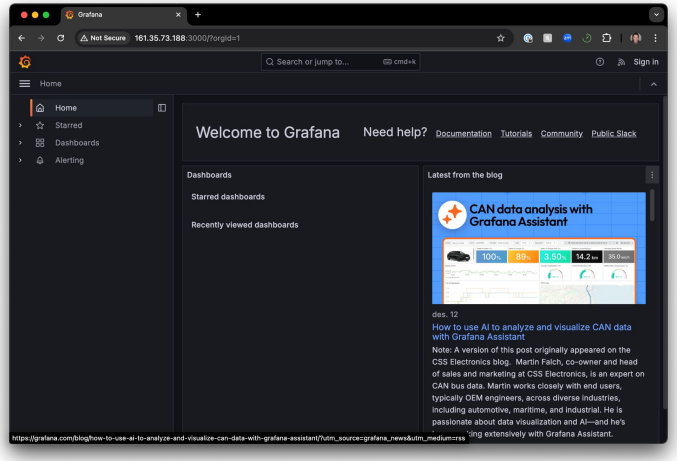
Verifying the Stack (3/4)

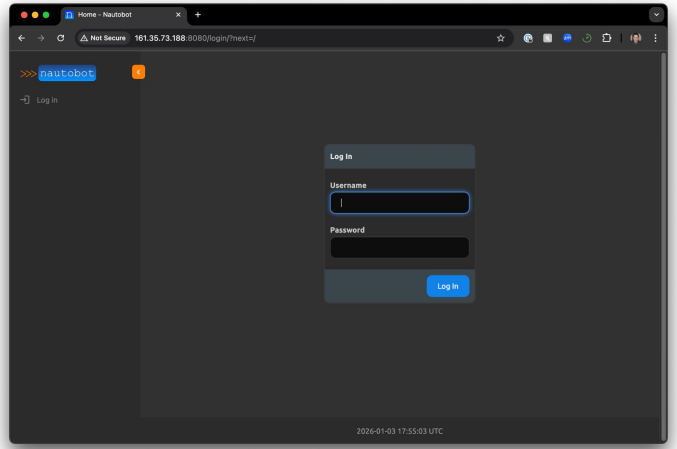
- Access SRLinux boxes and validate connectivity over virtual link (creds: *admin/NokiaSrl1!*)

```
root@netobs-droplet:~/network-observability-lab#ssh admin@srl1
Warning: Permanently added 'srl1' (ED25519) to the list of known hosts.
(admin@srl1) Password:
Last login: Tue Jan  6 11:58:42 2026 from 198.51.100.1
Loading environment configuration file(s): ['/etc/opt/srlinux/srlinux.rc']
Welcome to the Nokia SR Linux CLI.
--{ + running }--[  ]--
A:admin@srl1# ping 10.1.2.2 network-instance default
Using network instance default
PING 10.1.2.2 (10.1.2.2) 56(84) bytes of data.
64 bytes from 10.1.2.2: icmp_seq=1 ttl=64 time=115 ms
64 bytes from 10.1.2.2: icmp_seq=2 ttl=64 time=4.24 ms
```

Verifying the Stack (4/4)

- ❏ Access **Grafana** at http://<your-ip>:3000
- ❏ Access **Nautobot** at http://<your-ip>:8080





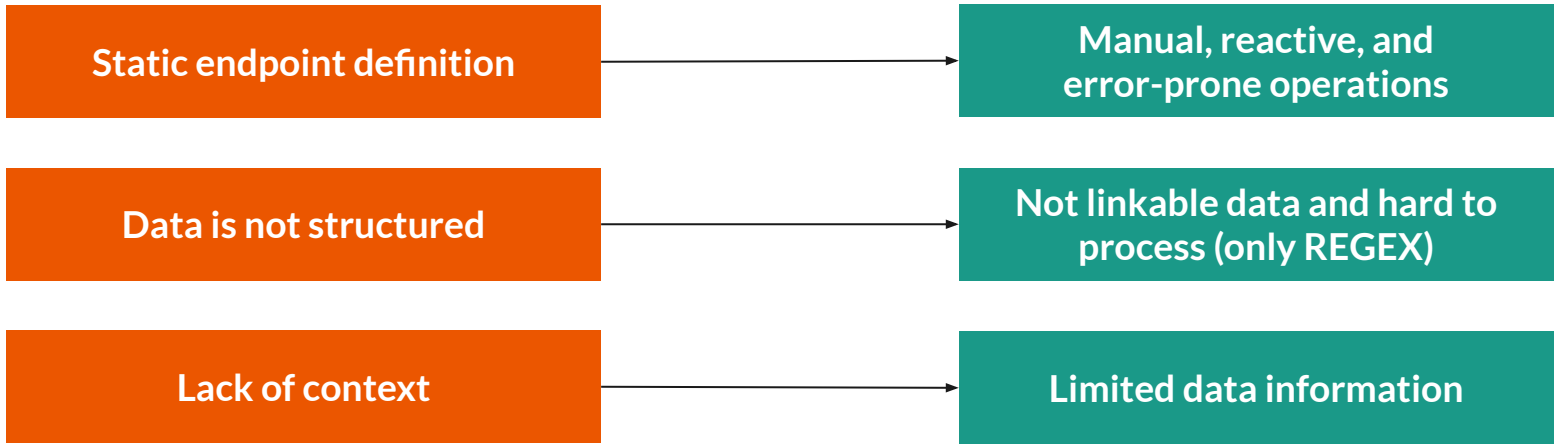
Wrap Up

- Observability goes beyond static monitoring to understand the WHY
- Within a complete network automation architecture, Observability is a key enabler
- Understanding the functionalities enables ownership

2

Data Collection & Normalization: Building Trustworthy Signals

Challenges of Traditional Data Collection

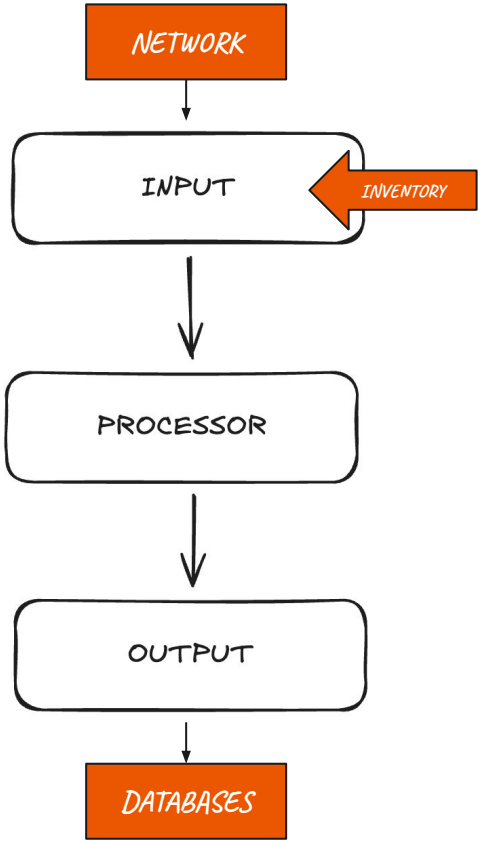


What Data Collection should look like?

- The target endpoints should be **dynamically** obtained and integrated with
- All the data type are **structured** according to a consistent data structure
- The data structure should incorporate **metadata** (labels) that allow connecting data
- The data identifier, value and labels should be able to adapt align with **standardization**
- The collector layer should be flexible to adapt to **different types** of data (using multiple tools if needed)
- The collectors should support **scale out** architecture to adapt to how the network infrastructure evolves

Collectors' Architecture

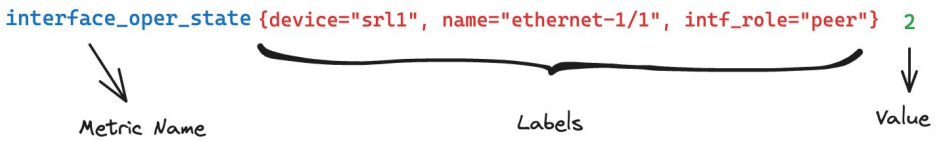
- **Input**: Defines what to collect (pull or push), the target and the paths
- **Processor**: Defines how to transforms the data: normalization and enrichment
- **Output**: Defines how to expose (passive mode) or export the data (active mode)



Classifying Data Inputs

Data Type	Protocols / Collection Methods	Notes / Examples
Metrics	SNMP, HTTP scraping, CLI polling, OpenTelemetry (OTLP), Streaming telemetry (gNMI)	Device metrics, host metrics, application metrics
Logs	OpenTelemetry (OTLP), file tailing, syslog	Application logs, system logs, structured logs
Traces	OpenTelemetry (OTLP)	Distributed tracing across services
Network Flows	NetFlow, IPFIX	Traffic flows, source/destination analysis
Protocol-specific	BMP, BGP, ARP, OSPF	BGP monitoring (BMP), ARP tables, BGP tables, OSPF tables
Packet Captures	PCAP (libpcap), SPAN / TAP	Full packet inspection, deep troubleshooting

Data Normalization



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

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

Metric Name Normalized

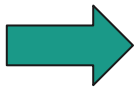
interface_oper_state {
name = “ethernet-1/53”,
device = “srl1”,
origin = “telegraf-01”,
} **1** @1767723255964010964

Labels Normalized:
- name
- device

Value Normalized:
- down: 0
- up: 1

Data Enrichment

```
interface_oper_state {
   name = “ethernet-1/53”,
   device = “srl1”,
} 1 @1767723255964010964
```



```
interface_oper_state {
   name = “ethernet-1/53”,
   device = “srl1”,
   role = “uplink”,
   device_role = “border-router”,
   status = “maintenance”,
   location = “lab-1”,
   region = “Europe”,
} 1 @1767723255964010964
```

Expanded Context

Data Output Options

- Write locally (e.g., file or local database)
- Write remotely (e.g., remote writes)
- Streaming to a message bus (e.g., Kafka)
- Expose API endpoint (e.g., HTTP scraping)

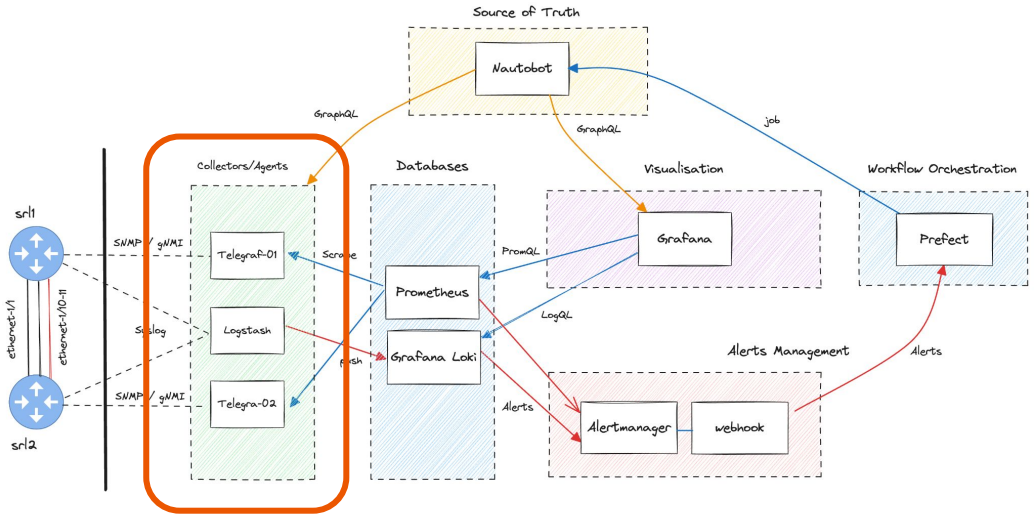
Exercise 2.1: Build Your First Normalized Telemetry Pipeline

Metrics with Telegraf

Workshop Exercises

1. Setting Up the Stack
2. **Build Your First Normalized Telemetry Pipeline**
3. Build Alerts & Explore Data
4. Build an Observability-Driven Automation Flow

Workshop Scenario




```
.
├── containerlab
│   ├── lab_vars.yml
│   ├── lab.yml
│   └── startups
│       ├── startup-config-srl1.json
│       └── startup-config-srl2.json
├── docker-compose.yml
├── grafana
│   ├── ...
├── logstash
│   └── logstash.cfg
├── loki
│   ├── ...
├── prometheus
│   ├── ...
├── README.md
├── telegraf
│   ├── telegraf-01.conf.toml
│   ├── telegraf-02.conf.toml
│   └── telegraf.Dockerfile
```

Telegraf Inputs: ICMP

```
[[inputs.ping]]
  interval = "10s"
  count = 3
  ping_interval = 1.0
  timeout = 5.0
  urls = ["srl1"]

  [inputs.ping.tags]
    collection_type = "ping"
    watcher_type = "healthcheck"
    service = "srl1"
```

Telegraf Inputs: SNMP

```
[[inputs.snmp]]
  agents = ["srl1"]
  version = 2
  community = "${SNMP_COMMUNITY}"
  interval = "60s"
  timeout = "10s"
  retries = 3

 [[inputs.snmp.tags]]
    collection_type = "snmp"

 [[inputs.snmp.table]]
    name = "interface"

    [[inputs.snmp.table.field]]
      name = "name"
      # IF-MIB::ifDescr
      oid = "1.3.6.1.2.1.2.2.1.2"
      is_tag = true
```

Telegraf Inputs: gNMI

```
[[inputs.gnmi]]
  addresses = ["srl1:57400"]
  username = "${NETWORK_AGENT_USER}"
  password = "${NETWORK_AGENT_PASSWORD}"
  redial = "20s"
  tagexclude = ["path"]
  tls_enable = true
  insecure_skip_verify = true

  [inputs.gnmi.tags]
    collection_type = "gnmi"

  [[inputs.gnmi.subscription]]
    name = "interface"
    path ="/interface[name=*]/statistics"
    subscription_mode = "sample"
    sample_interval = "10s"
```

Other options:

- Kubernetes
- Docker
- Cloud provider services
- HTTP
- Databases
- Message Queues
- Tail (files, logs)
- **EXEC**

Telegraf Outputs

```
[[outputs.file]]
  files = ["stdout"]

[[outputs.prometheus_client]]
  # HTTP port to listen on
  listen = ":9004"
```

Other options:

- Remote Write to TSDB: Prometheus, VictoriaMetrics, InfluxDB
- Message buses: Kafka, NATS, AWS Kinesis
- Managed Services: Datadog, NewRelic
- Files

Applying Telegraf changes

```
root@netobs-droplet:~/network-observability-lab# netobs lab update telegraf-01 -s webinar
[18:07:43] Updating lab environment for scenario: webinar
                Removing service(s): ['telegraf-01']
      Running command: docker compose --project-name netobs -f chapters/webinar/docker-compose.yml --verbose rm --stop --force --volumes  telegraf-01
[+] stop 1/1
[+] remove 1/1elegraf-01 Stopped                                                0.5s
 ✔ Container telegraf-01 Removed                                                0.7s
[18:07:44] Successfully ran: remove containers
———————————————————— End of task: remove containers ————————————————————————————————

      Starting service(s): ['telegraf-01']
      Running command: docker compose --project-name netobs -f chapters/webinar/docker-compose.yml --verbose up -d --remove-orphans telegraf-01
WARN[0000] No services to build
[+] up 1/1
 ✔ Container telegraf-01 Created                                                0.1s
[18:07:46] Successfully ran: start stack
————————————————————End of task: start stack ————————————————————————————————————

      Lab environment updated for scenario: webinar
```

Getting Telegraf Outputs (without processing)

```
root@netobs-droplet:~/network-observability-lab# netobs docker logs telegraf-01 --tail 10
[18:11:01] Showing logs for service(s): ['telegraf-01']
        Running command: docker compose --project-name netobs -f chapters/batteries-included/docker-compose.yml logs --tail=10 telegraf-01
telegraf-01  | interface,agent_host=srl1,collection_type=snmp,host=telegraf-01,name=ethernet-1/40 in_octets=0i,out_octets=0i 1767723300000000000
telegraf-01  | interface,agent_host=srl1,collection_type=snmp,host=telegraf-01,name=ethernet-1/3 in_octets=0i,out_octets=0i 1767723300000000000
telegraf-01  | interface,collection_type=gnmi,host=telegraf-01,name=ethernet-1/53,source=srl1 admin_state="disable",oper_state="down" 1767723255964010964
telegraf-01  | interface,collection_type=gnmi,host=telegraf-01,name=ethernet-1/54,source=srl1 admin_state="disable",oper_state="down" 1767723255964010964
telegraf-01  | interface,collection_type=gnmi,host=telegraf-01,name=ethernet-1/55,source=srl1 admin_state="disable",oper_state="down" 1767723255964010964
telegraf-01  | interface,collection_type=gnmi,host=telegraf-01,name=ethernet-1/56,source=srl1 admin_state="disable",oper_state="down" 1767723255964010964
telegraf-01  | interface,collection_type=gnmi,host=telegraf-01,name=ethernet-1/57,source=srl1 admin_state="disable",oper_state="down" 1767723255964139249
telegraf-01  | interface,collection_type=gnmi,host=telegraf-01,name=ethernet-1/58,source=srl1 admin_state="disable",oper_state="down" 1767723255964139249
telegraf-01  | interface,collection_type=gnmi,host=telegraf-01,name=mgmt0,source=srl1 admin_state="enable",oper_state="up" 1767723255964139249
telegraf-01  | ping,collection_type=ping,host=telegraf-01,service=srl1,url=srl1,watcher_type=healthcheck
percent_packet_loss=0,minimum_response_ms=2.385,average_response_ms=3.747,result_code=0i,packets_transmitted=3i,packets_received=3i,ttl=64i,maximum_response_ms=4.677,standard_deviation_ms=0.984 1767723262000000000

        Successfully ran: show logs
──────────────────────────────────────────────── End of task: show logs ────────────────────────────────────────────────
```

Implementing Telegraf Normalization

```
[[processors.rename]]
  [[processors.rename.tagpass]]
    collection_type = ["snmp"]

  [[processors.rename.replace]]
    tag = "agent_host"
    dest = "device"
```

```
[[processors.rename]]
  [[processors.rename.tagpass]]
    collection_type = ["gnmi"]

  [[processors.rename.replace]]
    tag = "source"
    dest = "device"
```

```
[[processors.rename]]
  [[processors.rename.tagpass]]
    collection_type = ["ping"]

  [[processors.rename.replace]]
    tag = "url"
    dest = "device"
```

Implementing Telegraf Enrichment

```
[[processors.regex]]
  namepass = ["interface"]
  [processors.regex.tagpass]
    device = ["srl1"]
  [[processors.regex.tags]]
    key = "name"
    pattern = "^Ethernet.*$"
    result_key = "intf_role"
    replacement = "peer"
  [[processors.regex.tags]]
    key = "name"
    pattern = "^Management.*$"
    result_key = "intf_role"
    replacement = "mgmt"
```

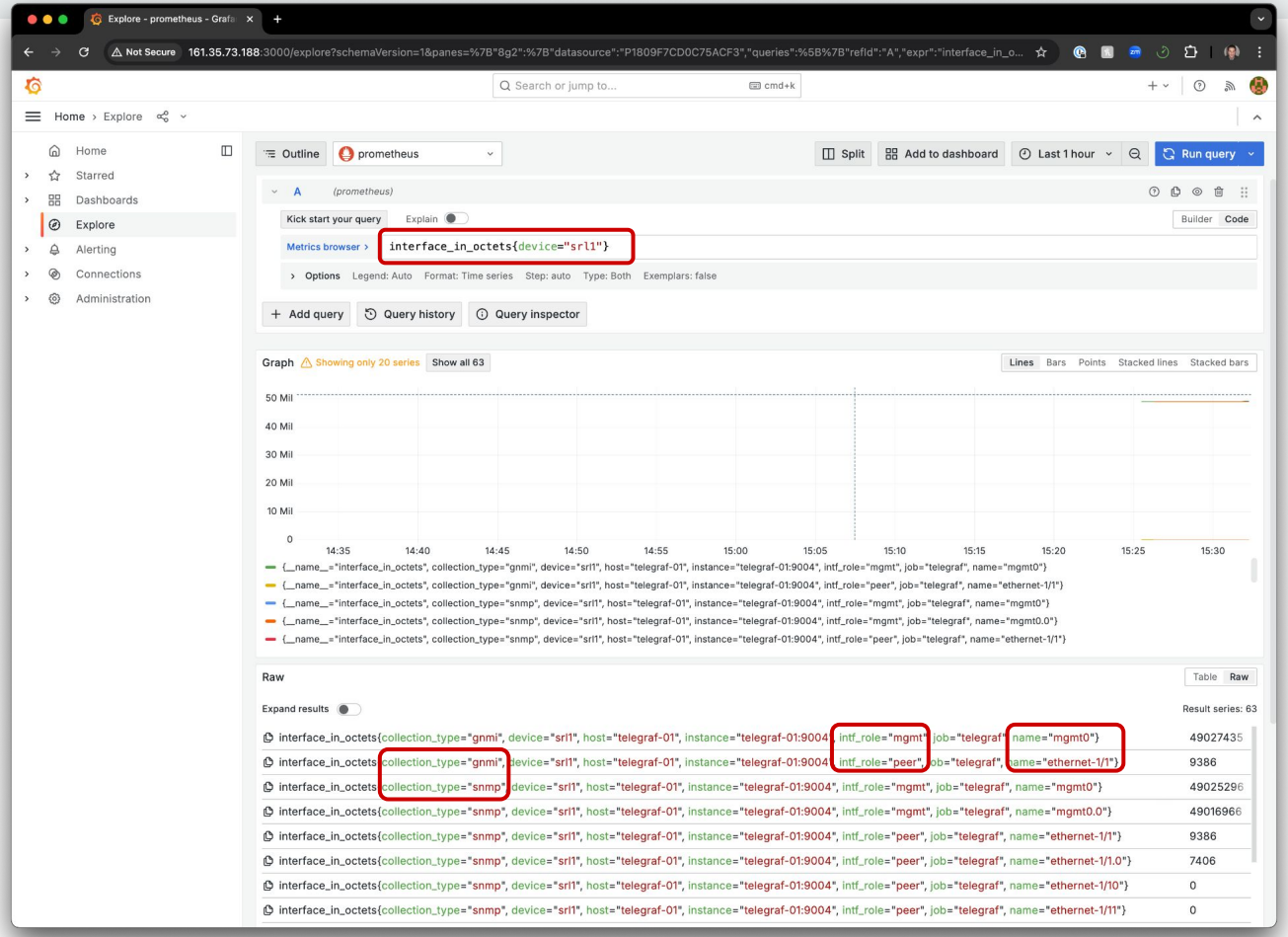
Other options:

- override
- converter
- enum
- filter
- **starlark**
- printer
- date

Getting Telegraf outputs (processed)

root@netobs-droplet:~/network-observability-lab# **netobs docker logs telegraf-01 --tail 10**
[18:11:01] Showing logs for service(s): ['telegraf-01']
Running command: docker compose --project-name netobs -f chapters/batteries-included/docker-compose.yml logs --tail=10 telegraf-01
telegraf-01 | interface,collection_type=gnmi,**device=srl1**,host=telegraf-01,**intf_role=peer**,name=ethernet-1/58 oper_state=2i,admin_state=2i 1768143286055505407
telegraf-01 | interface,collection_type=gnmi,**device=srl1**,host=telegraf-01,**intf_role=mgmt**,name=mgmt0 oper_state=1i,admin_state=1i 1768143286055505407
telegraf-01 | cpu,collection_type=gnmi,**device=srl1**,host=telegraf-01,index=all,slot=A used=41i 1768143286057268129
telegraf-01 | memory,collection_type=gnmi,**device=srl1**,host=telegraf-01,slot=A utilization=67i 1768143286057436961
telegraf-01 | interface,collection_type=snmp,**device=srl1**,host=telegraf-01,**intf_role=peer**,name=ethernet-1/50 in_octets=0i,out_octets=0i 1768142580000000000
telegraf-01 | interface,collection_type=snmp,**device=srl1**,host=telegraf-01,**intf_role=peer**,name=ethernet-1/30 out_octets=0i,in_octets=0i 1768142580000000000
telegraf-01 | interface,collection_type=snmp,**device=srl1**,host=telegraf-01,**intf_role=peer**,name=ethernet-1/32 in_octets=0i,out_octets=0i 1768142580000000000
telegraf-01 | ping,collection_type=ping,**device=srl1**,host=telegraf-01,service=srl1,watcher_type=healthcheck
minimum_response_ms=4.032,maximum_response_ms=4.594,standard_deviation_ms=0.231,percent_packet_loss=0,ttl=64i,average_response_ms=4.336,result_code=0i,packets_transmitted=3i,packets_received=3i 1768142592000000000
———————————————————————————— End of task: show logs ————————————————————————————

Verify Metrics in Storage



Exercise 2.2: Build Your First Normalized Telemetry Pipeline

Logs with Logstash

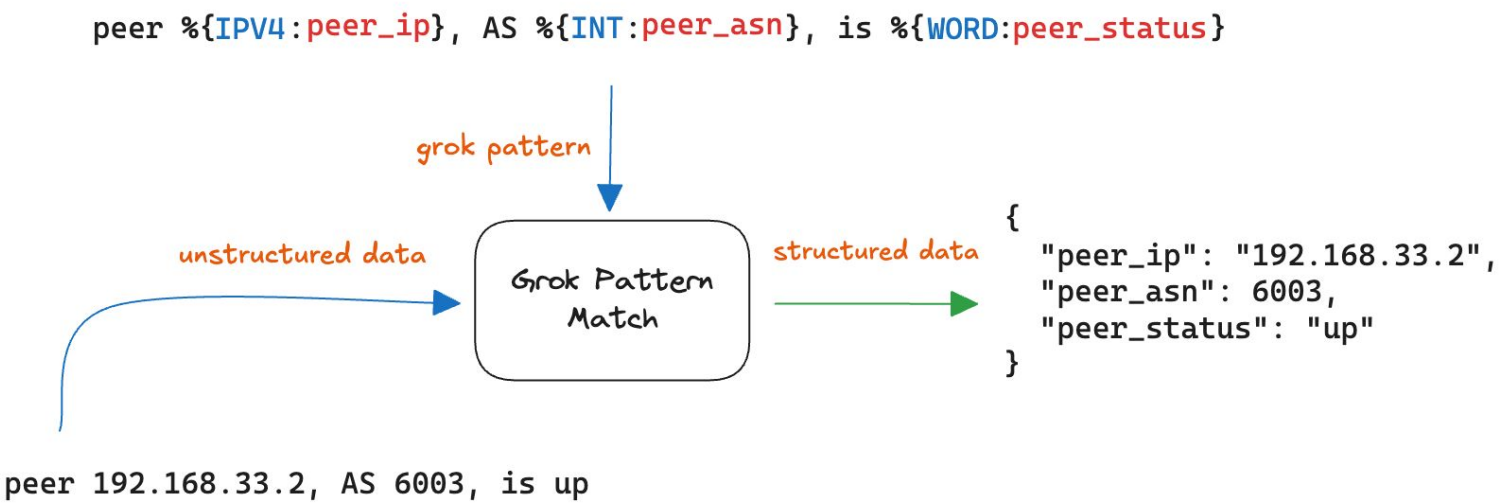
Logstash Input

```
input {
    syslog {
        port => 1515
        use_labels => true
        id => "srlinux"
        tags => [ "syslog", "srlinux" ]
        timezone => "Europe/Rome"
    }
}
```

Other options:

- beats
- raw tcp/udp
- http
- kafka
- file

Logstash Grok



Logstash Grok and Mutate

```
filter {
    if [type] == "syslog" {
        grok {
            patterns_dir => [ "/var/lib/logstash/patterns" ]
            match => { "message" =>
"%{SRLPROC:subsystem}\|%{SRLPID:pid}\|%{SRLTHR:thread}\|%{SRLS
EQ:sequence}\|%{SRLLVL:initial}:\s+(?<log_message>(.*))" }
        }
        mutate {
            rename => { "severity_label" => "level" }
            lowercase => [ "level" ]
        }
        # Add a field for metadata example
        mutate {
            add_field => {
                "netpanda" => "bethepacket"
            }
        }
```

Other options:

- json
- date
- geoip
- drop
- translate
- **ruby**

Logstash Output

Other options:

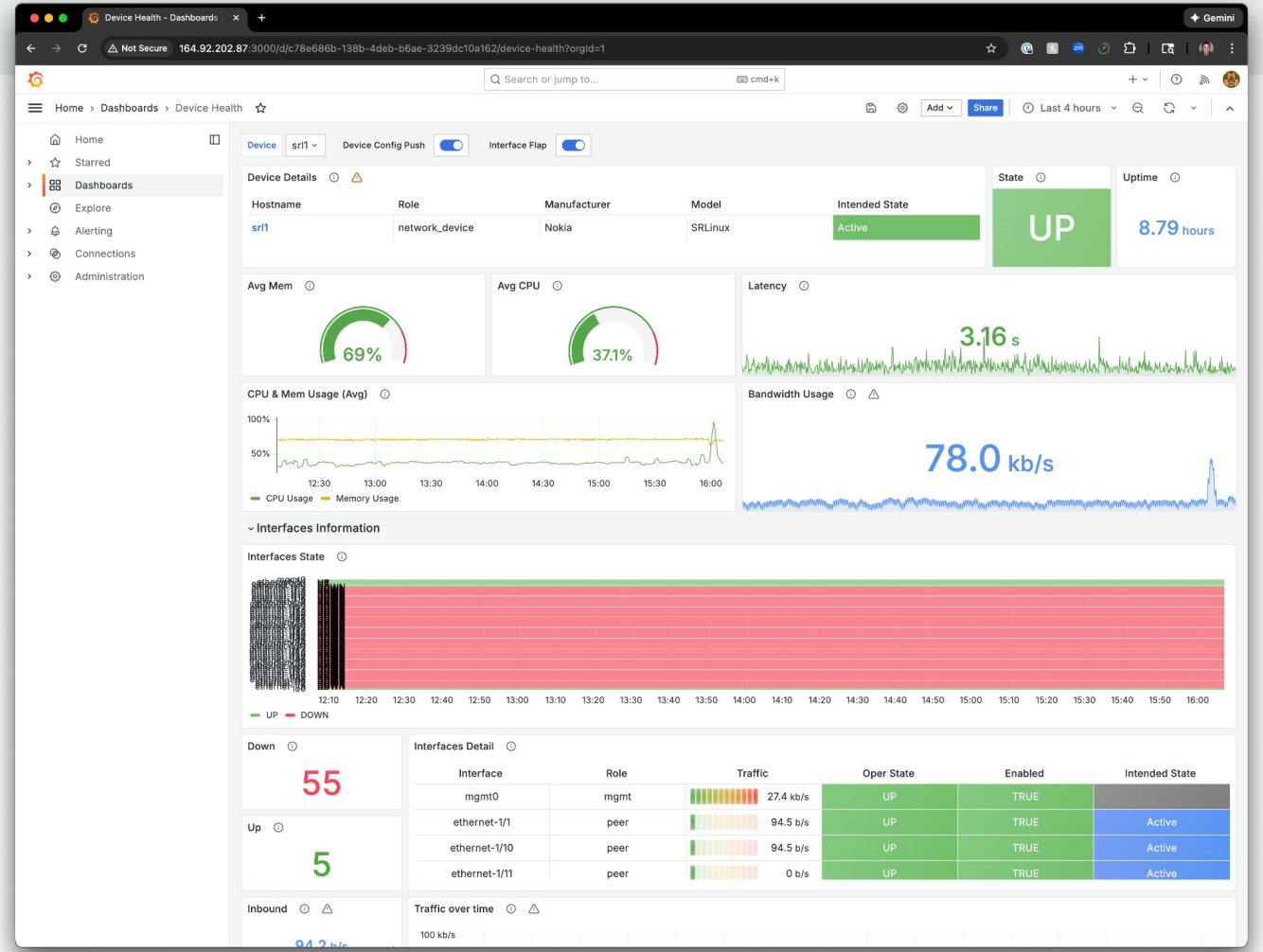
- elasticsearch
- kafka
- stdout
- filte
- http

```
output {
    if [type] == "syslog" {
        loki {
            url => "http://loki:3001/loki/api/v1/push"
            metadata_fields => ["netpanda"]
            include_fields => [
                "device",
                "level",
                "facility",
                "facility_label",
                "vendor_facility",
                "vendor_facility_process",
                "event_type",
                "interface",
                "interface_status",
                "priority",
                "severity",
                "type"
            ]
        }
    }
}
```

Getting Logstash outputs

```
root@netobs-droplet:~/network-observability-lab# netobs docker logs logstash --tail 16
[16:52:40] Showing logs for service(s): ['logstash']
        Running command: docker compose --project-name netobs -f
chapters/batteries-included/docker-compose.yml logs --tail=16 logstash
logstash  | {
logstash  |         "message" => "<180>1 2026-02-01T16:52:27.984497+00:00 srl1
sr_xdp_lc_1 - - -  debug|2639|3138|00170|W: csim_pd   csim_platform.cc:2553
UpdateLicenseValidity  No valid license, limiting packet rate to 10000pps\n",
logstash  |        "facility" => 0,
logstash  |          "host" => "198.51.100.2",
logstash  |        "severity" => 0,
logstash  |        "priority" => 0,
logstash  |        "@version" => "1",
logstash  |      "@timestamp" => 2026-02-01T16:52:27.989Z,
logstash  |    "facility_label" => "kernel",
logstash  |           "tags" => [
logstash  |      [0] "syslog",
logstash  |      [1] "srlinux",
logstash  |      [2] "_grokparsefailure_sysloginput"
logstash  |    ],
logstash  |    "severity_label" => "Emergency"
logstash  | }
        Successfully ran: show logs
```

Visualize Metrics and Logs in a Graph



Wrap Up

In this scenario:

- You learn about the challenges of network data collection
- The basic collector architecture is INPUT -> PROCESS -> OUTPUT
- You implemented metrics collection with Telegraf and logs with Logstash
    - Pick the right tool for each ocasion
- Data collection is just the beginning of the Observability journey

3

Turning Data Into Insight: Alerting, Queries & Visualization

Interpretation layer: queries → rules → alerts → dashboards

Turning Data Into Insight: Alerting, Queries & Visualization

- In Section 2 we *collected* metrics + logs
- In this sections we will learn how to **use** them:
  - **Explore** the raw signals (metrics + logs)
  - **Query** them into answers
  - **Alert** on decisions (intent vs reality)
  - **Visualize** health + history + context in dashboard

Grafana - `http://<lab-address>:3000`

User: **`netobs`**

Password: **`netobs123`**

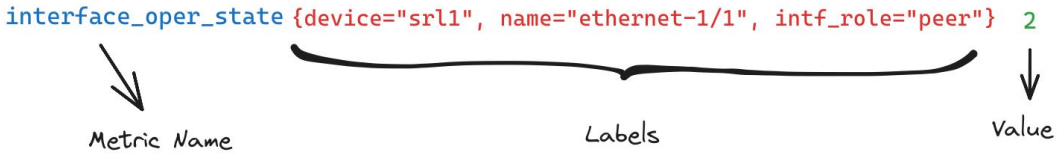
Exploring the data: Metrics taxonomy

- Metric name: `interface_oper_state`
- Labels:
  - `device="srl1"` (which box/device)
  - `name="ethernet-1/1"` (which interface)
  - `intf_role="peer"` (semantic role from enrichment)
  - Sample value: 2 (meaning depends on mapping)
  - Timestamp: (implicit per-sample)

Queries interpret signals into answers:

How many interfaces are **up** per device?

```
count by (device) (
  interface_oper_state{intf_role="peer"} == 1
)
```



1 = UP

2 = DOWN

Exploring the data: Prep - traffic generation

```
> ssh admin@srl1

Password: NokiaSrl1!

Welcome to the Nokia SR Linux CLI.

--{ + running }--[  ]--

A:admin@srl1#

--{ + running }--[  ]--

A:admin@srl1# ping 10.1.2.2 network-instance default -s 1400 -M do -i
0.5
```

Exploring the data: PromQL

Query: What is the inbound throughput of a network device?

1. We identify the metric (interface_in_octets) and filter for an specific device and interface

```
interface_in_octets{device="srl1", name="ethernet-1/1"}
```

2. Metrics are of type **counter**, and we need to calculate the traffic using rate() function

```
rate(interface_in_octets{device="srl1", name="ethernet-1/1", collection_type="gnmi"})[2m] * 8
```

3. Now we can sum() the traffic of all in-band interfaces to get the inbound throughput of a device

```
sum(rate(interface_in_octets{device="srl1", name!~"mgmt0.*", collection_type="gnmi"}[2m])) * 8
```

4. Now per device

```
sum  by (device) rate(interface_in_octets{name!~"mgmt0.*", collection_type="gnmi"}[2m]) * 8
```

Exploring the data: PromQL (Part 2)

Query: What are the interfaces of the devices that are enabled but operationally up?

1. Check **interfaces state is enabled** (these are enum-like metrics where the value tells you the state)

```
interface_admin_state{intf_role="peer"} == 1
```

2. Check **interfaces operational state = up**

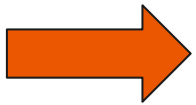
```
interface_oper_state{intf_role="peer"} == 1
```

3. Match the interfaces and the devices from both views

```
and on (device, name)
```

4. Group results by device + interface

```
count by (device, name) ( … )
```



Resulting Query:

```
count by (device, name) (
  (interface_admin_state{intf_role="peer"} == 1)
  and on (device, name)
  (interface_oper_state{intf_role="peer"} == 1)
)
```

Exercise 3.1: Explore the Metrics Data

Lab 3.1 Metrics Discovery

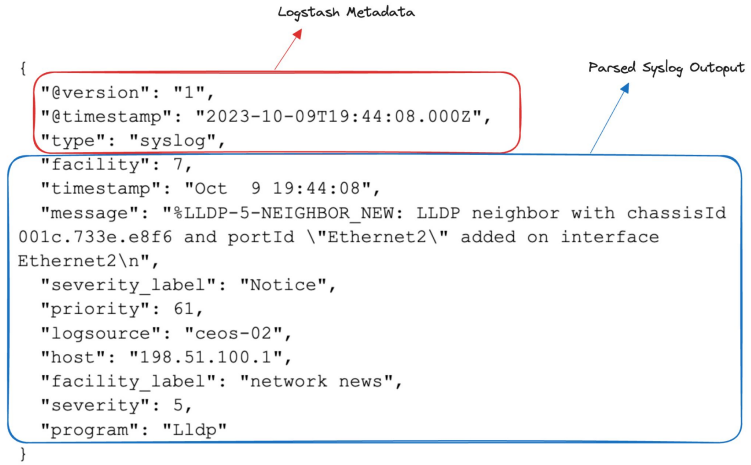
1. “How many peer interfaces are ENABLED vs DISABLED on a device?”
2. “How many BGP sessions are enabled but operationally down?”
    a. HINT: bgp_admin_state and bgp_oper_state
    b. HINT: Look for the value-mapping in your telegraf configuration)
3. “What is the total amount of BGP IPv4 received routes from a neighboring device?”
    a. HINT: bgp_received_routes
    b. HINT: Filter on afi_safi_name="ipv4-unicast"
4. “Show the interfaces that have flapped more than 3 times in the span of 2 minutes
    a. HINT: interface_carrier_transitions is a counter for interface state changes on its carrier.
    b. HINT: Use rate/increase functions to determine the rate of change in the span of 2 minutes
    c. HINT: Trigger interfaces flap with - `netobs utils device-interface-flap --device srl1 --interface ethernet-1/1 --count 10 --platform nokia_srl`

Exploring the data: Logs taxonomy

- A log entry is defined by labels (indexed)

```
{device="srl1", job="netdev-logs", facility_label="srlinux"}
```

- A log content is the message/line content (not indexed)
- Rule of thumb:
  - Labels = fast stream selection (indexed)
  - Line fields/message = filter after selection
- LogQL taxonomy



Stream Selector

Log Pipeline (Optional)

```
{ label1="value1", label2="value2" } <filter operator> <log pipeline expression>
```

Exploring the data: Prep - interface flap

```
# Shut/Unshut interface 10 times with a delay of 10 seconds

> netobs utils device-interface-flap \

   --device srl1 \

   --interface ethernet-1/1 \

   --count 10 \

   --delay 10 \

   --platform nokia_srl
```

Exploring the data: LogQL

Query: How frequent do we have someone enabling/disabling interfaces?

1. We identify the logs for **`srl1`**

```
{device="srl1"}
```

2. Filter license-related messages to reduce noise

```
{device="srl1"} != "license"
```

3. Next we are interested in the **`admin-state`** changes that we have made on the device

```
{device="srl1"} != "license" |= "set / interface" |= "admin-state"
```

4. Now, let's create a metric of the rate of changes performed on the interfaces of the device

```
sum(rate({device="srl1"} != "license" |= "set / interface" |= "admin-state" [5m]))
```

Exercise 3.2: Explore the Logs Data

Lab 3.2 Logs Discovery

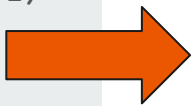
1. “Can I get only the **`enable`** events from the interfaces?”
2. “Can you filter only when an user is connected to the device and running commands?”
    a. HINT: **`aaa`** and **`debug`** modules provide overall view of commands and VTY sessions to the device

Actioning on data: Alerts

- A **query** answers a question
- An **alert** is the same query plus: “if **true** -> **act**”

Decision: “Alert when a peer interface is configured UP but operationally DOWN”

```
count by (device, name) (
  (interface_admin_state{intf_role="peer"} == 1)
  and on (device, name)
  (interface_oper_state{intf_role="peer"} == 2)
) > 0
```



- **Intent**: `admin_state == 1` (we expect it UP)
- **Reality**: `oper_state == 2` (it’s actually DOWN)
- `and on(device,name)` ensures we compare the **same interface**

Actioning on data: Alerts (cont.)

With decision at hand - we create a rule:

- Evaluate **`for: 2m`** to avoid flappy noise
- Add **`annotations`** so humans know what to do.
- Add **`labels`** to provide more context or categorization for later processing.

```yaml
groups:
- name: interface_intent_mismatch
  rules:
  - alert: InterfaceAdminUpOperDown
    expr: |
      count by (device, name) (
        (interface_admin_state{intf_role="peer"} == 1)
        and on (device, name)
        (interface_oper_state{intf_role="peer"} == 2)
      ) > 0
    for: 2m
    labels:
      severity: warning
      category: network
    annotations:
      summary: "Interface intent mismatch detected"
      description: |
        Interface {{ $labels.name }} on device {{ $labels.device }}
        is configured as UP (admin state) but is currently DOWN (oper state).

        This usually indicates a cabling, peer, or physical-layer issue.
```

Exercise 3.3: Actioning on Data - Alerts

Lab 3.3: Alerts

1. Activate the alert rule
2. Update Prometheus rules configuration:
    a. HINT: You can use Prometheus reload API endpoint or by running the command: **`netobs lab update prometheus`**
3. Check alert being triggered in Prometheus http://<lab-address>:9090/#/alerts
4. After alert is firing check Alertmanager http://<lab-address>:9093/

Actioning on data: Create Visualizations

- Use **queries** to tell the **operational story**
- Ask - which story are we telling?

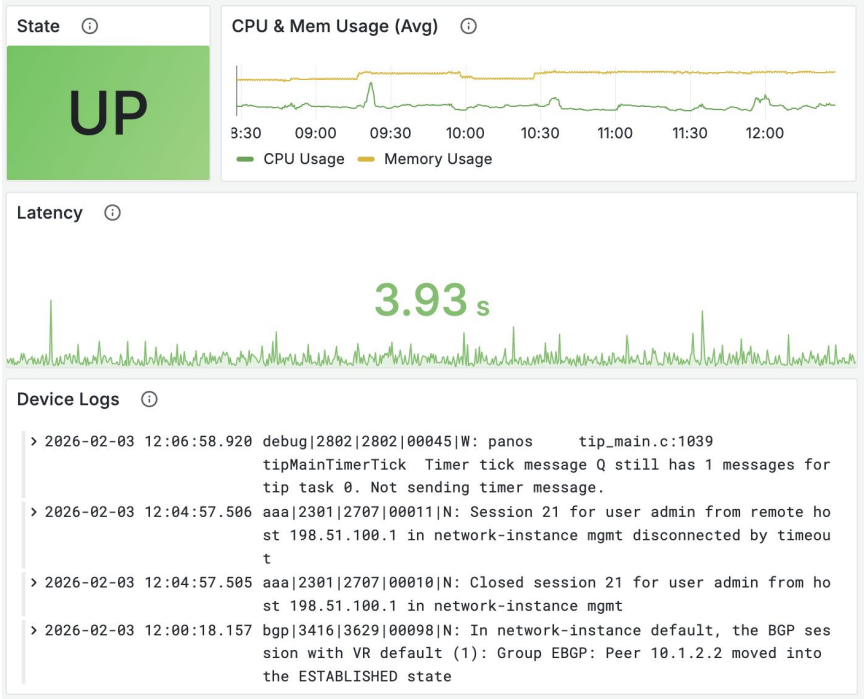
For one device, we want to answer:

- How healthy is it right now? (summary)
- When did it change? (timeline)
- What happened around it? (logs/context)

NOTE: Grafana - `http://<lab-address>:3000`

User: **`netobs`**

Password: **`netobs123`**



Actioning on Data: A Story of Interfaces

Let’s create a dashboard for telling the story of the interfaces in a device operationally.

- Dashboard Settings > Title
- Variables > New Variable
    - Variable Type: Query
    - Name: `device`
    - Label: `Device`
    - Query Options:
        - Query Type: `Lavel Values`
        - Label*: `device`
        - Metric: `device_uptime`

Actioning on Data: Interfaces Admin Panel

Visualization:

- Type: State Timeline
- Title: Interfaces Admin State
- Value Mapping:
  - 1 -> Enabled -> Blue
  - 2 -> Disabled -> Purple
- Standard Options > Color Scheme: Single Color

Query:

- Datasource: prometheus
- Legend: {{ name }}

```
interface_admin_state{
      intf_role="peer",
      name=~"ethernet-1/1([0-2])?",
      device="$device"
}
```

Exercise 3.4: Actioning on Data - Dashboards

Lab 3.4: More Interfaces Panels...

Create an **Interface Logs** Panel

- Type: Logs
- Datasource: Loki
- Title: Interface Logs -$device

```
{device="$device"} != "license"

|= "interface"

|~ "down|up|admin-state"

| line_format "{{.device}} {{.facility_label}} | {{__line__}}"
```

Lab 3.4: More Interfaces Panels...

Create an **Interface Oper State** Panel

- Type: State Timeline
- Datasource: prometheus
- Title: Interface Operating State -$device
- Value Mapping:
  - 1 -> UP -> Green
  - 2 -> DOWN -> Red
- Legend: {{name}}

```
interface_oper_state{
    intf_role="peer",
    name=~"ethernet-1/1([0-2])?",
    device="$device"
}
```

Lab 3.4: More Interfaces Panels...

Create an **Interface Traffic** Panel

- Type: Timeseries
- Datasource: prometheus
- Title: Interfaces Traffic
- Visibility:
  - Mode > Table
  - Placement > Right
  - Values > Last*
- Standard Options > Units: bits/sec (SI)
- Legend: In|Out: {{name}}

```
# Series A: IN Traffic

rate(

interface_out_octets{name=~"ethernet-1/1([0-2])?",
device="$device",
collection_type="gnmi"}[$__rate_interval]

) * 8

# Series B: OUT Traffic

rate(

interface_in_octets{name=~"ethernet-1/1([0-2])?",
device="$device",
collection_type="gnmi"}[$__rate_interval]

) * 8 * -1
```

4

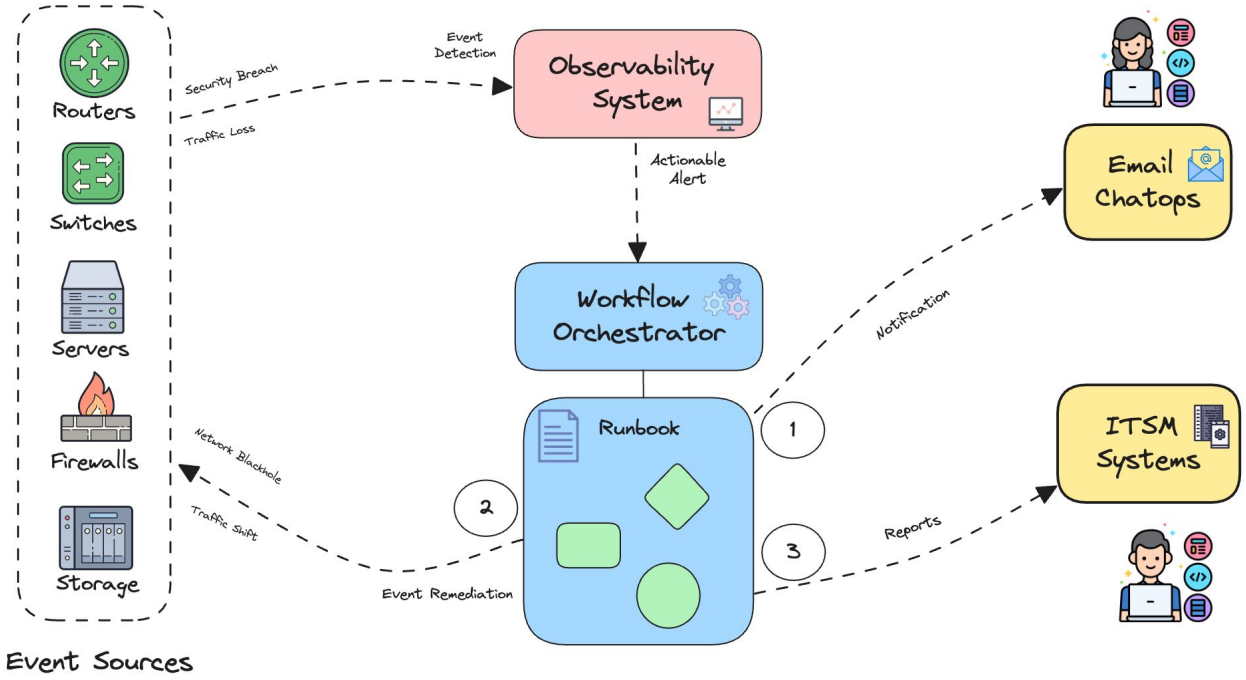
Day-2 Automation: From Observability to Intelligent Workflows

Day-2 Automation starts with *evidence*, not actions

- Automation is a **decision loop**, not “runbook as code”.
- The workflow must be **safe-by-default** (gates, context, audit trail).

The standard view for day-2:

- Monitoring pages you.
- Observability gives you enough **information to act**.
- **Automation is the “Day-2” layer** sitting on top of observability data.

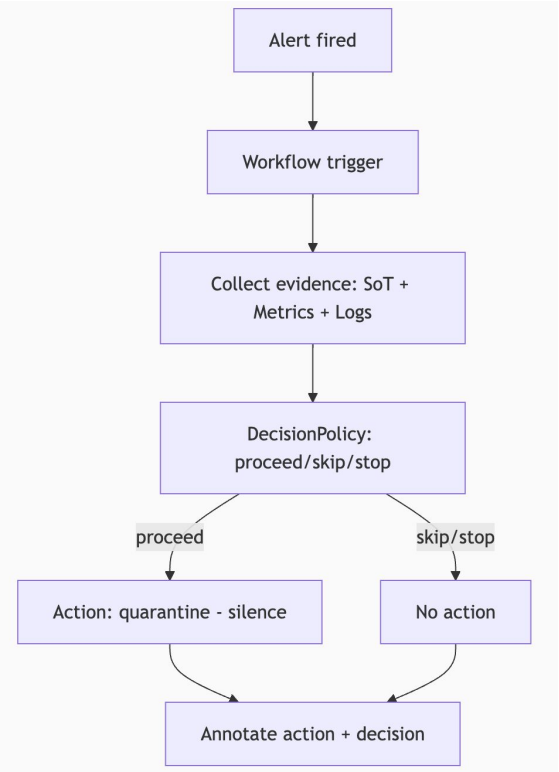


Observability -> Automation Loop

- **Alerts** are **triggers**
- **Evidence** is collected
- **Policy decides**
- **Action + annotation** closes the loop.

Workshop Flow:

1. **Explore Workshop SDK** and its features
2. **Play** around with the **methods and data**
3. Move to Prefect **workflows development**
4. Trigger and **validate executions**



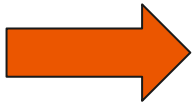
Auto: Prep - Prefect Server

```
# Navigate to webinar folder

cd ~/network-observability-lab/chapters/webinar

# Setup prefect to talk to the docker prefect server

prefect config set PREFECT_API_URL=http://localhost:4200/api

# Check the variable is set

prefect config view
```

Auto: IPython = your observability REPL

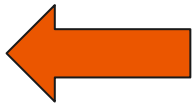
- IPython used to explore SDK and data
- Imports pasted for easier load and prettier outputs

```
# Explore Python SDK with ipython

ipython
```



```
from rich.pretty import install

install()


from netobs_workshop_sdk import (

    WorkshopSDK,

    EvidenceBundle,

    DecisionPolicy,

    Decision,

)

sdk = WorkshopSDK()
```

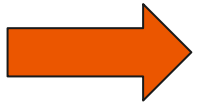
```
sdk.prom  # Prometheus client

sdk.loki  # Loki client

sdk.nb    # Nautobot client

sdk.am    # Alertmanager client
```



Auto: Start with intent

"What should be true" **must gate** "what we do" —> **SoT + maintenance + peer intent gate**

```
# Get a Device object from SoT
dev = sdk.nb.get_device("srl1")
dev["name"]
```



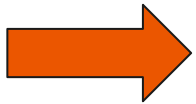
```
# Create SoT gate
gate = sdk.nb.build_bgp_intent_gate(
    device="srl1",
    peer_address="10.1.2.2",
    afi_safi="ipv4-unicast",
)
gate
```

```
# Get a Device object from SoT
sdk.nb.is_device_in_maintenance(dev)
```

Auto: Reality check - what does the metrics say?

With PromQL you can see and the **metrics snapshot you can reason on**

```
# See PromQL queries
qs = sdk.bgp_queries(
    device="srl1",
    peer_address="10.1.2.2",
    afi_safi="ipv4-unicast",
    instance_name="default",
)
qs
```



```
# Run queries
# Check raw results
for k, q in qs.items():
    res = sdk.prom.instant(q)
    print(k, "=>", res[:1])
```

```
# Get a structured data snapshot
sdk.bgp_metrics_snapshot(
    device="srl1",
    peer_address="10.1.2.2",
    afi_safi="ipv4-unicast",
    instance_name="default",
)
```

Auto: Reality check - what does the logs add?

Metrics tell you **what**; logs often explain **why**.

```
# If needed generate some logs

cd ~/network-observability-lab

# Run the flap interface command

netobs utils device-interface-flap \

  --device srl1 \

  --interface ethernet-1/1 \

  --count 2 \

  --delay 2 \

  --platform nokia_srl
```

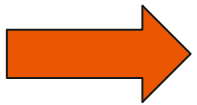
```
# Check the LogQL used

sdk.bgp_logql(device="srl1", peer_address="10.1.2.2")
```

```
# Check some logs

logs = sdk.bgp_logs(

    device="srl1", peer_address="10.1.2.2", minutes=30, limit=50

)

len(logs), logs[:2]
```

Auto: Correlation -> build an EvidenceBundle

Correlation is where observability becomes automation-grade.

```
# Collect BGP Evidence

ev = sdk.collect_bgp_evidence(

    device="srl1",

    peer_address="10.1.2.2",

    afi_safi="ipv4-unicast",

    instance_name="default",

    log_minutes=30,

    log_limit=50,

)

ev.summary()
```



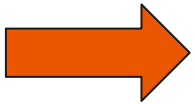
```
# Check the raw data collected

ev.sot

ev.metrics

ev.logs[:3]
```

Auto: DecisionPolicy -> “encoded decisions”

Policies exist to make “why” explicit and auditable.

```
# SoT only gate

policy = DecisionPolicy()

policy.evaluate(ev.sot)
```

```
# Intent vs Reality

policy = DecisionPolicy()

policy.evaluate(ev.sot, ev.metrics)
```



```
# Try with a broken peer

ev_broken = sdk.collect_bgp_evidence(

    device="srl2",

    peer_address="10.1.11.1",

    afi_safi="ipv4-unicast",

    instance_name="default",

    log_minutes=30,

    log_limit=50,

)

DecisionPolicy().evaluate(ev_broken.sot, ev_broken.metrics)
```

Action: quarantine via Alertmanager silence

Actions are the last step, not the first → and only after a *proceed*

```
# Quarantine action - silence alert
silence_id = sdk.quarantine_bgp(
    device=ev.device, peer_address=ev.peer_address, minutes=20
)
silence_id
```

Check Silences in http://<lab-address>:9093/#/silences

Close the loop: annotate decisions + actions

If you can’t explain it later, it’s unsafe automation.

```
# General annotation
sdk.annotate(
    labels={
        "source": "prefect",
        "workflow": "demo_quarantine_bgp",
        "device": ev.device,
        "peer_address": ev.peer_address,
    },
    message=f"QUARANTINE applied (silence_id={silence_id})",
)
```

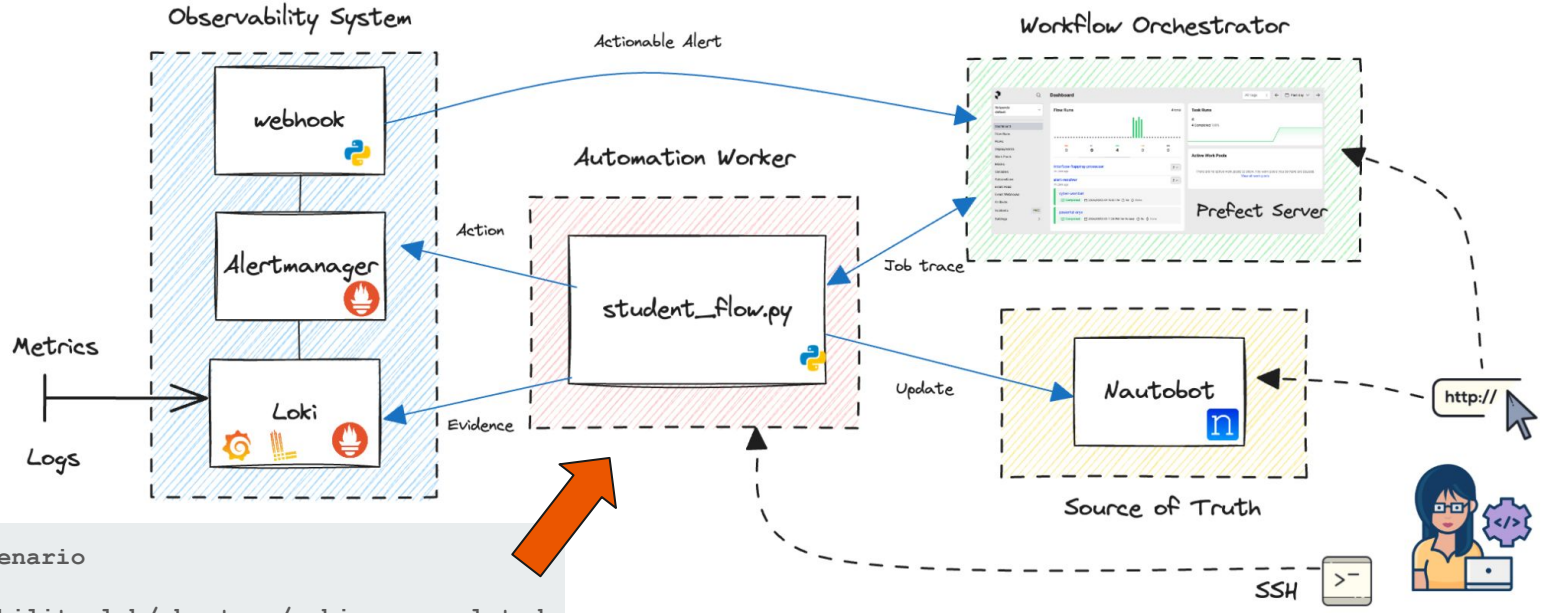
```
# Decision-specific annotation
sdk.annotate_decision(
    workflow="demo_quarantine_bgp",
    device=ev.device,
    peer_address=ev.peer_address,
    decision="skip",
    message="SKIP quarantine: peer not intended in SoT",
)
```

LLM-ready RCA payload (structured + scoped)

RCA becomes feasible when evidence is already correlated + normalized.

```
# Payload for RCA

payload = ev.to_rca_payload()

payload.keys()
```

Run it on Prefect




```
# Navigate to the scenario

cd ~/network-observability-lab/chapters/webinar-completed

# Prepare the runner

python student_flow.py
```

Trigger 1: Actionable (expect quarantine)

```
prefect deployment run 'alert-receiver/alert-receiver' \

  --param alertname='BgpSessionNotUp' \

  --param status='firing' \

  --param alert_group='{"status":"firing","groupLabels":{"alertname":"BgpSessionNotUp"},

"alerts":[{"labels":{"alertname":"BgpSessionNotUp","device":"srl2","peer_address":"10.1.11.1",

"afi_safi_name":"ipv4-unicast","name":"default"},

"annotations":{"summary":"expect QUARANTINE (mismatch demo)"}}]}'
```

Trigger 2: Skip (healthy peer)

```
prefect deployment run 'alert-receiver/alert-receiver' \

  --param alertname='BgpSessionNotUp' \

  --param status='firing' \

  --param alert_group='{"status":"firing","groupLabels":{"alertname":"BgpSessionNotUp"},

"alerts":[{"labels":{"alertname":"BgpSessionNotUp","device":"srl2","peer_address":"10.1.2.1",

"afi_safi_name":"ipv4-unicast","name":"default"},

"annotations":{"summary":"expect SKIP (peer is healthy)"}}]}'
```

Trigger 3: Resolved Path (alert is resolved)

```
prefect deployment run 'alert-receiver/alert-receiver' \

  --param alertname='BgpSessionNotUp' \

  --param status='resolved' \

  --param alert_group='{"status":"resolved","groupLabels":{"alertname":"BgpSessionNotUp"},

"alerts":[{"labels":{"alertname":"BgpSessionNotUp","device":"srl2","peer_address":"10.1.11.1",

"afi_safi_name":"ipv4-unicast","name":"default"},

"annotations":{"summary":"resolved test"}}]}'
```

Exercise 4: Actioning on Data - Event Driven Automation

Lab 4: Event Driven Automation Workflows

1. Work with the lab environment to create an alert that triggers an automation workflow that:
    a. is actionable (quarantine)
    b. is skippable
    c. is resolved
2. What happens if a device is in `maintenance` mode from the SoT?
    a. Change a device into maintenance mode and verify the logic follows the pattern expected

Thank you!