HPE

AutoCon 6 Workshop

The Self Driving Network: From Drift detection to Autonomous remediation with Streaming Telemetry



Josemi Izquierdo
Automation Technical Marketing Engineer



David Roy
Routing Technical Marketing Engineer



Mark Harman
Security Solutions Architect



Jessica Garrison
Automation Architect & Junos Specialist

HPE JUNIPER NETWORKS

Josemi



David



Mark

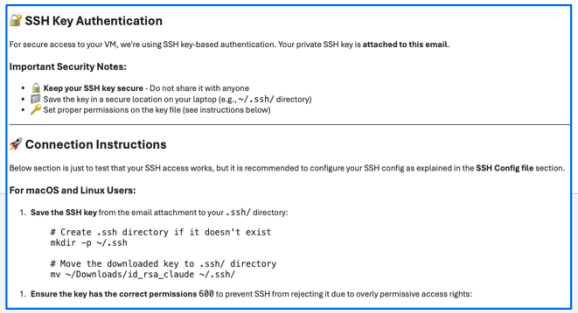


Jessica



The Self Driving Network: From Drift detection to Autonomous remediation with Streaming Telemetry

Access Credentials



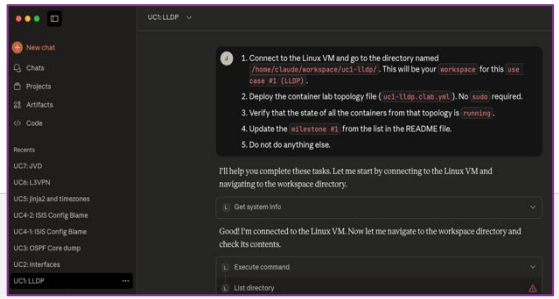
- ❑ Add the correct SSH key and config from the email Josemi (Jose Miguel Izquierdo) sent titled “AutoCon6 WS-D4 preparation” and

Request IP Address



- ❑ We whitelisted the source IP provided by NAF
- ❑ Find Josemi, David, Mark, or Jessica to ask
- ❑ Check Connectivity

Other Applications



- ❑ VS Code recommended as IDE, specific plugins recommended, too

Follow the steps above to get ready for the workshop

Meet Your Workshop Proctors



```
proctor:
  name: josemi izquierdo
  lead: workshop,lab
  autocon_event_count: 6
  juniper_years: 14
  subjects:
    - system integration
    - llm/mcp
    - automating on junos
    - routing director
```



```
proctor:
  name: david roy
  lead: openjts,routing
  autocon_event_count: 2
  juniper_years: 4
  subjects:
    - routing
    - telemetry
    - hardware
```



```
proctor:
  name: mark harman
  lead: security
  autocon_event_count: 0
  juniper_years: 4
  subjects:
    - security
    - automation
```



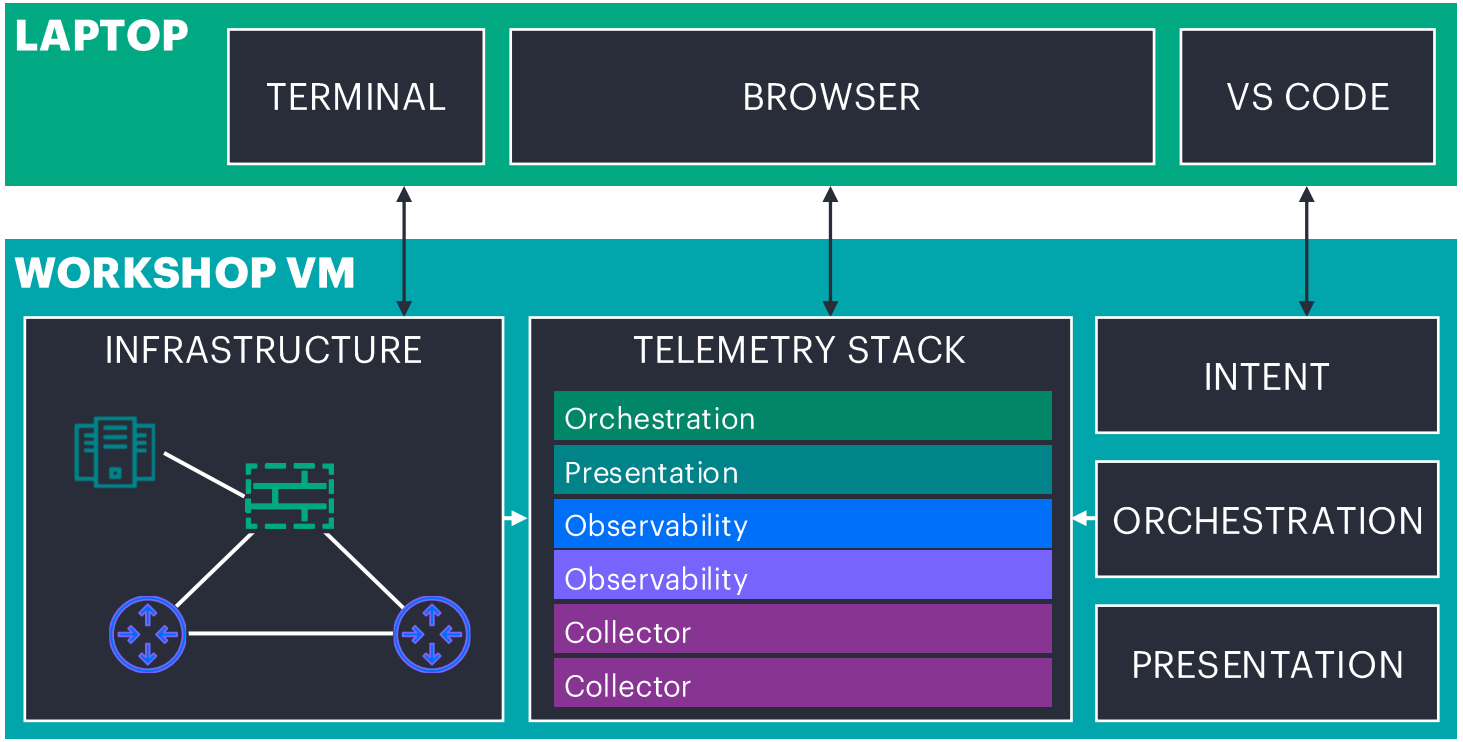
```
proctor:
  name: jessica garrison
  lead: slides,proctor
  autocon_event_count: 5
  juniper_years: 21
  subjects:
    - junos
    - automation
    - telemetry
```



Workshop Goals & Use Cases

Goals

- Understand how network intent and streaming telemetry help detect drift.
- Follow the closed loop from detection and planning to remediation.
- Compare human-led, human-approved, and autonomous responses.
- Explore governance and how NAF blocks can be interchanged.



Labs 0-5

Logging Drift: Detect and remove leftover debug logging, while respecting an active maintenance window.

BGP Peer Drift: Correct a wrong peer AS that has brought down the BGP session between pe1 and pe2.

Prefix Limit: Detect a route leak from pe2 and protect pe1 with an import clamp until the leak is fixed.

Security Policy Drift: Explore detecting and correcting firewall rule-order drift; this use case is deferred and not runnable in the current lab.

Swap a Block: Replay BGP peer drift while swapping Intent, Executor, and Collector implementations.

Workshop Outline

Five hands-on labs that take you from drift detection to autonomous remediation

Welcome & Kickoff — START

- Meet your proctors
- Workshop goals & use cases
- Quick demo video

Bring Up the Lab — LAB 0

- Set up your workshop VM
- Bring up the telemetry pipeline

Telemetry Foundations — LEARN

- Telemetry basics
- L1 / L2 / L3: who authorizes the fix?
- Intro to the NAF Framework
- Our stack

Logging Drift — LAB 1

- How data flows from device to Grafana
- What happens to the data along the way

BGP Peer Drift — LAB 2

- cRPD and device configurations
- Deeper dive: telemetry sensors

Prefix Limit — LAB 3

- Add a router to JTSO
- Deeper dive: how OpenJTS works together

Security Policy Drift — LAB 4

- cSRX / vSRX and sensor mixing
- Concept walkthrough; not runnable in this lab

Swap a Block — LAB 5

- Deeper dive into the NAF Framework
- Replay BGP peer drift with swapped blocks

Wrap-Up — FINISH

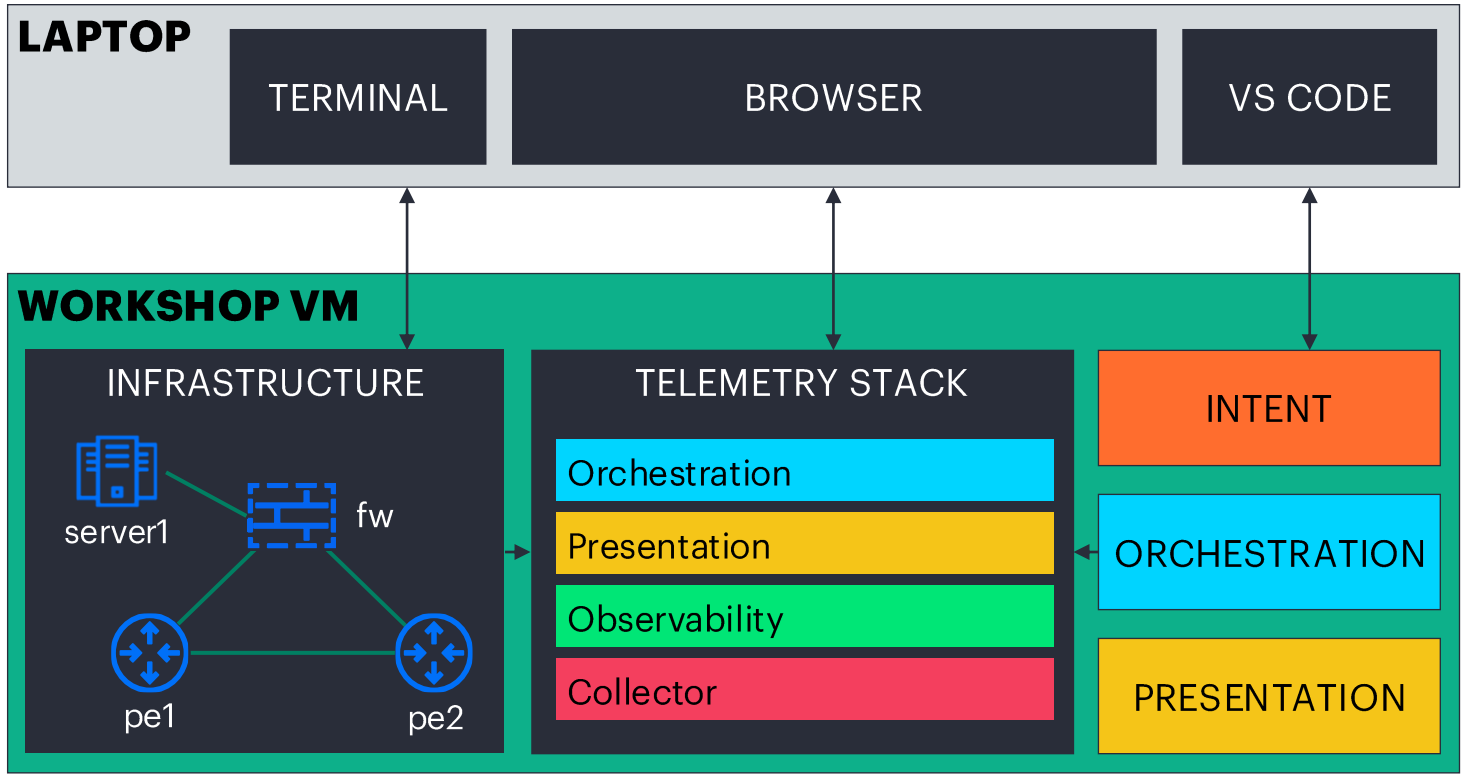
- Goals completed
- Resources to keep learning



Lab 0 – Bring up the pipeline

GOALS

Confirm the lab is up (alias *check_vm*), add the routers to JTSO, assign the NAF profile, and observe the first green BGP dashboard



STEPS

1. **Check the lab:** run *check_vm* alias (images, licences, topology, BGP, OpenJTS, ntfy). If down, fix via script, manually or ask a proctor
2. **Add routers in JTSO manually:** login lab/lab123; crpd-pe1 (172.20.20.11), crpd-pe2 (.12), csrx (.21), server1 (.31)
3. **Assign the naf profile manually:** Profiles → Associations, for all three routers
4. **Verify telemetry manually:** telegraf_crpd / telegraf_csrx up; Grafana NAF PROFILE with Device = All

Lab 0: Workshop VM in 4 Steps

From ~/naf-ac6-workshop. Step 1 is checked by one script

Shortcut: `check_vm` prints **OK** / **ISSUE** / **FAIL** per check and asks before each fix. `(--no-fix)` only reports.

Step 1
Check the lab

```
check_vm
```

– Checked with the script: images and licences; the containerlab topology and BGP; OpenJTS and ntfy.

– Everything should already be up.

– If not, fix it with the script, manually, or ask a proctor.

Step 2
Add devices in JTSO

http://localhost:18090

– Admin → Settings: lab / lab123, TLS disabled

– Routers: crpd-pe1 .11, crpd-pe2 .12, csrx .21, server1 .31

– Family, Model, Version fill in

Step 3
Assign the naf profile

Profiles → Associations

– All 3 routers → naf (optional bgp on PEs)

– telegraf_crpd and telegraf_csrx start

– Grafana: localhost:18080/d/naf-profile

Step 4
Python environment

```
naf-setup
```

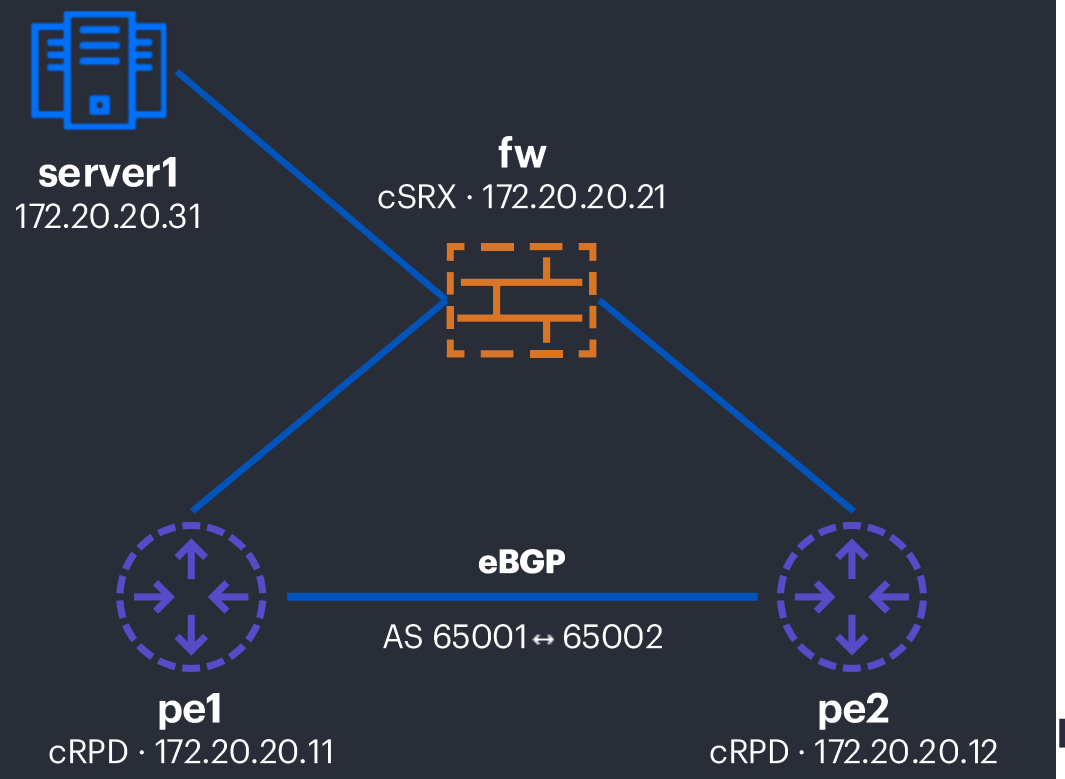
– Creates .venv, installs PyEZ, Ansible, FastAPI

– Runs pytest, resets intent to v1

– New terminals: `naf-env`

Lab 0 - What we setting up in the lab

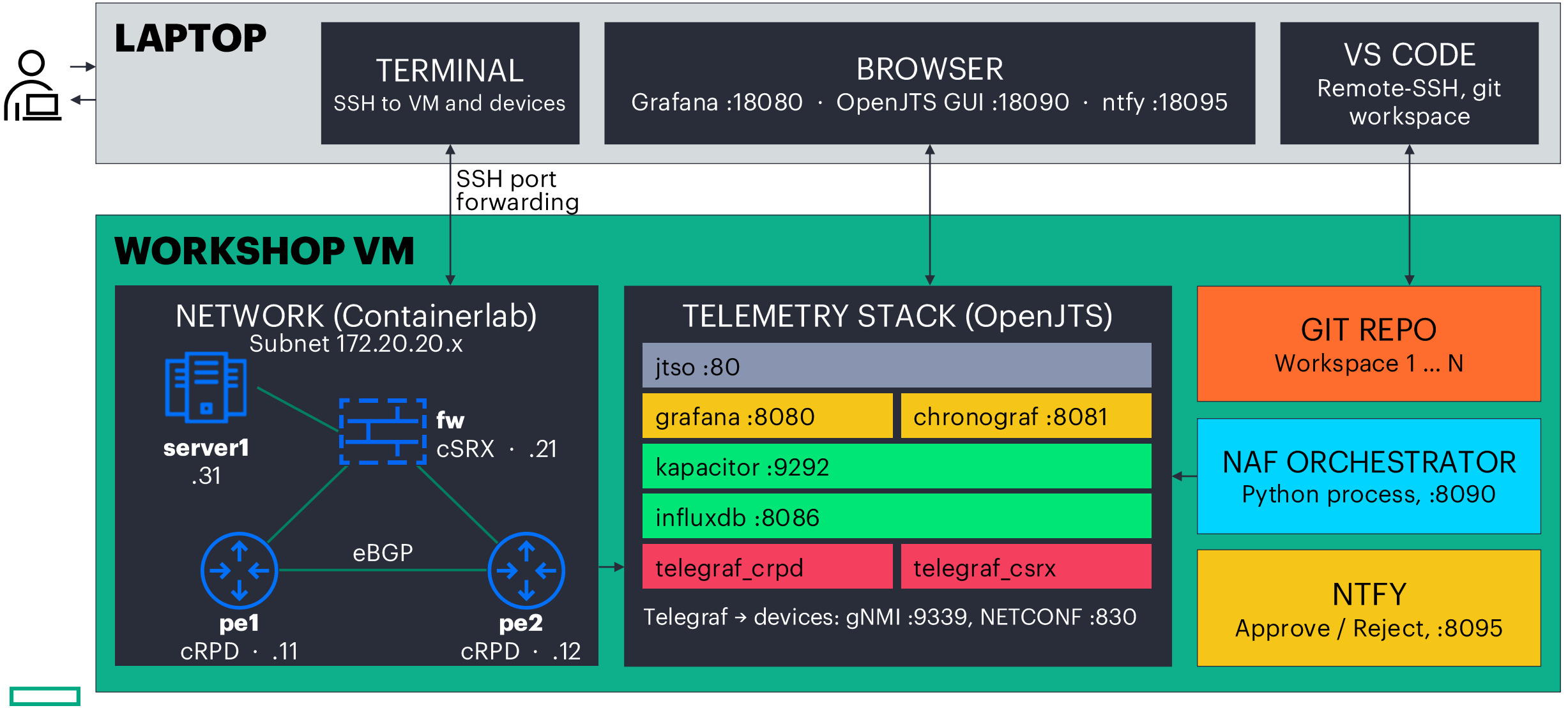
NETWORK (ContainerLab)



TELEMETRY STACK (OpenJTS)

JTSO	**JTSO :80** Maps routers → naf profile → Telegraf config
Presentation	**Grafana :8080 · ntfy :8095** NAF PROFILE dashboard, Approve / Reject
Orchestration	**NAF orchestrator :8090** L1 / L2 / L3 autonomy · PyEZ executor
Observability	**InfluxDB :8086 → Kapacitor :9292** Drift alerts on config and BGP changes
Collector	**Telegraf · gNMI / NETCONF** telegraf_crpd (pe1, pe2) · telegraf_csrx

Lab 0 – Workshop VM with OpenJTS



Lab 0 – Reset the lab when starting a new lab (GUIDE-SETUP.md)

Every L1 / L2 / L3 guide assumes this setup (a couple of minutes).
Open 3 terminals: two on the workshop VM, one per device you log in to.

Step 1 — Terminal 1: Environment

First time on the VM:

- `naf-setup`

Every new terminal:

- `naf-env`

Prompt starts with *(.venv)* in *~/naf-ac6-workshop*

Step 2 — Terminal 2: Orchestrator

Start it and leave it running:

- `scripts/orchestrator.start.sh`

Check status:

- `scripts/orchestrator.status.sh`

Ctrl-C stops it

Step 3 — Terminal 1: Seed, load, reset

- `python -m observability.seed_intent`
- `scripts/load_ticks.sh`
- `reset-lab`
- `ohealth`

Intent, rules, baseline, orchestrator

Step 4 — Device terminals

One terminal per device the use case needs:

- `ssh pe1`
- `ssh pe2`
- `ssh csrx, then cli`
- `ssh server1`

Step 5 — Check before each run

- **.venv** active
- Orchestrator **up**
- Kapacitor tasks **loaded**
- BGP **up** on pe1
- **pend** is empty
- **olog** last alert recovered

Three shifts behind the self-driving network

What we cover in this section before the hands-on labs

01 How we collect

FROM

SNMP polling

Pull-based, periodic, bursty replies

TO

Streaming telemetry

Push-based, model-driven, high precision

02 What we watch

FROM

Reactive troubleshooting

Find out after something breaks

TO

Continuous drift detection

Compare live state to intent, all the time

03 Who acts

FROM

Manual fixes

A person investigates and changes the device

TO

L1 · L2 · L3

Observe, Assist, or Autonomous remediation

Up next: SNMP vs. streaming → sensors and dial-in/out → reference architecture → labs

From SNMP Polling to Streaming Telemetry

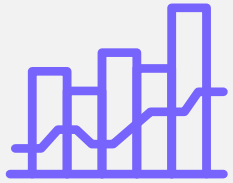
Decades of periodic SNMP polling can't keep up with modern scale, speed, and automation

SNMP



Polling Model

- Pull-based with bursty replies
- Polling interval in minutes
- Low-precision, collector-driven timestamps



Data and Investment

- Unoptimized, incomplete data modeling
- No further IETF or vendor development



Infrastructure

- Centralized design, limited scalability
- Unreliable
- Limited security (SNMPv3 only)

Telemetry

Streaming Telemetry



- Push-based
- Structured (model-driven)
- High precision
- Vendor invested
- Data richness
- Scalability
- Flexibility
- Reliability
- Security

L1 | L2 | L3: Who Authorizes the Fix?

At every level: the system detects drift and plans the fix. Only who authorizes the remediation changes.

L1 Observe

WHO AUTHORIZES THE FIX?

A person, manually

The system detects drift and presents a proposed fix, but does not change the device. A person is notified and fixes it.

L2 Assist

WHO AUTHORIZES THE FIX?

A person approves

The system proposes a fix and waits. A person reviews it and approves or rejects it; only when approved, does the change occur.

L3 Autonomous

WHO AUTHORIZES THE FIX?

No per-event approval

The system detects drift and applies the fix without a person involved directly with the event. These event usually have a lower blast radius, consistent and regularly seen behavior, and are predictable in nature.

L1 recommends. L2 asks. L3 acts.

Primary Telemetry Use Cases

Cloud-Ready & Scalable, Open & Standardized (OpenConfig/IETF standards)



Performance Monitoring & Accounting

- ✓ Monitoring metrics at high resolution
- ✓ Path Optimizations using telemetry data
- ✓ Accounting based on actual usage
- ✓ AI-powered insights: device optimization & health



Intelligent Traffic Analytics

- ✓ Identify and resolve anomalies in the networks
- ✓ Closed loop Automation
- ✓ Troubleshoot Packet loss / latency in the network for AI Clusters
- ✓ Predictive maintenance & self-healing networks



Optimize & Predict Power Gains

- ✓ Start going green
- ✓ Reduce Power consumptions
- ✓ Help with capacity planning
- ✓ Reduce carbon footprint

SNMP vs. Streaming Telemetry





Definition: **Standard or Enterprise (HPE Juniper) MIBs**	Definition: **OpenConfig or Native (HPE Juniper) YANG Data Models**
MIB Name: **IETF IF-MIB**	Sensor Name: **OpenConfig Interfaces** model
Root OID: **1.3.6.1.2.1.2**	Root PATH (aka. Sensor path): **/interfaces**
A MIB counter: **1.3.6.1.2.1.31.1.1.1.6.<ifIndex>** (ifHCInOctets)	A Telemetry counter: **/interfaces/interface[name=*]/state/counters/in-octets**

SNMP, Native Telemetry, and OpenConfig Telemetry

	SNMP	Telemetry	
	SNMP	**Native Telemetry**	**OpenConfig Telemetry**
Collect Mode	Pull	Push	Push
Payload Format	BER	ProtoBuf (GPB)	ProtoBuf (GPB) or JSON
Protocol Interaction Layer	SNMPv1, v2, v3	ProtoBuf (GPB)	gRPC
Encryption/Authentication	Community or key	Binary (GPB)	TLS & Login/Password
Transport Layer	UDP	UDP	TCP
Default Port	161	Configurable (local and remote)	9339 (Configurable)
Data Model	MIB	Vendor Specific	OpenConfig
Data Format	SMI/ASN-1	YANG	YANG
		Dial-Out	Dial-In



Which sensors should I use?

Native

Data Model: **Juniper**

Native telemetry defaults to a push (dial-out) model but also supports dial-in for on-demand requests. Both use YANG and GPB.

Why people use it?
Supports more sensors than the OpenConfig model. Company and efficient, high performance and little overhead.

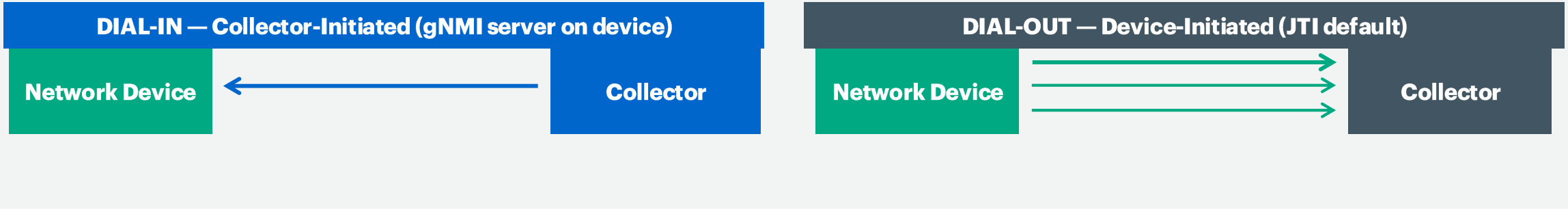
OpenConfig

Data Model: **OpenConfig**

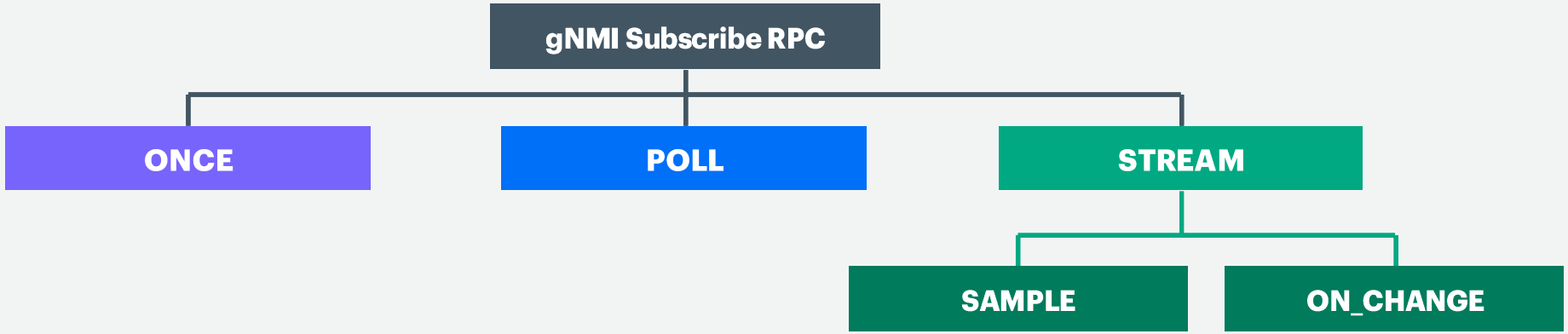
OpenConfig is a request service and therefore, requires authentication. Both use YANG and GPB. Created in Collaboration with HPE Juniper.

Why people use it?
Standards based so there is more open-source tooling behind it. Users like that it is vendor-independent.

Dial-In (OpenConfig) vs. Dial-Out (Native)

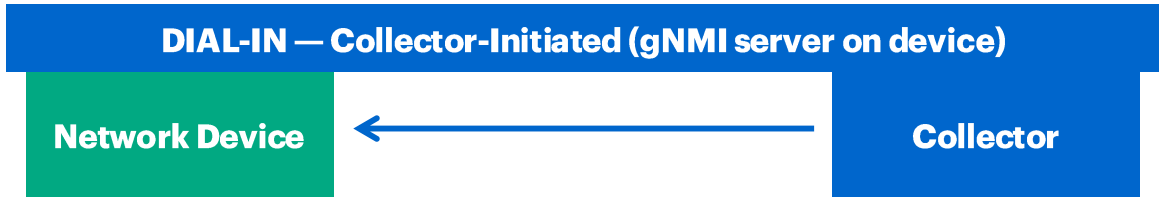


gNMI Subscription Modes | The 3 Request Modes

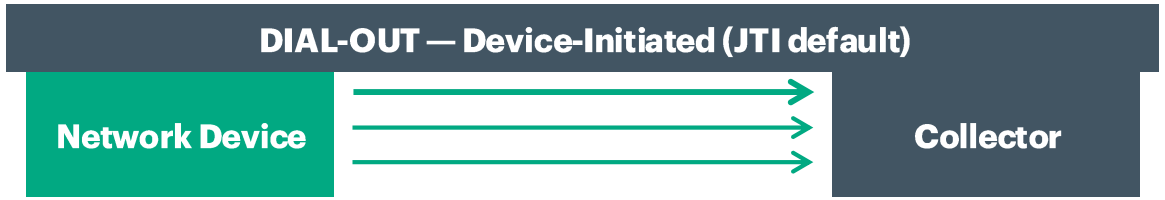


Dial-In vs. Dial-Out

Connection Direction Comparison



Collector subscribes to device gNMI server
Port: 9339 (default) | TLS required in prod



Device streams data → Collector
Transport: gRPC (TCP) or UDP

Dial-In (Collector-Initiated) — On-Demand

The collector connects to the device's gNMI server (TCP port 9339 by default).

Requires: set system services extension-service request-response grpc.

Supports ONCE, POLL, and STREAM (SAMPLE + ON_CHANGE) subscription modes.

Best suited for on-demand Get operations and ON_CHANGE subscriptions.

Both modes use identical YANG data models and sensor paths.

Dial-Out (Device-Initiated) — JTI Default

The Juniper device acts as the gRPC client and initiates the connection to the collector.

Configured via CLI: streaming-server, export-profile, and sensor stanzas.

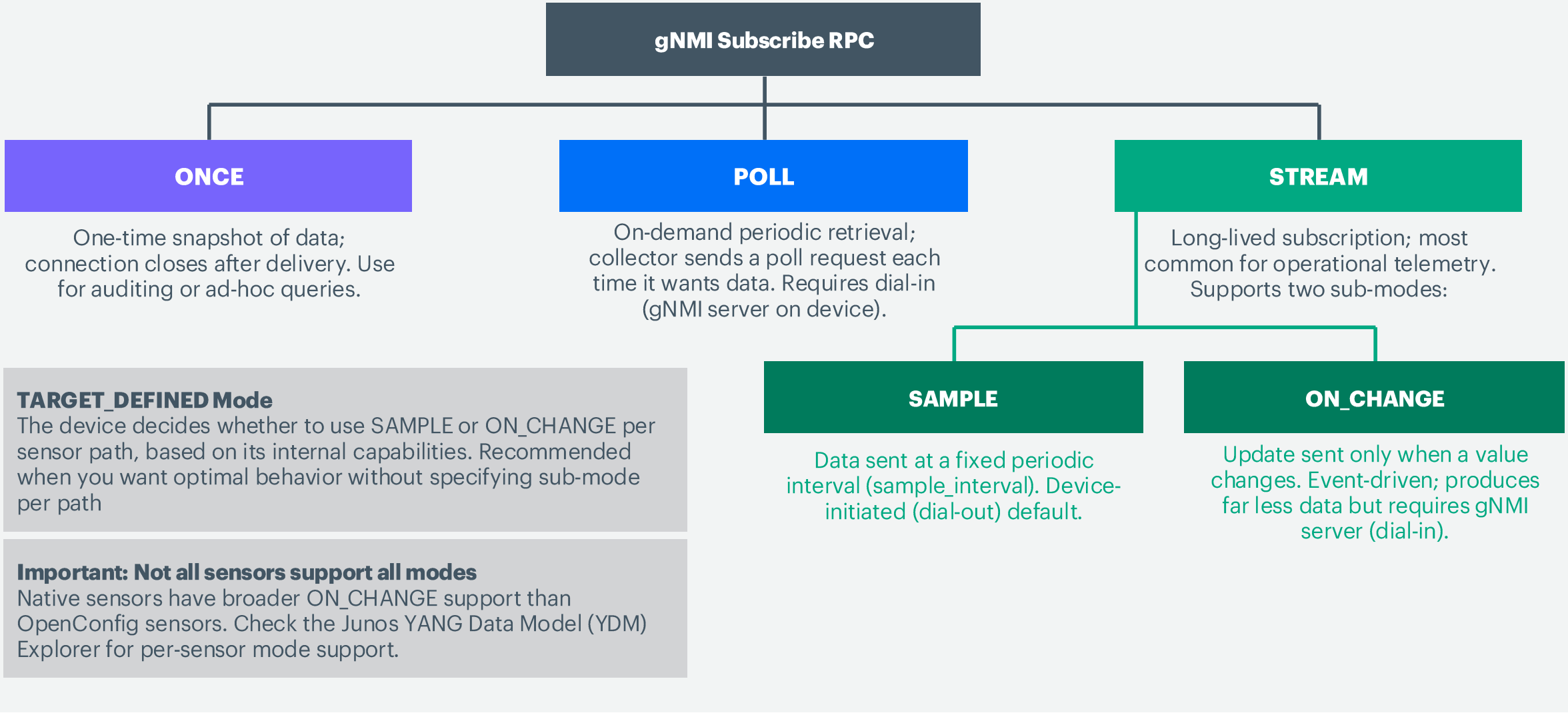
Supports both UDP (native GPB) and TCP/gRPC transports.

Used for persistent STREAM subscriptions.

This is JTI's primary streaming mode — most production deployments use dial-out.

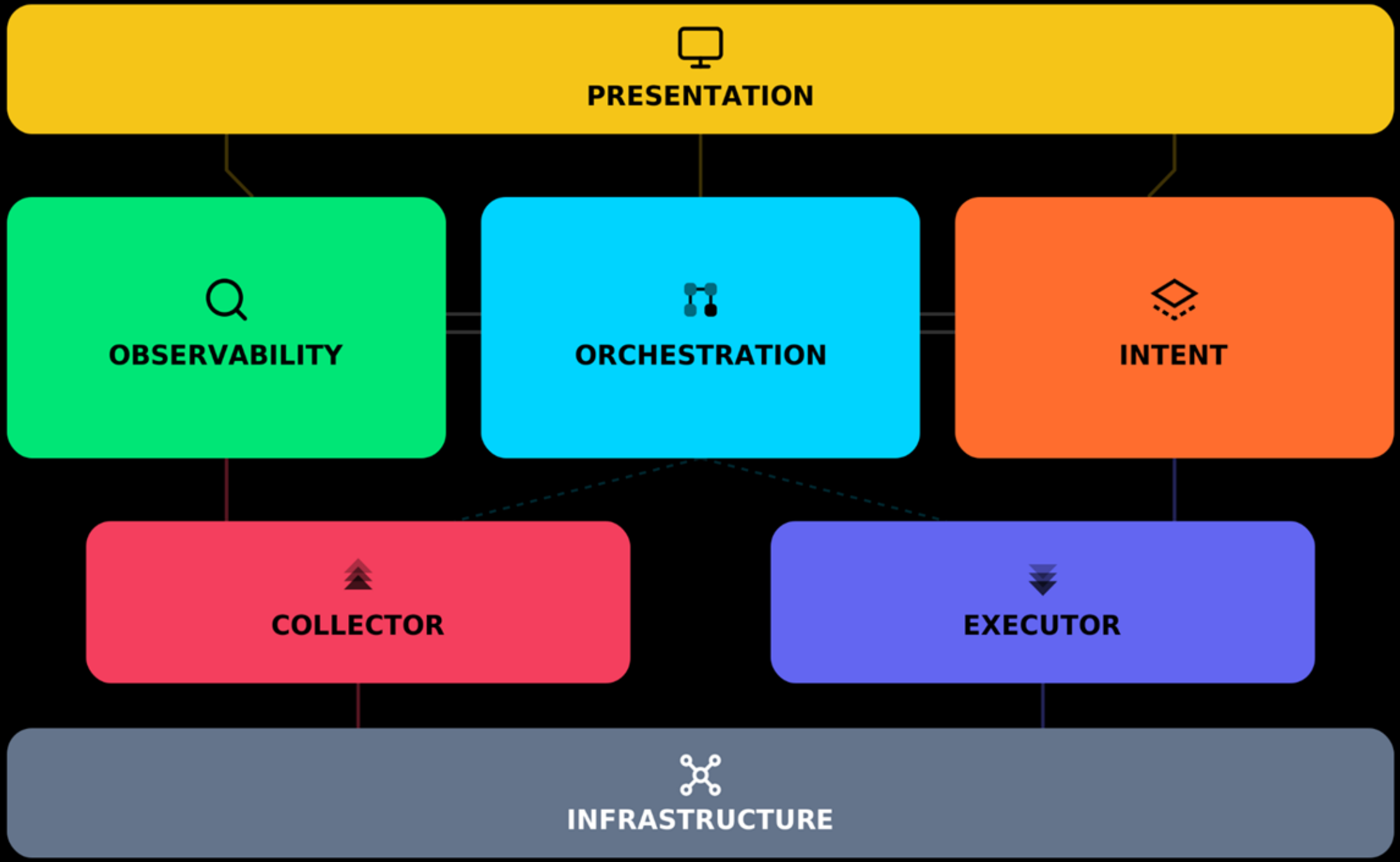
Both modes use identical YANG models and sensor paths

gNMI Subscription Modes | The 3 Request Modes



TARGET_DEFINED Mode
The device decides whether to use SAMPLE or ON_CHANGE per sensor path, based on its internal capabilities. Recommended when you want optimal behavior without specifying sub-mode per path

Important: Not all sensors support all modes
Native sensors have broader ON_CHANGE support than OpenConfig sensors. Check the Junos YANG Data Model (YDM) Explorer for per-sensor mode support.


PRESENTATION
OBSERVABILITY
ORCHESTRATION
INTENT
COLLECTOR
EXECUTOR
INFRASTRUCTURE

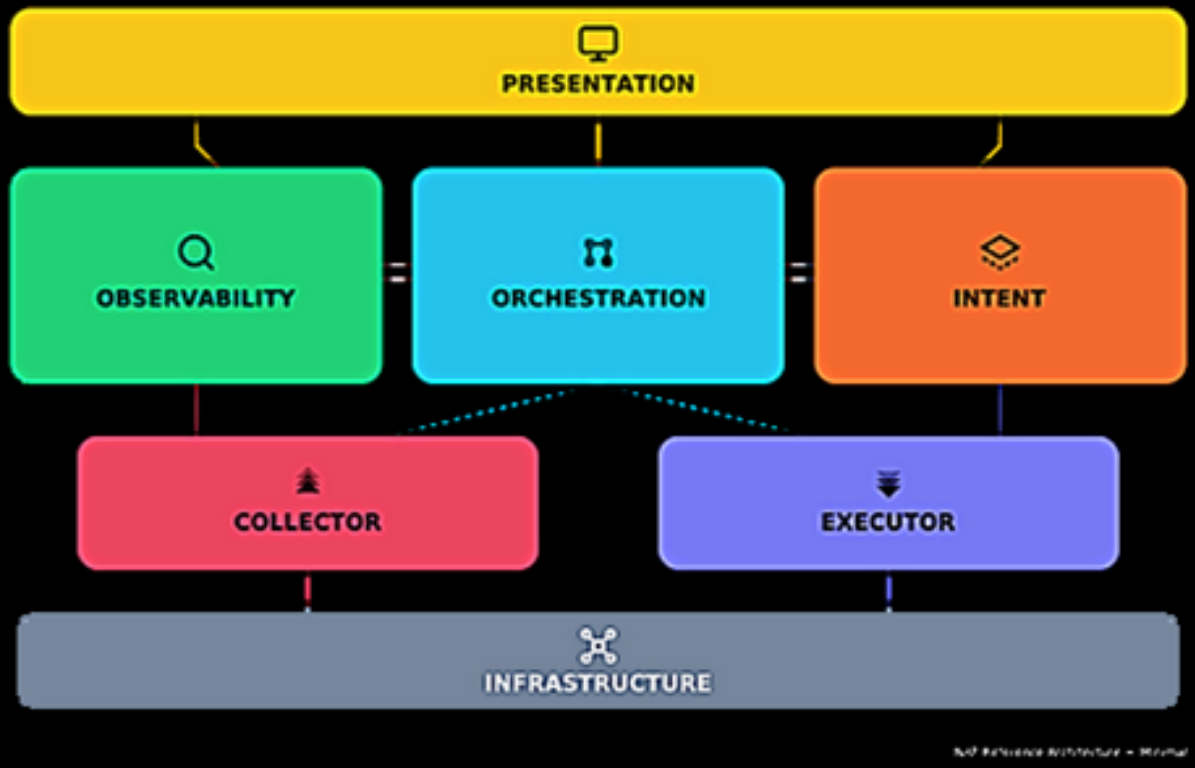
A Functional Reference Architecture

The key idea

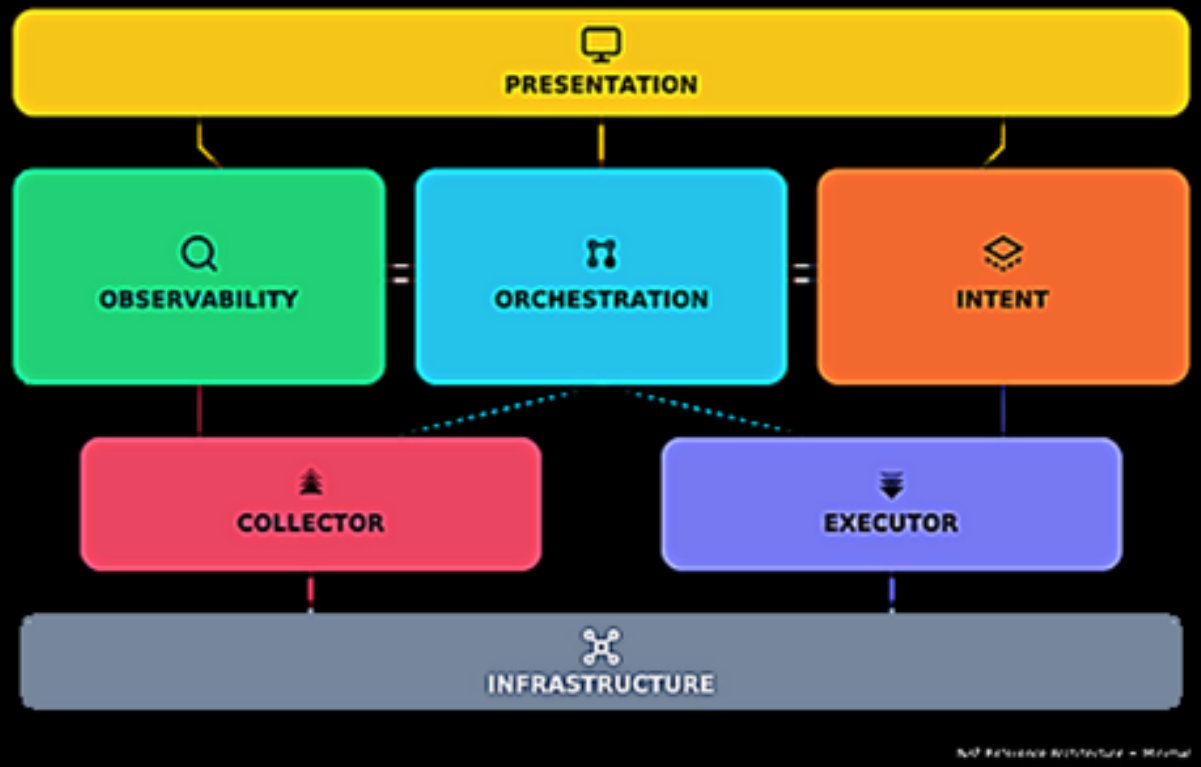
The NAF Framework is a **functional reference architecture**, not a required software stack. It describes **six functions** that can be composed from one or more tools. Network infrastructure is part of the wider architecture, but is separate from the six framework blocks.

Workshop framing

This lab builds a small, working version of the **NAF closed loop**. It shows how desired intent, live telemetry, event processing, orchestration, device changes, and human oversight fit together. It uses selected capabilities; it is not a production-complete implementation of every NAF recommendation.



A Functional Reference Architecture



Intent: Defines the logic to handle and the persistence layer to store the desired state of the network, including both configuration and operational expectations.

Observability: It persists the actual network state, and defines the logic to process it.

Orchestration: Defines how the automation tasks are coordinated and executed in response to events.

Executor: Encompasses the actual tasks applied to the network to drive changes (e.g., updating configuration) as guided by the intended state.

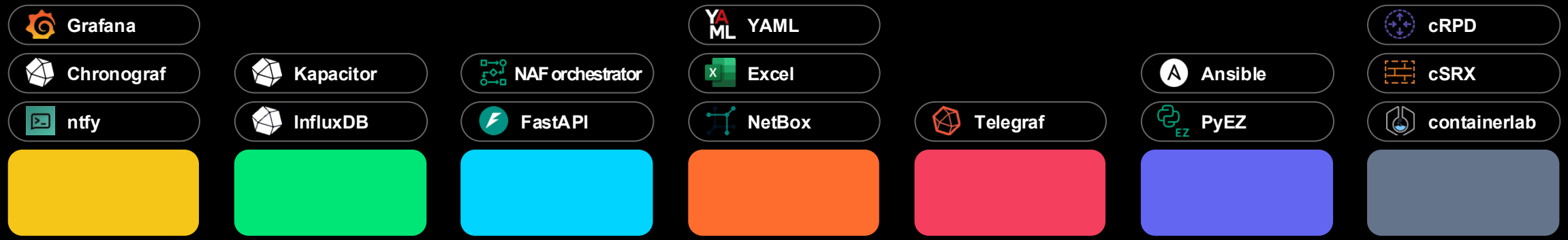
Collector: In contrast to Executor, this component focuses on retrieving (i.e., reading) the actual state of the network.

Presentation: Provides the interfaces through which users interact with the system, including dashboards, graphical user interfaces (e.g., ITSM), and CLI tools.

Which building blocks from the NAF Framework will our labs be using?



Which building blocks from the NAF Framework will our labs be using?



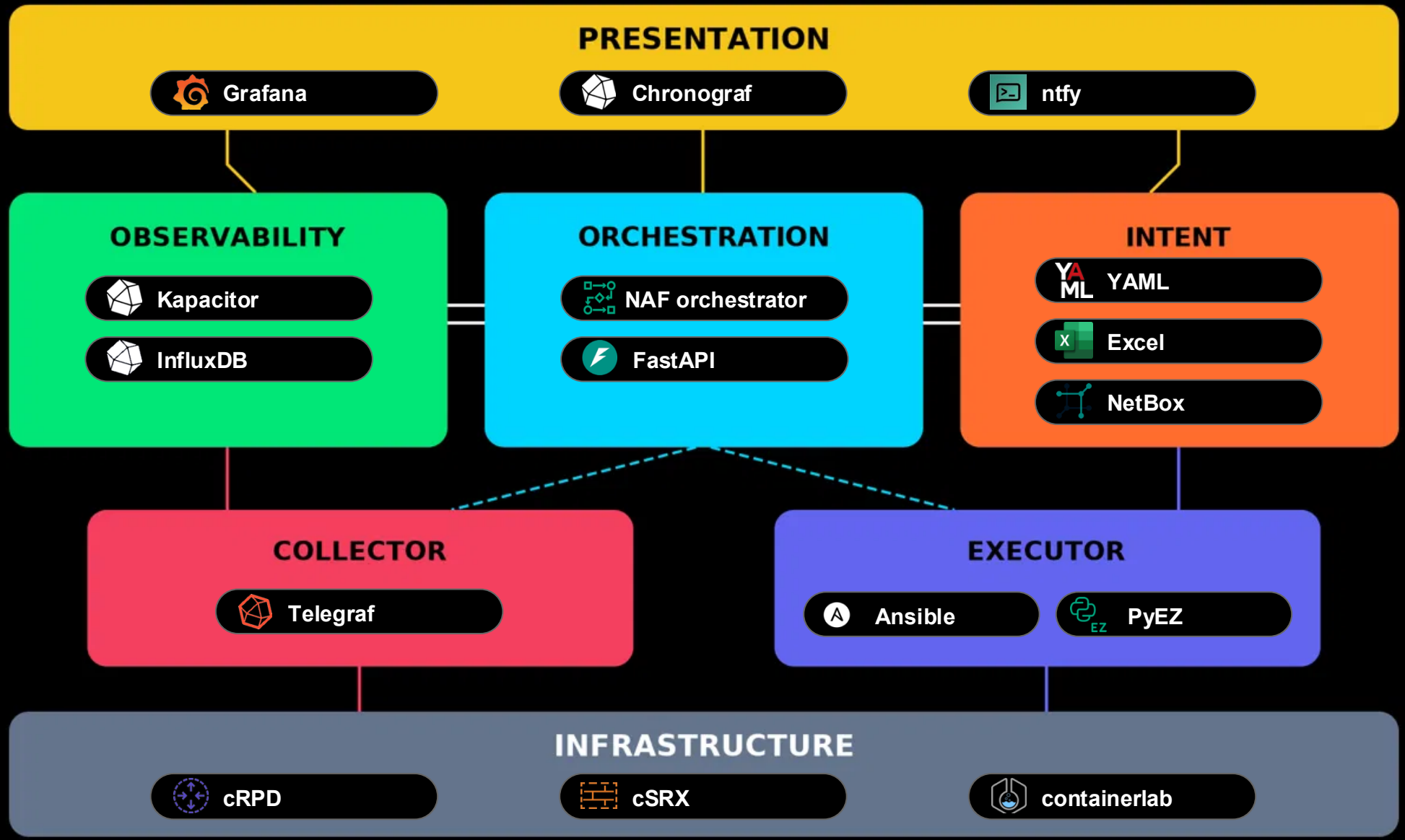
NAF Blocks at a Glance

Block	Purpose	Lab tool
Presentation	Shows status and decisions to people	**Grafana, ntfy, CLI**
Intent	Desired network state	**YAML (or Excel / fake NetBox)**
Orchestration	Decides what workflow to run	**FastAPI orchestrator**
Observability	Stores and analyzes data, raises alerts	**InfluxDB + Kapacitor**
Executor	Makes the network change	**PyEZ (or Ansible)**
Collector	Reads actual state and telemetry	**Telegraf (gNMI, NETCONF)**
Infrastructure	The network being managed	**Containerlab: 2 cRPD + 1 cSRX**

Classify components by the function they perform, not by product name.

Network Automation Framework

REFERENCE ARCHITECTURE



Use Case 1: Logging Drift

Collector	The openJTS **naf** profile **polls commit information** from the device.
Observability	**Kapacitor** reports that a **commit occurred**.
Intent	Holds the **logging settings** and **maintenance windows** the change is checked against.
Orchestrator	Checks the **device diff against intent** and whether a **maintenance window is open**.

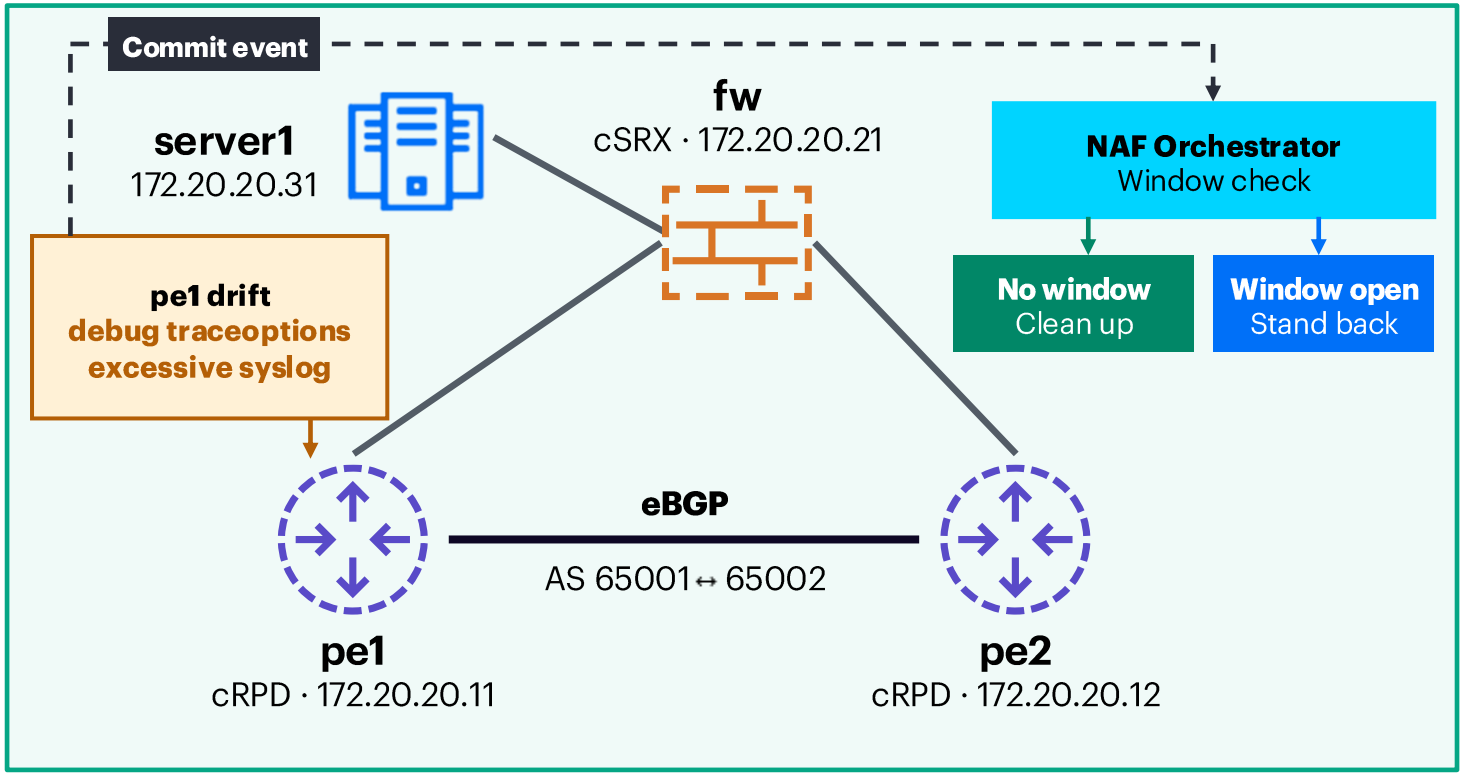
Optional syslog path

A Telegraf **syslog listener** can replace polling, demonstrating a **Collector swap**.

Lab 1 - Logging Drift

GOALS

Remove leftover debug logging when no maintenance window authorizes it, while preserving changes during an active window.

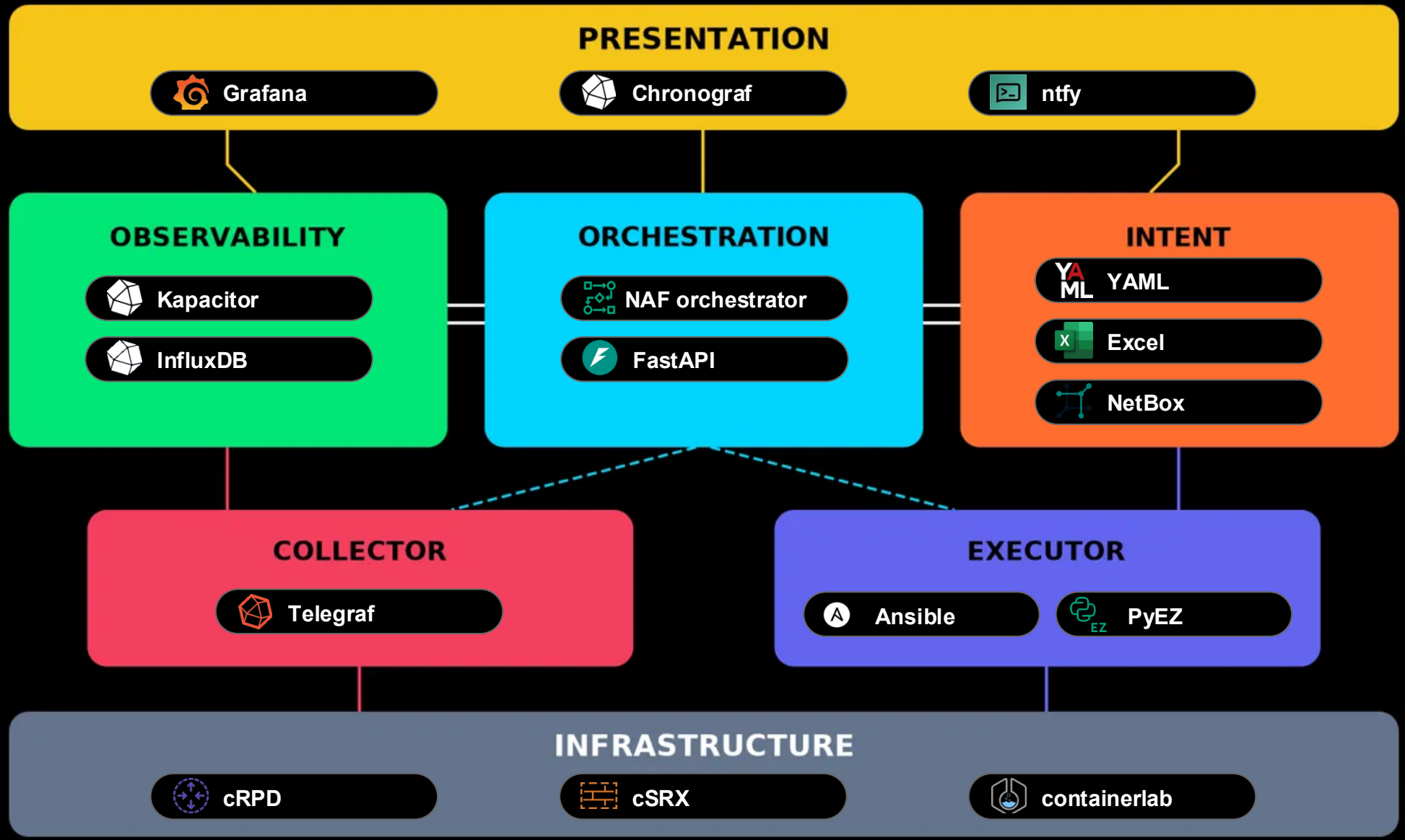


STEPS

1. **Confirm the baseline logging configuration** on pe1.
2. **Introduce debug traceoptions and excessive syslog settings**, then commit the change.
3. **At L1,** inspect the detected drift and remove the debug configuration by hand.
4. **Repeat at L2** and review the proposed cleanup before approving it.
5. **Repeat at L3 without a maintenance window** and verify that the orchestrator cleans up the drift.
6. **Open a valid maintenance window**, trigger another commit, and verify that the orchestrator leaves the configuration unchanged.
7. **Expire the window**, trigger another commit, and verify that the orchestrator cleans up the drift.

Network Automation Framework

REFERENCE ARCHITECTURE



Use Case 2: BGP Peer Drift

Intent	Says **pe1** should use peer **AS 65002**.
Collector	**Telegraf** reads BGP peer state over gNMI; pe1 reports **AS 65099** and its session is down.
Observability	**Kapacitor** compares the actual peer AS with intent and **alerts on the mismatch**.
Orchestrator	Plans to **restore pe1's peer AS**, applying the L1, L2, or L3 policy.
Executor	Can **apply the change** to the Junos configuration.

Containerized Routing Protocol Daemon (cRPD)

1 Best-of-breed routing

BGP filters, route policies, standards compliance; stable and highly performant

2 Operability

Junos familiarity with powerful programming interfaces: pRPD and OpenConfig

3 Agility

Fast, lightweight deployment, iteration and reconfiguration with auto scaling

4 Observability

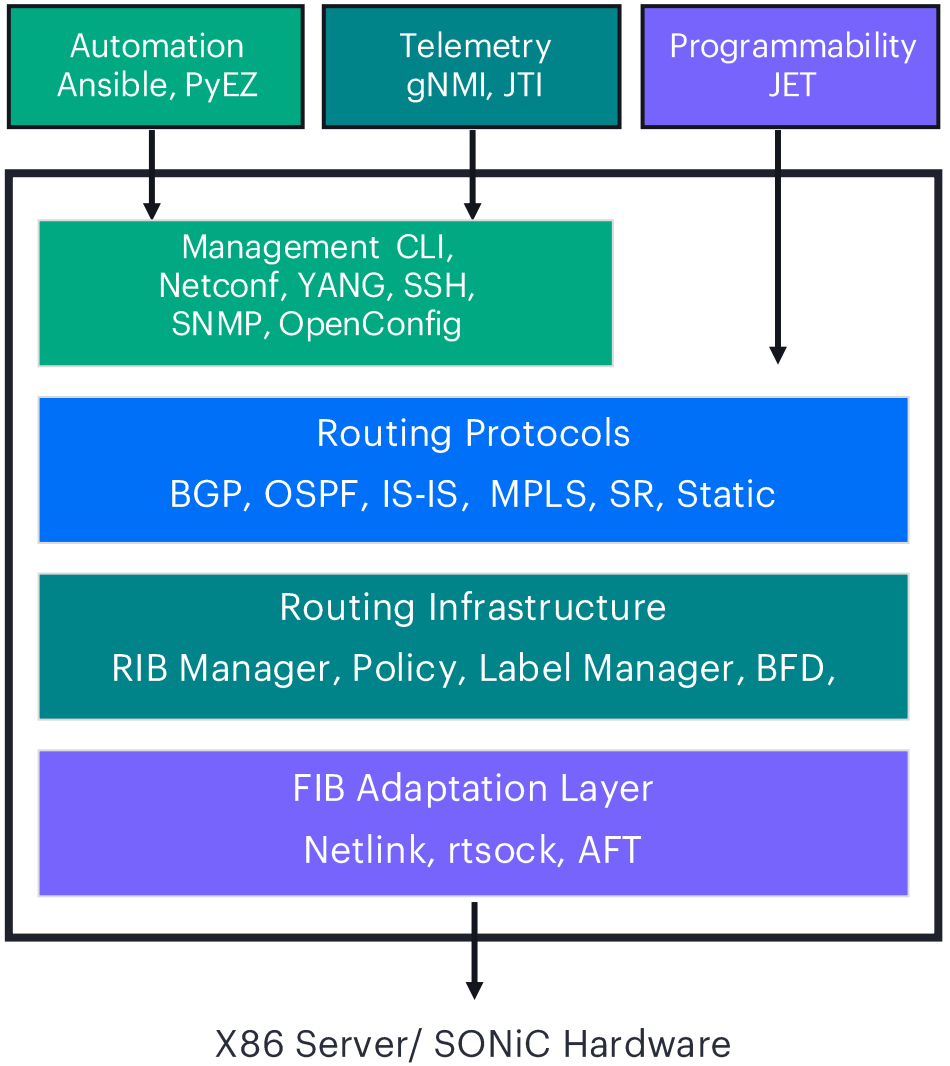
State, health and performance telemetry with gRPC streaming

5 Resilience

Fast automatic recovery from failures

6 Supportability

JTAC support and regular service releases for urgent fixes and enhancements



Containerized Routing Protocol Daemon (cRPD

Routing on the host

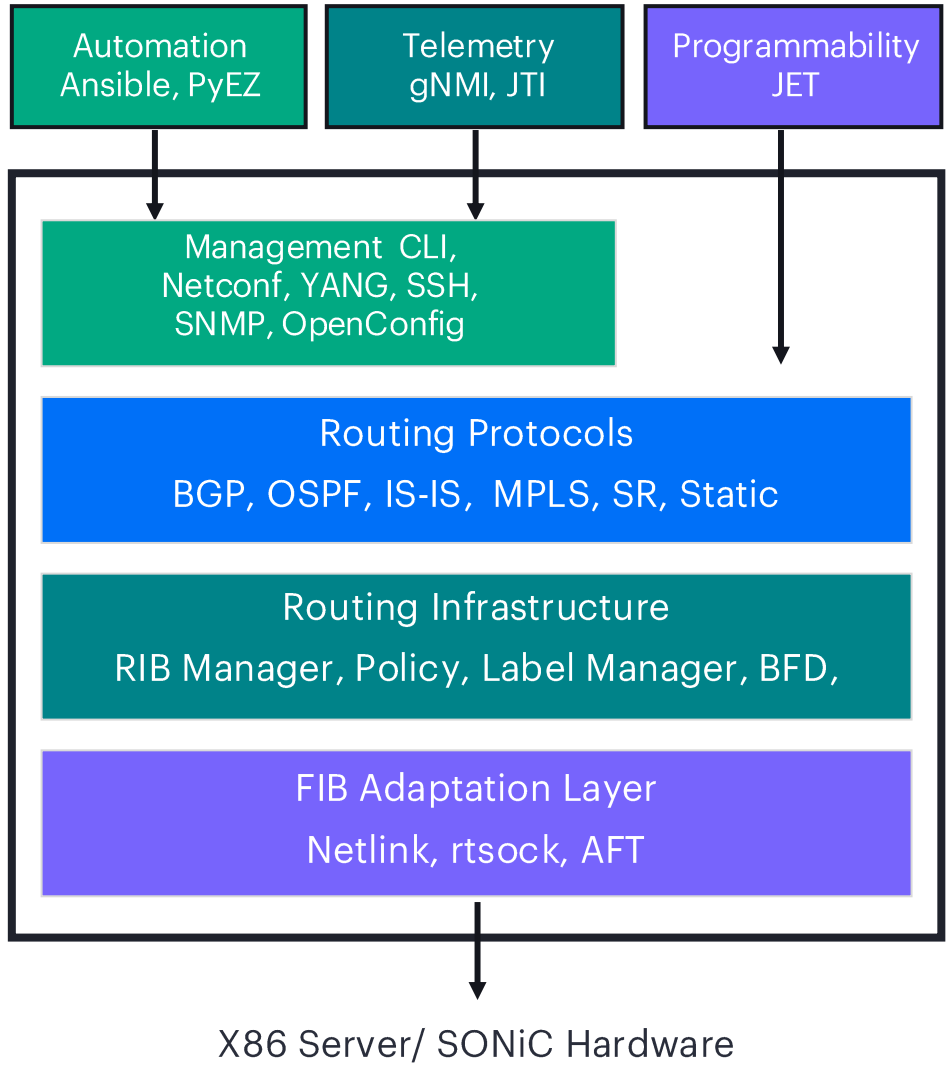
Brings Layer 3 to the application: the host controls the RIB. Server multihoming improves resiliency and uplink use; egress traffic engineering balances load.

Route reflector / route server

Reflects routes in scaled-up iBGP networks, Kubernetes clusters and Internet Exchanges; load balances BGP clients across cRPD instances.

Telemetry and automation

Streams device state with gNMI and JTI and is automated with Ansible and PyEZ: the foundation for drift detection in this workshop.



Device Configuration

Minimum config so the collector can reach each device

PE1 and PE2 (cRPD)	What it does
`set system services ssh`	SSH access (NETCONF runs over SSH)
`set system services netconf ssh`	NETCONF over SSH, TCP 830 (RPC polling)
`set system services extension-service request-response grpc clear-text port 9339`	gNMI dial-in on TCP 9339, no TLS
`set system services extension-service request-response grpc skip-authentication`	gRPC sessions accepted without a login

cSRX	What it does
`set system services ssh`	SSH access (NETCONF runs over SSH)
`set system services netconf ssh`	NETCONF over SSH, TCP 830 (RPC polling)

Lab only: for production, replace `clear-text` with `ssl-profile <name>` (TLS) and remove `skip-authentication`.



BGP Peer Changes on PE1 and PE2

OpenConfig gNMI sensors

Base path: `/network-instances/network-instance/protocols/protocol/bgp/neighbors/neighbor/`

What changed	Path (after base)	Field	Collection
Peer AS	`state/peer-as`	**`session_peer_as`**	Every 60 s
Session state	`state/session-state`	**`session_state`**	**On change**
Session flaps	`state/established-transitions`	**`session_flaps`**	Every 60 s
NOTIFICATIONs sent	`state/messages/sent/NOTIFICATION`	**`session_notif_sent`**	Every 60 s
Peer description	`state/description`	**`session_description`**	Every 60 s

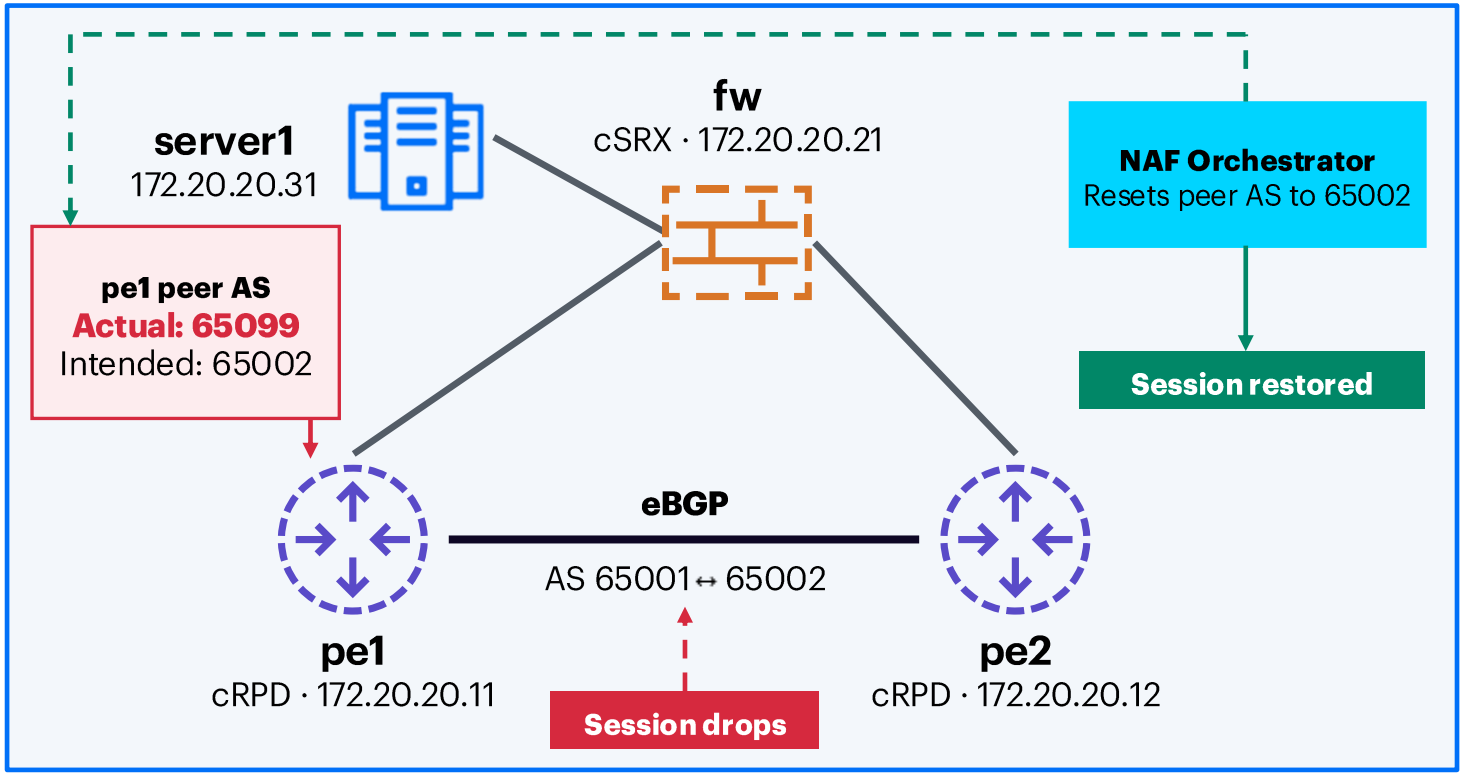
Session state
Already collected; direct state-change notifications require the proposed new rule.

Tip: configured vs. running
NETCONF `NAF_CONFIG` reports the configured peer AS (`cfg_peer_as`). Compare it with `session_peer_as` to confirm a peer AS change took effect.

Lab 2 – BGP Peer Drift

GOALS

Detect and correct an incorrect BGP peer AS so the peering session returns to its intended state.

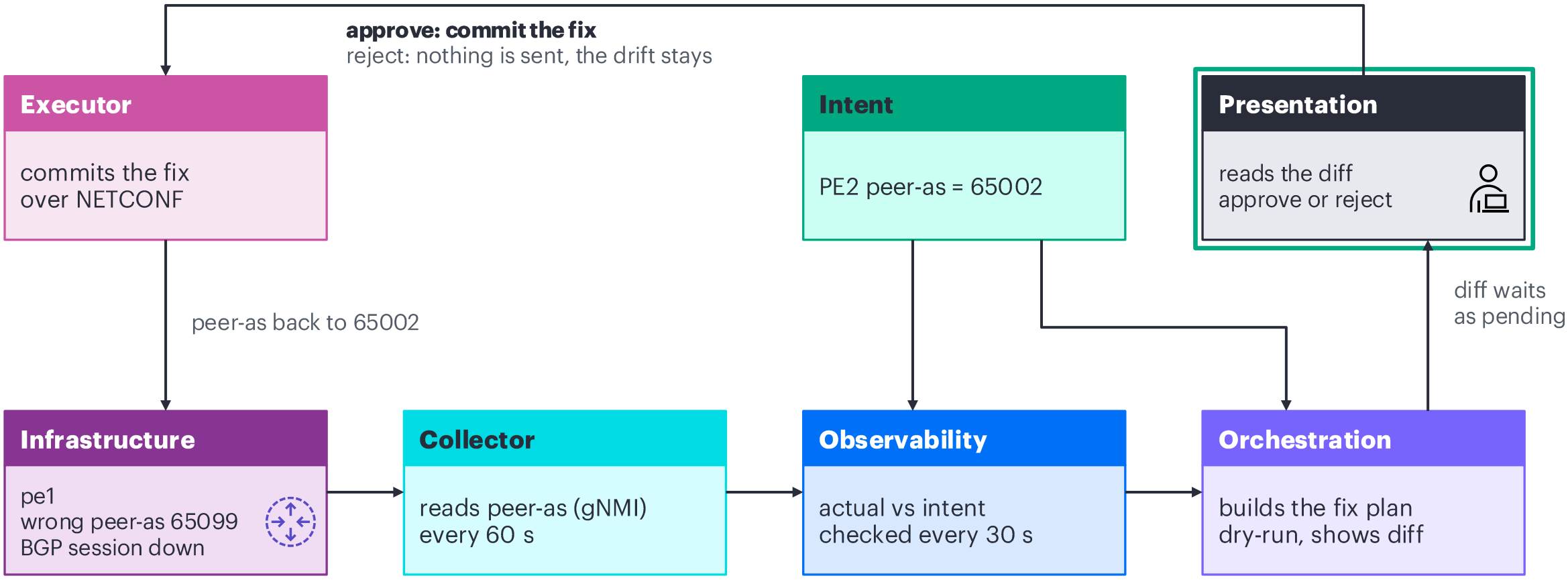


STEPS

1. **Confirm pe1 and pe2** have a healthy BGP session.
2. **Set autonomy to L2** and introduce the wrong peer AS on pe1.
3. **Observe the alert** and compare the reported peer AS with intent.
4. **Review the orchestrator's proposed correction**, then repair the peer AS by hand.
5. **Verify the BGP session recovers** and compare the human involvement at each level.

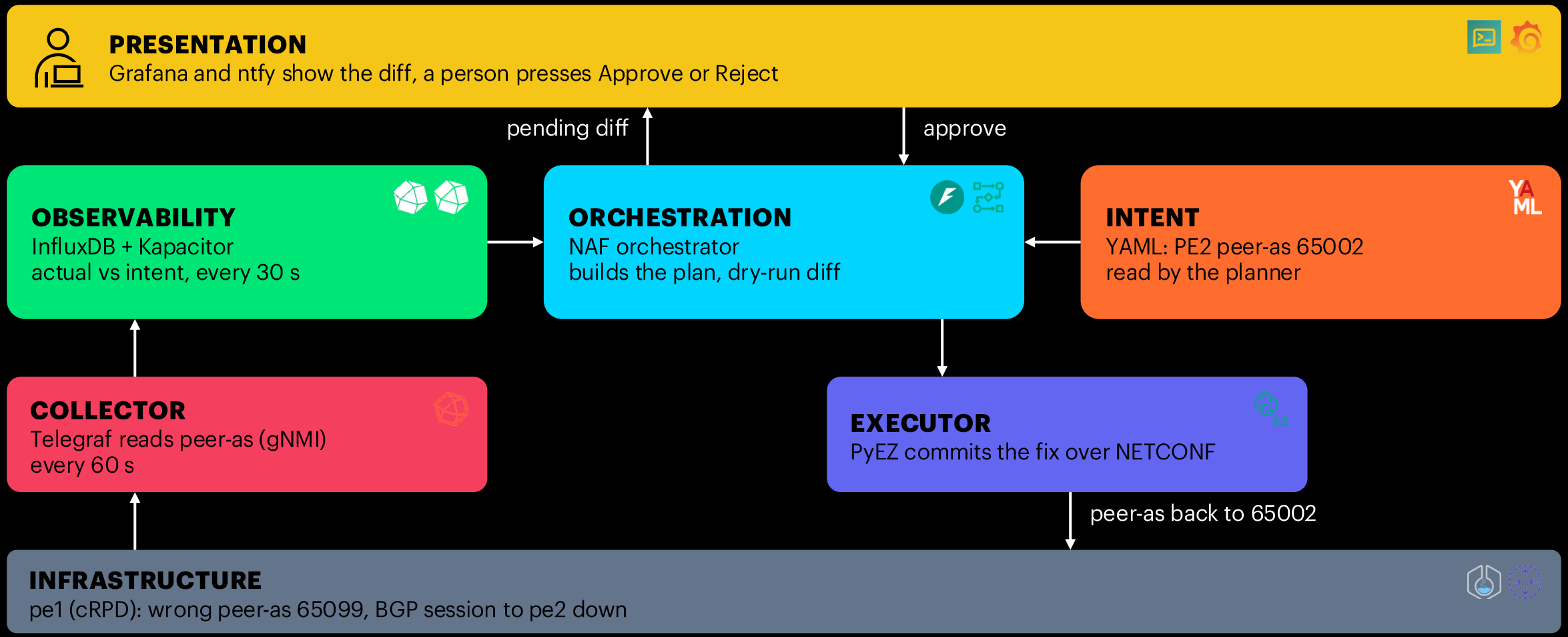
BGP Peer Drift: L2 Assist

Intent is set,



L2 asks. Nothing is committed until a person approves the exact diff.

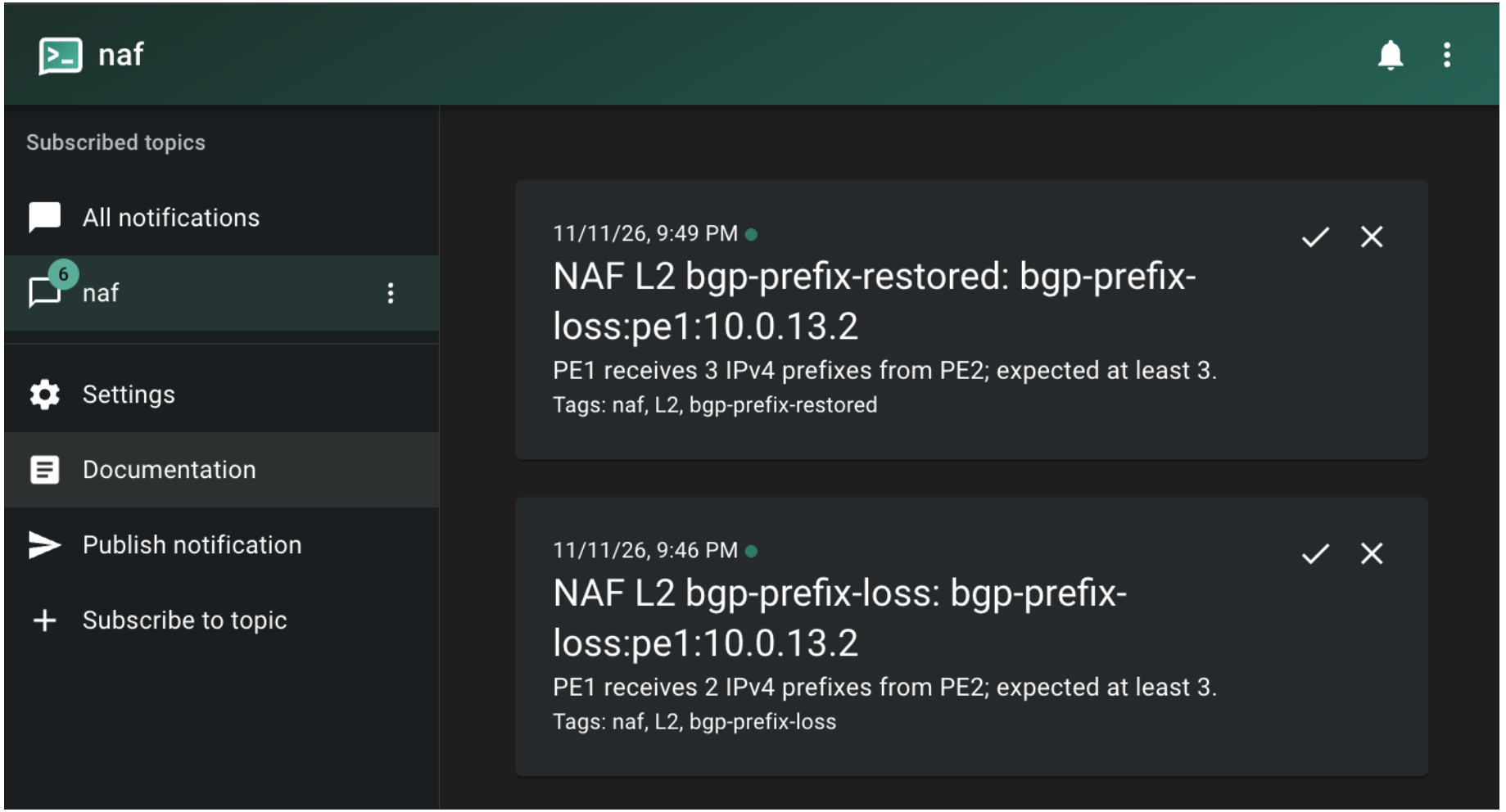
L2 Assist: the loop prepares, a human approves



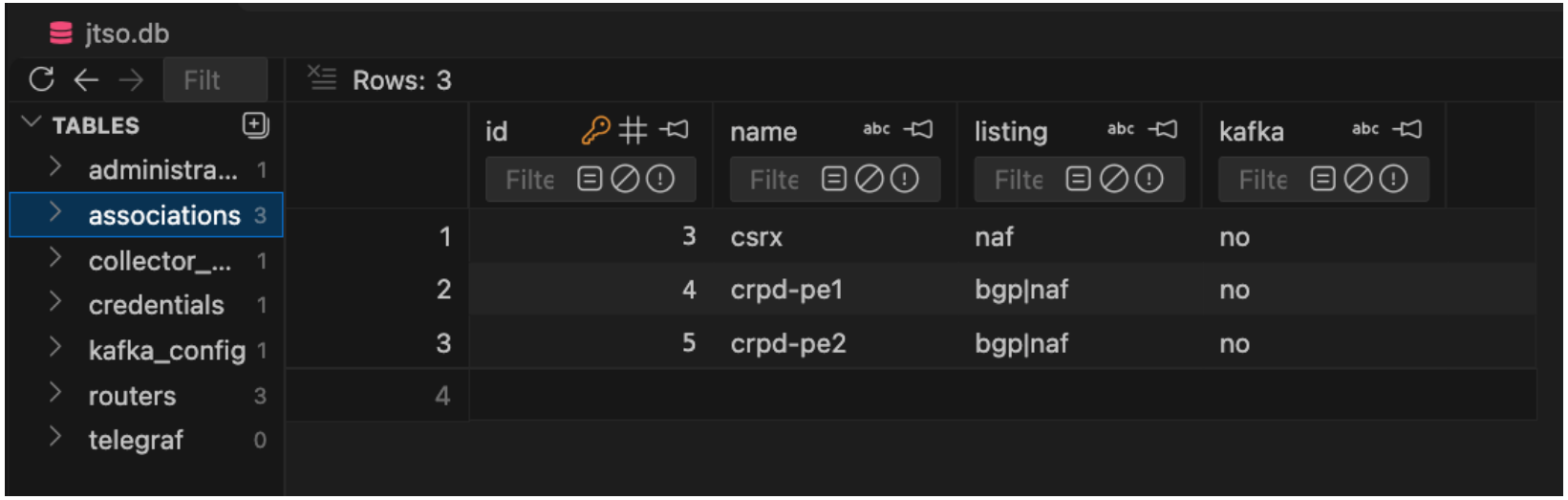
Lab 2 – BGP Peer Drift

ANOTHER VARIATION

When the upstream router changes configuration or state and it impacts your network.



SQL


jtso.db
Filt
Rows: 3
TABLES
administra... 1
associations 3
collector_... 1
credentials 1
kafka_config 1
routers 3
telegraf 0
id
name
listing
kafka
Filte
1 3 csrx naf no
2 4 crpd-pe1 bgp|naf no
3 5 crpd-pe2 bgp|naf no
4


jtso.db
Filt
Rows: 3
Filter 3 rows...
TABLES
administra... 1
associations 3
collector_... 1
credentials 1
kafka_config 1
routers 3
telegraf 0
id
name
short
family
model
version
profile
Filte
1 1 172.20.20.11 crpd-pe1 crpd cRPD 26.2R1.7 1
2 2 172.20.20.12 crpd-pe2 crpd cRPD 26.2R1.7 1
3 3 172.20.20.21 csrx csrx csrx 26.2R1.7 1
4

Workflow: device to dashboard

How we did it. Other options we see people doing.

Enable routers

```
set system services
netconf ssh
set system services
extension-service
request-response grpc
clear-text port 9339
```

Configure Netconf and gNMI users.

– gRPC server: TCP 9339, clear text or TLS

– Netconf server: TCP 830

Credentials

```
Username: lab
Password: lab123
```

Fill in the Netconf and gNMI username and password in the Credential menu.

– TLS and mutual authentication are optional

Add routers

crpd-pe1 → .11

Submit a short name and the hostname or IP address.

– JTSO opens a Netconf session and gathers router family, model, and version

Assign profiles

Profile: naf
→ crpd-pe1
→ crpd-pe2
→ csrx

Select a device, assign one or more profiles, press “create association”.

– JTSO fully re-configures the stack

Visualize

localhost:18080/d/naf-profile

Switch to Grafana and use the pre-included dashboards.

– “No Data”: no data sent, or sensor path not supported

What is OpenJTS?

Juniper Telemetry Stack: open-source gRPC/gNMI Telemetry for Juniper routers



Designed for effortless adoption

In a few clicks, you can collect and visualize the most common data provided by Juniper's Telemetry solution.



One simple web GUI

JTSO (JTS Orchestrator) is the only entry point to manage the OpenJTS stack. The configuration of all components is done via a simple web GUI.

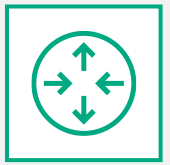


Three GitHub repositories

OpenJTS: infrastructure for deploying the stack, includes all the profiles

JTSO: source code of the orchestrator

JTS Telegraf: Telegraf fork with plugins and enhancements



Supported platforms

PTX10K, MX (vMX, Neo, and 10K platforms) and ACX7K

Junos OS and Junos OS Evolved 20.1 and onwards

OpenJTS stack: complete view

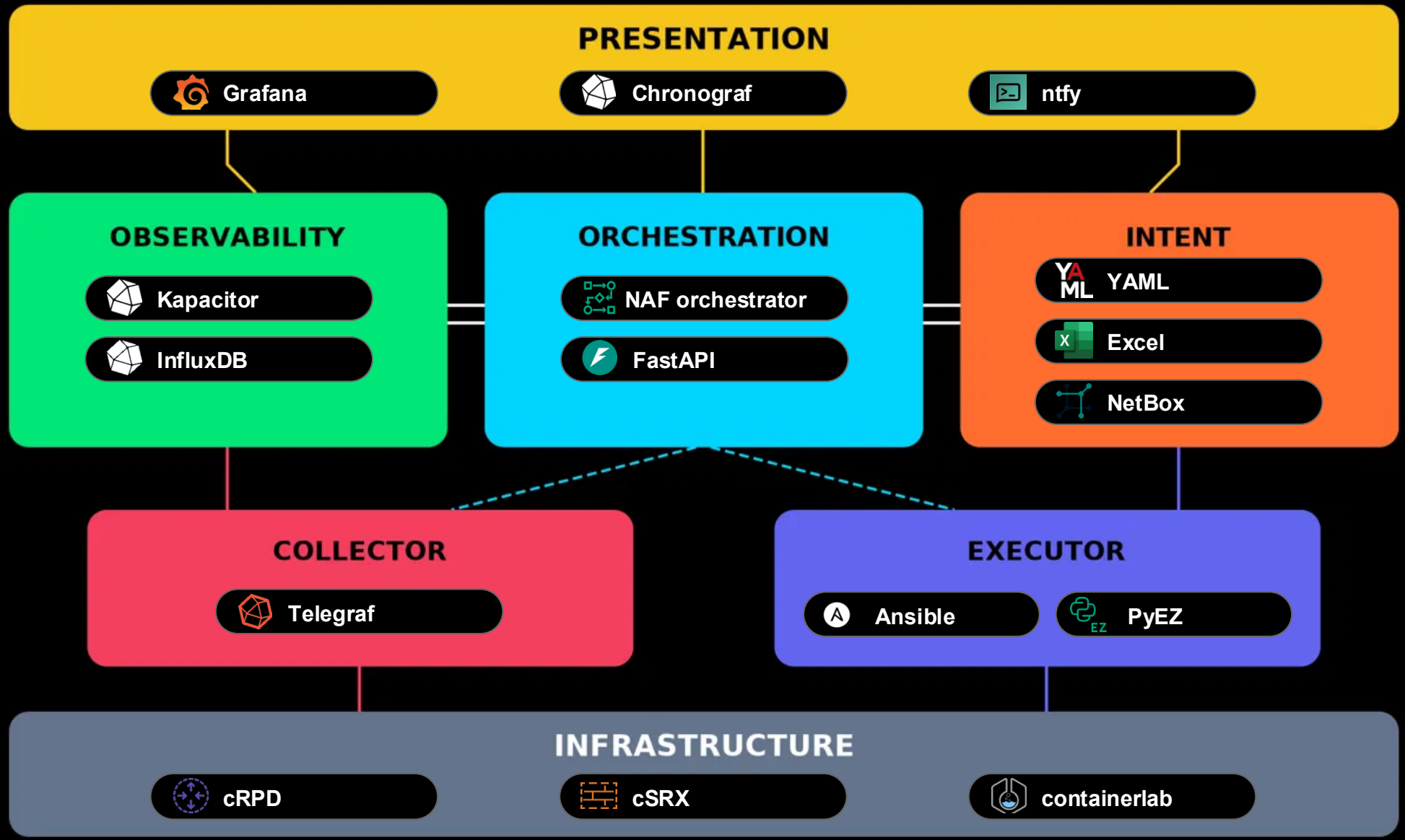
JTSO orchestrates four open-source layers: containers, images, ports and data flow

JTSO Orchestrate Orchestration	One web GUI configures every layer: routers, profiles, Telegraf, dashboards, alarms **jtso · jtso:latest · :80 · OpenJTS GUI** **Inventory** NETCONF facts (family, model, version) · **Profiles** router↔ profile, dashboards, alarms · **Lifecycle** writes Telegraf configs, restarts collectors
Grafana Visualize Presentation	Pre-built dashboards per profile **grafana · grafana:10.3.1 · :8080** **chronograf · chronograf:1.10 · :8081** (InfluxDB / Kapacitor UI)
Kapacitor Alarm and aggregate Observability · alarming	TICK tasks on the time series, loaded from each profile **kapacitor · kapacitor:1.7 · :9292** Plus tasks: **bgp_peer_drift · config_change · prefix_limit**
InfluxDB Store Observability · storage	Time series database **influxdb · influxdb:1.8 · :8086** **database jtsdb**: one measurement per sensor (COMMIT · NETWORK_INSTANCE_BGP · ...)
JTS Telegraf Collect and enrich Collector	gNMI streaming, NETCONF metadata (in: gNMI :9339, NETCONF :830) **jts_telegraf · telegraf_crpd · telegraf_csrx · telegraf_mx, _ptx, ...** crpd: gNMI + NETCONF (cRPD family) · csrx: NETCONF only (cSRX family) · one per device family, exited when not in use

Data flows up: Telegraf writes jtsdb → Kapacitor reads jtsdb, Grafana queries jtsdb · **Control:** JTSO configures every layer

Network Automation Framework

REFERENCE ARCHITECTURE



Use Case 3: Prefix Limit

Intent	Holds the **prefix expectations** for the peer.
Collector	**Telegraf** supplies **prefix counts** and **session-flap** signals.
Observability	**Kapacitor** detects a **leak** or related instability.
Orchestrator	Can apply a **compensating import clamp** to protect pe1.

Human follow-up

The clamp **remains after recovery**. A person must **fix the leak** and **remove the clamp**.

BGP Prefix Changes on PE1 and PE2

Per neighbor, per address family, every 60 seconds

Base path: `/network-instances/network-instance/protocols/protocol/bgp/neighbors/neighbor/afi-safis/afi-safi/state/prefixes/`

Leaf	Field	Meaning
`received-pre-policy`	`safi_received_pre_policy`	Received before import policy
`received`	**`safi_received`**	**Received (post-policy)**
`accepted`	`safi_accepted`	Accepted
`rejected`	`safi_rejected`	Rejected
`installed`	`safi_installed`	Installed
`sent`	`safi_sent`	Advertised to the peer

Other prefix views in the JSON

Per peer group (gNMI, 60 s)
`group_safi_received, group_safi_accepted, group_safi_installed, group_safi_sent`

Whole BGP instance (gNMI, 60 s)
`global_total_prefixes, global_total_paths`

Route tables (NETCONF `ROUTE_SUMMARY`, 60 s)
`rt_destinations, rt_total, rt_active, rt_hidden`

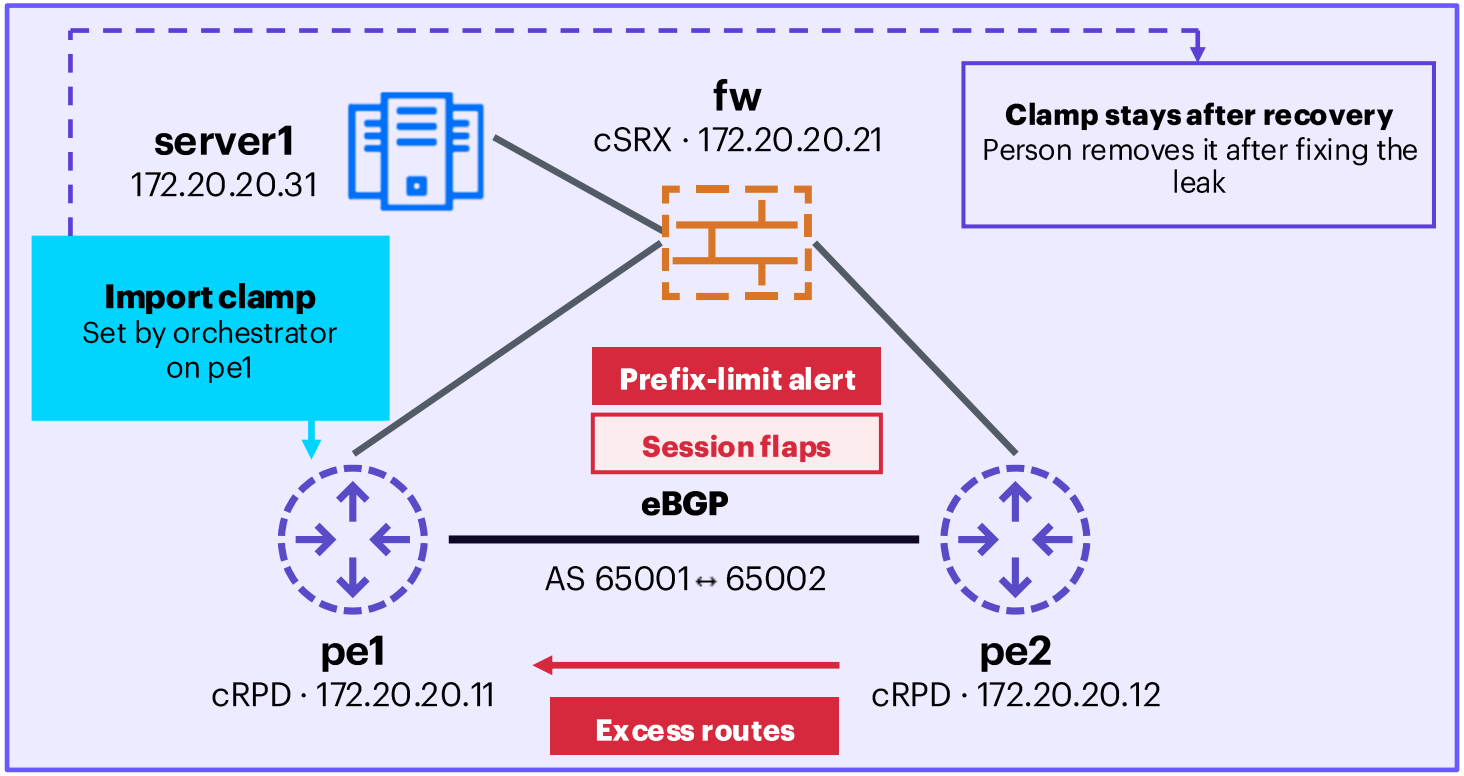
Tip: A drop in `safi_received` with no change in `session_flaps` points to a policy or upstream change, not a flap. Check `commit_epoch` and `cfg_import` for a recent commit.



Lab 3 – Prefix Limit

GOALS

Detect a route leak and protect pe1 with an import clamp until the leak is fixed.



STEPS

1. **Confirm the pe1-pe2 BGP session is healthy** and the prefix count is within intent.
2. **Introduce a staged route leak from pe2** and observe the warning followed by the critical alert.
3. **Compare prefix telemetry, session flaps**, and the proposed import clamp.
4. **At L1, inspect the proposed response**; at L2, review and approve the clamp.
5. **At L3, verify that the orchestrator applies the clamp** and the BGP session stops flapping.
6. **Confirm the alert recovers** but the compensating clamp remains in place.
7. **Stop the route leak**, then remove the clamp by hand and verify normal routing.

Break 15:45-16:15

© 2026 Hewlett Packard Enterprise Development LP

What is the cSRX?

HPE Networking's containerized firewall. Not a full Junos router.

Docker commands

Environment variables
Very quick to deploy

Limited routing and functions

Static routing only
No dynamic routing protocols, No HA, No software upgrades. If needed use full VM (vSRX)

Deployment as code

Containerized firewall (cSRX)
Scripted container creation

cSRX

Focused on container security

Tips for using cSRX

- **Netconf:** NETCONF XML to common method to automate.
- **SSH enabled:** ssh is not enabled by default.
- **Meant to be disposable:** need to upgrade, new container. Need a different baseline, new variables.
- **Reproducible:** Make resets deliberate and reproducible. Ie, not stopping and starting. Make a new container with corrected configuration (not today. In production)
- **Limited scale per FW:** scaling cSRX means more containers running cSRX, not more cores/ram.

What is the vSRX?

HPE Networking's virtual firewall. A Junos OS–based virtual firewall combining SRX security and routing in a virtual machine

Deployment Options	Full routing and functions	SRX security services
Virtual Machine Deployment Vmware ESXi, KVM, Hyper-v, AWS, Azure, GCS	**BGP/OSPF/ISIS/MPLS/BFD/etc.** Many routing and switch options of JUNOS are available with the full VM (vSRX)	**NGFW in a VM (vSRX)** Next-generation security capabilities depend on licensing and release

vSRX

SRX security and Junos routing in a VM—not a container

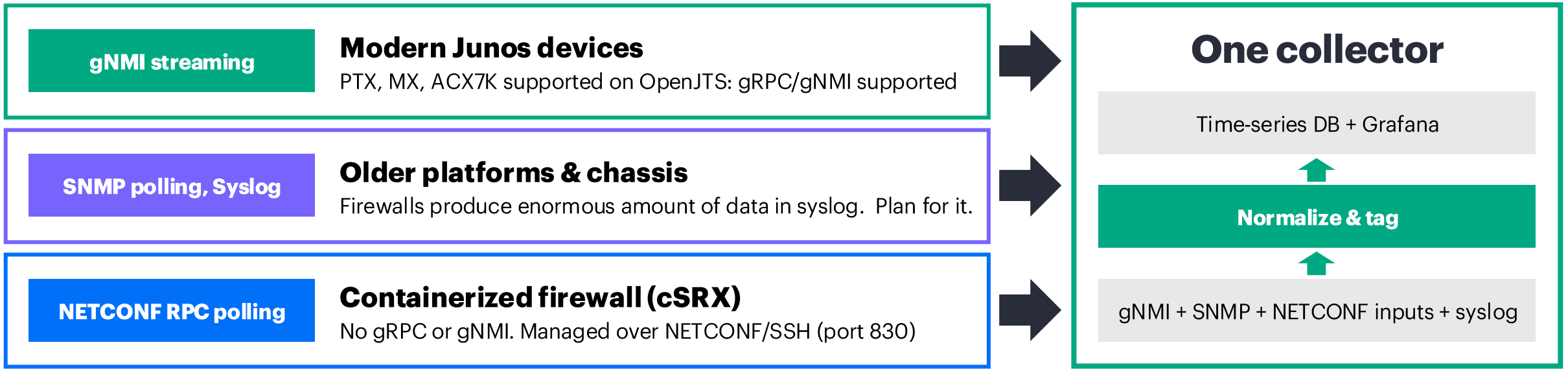
Tips for using vSRX

- **Netconf:** NETCONF XML to common method to automate.
- **SSH enabled:** ssh is not enabled by default.
- **Sizing :** Allocate core count CPU and RAM and virtual NIC and can use SR-IOV for high performance. Up to 32 cores supported today for market leading performance.
- **Operations:** backup and upgrades full featured.
- **Plan continuous operations:** High Availability (HA) and Multi-Node High Availability (MNHA) both are enabled for vSRX

Note: we are not using vSRX today

Mixing Sensors: gNMI, SNMP & NETCONF

Use the best sensor each device supports, and land everything in one collector



Tips for mixing sensors

- **Stream first:** use gNMI wherever the platform supports it.
- **Poll gently:** pull-based, so pick intervals the Routing Engine can handle and expect gaps between polls.
- **Tag the source:** label each metric with protocol and device.
- **Normalize:** map SNMP OIDs and NETCONF XML to common metric names and units.
- **Secure it:** SNMPv3, and NETCONF over SSH with a read-only user.
- **Plan the exit:** swap in gNMI as hardware is refreshed.

Telemetry Sensors vs. NETCONF Polling

Two ways this lab collects data from PE1, PE2, and cSRX

	OpenConfig sensor (gNMI dial-in)	NETCONF RPC poll
How data moves	Push: collector subscribes once, device streams	Pull: collector sends an RPC every interval
Transport / port	gRPC over TCP 9339	NETCONF over SSH, TCP 830
Data format	OpenConfig YANG paths	Junos XML RPC replies
Modes	SAMPLE, ON_CHANGE, TARGET_DEFINED	Request / response only (no ON_CHANGE)
Lowest interval	2 s (PFE sensors, Junos 23.4R2+); interval set in multiples of 2 s	No Junos minimum; set by the collector. Floor = time for the device to answer the RPC
Default interval	30 s if the collector sets none	None; the collector decides
Seeing a change	ON_CHANGE: sent as soon as the value changes	Seen at the next poll (up to 30 s in this lab)
Device work	Subscription set up once, then streamed	mgd processes a full RPC on every poll
Best for	Counters and state that have a sensor (BGP, interfaces)	Data with no sensor: commits, config, files, SRX security
In this lab	PE1/PE2: 60 s SAMPLE; session-state ON_CHANGE	PE1/PE2/cSRX: 30 s and 60 s



cSRX Monitoring

NETCONF only (no gRPC configured on cSRX)

Measurement	RPC	Fields written to InfluxDB	Interval
INTERFACES	get-interface-information	if_mtu, if_admin_status, if_oper_status	30 s
SYSTEM_STORAGE	get-system-storage	storage_used_pct, filesystem_name (/ only)	60 s
SECURITY_FLOW	get-flow-session-information <summary/>	flow_active_sessions, flow_active_unicast_sessions, flow_failed_sessions, flow_active_drop_sessions, flow_max_sessions	60 s
SECURITY_FLOW	get-flow-statistics-all	flow_pkts_rx_rate, flow_pkts_tx_rate, flow_pkts_fwd_rate, flow_pkts_drop_rate	30 s
SECURITY_SPU	get-performance-summary-information	spu_cpu_pct, spu_mem_pct, spu_flow_sessions, spu_max_flow_sessions	30 s
COMMIT	get-commit-information	commit_epoch, commit_user, commit_client (latest)	30 s
NAF_LOGFILES	file-list /var/log/	log_file_size, log_file_size_rate	30 s
NAF_SECPOL_HITS	get-security-policies-hit-count	hit_count, policy_action tags: policy_name, from_zone, to_zone	30 s
naf_secpolicy	get-firewall-policies Python poller	rule_order	30 s default



Use Case 4: Security Policy Drift

The scenario	A **cSRX firewall** guards a fictional store the night before it opens. Keep it working and **prove it stays safe** with evidence.
The evidence	Evidence comes from **NETCONF snapshots, RT_FLOW logs** and **client probes**. cSRX has no streaming telemetry.
The guardrail	PCI-aligned exercise, **not a PCI DSS assessment**. No payment data, just simulated bad policy.

L1 · Observe: loop reports, you fix

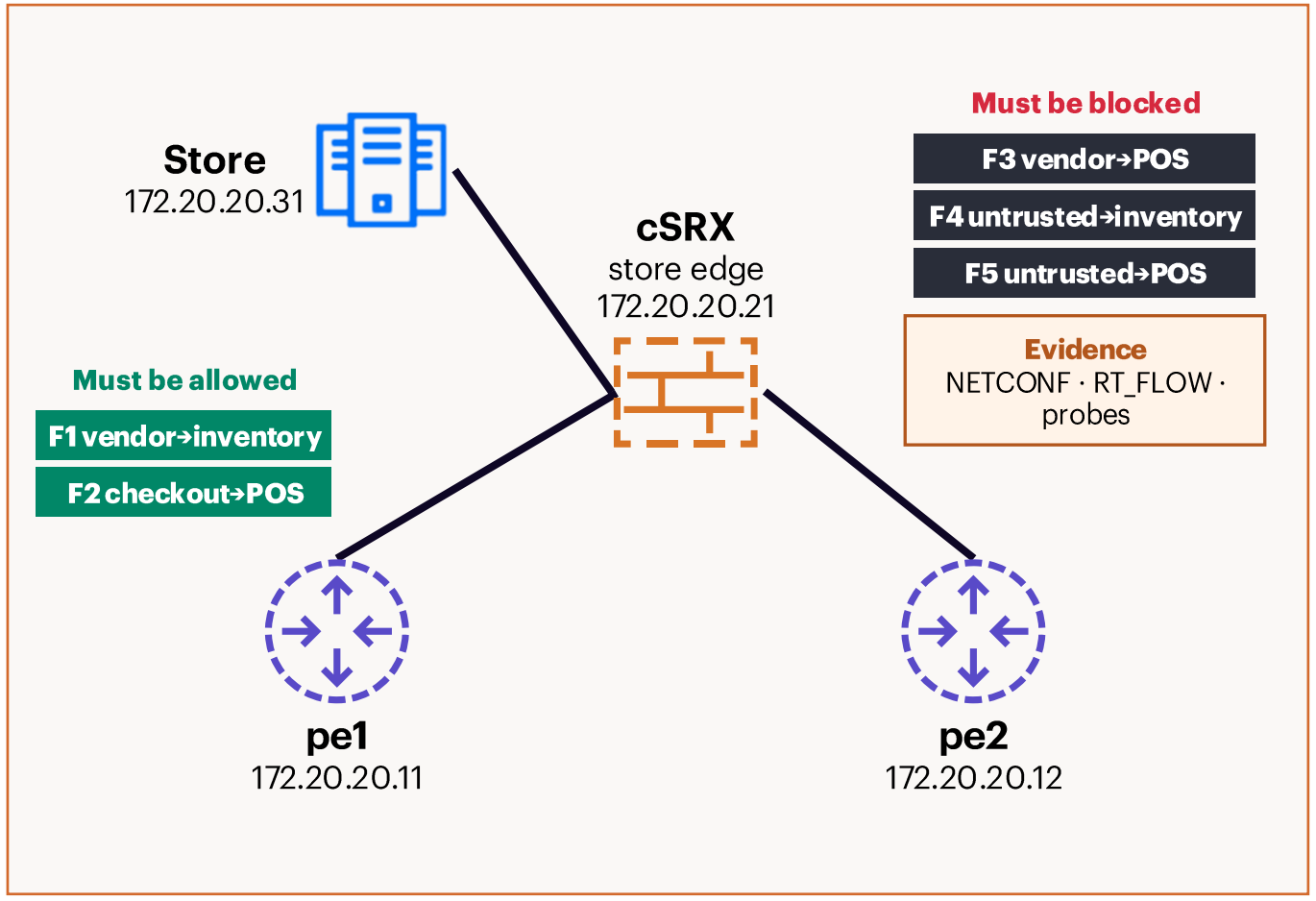
L2 · Assist: you approve the diff

L3 · Autonomous: you verify

Lab 4 - Security Policy Drift

GOALS

Keep the store working and prove it stays safe, using evidence rather than a green light.

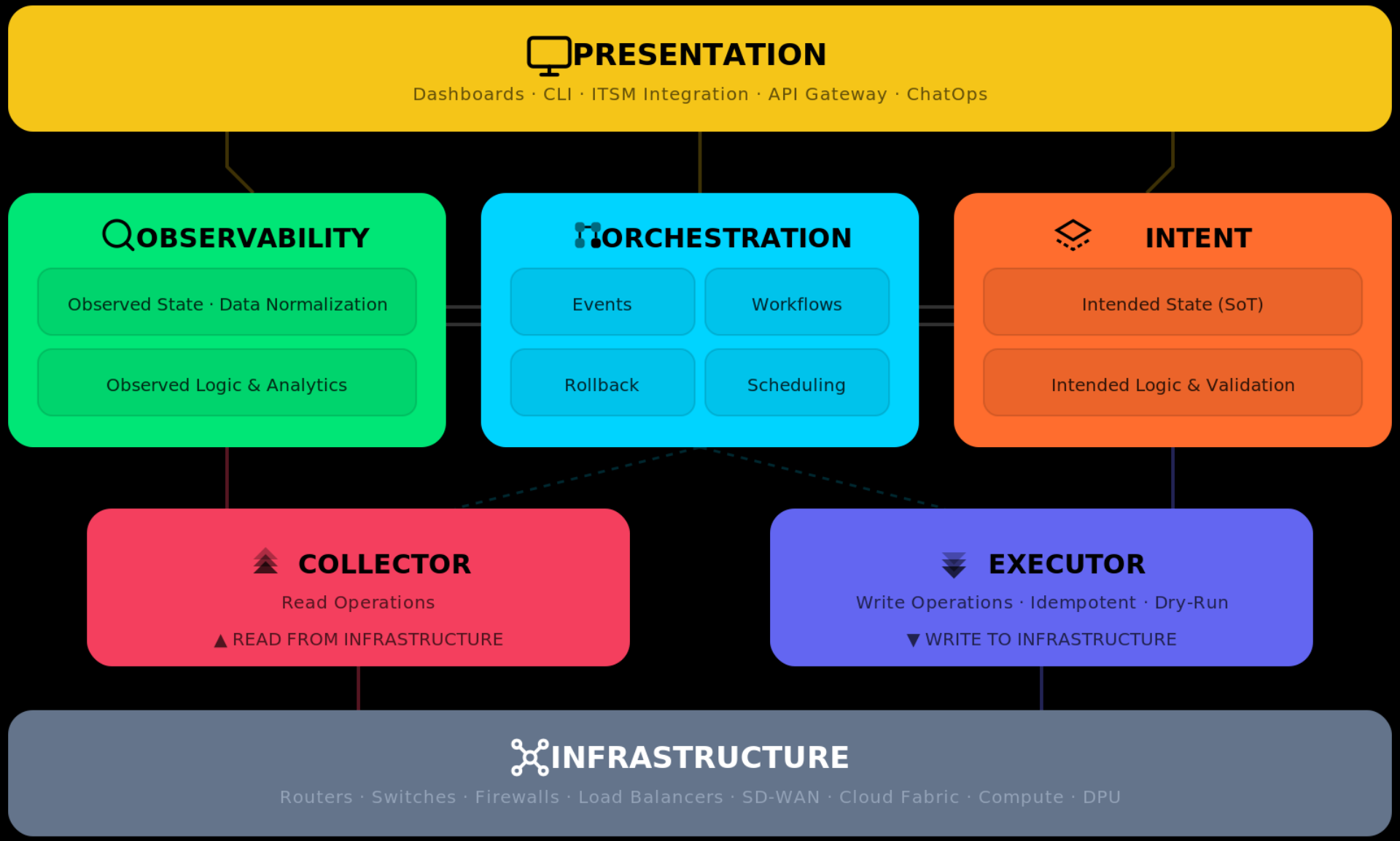


STEPS

1. **Before you start**– reset to baseline
2. **L1: Something is exposed** – Break it (`break.sh`). Check it (`poller.sh`) tool provided. Check notified in JTS, fix by hand. Check (`latency.sh`) how how long you where exposed.
3. **L2: Review the /pending Fix or** – Break it. See proposed fix, approve. Check how how long you where exposed.
4. **L3: Break, the loop commits and verifies** – Break it. verify its autonomous fix from evidence. Check how how long you where exposed.

Network Automation Framework

REFERENCE ARCHITECTURE



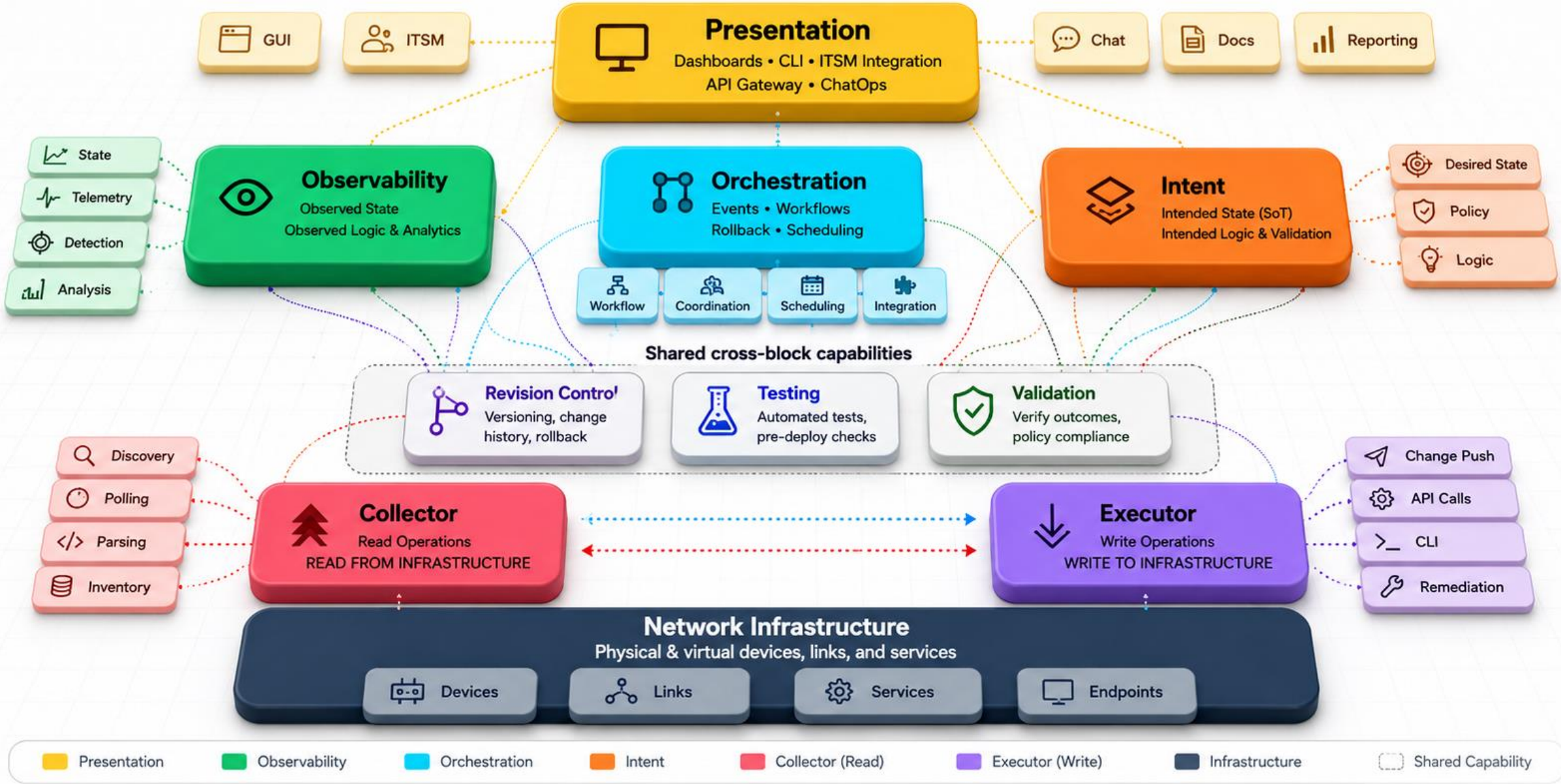
Applications at a Glance

Block	Application	What it does in the lab	Used in
Presentation	**Grafana**	NAF PROFILE dashboard: drift panels, decision log	All labs
	ntfy	Push notifications; Approve and Reject at L2	All labs
	Chronograf	openJTS UI for InfluxDB and Kapacitor	Optional
Intent	**YAML**	intent.yml: the default source of truth	All labs
	Excel	intent.xlsx: the same intent in a spreadsheet	**Swap a block**
	Fake NetBox	NetBox-style API (port 8001), same intent	**Swap a block**
Orchestration	**NAF orchestrator**	Picks a planner, reads intent, applies L1, L2 or L3	All labs
	FastAPI	Orchestrator API: alerts, pending, approve, reject	All labs
Observability	**InfluxDB**	Stores telemetry and seeded intent (jtsdb)	All labs
	Kapacitor	TICKscripts compare actual vs intent, then alert	All labs
Executor	**PyEZ**	Default: NETCONF dry-run diff, then commit	All labs
	Ansible	Alternative executor over NETCONF, same plan	**Swap a block**
Collector	**Telegraf (openJTS)**	Reads gNMI and NETCONF data (naf profile)	All labs
	Python NETCONF poller	Reads cSRX rule order, or commits as a swap	**Security policy drift & swap a block**
	Syslog listener	Devices push commit events, no polling	**Swap a block**
Infrastructure	**containerlab**	Runs the lab network on the workshop VM	All labs
	Juniper cRPD x 2	pe1 and pe2, the eBGP routers	All labs
	Juniper cSRX + server1	Firewall, with a Linux server behind it	**Security policy drift**

Yellow = only in that lab. Swap a block replays BGP peer drift and swaps one block per round.

Each Functional Block Has an Entourage

Every block has a primary role plus an entourage of capabilities.
Revision Control, Testing, and Validation are shared cross-block capabilities.



Use Case 5: Swap a Block

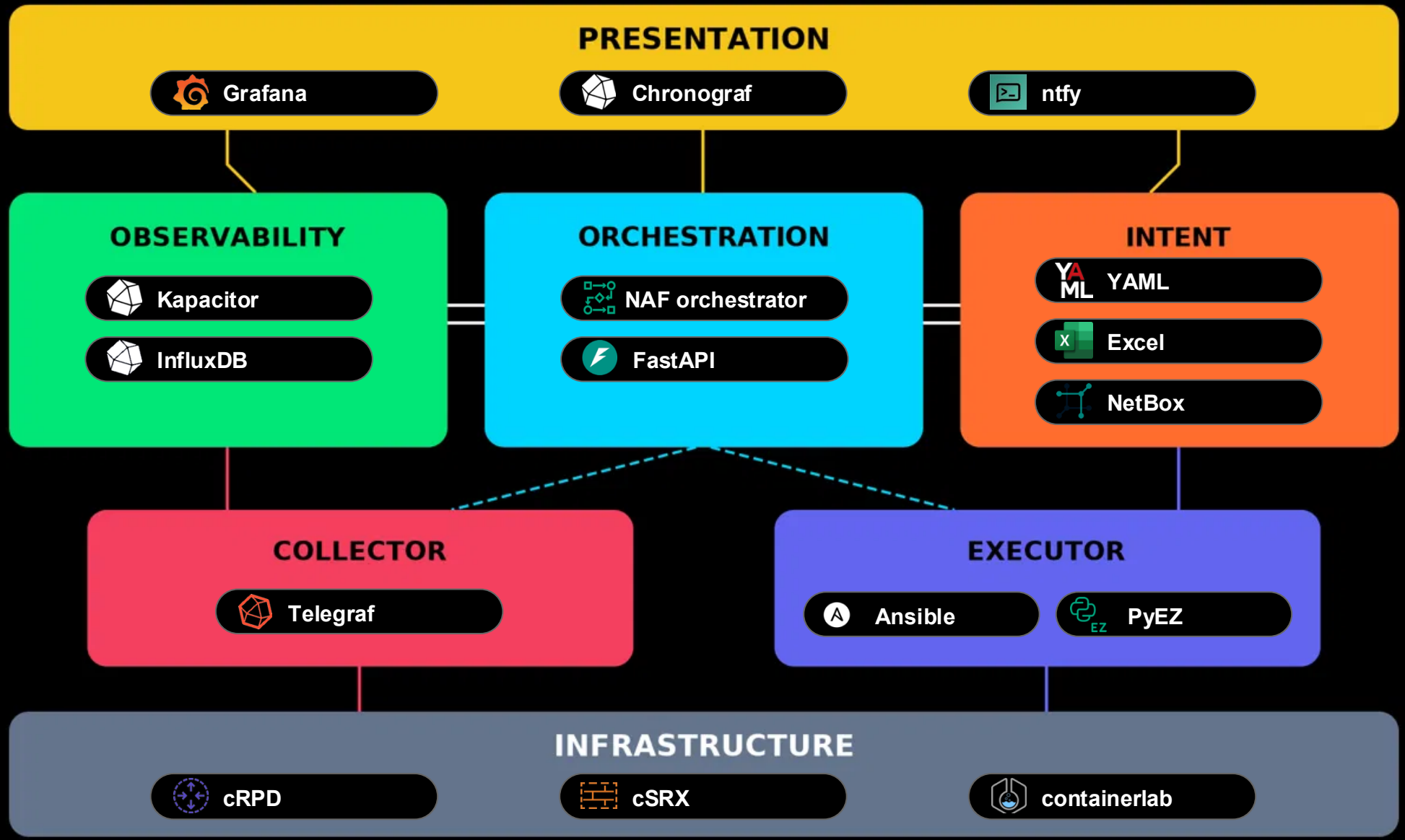
Intent	Swap the backend: **YAML**, **Excel**, or the **fake NetBox API**.
Collector	Swap the path: **Telegraf** (gNMI and NETCONF), the **syslog listener**, or the **Python NETCONF poller**.
Executor	Swap the tool: **PyEZ** or **Ansible**.

A modularity exercise, not a different fault

Replay **BGP peer drift** while swapping blocks. The aim is to **preserve the same event and plan** across implementations.

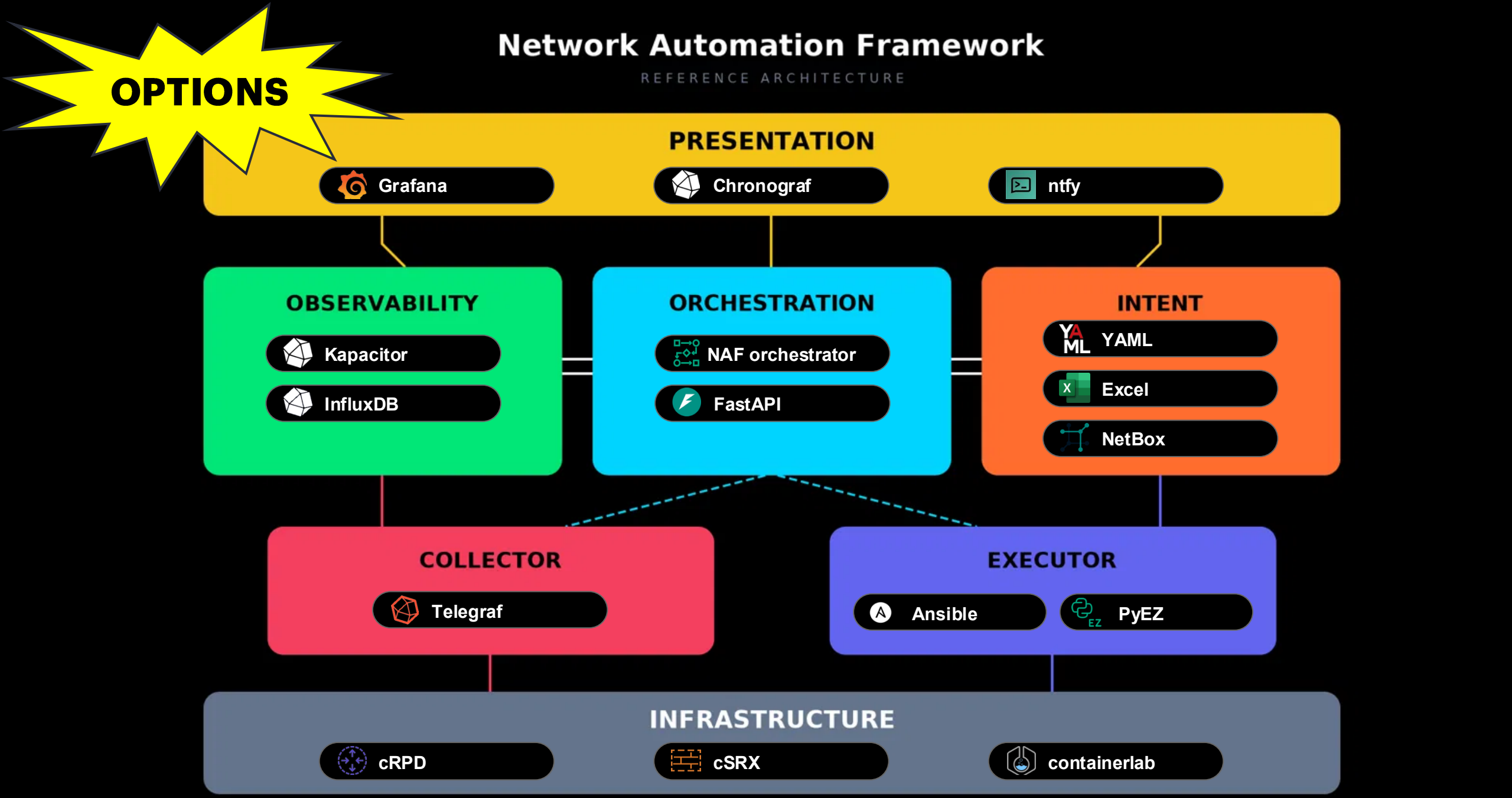
Network Automation Framework

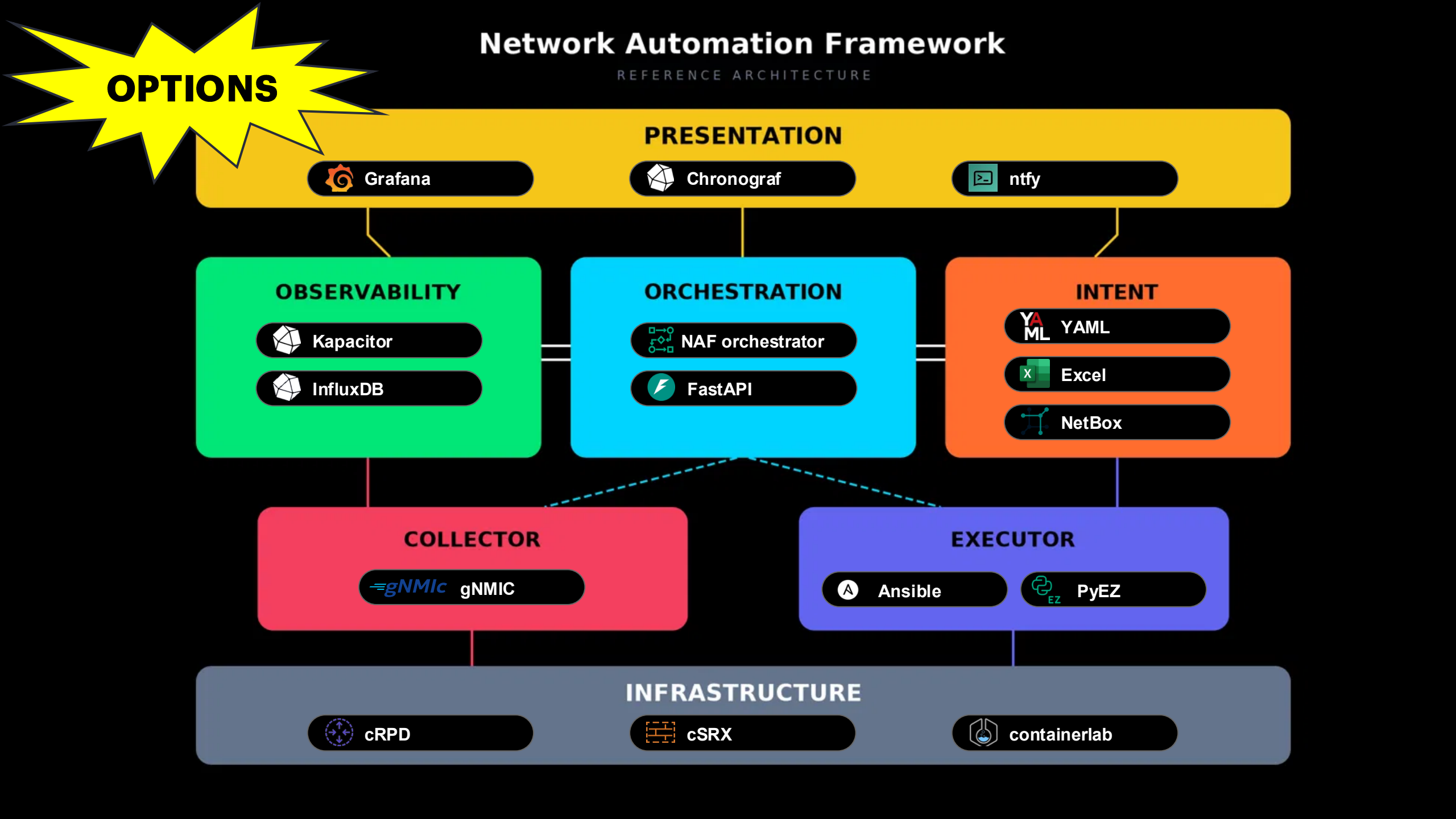
REFERENCE ARCHITECTURE



Network Automation Framework

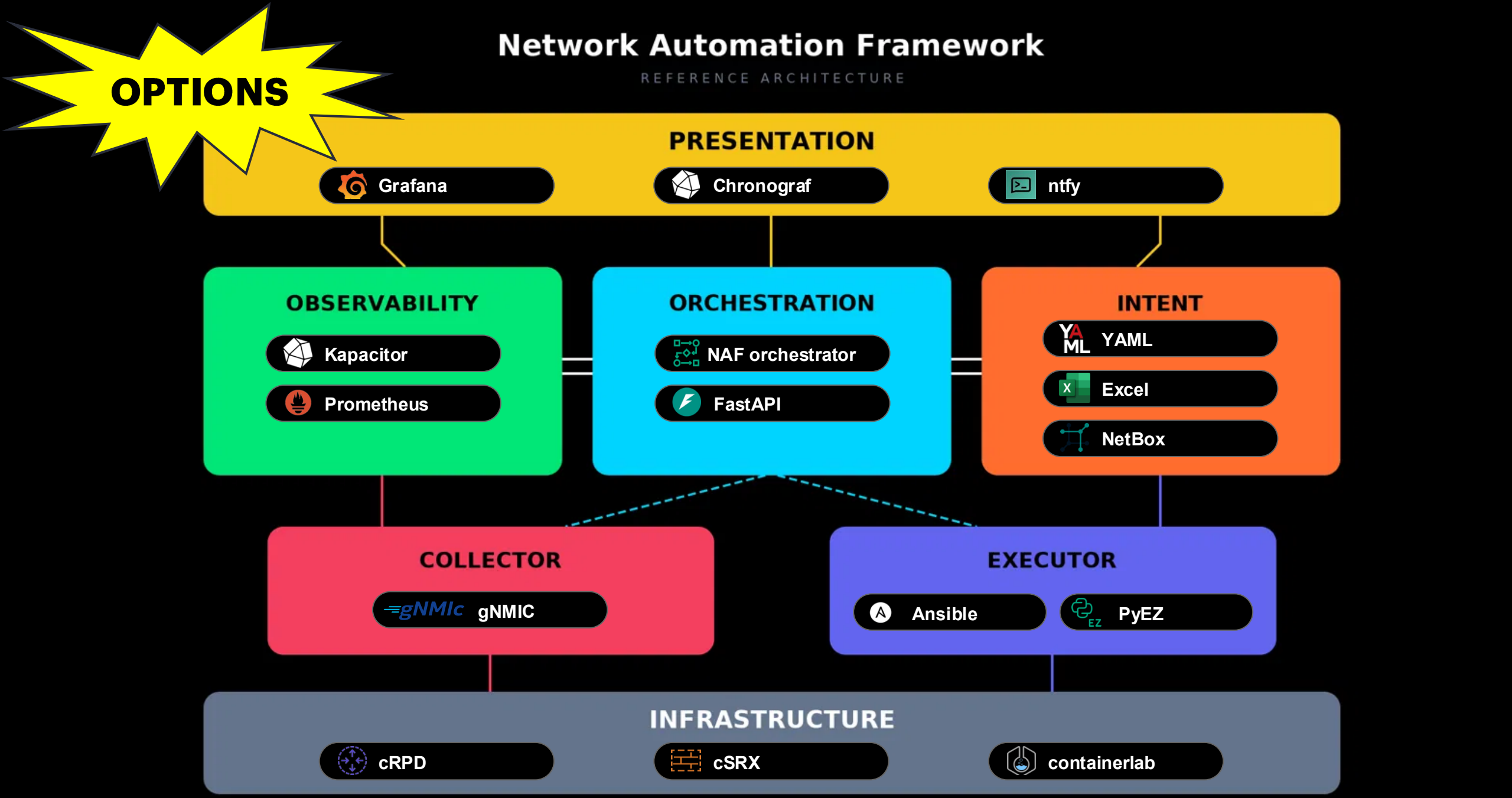
REFERENCE ARCHITECTURE




OPTIONS
Network Automation Framework
REFERENCE ARCHITECTURE
PRESENTATION
Grafana
Chronograf
ntfy
OBSERVABILITY
Kapacitor
InfluxDB
ORCHESTRATION
NAF orchestrator
FastAPI
INTENT
YAML
Excel
NetBox
COLLECTOR
gNMIC
EXECUTOR
Ansible
PyEZ
INFRASTRUCTURE
cRPD
cSRX
containerlab

Network Automation Framework

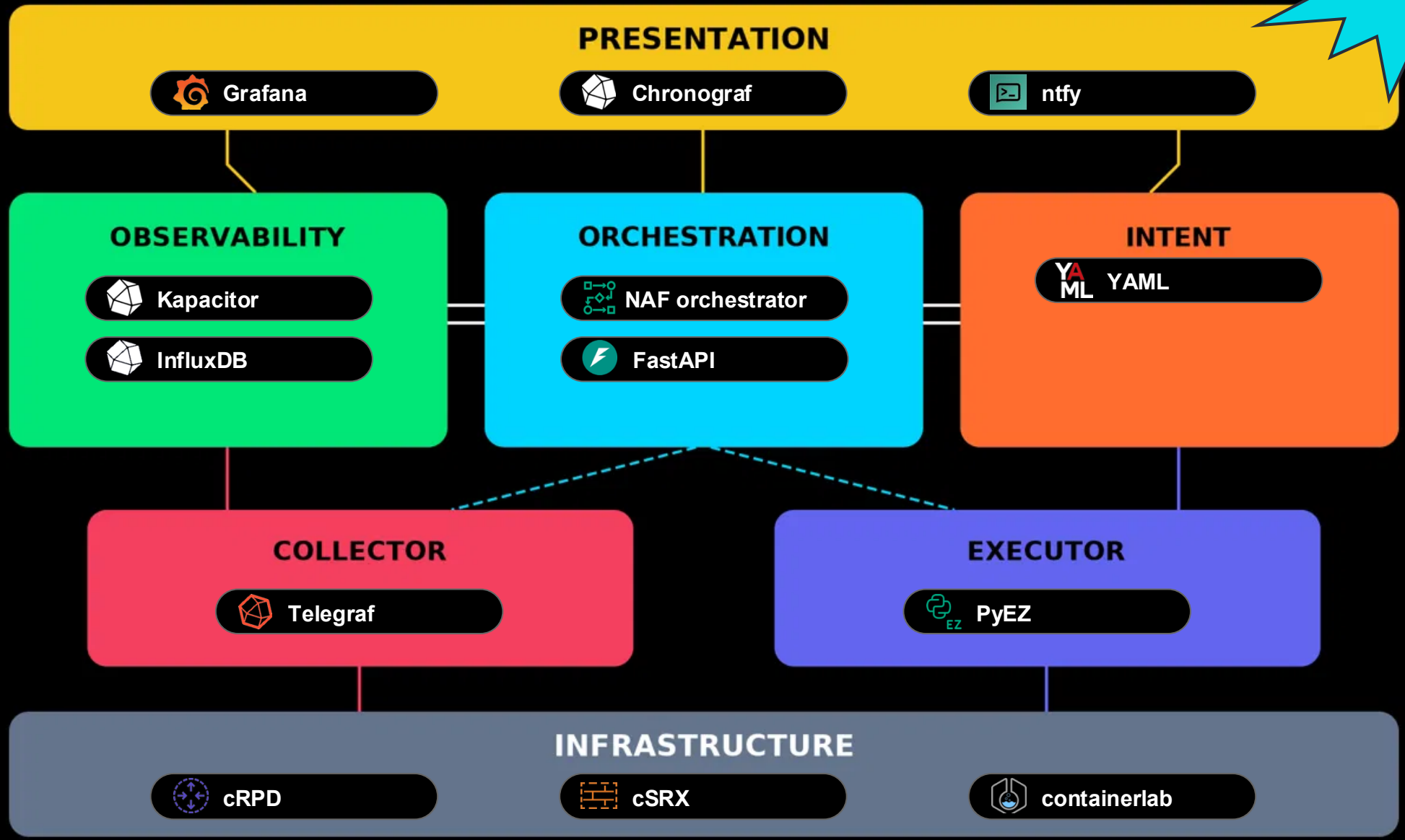
REFERENCE ARCHITECTURE



Network Automation Framework

REFERENCE ARCHITECTURE

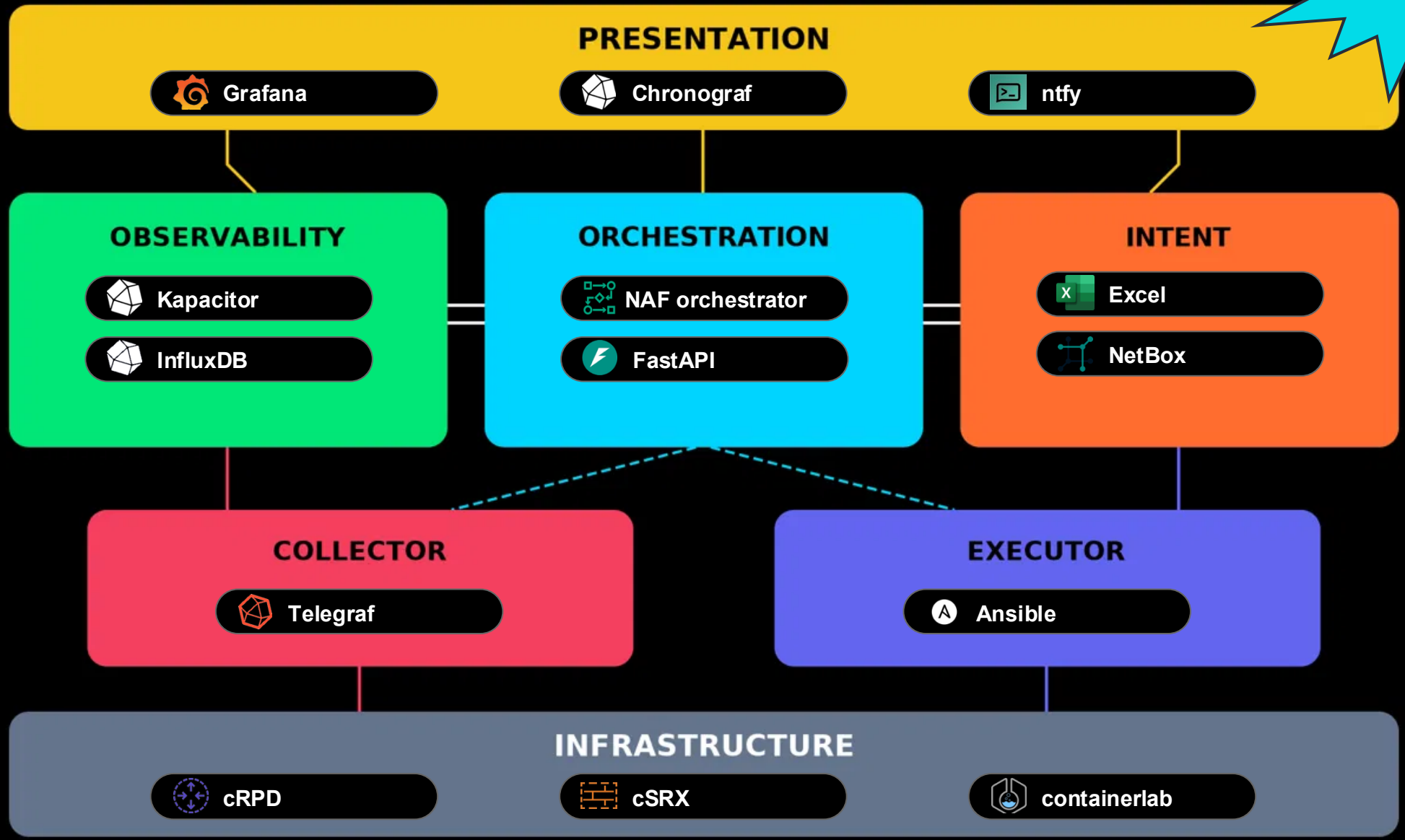
LAB 5



Network Automation Framework

REFERENCE ARCHITECTURE

LAB 5

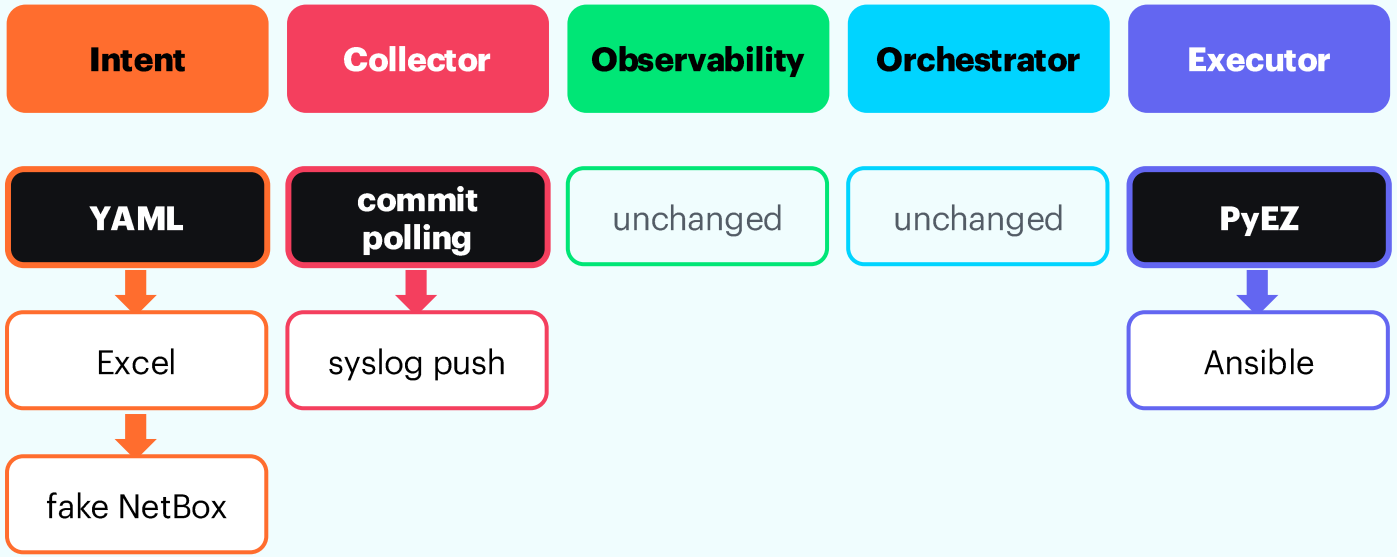


Lab 5 - Swap a Block

GOALS

Demonstrate that changing the Intent, Executor, or Collector implementation preserves the same scenario outcome.

Swap one block, keep the rest of the loop



Filled = starting option. Each swap replays the same BGP peer drift.

STEPS

1. **Start with the BGP peer drift exercise at L2** using YAML intent and the PyEZ executor.
2. **Replay the drift with Excel intent** and compare the alert, plan, and pending change with the baseline.
3. **Switch to fake NetBox intent**, replay the drift, and compare the result again.
4. **Switch from the PyEZ executor to Ansible**, replay the drift, and compare the plan and change.
5. **Switch the logging-drift collector from commit polling to syslog push** and compare the alert and remediation result.
6. **Review any pending change** before moving to the next round.
7. **Restore the baseline settings** and stop any helper services started for the swaps.

Same alert. Same remediation result.

Mapping the lab to the Framework

The same product can contribute to more than one block: classify components by the function they perform, not only by productname.
Product boundary: openJTS supplies services used across the architecture (Telegraf collects, InfluxDB stores, Kapacitor processes, Grafana presents). **JTSO** manages devices and assigns the telemetry profile; it does not make the whole openJTS stack a single NAF block.

	PRESENTATION	OBSERVABILITY	ORCHESTRATION	INTENT	COLLECTOR	EXECUTOR
1 Logging Drift Commit outside window	·	Kapacitor	Diff vs. intent	Logging settings	openJTS naf poll	·
2 BGP Peer Drift Peer AS mismatch	·	Kapacitor alert	Restore peer AS	Peer AS 65002	Telegraf (gNMI)	Apply to Junos
3 Prefix Limit Prefix leak on pe1	·	Kapacitor	Import clamp	Prefix expectations	Telegraf counts	·
4 Security Policy Live run deferred	·	Alert logic	Planner	·	Collector	·
5 Swap a Block Same event, new parts	·	·	·	YAML / Excel / NetBox	Telegraf / syslog / poller	PyEZ / Ansible

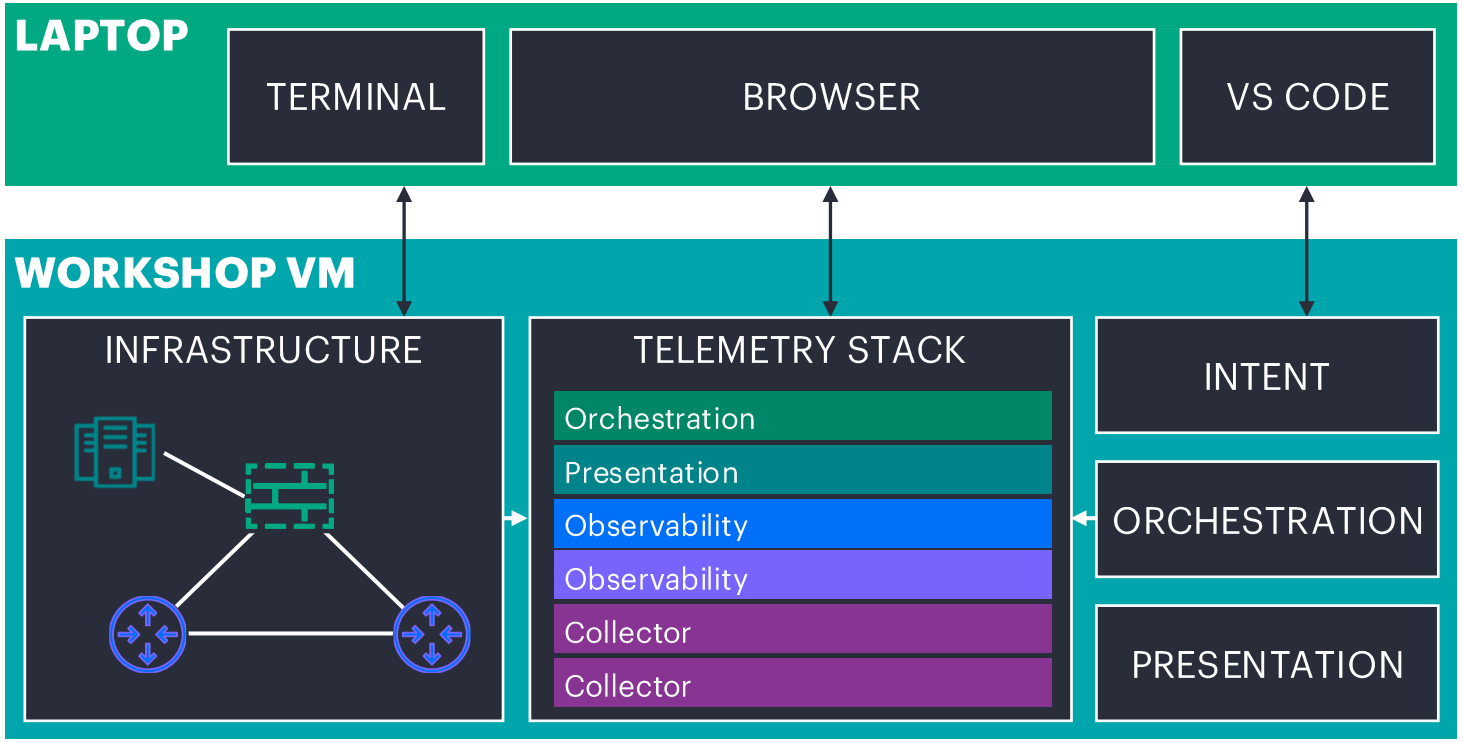
Mapping our products to the Framework

	PRESENTATION	OBSERVABILITY	ORCHESTRATION	INTENT	COLLECTOR	EXECUTOR
Aruba Central* AI-native, self-driving automation	●	●	●	●	●	●
Mist Wireless Assurance Autonomous operations for Wi-Fi	●	●	●	●	●	●
Mist Wired Assurance Agentic AI for enterprise switching	●	●	●	●	●	●
EdgeConnect Orchestrator* SD-WAN policy and monitoring	●	●	●	●	●	●
Mist WAN Assurance Simplifies SD-WAN operations	●	●	●	●	●	●
Security Director* Security management, single UI	●	●	●	●	●	●
ClearPass Role- and device-based access control	●	●	●	●	●	●
Mist Access Assurance Secure NAC, wired and wireless	●	●	●	●	●	●
Data Center Assurance Cloud-hosted, AI-native DC operations	●	●	●	●	●	●
Apstra Data Center Director Design, deploy, operate AI-ready DCs	●	●	●	●	●	●
Juniper Routing Assurance Extends Mist AI to the WAN	●	●	●	●	●	●
Juniper Routing Director End-to-end transport & WAN automation	●	●	●	●	●	●
Juniper Support Insights Automated insights for Juniper devices	●	●	●	●	●	●

Workshop Goals & Use Cases

Goals

- ✓ Understand how network intent and streaming telemetry help detect drift.
- ✓ Follow the closed loop from detection and planning to remediation.
- ✓ Compare human-led, human-approved, and autonomous responses.
- ✓ Explore governance and how NAF blocks can be interchanged.



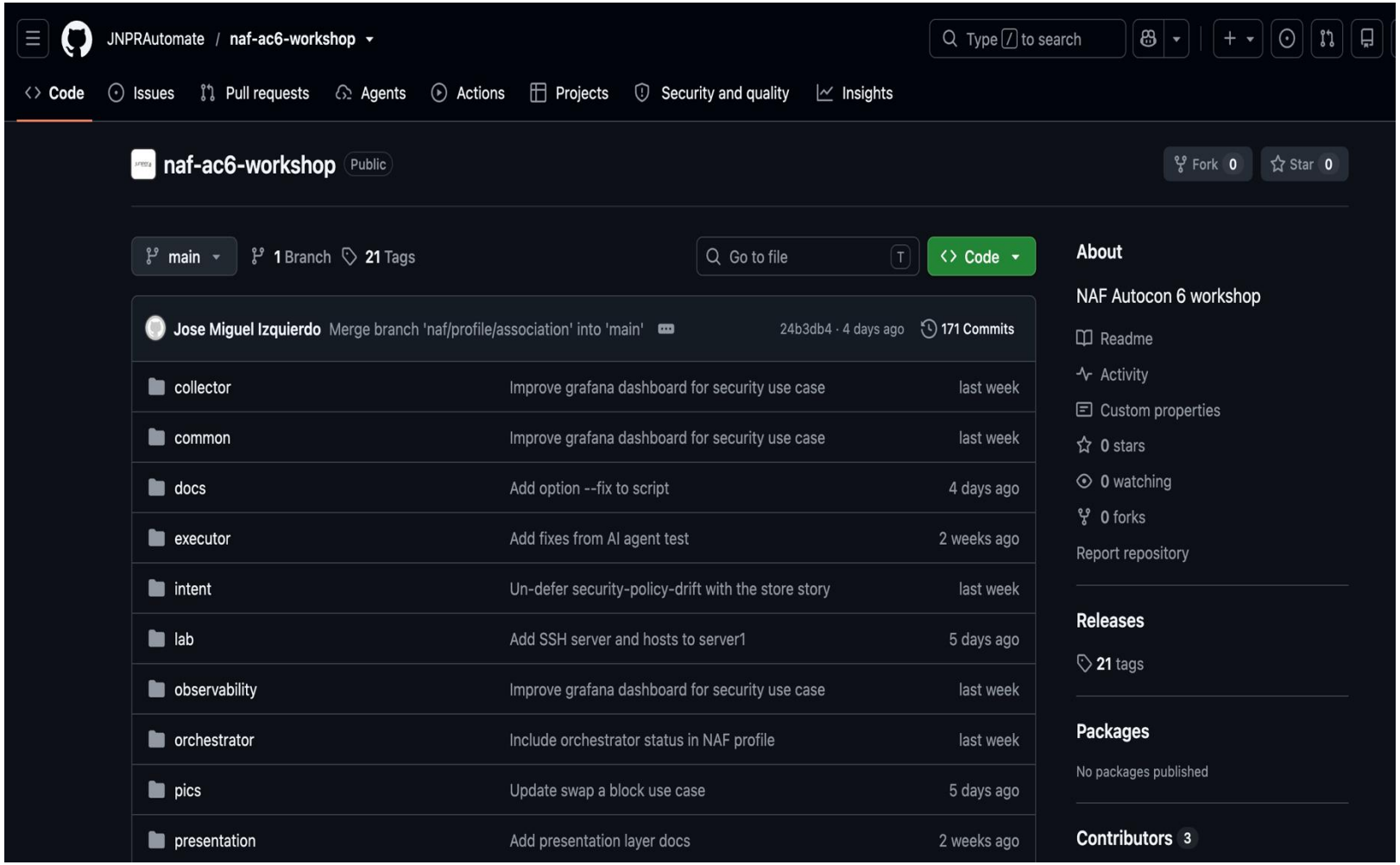
Labs 0-5

- ✓ **Logging Drift**
  Detect and remove leftover debug logging, while respecting an active maintenance window.
- ✓ **BGP Peer Drift**
  Correct a wrong peer AS that has brought down the BGP session between pe1 and pe2.
- ✓ **Prefix Limit**
  Detect a route leak from pe2 and protect pe1 with an import clamp until the leak is fixed.
- ✓ **Security Policy Drift**
  Explore detecting and correcting firewall rule-order drift; this use case is deferred and not runnable in the current lab.
- ✓ **Swap a Block**
  Replay BGP peer drift while swapping Intent, Executor, and Collector implementations.

Workshop available on GitHub

https://github.com/JNPRAutomate/naf-ac6-workshop

- Open-source
- Instructions for both macOS and Windows
- Can be used with JCL



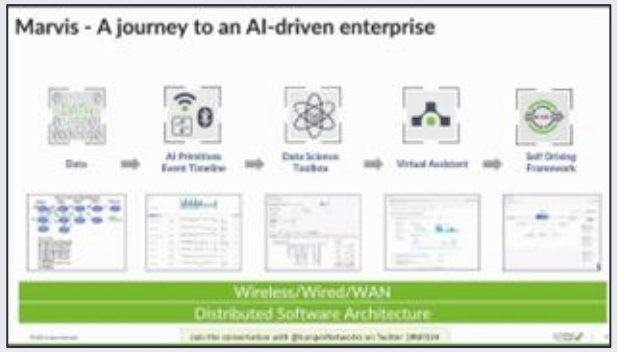
Want to learn more about the self-driving network

Throughout history

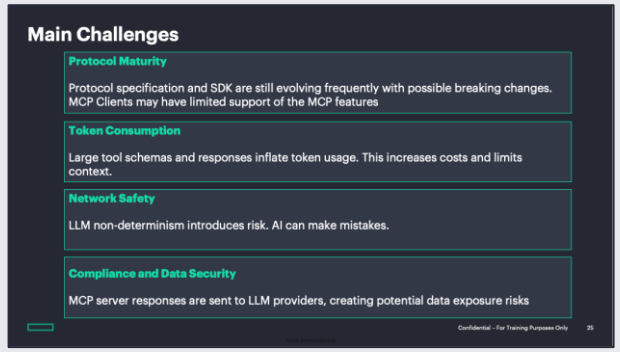
Self-driving @ IETF Kireeti Kompalla (2016)



Juniper Network presents Mist AI @ AI Field Day (2020)



Agentic AI



Content from around the web	Most popular open source from HPE Networking	Virtualized and Containerized Products
NAF Framework	Junos PyEZ & Examples	Free vJunos (Router, Switch, EVO, cEVO)
TM Forum	juniper.device Ansible Collection & Ex	cRPD (Free Trial)
	Junos-MCP	cSRX (Free Trial)
	Terraform for Mist & Apstra	vSRX (Free Trial)

Thank You



© 2026 Hewlett Packard Enterprise Development LP

Troubleshooting Guide I

Problem		
SSH Config (from e-mail)	• SSH key • SSH config (copied, variables, etc…)	➢ ssh naf-ac6-ws-vm ➢ ssh pe1 ➢ ssh pe2 ➢ ssh csrx ➢ ssh server1
VSCode	• Connect to host (naf-ac6-ws-vm) • VSCode extensions	• ContainerLab • Markdown Preview Enhanced • Markdown All in One • Auto-Open Markdown Preview
naf-setup (alias)	Runs ~200 tests to verify if your environment is OK	
naf-env (alias)	Creates the virtual environment (.venv)	All SW requirements are in virtual environment. Make sure you run VM commands from prompt containing (.venv)
Better to be at repo dir	Above alias take you there, else: *goworkshop*	Safer to even do *naf-env*
Container Lab	Make sure topology is *up* and *running*	> clab inspect –a (you can check also VSCode plugin)
openJTS	Check if containers are *up* and *running* Check logs	> docker ps > tail –f /var/log/jtso.log
Empty family/model/version	NETCONF login failed	Wrong IP

Troubleshooting Guide II

Problem		
URLs	Grafana: http://localhost:18080/ Notify (ntfy): http://localhost:18095/ OpenJTS: http://localhost:18090/	
reset-lab (alias)	Restores Container Lab topology to known state	
Telemetry	Check gNMI telemetry (command at repo)	gnmic -a 172.20.20.11:9339 --insecure -u lab -p lab123 -e json subscribe \ --mode once --format flat --path "/network-instances/network-instance/protocols/protocol/bgp/neighbors"
Intent	*show-intent* (alias)	print the current intent as YAML
Seeding intent	*seed-intent (alias)*	*copy intent into InfluxDB for Kapacitor and Grafana*
check_vm (alias)	Verifies a few health checks specific for a running repo	Different options (--fix, --no-fix) to help you fix issues it finds
Orchestrator	Orchestrator status (UP/DOWN)	*> /home/hpen/naf-ac6-workshop/scripts/orchestrator.start.sh* *> /home/hpen/naf-ac6-workshop/scripts/orchestrator.stop.sh* *> /home/hpen/naf-ac6-workshop/scripts/orchestrator.status.sh* *> ohealth (alias)*