Building Network Visualizers

Lucas Voron

July 29th 2026

Agenda

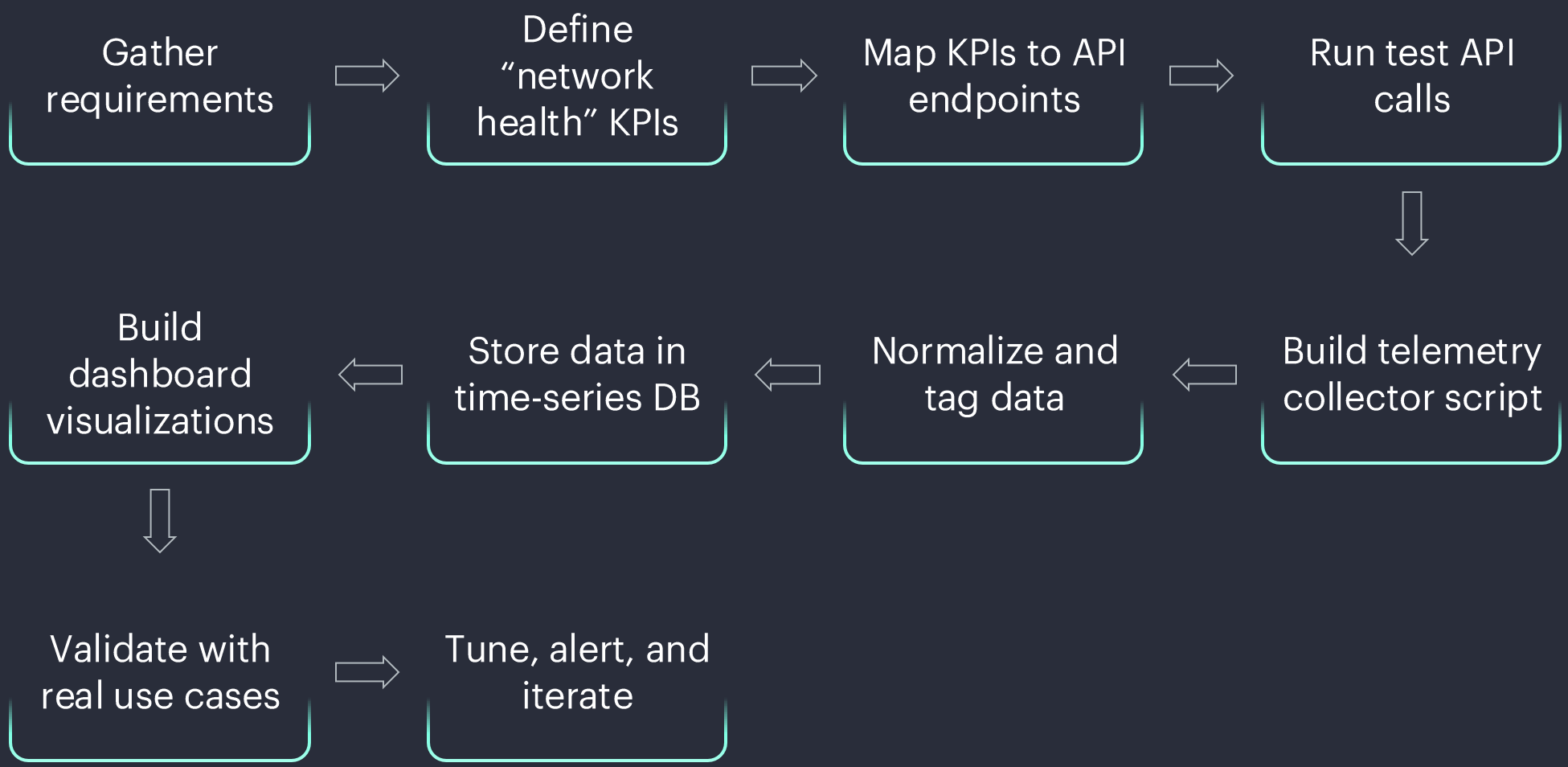
HPE DEV Meetup Session

01	Importance of API in Visualizers
02	Build Process Breakdown
03	API Discovery & Testing
04	Defining Network Health
05	Live API Monitoring Calls
06	Storage Considerations
07	Graphical Recommendations
08	Scale, Multi-threading, & Rate Limits
09	Topology Considerations
10	Next Steps, Demo, & Resources

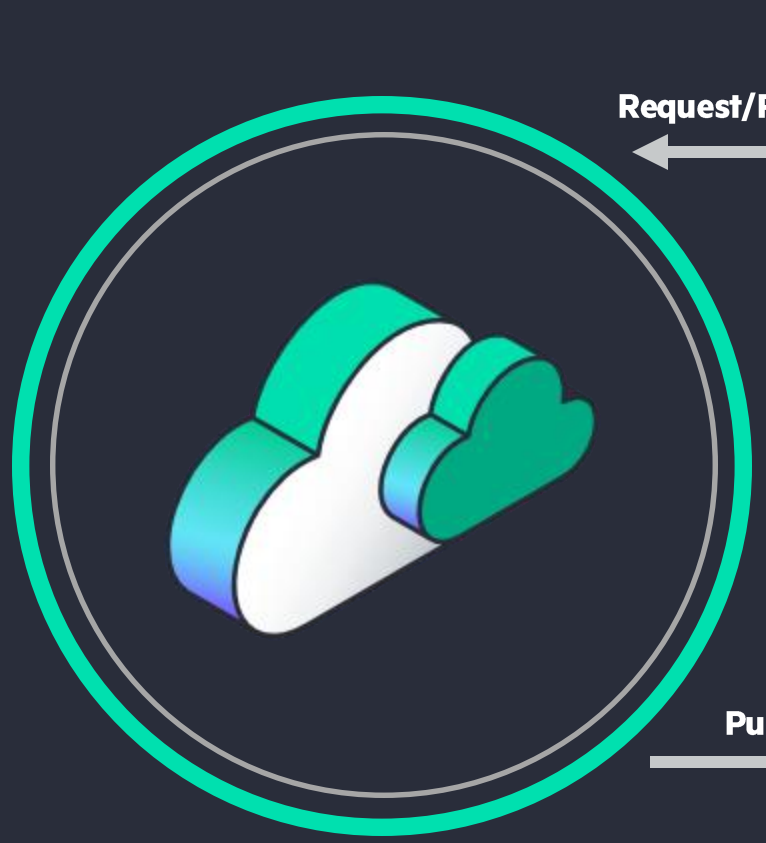
Importance of API in Visualizers

- No one-size-fits-all monitoring
- Match signals to your environment
- One question per visualizer
- Reduce outage guesswork quickly

Build Process Breakdown



Central Automation Capabilities



REST API

- HTTP interface for monitoring, reporting, troubleshooting, and configuration endpoints
- Supports GET, POST, PUT, PATCH, and DELETE operations

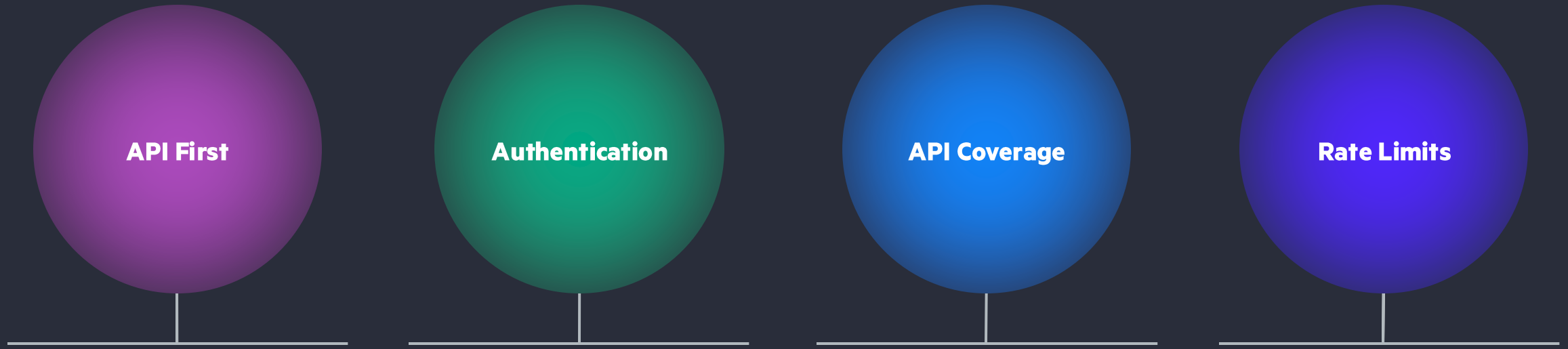
Webhooks

- Push-based notifications that send alerts and events to a URL of your choice in real time
- Useful for triggering workflows when device status changes, alerts fire, or clients connect

Streaming API

- Continuous data stream of telemetry, events, and state changes over a set connection
- Lower latency than polling REST endpoints for high-frequency monitoring use cases

Central REST APIs



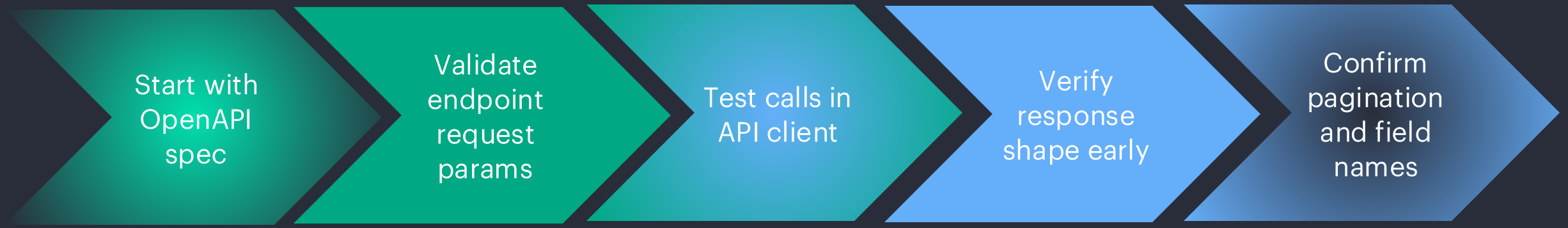
- Standardized Open API standard compliant API endpoints
- 100% API coverage

- OAuth 2.0 and JWT-based access tokens
- GreenLake-generated tokens are honored by Central
- Unified token management across GreenLake and Central workflows

- Monitoring, reporting, and troubleshooting use cases
- AP, Switch, and Gateway details

- Per-second rate limit is 10 API calls/second
- No per-day rate limit.

API Discovery & Testing



Official Postman Collection

Official Postman Workspace containing Central REST API collections for Monitoring

Defining Network Health

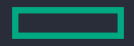
- Choose pain-driven health signals
- Track latency, loss, utilization, uptime
- Define Good, Fair, Poor
- Tag data for filtering
- Define unhealthy in context

PyCentral-v2

Official Python library that wraps the Central REST API into importable classes and methods

Developer Hub

Walkthrough



Storage Considerations

How will you store data attributes?

01

Group metrics by measurement families

02

Use tags for dimensions

03

Use fields for values

04

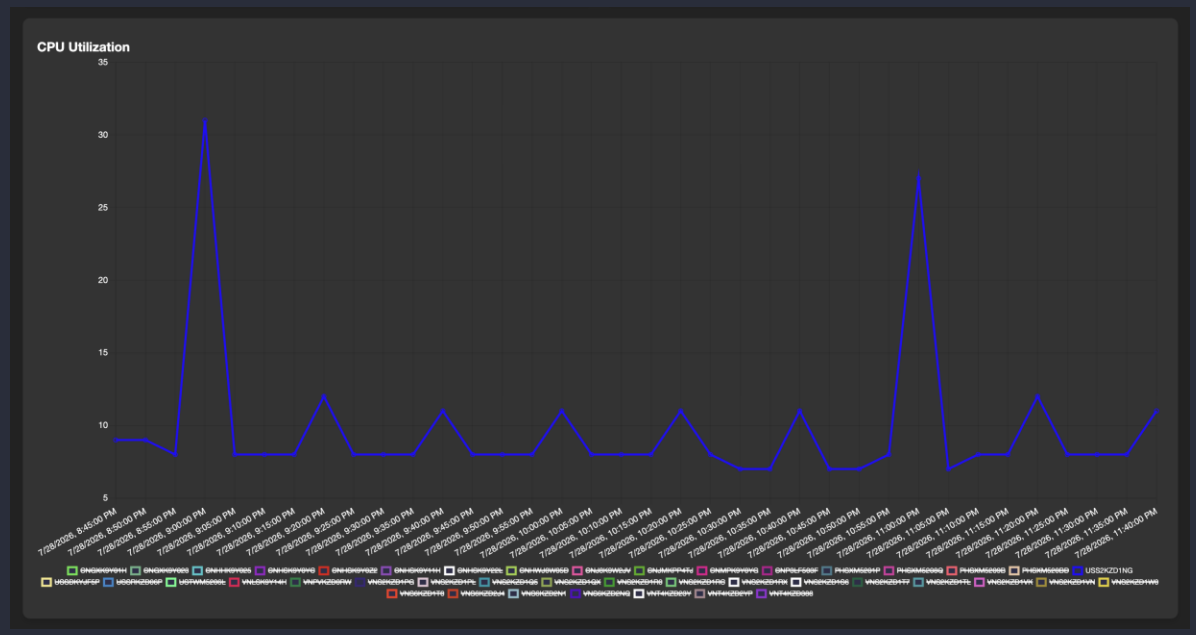
Keep timestamps fully consistent

05

Plan retention before scaling

Graphical Recommendations

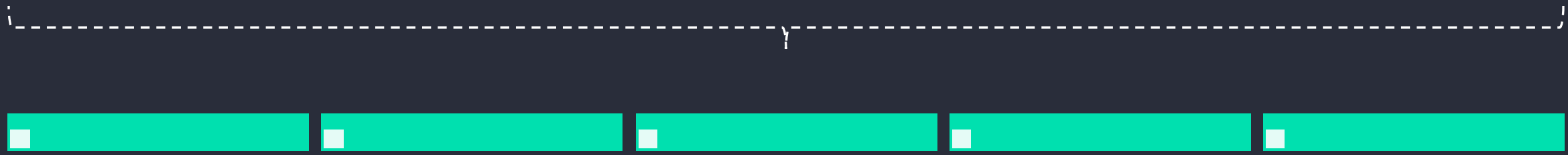
- Use lines for trends
- Use stacks for composition
- Use heatmaps for concentration
- Annotate incidents and changes
- Align times across panels



Scale, Multi-threading, Rate Limits

Network Setup
Scale of devices & number of calls

API Calling Constraints
Number of calls made per second



Use bounded worker pools

Batch writes to storage

Backoff retries on failures

Process sites, WLANs, devices, radios

Use cursor-based pagination

Timeseries Databases

InfluxDB

- metric-heavy ingest
- fast database write

TimescaleDB

- SQL analytics
- relational statistics reporting

Prometheus

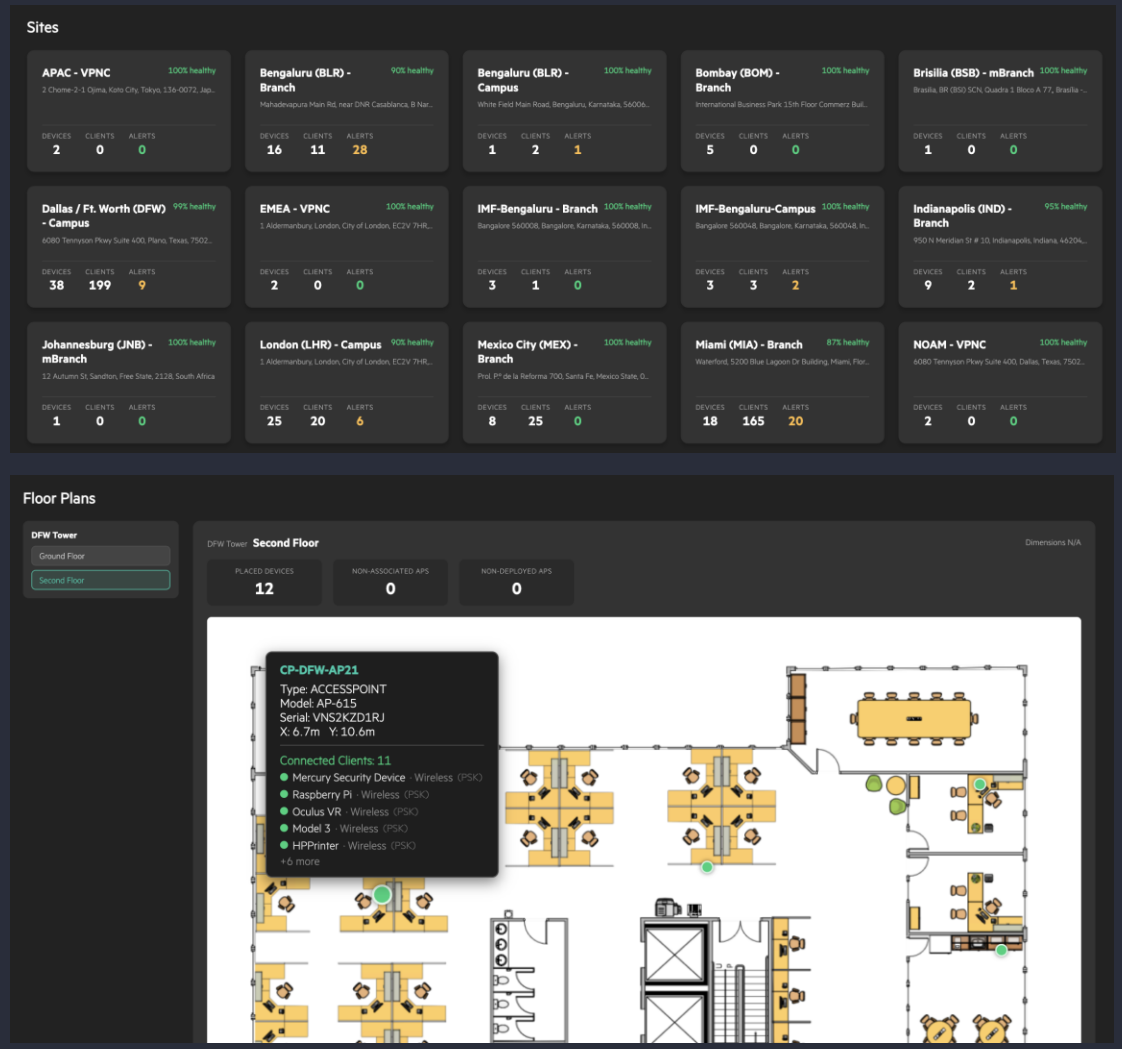
- polling-based workflows
- alerting first

VictoriaMetrics

- long term storage
- high-volume

Topology Considerations

- Model topology operators understand
- Design for roaming clients
- Split views by device role
- Add floor and location context
- Avoid dashboard-specific schema bias



Tech Stack

Site & Device Level Details

API First

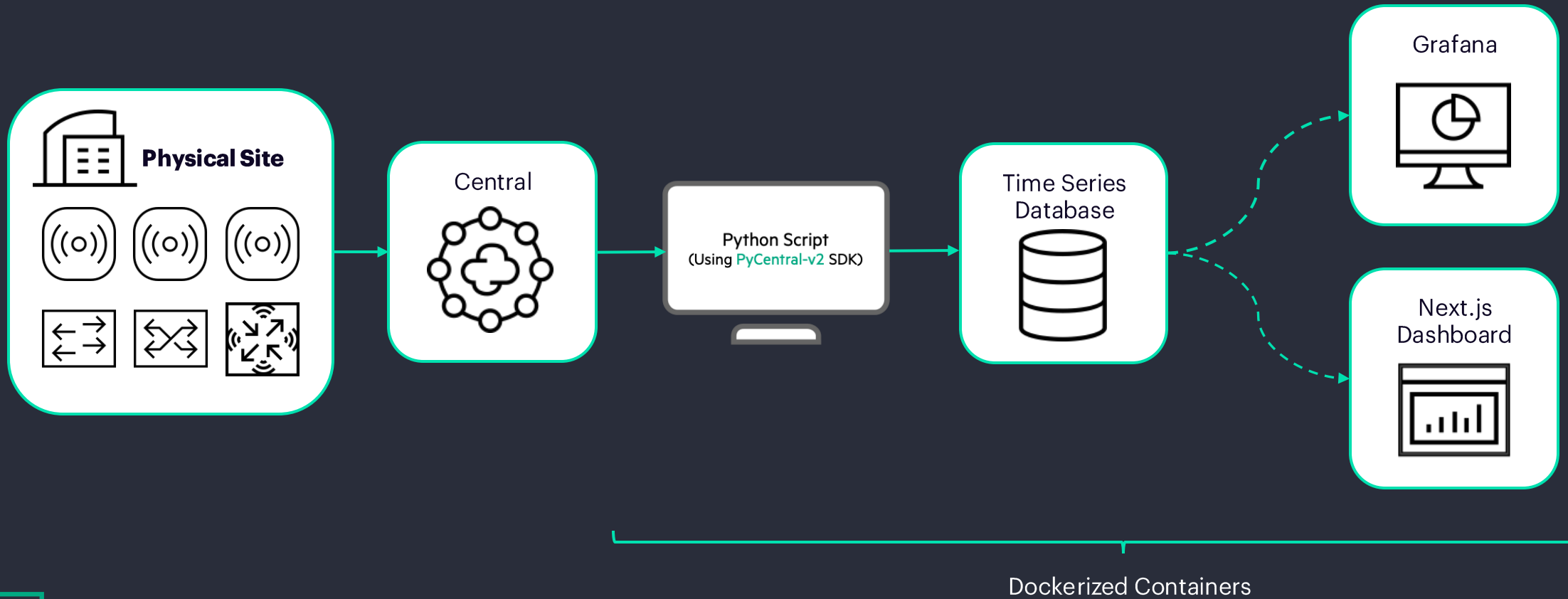
Code Making API Calls

Data Storage

Custom monitoring dashboards

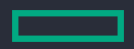
Real-time insights

Enhanced troubleshooting



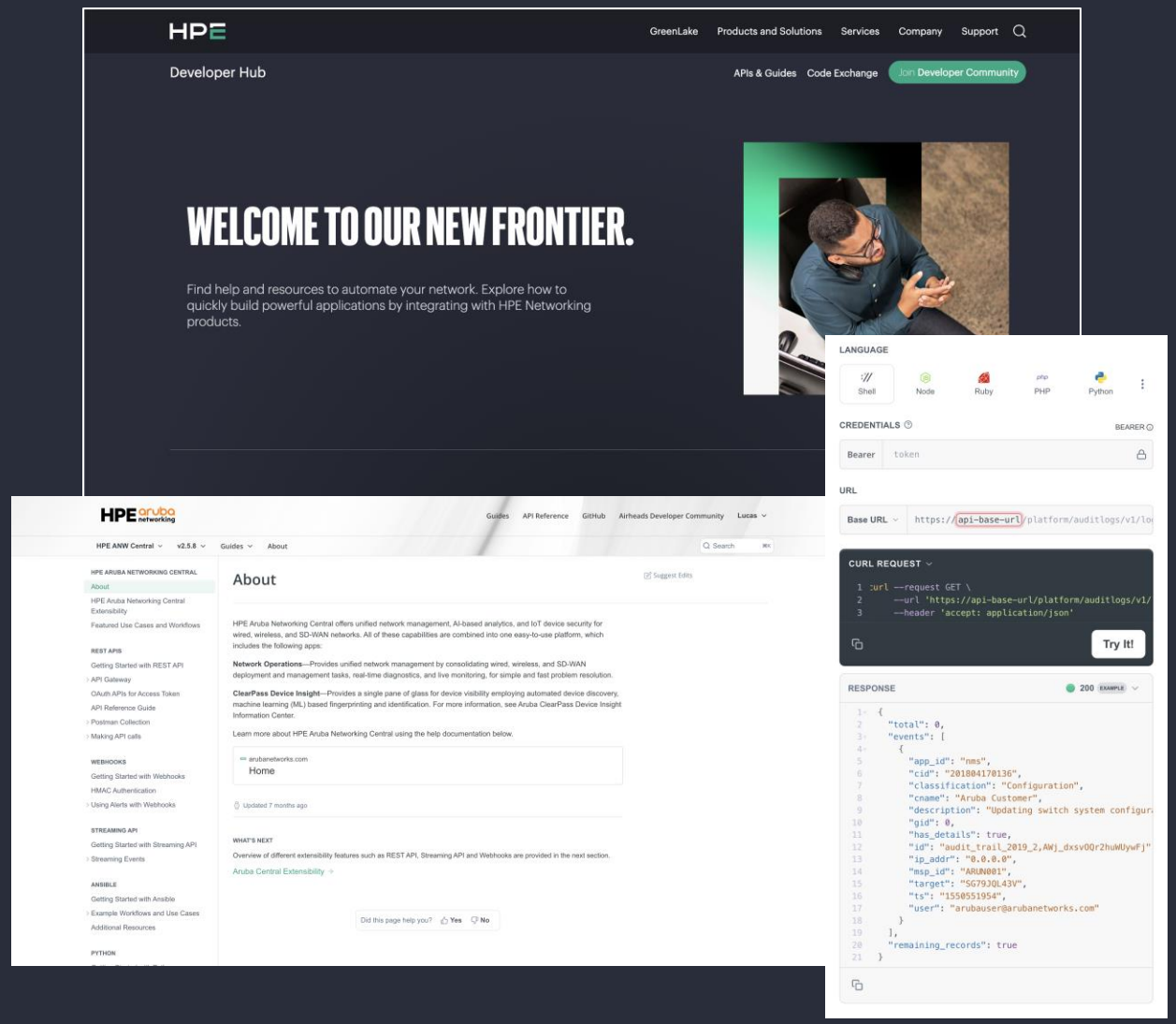
Dashboard Demo

Live



Next Steps

- This dashboard is available right now!
- Open Source Project
- Setup guide available
- Guide Online



Additional resources

HPE NW Developer Hub

HPE NW GitHub

Airheads Developer Community

Ansible Galaxy

Email us at network-automation@hpe.com

Join our Slack Channels in the HPE Dev Workspace

- #aruba-central
- #aruba-clearpass
- #aruba-switching
- #aruba-wireless
- #aruba-edgeconnect-sdwan
- #aruba-axis-sse

Thank You