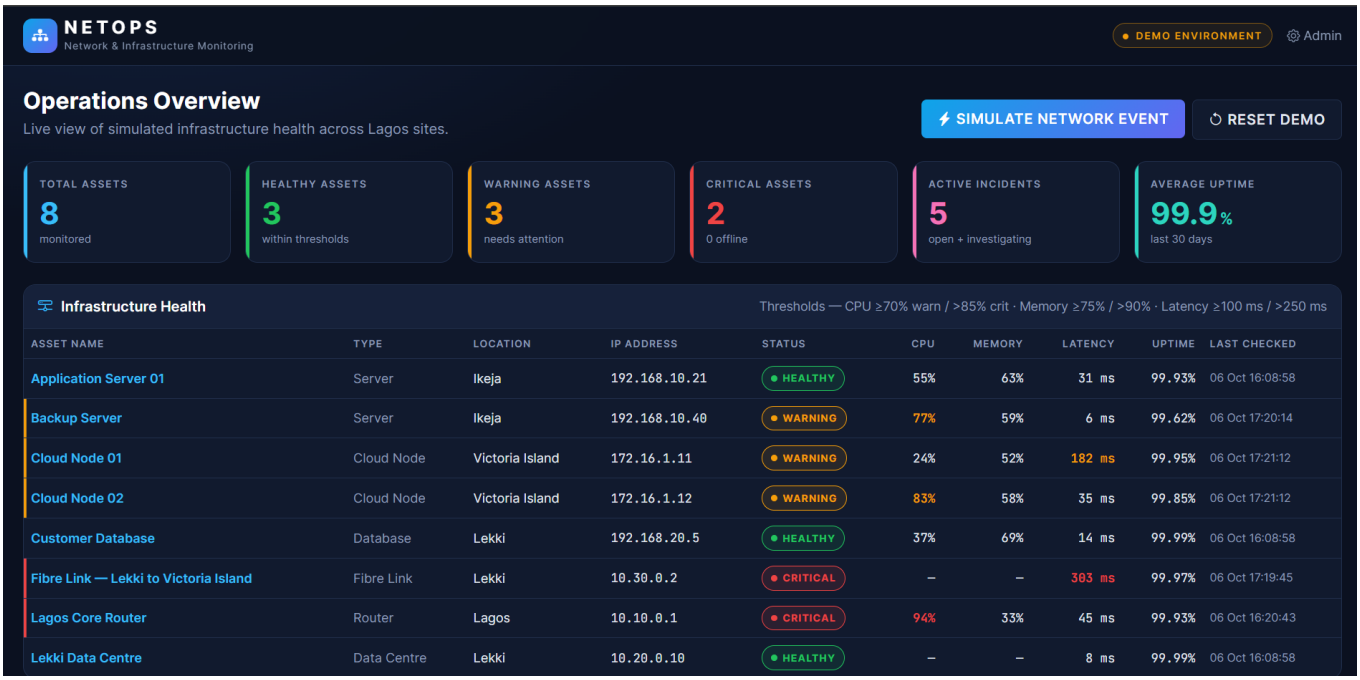
SOFTWARE ENGINEERING PORTFOLIO

NetOps

Network & Infrastructure Monitoring Dashboard

Project documentation | October 2026

A Python and Django application that demonstrates how a Network Operations Centre (NOC) could monitor infrastructure health, detect threshold breaches, and manage incidents from one dashboard.



Purpose and scope

NetOps brings asset information, performance metrics and incident handling together in a professional dark dashboard. It uses fictional servers, routers, cloud nodes, a fibre link, a database and a data-centre asset across Lagos locations.

Demo environment: all assets, IP addresses, metrics and incidents are fictional. The application does not connect to, scan or ping real networks. A cloud node represents a simulated cloud computing resource, such as a virtual server; it is not an actual AWS connection.

What the application demonstrates

Persistent asset and incident records; health classification; automatic incident creation when checks run; duplicate prevention for active incidents; rule-based recommendations; investigation and resolution; simulated events; reset controls; charts; asset details and Django Admin.

Guide contents: architecture and data (page 2), monitoring rules (page 3), workflows (page 4), setup and demonstration (page 5), and limitations and future development (page 6).

Architecture and data

The browser displays the interface. Django handles requests and application logic, then reads or updates records in SQLite. Templates turn the resulting data into HTML; Bootstrap styles it and Chart.js draws the charts.

Technology	Role in NetOps
Python	Implements classification, simulated metrics, incident checks and recommendations.
Django	Provides models, views, URLs, templates, database access, migrations and Admin.
SQLite	Stores assets and incidents persistently in a local database file.
HTML and CSS	Define page structure and visual presentation.
Bootstrap 5	Provides layout and interface components for the dashboard.
JavaScript and Chart.js	Support browser behaviour and visualise asset latency, health distribution and CPU usage.

Asset record

An asset stores its name, type, location, IP address, CPU usage, memory usage, latency, uptime percentage, reachability, health status and last-checked time. CPU and memory may be absent when they do not apply, for example to the fibre link.

Incident record

Each incident links to an asset and stores the triggering metric, severity, title, description, recommended action, lifecycle status and timestamps. Metrics are CPU, memory, latency or offline. Statuses are Open, Investigating and Resolved.

Project organisation

netops/: configuration and main routing.
monitoring/: models, services, views, URLs, Admin, tests, migrations and seed command.
templates/: dashboard and asset detail pages.
static/: styles, scripts and locally bundled frontend libraries.

Frontend libraries are bundled locally so the demo can operate offline after its Python dependencies have been installed.

Monitoring rules and recommendations

The threshold engine is ordinary Python logic. It checks each applicable metric against fixed boundaries. The most severe result determines the asset health. Unreachable assets are Offline and generate a Critical offline incident.

Metric	Healthy	Warning	Critical
CPU usage	Below 70%	70% to 85%	Above 85%
Memory usage	Below 75%	75% to 90%	Above 90%
Latency	Below 100 ms	100 to 250 ms	Above 250 ms

Understanding the metrics

CPU: how much processing capacity is in use. **Memory:** how much RAM is in use. **Latency:** the simulated response delay in milliseconds; 22 ms means 0.022 seconds and is Healthy under this project's rules. **Uptime:** a simulated availability percentage labelled for the last 30 days, not a measurement from collected history.

Exact rule-based recommended actions

Trigger	Recommendation displayed
High CPU	Review high-utilisation processes and available compute capacity. Consider scaling resources if utilisation remains elevated.
High memory	Review memory-intensive processes and available memory capacity. Check for memory leaks and consider adding capacity if usage stays high.
High latency	Check network connectivity, packet loss, routing, and upstream link health.
Offline asset	Verify device connectivity and power status. Escalate to the infrastructure team if the asset remains unreachable.

Recommendations are selected from predefined text by the triggering metric. They are the same for Warning and Critical within that metric; they are not generated by AI and do not perform repairs.

Uptime has no threshold rule or incident recommendation in the current implementation. Low memory usage also does not trigger an incident. Missing CPU or memory values do not create breaches.

Application workflows

Simulate Network Event: end-to-end flow

1. The user clicks the button. The browser submits a POST request to Django.

2. The simulation service selects one or two fictional assets. The first receives a Warning, Critical or Offline event; an optional second can receive an event or recovery.

3. The service changes the selected metrics or reachability and saves the changes in SQLite.

4. The threshold engine classifies the new values and updates asset health.

5. For each detected breach, the service creates an incident or updates an existing active incident for the same asset and metric.

6. The dashboard reloads with updated database-backed KPI cards, asset rows, incidents and charts.

Duplicate prevention and severity escalation

The service looks for an Open or Investigating incident with the same asset and metric. Repeated checks reuse that incident. If a Warning becomes Critical, it escalates the existing incident. This application-level check is not a database guarantee against simultaneous requests.

Investigate and resolve

Mark as Investigating changes an active incident's status. **Resolve** records it as Resolved and sets its resolution timestamp. Resolution does not repair the asset or reset its metrics. If the breach persists, a later check can create a new incident. Metric recovery does not automatically resolve an existing incident.

Reset Demo

Reset deletes all incidents and non-demo assets, restores the eight baseline fictional assets and runs threshold checks. The baseline contains six Healthy assets, two Warning assets and two active incidents. Use reset only when you want to restore the demonstration dataset.

Dashboard and details

KPI cards summarise current records, including healthy/warning/critical assets, active incidents and average simulated uptime. Asset names open detail pages. The dashboard displays database state; opening a page does not independently poll infrastructure or rerun threshold checks.

Setup and demonstration

Requirements: Python 3.9 or newer and the dependencies in requirements.txt. Django is constrained to >=4.2,<5.3; install a release compatible with your Python version.

Windows quick start

From the project folder, run **setup_netops.bat** to create the environment, install dependencies, migrate, seed data and run tests. Run **start_netops.bat** to start the local demo. Check the setup output or log for errors before demonstrating.

Manual setup

```
python -m venv venv
.\venv\Scripts\Activate.ps1
pip install -r requirements.txt
python manage.py migrate
python manage.py seed_demo
python manage.py runserver
```

For macOS/Linux, activate with **source venv/bin/activate** instead. Open **http://127.0.0.1:8000/** in the browser. Stop the server with Ctrl+C.

Optional Admin account: **python manage.py createsuperuser**, then visit **/admin/**. Run the test suite with **python manage.py test monitoring**. Restore the baseline from the command line with **python manage.py seed_demo --reset**.

Suggested interview demonstration

1. Explain that all data is simulated and identify the KPI cards and asset metrics.

2. Open an asset detail page and explain one health classification.

3. Click Simulate Network Event and trace the changed metrics to an incident and recommendation.

4. Mark an incident as Investigating, then Resolve it; explain that resolution records handling, rather than fixing the resource.

5. Reset the demo and show the baseline.

Source code

github.com/Ernest-Cyril/netops-monitoring-dashboard

Design decisions and practical limits

Why this stack

Python keeps the monitoring logic readable. Django provides a complete web framework and Admin. SQLite makes a local portfolio demonstration easy to set up. Bootstrap and Chart.js provide a clear interface without requiring a large frontend framework.

Validation provided in the repository

The test suite covers threshold classification, incident creation, duplicate handling, escalation, incident lifecycle, demo seeding/reset and dashboard simulation. This documentation was prepared by reviewing the source; it does not claim a fresh test run or production performance validation.

Current limitations

There is no connection to real infrastructure, continuous background monitoring, historical telemetry pipeline or automatic remediation. Uptime values are synthetic. App actions are designed for a local demo, without a complete role-based access model. SQLite and application-level duplicate checks are suitable for the prototype; production concurrency would need stronger controls.

Possible future development

With appropriate access and authorisation: integrate real telemetry APIs or monitoring platforms, schedule checks, retain metric history, add notifications and role-based permissions, introduce database-level safeguards, and deploy with production configuration. These are future ideas, not existing features.

Interview-ready explanation

What did you build?
"NetOps is a Django web application that simulates infrastructure monitoring. It stores assets and incidents in SQLite, classifies CPU, memory and latency against thresholds, and lets a user investigate and resolve incidents."

How are incidents and recommendations created?
"When a simulated event changes an asset, my Python service checks the values. A breach creates or escalates an active incident. The service chooses a predefined recommendation based on the metric, so the recommendations are rule-based."

Does it monitor real networks?
"This version uses fictional data to demonstrate the workflow. Real telemetry and scheduled monitoring would be separate future integrations."

Documentation basis: monitoring models, services, views, seed command, tests, requirements, startup scripts and README in the NetOps project.