CDW Documentation

DGX Spark Monitoring Stack

DGX Spark Monitoring Stack

Overview

- **nv-monitor:** single C binary that monitors CPU, GPU, memory and exposes Prometheus metrics
- **Prometheus:** scrapes and stores metrics from nv-monitor every 5 seconds
- **Grafana:** visualizes metrics in a live dashboard
- **demo-load:** synthetic CPU + GPU load generator for testing the pipeline
- Everything runs on the DGX Spark (spark02) — Prometheus and Grafana run in Docker containers

Step 1 — SSH into the DGX Spark

From your Mac terminal:

```
ssh swilson@100.91.118.118
```

All steps below are run on the Spark unless noted otherwise.

[SCREENSHOT: Mac terminal showing swilson@spark02:~$ prompt after successful SSH login]

Step 2 — Install Build Dependencies

```
sudo apt install build-essential libncurses-dev -y
```

- **build-essential:** gcc, make, and standard C libraries
- **libncurses-dev:** required for the terminal UI (ncursesw wide character support)

Both packages were already installed on spark02 (12.10ubuntu1 and 6.4+20240113). If already installed, apt reports “already the newest version” and exits cleanly.

Step 3 — Clone and Build nv-monitor

```
cd ~
git clone https://github.com/wentbackward/nv-monitor
cd nv-monitor
make
```

If the repo already exists from a previous run, git will print “destination path 'nv-monitor' already exists” and make will print “Nothing to be done for 'all'” — both are fine, the binary is already built.

Verify it works by launching the interactive TUI:

```
./nv-monitor
```

Press **q** to quit.

What the TUI shows

- **CPU section:** per-core usage bars, ARM core type labels (X925 = performance, X725 = efficiency)
- **GPU section:** utilization, temperature, power draw, clock speed
- **Memory section:** used, buf/cache, swap
- **VRAM:** shows “unified memory (shared with CPU)” on GB10 — nvmlDeviceGetMemoryInfo returns NOT_SUPPORTED, which is expected
- **History chart:** rolling 20-sample graph of CPU (green) and GPU (cyan)

[SCREENSHOT: nv-monitor TUI showing all 20 cores (0-9 X725 efficiency, 10-19 X925 performance), GPU 0 NVIDIA GB10 at 42C 4.7W 208MHz, MEM 5.4G used / 121.7G, unified memory label, uptime 11d 17h]

Step 4 — Run nv-monitor as a Prometheus Exporter

Start nv-monitor in headless mode with a Bearer token:

```
cd ~/nv-monitor
./nv-monitor -n -p 9101 -t my-secret-token &
```

Flags explained

- **-n:** headless mode — no TUI, runs silently in background
- **-p 9101:** expose Prometheus metrics endpoint on port 9101
- **-t my-secret-token:** require this Bearer token on every request
- **&:** run in background so the terminal stays free

On startup it prints:

```
Prometheus metrics at http://0.0.0.0:9101/metrics
Running headless (Ctrl+C to stop)
```

Verify it is working:

```
curl -s -H "Authorization: Bearer my-secret-token" localhost:9101/metrics |
head -10
```

You should see output starting with `# HELP nv_build_info`.

[SCREENSHOT: Terminal showing nv-monitor background process (PID 52653) and curl output with # HELP nv_build_info, nv_uptime_seconds, nv_load_average metrics]

Available nv-monitor metrics

- `nv_cpu_usage_percent` — per-core CPU usage
- `nv_cpu_temperature_celsius` — CPU temperature
- `nv_gpu_utilization_percent` — GPU utilization
- `nv_gpu_power_watts` — GPU power draw in watts
- `nv_gpu_temperature_celsius` — GPU temperature
- `nv_memory_used_bytes` — RAM used in bytes
- `nv_load_average` — system load average (1m, 5m, 15m)
- `nv_uptime_seconds` — system uptime

Step 5 — Create the Prometheus Configuration

```
mkdir ~/monitoring
```

```
cat > ~/monitoring/prometheus.yml << 'EOF'
global:
  scrape_interval: 5s
```

```
scrape_configs:
  - job_name: 'nv-monitor'
    authorization:
      credentials: 'my-secret-token'
    static_configs:
      - targets: ['172.17.0.1:9101']
EOF
```

Why 172.17.0.1 and not localhost?

- Docker containers have their own network namespace
- `localhost` inside a container refers to the container itself, not the host machine
- `172.17.0.1` is the Docker bridge gateway — the IP containers use to reach the host
- Find it with: `docker network inspect bridge | grep Gateway`

Step 6 — Start Prometheus and Grafana in Docker

```
docker run -d \
  --name prometheus \
  -p 9090:9090 \
  -v ~/monitoring/prometheus.yml:/etc/prometheus/prometheus.yml \
  prom/prometheus
```

```
docker run -d \
  --name grafana \
  -p 3000:3000 \
  grafana/grafana
```

Connect both containers to a shared Docker network so Grafana can reach Prometheus by name:

```
docker network create monitoring
docker network connect monitoring prometheus
docker network connect monitoring grafana
```

Verify both are healthy:

```
docker ps
curl -s localhost:9090/-/healthy
curl -s localhost:3000/api/health
```

Expected responses:

- `Prometheus Server is Healthy.`
- `{“database”:“ok”,“version”:“12.4.2”,…}`

Step 7 — Allow Docker Bridge to Reach nv-monitor

Docker containers live in the `172.17.x.x` subnet. The firewall must allow them to reach port 9101 on the host.

Note: spark02 does not have UFW installed. Use iptables directly:

```
sudo iptables -I INPUT -s 172.17.0.0/16 -p tcp --dport 9101 -j ACCEPT
```

This is the critical rule that allows Prometheus to scrape nv-monitor.

SUDO POLICY VIOLATION broadcast messages

spark02 has a sysadmin audit policy that broadcasts a message to all terminals when sudo is used. The command still executes — the broadcast is just a notification to the admin team. It is not an error.

Step 8 — Access UIs from Your Mac via SSH Tunnel

SSH port forwarding is simpler, more secure, and works over Tailscale without opening firewall ports.

On your **Mac**, open a **new local terminal** (not an SSH session — the prompt must show your Mac hostname, not spark02):

```
ssh -L 9090:localhost:9090 -L 3000:localhost:3000 swilson@100.91.118.118
```

Keep this terminal open. Then open in your Mac browser:

- **Prometheus:** http://localhost:9090/targets
- **Grafana:** http://localhost:3000

Common mistake — running the tunnel from inside the Spark

If you run the SSH tunnel command from a terminal that is already SSH'd into spark02, it will SSH back to itself and fail with “Address already in use” because ports 9090 and 3000 are already bound by the Docker containers on the Spark. Always run the tunnel from a Mac local terminal.

Why SSH tunneling?

- Works over Tailscale without needing to open firewall ports
- Encrypted by default — no plaintext traffic over the network
- No additional firewall rules needed
- Easy to disconnect by closing the terminal

Step 9 — Verify Prometheus is Scraping

Open `http://localhost:9090/targets` in your browser.

You should see the **nv-monitor** job with state **UP** and a scrape duration of ~2ms.

[SCREENSHOT: Prometheus Target health page showing nv-monitor job, endpoint http://172.17.0.1:9101/metrics, state UP (green), last scrape 11s ago, duration 2ms]

Step 10 — Configure Grafana

Open `http://localhost:3000` in your browser.

- Login: **admin** / **admin**
- Change password when prompted

Add Prometheus as a data source

1. Click **Connections** in the left sidebar
2. Click **Data sources**
3. Click **Add data source**
4. Select **Prometheus**
5. Set URL to: `http://prometheus:9090`
6. Click **Save & test**
7. You should see: **Successfully queried the Prometheus API**

[SCREENSHOT: Grafana data source config page showing URL http://prometheus:9090 and green “Successfully queried the Prometheus API” confirmation banner]

Why ''http://prometheus:9090'' works

Both containers are on the same Docker network (`monitoring`). Docker provides DNS resolution between containers on the same network, so `prometheus` resolves to the Prometheus container's IP automatically.

Step 11 — Build the Dashboard

1. Click **Dashboards → New → New dashboard → + Add visualization**
2. Add each panel below one at a time

3. For each panel: select metric in Builder, set title in right panel options, confirm Time series visualization, click Back to dashboard

Dashboard panels

- **CPU Usage %** — metric: `nv_cpu_usage_percent` — type: Time series
- **CPU Temperature** — metric: `nv_cpu_temperature_celsius` — type: Time series
- **GPU Utilization %** — metric: `nv_gpu_utilization_percent` — type: Time series
- **GPU Power (W)** — metric: `nv_gpu_power_watts` — type: Time series
- **GPU Temperature** — metric: `nv_gpu_temperature_celsius` — type: Time series
- **Memory Used** — metric: `nv_memory_used_bytes` — type: Gauge — unit: bytes (SI)

Save the dashboard as **DGX Spark Monitor**. Set auto-refresh to **10s**.

Important: always use prometheus-10 as the data source

When adding panels, make sure the Data source dropdown shows **prometheus-10** (the data source you configured), not the default “prometheus” placeholder. If a panel shows “No data”, check the data source selection first.

Panel shows No data

Switch to **Last 5 minutes** time range and click **Run queries**. If still no data, click **Code** in the query editor and type the metric name directly (e.g. `nv_gpu_utilization_percent`), then run queries. The GPU utilization panel will show a flat 0% line at idle — that is correct, not an error.

Step 12 — Load Test with demo-load

demo-load is included in the nv-monitor repo and built by default with `make`.

```
cd ~/nv-monitor
./demo-load --gpu
```

Output:

```
Starting CPU load on 20 cores (sinusoidal, phased)
Starting GPU load on 1 GPU (sinusoidal)
Will stop in 5m 0s (Ctrl+C to stop early)
GPU 0: calibrating... done (kernel=0.01ms, blocks=1024)
GPU 0: load active
```

This generates sinusoidal CPU and GPU load simultaneously for 5 minutes. Watch the Grafana dashboard for live spikes.

[SCREENSHOT: demo-load terminal output showing GPU 0: load active]

[SCREENSHOT: Grafana dashboard with all 4 panels spiking — GPU Power jumping from 4.5W to 12W, CPU Usage % hitting 80-100% across all cores, GPU Utilization spiking, CPU Temperature climbing from 45C to 70C+]

Press **Ctrl+C** to stop early, or wait 5 minutes for it to finish automatically.

Reconnecting After Reboot or Disconnect

nv-monitor and Docker containers do not auto-restart. To bring everything back:

On spark02:

```
cd ~/nv-monitor
./nv-monitor -n -p 9101 -t my-secret-token &
docker start prometheus grafana
```

On your Mac (new local terminal):

```
ssh -L 9090:localhost:9090 -L 3000:localhost:3000 swilson@100.91.118.118
```

Then open http://localhost:3000.

Tearing Everything Down

```
docker stop prometheus grafana
docker rm prometheus grafana
docker network rm monitoring
pkill nv-monitor
rm -rf ~/monitoring
```

Troubleshooting

nv-monitor binary does not exist after git clone

A file named `nv-monitor` already existed in the home directory (bad previous download).

```
rm nv-monitor
git clone https://github.com/wentbackward/nv-monitor
cd nv-monitor
make
```

Prometheus target shows DOWN — context deadline exceeded

Two causes, apply both fixes:

Fix 1 — Use the correct target IP in `prometheus.yml`:

```
targets: ['172.17.0.1:9101']
```

Then restart: `docker restart prometheus`

Fix 2 — Allow Docker bridge through the firewall (spark02 uses iptables, not UFW):

```
sudo iptables -I INPUT -s 172.17.0.0/16 -p tcp --dport 9101 -j ACCEPT
```

UFW command not found

spark02 does not have UFW installed. Use iptables directly (see Step 7).

SSH tunnel fails with "Address already in use"

You ran the tunnel command from inside an existing SSH session to spark02 instead of from a Mac local terminal. Open a new terminal on your Mac (prompt should show your Mac hostname) and run the tunnel command from there.

Grafana cannot connect to Prometheus — lookup prometheus: no such host

Containers are not on the same Docker network.

```
docker network create monitoring
docker network connect monitoring prometheus
docker network connect monitoring grafana
```

Then set Grafana data source URL to `http://prometheus:9090`.

Browser shows ERR_CONNECTION_RESET for port 9090 or 3000

Docker bypasses UFW iptables rules. Use SSH tunnel instead:

```
ssh -L 9090:localhost:9090 -L 3000:localhost:3000 swilson@100.91.118.118
```

Grafana panel shows No data

```
1. Check the Data source dropdown — must be **prometheus-10**, not the
placeholder "prometheus"
2. Change time range to **Last 5 minutes** and click **Run queries**
3. Switch to **Code** mode and type the metric name directly
4. GPU utilization at 0% is correct at idle — it is not an error
```

Memory Used shows raw number like 4003753984

No unit set on the panel. Edit panel → Standard options → Unit → **bytes (SI)**.

SUDO POLICY VIOLATION broadcast messages

This is a sysadmin audit policy on spark02. Commands still execute — this is just a notification to the admin team that sudo was used. It is not an error.

AI Home