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.

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)

Step 3 — Clone and Build nv-monitor

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

```
make
```

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
- **GPU Processes:** PID, user, type (C=compute, G=graphics), CPU%, GPU memory, command
- **History chart:** rolling 20-sample graph of CPU (green) and GPU (cyan)

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

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`.

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”,…}`

Step 7 — Allow Docker Bridge to Reach nv-monitor

Docker containers live in the `172.17.x.x` subnet. The firewall blocks them from reaching port 9101 on the host by default.

```
sudo ufw allow from 172.17.0.0/16 to any port 9101
```

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

Step 8 — Access UIs from Your Mac via SSH Tunnel

Docker's iptables rules bypass UFW, making direct browser access unreliable. SSH port forwarding is simpler, more secure, and works over any network including Tailscale.

On your **Mac**, open a new terminal:

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

Then open in your Mac browser:

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

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 under 10ms.

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**

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**
2. Click **+ Add visualization**
3. Add each panel below one at a time

Dashboard panels

- **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
- **CPU Usage %** — metric: `nv_cpu_usage_percent` — type: Time series
- **CPU Temperature** — metric: `nv_cpu_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**.

Step 12 — Load Test with demo-load

Build the synthetic load generator:

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

This generates sinusoidal CPU and GPU load simultaneously. Watch the Grafana dashboard for live activity.

Verify the GPU is under load:

```
nvidia-smi
```

Expected output shows:

- GPU-Util: ~40%
- Temperature: ~60°C
- Power: ~17W
- Process: `./demo-load` using ~170MiB GPU memory

Press **Ctrl+C** to stop the load.

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 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:

```
sudo ufw allow from 172.17.0.0/16 to any port 9101
```

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
```

CPU Usage % panel shows No data

The label filter `{cpu="total"}` does not exist. Remove the filter:

```
nv_cpu_usage_percent
```

Also change visualization from Stat to Time series.

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 the Spark. Commands still execute — this is just a notification to the admin team that sudo was used. It is not an error.

AI Home