K3s Jetson Node Monitoring — CPU/GPU/RAM via Prometheus + Node Exporter (AGX/Orin)

This guide shows how to monitor **CPU, GPU, and RAM** on NVIDIA Jetson (AGX/Orin) worker nodes in a **K3s** cluster using **Prometheus** and **Node Exporter**. It includes exact file contents, systemd units, verification steps, troubleshooting, and tips for scaling to more nodes.

> **Topology assumed**
>
> - **Master/Prometheus** on a Linux host (e.g., ###, IP #####).
> - **Workers** are Jetsons (e.g., ###, ###).
> - We run **node_exporter** and a **GPU metrics writer** directly on each worker.

---

1) Prerequisites

On each **Jetson worker**:

- Ubuntu 20.04/22.04 (JetPack kernel OK)
- `node_exporter` binary (arm64)
- `tegrastats` available in PATH (comes with JetPack)
- `systemd` (for persistent services)
- Open inbound **TCP 9100** from Prometheus host (same LAN)

On **Prometheus (master)**:

- Prometheus 2.5x+ binary (amd64 OK)
- Network reachability to each worker's port **9100**

> If using firewalls, allow `tcp/9100` from the Prometheus host to the Jetsons.

---

2) Install Node Exporter on a Jetson worker

> Replace `$USER` with the login user on the Jetson (e.g., `agx`, `orin1`).

```
# On the Jetson worker (AGX/Orin)
cd ~
wget https://github.com/prometheus/node_exporter/releases/download/v1.9.1/
node_exporter-1.9.1.linux-arm64.tar.gz
sudo tar -xzf node_exporter-1.9.1.linux-arm64.tar.gz -C /usr/local --strip-
components=1 node_exporter-1.9.1.linux-arm64/node_exporter
```

```
# Create textfile directory for custom metrics
sudo mkdir -p /var/lib/node_exporter
sudo chown -R $USER:$USER /var/lib/node_exporter
sudo chmod 755 /var/lib/node_exporter
```

2.1 systemd unit for node_exporter (Jetson-friendly)

Jetson kernels can hang some default collectors. We **disable defaults** and enable a small, safe set + textfile collector.

Create `/etc/systemd/system/node_exporter.service`:

```
[Unit]
Description=Node Exporter
After=network.target

[Service]
User=%i
ExecStart=/usr/local/bin/node_exporter \
  --web.listen-address=:9100 \
  --collector.disable-defaults \
  --collector.cpu \
  --collector.meminfo \
  --collector.loadavg \
  --collector.netdev \
  --collector.time \
  --collector.textfile \
  --collector.textfile.directory=/var/lib/node_exporter \
  --log.level=info
Restart=on-failure
RestartSec=2

[Install]
WantedBy=multi-user.target
```

> If your distro doesn't support templated `%i` users, just set `User=<your-jetson-user>` (e.g., `User=orin1`).

Enable + start:

```
sudo systemctl daemon-reexec
sudo systemctl daemon-reload
sudo systemctl enable node_exporter
sudo systemctl start node_exporter
```

Verify locally:

```
curl -s http://localhost:9100/metrics | head -n 20
```

If it prints metrics (Go/runtime and node_*), it's good. If it hangs, see **Troubleshooting**.

---

3) Jetson GPU/CPU/RAM custom metrics (textfile collector)

We'll run a tiny script that queries `tegrastats` every second and writes Prometheus-compatible metrics to `/var/lib/node_exporter/gpu_metrics.prom`.

3.1 Script: `/usr/local/bin/gpu_metrics_exporter.sh`

```
#!/bin/bash

OUT="/var/lib/node_exporter/gpu_metrics.prom"
LOG="/tmp/gpu_exporter_debug.log"

while true; do
  echo "[$(date)] Running scrape..." >> "$LOG"

  # Capture tegrastats output (1 line only)
  TSTAT=$(timeout 3s tegrastats | head -n 1)
  echo "[$(date)] TSTAT=$TSTAT" >> "$LOG"

  # Extract metrics using updated and correct regex
  RAM_USED_MB=$(echo "$TSTAT" | grep -oP 'RAM \K[0-9]+(?=/)')
  GPU_USAGE=$(echo "$TSTAT" | grep -oP 'GR3D_FREQ \K[0-9]+(?=%)')
  CPU_RAW=$(echo "$TSTAT" | grep -oP 'CPU \[\K[^\]]+')
  CPU_USAGE=$(echo "$CPU_RAW" | grep -oP '[0-9]+(?=%)' | awk '{sum+=$1} END
{print sum}')

  # Fallback values if parsing failed
  RAM_USED_MB=${RAM_USED_MB:-0}
  GPU_USAGE=${GPU_USAGE:-0}
  CPU_USAGE=${CPU_USAGE:-0}

  # Write to Prometheus-compatible format
  cat <<EOF > "$OUT"
jetson_gpu_usage_percent $GPU_USAGE
jetson_memory_used_mb $RAM_USED_MB
jetson_cpu_combined_percent $CPU_USAGE
EOF
```

```
  echo "[$(date)] Wrote GPU=$GPU_USAGE, RAM=$RAM_USED_MB, CPU=$CPU_USAGE" >>
"$LOG"

  sleep 1
done
```

Permissions:

```
sudo chmod +x /usr/local/bin/gpu_metrics_exporter.sh
```

3.2 systemd unit: `/etc/systemd/system/gpu-metrics.service`

```
[Unit]
Description=Jetson GPU/CPU/RAM Metrics Exporter (tegrastats → textfile)
After=network.target

[Service]
ExecStart=/usr/local/bin/gpu_metrics_exporter.sh
Restart=always
RestartSec=1

[Install]
WantedBy=multi-user.target
```

Enable + start:

```
sudo systemctl daemon-reexec
sudo systemctl daemon-reload
sudo systemctl enable gpu-metrics
sudo systemctl start gpu-metrics
```

Check file updates:

```
watch -n1 'cat /var/lib/node_exporter/gpu_metrics.prom'
```

You should see the three lines updating (values will change as load changes):

```
jetson_<hostname>_gpu_usage_percent 0
jetson_<hostname>_cpu_combined_percent 23
jetson_<hostname>_memory_used_mb 1874
```

4) Prometheus configuration (on master)

On the Prometheus server (e.g., `nuc`), add each Jetson worker as a target.

`prometheus.yml` minimal example:

```
global:
  scrape_interval: 5s

scrape_configs:
  - job_name: 'master-node'
    static_configs:
      - targets: ['localhost:9100']

  - job_name: 'jetson-workers'
    static_configs:
      - targets:
        - '192.168.1.135:9100'   # agx-desktop
        - '192.168.1.77:9100'    # orin1-desktop
```

Restart Prometheus, then browse `http://<MASTER_IP>:9090` → **Status → Targets** and confirm all are **UP**.

Example PromQL:

- GPU % (current node):

  ```
  jetson_{job="jetson-workers"}_gpu_usage_percent
  ```

- Sum CPU % across Jetsons:

  ```
  sum by (instance) (jetson_*_cpu_combined_percent)
  ```

- Memory used (MB→GB):

  ```
  jetson_*_memory_used_mb / 1024
  ```

  Optional: Create Grafana panels for these metrics. A simple Stat/Time-series works great.

5) Verification Checklist (per worker)

1. **Services enabled**

```
systemctl is-enabled node_exporter   # → enabled
systemctl is-enabled gpu-metrics     # → enabled
```

2. **Processes running**

```
ps aux | grep '[n]ode_exporter'
ps aux | grep '[g]pu_metrics_exporter.sh'
```

3. **Local curl works**

```
curl -s http://localhost:9100/metrics | head -n 10
curl -s http://localhost:9100/metrics | grep -E '^jetson_'
```

4. **Remote curl from master works**

```
curl -s http://<JETSON_IP>:9100/metrics | grep -E '^jetson_'
```

5. **Prometheus target UP** (on master):
6. Web → Status → Targets → `jetson-workers`

---

6) Troubleshooting (Jetson-specific)

Symptom: `/metrics` hangs or is very slow

- Root cause is usually problematic collectors on Jetson kernels.
- **Fix:** Use the service flags shown above: `--collector.disable-defaults` and enable only `cpu, meminfo, loadavg, netdev, time, textfile`.
- If still slow, temporarily run without `--collector.textfile` to isolate textfile path/permissions.

Symptom: Remote curl connects but no response

- Confirm listening on all interfaces:

```
ss -tuln | grep 9100   # Expect: LISTEN *:9100
```

- Ensure no firewall blocks between Prometheus and Jetson. On Jetson, `ufw` is usually absent; iptables kube rules typically don't touch 9100.

Symptom: Node Exporter logs show `broken pipe`

- That's often just the client closing the connection early (e.g., `head` truncates). If full requests work, this is harmless.

Symptom: Values never change

- Check that `gpu-metrics` service is active and `tegrastats` prints live data.
- Review `/tmp/gpu_exporter_debug.log` for parsing.

Symptom: `.prom` file partial/garbled

- Always **write atomically** (`tmp` → `mv`). Make sure directory perms are readable by the node_exporter user.

---

7) Adding more Jetson workers

Repeat **Sections 2–3** on each worker. The script prefixes metrics with the sanitized **hostname**, so per-node metrics won't collide. Just add each worker's `IP:9100` to Prometheus.

---

8) Optional: Run Node Exporter as a DaemonSet (K8s)

You can deploy node_exporter via a DaemonSet, but for Jetson-specific GPU parsing via `tegrastats`, a local systemd unit + textfile collector is simpler and more controllable on embedded devices.

---

9) Security notes

- Port 9100 exposes host metrics; restrict network to your monitoring subnet.
- Consider running node_exporter as an unprivileged user.
- Avoid exposing 9100 to the wider internet.

---

10) Quick commands (cheat sheet)

Worker (Jetson):

```
# enable + start
sudo systemctl enable --now node_exporter gpu-metrics

# status
systemctl status node_exporter
systemctl status gpu-metrics
```

```
# logs (last 100 lines)
sudo journalctl -u node_exporter -n 100 --no-pager
sudo journalctl -u gpu-metrics -n 100 --no-pager

# test locally
curl -s http://localhost:9100/metrics | grep -E '^jetson_'
```

Prometheus (master):

```
curl -s http://<JETSON_IP>:9100/metrics | head -n 10
curl -s http://<JETSON_IP>:9100/metrics | grep -E '^jetson_'
```

That's it. With this setup, each Jetson publishes CPU, RAM, and GPU utilization to Prometheus every second via the textfile collector, and Node Exporter serves everything cleanly on port 9100. Share this doc with your colleagues to replicate the exact configuration reliably.