Implementation of a Cloud-Based Monitoring System Using Prometheus, Node Exporter, and Grafana

Ziyad Rhazi

December 6, 2025

Abstract

This report presents the design and implementation of a monitoring system deployed on a Microsoft Azure Virtual Machine. The system integrates Prometheus for time-series metrics collection, Node Exporter for host-level monitoring, and Grafana for interactive dashboards. The objective is to build a practical, production-style monitoring stack capable of visualizing CPU, memory, disk, and network performance metrics in real time. The report describes the architecture, installation steps, configuration choices, and resulting dashboards.

1 Introduction

Monitoring is essential for ensuring that cloud-based services remain available, responsive, and efficient. Virtual machines (VMs) running in the cloud can experience resource saturation, performance degradation, or network bottlenecks. Without observability, these issues may go unnoticed until they affect users.

The goal of this project is to implement a lightweight, open-source monitoring solution on an Azure Linux VM using:

- **Prometheus**: a pull-based time-series monitoring system.
- **Node Exporter**: an agent that exposes Linux system metrics.
- **Grafana**: a visualization platform for dashboards and charts.

This stack is widely used in industry by DevOps and SRE teams. The implementation closely follows real-world practices, including the use of systemd services and Azure Network Security Group (NSG) rules.

2 System Architecture

Figure 1 shows the overall architecture deployed in this project.

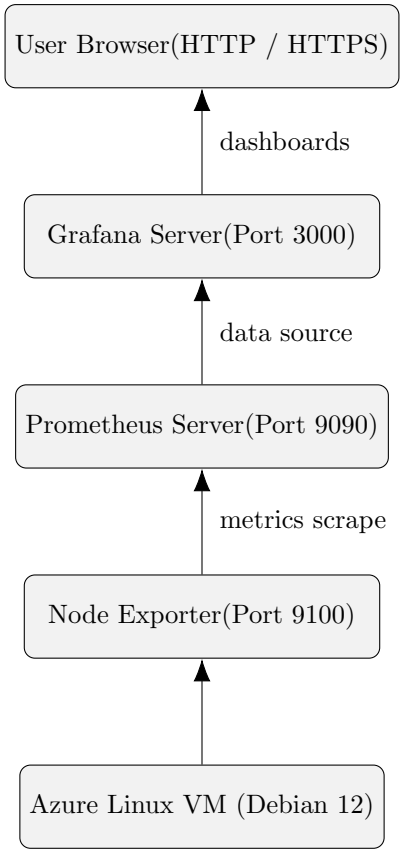

Figure 1: Monitoring System Architecture on Azure

The components interact as follows:

1. Node Exporter runs on the VM and exposes operating system metrics via HTTP on port 9100.
2. Prometheus periodically scrapes metrics from Node Exporter and stores them in its local time-series database.
3. Grafana is configured to use Prometheus as a data source and queries metrics for visualization.
4. The user accesses Grafana through a web browser using the VM public IP address.

3 Azure VM and Network Configuration

A Debian 12 (Bookworm) Virtual Machine was created in Microsoft Azure. The following inbound NSG rules were configured to allow remote access:

- TCP 22 for SSH administration.
- TCP 9090 for Prometheus web UI.
- TCP 3000 for Grafana dashboards.

On the VM, packages were updated before any installation:

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y wget curl tar
```

For simplicity, the monitoring stack is deployed directly on the VM without containers.

4 Prometheus Installation and Configuration

4.1 User and Directory Setup

A dedicated system user and directories were created for Prometheus:

```
sudo useradd --no-create-home --shell /usr/sbin/nologin prometheus
sudo mkdir -p /etc/prometheus
sudo mkdir -p /var/lib/prometheus
sudo chown prometheus:prometheus /var/lib/prometheus
```

4.2 Downloading and Installing Prometheus

The Prometheus binary release was downloaded from the official GitHub repository:

```
cd /tmp
wget https://github.com/prometheus/prometheus/releases/download/v2.54.1/\
prometheus-2.54.1.linux-amd64.tar.gz
tar xvf prometheus-2.54.1.linux-amd64.tar.gz
cd prometheus-2.54.1.linux-amd64

sudo mv prometheus /usr/local/bin/
sudo mv promtool /usr/local/bin/
sudo chown prometheus:prometheus /usr/local/bin/prometheus \
  /usr/local/bin/promtool

sudo mv consoles /etc/prometheus
sudo mv console_libraries /etc/prometheus
sudo mv prometheus.yml /etc/prometheus/prometheus.yml
sudo chown -R prometheus:prometheus /etc/prometheus
```

4.3 Prometheus Configuration

The configuration file `/etc/prometheus/prometheus.yml` was adapted to scrape both Prometheus itself and Node Exporter:

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'node_exporter'
    static_configs:
      - targets: ['localhost:9100']
```

4.4 Systemd Service for Prometheus

A systemd unit file was created at `/etc/systemd/system/prometheus.service`:

```
[Unit]
Description=Prometheus Monitoring
Wants=network-online.target
After=network-online.target
```

```
[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
  --config.file=/etc/prometheus/prometheus.yml \
  --storage.tsdb.path=/var/lib/prometheus \
  --web.console.templates=/etc/prometheus/consoles \
  --web.console.libraries=/etc/prometheus/console_libraries

Restart=on-failure

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

The service was enabled and started:

```
sudo systemctl daemon-reload
sudo systemctl enable prometheus
sudo systemctl start prometheus
sudo systemctl status prometheus
```

At this point, the Prometheus web UI was reachable at:

http://<VM_PUBLIC_IP>:9090

5 Node Exporter Installation

5.1 Downloading and Installing Node Exporter

Node Exporter is responsible for exporting host-level metrics such as CPU, memory, disk, and network usage.

```
cd /tmp
wget https://github.com/prometheus/node_exporter/releases/download/v1.8.2/\
node_exporter-1.8.2.linux-amd64.tar.gz
tar xvf node_exporter-1.8.2.linux-amd64.tar.gz
cd node_exporter-1.8.2.linux-amd64
sudo mv node_exporter /usr/local/bin/
```

A system user was created:

```
sudo useradd --no-create-home --shell /usr/sbin/nologin node_exporter
sudo chown node_exporter:node_exporter /usr/local/bin/node_exporter
```

5.2 Systemd Service for Node Exporter

The unit file /etc/systemd/system/node$_e$*xporter.servicewasdefinedas* :

[Service] User=node$_e$*xporterGroup* = *node*$_e$*xporterType* = *simpleExecStart* = */usr/local/bin/node*$_e$*xporter*

Restart=on-failure

[Install] WantedBy=multi-user.target

The service was activated:

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

A quick check confirmed that metrics were available locally:

```
curl http://localhost:9100/metrics | head
```

The Prometheus *Targets* page then showed both `prometheus` and `node_exporter` in the `UP` state.

6 Grafana Installation and Configuration

6.1 Repository and Key Configuration

Grafana is not available in the default Debian repositories, so the official Grafana repository was added:

```
cd /tmp
wget https://packages.grafana.com/gpg.key -O grafana.key
sudo gpg --dearmor grafana.key
sudo mv grafana.key.gpg /usr/share/keyrings/grafana.gpg

echo 'deb [signed-by=/usr/share/keyrings/grafana.gpg] \
https://packages.grafana.com/oss/deb stable main' | \
  sudo tee /etc/apt/sources.list.d/grafana.list

sudo apt update
sudo apt install -y grafana
```

6.2 Starting Grafana

The Grafana service was enabled and started:

```
sudo systemctl daemon-reload
sudo systemctl enable grafana-server
sudo systemctl start grafana-server
sudo systemctl status grafana-server
```

Grafana became accessible at:

`http://<VM_PUBLIC_IP>:3000`

The default credentials (`admin` / `admin`) were used and the password was changed as required.

6.3 Adding Prometheus as a Data Source

Within the Grafana web interface:

1. Navigate to *Configuration* → *Data sources*.
2. Click *Add data source* and select *Prometheus*.
3. Set the URL to `http://localhost:9090`.
4. Click *Save & Test*; Grafana reported that the data source was working.

7 Dashboard Design

A dashboard named **Azure VM Monitoring** was created in Grafana. It contains several panels built with PromQL queries to visualize key resource metrics.

7.1 CPU Usage

CPU usage was computed using the `node_cpu_seconds_total` metric:

```
100 - (avg by(instance) (irate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)
```

This panel shows the percentage of CPU time spent in non-idle states.

7.2 Memory Usage

Memory pressure was monitored using:

```
100 * (1 - (node_memory_MemAvailable_bytes /
           node_memory_MemTotal_bytes))
```

7.3 Disk Usage

Disk usage on the root filesystem was visualized as:

```
100 * (node_filesystem_size_bytes{mountpoint="/"} -
       node_filesystem_free_bytes{mountpoint="/"}) /
       node_filesystem_size_bytes{mountpoint="/"}
```

7.4 Network Traffic

Network throughput was monitored using:

```
rate(node_network_receive_bytes_total[1m])
rate(node_network_transmit_bytes_total[1m])
```

Panels were configured as time series graphs. Together, these panels provide a clear, real-time overview of the VM's health.

8 Conclusion

This project successfully implemented an end-to-end monitoring system on an Azure Linux VM using Prometheus, Node Exporter, and Grafana. The solution collects and visualizes CPU, memory, disk, and network metrics, demonstrating how open-source tools can be combined to build a robust observability stack.

The final setup resembles industry practices and can be extended with additional exporters, alerting rules, and more complex dashboards. As a learning exercise, it provided hands-on experience with Linux services, network configuration, time-series monitoring, and dashboard design.