Monitoring for U-BMC with Grafana and Prometheus

OSS
ONE STOP SYSTEMS

(877) 438-2724 onestopsystems.com

Monitoring for U-BMC with Grafana and Prometheus

In this paper, we provide a detailed guide for setting up a monitoring system for the U-BMC using Prometheus and Grafana. With this monitoring system, you can collect metrics and generate dashboards that provide a comprehensive overview of your system's performance and health. Examples are presented in the paper for the automated generation of Docker containers for the components of the monitoring system. We integrate monitoring services into the host system and combine them with U-BMC monitoring. This creates a complete performance monitoring solution that effectively addresses typical performance problems. We show how Prometheus instances can serve as a data source for monitoring multiple systems through its federation feature, and configure them to gather telemetry even when the systems are offline.

Introduction

Our goal is to ensure that our systems are dependable and consistent. Through anticipating possible failures, we can take proactive measures to prevent them and maintain system stability. By ensuring optimal performance, the system can reliably handle the expected workload. To make well-informed decisions regarding maintenance, upgrades, and expansion, it is essential to have a thorough comprehension of the health and performance of the system. Overall, monitoring systems are crucial to system reliability and stability. They provide the insights from the network, machine, and application levels, but they also have limitations. We can only monitor what we can measure.

Alternately and complimentary to monitoring, we can conduct audits of our system and network performance, execute testing and validation, analyze log data, or ask our users for feedback. These are all important and valuable, but they are not a substitute for the ease and speed of a monitoring system. As such, the information shared here is relevant to anyone who is involved in the design, deployment, and maintenance of systems connected at the edge.

Problem Statement

So, let's set ourselves to the task of creating a monitoring system for the U-BMC and the underlying computer host system. We want to use open source software to create a monitoring system that is easy to use and easy to maintain. Metrics exposed by the Redfish API are only available for the instant in time that the system is being queried for information. Out on the edge, we may encounter challenges to the reliability and stability of our system. This includes the possibility of network outages that take down our monitoring system's ability to call home and report its health and performance.

We need a history of the health and performance of the system. We should store the data collected by the monitoring system somewhere persistent. We also consider that our system is part of a fleet of systems that we'd like to centralize our monitoring data for analysis and cross-comparison.

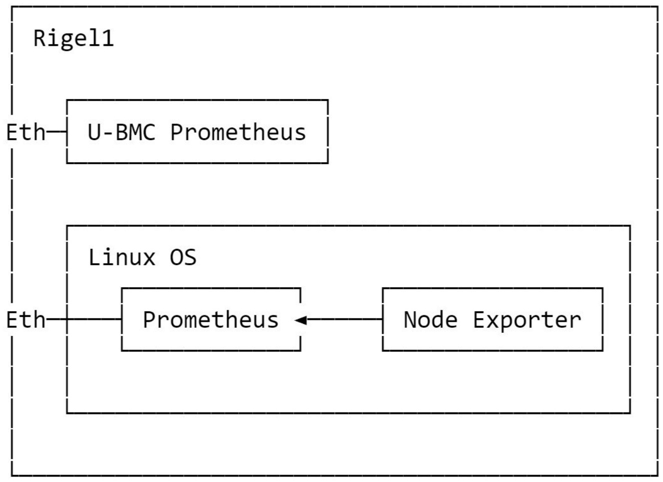
Our aim is to promptly react to variations in the system's health and performance. Additionally, we require the ability to visualize the data collected by the monitoring system. Furthermore, we have an automated process that must be activated when the system's health and performance deteriorate. We want to correlate warnings and errors with the health and performance of the system.

Solution

To tackle the problems outlined in our problem statement, we suggest utilizing Prometheus and Grafana. These two open-source projects collaborate to deliver a monitoring solution, and with our configuration we will tailor it for the Rigel system.

Prometheus Data Sources

Prometheus creates a time-series database that can store metrics from each of the components we want to monitor. Since an instance of Prometheus is already operational on the U-BMC, we can readily collect metrics from it. We'll add Prometheus to the host system to monitor the health of the host computer system, and capture metrics that are only available from the computer system's host operating system.



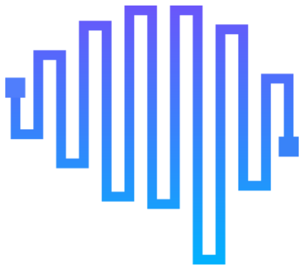
Aside: In our example above, we're using the hostname Rigel1 to point to the Linux OS and i-Rigel1 to point to the U-BMC. We will have two Prometheus instances running, one on U-BMC and the other on the host system. Each of them have a unique hostname, and each of them are configured to scrape metrics from the sensors that it can access.

In order to populate Prometheus with host operating system metrics, we'll need to install the Node Exporter on the host system. The Node Exporter collects host OS metrics and exposes them via a webserver for Prometheus scraping in a compatible format.

The instructions for installing the Node Exporter are here https://prometheus.io/docs/guides/node-exporter/. The Node Exporter will start an HTTP server that listens on port 9100. It collects a default set of metrics for CPU, network, memory, and disk usage on the host system. You can configure the Node Exporter to change the collector behavior to add additional collectors. For further details on adjusting the collectors that are utilized, refer to the Node Exporter documentation available at https://github.com/prometheus/node_exporter. After installing Node Exporter, you can verify its functionality by accessing the metrics by hitting http://localhost:9100/metrics with your browser.

By following the instructions linked above, you should have also installed Prometheus on the host system and set it up to scrape metrics from the Node Exporter. Here's an example of the configuration for the Prometheus server that scrapes metrics from the Node Exporter. See the code in the repository for "node-exporter-scraper.yml".

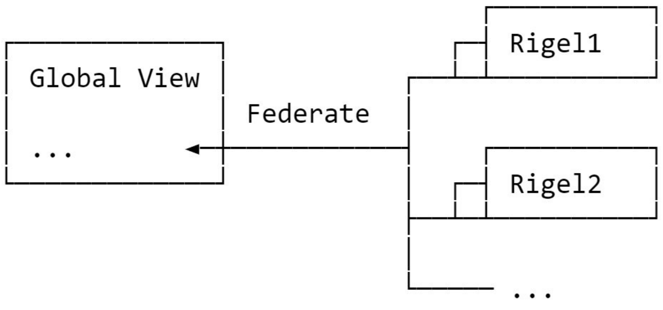
Check your Prometheus configuration using the HTTP server that Prometheus starts. The targets page, http://rigel1:9090/targets will show you the status of the scrape targets, which should include the Node Exporter. You can expand the target to see the status of the last scrape and the last error message. You can check that the Prometheus configuration is adding the metrics to the time-series database by checking the graph page, http://rigel1:9090/graph. You can type an expression in the expression field and click the Execute button to see the results.



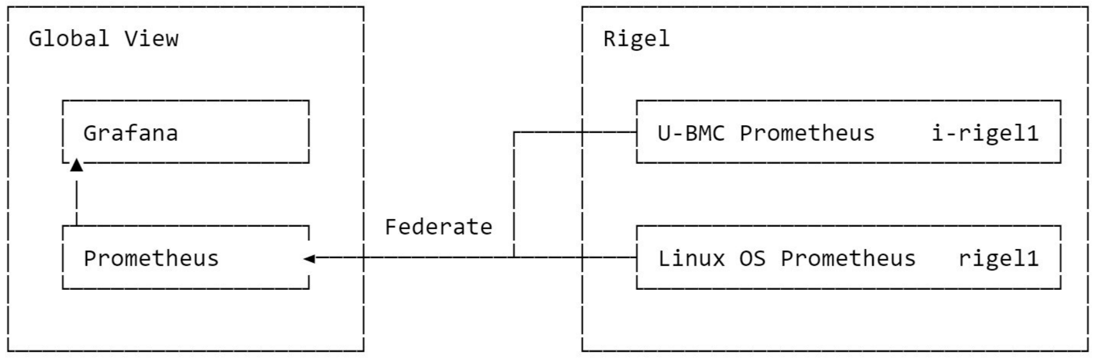
With the existing Prometheus server running in the U-BMC, and the new one you just created on the host, we now have two different services constantly scraping metrics. You have the ability to fine-tune the scrape interval specified in the node-exporter-scraper.yml configuration file. As long as Prometheus is running, it continuously scrapes the metrics and saves them to a time-series database. At any time, we can retrieve telemetry, even during times when our system has lost connection to the internet.

Prometheus Federation and Grafana

The system providing the global view should be placed on the network in a position to access the Prometheus servers whenever a connection to the Rigel system is restored. We'll use Grafana to display the metrics collected by all of our Prometheus instances on the Rigel and transform them into visualizations, enabling analysis and cross-comparison. We'll install Prometheus and Grafana on the workstation that we want to present a global view to; the global view workstation is the host which will aggregate the metrics from the fleet of systems.



Prometheus has the capability to run in a variety of different modes to suit the purpose of their users who are either pushing or pulling metric data. The federation mode we are employing on the global view workstation gathers data from multiple Prometheus servers into a centralized Prometheus server.



Installing Prometheus and Grafana on the Global View Workstation

We've created a docker compose file to help with the installation of Prometheus and Grafana on the global view workstation. Our docker compose file creates docker containers for each of the Prometheus and Grafana containers. Before we create the containers, we need to configure the Prometheus instance that federates the metrics from the other source Prometheus instance we created earlier and the U-BMC instance. The only configuration file you should need to edit before creating the containers are created is federate.yml. Change the file to include your targets: the IP addresses or hostnames of the U-BMC Prometheus and the Rigel operating system's Prometheus. Both of these Prometheus instances are running on the Rigel system, so we'll use the hostnames of the Rigel system's Linux OS and the U-BMC as the targets. Both services are running on port 9090, so this is the example snippet of the configuration file that you should edit.

```
- job_name: federate-u-bmcs
  static_configs:
  - targets:
      - 'i-rigel1:9090'
      #- 'i-rigel2:9090'
...
 - job_name: federate-nodes
  static_configs:
  - targets:
      - 'rigel1:9090'
      #- 'rigel2:9090'
```

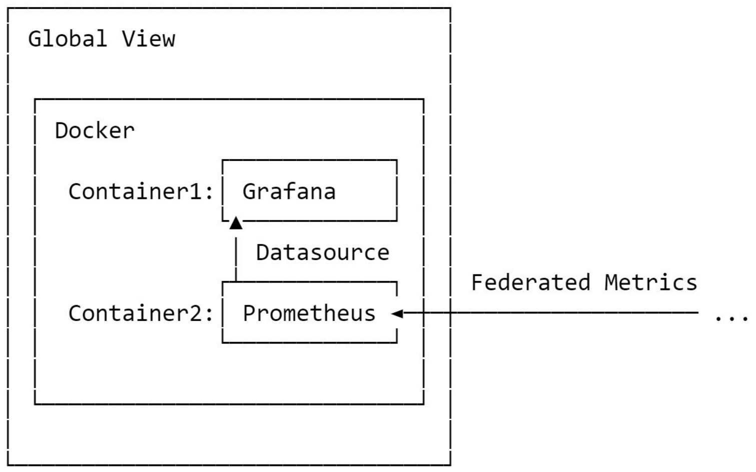
Assuming you have more than one Rigel, you'll add the targets to the list of targets for each of the jobs that you want to scrape. You can easily repeat this for each of the Rigel systems you want to monitor; just uncomment the lines for i-rigel2 and rigel2.

After you've configured the targets, we can create the containers by running the docker compose script. See the code repository for a file called "docker-compose.yml".

Run the docker compose script like this:

```
docker compose up -d
```

With the execution of that compose script, we'll have a Prometheus instance running and configured to collect metrics from other Prometheus instances using the federation feature. We'll also have a Grafana instance running and configured to use the Prometheus instance as a datasource. This Prometheus datasource was added to the Grafana instance by using the provisioning feature of Grafana, which simply configures datasources from a config file. You can add more datasources to the Grafana instance, but for now we'll just use the Prometheus instance that we've automatically provisioned.



You can list these containers using the command docker ps to list container IDs and to see a summary of how the containers are configured.

```
CONTAINER ID      IMAGE                                        COMMAND
e00c2e27b59a      prom/prometheus:v2.37.6                  "/bin/prometheus --c…"
02e062bf2436      grafana/grafana:latest                   "/run.sh"

CREATED                 STATUS                   PORTS
16 minutes ago    Up 16 minutes     0.0.0.0:9090->9090/tcp, :::9090->9090/tcp
16 minutes ago    Up 16 minutes     0.0.0.0:3000->3000/tcp, :::3000->3000/tcp

NAMES
ubmc-grafana_prometheus_1
ubmc-grafana_grafana_1
```

Configuring Grafana

We've automated the configuration of Grafana to use the Prometheus instance as a datasource, so our next task is to create the dashboards that we want to use. There are a few dashboards that we've created to ease the process of learning how to create dashboards. We'll import the dashboards into the Grafana instance using the Grafana API.

Navigate to the Grafana instance at http://localhost:3000 and log in with the default username and password of admin/admin. Once you're logged in and set a new admin password, you'll be presented with the Grafana home page that we've customized to be a welcome page for the Global View system. The welcome page includes links to a list of dashboards which will be empty until we import the dashboards.

To help automate creating the dashboards, we need to configure the Grafana instance to allow us to access to the Grafana API. Create a new Service Account by navigating to http://localhost:3000/org/serviceaccounts on the global view workstation. Add a service account using the role of administrator or editor, and then generate a token. You'll be able to use the token to access the API using the HTTP Authorization header. Here's an example of the HTTP request to get the list of datasources using the Grafana API.

```
GET localhost:3000/api/datasources
Accept: application/json
Content-Type: application/json
Authorization: Bearer glsa_4ZCTF620R0dKAuTMMjENdl3xLKCwZWgD_0c7c1e2b
```

The API will return a list of datasources that are configured for the Grafana instance, so you should see the Prometheus federation server in the list.

```
[{
  "id": 1,
  "uid": "XXlwm1L4k",
  "orgId": 1,
  "name": "Prometheus"
}]
```

Now we can use the API to create our dashboards, and we included a file you can use to automate creating the dashboards. The file runs in IntelliJ IDEs and includes commented curl commands for dashboard creation if you don't have the IDE. Replace the previously created API token in the file with the token generated for the service account. See the source code repository for this file, "grafana/create-dashboards.http".

Now that we've created the dashboards, we can navigate Grafana's welcome page and see the dashboards that we've created. You can open the dashboards that we've created by clicking on the links on the welcome page. We have already automatically saved variables from the Prometheus metrics for a few dashboards, which results in us using the appropriate hostnames of your U-BMC systems.

Now let's configure the dashboard for the node metrics. Grafana curates a dashboard repository here: https://grafana.com/grafana/dashboards/, and you can copy the ID to load a dashboard created by the community. Using the left navigation in your Grafana instance, go to Dashboards, and click on Import. Alternatively, you can click this link to jump to that location: http://localhost:3000/dashboard/import. To load the Node Exporter dashboard from the community repository, simply enter the ID 1860 and click Load. The dashboard configuration will be loaded, and all you should need to change is the data source. Choose the Prometheus data source from the dropdown list, and then click the Import button.

Navigate back to the welcome page, and a new dashboard called “Node Exporter Full” will be listed. Now you should be able to see the node metrics for the Rigel operating system.

You should have a very good starting point for creating a monitoring solution for your systems, but there’s another Prometheus exporter we can add to the host operating system to get additional metrics from the GPU. By now you should know the pattern; we need to add an exporter to the host operating system, and then configure Prometheus to scrape the metrics from the exporter. NVIDIA has created a Prometheus exporter that we can use to get metrics from the GPU, and you can find the installation instructions here: https://docs.nvidia.com/datacenter/cloud-native/gpu-telemetry/dcgm-exporter.html. If you prefer to download from the source code, see this repository: https://github.com/NVIDIA/dcgm-exporter. It can be a little tricky to get it started, and you may need to restart the docker service or reinstall NVidia libraries. Make sure your user is in the docker group, you run docker as an administrator, and that you are able to execute the nvidia-smi command. Here’s the command that we used to start the exporter.

```
docker run -d --gpus all --rm -p 9400:9400 nvcr.io/nvidia/k8s/dcgm-exporter:2.0.13-2.1.2-
ubuntu18.04
```

Once you’ve started the DCGM exporter, you’ll need to configure Prometheus to scrape the metrics from the exporter. We can reuse the Prometheus instance on the operating system, so uncomment the scrape configuration in the Prometheus configuration file on the Linux operating system that has been commented out.

```
- job_name: 'dcgm'
  scrape_interval: 15s
  static_configs:
  - targets: ['localhost:9400']
```

Restart the Prometheus instance, and you should be able to see the metrics from the exporter in the Prometheus UI, located at http://rigel1:9090/targets. Lastly, we need to change the federate.yml configuration to include “dcgm” metrics from the Prometheus instance on the operating system. Here is the relevant section of configuration from the federate.yml file.

```
params:
  'match[]':
    - '{job="node"}'
    - '{job="dcgm"}'
```

Restart the Prometheus instance on the global view workstation, and you can verify that the GPU metrics are being federated. Using the targets page in the Prometheus UI http://localhost:9090/targets, you should see the target “federate-nodes” has an additional match for the “dcgm” job. Over on the graph page http://localhost:9090/graph, you can see the metrics from the exporter by selecting the DCGM_ metrics from the dropdown list.

Now, we can add a new dashboard to Grafana to display the metrics from the exporter, here is the link: https://grafana.com/grafana/dashboards/12239-nvidia-dcgm-exporter-dashboard/. Copy the ID 12239 and import the dashboard into Grafana. Just like last time, you'll need to change the data source to Prometheus, and then click Import.

With this comprehensive monitoring system, we can easily respond to changes in the health and performance of the U-BMC system, including warnings and errors, by automatically analyzing and interpreting the data.

Benefits

The solution presented in this white paper offers several benefits for organizations that need to monitor and manage sophisticated edge systems. These benefits include:

- **Improved visibility:** By using Prometheus and Grafana to monitor the health and performance of U-BMC systems, organizations can gain better visibility into the status of their systems. The dashboards provided by Grafana make it easy to view metrics in real-time, which can help organizations identify issues before they become critical.
- **Centralized monitoring:** With Prometheus, organizations can centralize the data collected by a fleet of systems in a single location for analysis and cross-comparison. This can help organizations identify trends and patterns across their entire U-BMC infrastructure, which can lead to more effective management and troubleshooting.
- **Automated responses:** The ability to automatically respond to changes in the health and performance of the system is critical for organizations that need to maintain uptime and ensure system reliability. By using Prometheus to monitor U-BMC systems, organizations can set up alerts and triggers that automatically respond to changes in the system's health and performance.
- **Customizability:** Prometheus and Grafana are highly customizable, which means that organizations can tailor their monitoring and visualization tools to meet their specific needs. This includes the ability to create custom dashboards and alerts that are tailored to specific use cases.

Overall, the solution presented in this white paper offers organizations a powerful set of tools for monitoring and managing U-BMC systems. By providing improved visibility, centralized monitoring, automated responses, and customizability, organizations can ensure that their systems are running smoothly and efficiently.

Conclusion

In conclusion, the use of Prometheus and Grafana provides a comprehensive solution to the problem of monitoring and analyzing the health and performance of the U-BMC system. By setting up the JSON exporter, Prometheus server, and Grafana dashboard, it is possible to collect, store, visualize, and analyze the metrics of the U-BMC system in a centralized and automated way.

The benefits of this approach include improved system reliability, faster response times to issues, and better overall system performance. The ability to monitor and analyze the U-BMC system in real-time enables proactive problem-solving and improves the overall uptime and performance of the system. With the use of these tools, organizations can ensure that their U-BMC systems are operating at peak efficiency and avoid costly downtime or maintenance issues.

Thank you for taking the time to read our white paper. We hope the information provided has been helpful and informative. At One Stop Systems, we strive to provide our customers with the best possible products and services. If you have any questions or comments about the content of this white paper or about our company in general, we would love to hear from you. Your feedback is important to us and helps us to continue improving our offerings. So please don’t hesitate to reach out to us with any additional questions or feedback. We look forward to hearing from you and thank you for your interest in One Stop Systems.