Search

On This Page

NVIDIA

Topics

RDG f
Monit and DTS

Created o

Is this page helpful?

Scope

This Refer
up a centr
running or
Kubernete

Leveraging

monitoring
acceleratio
and perfor

The inform
engineers,
enabled Ku

led instructions for setting
Service (
naged us

and man

es NVIDIA
ta center

ministrat
a high-performance, DPU-
rces.

ⓘ Wa

- F, successfully provisioned the
  gst them DOCA Telemetry
- o: **RDG for DPF with OVN-**

✓ Note

- This reference implementation, as the name implies, is a specific, opinionated deployment example designed to address the use case described above.
- While other approaches may exist to implement similar solutions, this document provides a detailed guide for this particular method.

Abbreviations and Acronyms

Term	Definition	Term	Definition
DOCA	Data Center Infrastructure-on-a-Chip Architecture	K8S	Kubernetes
DPF	DOCA Platform Framework	OVN	Open Virtual Network
DPU	Data Processing Unit	PVC	Persistent Volume Claim
DTS	DOCA Telemetry Service	RDG	Reference Deployment Guide
HBN	Host Based Networking	TSDB	Time Series Database

Introduction

DOCA Platform Framework (DPF) is a system for provisioning and orchestrating NVIDIA BlueField DPUs and DPU services in a Kubernetes cluster.

DPF simplifies DPU management by providing orchestration through a Kubernetes API, handling DPU provisioning and lifecycle management, and enabling efficient deployment and orchestration of infrastructure services on DPUs.

One of those services is DOCA Telemetry Service (DTS), which collects data from built-in providers and external telemetry applications. DTS supports several export mechanisms, including a Prometheus endpoint that can be scraped by a Prometheus server. Using Grafana as a visualization platform for the collected data, users can conveniently monitor their DPU resources.

In large DPU clusters provisioned and managed by DPF, with associated DTS services running on them, an automated and scalable approach for monitoring those DTS instances is essential to prevent overburdening the cluster and system administrators.

By utilizing DPF orchestration capabilities, Kubernetes-native tools, and Prometheus service discovery, an efficient monitoring solution can be achieved.

This guide provides a practical example of such a solution, demonstrating how to enable centralized DPU monitoring.

References

- **NVIDIA BlueField DPU**
- **NVIDIA DOCA**
- **NVIDIA DOCA HBN Service**
- **NVIDIA DOCA Telemetry Service**
- **NVIDIA DOCA BlueMan Service**
- **NVIDIA DPF Release Notes**
- **NVIDIA DPF GitHub Repository**
- **NVIDIA DPF System Overview**
- **NVIDIA DPF HBN and OVN-Kubernetes User Guide**
- **NVIDIA Ethernet Switching**
- **NVIDIA Cumulus Linux**
- **What is K8s?**
- **OVN-Kubernetes**
- **Prometheus**
- **Grafana**

Solution Architecture

Key Components and Technologies

- **NVIDIA BlueField® Data Processing Unit (DPU)**
  The NVIDIA® BlueField® data processing unit (DPU) ignites unprecedented innovation for modern data centers and supercomputing clusters. With its robust compute power and integrated software-defined hardware accelerators for networking, storage, and security, BlueField creates a secure and accelerated infrastructure for any workload in any environment, ushering in a new era of accelerated computing and AI.

- **NVIDIA DOCA Software Framework**
  NVIDIA DOCA™ unlocks the potential of the NVIDIA® BlueField® networking platform. By harnessing the power of BlueField DPUs and SuperNICs, DOCA enables the rapid creation of applications and services that offload, accelerate, and isolate data center workloads. It lets developers create software-defined, cloud-native, DPU- and SuperNIC-accelerated services with zero-trust protection, addressing the performance and security demands of modern data centers.

- **NVIDIA ConnectX SmartNICs**
  10/25/40/50/100/200 and 400G Ethernet Network Adapters
  The industry-leading NVIDIA® ConnectX® family of smart network interface cards (SmartNICs) offer advanced hardware offloads and accelerations. NVIDIA Ethernet adapters enable the highest ROI and lowest Total Cost of Ownership for hyperscale, public and private clouds, storage, machine

learning, AI, big data, and telco platforms.

- **NVIDIA LinkX Cables**
  The NVIDIA® LinkX® product family of cables and transceivers provides the industry's most complete line of 10, 25, 40, 50, 100, 200, and 400GbE in Ethernet and 100, 200 and 400Gb/s InfiniBand products for Cloud, HPC, hyperscale, Enterprise, telco, storage and artificial intelligence, data center applications.

- **NVIDIA Spectrum Ethernet Switches**
  Flexible form-factors with 16 to 128 physical ports, supporting 1GbE through 400GbE speeds.
  Based on a ground-breaking silicon technology optimized for performance and scalability, NVIDIA Spectrum switches are ideal for building high-performance, cost-effective, and efficient Cloud Data Center Networks, Ethernet Storage Fabric, and Deep Learning Interconnects.
  NVIDIA combines the benefits of **NVIDIA Spectrum™** switches, based on an industry-leading application-specific integrated circuit (ASIC) technology, with a wide variety of modern network operating system choices, including **NVIDIA Cumulus® Linux** , **SONiC** and **NVIDIA Onyx®**.

- **NVIDIA Cumulus Linux**
  NVIDIA® Cumulus® Linux is the industry's most innovative open network operating system that allows you to automate, customize, and scale your data center network like no other.

- **Kubernetes**
  Kubernetes is an open-source container orchestration platform for deployment automation, scaling, and management of containerized applications.

- **OVN-Kubernetes**
  OVN-Kubernetes (Open Virtual Networking - Kubernetes) is an open-source project that provides a robust networking solution for Kubernetes clusters with OVN (Open Virtual Networking) and Open vSwitch (Open Virtual Switch) at its core. It is a Kubernetes networking conformant plugin written according to the CNI (Container Network Interface) specifications.

Solution Design

The solution design is based on **RDG for DPF with OVN-Kubernetes and HBN Services - Solution Design**.

K8s Cluster Logical Design - Monitoring Stack

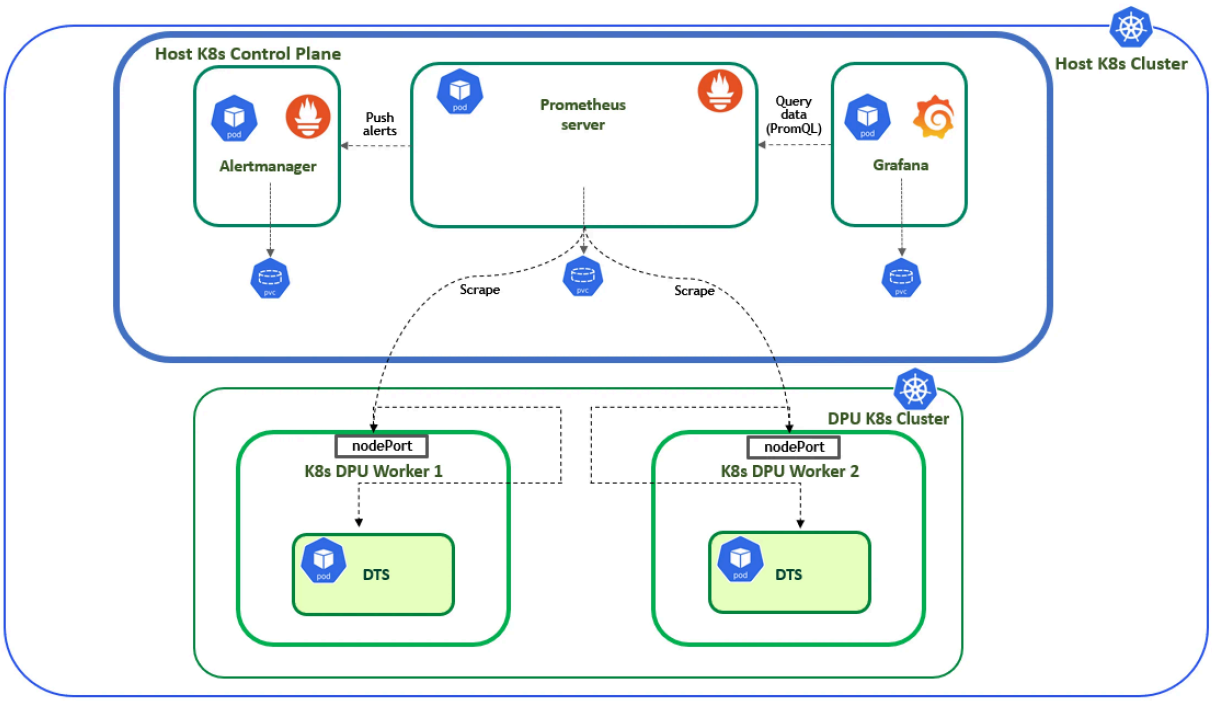
The following K8s logical design illustration demonstrates the main components of the monitoring stack in this solution:

- 1 x Prometheus server pod - scrapes metrics from instrumented jobs.
- 1 x Grafana pod - provides visualization for the collected data.
- 1 x Alertmanager pod - handles alerts sent by the Prometheus server.

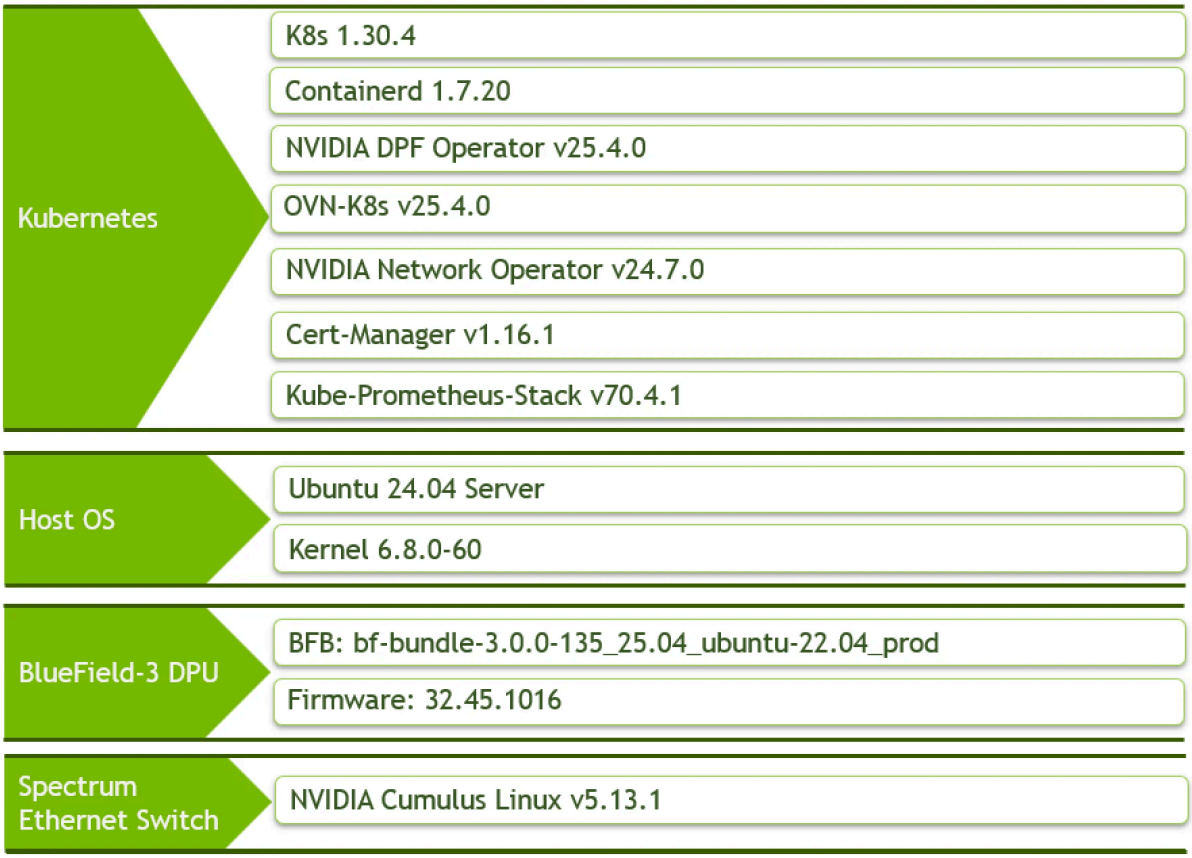
The entire monitoring stack is deployed and managed using the **kube-prometheus-stack** Helm chart. Each pod is deployed as a StatefulSet, which also manages the PVCs providing persistent storage. Using service discovery (DNS based in this example), and by configuring the DTS DPUService to expose its Prometheus endpoint port to the host cluster, the Prometheus server can automatically detect every DTS instance in the DPU K8s cluster and pull metrics from it.

✓ Note

- A more detailed explanation about the solution is provided in the following sections of the RDG.
- The DPU K8s control plane is omitted from this scheme to simplify the view. For further details about the K8s cluster design, refer to **RDG for DPF with OVN-Kubernetes and HBN Services - K8s Cluster Logical Design**.



Software Stack Components



ⓘ Warning

Make sure to use the **exact same versions** for the software stack as described above.

Bill of Materials

The bill of materials is based upon the same hardware as demonstrated in **RDG for DPF with OVN-Kubernetes and HBN Services - Bill of Materials**.

Deployment and Configuration

Node and Switch Definitions

Refer to **RDG for DPF with OVN-Kubernetes and HBN Services - Node and Switch Definitions**.

Wiring

Refer to **RDG for DPF with OVN-Kubernetes and HBN Services - Wiring**.

Fabric Configuration

Refer to **RDG for DPF with OVN-Kubernetes and HBN Services - Fabric Configuration**.

DTS Upgrade to Configure ConfigPorts

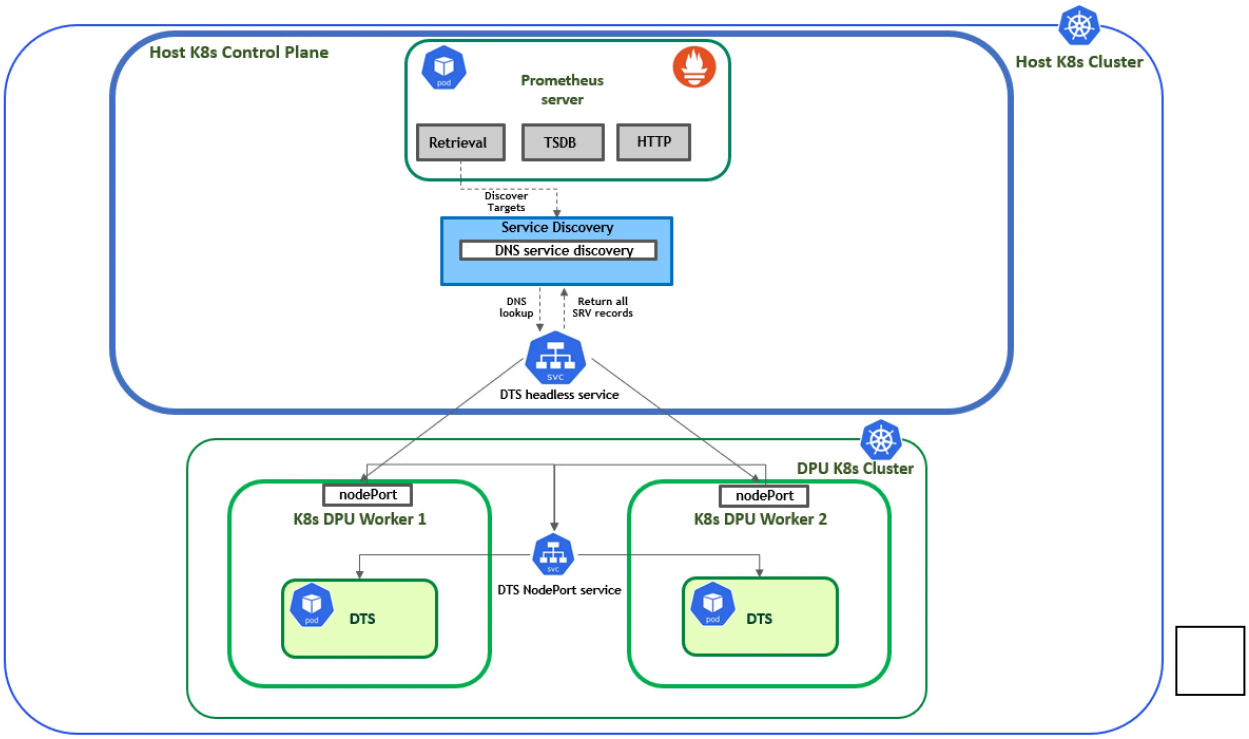
The following section explains how to leverage the DNS capabilities provided by Kubernetes to obtain a dynamic list of all the replicas for a service. This allows Prometheus to be kept informed automatically about which DTS instances it needs to scrape, without needing to statically reconfigure it every time an additional DTS instance is added to the K8s cluster.

To achieve this, several configurations are required:

- A headless service on the host cluster, which in turn will create an SRV record, allowing Prometheus to utilize its DNS-based service discovery feature.
- The DTS Prometheus endpoint port (`9100` by default) needs to be exposed to the host cluster via a NodePort service.
- An EndPointSlice to back the headless service on the host cluster with the DPU IPs as its endpoints and the nodePorts values as its port.

Fortunately, all of these can be configured using the `ConfigPorts` field for the DTS DPUService. For more information on this feature, refer to: **dpuservice-configPorts** .

The following illustration demonstrates the explanation provided above:



Proceed with the following configuration:

1. Upgrade the DTS DPUService using the following configuration:

manifests/05-dpudeployment-installation/dpuserviceconfig_dts.yaml Collapse Source

```
---
apiVersion: svc.dpu.nvidia.com/v1alpha1
kind: DPUServiceConfiguration
metadata:
  name: dts
  namespace: dpf-operator-system
spec:
  deploymentServiceName: "dts"
  serviceConfiguration:
    configPorts:
      serviceType: None
      ports:
        - name: httpserverport
          protocol: TCP
          port: 9100
```

manifests/05-dpudeployment-installation/dpuservicetemplate_dts.yaml Collapse Source

```
---
apiVersion: svc.dpu.nvidia.com/v1alpha1
kind: DPUServiceTemplate
metadata:
  name: dts
  namespace: dpf-operator-system
spec:
  deploymentServiceName: "dts"
  helmChart:
    source:
      repoURL: $HELM_REGISTRY_REPO_URL
      version: 1.0.6
      chart: doca-telemetry
    values:
      exposedPorts:
```

```
        ports:
          httpserverport: true
```

2. Run the following command:

> **Note**
>
> - The following command assumes the user has cloned the doca-platform Git repository, changed to the necessary directory inside it and defined the variables required for the DPF installation.
> - For more information, check: **RDG for DPF with OVN-Kubernetes and HBN Services - DPF Installation Software Prerequisites and Required Variables**.

Jump Node Console — Collapse Source

```
$ cat manifests/05-dpudeployment-installation/*dts.yaml | envsubst
```

3. Verify that the DTS DPUService is in the ready state and that a headless service has been created on the host cluster:

> **Note**
>
> The following verification commands may need to be run multiple times to ensure the condition is met.

Jump Node Console — Collapse Source

```
$ kubectl wait --for=condition=ApplicationsReady --namespace dpf-c
dpuservice.svc.dpu.nvidia.com/dts-mk55x condition met

$ kubectl get svc -n dpf-operator-system | grep dts
dts-mk55x                                          ClusterIP   No
```

4. Verify that the SRV record is resolvable from the host cluster:

a. In this example the `master1` node is used (since it has an IP in the pod subnet). SSH into the respective node:

Jump Node Console — Collapse Source

```
depuser@jump:~$ ssh master1
```

b. Resolve the headless service SRV record, which should return **all** DTS endpoint SRV records (2 in this example):

> ✓ **Note**
>
> Replace `dts-mk55x` with your service name.

Master1 Console — Collapse Source

```
depuser@master1:~# dig srv _httpserverport._tcp.dts-mk55x.dpf-c
0 50 30342 worker1-0000-89-00.dts-mk55x.dpf-operator-system.svc
0 50 30342 worker2-0000-89-00.dts-mk55x.dpf-operator-system.svc
```

Setup Centralized Monitoring Stack

Prometheus is a monitoring platform that collects metrics from monitored targets by scraping metrics HTTP endpoints on these targets.

Grafana is an open-source software which allows users to query , visualize, alert on, and explore their metrics, logs, and traces wherever they are stored and turn TSDBs data into insightful graphs and visualizations.

In this RDG, the monitoring stack will be installed using the **kube-prometheus-stack** Helm chart.

This chart installs the core components of the **kube-prometheus stack**, including a collection of Kubernetes manifests, Grafana dashboards, Prometheus rules, and documentation and scripts. Together they provide an easy-to-operate, end-to-e monitoring solution for Kubernetes cluster using Prometheus Operator.

1. Add the Prometheus-Community repository and update it:

Jump Node Console — Collapse Source

```
$ helm repo add prometheus-community https://prometheus-community.
$ helm repo update
```

2. The following `kube-prometheus-stack.yaml` values file will be applied:

✓ Note

- The Prometheus server is already configured with **DNS-based service discovery** to automatically discover all DTS instances in the cluster (DNS points to the headless service SRV record created earlier).
- The Prometheus server, Grafana, and Alertmanager StatefulSets are backed by PVCs using the `local-path` StorageClass, each with a size of `10Gi`. By default, the PVCs are retained in case of StatefulSet deletion or scale-down.
- All of the services in the stack are deployed with a service of type `NodePort` for easy access to their UIs from a browser in the jump host.
- All of the pods are configured to run on the control plane nodes with an anti-affinity for better load sharing.

Expand

kube-prometheus-stack.yaml — Collapse Source

```
alertmanager:
  service:
    type: NodePort
  alertmanagerSpec:
    storage:
      volumeClaimTemplate:
        spec:
          storageClassName: local-path
          accessModes: ["ReadWriteOnce"]
          resources:
```

```
            requests:
              storage: 10Gi
    nodeSelector:
      node-role.kubernetes.io/control-plane: ""
    affinity:
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                  - key: app.kubernetes.io/name
                    operator: In
                    values:
                      - prometheus
                      - grafana
              topologyKey: kubernetes.io/hostname
    tolerations:
      - key: node-role.kubernetes.io/control-plane
        operator: Exists
        effect: NoSchedule
      - key: node-role.kubernetes.io/master
        operator: Exists
        effect: NoSchedule
grafana:
  persistence:
    enabled: true
    storageClassName: "local-path"
  nodeSelector:
    node-role.kubernetes.io/control-plane: ""
```

3. Install the kube-prometheus-stack Helm chart using the following command:

Jump Node Console — Collapse Source

```
$ helm install --create-namespace --namespace kube-prometheus-stac
```

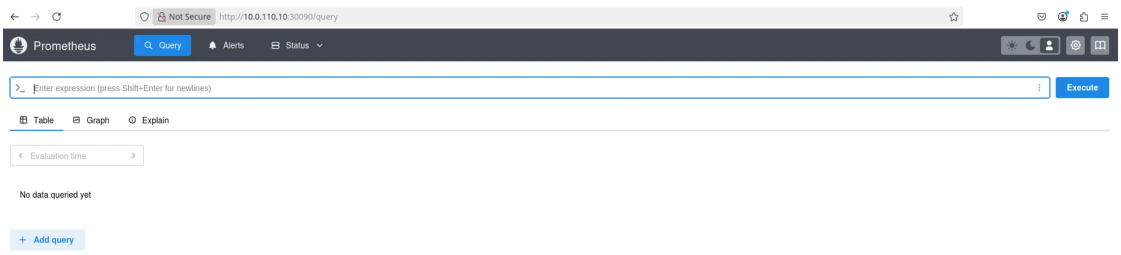
4. Verify that all the pods in the kube-prometheus-stack namespace are in ready state:

Jump Node Console — Collapse Source

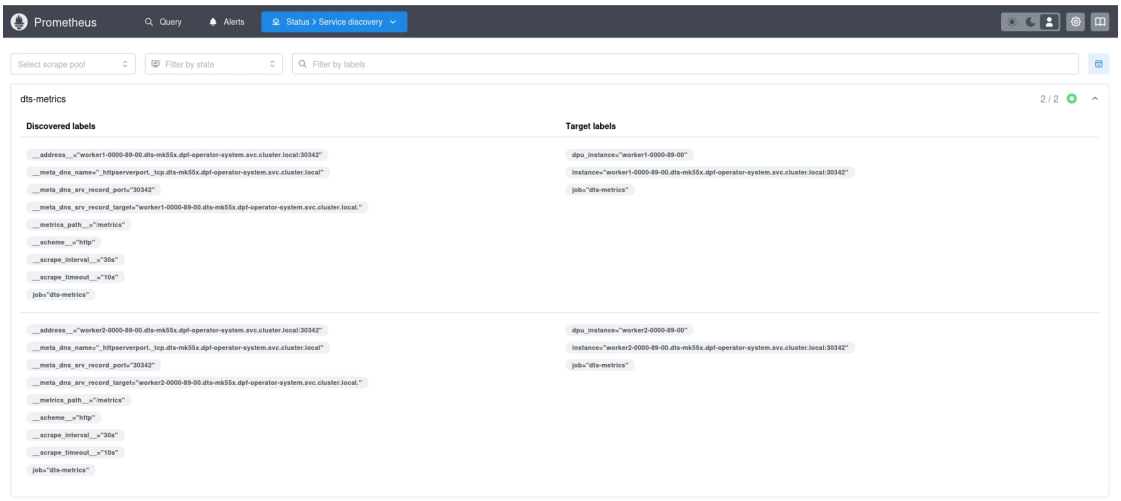
```
$ kubectl wait --for=condition=ready --namespace kube-prometheus-s
pod/alertmanager-kube-prometheus-stack-alertmanager-0 condition me
pod/kube-prometheus-stack-grafana-0 condition met
pod/kube-prometheus-stack-operator-584fccf98d-w8hnc condition met
pod/prometheus-kube-prometheus-stack-prometheus-0 condition met
```

5. Verify in the Prometheus UI that the DNS service discovery works well.
   a. Enter an RDP session, open a web browser, and enter **http://<TARGETCLUSTER_API_SERVER_HOST>:30090** to access the Prometheus web UI:

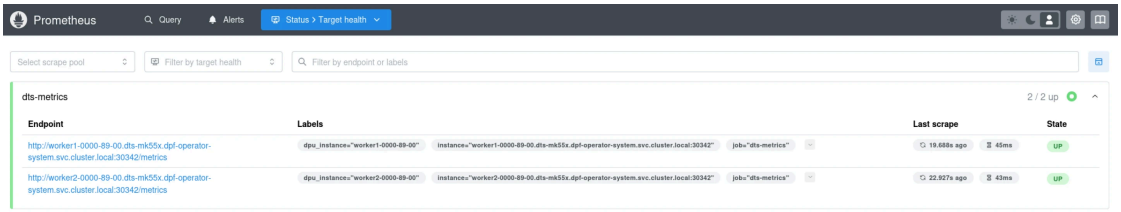
> **Info**
>
> - By default, in the `kube-prometheus-stack` chart, port `30090` is the port for `NodePort` type service for Prometheus UI.
> - `10.0.110.10` is the IP address corresponding to the variable `TARGETCLUSTER_API_SERVER_HOST` in this RDG.



   b. Navigate to Status → Service Discovery. You should see something similar to the following under `dts-metrics` job:



c. Navigate to Status → Target health to verity both DTS endpoints are in the **'UP'** state under `dts-metrics` job:



Display metrics in Grafana

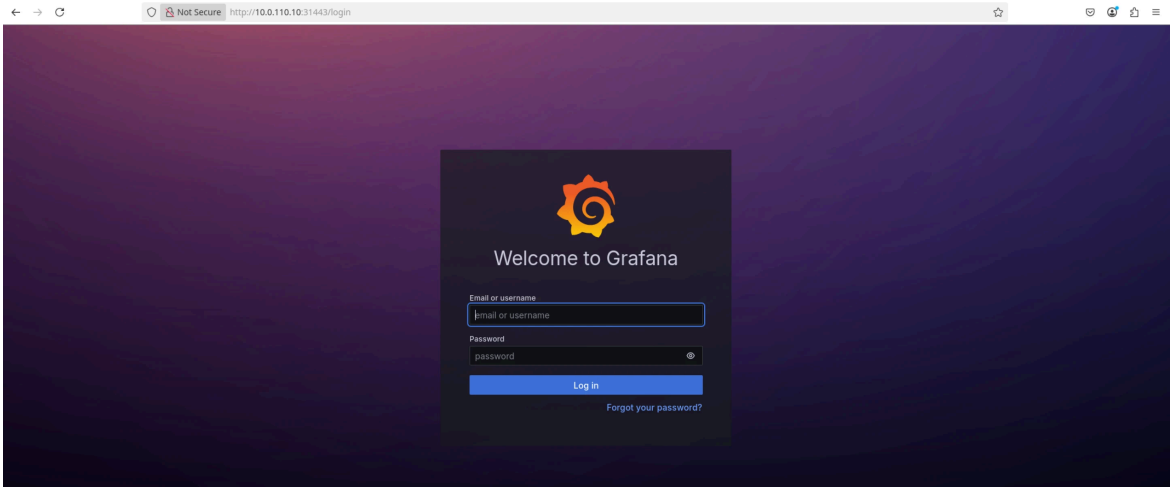
To view the metrics DTS exposed by Grafana and construct useful monitoring graphs, access the Grafana UI:

1. Find out the `nodePort` of the Grafana `NodePort` service:

Jump Node Console — Collapse Source

```
$ kubectl get svc -n kube-prometheus-stack kube-prometheus-stack-g
NAME                                    TYPE       CLUSTER-IP      EXTERNA
kube-prometheus-stack-grafana   NodePort   10.233.29.146   <none>
```

2. In the RDP session, open a web browser and enter **http://<TARGETCLUSTER_API_SERVER_HOST>:<Grafana-svc-nodePort>**(`10.0.110.10` and `31443` respectively in this example):



3. Enter the admin login credentials to access the Grafana home page:
   a. To obtain the Grafana secret name in the `kube-prometheus-stack` operator namespace, run:

Jump Node Console — Collapse Source

```
$ kubectl get secrets -n kube-prometheus-stack | grep grafana
NAME
kube-prometheus-stack-grafana
```

   b. Run the following command to obtain the admin username:

Jump Node Console — Collapse Source

```
$ kubectl get secrets -n kube-prometheus-stack kube-prometheus-
```

   c. Output example:

Jump Node Console — Collapse Source

```
admin
```

   d. Run the following command to obtain the admin password:

Jump Node Console Collapse Source

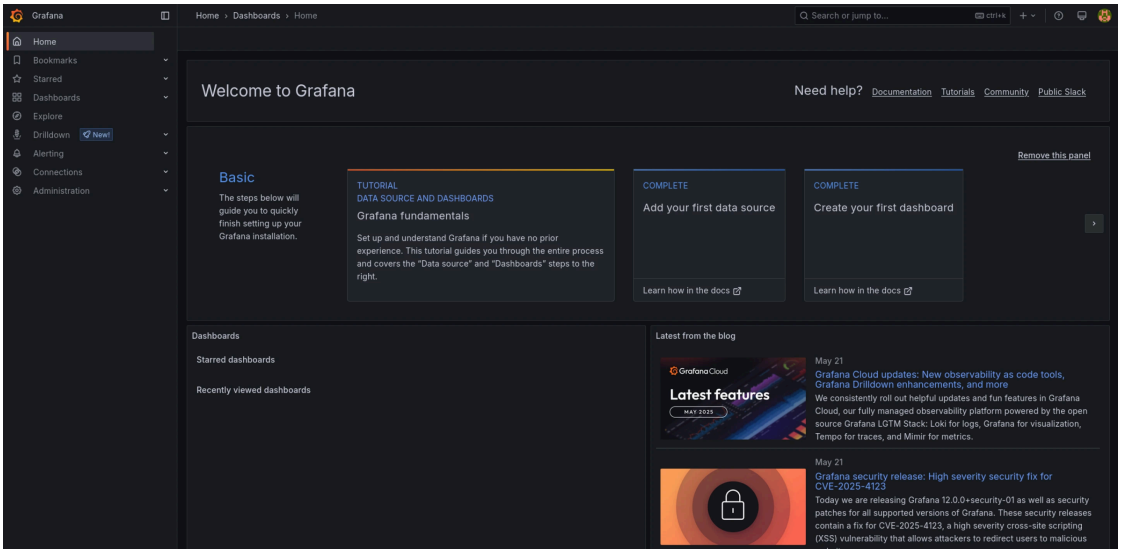
```
$ kubectl get secrets -n kube-prometheus-stack kube-prometheus-
```

e. Output example:

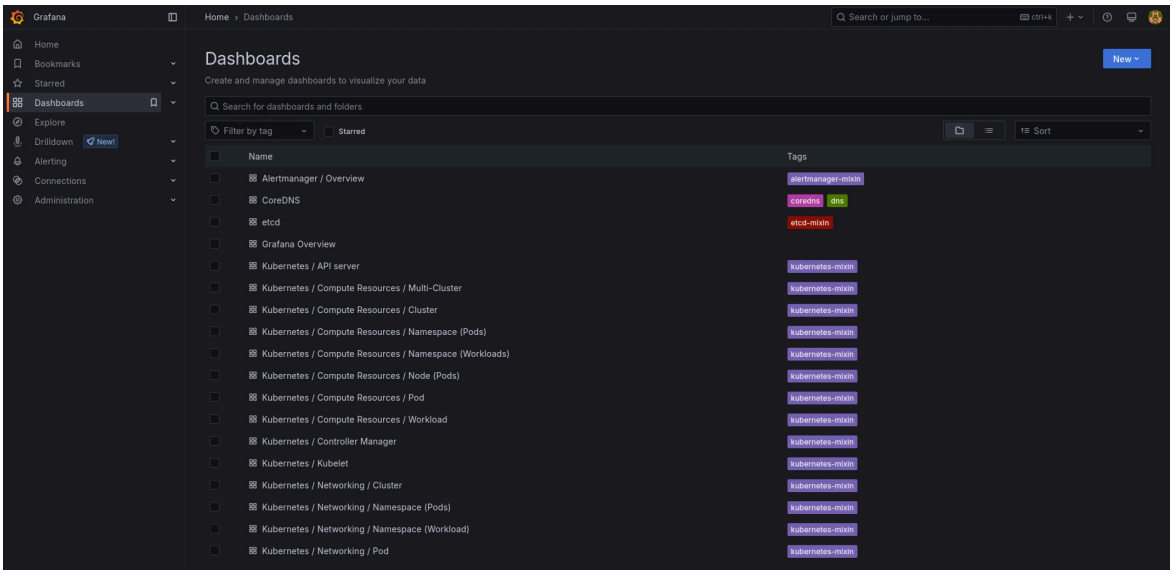
Jump Node Console Collapse Source

```
prom-operator
```

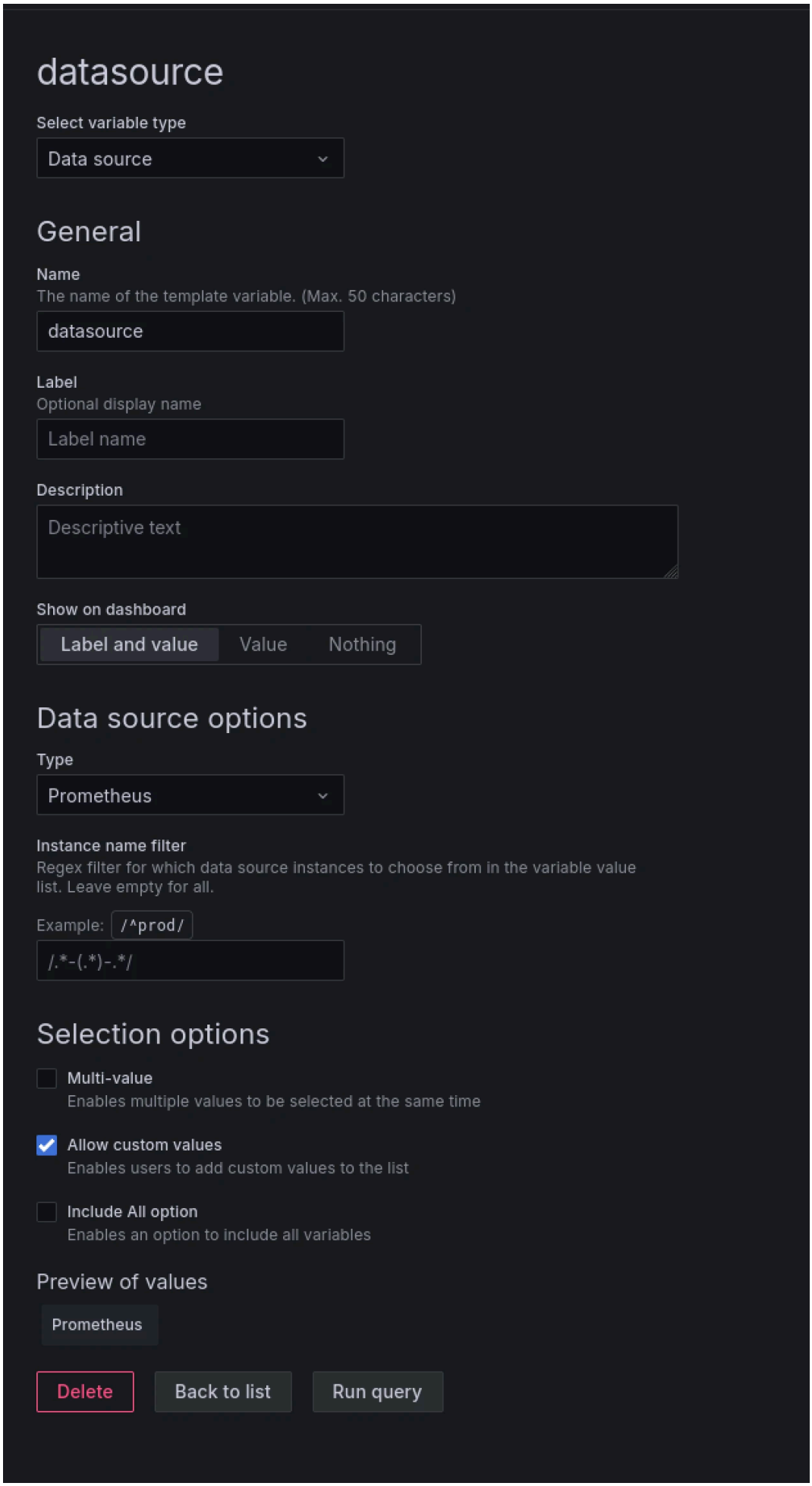
f. Return to the login page and enter the credentials previously obtained:



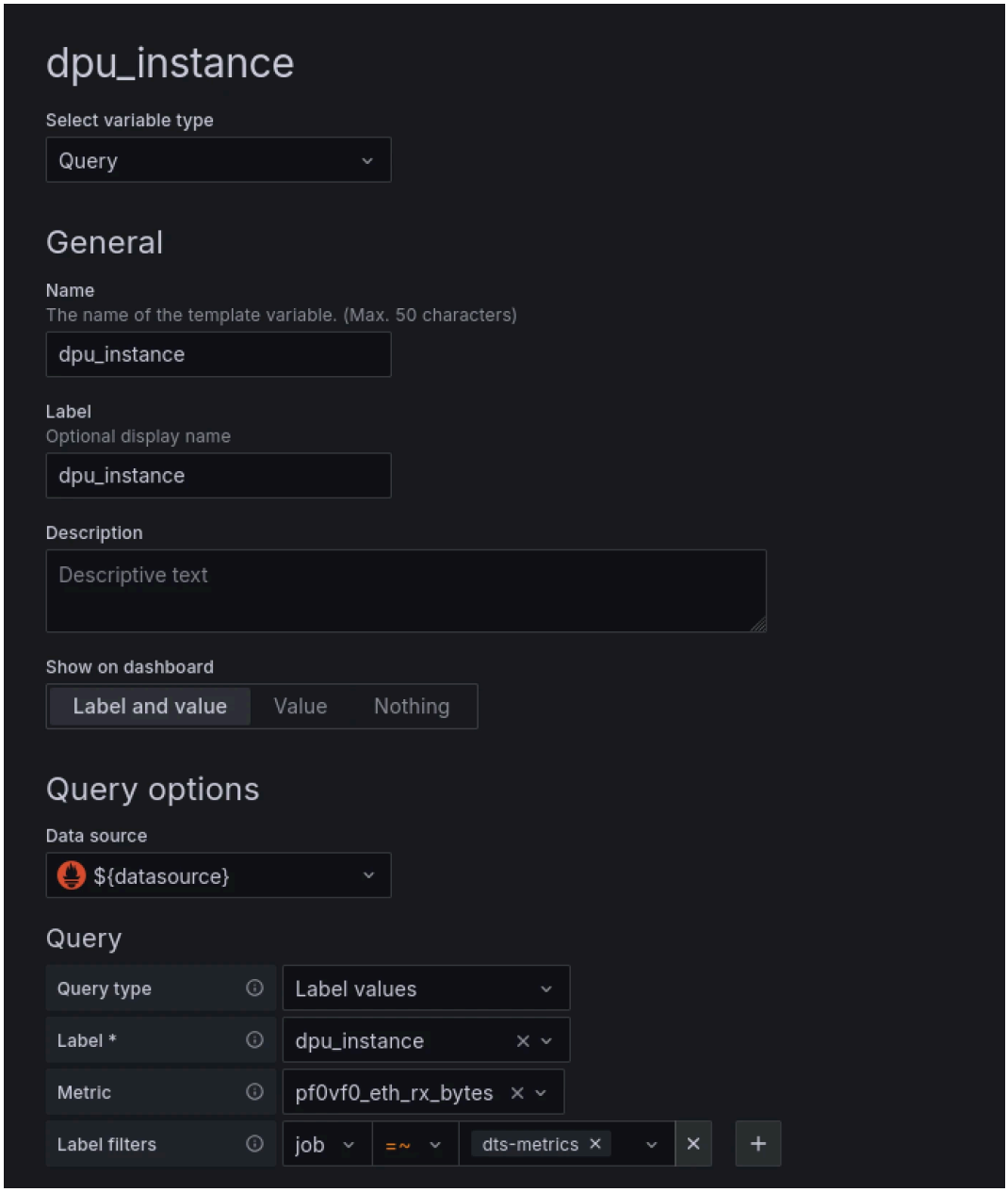
4. Navigate to the Dashboards page where pre-configured dashboards installed by the kube-prometheus-stack helm chart are already available:



5. Click on New → New Dashboard → Add visualization → Select Prometheus as Data Source. After that, start adding panels based on different DTS metrics. For instance:
    a. Click **Back to dashboard** in the top-right corner of the new panel.
    b. Click **Settings** in the top-right corner of the new dashboard.
    c. Go to the **Variables** tab.
    d. Add the following variables:
        i. "Select variable type": `Data source`, "General": ("Name": `datasource`), "Data source options": ("Type": `Prometheus`)


datasource
Select variable type
Data source
General
Name
The name of the template variable. (Max. 50 characters)
datasource
Label
Optional display name
Label name
Description
Descriptive text
Show on dashboard
Label and value
Value
Nothing
Data source options
Type
Prometheus
Instance name filter
Regex filter for which data source instances to choose from in the variable value list. Leave empty for all.
Example: /^prod/
/.*-(.*)-.*/
Selection options
Multi-value
Enables multiple values to be selected at the same time
Allow custom values
Enables users to add custom values to the list
Include All option
Enables an option to include all variables
Preview of values
Prometheus
Delete
Back to list
Run query

ii. "Select variable type": `Query`, "General": ("Name": `dpu_instance`, "Label": `dpu_instance`), "Query options": ("Data source": `$(datasource)`, "Query": ("Query type": `Label values`, "Label": `dpu_instance`, "Metric": `pf0vf0_eth_rx_bytes`, "Label filters": `job =~ dts-metrics`))



e. Return to the main dashboard page and click **Edit** on the previously added empty panel.

f. Configure the new panel as follows:

i. Under the 1st query row (marked as **'A'** by default), switch from **Builder** to **Code.**

ii. Enter the following query to display the average rate of received bits per second in the last 5 minutes:

```
                                                                    
```

Collapse Source

Network Received PromQL

```
rate(label_replace({__name__=~".*_eth_rx_bytes", job="dts-me
```

iii. On the right-side of the screen under "Panel options", configure:

- Title: `Network Received`
- Description: `Network received (bits/s)`
- Unit: `bits/sec(SI)`
- Min: `0`

iv. Click **Run queries** to display the data in the panel.

g. Click **Back to dashboard** and then choose Add → Visualization.

h. Configure an additional panel:

i. Run the following query to display the average rate of transmitted bits per second in the last 5 minutes:

Collapse Source

Network Transmitted PromQL

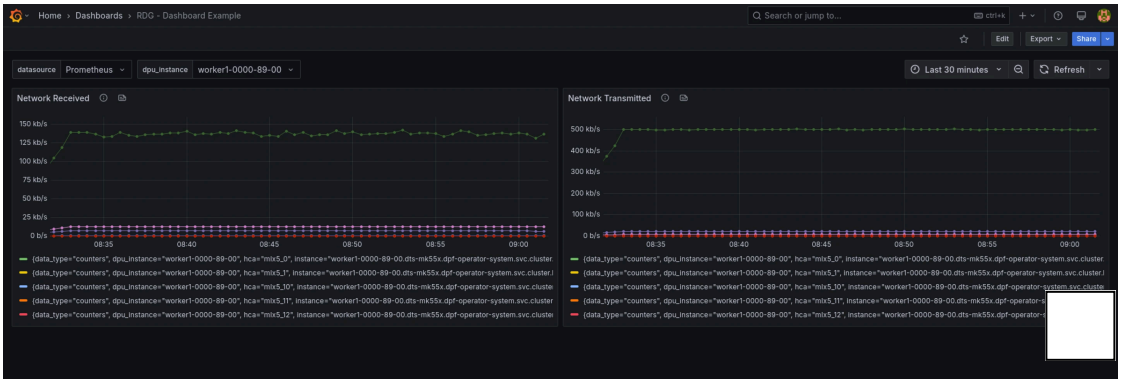
```
rate(label_replace({__name__=~".*_eth_tx_bytes", job="dts-me
```

ii. On the right-side of the screen under "Panel options", configure:

- Title: `Network Transmitted`
- Description: `Network transmitted (bits/s)`
- Unit: `bits/sec(SI)`
- Min: `0`

iii. Click **Run queries** to display the data in the panel.

i. Click **Back to dashboard** and align the panels:



j. Click **Save dashboard** in the top-right corner of the dashboard.

Authors

Guy Zilberman

Guy Zilberman is a solution architect at NVIDIA's Networking Solution s Labs, bringing extensive experience from several leadership roles in cloud computing. He specializes in designing and implementing solutions for cloud and containerized workloads, leveraging NVIDIA's advanced networking technologies. His work primarily focuses on open-source cloud infrastructure, with expertise in platforms such as Kubernetes (K8s) and OpenStack.

Notice

Last updated on May 22, 2025