Weighing Options with Prometheus and gNMI

NANOG91 - JUN-2024

Mau Rojas

github.com/cloud-native-everything

Abstract

- Overview of Prometheus and gNMI for network monitoring
- Exploring integration options:
  - gNMIc
  - Prometheus client library (i.e. Python)
  - Exporter installed directly in Network Element
  - Unified Telemetry Model (NMS or SDN)
- Dissect advantages and challenges of each option.
- Selecting the best monitoring solution based on network needs and resources.

Prometheus and GNMI

Why Prometheus?

Prometheus's time-series storage, pull model, rich querying, alerting features made it a popular and comprehensive telemetry solution.

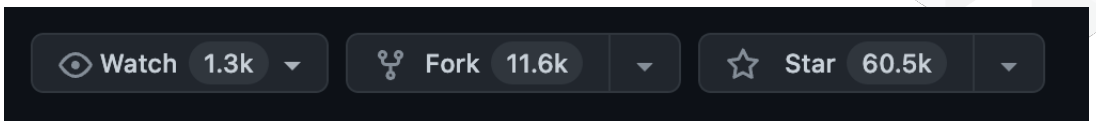
Large and active community, thanks in part to its integration with the Cloud Native Computing Foundation (CNCF). It has a vibrant ecosystem of exporters, integrations, and plugins. Its GitHub repository shows a significant number of contributors and frequent updates, and there are numerous forums, Slack channels, and other resources dedicated to its community

The Stack Prometheus/Grafana is highly popular, lightweight and more container friendly vs other solutions like Zabbix or Nagios.

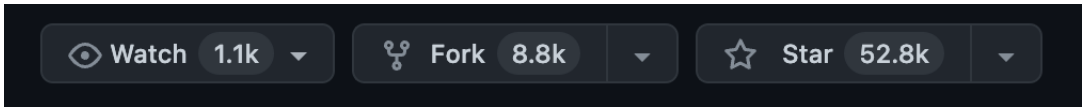
Why Prometheus?

metrics

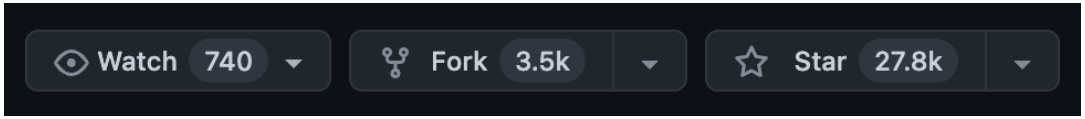
grafana



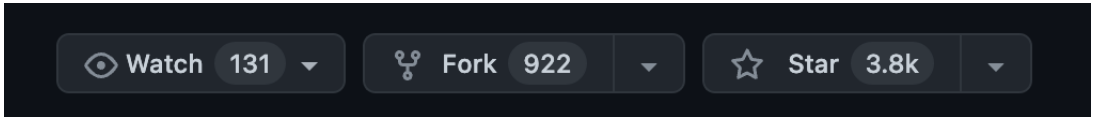
prometheus



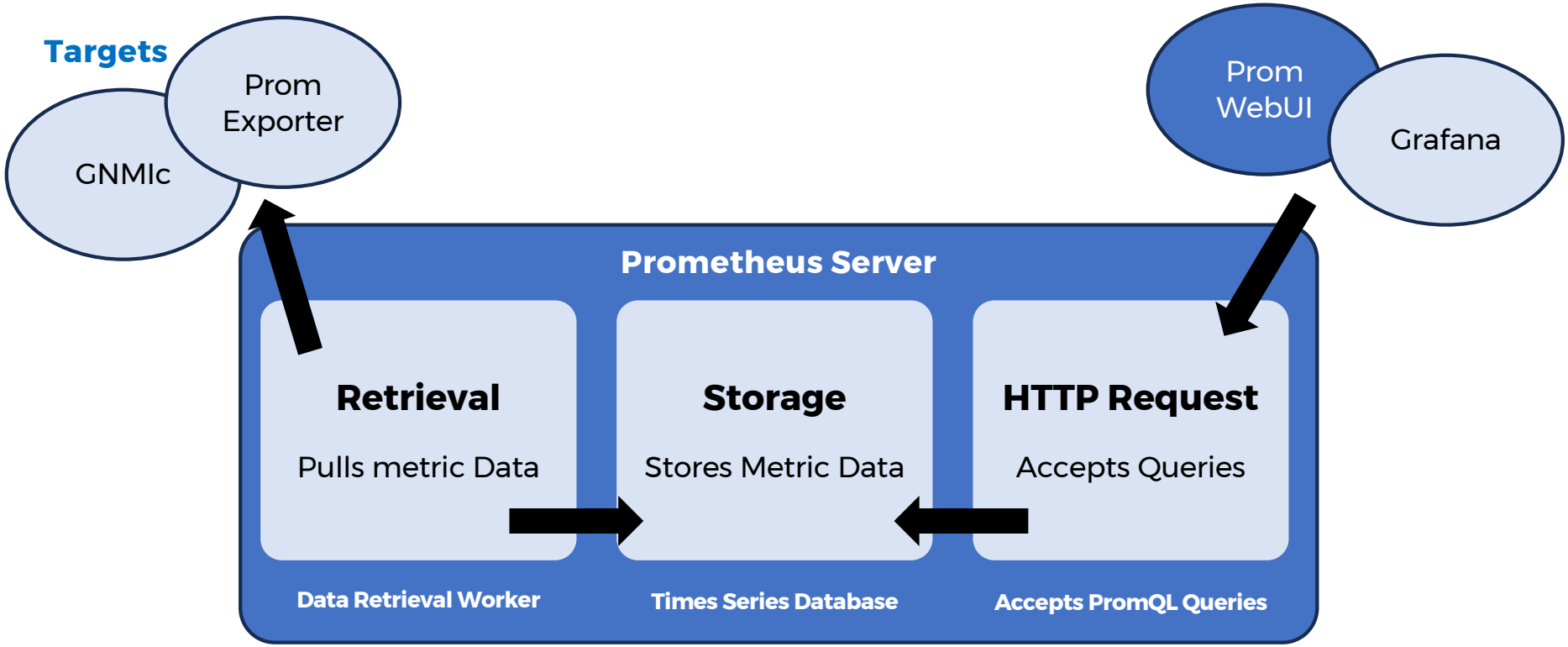
influxdb



zabbix

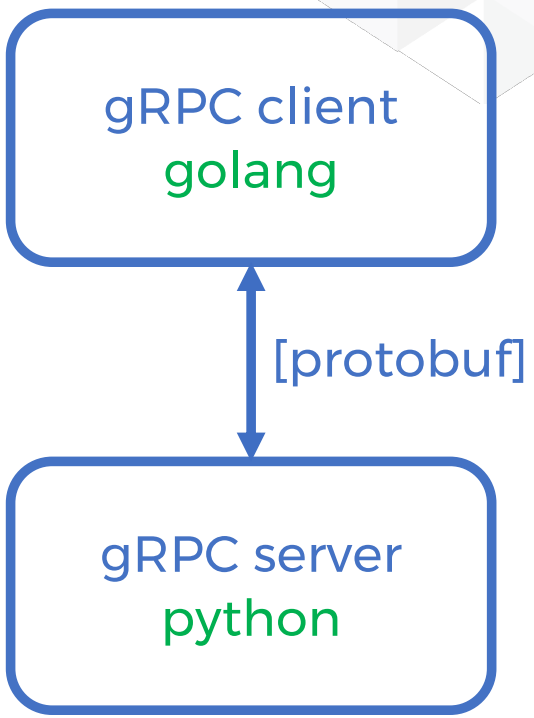


Prometheus Main Components



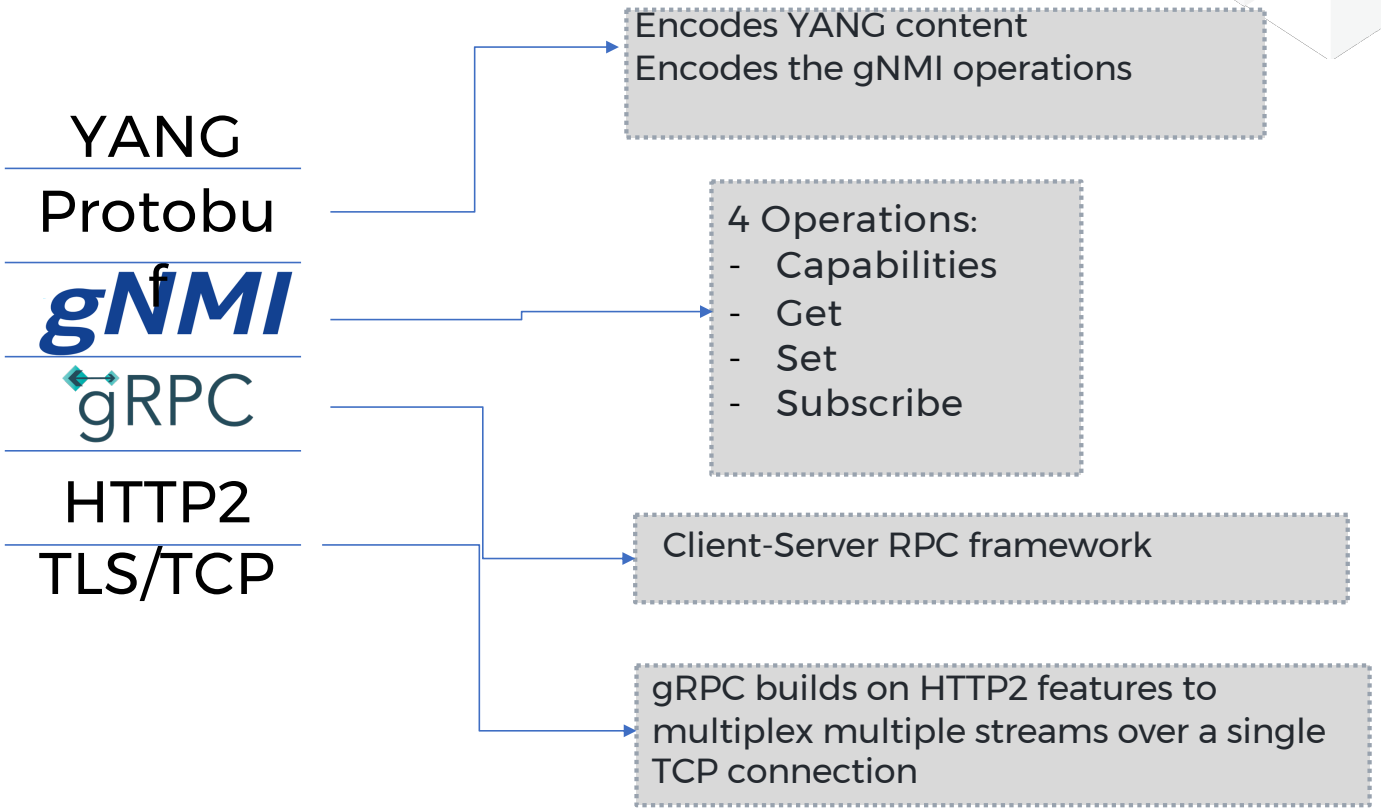
What is GNMI?

- gRPC Network Management Interface
- Open-source protocol developed by Google
- Versatile, efficient, and scalable
  - Protocol buffers
- Retrieve and configure network state information

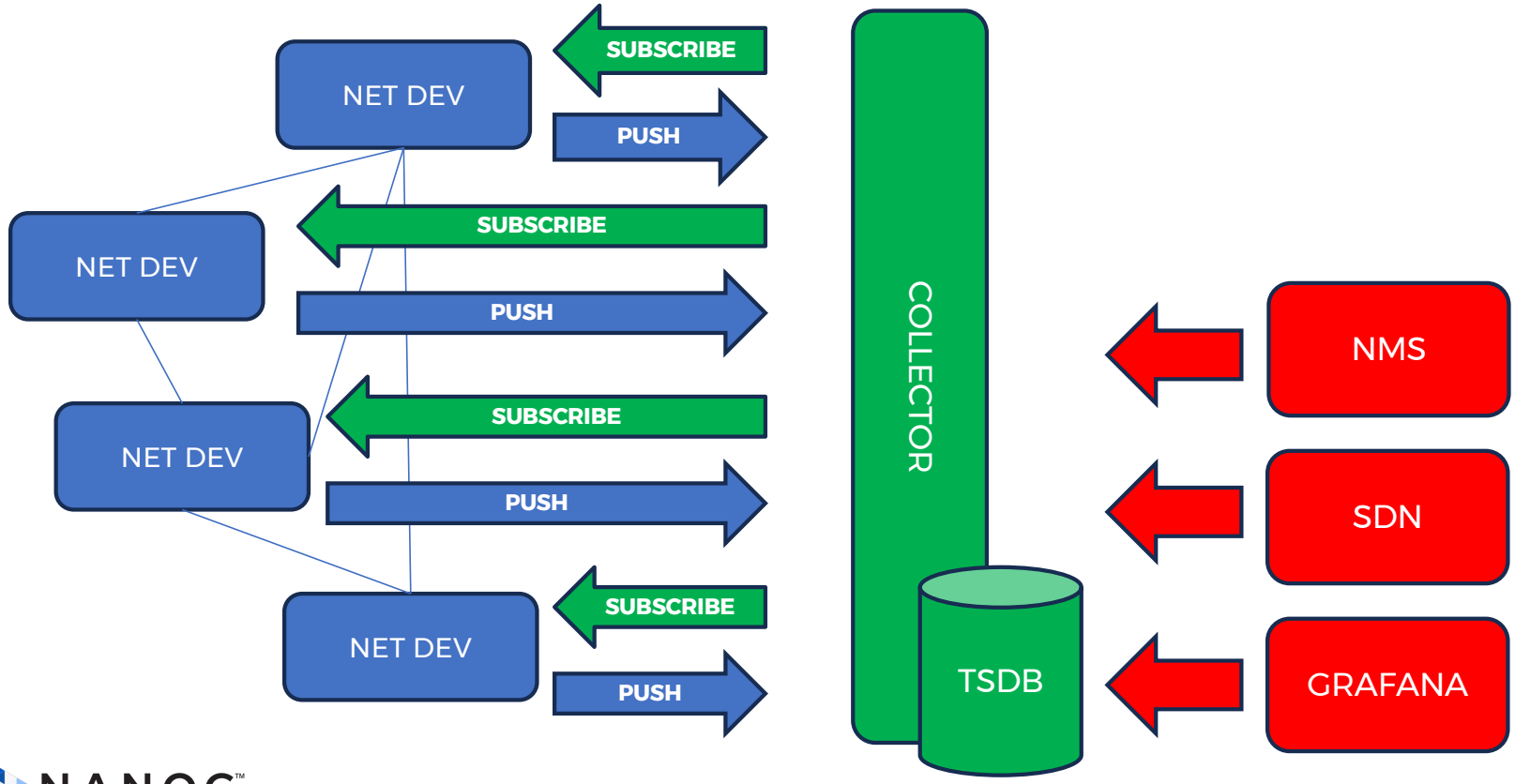


Source: https://gnmic.kmrd.dev/

gRPC Network Management Interface



GRPC Subscription



Integration Options

Integration Options

GNMIc

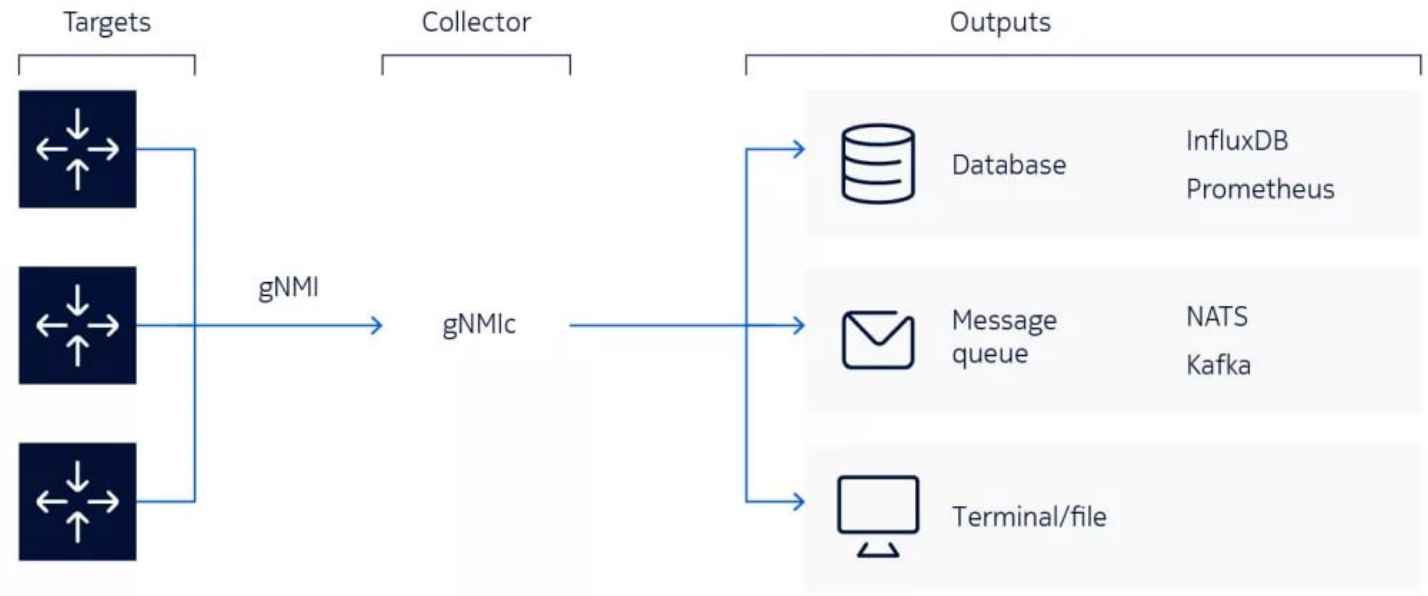
openconfig/gnmic

Exporter installed directly in Network Element

Client Library (ex. Python)

Centralized Exporter with Unified Telemetry Model (NMS or SDN)

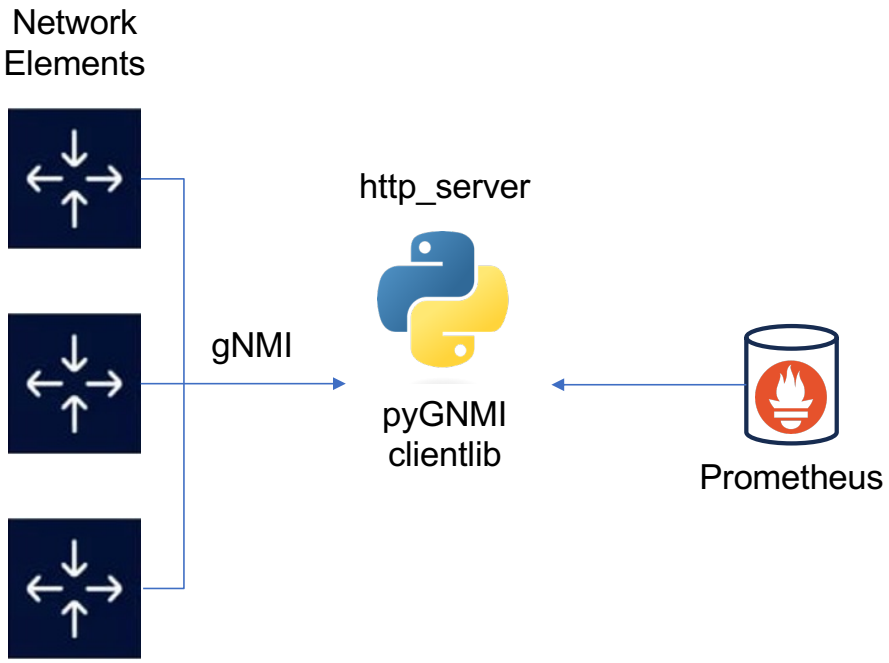
GNMIc as Telemetry Collector



GNMIc as Telemetry Collector

- GNMIc can be installed as a standalone application (e.g., using Docker) or within a Kubernetes cluster (recommended for enhanced resiliency).
- It subscribes to specified targets from which it receives telemetry data. These targets are configured via discovery services such as Consul.
  - Using discovery mode is advisable as it eliminates the need to restart the GNMIc process.
- The data can be output in various compatible formats, including Prometheus and Kafka.

Python Prometheus Client



Python Prometheus Client

- Create your custom collector using GNMI and Prometheus libraries, such as those available in Python.
- You can configure the information to be displayed in the Prometheus Exporter.
  - Establish a unified nomenclature across different devices and vendors.
  - Incorporate information from additional sources, such as weather data or customer support.
- Information is accessible and ready for Prometheus to retrieve.

Python Prometheus Client

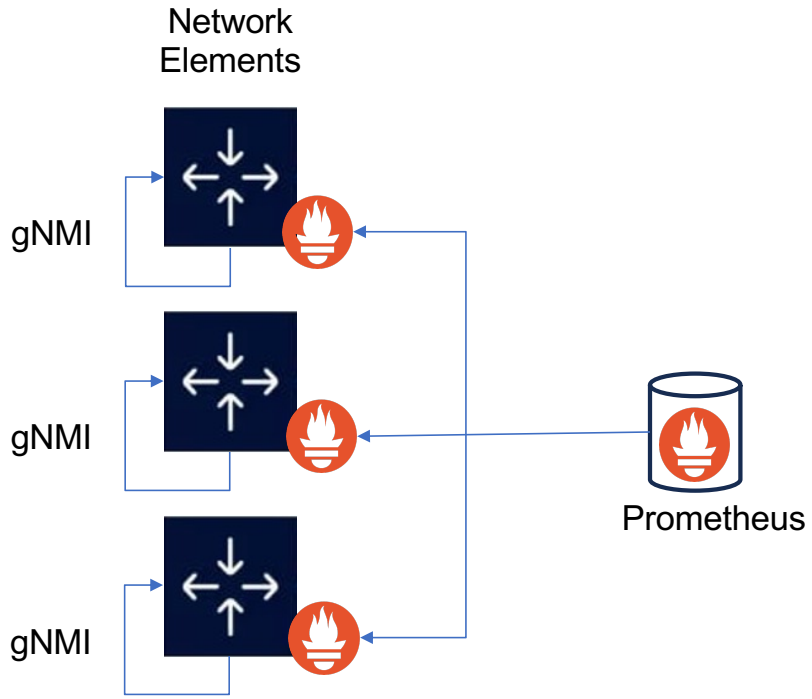
```
from pygnmi.client import gNMIclient, telemetryParser
with gNMIclient(target=(host_entry["ip_address"], host_entry["port"]),
           username=host_entry["username"],
           password=host_entry["password"], insecure=True) as gc:

   telemetry_stream = gc.subscribe(subscribe=subscribe)
```

```
from prometheus_client import start_http_server, Gauge
lsp_ingress_octets = Gauge('lsp_ingress_octets', 'Ingress octets for LSPs')

if __name__ == '__main__':
   start_http_server(8000)  # Port 8000 or any port you prefer
   while True:
        collect_telemetry_data(host_entry, subscribe)
        time.sleep(10)
```

Exporter in Network Element



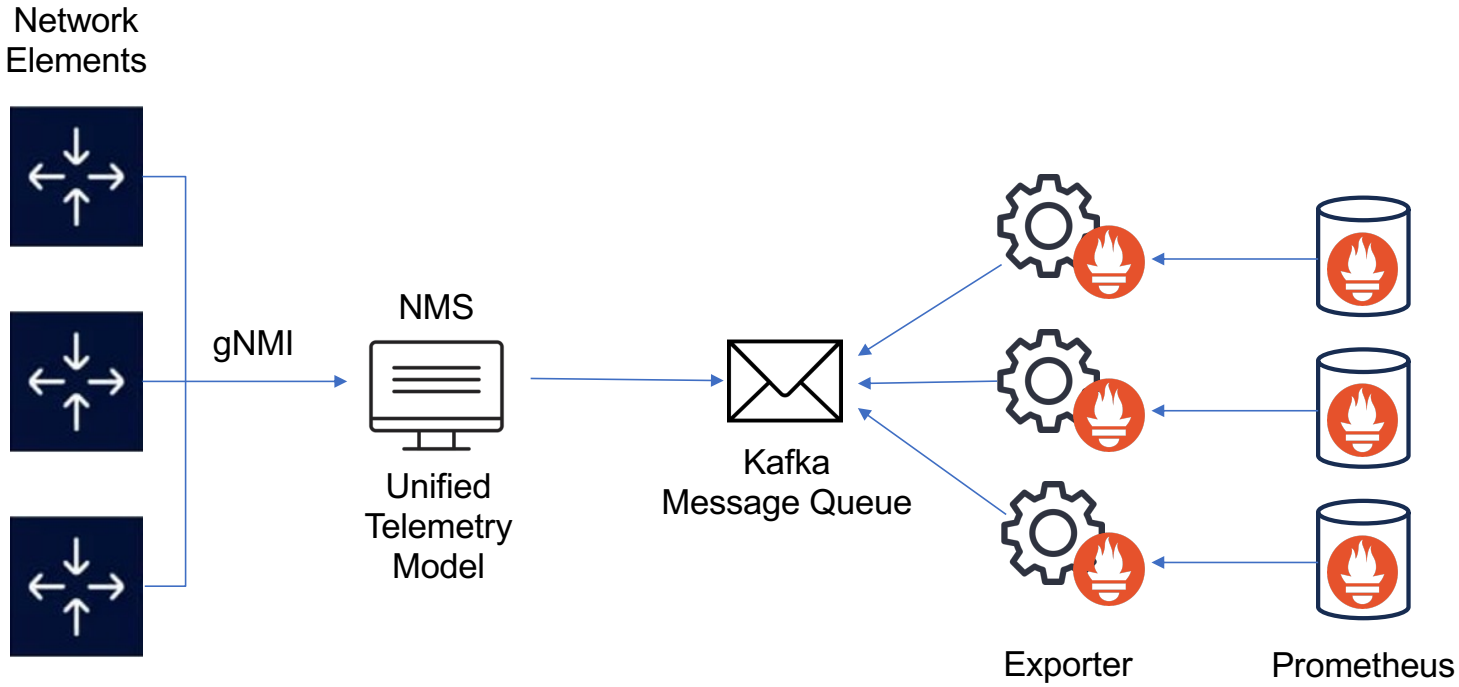
NANOG™

github.com/karimra/srl-prometheus-exporter

Exporter in Network Element

- Open Network Operating Systems (NOS), based on the Linux kernel such as **SONiC** and **SR Linux**.
- Install the Prometheus exporter as a container image or as a package directly on the network element.
- Prometheus retrieves data directly from the target as its source.

Centralized Exporter



NANOG™

github.com/cloud-native-everything/nsp-kafka-prometheus

Centralized Exporter

- An SDN/NMS application retrieves data from targets using GNMI subscriptions.
- The data is then transformed into a unified telemetry YANG model for analysis and visualization, such as the IETF format.
- Multiple external systems, for example, Grafana, can consume this data via RESTCONF, Kafka, or similar elements.
  - Additional coding may be required for integration.

Integration Options Analysis

Advantages

GNMIc (K8s Cluster)
- Container Ready
- Scaling and Resilient (multiple instances)
- Extensions: Processors, Message Queue, Dynamic Discovery

Exporter in Network Element
- High data fidelity
- Lower latency
- Highly Scalable

Python Prometheus Client
- Direct data handling
- High Customization

Centralized Exporter (NMS/SDN)
- Vendor-Neutral for Targets
- Highly Scalable and Resilient
- Vendor Support
- Additional Features: Automation Frameworks, Inventory, Fault Management

Challenges

GNMIc (K8s Cluster)

- Complex Setup (Kubernetes/Container)
- Dealing with Multiple vendors can add higher complexity

Exporter in Network Element

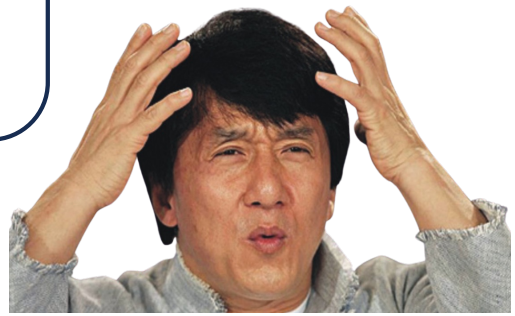
- Limited to specific vendors. Dependent on device capabilities
- Varied performance (uses resources in Network Elements)
- Complex to Maintain (device upgrades)

Python Prometheus Client

- Needs Coding expertise
- Less scale efficiency and Resilience
- Complex to Maintain
- Dealing with Multiple vendors can add even higher complexity

Centralized Exporter (NMS/SDN)

- Complex pipeline
- Reliant on NMS/SDN vendor



OPEX/CAPEX

GNMIc (K8s Cluster)
- Important initial CAPEX
- Lower OPEX through automation

Exporter in Network Element
- Varies, potentially lower CAPEX/OPEX if devices support apps.

Python Prometheus Client
- Lower CAPEX
- Potentially higher OPEX.

Centralized Exporter (NMS/SDN)
- High initial CAPEX
- Lower OPEX with unified management.

Recommended Use Cases

GNMIc (K8s Cluster)
- Large-scale Datacenters
- Containerized network environments like containerlab (service discovery)

Exporter in Network Element
- Datacenter/Edge
- Networks with advanced device capabilities for custom apps.

Python Prometheus Client
- Small/Medium Datacenter/Edge
- Networks needing tailored telemetry solutions

Centralized Exporter (NMS/SDN)
- Large, multi-vendor networks needing unified telemetry and scalability
- Core/Transport Networks

Almost done!

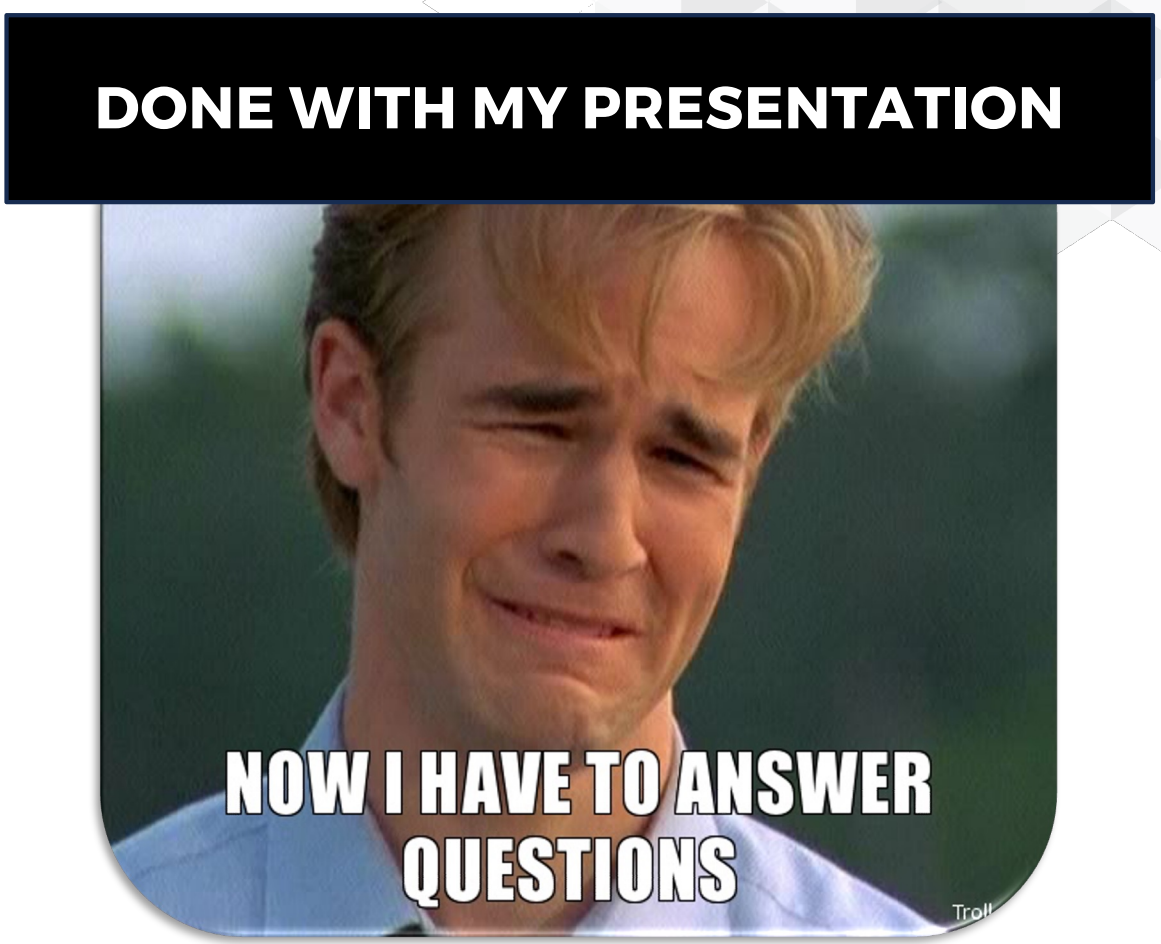
Final Words

- GNMIc is assumed to be running in a containerized environment, which can add to the complexity but also offers scalability and efficient resource usage.
- A custom Python service with PyGNMI and Prometheus_client gives flexibility and control, which could be advantageous for specific custom needs but might not scale as efficiently as containerized or vendor-provided solutions
  - Unless you take the time to containerize it and onboard it into Kubernetes.
- Local Prometheus exporter would be an elegant solution for network devices that allow custom applications, but this is not always the case, and the performance impact on the network device should be carefully considered, plus the operation overhead to maintain it.
- An NMS/SDN controller solution provides a vendor-neutral and scalable approach, ideal for large enterprises with a multi-vendor environment. The cost can be justified by the unified model and the scalability features it provides.

Additional resources

- gNMIc – NANOG Talk - An intuitive gNMI CLI and a feature-rich telemetry collector – Karim
  - https://youtu.be/v3CL2vrGD_8
- pyGNMI and ChatGPT to troubleshoot EVPN Datacenter Fabrics
  - https://youtu.be/dyY4PUFV2nw
- Kubernetes 101 for Network Professionals
  - https://youtu.be/n2kgApcXij0

Thanks!



NANOG™