MONITORING METRICS WITH PROMETHEUS IN GO

Vinh Nguyen

Agenda

01 - Monitoring

03 - Metrics

02 - Prometheus

04 - Visualizations

Technology Used

docker

Prometheus

GO

Grafana

kubernetes

HELM

What is Monitoring

- Monitoring is how we use tools to monitor our *systems* and *applications*
- Monitoring will provide us with values about the *performance*, *health of the system*, and *application* so that we can quickly detect problems with the system
- We will have two primary objects: ***System*** and ***Business***

System Monitoring

- **Infrastructure monitoring**: CPU, memory, disk space
- **Application monitoring**: application's response time, application's status, number of requests per sec

Infrastructure Monitoring

Grafana

How Infrastructure Monitoring Works

Agent Installation

Data Collection

Data Processing & Analysis

Visualization & Reporting

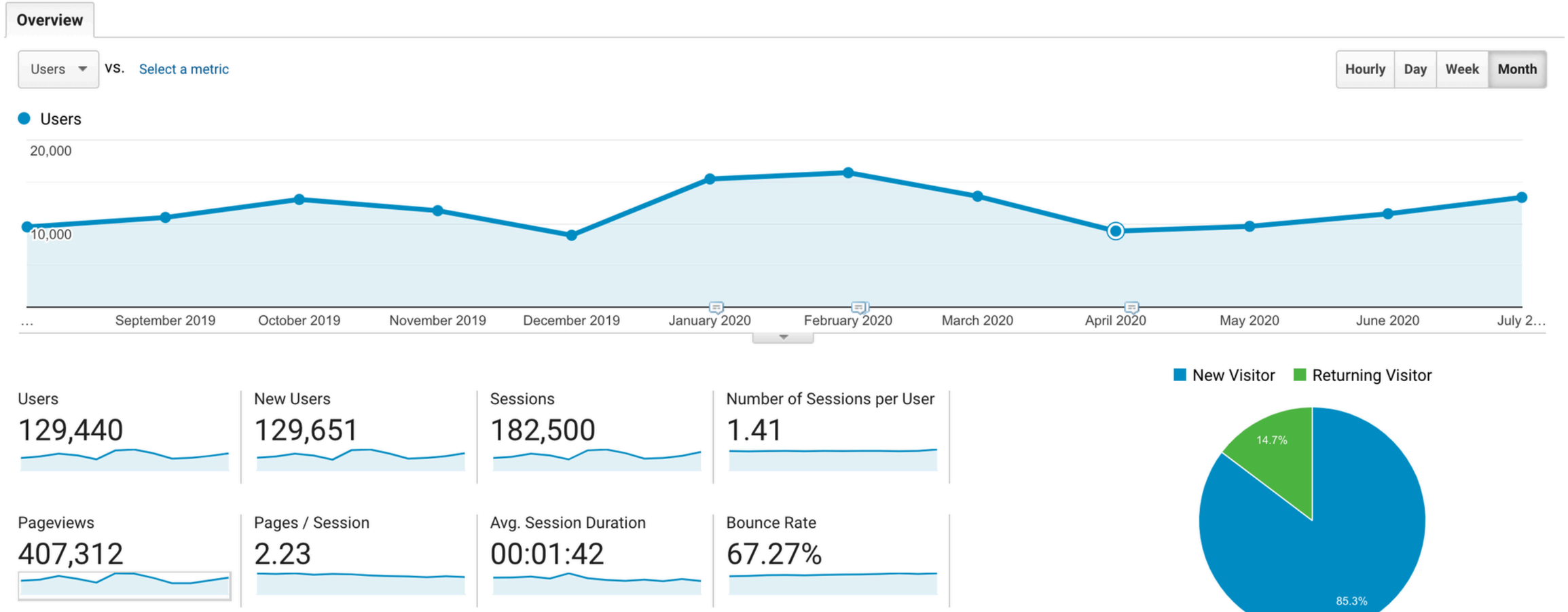
Business Monitoring

- Business-related values are usually values of user interaction, with actions like *clicking buttons*, *scrolling*, *filling out forms*

- For example, at the Frontend layer to track user behavior we will use **Google Analytics**

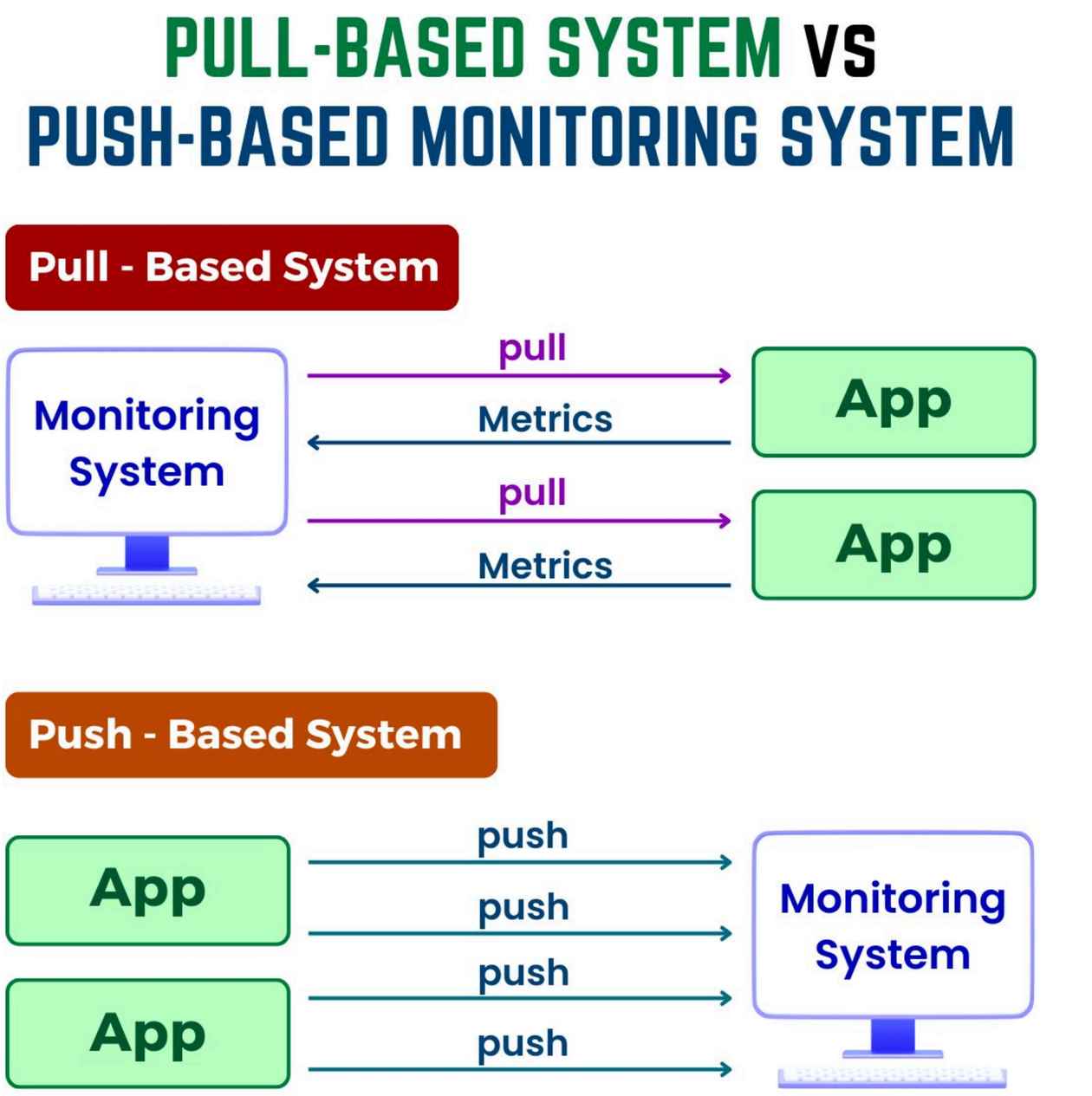
Google Analytics

Google Analytics



Google Analytics

How monitoring tools collect data



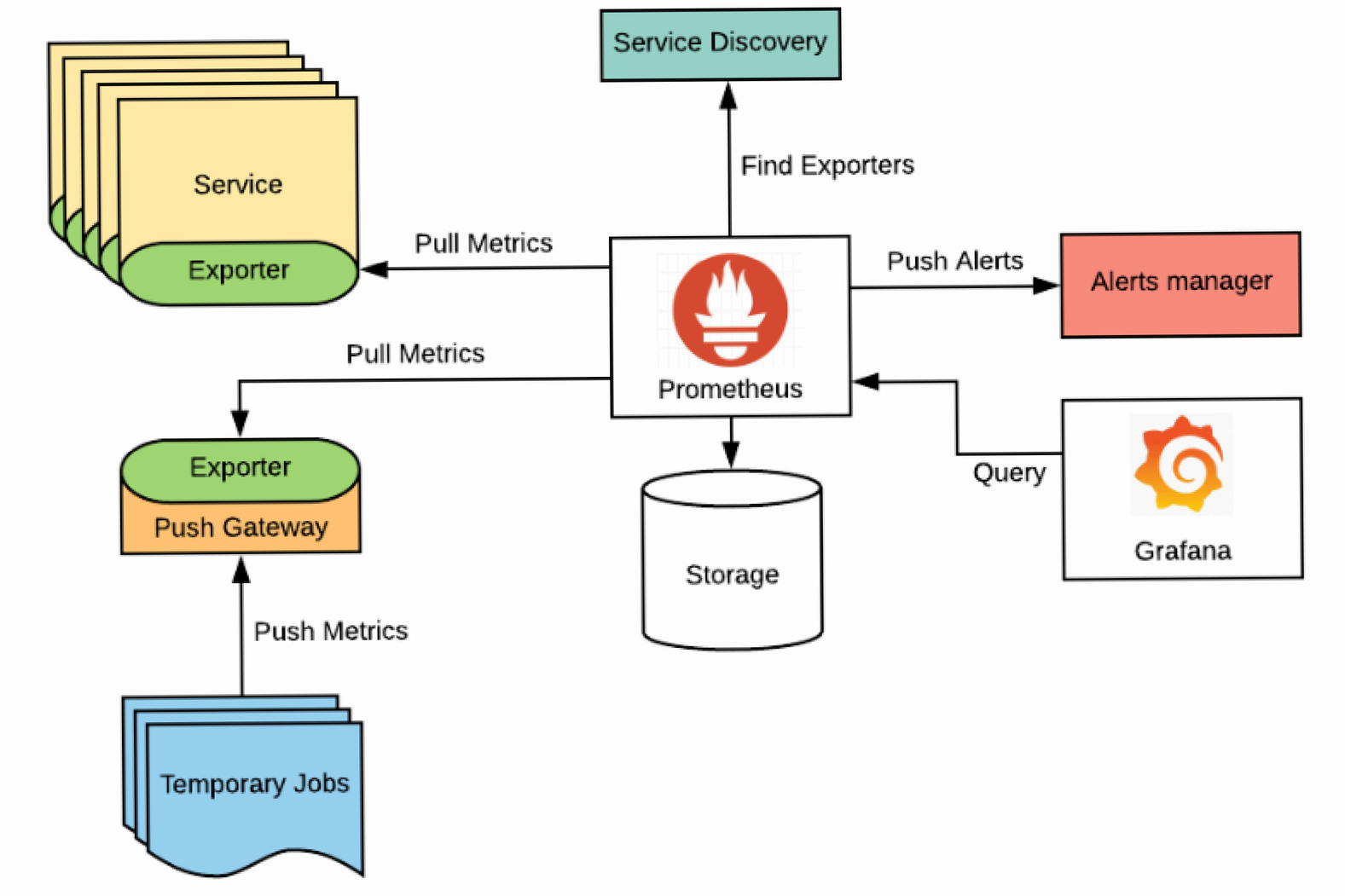
Prometheus

- Prometheus is an open-source systems monitoring and alerting toolkit, written in Go, built by SoundCloud in 2012.

- It **collects metrics data** and **stores** that data in a time series database

Prometheus

Architecture



Prometheus

Features

- Store data in time series
- Query data with PromQL
- Pull-based and Push-based sytem
- Service Discovery
- Interate with other tools for data visualization

Prometheus

Configuration

prometheus.yml

```yaml
# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds
  evaluation_interval: 15s # Evaluate rules every 15 seconds

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
        - targets:
           - alertmanager:9093

# Load rules
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped
  - job_name: "prometheus"
    metrics_path: '/metrics'

    static_configs:
      - targets: ["localhost:9090"]
```

Prometheus

Running on Docker

```
# Create persistent volume for your data
 $ docker volume create prometheus-data

# Start Prometheus container
 $ docker run --name prometheus -p 9090:9090 -d \
    -v /path/to/prometheus.yml:/etc/prometheus/prometheus.yml \
    -v prometheus-data:/prometheus \
    prom/prometheus

# To verify it:
 $ curl localhost:9090/metrics
```

docker

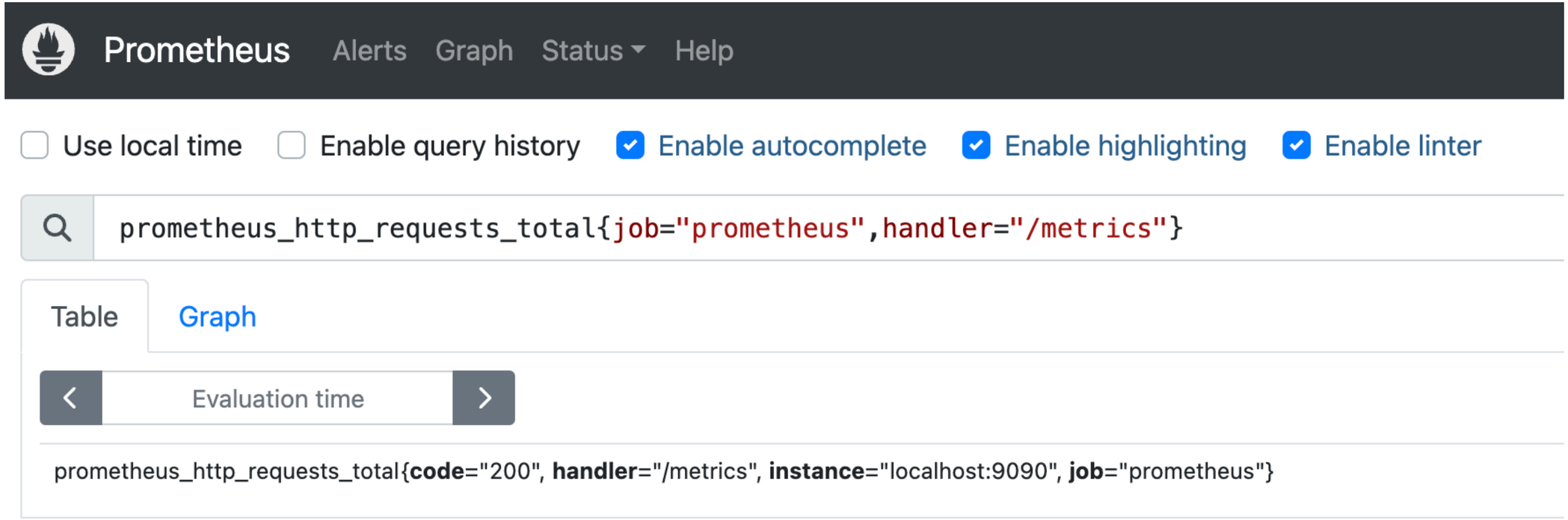
Prometheus

Running on K8s

```
# Start Kubernetes
[x] Enable Kubernetes on Docker Desktop

# Add value.yaml configuration
https://github.com/prometheus-community/helm-charts/blob/main/charts/prometheus/values.yaml

# Create namespace with kubectl
kubectl create ns monitoring

# Install Prometheus in K8s with Helm
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm install -n monitoring prometheus -f values.yaml prometheus-community/prometheus
helm search repo prometheus-community/kube-state-metrics

# Run Port-forward

# To verify it:
curl localhost:9090/metrics
```

Prometheus Web UI

http://localhost:9090



Prometheus

Node Exporter

- Run docker

```
$ docker compose up -d node-exporter
```

- To verify it

```
$ curl localhost:9100/metrics
```

- Query

```
node_cpu_seconds_total{job="node"}
```

Prometheus

Go App

- Run docker

  $ docker compose up -d app

- To verify it

  $ curl localhost:5000/metrics

- Query

  test_golang_metric{instance="goapp:5000", job="goapp"}

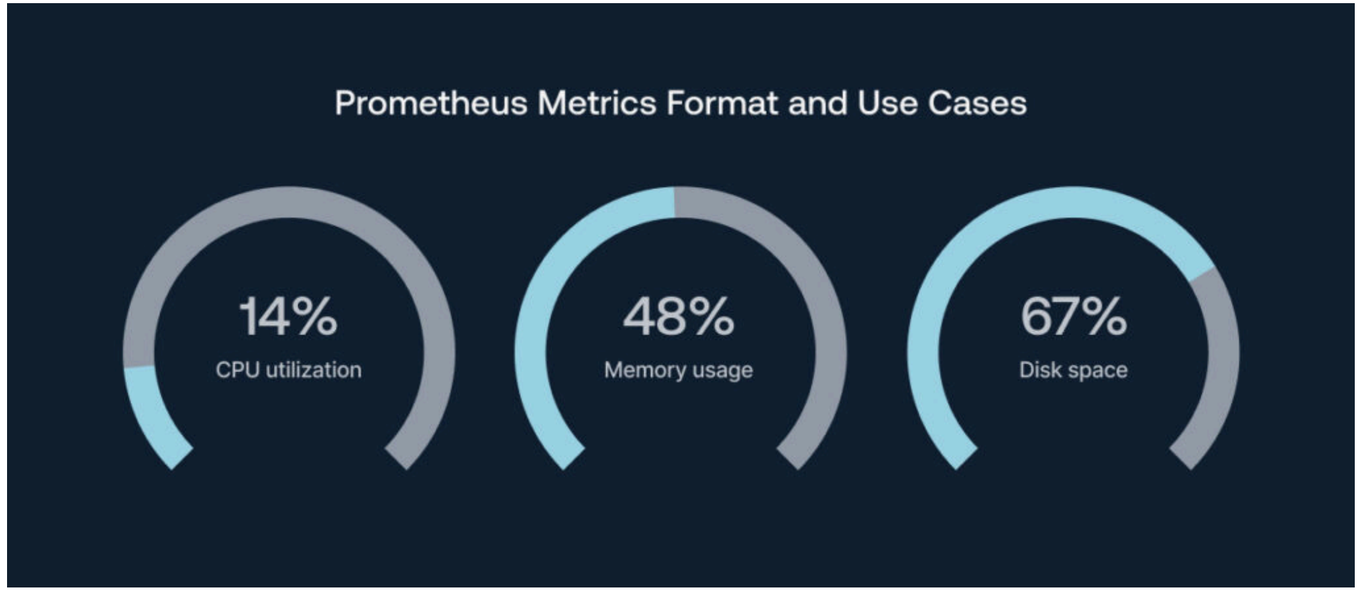
Prometheus

Metrics

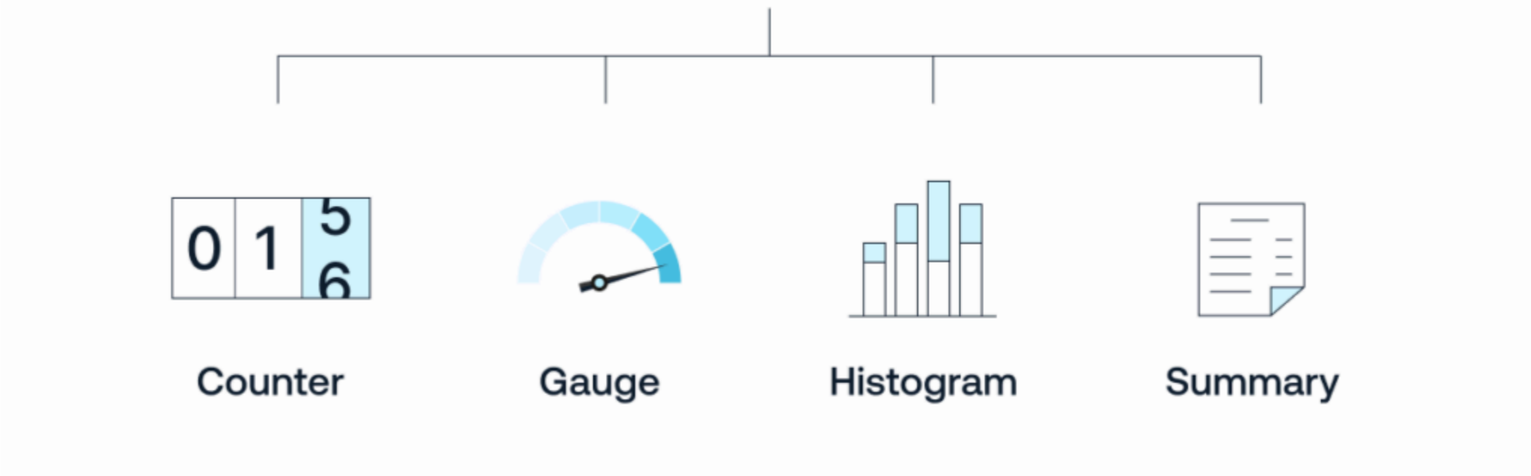
- Metric format

<metric name>{<label name>=<label value>, ...} <samples>

prometheus_http_requests_total{code="200",handler="/metrics"} 67



Prometheus Metrics Types



`prometheus_http_requests_total`
Counter of HTTP requests

`prometheus_http_request_duration_seconds`
Histogram of latencies for HTTP requests

`prometheus_target_metadata_cache_bytes`
The number of bytes that are currently used for storing metric metadata in the cache

`prometheus_engine_query_duration_seconds`
Summary query timings

Example Metrics

Metric	Meaning
rate(node_cpu_seconds_total{mode="system"}[1m])	The average amount of CPU time spent in system mode, per second, over the last minute (in seconds)
node_filesystem_avail_bytes	The filesystem space available to non-root users (in bytes)
rate(node_network_receive_bytes_total[1m])	The average network traffic received, per second, over the last minute (in bytes)

Grafana

Visualizations

Create a Dashboard

- Total request

prometheus_http_requests_total{job="prometheus", handler="/metrics"}

- Node metric

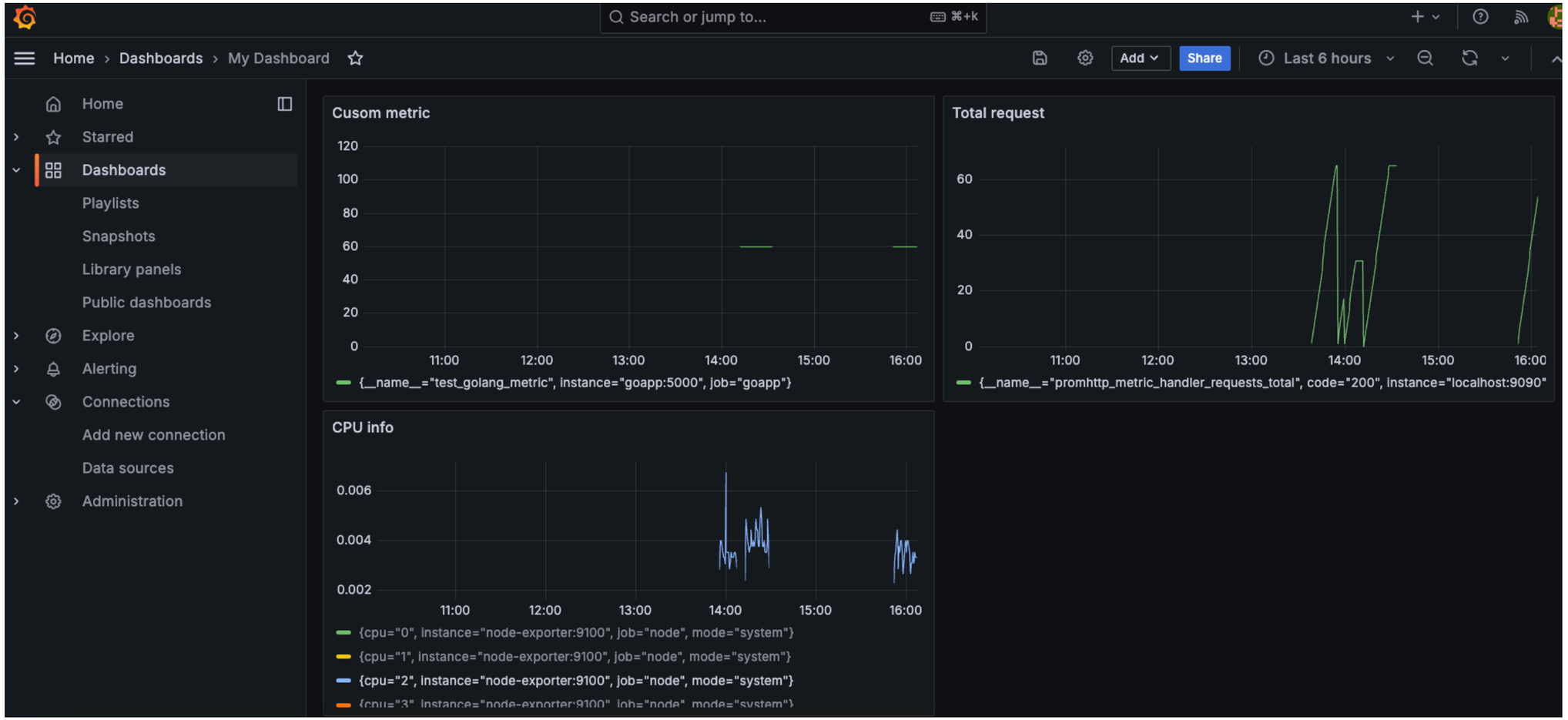
rate(node_cpu_seconds_total{mode="system"}[1m])

- Custom metric

test_golang_metric{job="goapp"}

Grafana

Grafana Panels



Grafana

THANK YOU