



## Kubernetes Observability

Project for learning about Kubernetes observability.

[Explore the docs »](#)

[Project Page](#) - [Report Bug](#) - [Request Feature](#)







► [Table of Contents](#)

## About The Project

This project is for learning about kubernetes observability.

[\(back to top\)](#)

## Built With

-  **GITHUB**
-  **LINUX**
-  **WINDOWS**
-  **SHELL SCRIPT**
-  **POWERSHELL**
-  **KUBERNETES**



[\(back to top\)](#)

## Getting Started

This project is for getting started with kubernetes observability tools and best practices.

Some tools for learning:

- Prometheus
- Alertmanager
- Grafana
- Grafana Loki
- Grafana Tempo

## Prerequisites

- Linux System Up
- Cluster Kubernetes Up
- Git

## Installation

Clone the repo

```
git clone https://github.com/marcossilvestrini/kubernetes-observability.git
cd kubernetes-observability || exit
```

[\(back to top\)](#)

## Usage

I publish some exemples for use in this repository.

(back to top)

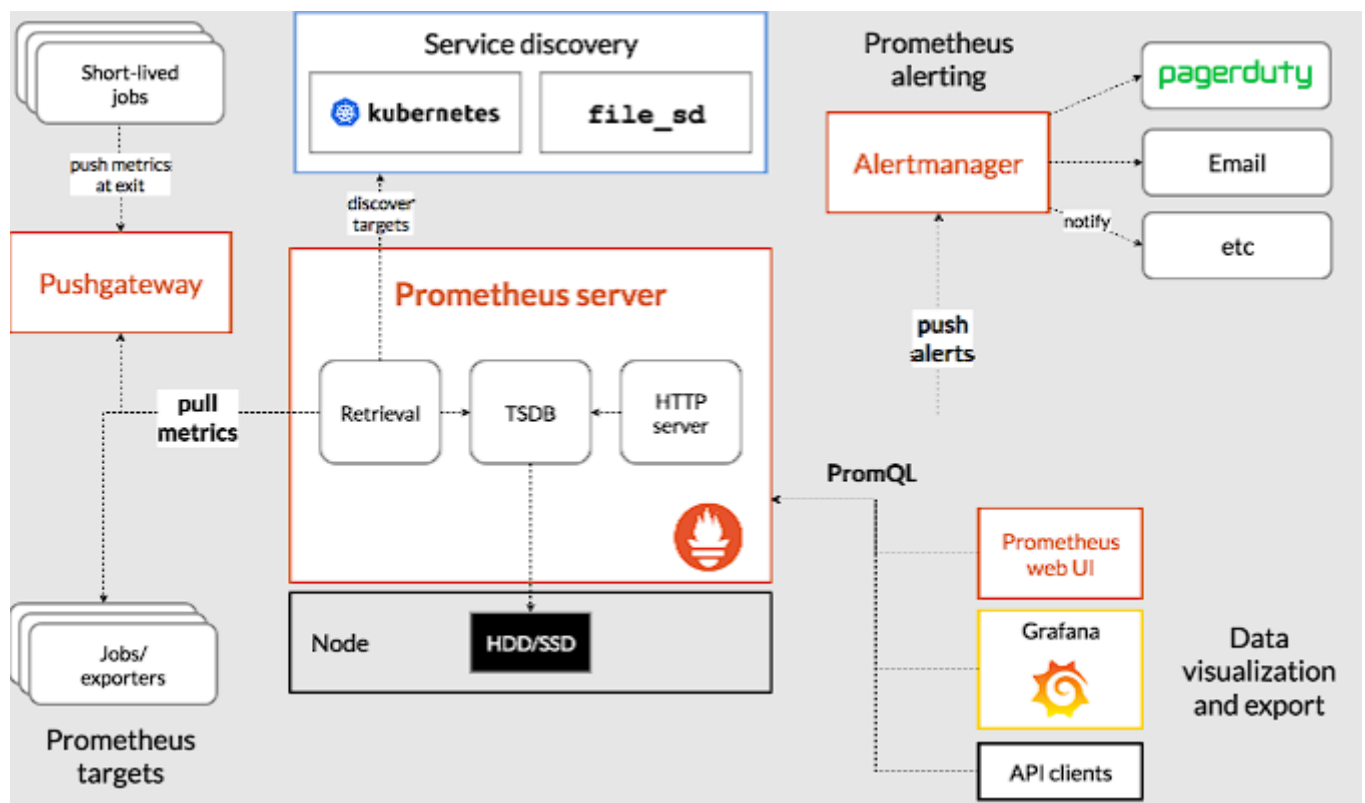
## Roadmap

- ☒ Create repositoty
- ☐ Prometheus
- ☐ Alertmanager
- ☐ Grafana
- ☐ Grafana Loki
- ☐ Grafana Tempo
- ☐ Grafana Alloy
- ☐ Others tools

See the open issues for a full list of proposed features (and known issues).

(back to top)

## Prometheus



Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud.



Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community.

The Prometheus ecosystem consists of multiple components, many of which are optional:

- the main Prometheus server which scrapes and stores time series data
- client libraries for instrumenting application code
- a push gateway for supporting short-lived jobs
- special-purpose exporters for services like HAProxy, StatsD, Graphite, etc.
- an alertmanager to handle alerts
- various support tools

For more information about Prometheus access official documentation:

<https://prometheus.io/docs/introduction/overview/>

## Metric names and labels

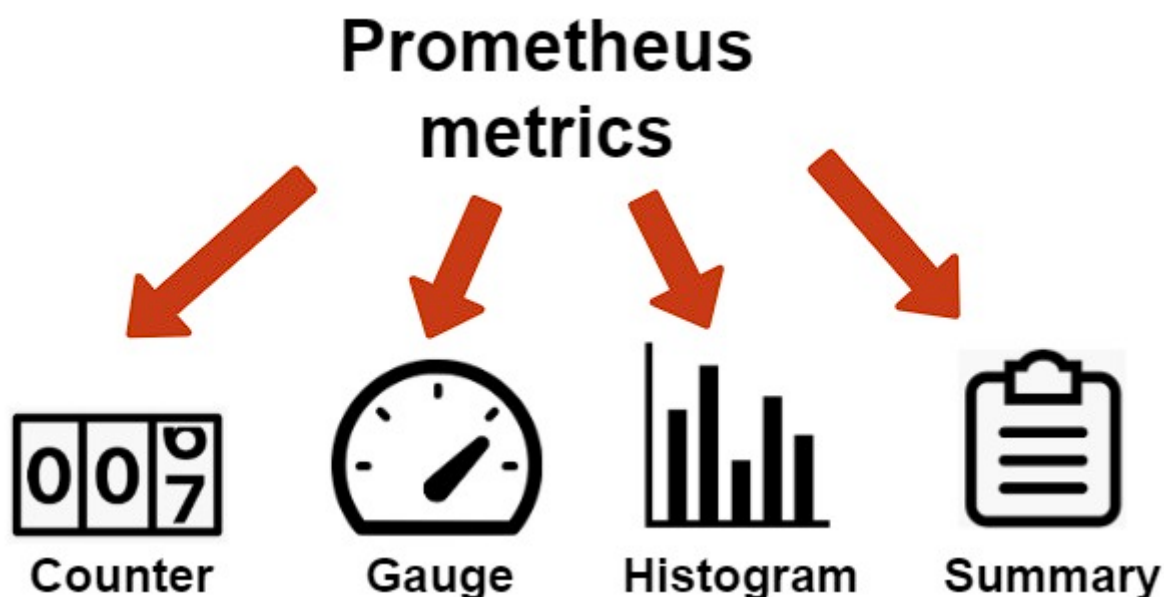
**Example metric name:**

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

**Example metric name with labels:**

```
api_http_requests_total{method="POST", handler="/messages"}
```

## Metrics Types



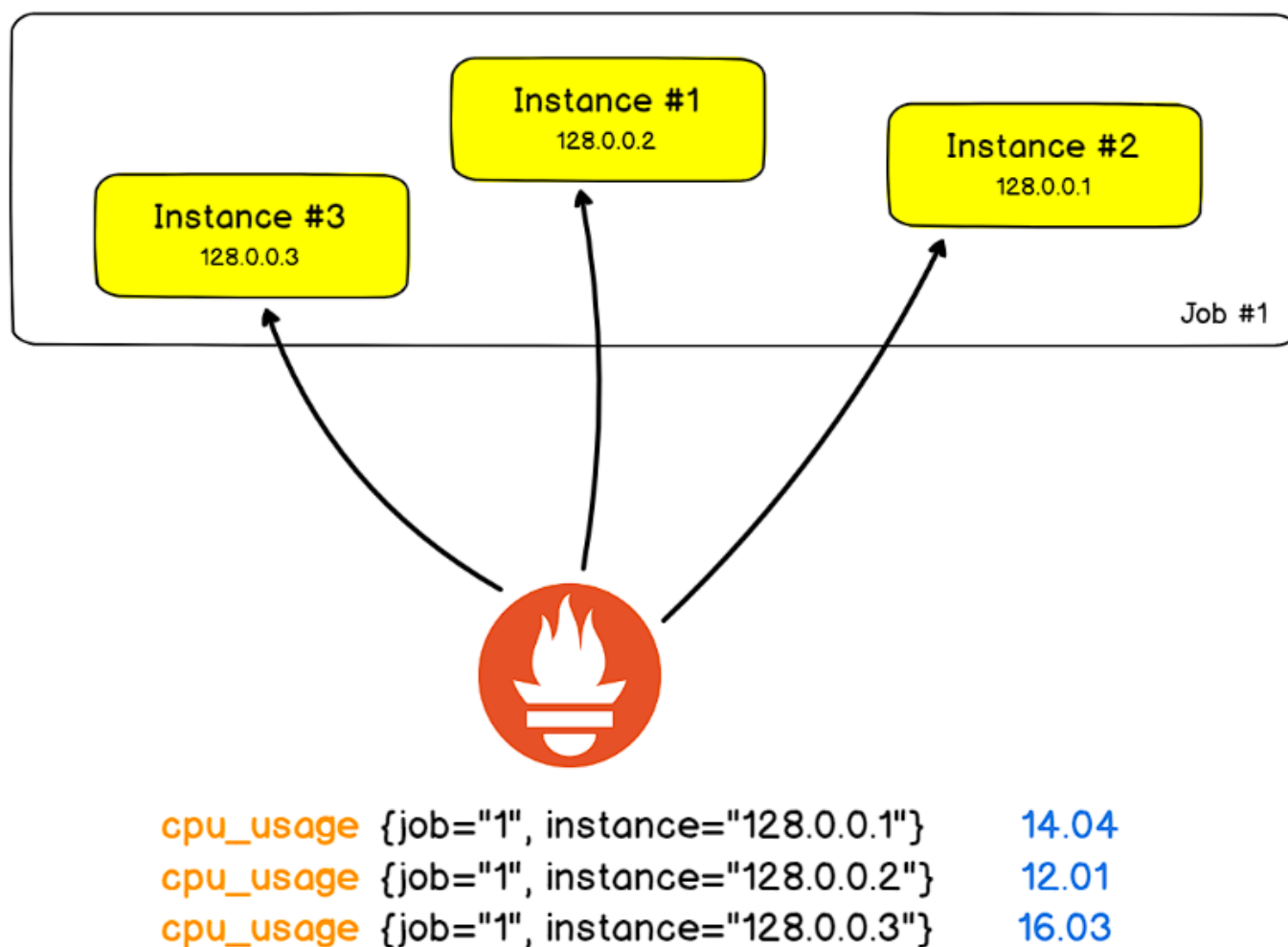
□ **Counter** – accepts and stores only those values that will increase with time.

**Gauge** – stores the values that can take different values, that can both increase and decrease.

**Histogram** – samples observations (usually things like request durations or response sizes) and counts them in configurable buckets. It also provides a sum of all observed values, allowing you to calculate averages.

**Summary** – histogram with a more detailed data representation using additional statistics (quantiles).

## Jobs and Instances



In Prometheus terms, an endpoint you can scrape is called an instance, usually corresponding to a single process.

A collection of instances with the same purpose, a process replicated for scalability or reliability for example, is called a job.

## Prometheus remote-write Specification

The remote write protocol is designed to make it possible to reliably propagate samples in real-time from a sender to a receiver, without loss.

- a "Sender" is something that sends Prometheus Remote Write data.
- a "Receiver" is something that receives Prometheus Remote Write data.

- a "Sample" is a pair of (timestamp, value).
- a "Label" is a pair of (key, value).
- a "Series" is a list of samples, identified by a unique set of labels.

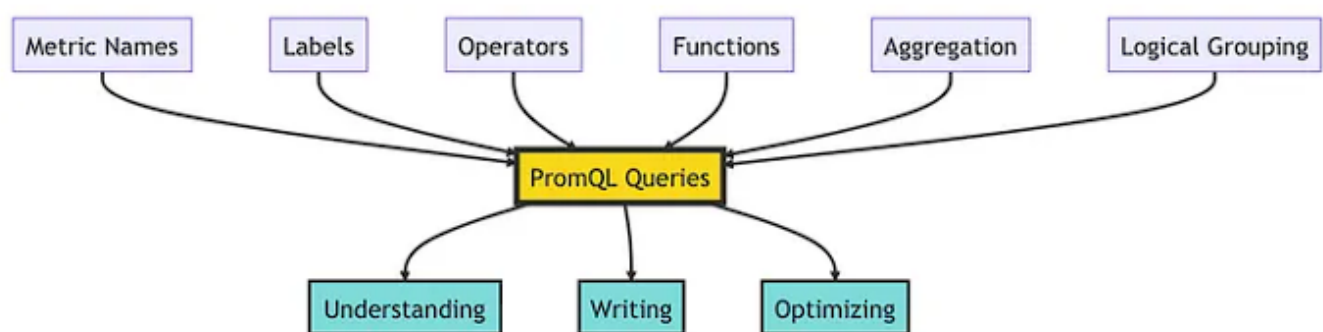


## Compatible Senders and Receivers

The spec is intended to describe how the following components interact:

- Prometheus (as both a "sender" and a "receiver")
- Avalanche (as a "sender") - A Load Testing Tool Prometheus Metrics.
- Cortex (as a "receiver")
- Elastic Agent (as a "receiver")
- Grafana Agent (as both a "sender" and a "receiver")
- GreptimeDB (as a "receiver")
- InfluxData's Telegraf agent. (as a sender, and as a receiver)
- M3 (as a "receiver")
- Mimir (as a "receiver")
- OpenTelemetry Collector (as a "sender" and eventually as a "receiver")
- Thanos (as a "receiver")
- Vector (as a "sender" and a "receiver")
- VictoriaMetrics (as a "receiver")

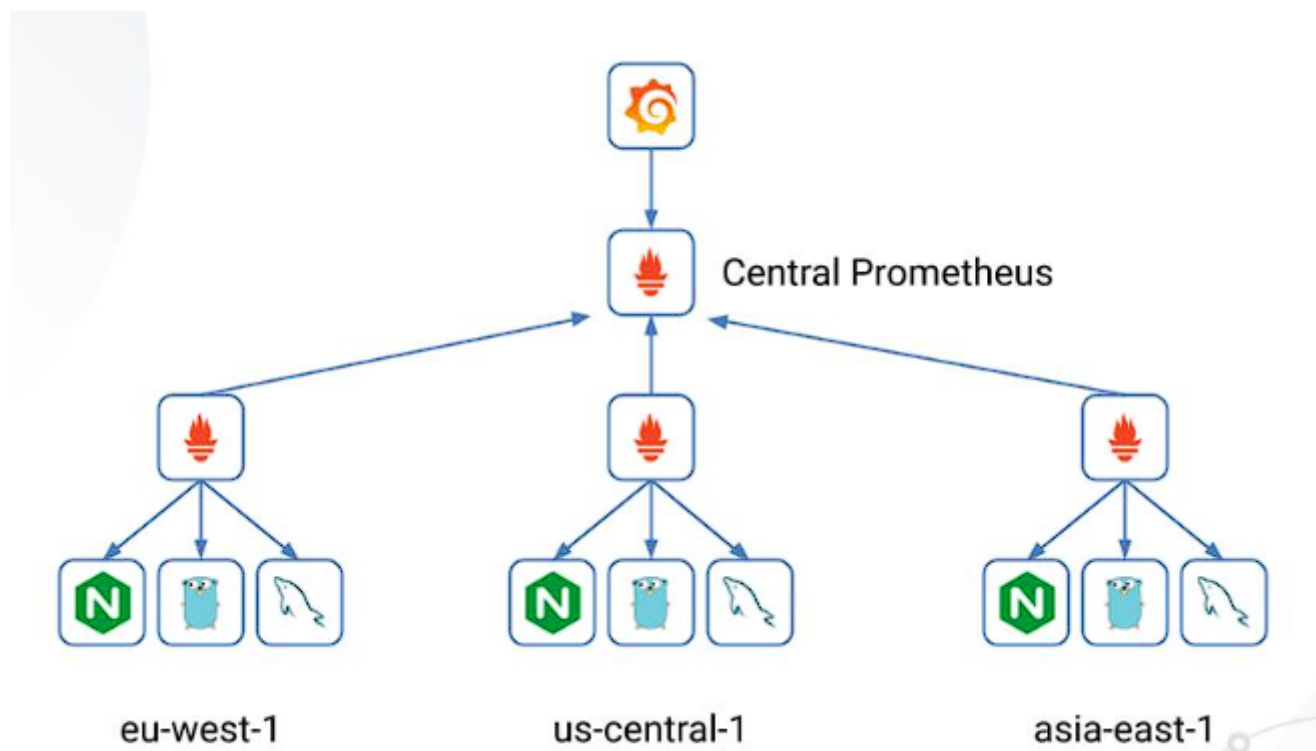
## PromQL



Prometheus provides a functional query language called PromQL (Prometheus Query Language) that lets the user select and aggregate time series data in real time.

The result of an expression can either be shown as a graph, viewed as tabular data in Prometheus's expression browser, or consumed by external systems via the HTTP API.

### Query examples



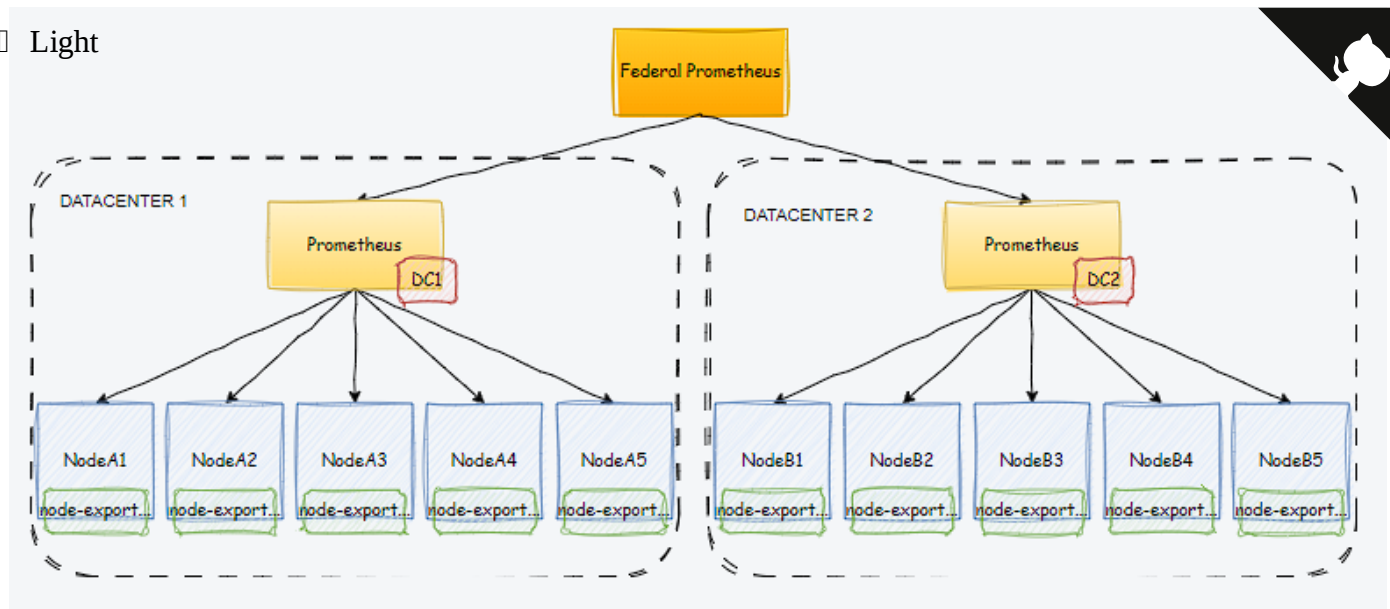
Federation allows a Prometheus server to scrape selected time series from another Prometheus server.

## Hierarchical Federation

Hierarchical federation allows Prometheus to scale to environments with tens of data centers and millions of nodes.

In this use case, the federation topology resembles a tree, with higher-level Prometheus servers collecting aggregated time series data from a larger number of subordinated servers.

This means we have bigger Prometheus servers that collect time-series data from smaller ones. We have a top-down approach where data is gathered from different levels.



## Cross-service federation

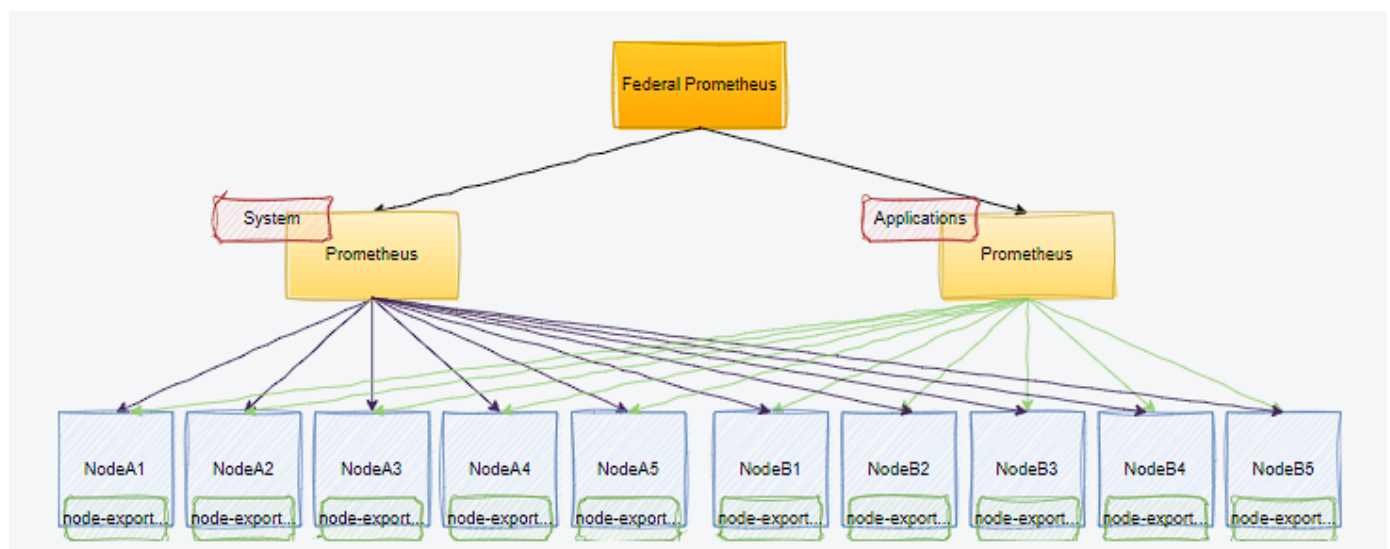
This method involves one Prometheus server monitoring a particular service or group of services, gathering specific time-series data from another server that is monitoring a different set of services.

For example, a cluster scheduler running multiple services might expose resource usage information (like memory and CPU usage) about service instances running on the cluster.

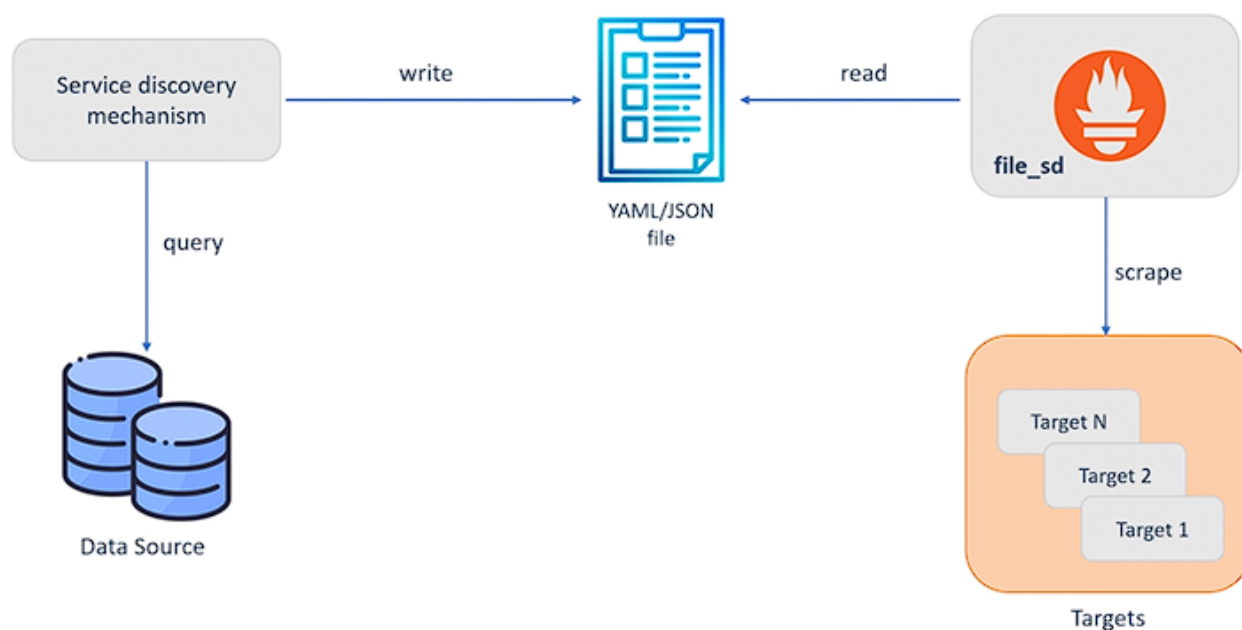
On the other hand, a service running on that cluster will only expose application-specific service metrics.

Often, these two sets of metrics are scraped by separate Prometheus servers. Using federation, the Prometheus server containing service-level metrics may pull in the cluster resource usage metrics about its specific service from the cluster Prometheus, so that both sets of metrics can be used within that server.

By doing this, we can run queries and alerts on the merged data from both servers.







Prometheus provides a generic HTTP Service Discovery, that enables it to discover targets over an HTTP endpoint.

The HTTP Service Discovery is complimentary to the supported service discovery mechanisms, and is an alternative to File-based Service Discovery.

- static\_configs does not scale to more dynamic environments where instances are added/removed frequently
- Prometheus can integrate with service discovery mechanisms to automatically update it's view of of running instances
  - when new instances are added Prometheus will begin scraping, when lost from discovery the time series will also be removed
  - built-in integrations with Consul, Azure, AWS or file based if custom mechanism required
- JSON/YAML file can be published by the platform specifying all targets to scrape from. Prometheus uses it to automatically update targets

## Example using http\_sd\_file

Targets		
http_sd ▾	All Unhealthy Collapse All	Filter by endpoint or labels
http_sd (3/3 up) <a href="#">show less</a>		
Endpoint	State	Labels
http://192.168.0.130:9100/metrics	UP	<div> <div>instance="192.168.0.130:9100" job="http_sd" ^</div> <div> <b>Discovered labels:</b> <div> <div>_address_="192.168.0.130:9100"</div> <div>_meta_filepath="/home/vagrant/prometheus-server/http_sd.json"</div> <div>_meta_prometheus_job="node"</div> <div>_metrics_path_="/metrics"</div> <div>_scheme_="http"</div> <div>_scrape_interval_="15s"</div> <div>_scrape_timeout_="10s"</div> <div>job="http_sd"</div> </div> </div> </div>
http://192.168.0.131:9100/metrics	UP	<div> <div>instance="192.168.0.131:9100" job="http_sd" ▾</div> </div>
http://192.168.0.130:9091/metrics	UP	<div> <div>instance="192.168.0.130:9091" job="http_sd" ▾</div> </div>

prometheus.yaml for scrap the services in target http\_sd.json

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s
scrape_configs:
  # Service Discovery with file_sd
  - job_name: 'http_sd'
    basic_auth:
      username: "skynet"
      password: "prometheus"
    file_sd_configs:
      - files:
          - /home/vagrant/prometheus-server/http_sd.json
```

http\_sd.json

```
[
  {
    "targets": ["192.168.0.130:9100", "192.168.0.131:9100"],
    "labels": {
      "__meta_prometheus_job": "node"
    }
  },
  {
    "targets": ["192.168.0.130:9091"],
    "labels": {
      "__meta_prometheus_job": "pushgateway"
    }
  }
]
```

# Install Prometheus



```
# Download files - https://prometheus.io/download/
wget https://github.com/prometheus/prometheus/releases/download/v2.51.2/prometheus-2

# Extract files
tar xvfz prometheus-*.tar.gz
rm prometheus-*.tar.gz
cd prometheus-*

# Check version
./prometheus --version
```

## Configure Prometheus

See my configuration file prometheus.yaml

```
vim prometheus.yaml
```

```
# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1
  # scrape_timeout is set to the global default (10s).

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

# Load rules once and periodically evaluate them according to the global 'evaluation
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from
  - job_name: "prometheus"

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

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

# Start Prometheus



```
# Start
./prometheus --config.file=prometheus.yml

# Start with PM2 - npm install pm2@latest -g
pm2 start prometheus --name prometheus-server -- --config.file=prometheus.yml
```

## Important Endpoints

```
http://localhost:9090 # all endpoints
http://localhost:9090/graph # PromQL expressions
http://localhost:9090/metrics # metrics
http://localhost:9090/targets # scrape_configs jobs
```

## Using the expression browser

You can use the expression in Table or Graph mode.

Open the page <http://localhost:9090>

```
# Check all http metrics
promhttp_metric_handler_requests_total

# Check http metrics with http status code 200
promhttp_metric_handler_requests_total{code="200"}

# Count http metrics
count(promhttp_metric_handler_requests_total)

# Rate function
rate(promhttp_metric_handler_requests_total{code="200"}[1m])
```

## Prometheus Exporters

An exporter is a binary running alongside the application you want to obtain metrics from. The exporter exposes Prometheus metrics, commonly by converting metrics that are exposed in a non-Prometheus format into a format that Prometheus supports.

### Node Exporter

The Prometheus Node Exporter exposes a wide variety of hardware- and kernel-related metrics.

### Instal Node Exporter

□ Light

```
# Download - https://prometheus.io/download#node_exporter
wget https://github.com/prometheus/node_exporter/releases/download/v1.7.0/node_exp

# Extract
tar xvfz node_exporter-*. *-amd64.tar.gz
cd node_exporter-*. *-amd64
```



## Start Node Exporter

```
# Start
./node_exporter

# Start with PM2 - npm install pm2@latest -g
pm2 start node_exporter --name node_exporter
```

## Endpoints Node Exporter

```
# Access metrics
http://localhost:9100/metrics
```

## Configure Node Exporter

For enable scrap for node exporter, you can configure prometheus.

```
# Edit prometheus file and add job node
vim prometheus.yaml
```

```
...
scrape_configs:
- job_name: node
  static_configs:
  - targets: ['localhost:9100']
...
```

Restart prometheus service for apply new job.

## PushGateway

The Prometheus Pushgateway is an intermediary service which allows ephemeral and batch jobs to expose their metrics to Prometheus.

Since these kinds of jobs may not exist long enough to be scraped, they can instead push their metrics to a Pushgateway.

The Pushgateway then acts as a temporary metrics store which Prometheus scrapes.

Light This setup is particularly useful for capturing the outcome of a job that does not run continuously such as a batch job in a CI system, or a backup script running at a scheduled time. It simplifies monitoring these kinds of jobs without needing to run a long-lived Prometheus instance that might outlive the jobs themselves.



## Install PushGateway

```
# Download
wget -q https://github.com/prometheus/pushgateway/releases/download/v1.8.0/pushgateway-*.tar.gz

# Extract
tar xvfz pushgateway-*.tar.gz
cd pushgateway-*.tar.gz

# Start
# Start with PM2 - npm install pm2@latest -g
pm2 start pushgateway --name pushgateway -- --web.listen-address "192.168.0.130:9091"
```

## Configure PushGateway

```
# Edit prometheus file and add job pushgateway
vim prometheus.yml

scrape_configs:
  - job_name: 'pushgateway'
    honor_labels: true
    static_configs:
      - targets: ['192.168.0.130:9091'] # prometheus server for scraping

# Restart prometheus

# restart with pm2
pm2 restart prometheus-server
```

## Create metrics for test pushgateway

```
echo 'training_completion{course="CKA", status="complete"} 1' > metrics.txt
echo 'training_completion{course="CKS", status="in_progress"} 0.5' >> metrics.txt
echo 'training_completion{course="LPIC2", status="not_started"} 0' >> metrics.txt
curl --data-binary @metrics.txt http://192.168.0.130:9091/metrics/job/training_metri
```

## Endpoints PushGateway

Light  
# Access metrics  
http://localhost:9091



## Use PromQL for find metrics pushgateway target

The screenshot shows the Prometheus web interface. At the top, there's a navigation bar with 'Prometheus', 'Alerts', 'Graph', 'Status', and 'Help'. Below this, there are several toggle switches: 'Use local time' (checked), 'Enable query history' (unchecked), 'Enable autocomplete' (checked), 'Enable highlighting' (checked), and 'Enable linter' (checked). A search bar contains the query 'training\_completion'. Below the search bar, there are two tabs: 'Table' (selected) and 'Graph'. The 'Table' view shows a table with three rows of metrics. The first row is 'training\_completion{course="CKA", job="training\_metrics", status="complete"}', the second is 'training\_completion{course="CKS", job="training\_metrics", status="in\_progress"}', and the third is 'training\_completion{course="LPIC2", job="training\_metrics", status="not\_started"}'. The table has a header 'Evaluation time' with left and right navigation arrows.

## Promlens

### Install Promlens

*Only work without basic authentication*

```
echo "Downloading Promlens..."
wget -q https://github.com/prometheus/promlens/releases/download/v0.3.0/promlens-0.3

echo "Extracting Promlens..."
tar xvfz promlens-*. *-amd64.tar.gz
rm promlens-*. *-amd64.tar.gz
cd promlens-*. *-amd64 || exit

echo "Starting Promlens..."
pm2 start promlens --name promlens -- --web.listen-address "192.168.0.130:8081"
cd || exit
```

### Promlens endpoints

```
# Access query builder
http://192.168.0.130:8081
```



✓ <https://demo.promlabs.com>

node\_filesystem\_files

node\_filesystem\_files

add sum() 6 results - 149ms - mountpoint:6, device:4, fstype:4, instance:1, job:1

+ Add another query

Table Graph Explain

**Metric help:** Filesystem total file nodes.

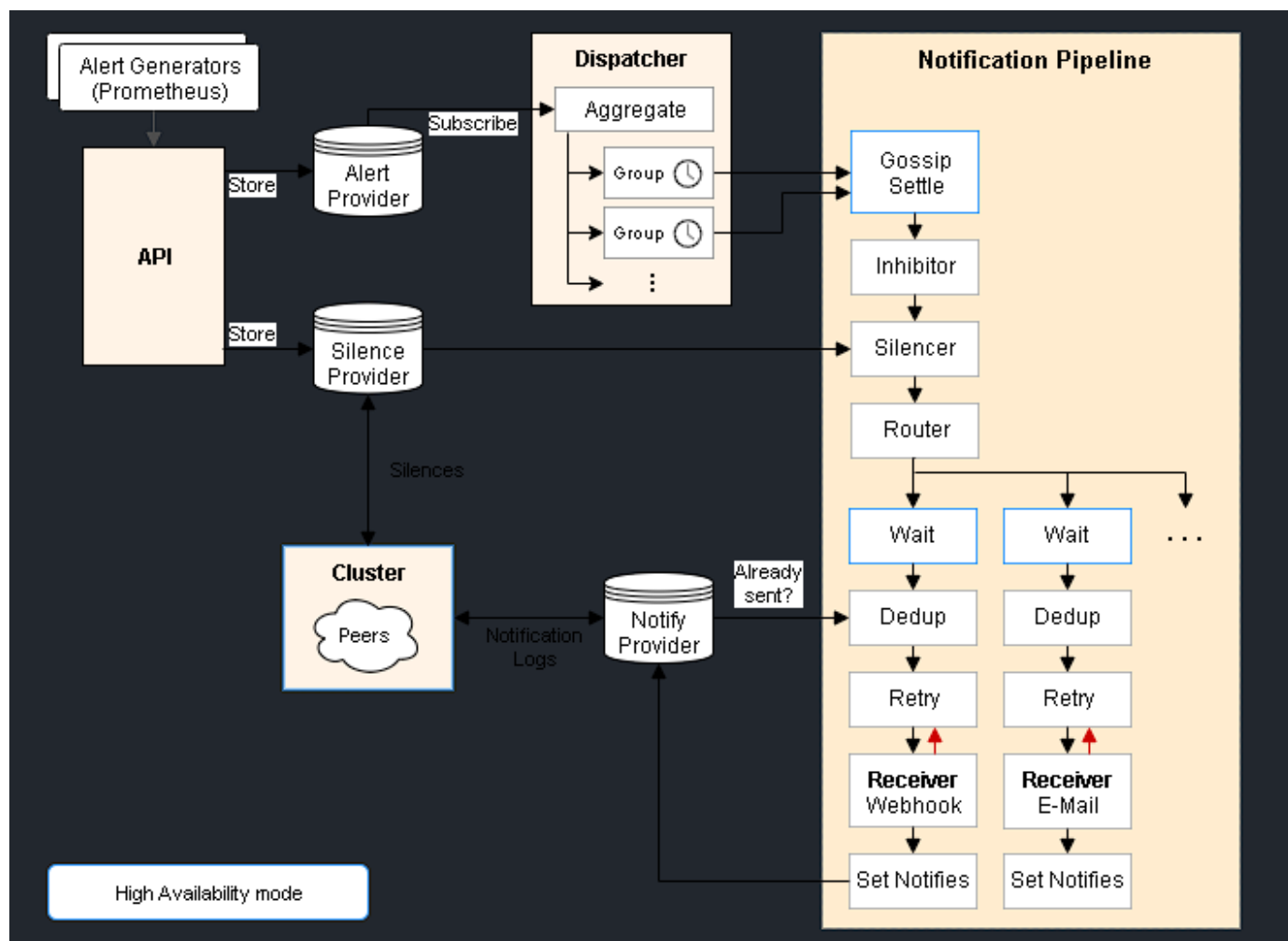
**Metric type:** gauge

This node selects the latest (non-stale) sample value within the last 5m for any series that match all of the following criteria:

- The metric name is node\_filesystem\_files.

If a series has no values in the last 5m, the series will not be returned.

## Alertmanager





For more information about Alertmanager access official documentation:  
<https://github.com/prometheus/alertmanager>



## Grafana

### Grafana Loki

### Grafana Tempo

### Grafana Alloy

(back to top)

## Contributing

Contributions are what make the open source community such an amazing place to learn, inspire, and create. Any contributions you make are **greatly appreciated**.

If you have a suggestion that would make this better, please fork the repo and create a pull request. You can also simply open an issue with the tag "enhancement". Don't forget to give the project a star! Thanks again!

1. Fork the Project
2. Create your Feature Branch ( `git checkout -b feature/AmazingFeature` )
3. Commit your Changes ( `git commit -m 'Add some AmazingFeature'` )
4. Push to the Branch ( `git push origin feature/AmazingFeature` )
5. Open a Pull Request

(back to top)

## License

Distributed under the MIT License. See `LICENSE` for more information.



## Contact

- Marcos Silvestrini - @mrsilvestrini
- marcos.silvestrini@gmail.com

Project Link: <https://github.com/marcossilvestrini/kubernetes-observability>

(back to top)

## Acknowledgments

- Prometheus
- Prometheus Configs
- Prometheus Default port allocations
- Pushgateway
- Exporters
- Node Exporter
- PromQL Article
- Prometheus Articles
  - Prometheus Federation
  - Prometheus Monitoring : The Definitive Guide in 2019
  - Prometheus Service Discovery
  - Kube Prometheus Stack Article

(back to top)