Introduction to Prometheus

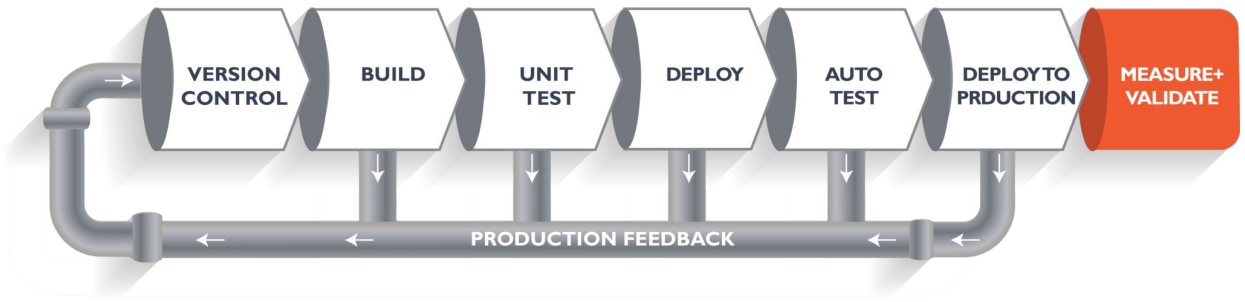
CLARUSWAY©
WAY TO REINVENT YOURSELF

Table of Contents

CLARUSWAY©
WAY TO REINVENT YOURSELF

1 Monitoring: What it is & why to

Monitoring: What it is & why to

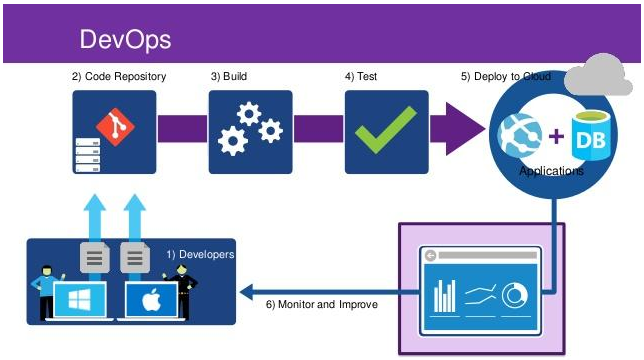


Grafana Nagios® Prometheus ATLASSIAN Opsgenie

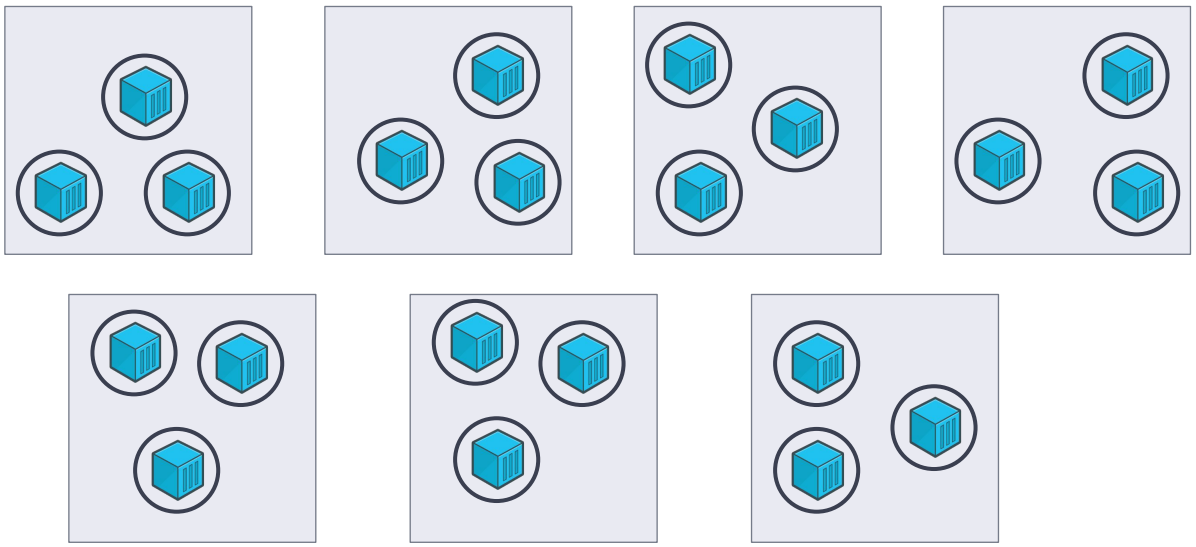
CLARUSWAY©
WAY TO REINVENT YOURSELF

Monitoring: What it is & why to

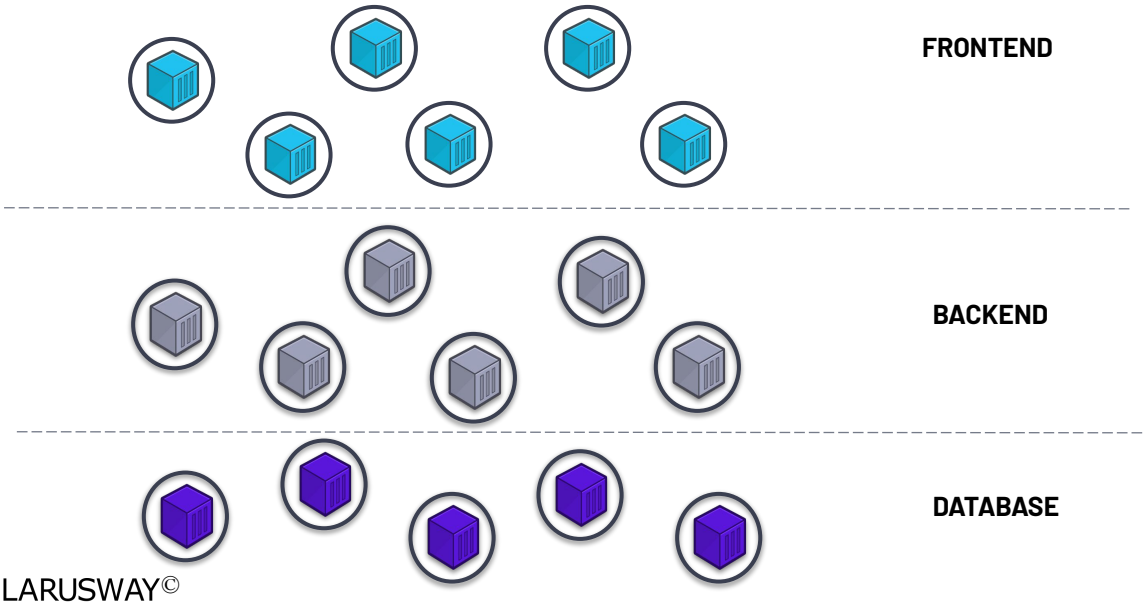
- Agility is essential to keeping pace
- Software teams expected to move faster, test earlier, and release more frequently, all while improving quality and reducing costs



Monitoring: What it is & why to



Monitoring: What it is & why to



Monitoring: What it is & why to

Ensure that a system or service is:

- Available
- Fast
- Correct
- Efficient
- etc.

Monitoring: What it is & why to

Potential Problems:

- Disk full → no new data stored
- Software → bug, request errors
- High temperature → hardware failure
- Network outage → services cannot communicate
- Low memory utilization → money wasted

Monitoring: What it is & why to

Need to observe your systems to get insight into:

- Request/event rates
- Latency
- Errors
- Resource usage
- Temperature, humidity, ...

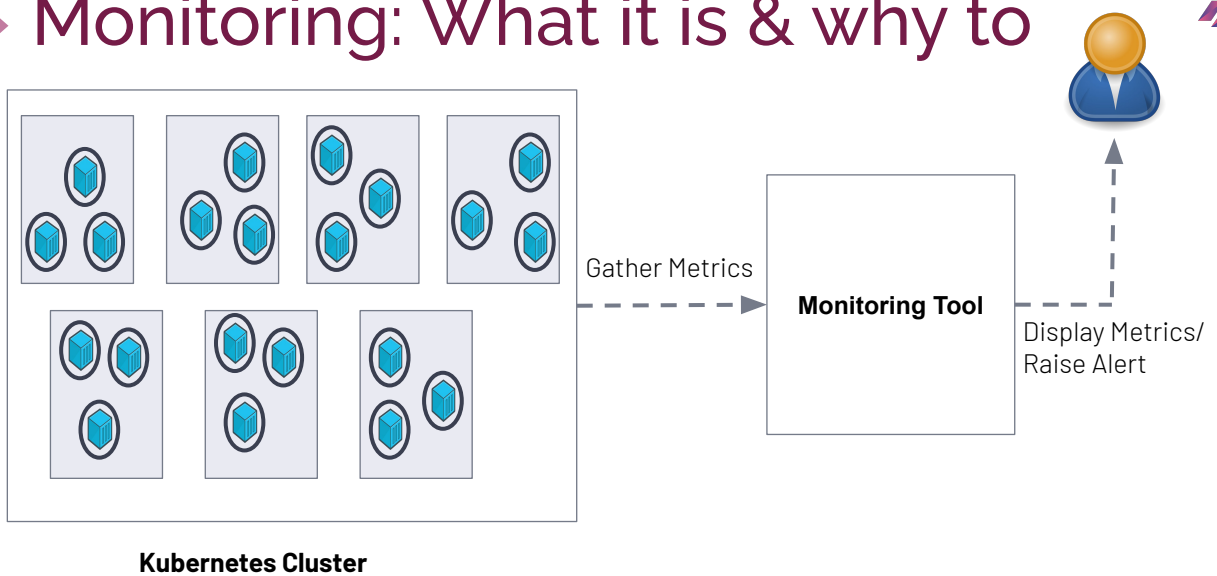
...and then react when something looks bad.

Monitoring: What it is & why to

What is required for monitoring?

- Gather operational metrics
- Raise alert
  - To human (via ticket/SMS/Email/…)
  - To automated handler/agent
- Support issue resolution (data for root cause analysis)
- Analyze trends & effects/impact of change

Monitoring: What it is & why to



2 What is Prometheus?

What is Prometheus?

Metrics-based monitoring & alerting stack

- Metrics collection and storage
- Querying, alerting, dashboarding
- For all levels of the stack!

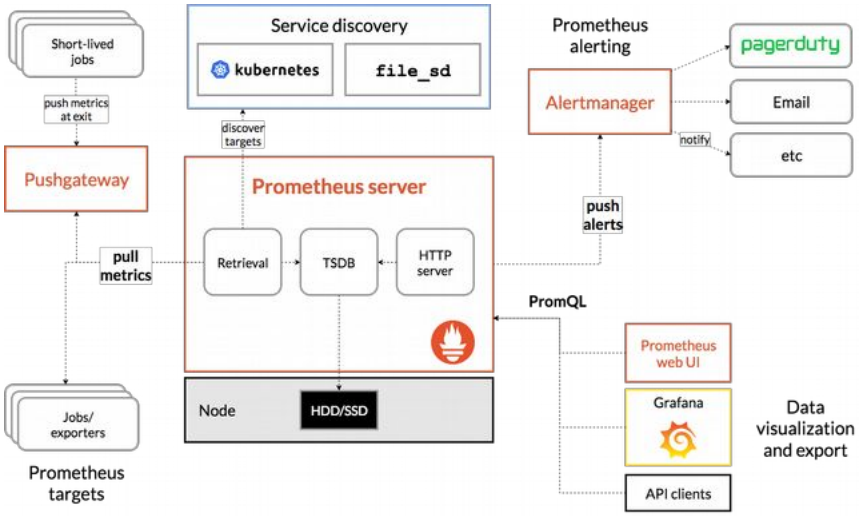
Made for dynamic cloud/container environments

What is Prometheus?

A quick overview of what Prometheus is about:

- **Gather metrics** into database
  - Scheduled pull/harvest/scrape actions – **HTTP/TCP** requests
  - Provide exporters (adapters) that expose metrics
- Make metrics available to consuming systems and humans
  - Such as **Grafana** (for dashboarding), **REST APIs**, through Prometheus UI – **Graphs**, **Console**, **PromQL**
- Analyze metrics according to **alert rules** and determine if alerts are "**firing**"
- Act on firing alerts and send **notifications**

What is Prometheus?



Terminology

- **Prometheus Server**: The main server that scrapes and stores the scraped metrics in a time series database
- **Time-series Database:** Designed to store data that changes with time
- **Scrape:** Prometheus server uses a pulling method to retrieve metrics
- **Target:** The Prometheus server's clients that it retrieves info from (Linux/Windows Server, single app, db, Apache server, etc.)

Terminology

- **Alert Manager:** Component responsible for handling alerts
- **Exporter**: Target libraries that convert and export existing metrics into Prometheus format

Terminology

- **Instance:** The endpoint that is scraped, usually corresponding to a single process
- **Job:** A collection of instances with the same purpose

For example, an API server job with four replicated instances:

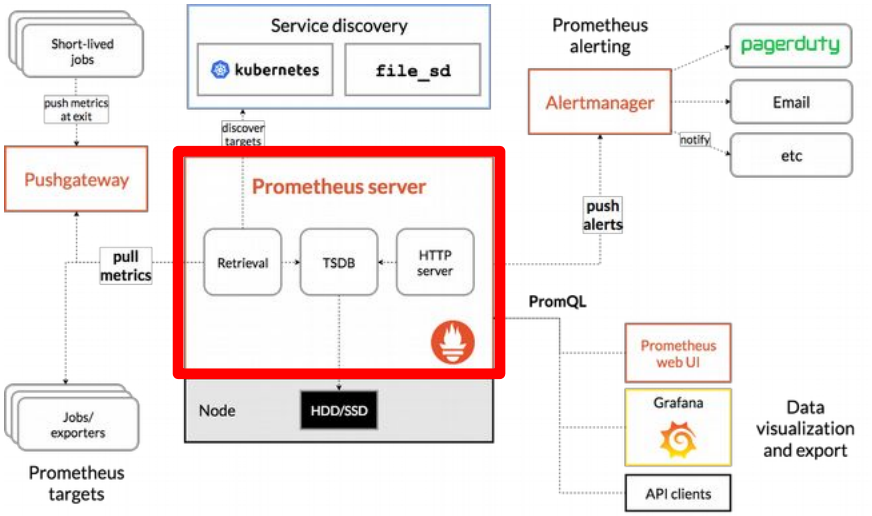
- job: api-server
  - instance 1: 1.2.3.4:5670
  - instance 2: 1.2.3.4:5671
  - instance 3: 5.6.7.8:5670
  - instance 4: 5.6.7.8:5671

Terminology

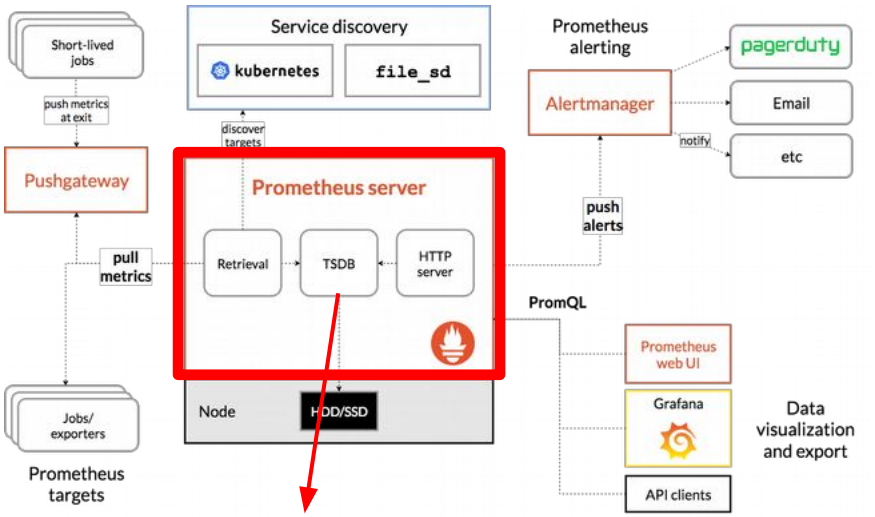
- Prometheus pulls (**scrape**) metrics from a client (**target**) over **http** and places the data into its **time series database** that you can query using its own query language: **promQL**
- Prometheus uses "**exporters**" that are installed/configured on the clients in order to convert and expose their metrics in a Prometheus format
- The **AlertManager** receives metrics from the Prometheus server, makes sense of the metrics and then forwards an alert to the chosen notification system

3 How Prometheus works

How Prometheus works

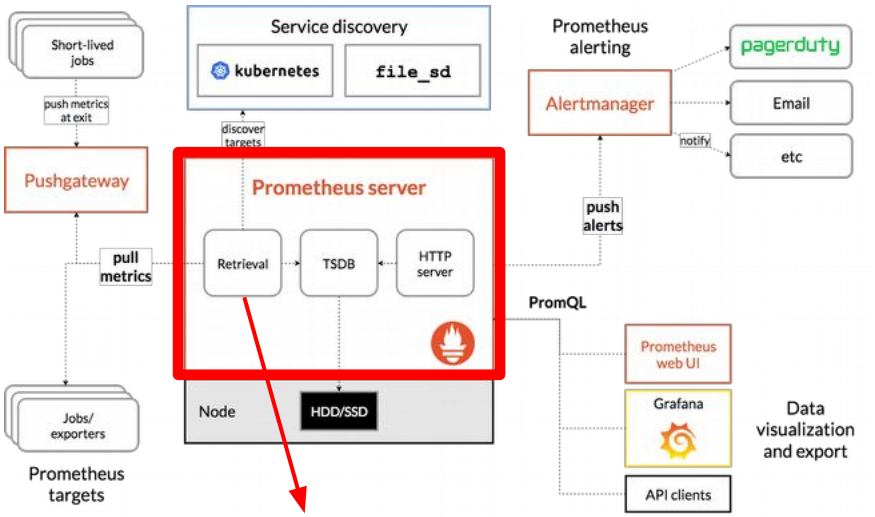


How Prometheus works



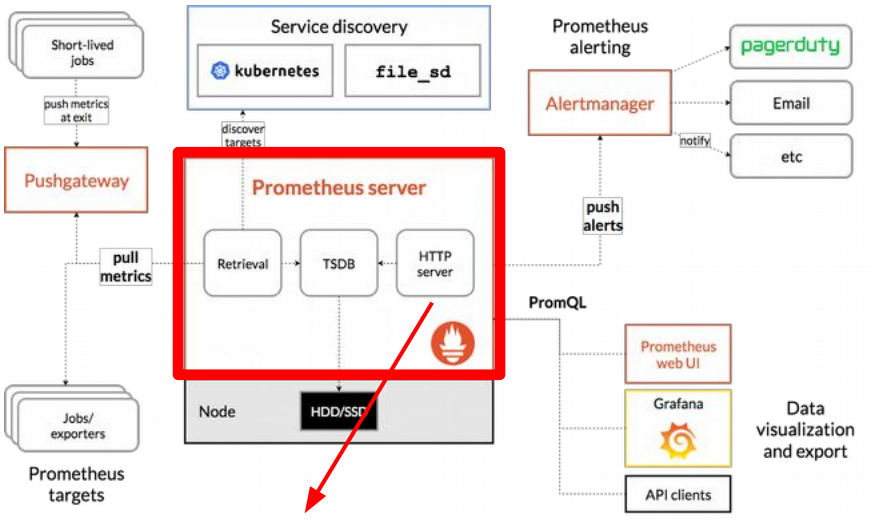
Time Series Database: Stores the metrics

How Prometheus works



Data Retrieval Worker: Pulls metrics data

How Prometheus works



HTTP Server: Accepts promQL queries

How Prometheus works

- Prometheus server monitors **targets** and each target has **metrics** that are monitored.

Targets
- Linux/Windows Server
- Single application
- Services like db
- Web servers
- etc.

Metrics
- CPU/RAM/Disk usage
- Exceptions count
- Requests count
- Requests duration
- etc.

How Prometheus works

Prometheus stores metrics as human-readable text-based format

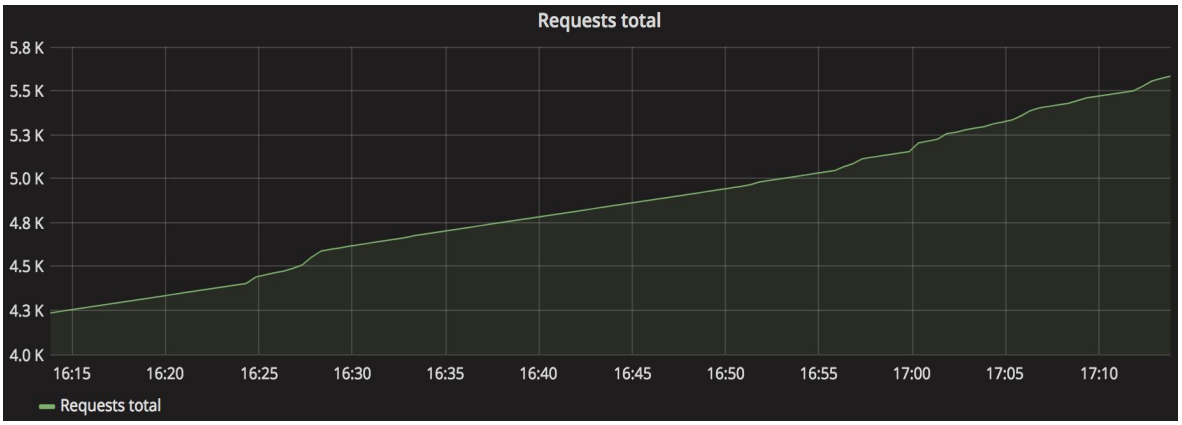
localhost:3000/metrics

```
# TYPE http_server_requests_total counter
# HELP http_server_requests_total The total number of HTTP requests handled by the Rack application.
http_server_requests_total{code="200",method="get",path="/"} 1.0
# TYPE http_server_request_duration_seconds histogram
# HELP http_server_request_duration_seconds The HTTP response duration of the Rack application.
http_server_request_duration_seconds_bucket{method="get",path="/",le="0.005"} 0.0
http_server_request_duration_seconds_bucket{method="get",path="/",le="0.01"} 0.0
http_server_request_duration_seconds_bucket{method="get",path="/",le="0.025"} 0.0
http_server_request_duration_seconds_bucket{method="get",path="/",le="0.05"} 0.0
http_server_request_duration_seconds_bucket{method="get",path="/",le="0.1"} 0.0
http_server_request_duration_seconds_bucket{method="get",path="/",le="0.25"} 0.0
http_server_request_duration_seconds_bucket{method="get",path="/",le="0.5"} 1.0
http_server_request_duration_seconds_bucket{method="get",path="/",le="1"} 1.0
http_server_request_duration_seconds_bucket{method="get",path="/",le="2.5"} 1.0
http_server_request_duration_seconds_bucket{method="get",path="/",le="5"} 1.0
http_server_request_duration_seconds_bucket{method="get",path="/",le="10"} 1.0
http_server_request_duration_seconds_bucket{method="get",path="/",le="+Inf"} 1.0
http_server_request_duration_seconds_sum{method="get",path="/"} 0.251396
http_server_request_duration_seconds_count{method="get",path="/"} 1.0
# TYPE http_server_exceptions_total counter
# HELP http_server_exceptions_total The total number of exceptions raised by the Rack application.
```

HELP: description of what metric is
TYPE: metric type

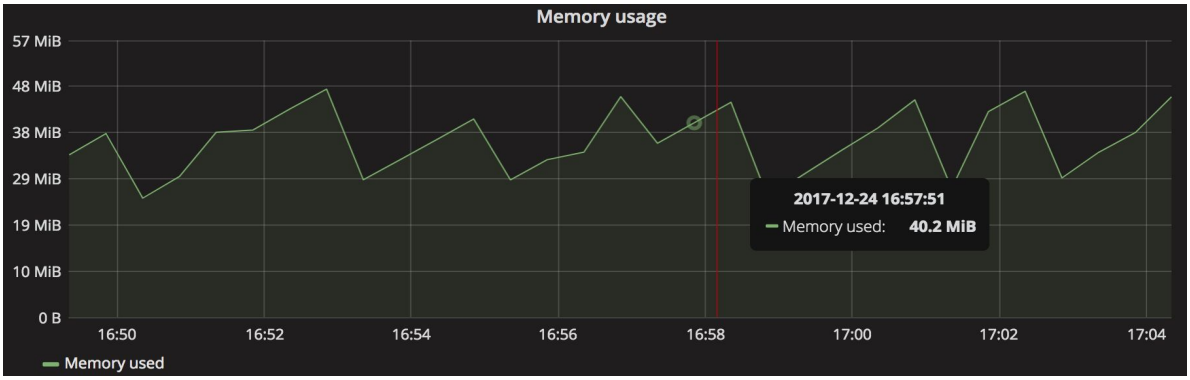
Metric Types

Counter: used for any value that **increases**, such as a request count or error count



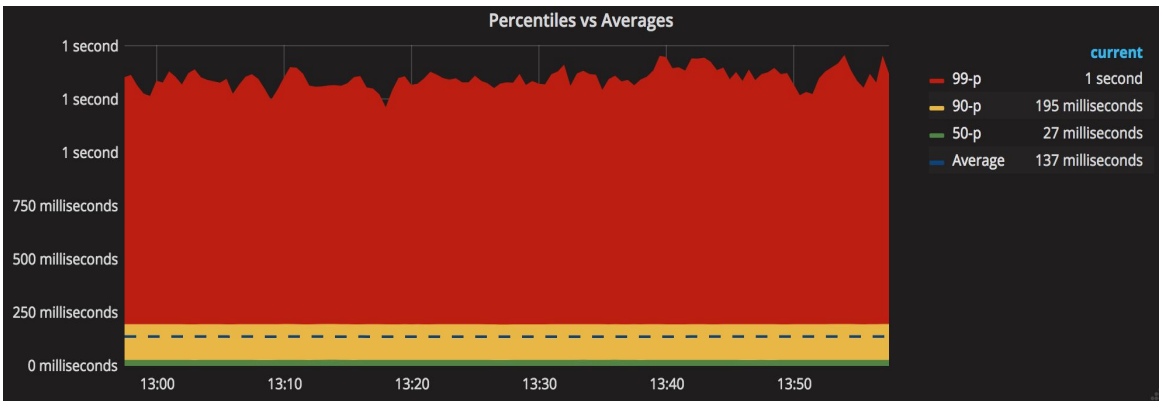
Metric Types

Gauge: used for values that **go down as well as up**, such as current memory usage or the number of items in a queue or the number of requests in progress



Metric Types

Histogram/Summary: measure the frequency of value observations. It tracks how long something takes or how big such as the size of a request.



Metric names and labels

- Every **time series** is uniquely identified by its **metric name** and optional **key-value pairs** called **labels**
- Notation:

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

- For example:

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


Collecting Metrics

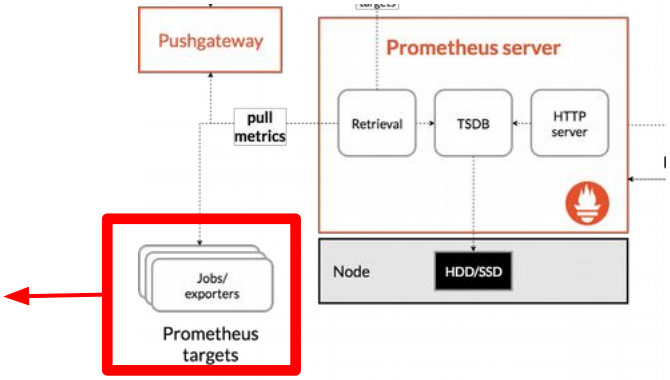
- Prometheus pulls metrics from the targets over HTTP:

  http://hostaddress: [port]/metrics

- Some services expose their metrics natively
- But many services requires an extra component that is called an **exporter**

Collecting Metrics

Exporter is a script or service that fetches metrics from the target, converts to correct format, and exposes **/metrics** so that Prometheus server can pull



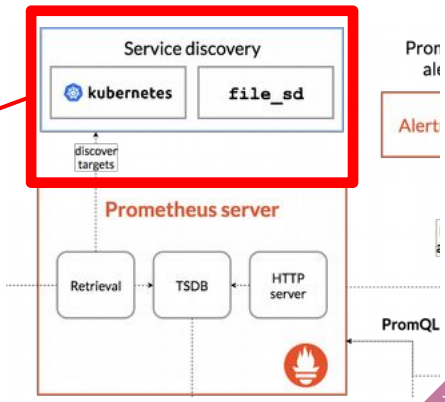
4 Configuring Prometheus

Configuring Prometheus

Prometheus is configured with **prometheus.yml** file that has the information of:

- which targets to scrape?
- at which interval?

Prometheus uses **service discovery** to find the targets mentioned in the YAML file



Configuring Prometheus

Prometheus comes with a sample configuration file

at which interval targets will be scraped

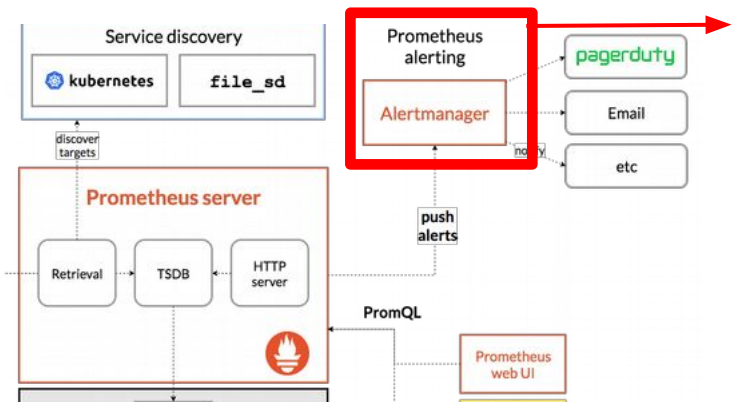
rules for gathering metric values or creating alerts

Resources that Prometheus monitors

```
# my global config
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # 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_interval'.
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 this config.
  - job_name: 'prometheus'

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

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

5 Alert Manager

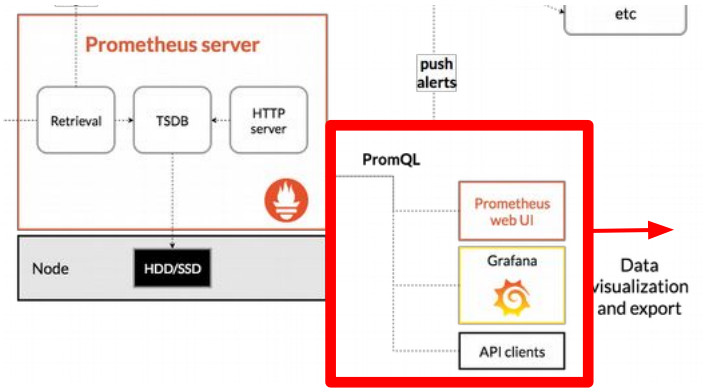
Alert Manager



Alertmanager fires alerts that are defined by rules in configuration file

6 Querying

Querying



promQL is used to visualize the data collected by Prometheus server

Querying

- Example queries:

```
# Request counter for the User Directory service
http_requests_total{service="users-directory"}

# Request counter for the Billing History Service
http_requests_total{service="billing-history"}

# Overall request counter regardless of service
sum(http_requests_total)
```


Querying

- Example query with Grafana:

THANKS!

Any questions?

You can find me at:

- @David - Instructor
- david@clarusway.com