Prometheus Cheatsheet

A quick reference guide for Prometheus, covering essential concepts, PromQL queries, configuration, and best practices for monitoring and alerting in a DevOps environment.

Core Concepts

Metrics and Data Model

Metric Types:

- **Counter:** A cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero on restart.
- **Gauge:** A metric that represents a single numerical value that can arbitrarily go up and down.
- **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.
- **Summary:** Similar to a histogram, a summary samples observations. While it also provides a total count of observations and a sum of all observed values, it calculates configurable quantiles over a sliding time window.

Data Model:

Prometheus fundamentally stores all data as time series: streams of timestamped values belonging to the same metric and the same set of labeled dimensions.

Labels:

Key-value pairs that allow Prometheus's dimensional data model to shine. Any given combination of labels for the same metric identify a particular dimensional instantiation of that metric (e.g. all HTTP requests that used the method `POST` to the `/api/tracks` handler).

Architecture

Prometheus Server	Scrapes and stores time-series data.
Service Discovery	Automatically discovers targets to scrape.
Exporters	Expose metrics from third-party systems (e.g., node_exporter for system metrics).
Alertmanager	Handles alerts sent by Prometheus.

Key Components

Prometheus Server:

The core component responsible for scraping metrics, storing them, and evaluating alerting rules.

Exporters:

Tools that expose metrics in a Prometheus-readable format. Examples include `node_exporter` for system metrics, `mysqld_exporter` for MySQL metrics, etc.

Alertmanager:

Handles alerts generated by Prometheus. It can group, deduplicate, and route alerts to various receivers (e.g., email, Slack, PagerDuty).

PromQL - Querying Prometheus

Basic Queries

`http_requests_total` - Returns all time series with the metric name `http_requests_total`.

`http_requests_total{job="prometheus"}` - Returns time series with the metric name `http_requests_total` and the label `job` set to `prometheus`.

`up` - Returns the current state of all monitored instances (1 for up, 0 for down).

Functions

`rate(metric[duration])`	Calculates the per-second average rate of increase of the time series in the range vector. Use for counters.
`irate(metric[duration])`	Calculates the per-second instant rate of increase of the time series in the range vector. Useful for graphing volatile counters.
`increase(metric[duration])`	Calculates the increase in the time series in the range vector. Good for graphing total increases.
`sum(metric) by (label)`	Sums the values of all time series with the same label values.
`avg(metric) by (label)`	Averages the values of all time series with the same label values.
`histogram_quantile(quantile, metric)`	Calculates the given quantile from a histogram.

Common Queries

CPU Usage:

```
100 - (avg by (instance)
(irate(node_cpu_seconds_total{mode="idle"}
[5m])) * 100)
```

Memory Usage:

```
(1 - (node_memory_MemAvailable_bytes /
node_memory_MemTotal_bytes)) * 100
```

Disk Usage:

```
(node_filesystem_size_bytes{mountpoint="/"} -
node_filesystem_free_bytes{mountpoint="/"}) /
node_filesystem_size_bytes{mountpoint="/"} *
100
```

Configuration

Prometheus Configuration File (prometheus.yml)

The main configuration file for Prometheus, written in YAML. It defines scrape configurations, alerting rules, and other settings.

global: - Global settings such as scrape interval and evaluation interval.
scrape_configs: - Defines the targets to scrape and how to scrape them.
rule_files: - Specifies the location of alerting and recording rules.

Scrape Configuration

A scrape configuration defines how Prometheus scrapes metrics from a target.

```
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
```

job_name: - The name of the job.
static_configs: - Defines a list of targets to scrape.
targets: - A list of hostnames or IP addresses to scrape.

Alerting Rules

Alerting rules define conditions under which alerts should be fired.

```
groups:
  - name: ExampleAlerts
    rules:
      - alert: HighCPUUsage
        expr: 100 - (avg by (instance)
(irate(node_cpu_seconds_total{mode="idle"}
[5m])) * 100) > 80
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: 'High CPU usage detected on
{{ $labels.instance }}'
```

alert: - The name of the alert.
expr: - The PromQL expression that triggers the alert.
for: - How long the condition must be true before an alert is fired.
labels: - Labels to add to the alert.
annotations: - Additional information about the alert.

Best Practices

Naming Conventions

Use consistent and descriptive names for metrics and labels.

Follow the `<prefix>_<unit>_<metric>` naming convention (e.g., `http_requests_total`).

Use labels to add dimensionality to your metrics (e.g., `method="GET"`, `status="200"`).

Alerting Strategies

Define meaningful alerts that provide actionable insights.

Use `for` clauses to prevent flapping alerts.

Group alerts based on severity and route them to the appropriate teams.

Monitoring Strategies

Monitor key performance indicators (KPIs) for your applications and infrastructure.

Use dashboards to visualize metrics and identify trends.

Implement service discovery to automatically monitor new instances.