DATADOG

English US1

DOCS > METRICS > METRIC TYPE MODIFIERS

Metric Type Modifiers

A metric type is an indication of what you are trying to represent with your metric and its emission source. The `COUNT` and `RATE` metric types are quite similar to each other, as they represent the same concept: the variation of a metric value over time. However, they use different logic:

- `RATE` : Normalized value variation over time (*per second*).
- `COUNT` : Absolute value variation over a given time interval.

Depending on your use case and submission method, one metric type may be more suited than the other for submission. For instance:

METRIC TYPE SUBMITTED	USE CASE
RATE	You want to monitor the number of requests received over time across several hosts.
RATE	You do not have control over the consistency of temporal count submissions across your sources, so you're normalizing by each individual interval to be able to compare them upstream.
COUNT	You want to count the number of times a function is called.
COUNT	Counting the amount of revenue that have been made over a given amount of time.

Since `RATE` and `COUNT` aren't the same metric type, they don't have the same behavior or shape within Datadog graphs and monitors. To change metrics on the fly between `RATE` and `COUNT` representations, use Datadog's in-application modifier functions within your graphs and monitors.

In-application modifiers

DATADOG

`as_rate()`	Sets the operations necessary to display the given metric in `RATE` form, giving you the absolute variation of a metric value per second.

Depending on the metric type you applied them to, the behavior differs:

COUNT RATE GAUGE

- Effect of `as_count()` :
  - Disables any interpolation.
  - Sets the time aggregator to `SUM` .
- Effect of `as_rate()` :
  - Disables any interpolation.
  - Sets the time aggregator to `SUM` .
  - Divides the result post-aggregation by the sampling interval in order to normalize it. For example, the following points submitted every second `[1,1,1,1].as_rate()` with a rollup interval of 20s would produce `[0.05, 0.05, 0.05, 0.05]` .

Note: There is no normalization on tiny intervals (when no time aggregation occurs), thus the raw metric value counts are returned.

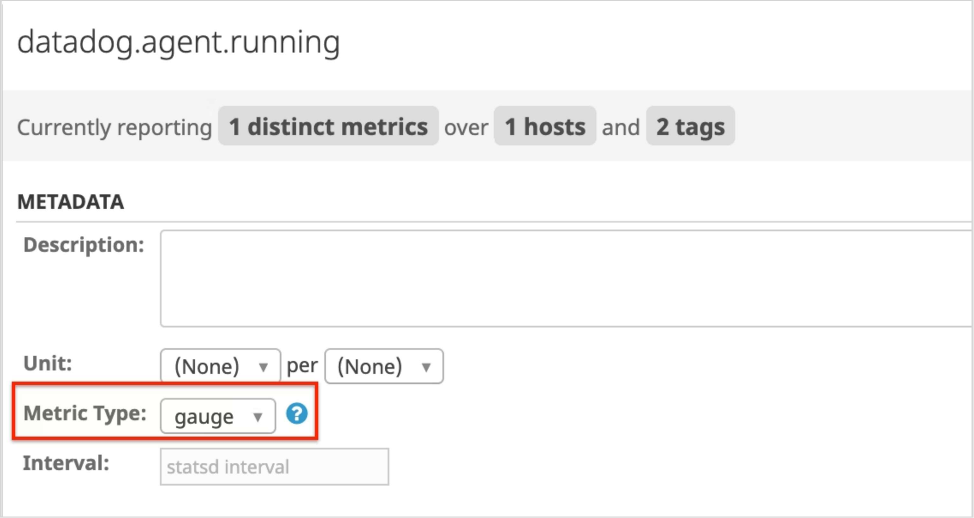
The weighted() modifier

Tags such as `pod name` or `container_name` cause high tag churn, especially when creating queries for cost management, capacity planning, or autoscaling for containerized applications. To ensure mathematical accuracy of queries on gauges regardless of tag churn, you can use a `.weighted()` in-application modifier. The `.weighted()` modifier enables Datadog to properly weight metric values based on the lifespan of these frequently churning tags.

The `.weighted()` modifier is automatically appended to queries on gauges only if both of the following conditions are met:

- The gauge metric is submitted regularly, such that there is no interpolation over gaps.
- The submission interval is correctly defined and set.

page:



datadog.agent.running

Currently reporting **1 distinct metrics** over **1 hosts** and **2 tags**

METADATA

Description:

Unit: (None) per (None)

Metric Type: gauge

Interval: statsd interval

Example use case:

1. You have a metric `app.requests.served` that counts requests served, but accidentally submitted it from StatsD as a `GAUGE` . The metric's Datadog type is, therefore, `GAUGE` .

2. You wanted to submit `app.requests.served` as a StatsD `COUNT` metric for time aggregation. This would help answer questions like *"How many total requests were served in the past day?"* by querying `sum:app.requests.served{*}` (this would not make sense for a `GAUGE` metric type.)

3. You like the name `app.requests.served` , so rather than submitting a new metric name with a more appropriate `COUNT` type, you could change the type of `app.requests.served` by updating:

- Your submission code, calling `dogstatsd.increment('app.requests.served', N)` after N requests are served, and
- The Datadog in-app type from the metric summary page to `RATE` .

This causes data submitted before the type change for `app.requests.served` to behave incorrectly. This is because it was stored in a format to be interpreted as an in-app `GAUGE` (not a `RATE`). Data submitted after step 3 is interpreted properly.

DATADOG

Can't find something?

Our friendly, knowledgeable solutions engineers are here to help!

CONTACT US

FREE TRIAL

Download mobile app

Download on the App Store

GET IT ON Google Play

RESOURCES

Pricing
Documentation
Support
Learning Center
Certification
Open Source

Resources
Webinars
Security
Privacy Center
Knowledge Center
Learning Resources

PRODUCT

Features
Infrastructure Monitoring
Container Monitoring

Browser Real User Monitoring
Mobile Real User Monitoring
Synthetic Monitoring

Sensitive Data Scanner

APM

Error Tracking

Continuous Profiler

Data Streams Monitoring

Database Monitoring

CI Pipeline Visibility

Test Visibility & Intelligent Test Runner

Service Catalog

Dynamic Instrumentation

Universal Service Monitoring

Bits AI

Workflow Automation

CoScreen

Dashboards

Watchdog

Alerts

Incident Management

Integrations

API

Case Management

ABOUT

Contact Us

Partners

Press

Leadership

Careers

Legal

Investor Relations

Analyst Reports

ESG Report

Vendor Help

BLOG

The Monitor

Engineering

Pup Culture

Security Labs

Terms | Privacy | Cookies

English