Observability FAQ

Document Level Classification

200

    - How to compute cpu utilization with k8s metrics
- New Relic Deprecation
    - Where can I get updates on deprecation/shutdown timelines
    - What can I use to replace my New Relic synthetic transactions

General

Where the heck are my metrics/logs/traces

Generally speaking, anything you're sending in from OpenTelemetry in Substrate will end up accessible from a Grafana datasource:

- Metrics - accessible from Prometheus "Chronosphere Tenant" datasources
- Logs - accessible from "Loki" datasources
- Traces - accessible from "Tempo" datasources

These are divided into tenants, which are logically separated groups of related data usually broken down (coarsely) by business unit. Epic is a large company, and it doesn't make sense to store all telemetry data in one place (and in some cases, we can't, due to policy!). Observability developed the tenant mapping scheme based on our understanding of the company's structure. It's not a perfect system, so if you think your data is in a tenant that doesn't make sense, please reach out. Some examples of tenants that exist today are: **online**, **eos**, **tech-services, ucs, mediatonic**. The source of truth for tenant mappings resides here currently.

Self-Service

The Observability team maintains an Epicenter app, Observability Self-Service, that can help you find your service and its metrics, logs, and traces.

Unprovisioned or Unknown tenants

If you looked everywhere and found your data in one of the "unknown" or "unprovisioned" data sources - that means we haven't figured out where to route your data. This is common with new AWS accounts. Please reach out so that we can get your data organized correctly.

Grafana

When will we go to Grafana 12

We're currently blocked on Grafana 12 because Epic has an internally developed data source plugin for Epic Metrics written in AngularJS. Grafana 12 removes support for Angular, so until the plugin can be rewritten (and extensively tested by its users) in React we cannot upgrade. We're trying to prioritize this work so stay tuned.

How can I access Grafana from Terraform

If you're using Epic's HCP terraform environment, a Grafana Service Account is available for you to use, accessible from vault from workspaces within your HCP terraform project. See the example code below for how to use this.

```
data "vault_generic_secret" "grafana_service_account" {
  path = "secret/observability-pulse/platform-shared-secrets/grafana/hc
}

provider "grafana" {
  url  = "https://grafana.ol.epicgames.net"
  auth = data.vault_generic_secret.grafana_service_account.data["servic
}
```

Some HCP projects can also access EOS Grafana, below is some example code that works for EOS if your project is enabled for EOS Grafana access.

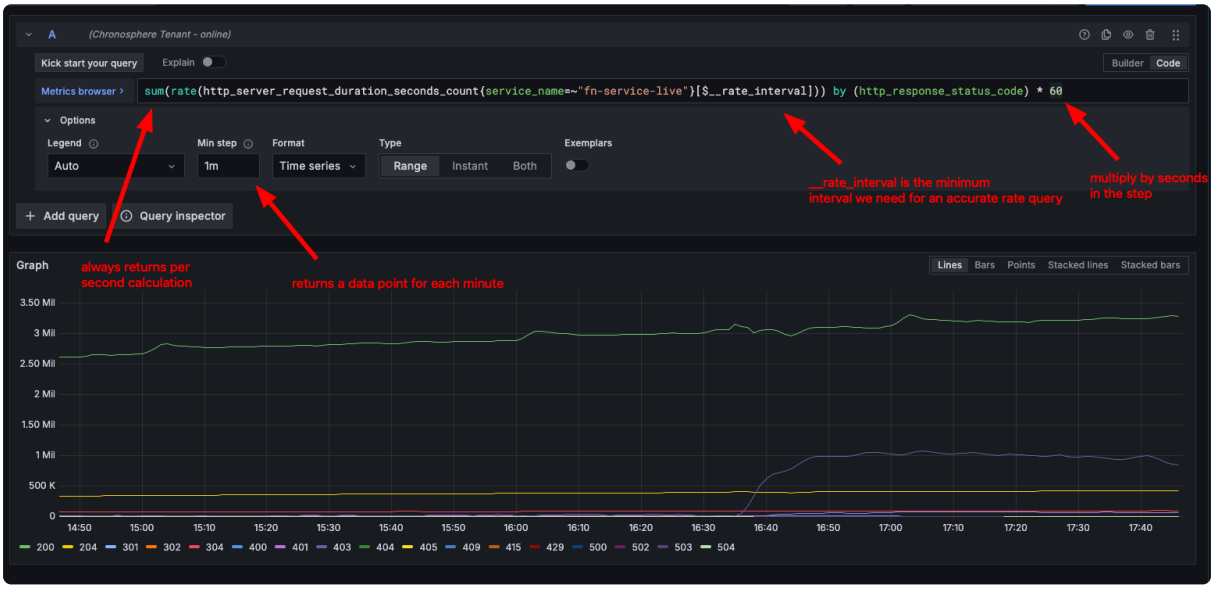
```
data "vault_generic_secret" "grafana_service_account" {
  path = "secret/observability-pulse/platform-shared-secrets/grafana/hc
}

provider "grafana" {
  url  = "https://grafana-eos.on.epicgames.com"
  auth = data.vault_generic_secret.grafana_service_account.data["servic
}
```

Writing Queries

How to compute rate per minute in PromQL

tl;dr: use `rate` * 60 with 1m step sizes - **increase** can lead you astray, so make sure your [range vector] and step size are aligned. Example of a 1m rate query:

```
sum(rate(http_server_request_duration_seconds_count{service_name=~"fn-s
```



Explanation

The Prometheus rate function calculates the per-second average rate of increase of the time series in the range vector. What is the range vector? That's the bit in `[square brackets]` you'll see on most PromQL queries, typically for `rate` queries this is `[$__rate_interval]`. `$__rate_interval` is a special macro introduced by Grafana to simplify writing rate queries. It is 4 times the Scrape Interval (how often data points come in). The scrape interval for our data sources is 1m, meaning `$__rate_interval` is 4 minutes. `rate` works with counter values that are reported at each scrape interval, and in order to compute a rate, it needs at least two samples. Grafana settled on 4 times the scrape interval (potentially 4 samples) due to various issues, described in this blog post if you want to deep dive on this. What's important to take away is that when you run a `rate({}[$__rate_interval])` you're computing a **per-second average based on 4 minutes of data.** This means there's some loss of precision - this is a deliberate tradeoff as reporting data frequently enough to do rates over 1m intervals would require a scrape interval of 15s, which would increase costs significantly.

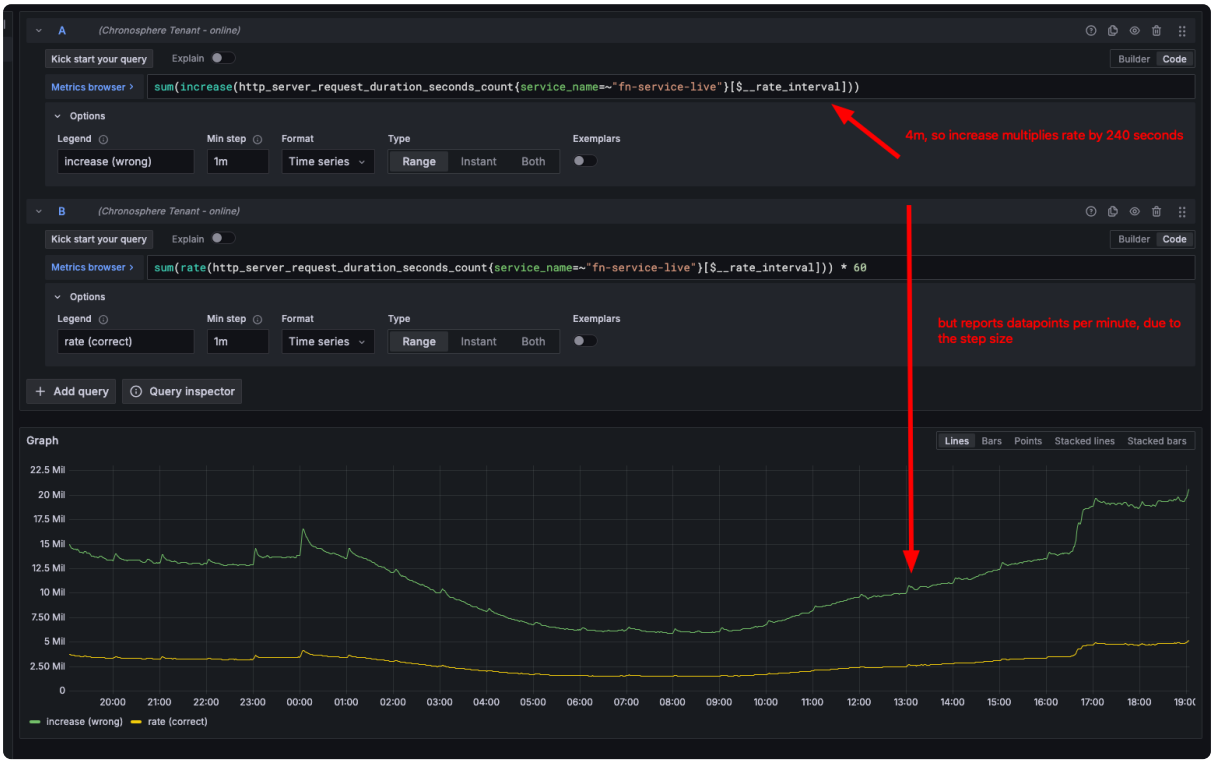
Let's talk now about **step size**. Most queries are **range queries** over the given time range, meaning your query is executed at equally-spaced steps between the start of your range and the end, which generates the data points you see that render the time series. If you query the last hour of data, with a 1 minute step size, you'll generate 60 data points. Each data point is generated using the specified query but with a slightly different start and end time. If you're familiar with SQL, it may help to think of the query being executed like this:

```
select rate(...) where time > start and time <= end # a 4 minute block
select rate(...) where time > start-1 minute and time <= end-1 minute
select rate(...) where time > start-2 minutes and time <= end-2 minutes
select rate(...) where time > start-3 minutes and time <= end-3 minutes
```

You can compute the (approximate) per-minute rate by multiplying the `rate()` calculation value, which is in seconds, by 60. You can extend this

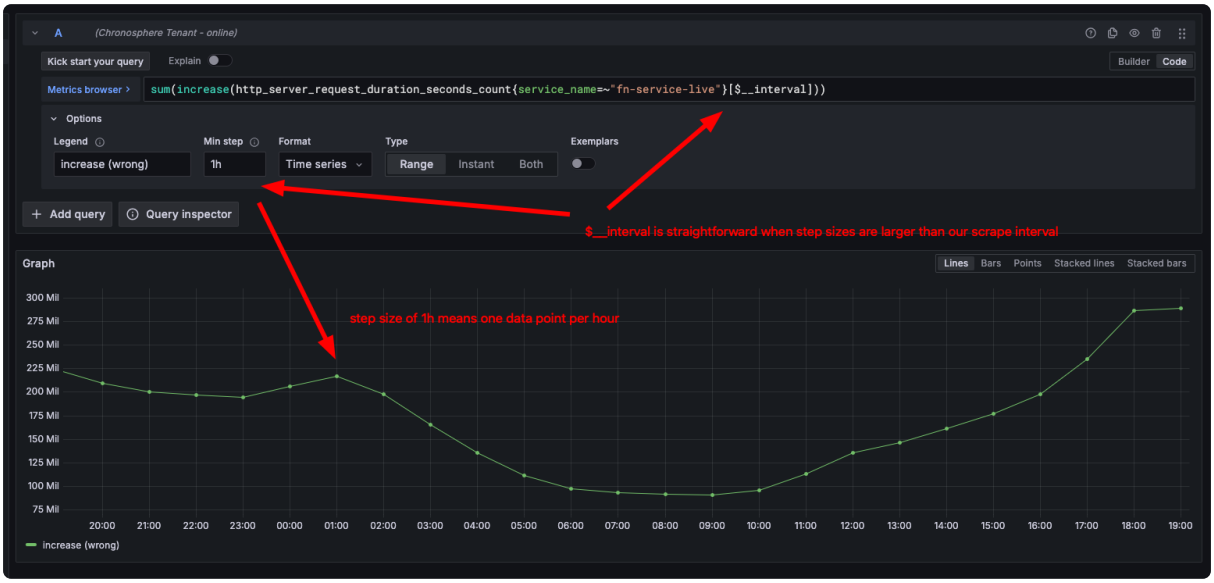
concept to get 5m, 1hr, or other rates, though keeping track of all of these seconds multipliers is a pain, so PromQL has the **increase** function.

Per the PromQL docs, **increase** "is syntactic sugar for `rate(v)` multiplied by the number of seconds under the specified time range window, and should be used primarily for human readability." For example, if you use `increase({}[5m])` it is essentially the same as running `rate([{}[5m]) * 300`. So why not use this for 1m queries? Because of the minimum value for `$__rate_interval` - a query of `increase({}[$__rate_interval]` ends up resolving to `rate({}[4m] * 240` - usually not what you expect when you're trying to report rate per minute. To make matters worse, if you use a step size of 1m you will end up with data points that are 4x the actual value. This is the source of a lot of confusion and why we wrote this FAQ section. Example:



So why not do `increase({}[1m])` ? Because there are not enough datapoints to compute a rate at 1m due to our scrape interval. This gets less complicated as you zoom out and query data over larger step sizes. For example, to compute an hourly request rate, `increase` is straightforward. Note we're using `$__interval` here rather than `$__rate_interval` - `$__interval` always matches the step size - in this

case it resolves to 1h, where there are plenty of samples to run `rate` or `increase`. The `$__rate_interval` value is mostly useful when querying data at small step sizes that are close to our scrape interval (1m).



How to compute cpu utilization with k8s metrics

This can be somewhat confusing because our k8s infra metrics report a `container_cpu_utilization` that is actually expressed in number of cores used, where "utilization" in other contexts usually refers to a percentage. To get the percentage, you can divide the # of cores used by the CPU requested, represented by the kube_pod_container_resource_requests metric. Note this will only work if you have defined resource requests for your pods.

```
container_cpu_utilization{service_name="fn-service-live", k8s_container
/ on () group_left() kube_pod_container_resource_requests{service_name=
```

You can use the same approach for memory if you've defined memory requests and/or limits.

```
container_memory_usage{service_name="fn-service-live", k8s_container_na
/ on () group_left() kube_pod_container_resource_requests{service_name=
```

For help with other basic infra metrics queries, take a look at our Kubernetes dashboards. These are general purpose boards that can be copied or borrowed from if you need help getting started.

- Kubernetes Service Resource Metrics
- Kubernetes Cluster Overview

New Relic Deprecation

Where can I get updates on deprecation/shutdown timelines

See the #ct-product-announcements-ext channel in slack. This is Core Tech's channel for low-volume/high-signal announcements.

What can I use to replace my New Relic synthetic transactions

Observability is looking into a replacement for synthetics later this year (2025), starting in Q3. For now, continue to use NR synthetics as-is.
