Monitoring 101: Collecting the right data

Alexis Lê-Quôc

Published: June 30, 2015

theory / alerting / metrics / monitoring-101

This post is part of a series on effective monitoring. Be sure to check out the rest of the series: Alerting on what matters and Investigating performance issues.

Monitoring data comes in a variety of forms—some systems pour out data continuously and others only produce data when rare events occur. Some data is most useful for *identifying* problems; some is primarily valuable for

investigating problems. More broadly, having monitoring data is a necessary condition for *observability* into the inner workings of your systems.

This post covers which data to collect, and how to classify that data so that you can:

1. Receive meaningful, automated alerts for potential problems
2. Quickly investigate and get to the bottom of performance issues

Whatever form your monitoring data takes, the unifying theme is this:

> *Collecting data is cheap, but not having it when you need it can be expensive, so you should instrument everything, and collect all the useful data you reasonably can.*

This series of articles comes out of our experience monitoring large-scale infrastructure for our customers. It also draws on the work of Brendan Gregg, Rob Ewaschuk, and Baron Schwartz.

Metrics

Metrics capture a value pertaining to your systems *at a specific point in time* — for example, the number of users currently logged in to a web application. Therefore, metrics are usually collected once per second, one per minute, or at another regular interval to monitor a system over time.

There are two important categories of metrics in our framework: work metrics and resource metrics. For each system that is part of your software infrastructure, consider which work metrics and resource metrics are reasonably available, and collect them all.

Work metrics

Work metrics indicate the top-level health of your system by measuring its useful output. When considering your work metrics, it's often helpful to break them down into four subtypes:

- **throughput** is the amount of work the system is doing per unit time. Throughput is usually recorded as an absolute number.
- **success** metrics represent the percentage of work that was executed successfully.
- **error** metrics capture the number of erroneous results, usually expressed as a rate of errors per unit time or normalized by the throughput to yield errors per unit of work. Error metrics are often

captured separately from success metrics when there are several potential sources of error, some of which are more serious or actionable than others.

- **performance** metrics quantify how efficiently a component is doing its work. The most common performance metric is latency, which represents the time required to complete a unit of work. Latency can be expressed as an average or as a percentile, such as “99% of requests returned within 0.1s”.

These metrics are incredibly important for observability. They are the big, broad-stroke measures that can help you quickly answer the most pressing questions about a system’s internal health and performance: is the system available and actively doing what it was built to do? How fast is it producing work? What is the quality of that work?

Below are example work metrics of all four subtypes for two common kinds of systems: a web server and a data store.

Example work metrics: Web server (at time 2015-04-24 08:13:01 UTC)

Subtype	Description	Value
throughput	requests per second	312
success	percentage of responses that are 2xx since last measurement	99.1
error	percentage of responses that are 5xx since last measurement	0.1
performance	90th percentile response time in seconds	0.4

Example work metrics: Data store (at time 2015-04-24 08:13:01 UTC)

Subtype	Description	Value
throughput	queries per second	949
success	percentage of queries successfully executed since last measurement	100
error	percentage of queries yielding exceptions since last measurement	0
error	percentage of queries returning stale data since last measurement	4.2
performance	90th percentile query time in seconds	0.02

Resource metrics

Most components of your software infrastructure serve as a resource to other systems. Some resources are low-level—for instance, a server's resources include such physical components as CPU, memory, disks, and network interfaces. But a higher-level component, such as a database or a geolocation microservice, can also be considered a resource if another system requires that component to produce work.

Resource metrics can help you reconstruct a detailed picture of a system's state, making them especially valuable for investigation and diagnosis of problems. For each resource in your system, try to collect metrics that cover four key areas:

1. **utilization** is the percentage of time that the resource is busy, or the percentage of the resource's capacity that is in use.
2. **saturation** is a measure of the amount of requested work that the resource cannot yet service, often queued.

3. **errors** represent internal errors that may not be observable in the work the resource produces.
4. **availability** represents the percentage of time that the resource responded to requests. This metric is only well-defined for resources that can be actively and regularly checked for availability.

Here are example metrics for a handful of common resource types:

Resource	Utilization	Saturation	Errors	Availability
Disk IO	% time that device was busy	wait queue length	# device errors	% time writable
Memory	% of total memory capacity in use	swap usage	N/A (not usually observable)	N/A
Microservice	average % time each request-servicing thread was busy	# enqueued requests	# internal errors such as caught exceptions	% time service is reachable
Database	average % time each connection was busy	# enqueued queries	# internal errors, e.g. replication errors	% time database is reachable

Other metrics

There are a few other types of metrics that are neither work nor resource metrics, but that nonetheless may help making a complex system observable. Common examples include counts of cache hits or database locks. When in doubt, capture the data.

Events

In addition to metrics, which are collected more or less continuously, some monitoring systems can also capture events: discrete, infrequent occurrences that can provide crucial context for understanding what changed in your system's behavior. Some examples:

- Changes: Internal code releases, builds, and build failures
- Alerts: Internally generated alerts or third-party notifications
- Scaling events: Adding or subtracting hosts

An event usually carries enough information that it can be interpreted on its own, unlike a single metric data point, which is generally only meaningful in context. Events capture *what happened*, at a point in *time*, with optional *additional information*. For example:

What happened	Time	Additional information
Hotfix f464bfe released to production	2015–05–15 04:13:25 UTC	Time elapsed: 1.2 seconds
Pull request 1630 merged	2015–05–19 14:22:20 UTC	Commits: ea720d6
Nightly data rollup failed	2015–05–27 00:03:18 UTC	Link to logs of failed job

Events are sometimes used used to generate alerts—someone should be notified of events such as the third example in the table above, which indicates that critical work has failed. But more often they are used to investigate issues and correlate across systems. In general, think of events like metrics—they are valuable data to be collected wherever it is feasible.

What good data looks like

The data you collect should have four characteristics:

- **Well-understood.** You should be able to quickly determine how each metric or event was captured and what it represents. During an outage you won't want to spend time figuring out what your data means. Keep your metrics and events as simple as possible, use standard concepts described above, and name them clearly.
- **Granular.** If you collect metrics too infrequently or average values over long windows of time, you may lose the ability to accurately reconstruct a system's behavior. For example, periods of 100% resource utilization will be obscured if they are averaged with periods of lower utilization. Collect metrics for each system at a frequency that will not conceal problems, without collecting so often that monitoring becomes perceptibly taxing on the system (the observer effect) or creates noise in your monitoring data by sampling time intervals that are too short to contain meaningful data.
- **Tagged by scope.** Each of your hosts operates simultaneously in multiple scopes, and you may want to check on the aggregate health of any of these scopes, or their combinations. For example: how is production doing in aggregate? How about production in the

Northeast U.S.? How about a particular role or service? It is important to retain the multiple scopes associated with your data so that you can alert on problems from any scope, and quickly investigate outages without being limited by a fixed hierarchy of hosts.

- **Long-lived.** If you discard data too soon, or if after a period of time your monitoring system aggregates your metrics to reduce storage costs, then you lose important information about what happened in the past. Retaining your raw data for a year or more makes it much easier to know what "normal" is, especially if your metrics have monthly, seasonal, or annual variations.

Data for alerts and diagnostics

The table below maps the different data types described in this article to different levels of alerting urgency outlined in a companion post. In short, a *record* is a low-urgency alert that does not notify anyone automatically but is recorded in a monitoring system in case it becomes useful for later analysis or investigation. A *notification* is a moderate-urgency alert that notifies someone who can fix the problem in a non-interrupting way such as email or chat. A *page* is an urgent alert that interrupts a recipient's work, sleep, or personal time, whatever the hour. Note that depending on severity, a notification may be more appropriate than a page, or vice versa:

Data	Alert	Trigger
Work metric: Throughput	Page	value is much higher or lower than usual, or there is an anomalous rate of change
Work metric: Success	Page	the percentage of work that is successfully processed drops below a threshold

Work metric: Errors	Page	the error rate exceeds a threshold
Work metric: Performance	Page	work takes too long to complete (e.g., performance violates internal SLA)
Resource metric: Utilization	Notification	approaching critical resource limit (e.g., free disk space drops below a threshold)
Resource metric: Saturation	Record	number of waiting processes exceeds a threshold
Resource metric: Errors	Record	number of errors during a fixed period exceeds a threshold
Resource metric: Availability	Record	the resource is unavailable for a percentage of time that exceeds a threshold
Event: Work-related	Page	critical work that should have been completed is reported as incomplete or failed

Conclusion: Collect ’em all

- Instrument everything and collect as many work metrics, resource metrics, and events as you reasonably can. Observability of complex systems demands comprehensive measurements.
- Collect metrics with sufficient granularity to make important spikes and dips visible. The specific granularity depends on the system you are measuring, the cost of measuring and a typical duration between changes in metrics—seconds for memory or CPU metrics, minutes for energy consumption, and so on.
- To maximize the value of your data, tag metrics and events with several scopes, and retain them at full granularity for at least 15 months.

We would like to hear about your experiences as you apply this framework to your own monitoring practice. If it is working well, please let us know on Twitter! Questions, corrections, additions, complaints, etc? Please let us know on GitHub.

Related jobs at Datadog

Software Engineer - Cloud Metrics — ENGINEERING

Boston, New York

Engineering Manager - Application Performance Monitoring (APM) — ENGINEERING

Boston

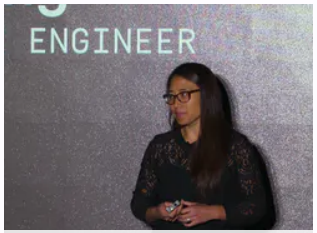
Software Engineer - Application Performance Monitoring — ENGINEERING

Paris

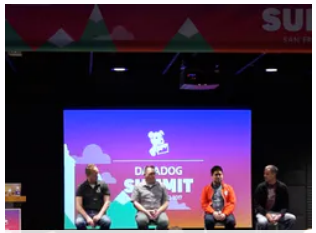
See all jobs at Datadog

Download to learn more☐☐

Related Posts



Watchdog for Infrastructure Metrics



Monitoring the cloud's critical infrastructure

Monitoring 101: Investigating performance issues

Monitoring 101: Alerting on what matters

Start monitoring your metrics in minutes

FIND OUT HOW

FREE TRIAL

Product

Features
Integrations
Dashboards
Log Management
APM
Continuous Profiler
Security Monitoring
Network Monitoring
Synthetic Monitoring
Real User Monitoring
Serverless
Pricing
Documentation
Support
Resources
Security

About

COVID-19 Update
Contact Us
Partners
Press
Leadership
Careers
Legal
Investor Relations
Analyst Reports

Social

Blog
Español
日本語
Instagram
LinkedIn
Twitter
YouTube

Language

日本語