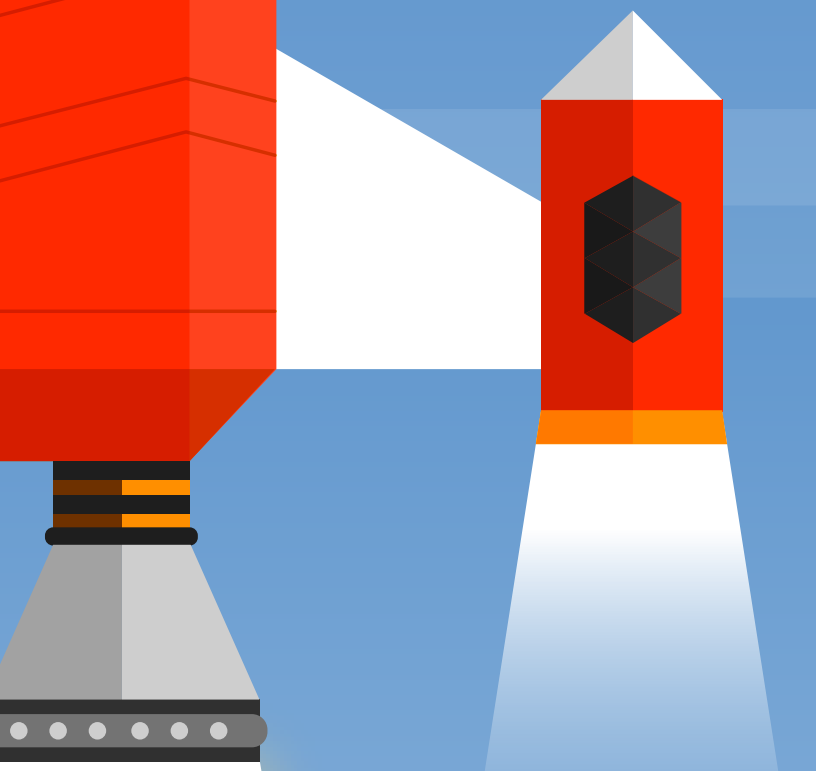
DATADOG | amazon web services

Monitoring in the Cloud



"Measure what is measurable,
and make measurable what is not so."

—Galileo

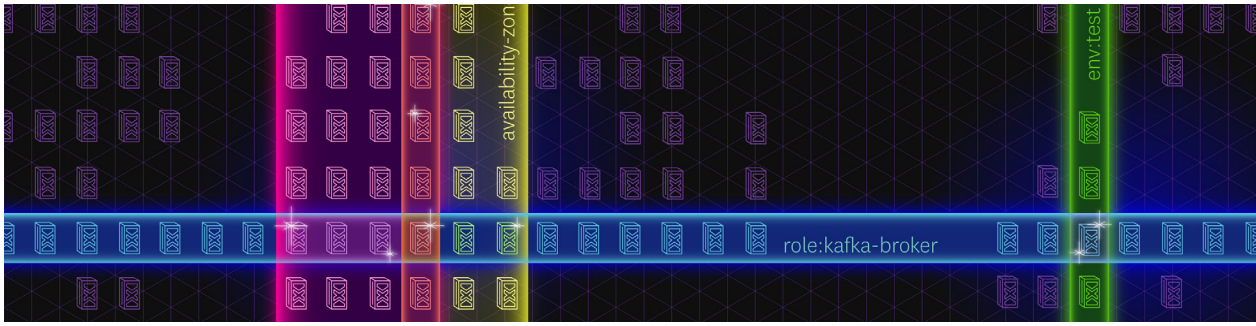
About Datadog

Datadog is a monitoring and analytics platform for cloud-scale application infrastructure. Combining metrics from servers, databases, and applications, Datadog delivers sophisticated, actionable alerts, and provides real-time visibility for your entire infrastructure. Datadog includes 100+ vendor-supported, prebuilt integrations and monitors hundreds of thousands of hosts. To learn more about Datadog, visit https://www.datadog.com

About AWS

For 10 years, Amazon Web Services has been the world's most comprehensive and broadly adopted cloud platform. AWS offers over 70 fully featured services for compute, storage, databases, analytics, mobile, Internet of Things (IoT) and enterprise applications from 33 Availability Zones (AZs) across 12 geographic regions in the U.S., Australia, Brazil, China, Germany, Ireland, Japan, Korea, and Singapore. AWS services are trusted by more than a million active customers around the world — including the fastest growing startups, largest enterprises, and leading government agencies — to power their infrastructure, make them more agile, and lower costs. To learn more about AWS, visit http://aws.amazon.com

Chapter 1: Understanding Modern Infrastructure



In the 10 years since Amazon Web Services (AWS) began offering cloud computing services to the world, the nature of IT infrastructure has changed dramatically.

The cloud has transformed the economics of infrastructure, essentially crumbling the barrier to entry for building applications on world-class technology. It has brought about a fundamental change on the operations side as well: the effortless scaling made possible by the cloud means that the typical organization's infrastructure is always in flux. The nature of the cloud has elevated the need for new methods and new tools for monitoring an infrastructure of constantly changing, often short-lived components.

In this eBook, we will outline an effective framework for monitoring modern infrastructure and applications, however large or dynamic they may be.

PETS VS CATTLE

A useful analogy in thinking about dynamic infrastructure is "pets versus cattle." Pets are unique, they have names, and you care greatly about the health and well-being of each.
Cattle, on the other hand, are numbered rather than named. They are part of a herd, and the overall health of the herd is your primary concern.

In most cases your servers, containers, and other cloud resources should be thought of as cattle. Therefore you should focus on aggregate health and performance of services rather than isolated datapoints from your hosts. Rarely should you page an engineer in the middle of the night for a host-level issue such as elevated CPU. If on the other hand latency for your web application starts to surge, you'll want to take action immediately.

MODERN APPROACHES TO MONITORING

Monitoring allows engineering teams to identify and resolve performance issues before they cause problems for end users. Comprehensive monitoring is a must now that development moves faster than ever—many teams deploy new code dozens of times per day.

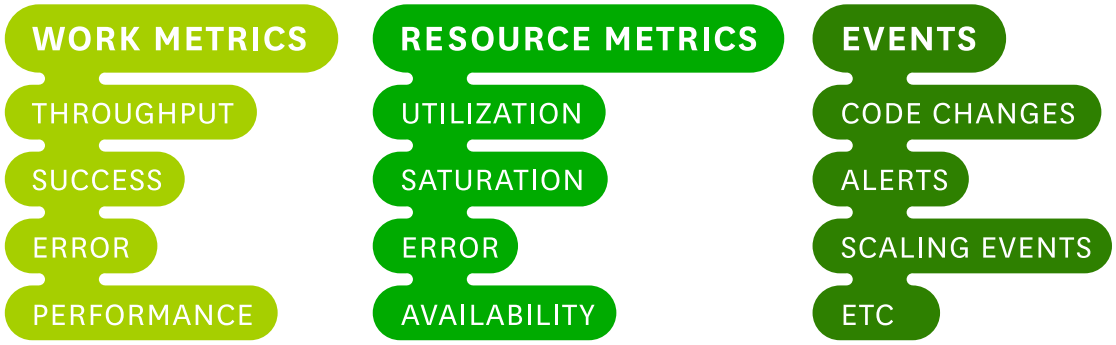
In the following chapters we will outline a practical monitoring framework for dynamic infrastructure. This framework comes out of our experience monitoring large-scale infrastructure for thousands of customers, as well as for our own rapidly scaling application on the AWS cloud. It also draws on the work of Brendan Gregg of Netflix, Rob Ewaschuk of Google, and Baron Schwartz of VividCortex.

Chapter 2: Collecting the Right Data

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."**

Most monitoring data falls into one of two categories: metrics and events.



METRICS

Metrics capture a value pertaining to your systems at a specific point in time. Therefore, metrics are usually collected at regular intervals to monitor a system's evolution over time.

There are two important categories of metrics: work metrics and resource metrics. For each system in your 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. These metrics are invaluable for surfacing real, often user-facing issues, as we'll discuss in the following chapter. It's helpful to break work metrics down into four subtypes, which we'll illustrate here with example metrics for a database instance on Amazon RDS:

WORK METRICS

METRIC TYPE	DESCRIPTION	EXAMPLE (DATABASE)
THROUGHPUT	THE AMOUNT OF WORK COMPLETED PER UNIT TIME	QUERIES PER SECOND
SUCCESS	THE PORTION OF WORK EXECUTED SUCCESSFULLY	QUERIES-QUERY ERRORS
ERROR	THE NUMBER, RATE, OR PERCENTAGE OF ERRONEOUS RESULTS	QUERY ERRORS
PERFORMANCE	MEASUREMENT OF HOW EFFICIENTLY A COMPONENT IS DOING ITS WORK	READ QUERY LATENCY

RESOURCE METRICS

Most components of your infrastructure serve as a resource to other systems. 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 a resource if another system requires that component to produce work.

Resource metrics are especially valuable for investigating problems (see chapter 4). For each resource in your system, try to collect metrics covering four key areas:

RESOURCE METRICS

METRIC TYPE	DESCRIPTION	EXAMPLE (DATABASE)
UTILIZATION	THE PERCENTAGE OF TIME THAT THE RESOURCE IS BUSY, OR HOW MUCH OF THE RESOURCE'S CAPACITY IS IN USE	OPEN DATABASE CONNECTIONS
SATURATION	THE AMOUNT OF REQUESTED WORK THAT THE RESOURCE CANNOT YET SERVICE	DISK QUEUE DEPTH
ERROR	INTERNAL ERRORS THAT MAY NOT BE OBSERVABLE IN THE WORK THE RESOURCE PRODUCES	FAILED CONNECTION ATTEMPTS
AVAILABILITY	THE PERCENTAGE OF TIME THAT THE RESOURCE RESPONDED TO REQUESTS	N/A

EVENTS

In contrast to metrics, which are collected more or less continuously, events are discrete, infrequent occurrences. Events capture what *happened*, at a point in *time*, with optional *additional information*. Some examples:

- **Changes:** Code releases, builds, and build failures
- **Alerts:** Notifications generated by your primary monitoring system or by third-party tools
- **Scaling events:** Adding or subtracting hosts or containers

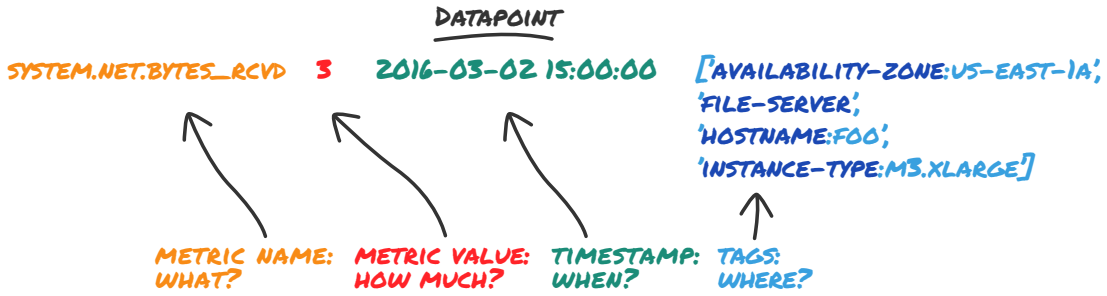
Events provide crucial context for understanding changes in your system's behavior.

TAGGING

As discussed in chapter 1, modern infrastructure is constantly in flux. Auto-scaling servers die as quickly as they're spawned, and containers come and go with even greater frequency. Adding tags to your metrics lets you adopt a "cattle, not pets" approach to see past these changes. Instead of monitoring your servers as unique entities, you can aggregate metrics to focus on different services, availability zones, instance types, software versions, roles—or any other relevant dimension.

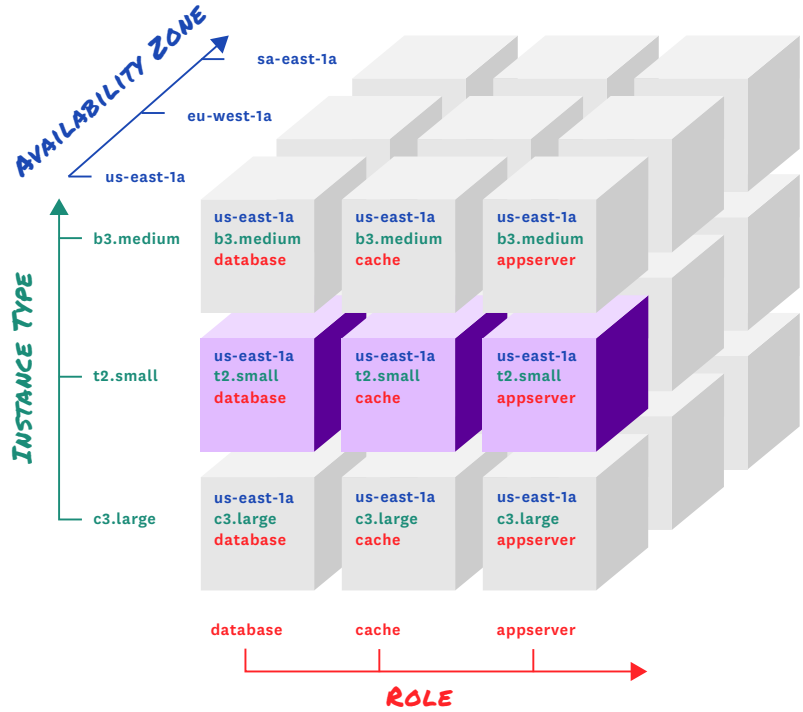
WHAT'S A METRIC TAG?

Tags are metadata that declare all the scopes attached to a datapoint. They allow you to filter or aggregate your metrics on the fly to extract meaningful views.



CREATING NEW DIMENSIONS WITH KEY:VALUE TAGS

When you add a key:value tag, you are adding a new dimension (the key) and a new attribute in that dimension (the value). For example, the last tag on the datapoint above declares an `instance-type` dimension, and gives the metric the attribute `m3.xlarge` in that dimension. You can then slice and dice your infrastructure along any tagged dimension.



Chapter 3: Alerting on What Matters

Automated alerts allow you to spot problems anywhere in your infrastructure, so that you can rapidly identify their causes and minimize service degradation and disruption.

LEVELS OF ALERTING URGENCY

Not all alerts carry the same degree of urgency. Some require immediate human intervention, and some merely point to areas where attention may be needed in the future.

ALERTS AS RECORDS (LOW SEVERITY)

Many alerts will not be associated with a service problem, so a human may never even need to be aware of them. Resource issues such as elevated CPU should usually generate a low-urgency alert that is recorded for future reference but does not interrupt anyone's work. Those occurrences are often transient and do not manifest as real problems, but a record of that anomaly could prove invaluable if significant issues do develop.

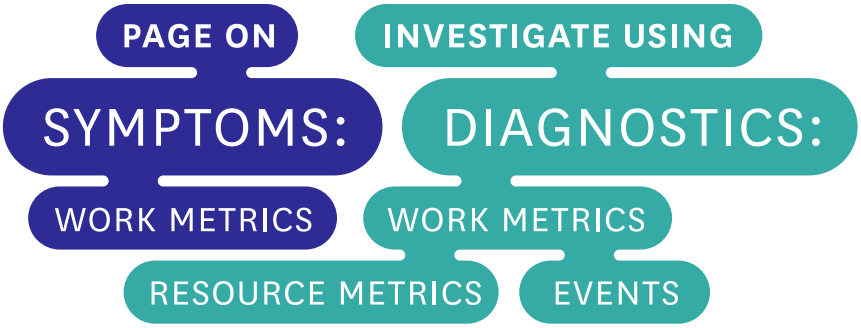
ALERTS AS NOTIFICATIONS (MODERATE SEVERITY)

The next tier of alerting urgency is for issues that do require intervention, but not right away. Perhaps a database's disk space is filling up and should be scaled out in the next several days. These alerts should be highly visible but noninterrupting—sent via email or posted to the relevant team's chat room.

ALERTS AS PAGES (HIGH SEVERITY)

The most urgent alerts should receive special treatment and be escalated to a page (as in "pager") to urgently request human attention. Any instance of web application response times exceeding your internal SLA would warrant immediate attention, whatever the hour.

PAGE ON SYMPTOMS



Pages deserve special mention: they are extremely effective for delivering information, but they can be quite disruptive if misused. In general, a page is appropriate when a system stops doing work with acceptable throughput, latency, or error rates.

The fact that your system stopped doing useful work is a *symptom*. It is a manifestation of an issue that may have any number of different *causes*. For example: if your website has been responding very slowly for the last three minutes, that is a symptom. Possible causes include high database latency, failed application servers, Memcached being down, high load, and so on.

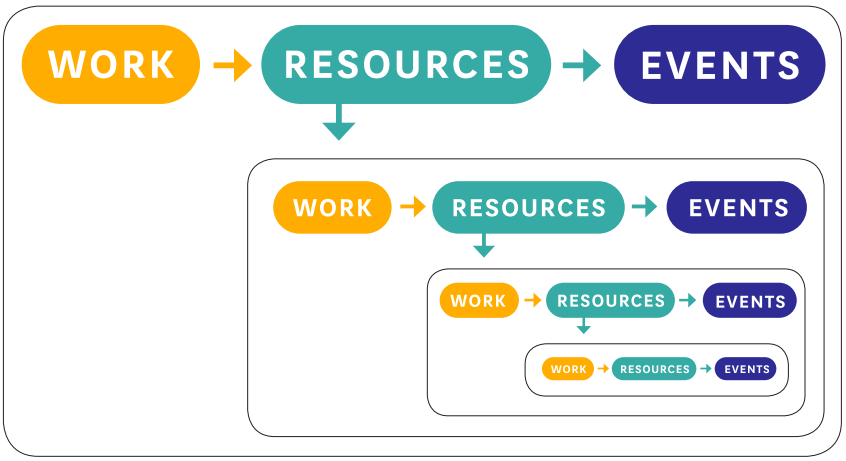
Building your pages around symptoms identified in your work metrics helps surface real, oftentimes user-facing problems, rather than hypothetical or internal problems.

Chapter 4: Investigating Performance Issues

Investigation is often the least structured aspect of monitoring, driven largely by hunches and guess-and-check. This chapter describes a more directed approach for finding and correcting root causes.

IT'S RESOURCES ALL THE WAY DOWN

Each of your systems that produces useful work likely relies on other systems, which in turn rely on lower-level systems to do *their* work. Thinking about which systems *produce* useful work, and which resources *support* that work, can help you approach your investigation systematically.



1 **Start at the top with work metrics**
First examine the work metrics for the highest-level system that is exhibiting problems. These metrics will usually set the direction for your investigation. For example, if the percentage of work that is successfully processed drops, diving into error metrics, and especially the types of errors being returned, will often help narrow your focus.

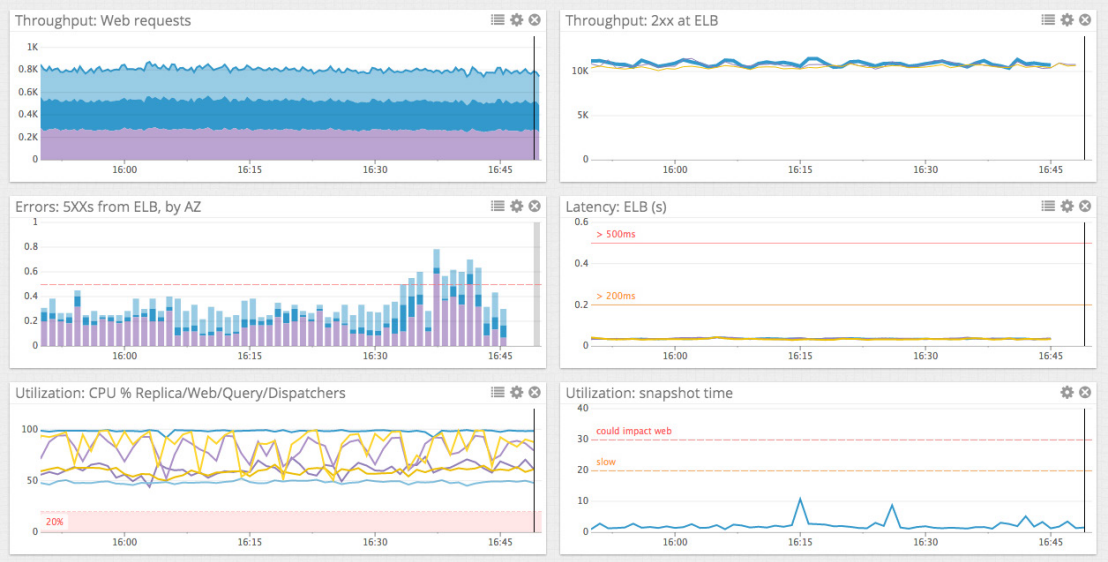
2 **Dig into resources**
Next examine the system's resources—physical resources as well as services that support the system. Well-designed dashboards (see below) enable you to quickly scan relevant resource metrics for each system. Are those resources unavailable? Are they highly utilized or saturated? If so, recurse into those resources and begin investigating each of them at step 1.

3 **Did something change?**
Next consider events that may be correlated with your metrics. Look for code releases, internal alerts, or other events that were recorded just before the problem developed.

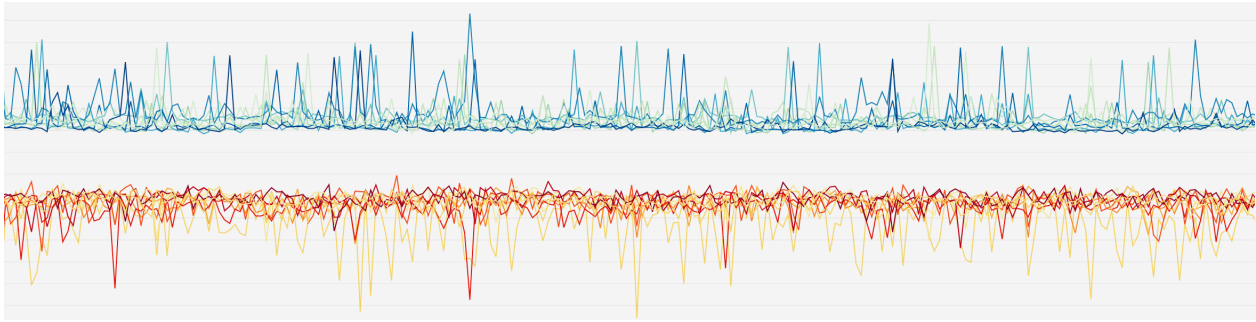
4 **Fix it (and don't forget it)**
Once you have determined what caused the issue, correct it. Your investigation is complete when symptoms disappear—you can now think about how to avoid similar problems in the future.

BUILD DASHBOARDS BEFORE YOU NEED THEM

To keep your investigations focused, set up dashboards in advance. You may want to set up one dashboard for your high-level application metrics, and one dashboard for each subsystem. Each system's dashboard should render the work metrics of that system, along with resource metrics of the system itself and key metrics of the subsystems it depends on. If possible, overlay relevant events on the graphs for correlation analysis.



Chapter 5: Datadog is Cloud-Scale Monitoring



We've now stepped through a high-level framework for data collection and tagging, automated alerting, and incident response. To apply these principles in your own cloud environment, you need a monitoring system that is as dynamic as your infrastructure.

Datadog was built to meet the unique needs of modern, cloud-scale infrastructure:

— **Comprehensive monitoring.** Out of the box, Datadog collects monitoring data from Amazon EC2, ELB, RDS, and other AWS services, plus more than 100 other technologies. Furthermore, the Datadog Agent can collect custom metrics from virtually any application.

— **Flexible aggregation.** Datadog's native support for tagging allows you to aggregate metrics and events on the fly to generate the views that matter most.

— **Effortless scaling.** Datadog scales automatically with your infrastructure, whether you have tens, hundreds, or thousands of instances. Datadog auto-enrolls new hosts and containers as they come online, using AWS- and user-provided tags to include the relevant metrics in existing graphs and alerts.

— **Sophisticated alerting.** Virtually any type of monitoring data can be used to trigger a Datadog alert: fixed or dynamic metric thresholds, outliers, events, status checks, and more.

— **Collaboration baked in.** Easily sharable dashboards, graphs, and annotations help teams stay on the same page. Seamless integrations with collaboration tools such as PagerDuty, Slack, and HipChat make conversations around metrics and system performance as frictionless as possible.

If you are ready to apply the monitoring principles you've learned in this eBook, you can sign up for a full-featured Datadog trial at www.datadog.com.