Gathering Metrics from Your Infrastructure and Applications

Posted December 26, 2017 15.6k MONITORING CONCEPTUAL

By: Justin Ellingwood

Introduction

Understanding the state of your systems is essential for ensuring the reliability and stability of your applications and services. Information about the health and performance of your deployments not only helps your team react to issues, it also gives them the security to make changes with confidence. One of the best ways to gain this insight is with a robust monitoring system that gathers metrics, visualizes data, and alerts operators when things appear to be broken.

In our introduction to metrics, monitoring, and alerting guide, we discussed some of the core concepts involved in monitoring software and infrastructure. Metrics are the primary material processed by monitoring systems to build a cohesive view of the systems being tracked. Knowing which components are worth monitoring and what specific characteristics you should be looking at is the first step in designing a system that can provide reliable, actionable insights about the state of your software and hardware.

In this guide, we will start by discussing a popular framework used to identify the most critical metrics to track. Afterwards, we will walk through how those indicators can be applied to components throughout your deployment. This process will focus on the fundamental resources of individual servers at first and then adjust the scope to cover increasingly larger areas of concern.

The Golden Signals of Monitoring

In the highly influential Google SRE (site reliability engineering) Book, the chapter on monitoring distributed systems introduces a useful framework called the **four golden signals of monitoring** that represents the most important factors to measure in a user-facing system. We will discuss each of these four characteristics below.

Latency

Latency is a measurement of the time it takes to complete an action. The specifics of how this is measured depends on the component, but some common analogues are processing time, response time, or travel time.

Measuring latency gives you a concrete measure of how long a specific task or action takes to complete. Capturing the latency of various components allows you to build a holistic model of the different performance characteristics of your system. This can help you find bottlenecks, understand which resources require the most time to access, and notice when actions suddenly take longer than expected. The authors of the SRE book emphasize the importance of distinguishing between successful and unsuccessful requests when calculating latencies, as they can have very different profiles that might skew the averages of a service.

Traffic

Traffic measures the "busyness" of your components and systems. This captures the load or demand on your services so that you can understand how much work your system is currently performing.

Sustained high or low traffic numbers can indicate that the service might need more resources or that a problem is preventing traffic from being routed correctly. However, for the majority of cases, traffic rates will be most useful in helping understand issues surfaced through other signals. For example, if latency increases beyond an acceptable level, being able to correlate that time frame with a spike in traffic is helpful. Traffic can be used to understand the maximum amount of traffic that can be handled and how the service degrades or fails at various stages of load.

Errors

It is important to track errors to understand the health of your components and how frequently they are failing to respond to requests appropriately. Some applications or services expose errors in clean, ready-made interfaces, but additional work may be required to gather the data from other programs.

Distinguishing between different types of errors can make it easier to pinpoint the exact nature of problems that are impacting your applications. This also gives you flexibility in alerting. You might need to be alerted immediately if one type of error appears, but for another, you might not be concerned as long as the rate is below an acceptable threshold.

Saturation

Saturation measures how much of a given resource is being used. Percentages or fractions are frequently used with resources that have a clear total capacity, but more creative measurements might be needed for resources with less well-defined maximum.

Saturation data provides information about the resources that a service or application depends on to operate effectively. Since a service provided by one component may be consumed by another, saturation is one of the glue metrics that surfaces the capacity problems of underlying systems. As such, saturation and latency problems in one layer might correspond with a marked increase in traffic or error measurements in the underlying layer.

Measuring Important Data Throughout Your Environment

Using the four golden signals as a guideline, you can begin to look at how those metrics would be expressed throughout the hierarchy of your systems. Since services are often built by adding layers of abstraction on top of more basic components, metrics should be designed to add insight at each level of the deployment.

We will look at different levels of complexity present in common distributed application environments:

- Individual server components
- Applications and services
- Collections of servers
- Environmental dependencies
- End-to-end experience

The ordering above expands the scope and level of abstraction with each subsequent layer.

Metrics to Collect for Individual Server Components

The base level metrics that are important to collect are those relevant to the underlying computers that your systems rely on. Although considerable effort in modern software development goes into abstracting the physical components and low level operating system details, every service relies on the underlying hardware and operating systems to do its work. Because of this, keeping an eye on the foundational resources of your machines is the first step in building an understanding of the health of your systems.

When considering which metrics to collect at the machine level, think about the individual resources available. These will include representations of your server's hardware as well as core abstractions provided by the OS, like processes and file descriptors. Looking at each component in terms of the four golden signals, certain signals may be obvious while others may be more difficult to reason about.

Brendan Gregg, an influential performance engineer, outlines many ways to get core metrics from Linux systems to satisfy the needs of a framework he calls the USE method for performance analysis (**u**tilization, **s**aturation, and **e**rrors). Since there is significant overlap between the USE method and the four golden signals, we can use some of his recommendations as a jumping off point for figuring out what data to collect from server components.

To measure **CPU**, the following measurements might be appropriate:

- **Latency**: Average or maximum delay in CPU scheduler
- **Traffic**: CPU utilization
- **Errors**: Processor specific error events, faulted CPUs
- **Saturation**: Run queue length

For **memory**, the signals might look like this:

- **Latency**: (none - difficult to find a good method of measuring and not actionable)
- **Traffic**: Amount of memory being used
- **Errors**: Out of memory errors
- **Saturation**: OOM killer events, swap usage

For **storage devices**:

- **Latency**: average wait time (`await`) for reads and writes
- **Traffic**: read and write I/O levels
- **Errors**: filesystem errors, disk errors in `/sys/devices`
- **Saturation**: I/O queue depth

The **networking** signals can look like this:

- **Latency**: Network driver queue
- **Traffic**: Incoming and outgoing bytes or packets per second
- **Errors**: Network device errors, dropped packets
- **Saturation**: overruns, dropped packets, retransmitted segments

Along with representations of physical resources, it is also a good idea to gather metrics related to operating system abstractions that have limits enforced. Some examples that fall into this category are file handles and thread counts. These are not physical resources, but instead constructs with ceilings set by the operating system to prevent processes from overextending themselves. Most can be adjusted and configured with commands like `ulimit` , but tracking changes in usage of these resources can help you detect potentially harmful changes in your software's usage.

Metrics to Collect for Applications and Services

Moving up a layer, we start to deal with the applications and services that run on the servers. These programs use the individual server components we dealt with earlier as resources to do work. Metrics at this level help us understand the health of our single-host applications and services. We've separated distributed, multi-host services into a separate section to clarify the factors most important in those configurations.

While the metrics in the last section detailed the capabilities and performance of individual components and the operating system, the metrics here will tell us how well applications are able to perform the work we ask of them. We also want to know what resources our applications depend on and how well they manage those constraints.

It is important to keep in mind that the metrics in this section represent a departure from the generalized approach we were able to use last time. The metrics that are most important from this point on will be very dependent on your applications' characteristics, your configuration, and the workloads that you are running on your machines. We can discuss ways of identifying your most important metrics, but your results will depend on what the server is specifically being asked to do.

For applications that serve clients, the four golden signals are often fairly straightforward to pick out:

- **Latency**: The time to complete requests
- **Traffic**: Number of requests per second served
- **Errors**: Application errors that occur when processing client requests or accessing resources
- **Saturation**: The percentage or amount of resources currently being used

Some of the more important metrics you'll want to keep track of are those related to dependencies. These will often be best expressed by saturation metrics related to individual components. For instance, application memory utilization, available connections, number of file handles opened, or number of workers active can help you understand the effect of your configuration applied in the context of the physical server.

The four golden signals were designed primarily for distributed microservices, so they assume a client-server architecture. For applications that do not use a client-server architecture, the same signals are still important, but the "traffic" signal might need to be reconsidered slightly. This is basically a measurement of busyness, so finding a metric that adequately represents that for your application will serve the same purpose. The specifics will depend on what your program is doing, but some general substitutes might be the number of operations or data processed per second.

Metrics to Measure Collections of Servers and Their Communication

Most services, especially when operated in a production environment, will span multiple server instances to increase performance and availability. This increased level of complexity adds additional surface area that is important to monitor. Distributed computing and redundant systems can make your systems more flexible, but network-based coordination is more fragile than communication within a single host. Robust monitoring can help alleviate some of the difficulties of dealing with a less reliable communication channel.

Beyond the network itself, for distributed services, the health and performance of the server group is more important than the same measures applied to any individual host. While services are intimately tied to the computer they run on when confined to a single host, redundant multi-host services rely on the resources of multiple hosts while remaining decoupled from direct dependency on any one computer.

The golden signals at this level look very similar to those measuring service health in the last section. They will, however, take into account the additional coordination required between group members:

- **Latency**: Time for the pool to respond to requests, time to coordinate or synchronize with peers
- **Traffic**: Number of requests processed by the pool per second

- **Errors**: Application errors that occur when processing client requests, accessing resources, or reaching peers
- **Saturation**: The amount of resources currently being used, the number of servers currently operating at capacity, the number of servers available.

While these have a definite resemblance to the important metrics for single-host services, each of the signals grows in complexity when distributed. Latency becomes a more complicated issue as processing can require communication between multiple hosts. Traffic is no longer a function of a single server's abilities, but is instead a summary of the groups capabilities and the efficiency of the routing algorithm used to distribute work. Additional error modes are introduced related to networking connectivity or host failure. Finally, saturation expands to include the combined resources available to the hosts, the networking link connecting each host, and the ability to properly coordinate access to the dependencies each computer needs.

Metrics Related to External Dependencies and the Deployment Environment

Some of the most valuable metrics to collect exist at the boundary of your application or service, outside of your direct control. External dependencies including those related to your hosting provider and any services your applications are built to rely on. These represent resources you are not able to administer directly, but which can compromise your ability to guarantee your own service.

Because external dependencies represent critical resources, one of the only mitigation strategies available in case of full outages is to switch operations to a different provider. This is only a viable strategy for commodity services, and even then only with prior planning and loose coupling with the provider. Even when mitigation is difficult, knowledge of external events affecting your application is incredibly valuable.

The golden signals applied to external dependencies may look similar to this:

- **Latency**: Time it takes to receive a response from the service or to provision new resources from a provider
- **Traffic**: Amount of work being pushed to an external service, the number of requests being made to an external API
- **Errors**: Error rates for service requests
- **Saturation**: Amount of account-restricted resources used (instances, API requests, acceptable cost, etc.)

These metrics can help you identify problems with your dependencies, alert you to impending resource exhaustion, and help keep expenses under control. If the service has drop-in alternatives, this data can be used to decide whether to move work to a different provider when metrics indicate a problem is occurring. For situations with less flexibility, the metrics can at least be used to alert an operator to respond to the situation and implement any available manual mitigation options.

Metrics that Track Overall Functionality and End-to-End Experience

The highest level metrics track requests through the system in context of the outermost component that users interact with. This might be a load balancer or other routing mechanism that is responsible for receiving and coordinating requests to your service. Since this represents the first touch point with your system, collecting metrics at this level gives an approximation of the overall user experience.

While the previously described metrics are incredibly useful, the metrics in this section are often the most important to set up alerting for. To avoid response fatigue, alerts—especially pages—should be reserved for scenarios that have a recognizable negative effect on user experience. Problems surfaced with these metrics can be investigated by drilling down using the metrics collected at other levels.

The signals we look for here are similar to those of the individual services we described earlier. The primary difference is the scope and the importance of the data we gather here:

- **Latency**: The time to complete user requests
- **Traffic**: Number of user requests per second
- **Errors**: Errors that occur when processing client requests or accessing resources
- **Saturation**: The percentage or amount of resources currently being used

As these metrics parallel user requests, values that fall outside of acceptable ranges for these metrics likely indicate direct user impact. Latency that does not conform to customer-facing or internal SLAs (service level agreements), traffic that indicates a severe spike or drop off, increases in errors rates, and an inability to serve requests due to resource constraints are all fairly straightforward to reason about at this level. Assuming that the metrics are accurate, the values here can be directly mapped against your availability, performance, and reliability goals.

Conclusion

In this guide, we began by discussing the four golden signals that tend to be most helpful for discovering and understanding impactful changes in your systems. Afterwards, we used the signals as a lens to evaluate the most important factors to track at different layers of a deployment.

Evaluating your systems from from top to bottom can help identify the critical components and interactions required to run reliable and performant services. The four golden signals can be a great starting point for structuring metrics to best indicate the health of your systems. However, keep in mind that while the golden signals are a good framework, you will have to be aware of other metrics specific to your situation. Collect whatever data you think will be most likely to warn of problems or help you troubleshoot when things go wrong.

By: Justin Ellingwood

Tutorial Series

An Introduction to Infrastructure and Application Monitoring

In this series, we explore what metrics and monitoring are and how to best use them to gain visibility into your systems and the responsiveness of your team. Collecting metrics from your infrastructure gives you insight into the health and performance of your systems. Metrics can be used to create dashboards to troubleshoot issues or give a summary of the state of your applications and resources. Alerts can be defined to notify you as soon as situations require your attention.

Show Tutorials

Related Tutorials

Putting Monitoring and Alerting into Practice

An Introduction to Metrics, Monitoring, and Alerting

An Introduction to Tracking Statistics with Graphite, StatsD, and CollectD

How To Use Traefik as a Reverse Proxy for Docker Containers on Debian 9

How To Install Elasticsearch, Logstash, and Kibana (Elastic Stack) on CentOS 7

0 Comments

Leave a comment...

Log In to Comment